Bases de Dades

Cartesianes

carles franquesa i niubò

Pròleg

Quan el Dr. Franquesa em va demanar que escrivís el pròleg del seu llibre "Bases de Dades Cartesianes" em vaig sentir honorat però també a l'hora preocupat per saber si podria estar a l'alçada del que se'm requeria. Suposo que va pensar en mi tot recordant els anys que vàrem compartir a l'assignatura d'Introducció a les Bases de Dades. Varen ser els darrers anys abans de la meva jubilació i en guardo un gran record que crec que els que alumnes de llavors compartiran amb nosaltres.

Fins a data d'avui, no he tingut coneixement d'altres llibres que versin sobre aquests continguts en la nostra llengua catalana, i tan sols per això, ja ens hauríem de felicitar els que estimem les bases de dades i al mateix temps aquesta llengua. Però no és només això.

A mesura que he anat cosint un camí de més de quaranta anys de docència en estudis superiors, he anat adquirint la convicció que allò més motivador i estimulant ha estat la vocació. Pels professors vocacionals no hi ha gratificació més gran que observar la magnitud dels projectes on ara hi treballen alumnes que un dia van arribar amb la incertesa als ulls, i al mateix temps amb una ingenuïtat voraç disposats alimentar-se d'allò se'ls indica, allò que cal preveure és el que transcendirà en les seves activitats professionals en un futur més proper del que sembla. Per això cal entusiasme, il·lusió, loquacitat i coherència en els continguts. Aquest llibre és un exponent de tots aquests factors. Més que un llibre de teoria, sembla un discurs. Una conversa.

Les Bases de Dades han evolucionat força des del seu començament allà pels anys 60 del segle passat. Llavors, el mateix hardware limitava el que es podia fer i les primeres implementacions van ser per resoldre problemes tals com el escandall de peces en la fabricació amb una estructura, fictícia per poc realista, de tipus jeràrquic: una peça, o bé un conjunt de peces, mai depenia de més d'una peça en un nivell superior. A poc de començar a utilitzar aquestes bases de dades jeràrquiques, el fonament teòric d'altres solucions s'estava cuinant per part de veritables pioners com Edgar F. Codd i, més tard Christopher J. Date. El conjunt de solucions trobades van suposar l'aparició teòrica de les bases de dades relacionals. Les limitacions del hardware van impedir el seu desenvolupament posposant-ne la implementació gairebé quinze anys.

Avui en dia moltes de les bases de dades ja són del tipus relacional tot i que ja s'apunten noves solucions agosarades que encara han de demostrar la seva utilitat real i que tenen uns fonaments matemàtics no tan robustos i per descomptat molt menys rigorosos que els del model relacional. En molts casos es tracta de permetre nous tipus de dades elementals com bases de dades d'objectes, o de bases de dades de molta grandària, normalment de tipus multimèdia, suportant-se més en la quantitat de recursos materials dels que es disposa que en la integritat de les dades.

En aquest llibre, el Dr. Franquesa fa un estudi molt complert de les bases de dades relacionals i, en particular proposa la solució real amb la implementació d'una d'elles.

El llibre es proposa assentar primerament una base teòrica inicial de tipus matemàtic seguida del que en diu "Anàlisi del Projecte" on es recull l'ambient en el que la solució informàtica es trobarà finalment circumscrita.

Segueix un capítol de Disseny previ ja a l'estudi nucleic del model Entitat - Relació per entrar al capítol següent a l'Àlgebra Relacional, on s'estudia quins mecanismes es poden utilitzar, i com s'utilitzen, per obtenir resultats de la base de dades preparada, dissenyada, com a solució al problema abordat pel projecte. En aquest capítol, el cinquè, l'autor proposa uns gràfics magnífics i molt aclaridors per a representar múltiples tuples.

Entra després a la descripció, central en varis aspectes, del Llenguatge Estructurat de Consultes que ens ha de servir per a primer descriure la pròpia base de dades i, després per arribar a utilitzar les dades que contindrà. Naturalment el fet de tenir les dades servirà de poc si no podem consultar-les, esborrar-les o afegir-ne quan ens convingui. Per fi, caldrà que tot el que s'ha vist i estudiat es posi en context en una implementació en particular, escollint per a fer-ho el SGBD PostgreSQL que és, no solament una de les millors implementacions, sinó també una de les més fàcils de trobar.

Em cal dir que, quan impartíem l'assignatura d'Introducció a les Bases de Dades, m'hagués agradat tenir aquest llibre com a base per les explicacions als alumnes. Al mateix temps, els serviria a ells com a referència actual i futura sobre les bases de dades.

He dit que m'agradaria haver tingut aquest llibre i no he estat exacte, el que m'hagués agradat hagués estat escriure'l!.

Josep Maria Bañeres,

Professor responsable de Bases de Dades,
ja retirat, de la Universitat de Barcelona.

Cabrils, octubre de 2014.

Índex

1 Teoria de Conjunts **11**
1.1 Definicions i Nomenclatura 11
1.1.1 Abstracció 13
1.1.2 Descripció de Conjunts 14
Per enumeració dels seus elements 14
A partir de propietats dels seus elements 15
1.2 Operacions entre Conjunts 16
1.2.1 Unió 17
1.2.2 Intersecció 18
1.2.3 Diferència 20
1.2.4 Producte Cartesià 21
1.3 Lògica de Predicats 23
1.3.1 Càlcul de Predicats 23
Operació de negació lògica 24
Operació lògica disjuntiva 24
Operació lògica conjuntiva 25
Lleis d'Augustus de Morgan 26
1.3.2 Relació amb la Teoria de Conjunts 27

2 Anàlisi del Projecte **29**
2.1 Sistemes Gestors de Bases de Dades 29
2.1.1 Arquitectura Client-Servidor 30
2.2 Principi d'Independència de les Dades 31
2.2.1 Arquitectura a Tres Nivells 32
2.3 Immersió en el Domini 34
2.4 Definició de Requeriments 35
2.4.1 Informació Estadística 35
2.5 Trajectòria de l'Error 36
2.6 Exemple per un Club Esportiu 37
2.6.1 Requeriments per al Club Esportiu 37

3 Disseny de Bases de Dades **41**
3.1 Models de Dades 44
3.2 Domini del Problema 45
3.3 Màximes de Qualitat d'un Disseny 45
3.3.1 Prohibir la Redundància 46
3.3.2 Prohibir Nuls Estructurals 47
3.3.3 Unificar Conceptes 47

4 Model Entitat Relació **49**
4.1 Entitats . . . 50
4.2 Atributs . . . 51
4.3 Atributs Clau . . . 54
4.4 Atributs Multivalorats . . . 55
4.5 Atributs Compostos . . . 57
4.6 Atributs Calculats . . . 58
4.7 Relacions . . . 60
4.7.1 Dependències d'existència . . . 61
4.7.2 Cardinalitat d'una relació binària . . . 62
4.8 Relacions 1:1 . . . 63
4.9 Relacions 1:N . . . 64
4.9.1 Entitats de suport a la interfície . . . 67
4.9.2 Entitats Febles . . . 69
4.9.3 Classificació de les Relacions 1:N . . . 71
4.10 Relacions M:N . . . 72
4.10.1 Atributs en relacions M:N . . . 76
4.11 Autorelacions . . . 77
4.11.1 Autorelacions 1:1 . . . 77
4.11.2 Autorelacions 1:N . . . 79
4.11.3 Autorelacions M:N . . . 80
4.12 Càlcul Automàtic de les Cardinalitats . . . 82
4.13 Relacions Ternàries . . . 83
4.13.1 Cardinalitats en les relacions ternàries . . . 85
4.13.2 Relació ternària amb tres relacions binàries . . . 86
4.14 Model Entitat Relació Estès . . . 87
4.14.1 Especialització i Generalització d'Entitats . . . 88
Especialitzacions o generalitzacions completes . . . 90
Especialitzacions o generalitzacions disjuntes . . . 90
4.14.2 Agregació d'Entitats . . . 91

5 Àlgebra Relacional **93**
5.1 Àlgebres . . . 94
5.2 Relacions . . . 94
5.2.1 Relació . . . 95
Tuples . . . 96
Càlcul relacional per tuples . . . 98
Dominis . . . 98
Definició formal de relació . . . 99
Compatibilitat entre relacions . . . 101
5.2.2 Visió Cartesiana d'una Relació . . . 102
5.3 Model Relacional d'una Base de Dades . . . 105
5.3.1 Atributs estructurals i atributs descriptius . . . 106
5.3.2 Claus Primàries . . . 106
5.3.3 Claus Foranes . . . 108
5.3.4 Transformació del Model ER al Model Relacional . . . 110
5.3.5 Diagrama d'Esquemes d'una Base de Dades . . . 117
5.4 Operacions Bàsiques amb Relacions . . . 120
5.4.1 Operació de Selecció . . . 121
5.4.2 Operació de Projecció . . . 123

Composició d'operacions . . . 125
5.4.3 Producte Cartesià de Relacions . . . 126
5.4.4 Unió de Relacions . . . 129
5.4.5 Diferència de Relacions . . . 130
5.4.6 Renomenament de Relacions . . . 131
5.5 Operacions Addicionals . . . 135
5.5.1 Intersecció de Relacions . . . 135
5.5.2 Reunió Natural . . . 137
Reunió interna . . . 138
Reunions externes . . . 140
5.5.3 Divisió de Relacions . . . 142
5.5.4 Assignació de Relacions . . . 143
5.6 Àlgebra Relacional Estesa . . . 145
5.6.1 Projecció Generalitzada . . . 145
5.6.2 Valors Nuls . . . 146
Lògica de tres valors . . . 147
5.6.3 Funcions d'Agregació . . . 147

6 Llenguatge Estructurat de Consultes 153
6.1 Entorn de Treball . . . 154
6.1.1 Familiarització amb l'SGBD . . . 156
Familiarització amb el `psql` . . . 156
Familiarització amb l'SQL . . . 157
6.2 Llenguatge de Definició de Dades . . . 158
6.2.1 Multiconjunts . . . 158
6.2.2 Taules . . . 158
6.2.3 Metadades . . . 159
6.2.4 Dominis dels Atributs . . . 160
Tipus primitius . . . 160
Tipus alfanumèrics . . . 161
Tipus temporals . . . 162
Tipus autoincrementals . . . 163
6.3 Construcció de la Base de Dades . . . 164
6.3.1 Espai de Treball . . . 165
6.3.2 Escript Principal . . . 165
6.3.3 Creació dels Dominis . . . 167
6.3.4 Creació de les Taules . . . 169
Integritat referencial . . . 172
Claus candidates . . . 177
6.4 Llenguatge de Manipulació de Dades . . . 186
Escripts d'inserció . . . 186
6.4.1 Consultes de Lectura . . . 187
Estructura fonamental . . . 187
Dades . . . 188
Clàusula `SELECT` . . . 190
Clàusula `FROM` . . . 192
Clàusula `WHERE` . . . 198
6.4.2 Funcions d'Agregació . . . 201
De tots els tipus d'atributs . . . 201
D'atributs numèrics . . . 203

Clàusula GROUP BY . . . 204
Clàusula HAVING . . . 206
6.4.3 Clàusula d'Ordenació . . . 207
Clàusula ORDER BY . . . 207
Consulta de lectura amb totes les clàusules . . . 209
6.4.4 Vistes . . . 210
6.4.5 Consultes d'Actualització . . . 212
Clàusula INSERT INTO . . . 212
Clàusula UPDATE SET . . . 215
Clàusula DELETE FROM . . . 216
6.4.6 Operacions amb Conjunts . . . 218
Clàusula UNION . . . 218
Clàusula EXCEPT . . . 222
Clàusula INTERSECT . . . 224
6.4.7 Operacions de Reunió de la Clàusula FROM . . . 225
Clàusula NATURAL INNER JOIN . . . 226
Clàusula NATURAL LEFT OUTER JOIN . . . 228
Clàusula NATURAL RIGHT OUTER JOIN . . . 229
Clàusula NATURAL FULL OUTER JOIN . . . 230
Clàusula INNER JOIN USING . . . 231
Clàusula INNER JOIN ON . . . 232
6.4.8 Valors Nuls . . . 233
Predicat IS NULL . . . 233
Predicat IS UNKNOWN . . . 234
6.4.9 Consultes Aniuades . . . 235
En l'argument de la clàusula SELECT . . . 236
En l'argument de la clàusula FROM . . . 237
6.4.10 Consultes Aniuades en la Clàusula WHERE . . . 238
Clàusula IN . . . 240
Clàusules SOME i ALL . . . 241
Clàusula EXISTS . . . 244

7 El SGBD PostgreSQL 249
7.1 Documentació de PostgreSQL . . . 250
7.2 Estructura del Projecte . . . 252
7.3 Usuaris i Seguretat en la Base de Dades . . . 254
7.3.1 Permisos . . . 255
Aportacions al projecte del club esportiu . . . 259
7.3.2 Privilegis . . . 260
7.3.3 Concurrència i Transaccions . . . 265
7.4 Components Lògics d'una Base de Dades . . . 270
7.4.1 Components de Dades . . . 270
Atributs indexats . . . 270
7.5 Components de Control . . . 273
7.5.1 Implementació de l'Aplicació per al Club Esportiu . . . 273
Implementació de funcions genèriques . . . 274
Implementació de funcions específiques . . . 275
7.6 Funcions . . . 275
7.6.1 Capçalera . . . 276
7.6.2 Cos . . . 277

7.6.3 Entrada i sortida de dades amb PL/pgSQL 279
7.6.4 Consultes d'actualització en PL/pgSQL 281
7.6.5 Consultes de lectura en PL/pgSQL 284
Consultes que retornen un sol valor 285
Clàusula PERFORM 289
Consultes que retornen registres 290
7.7 Disparadors 293
7.7.1 Funcions de Tipus TRIGGER 294
Capçalera de les funcions TRIGGER 294
Cos de les funcions TRIGGER 294
Valors de retorn dels disparadors 295
7.7.2 Sintaxis de Declaració d'un Disparador 295
7.8 Aplicacions Client 297
7.8.1 Llenguatge Amfitrió i SQL Incrustat 297
Connexió 298
Consultes de lectura 298
Consultes d'actualització 298
7.9 Administració de la Base de Dades 299
7.9.1 Parada i Iniciació del Sistema 299
MS-DOS 299
Línux 300
El programa pg_ctl 302
7.9.2 Manteniment i Còpies de Seguretat 302
7.9.3 Configuració d'un servidor PostgreSQL 304
Arxiu de paràmetres de configuració 305
Arxiu d'autenticació basada en el host 309
Arxiu de mapeig d'usuaris 311

A Estil **313**
A.1 Model Entitat Relació 313
A.2 Model Relacional 314
A.3 Llenguatge Procedural 314
A.4 Versions dels Sistemes Utilitzats 314

B Codis ASCII **315**

C Dades per l'Exemple del Club Esportiu **317**
C.1 Taula comarca 317
C.2 Taula ciutat 318
C.3 Taula persona 319
C.4 Taula telefons 320
C.5 Taula coneix 320
C.6 Taula soci 321
C.7 Taula treballador 321
C.8 Taula esport 322
C.9 Taula fa 323
C.10 Taula nomines 323

Preàmbul

A l'hora de formular continguts teòrics, els humans mostrem predilecció per suportar-los sobre la lògica matemàtica. La mateixa rel etimològica del terme *lògica* s'utilitza llavors com a sufix per indicar estudi, manera d'entendre. Això és així perquè un contingut teòric exposa un model d'alguna faceta de la realitat, i per poder-se desenvolupar precisa d'una coherència que troba prou ben fonamentada en la teoria de conjunts.

Les bases de dades relacionals, que ens disposem a esmicolar, no difereixen d'aquesta aproximació, i així, estructuren unes normes que finalment troben connexió amb la praxis en la implementació que se'n fa en màquines computadores.

Però bé, no avancem històries. Observem què tenim al davant, d'entrada. Assumint la força brutal que tenen dos conceptes tant importants com l'espai i el temps, quedi clar que el que aquí tractarem és l'espai. L'espai són els objectes, el temps són els accions. Orientar el discurs cap a les accions o els procediments convertiria aquest text en un llibre d'algorísmia. I aquí no es tracta d'això. Aquí abordem els objectes, o sigui els espais. De fet és la part senzilla, ja que sembla clar que els substantius són més fàcils de comprendre que els verbs. En la llengua catalana en tenim un bon grapat de proves. El diccionari, [2],[6], permet dir píxel, però no pixelar. Una cosa fa olor, no olora. I encara que sigui permès besar o petonejar, en català acostumem a fer-nos petons. En definitiva, és ben clar que som proclius a suportar la semàntica en els substantius. La prova més clara de que l'univers dels substantius és més senzill que el dels verbs és que als sustantius tan sols els classifiquem en femenins o masculins, singulars i plurals, o sigui gènere i nombre. En canvi pels verbs tenim les conjungacions verbals amb els seus passats, presents i futurs, a part de les tres formes impersonals. Així doncs, tenim sort, les dades viuen al món dels sustantius.

De fet, per estructurar la realitat, cal començar per les dades. I més filosòficament encara, el temps sorgeix a partir de la repetició en les refèrencies als objectes. El temps estableix diferències entre una cosa i ella mateixa. Els instants de creació poden servir per identificar gairebé qualsevol objecte. O sigui, que cal anar en compte a l'hora d'incorporar el temps en les dades que tenen referències temporals.

Per comprendre amb profunditat on cal situar la feina que es proposa, veiem primer una partició de les teories tal com les entenem. Convinguem en que una teoria pot entendre's com un conjunt de veritats inqüestionables més un conjunt de regles que treballen aquestes veritats per construir-ne de noves. Mirat des d'aquest aspecte qualsevol cosa és una teoria. Una bicicleta, i les aplicacions que se li puguin donar, per exemple, constitueixen una teoria. Tot això entra en escena per fer saber que respecte les teories hi ha dues classificacions en tres grups. Per un costat, es pot classificar una teoria com incompleta, categòrica, o contradictòria. I per altra banda com a teoria oberta, sòlida, o inconsistent. I segons es miri des del punt de vista deductiu o bé empíric, s'utilitza una o l'altra classificació. Però bé, mirat des de la distància, sense cap rigor, tot plegat ens resulta útil per disposar d'una terminologia que ubica els continguts que s'exposen en aquest text.

Una teoria oberta, o incompleta, és aquella que sense entrar en cap contradicció postula un conjunt de veritats que després es poden enriquir amb aplicacions concretes donant pas a moltes altres teories. Qualsevol àlgebra és una teoria oberta. I de manera molt més prosaica també podem considerar qualsevol cosa. Una bicicleta, un telèfon o una sabata. Tot això són teories obertes, ja que són veritats que no es contradiuen, cosa que les fa teories, i al mateix temps deixen marge per la interpretació, cosa que les fa obertes. Podríem parlar de bicicletes vermelles o verdes, establint dues teories diferents.

Teoria categòrica és aquella que pretén classificar el conjunt de totes les coses dites. És molt pretensiosa, ja que per qualsevol predicat formulable, o sigui, per qualsevol pregunta que es pugui respondre sí o no, l'anàlisi matemàtica ens pretén respondre. L'anàlisi matemàtica és l'única teoria categòrica que coneixem. Per això és una eina tan estimada en totes les disciplines.

Finalment, en les teories inconsistents es produeixen espais de confusió en els que un mateix predicat resulta cert i fals a l'hora. Són teories que moren aviat. No tenen massa utilitat.

De tot plegat és facil desprendre'n una analogia amb la vida d'una persona al llarg de les tres edats.

Doncs bé, el desenvolupament d'una base de dades per satisfer un conjunt de requeriments és una teoria oberta. O sigui incompleta. Això vol dir que qualsevol objectiu té més d'una aproximació correcta. Ningú ens podrà assegurar que la solució implementada sigui òptima.

De fet, a partir del seu disseny, ningú pot demostrar que una base de dades donarà resposta a totes les sol·licituds que se li puguin formular. Es tracta d'un territori relliscós. Per això, afluixem els nostres objectius. Un pretensió més assequible és la de detectar que alguna part de l'estructura de la base grinyola. Si hi ha errors en el disseny, és possible que els detectem. I quan siguem incapaços de detectar-ne, llavors podrem suposar que el disseny és correcte.

Des que van començar existir les bases de dades fins ara hi ha hagut un ús massiu d'aquestes tecnologies. La mateixa expressió, *base de dades*, és una de les primeres expressions que es va popularitzar de l'àmbit de la computació. Això ha provocat que gran quantitat de desenvolupadors d'aplicacions procuressin incorporar les bases de dades en les seves feines. L'ús de les bases de dades ha estat tant massiu que el coneixement de perquè serveixen s'ha diluït, de manera que a dia d'avui, una gran quantitat d'aplicacions que es venten de connectar-se a bases de dades, no n'hi treuen cap partit que no poguessin treure d'un sistema pla de fitxers.

Més greu encara que el malbaratament de recursos d'espai i temps que això suposa a les aplicacions, s'ha perdut la major part dels avantatges que un sistema gestor de bases de dades ofereix. És un problema greu, ja que els desenvolupadors de codi esmercen hores de feina en accions que el gestor resoldria molt millor del que poden fer ells. La responsabilitat d'aquest fet és sens dubte de tots plegats, de la societat. Queda malament parlar d'una aplicació que no utilitza una base de dades, i per tant, qualsevol aplicació s'assegura un prestigi usant-les.

Cal tenir això en compte, ja que tot plegat ens ve a explicar que ens trobem en un àmbit on hi ha un gran intrusisme, i per tant, cal desconfiar molt de les tendències.

Els arxius per realitzar els exercicis dels últims capítols contenen tant la implementació de la base de dades d'exemple, com les dades que hi ha a l'Apèndix C. Es poden descarregar de

https://drive.google.com/folderview?id=0B1-YyXS4zdgrQWhGOGg2WUExd28&usp=sharing.

Capítol 1

Teoria de Conjunts

La teoria de conjunts és profusament acceptada com la teoria més important de totes les teories. Dit això amb certa precaució per no fer enfadar els puristes, és segurament la primera, la teoria més abstracta.

Però bé, deixant-nos de superlatius, que no fan més que competir, i apropant-nos al tema de manera més pragmàtica, la teoria de conjunts és en l'essència del coneixement analític, i per tant científic, i per tant humà.

En aquest capítol es fa un repàs dels conceptes bàsics d'aquesta teoria, ja que són qüestions preliminars per als capítols posteriors. Es presenta la definició de les operacions que tracten amb conjunts. Tot i que d'alguna manera hi hagi paratges en que tot això pot semblar infantil, o fins i tot naïf, convé assentar sòlidament aquestes nocions bàsiques per no caure en malentesos. En la segona meitat es veu la lògica de predicats, cosina germana de la teoria de conjunts encara que més emparentada amb la deducció natural. Tot plegat és una mateixa cosa mirada des de diferents punts de vista.

1.1 Definicions i Nomenclatura

Independentment del rigor amb el què es podria descriure aquesta teoria, cal retenir tres conceptes.

En síntesi, anomenem *conjunt*, C, a una col·lecció d'*elements*, e_1, e_2, i e_3 per exemple, diferents entre ells. Diem que l'element e_1 *pertany* al conjunt C, i ho escrivim $e_1 \in C$, o sigui, dient "pertany" al símbol $\in$. I també podem dir que C *conté* l'element e_1, encara que en aquest cas no introduïm cap símbol.

Per definició, un conjunt no té elements repetits. Si volem treballar o dir alguna cosa sobre un conjunt amb elements repetits, llavors haurem de dir que ens trobem davant d'un *multiconjunt*. Però bé, en aquestes alçades del discurs aquesta definició que no té més transcendència.

Conjunt, element, i el tercer concepte essencial és el concepte de *subconjunt*. Resulta gairebé intuïtiu entendre que S és un subconjunt de C, cosa que anotem $S \subseteq C$, si S és un conjunt d'elements que tots ells pertanyen a C. També diem llavors que S *està inclòs* en C.

O sigui, un conjunt no és el conjunt de tots els seus subconjunts possibles. I per tant, un subconjunt no pertany al conjunt en el qual s'ha definit, hi és inclòs. Una altra cosa seria parlar del conjunt de subconjunts possibles. És una cosa diferent.

La definició de subconjunt contempla els dos casos singulars en que S és el conjunt buit, que anotem $\varnothing$ i que per definició està inclòs en tots els conjunts, o l'altre extrem, en que S és igual a C. Ara bé, si volem indicar que S és un subconjunt de C tal com normalment ho imaginem, llavors diem que S és un subconjunt *propi* de C. De manera formal, utilitzant el símbol $\Leftrightarrow$ que vol dir *si i només si*,

$$S \subset C \Leftrightarrow S \subseteq C \text{ i } S \neq \varnothing \text{ i } S \neq C.$$

Caixa 1.1. *Definició de subconjunt propi.*

En aquest cas, podem dir també que S està inclòs en C *estrictament*. I ho representem sense la ratlleta, $S \subset C$.

De vegades interessa explicar que entre dos conjunts, un d'ells és subconjunt de l'altre sense la necessitat de reflectir la continència de quin és subconjunt de quin. Llavors diem que C i S són *compatibles*, i ho expressem amb el símbol $\sim$.

En la Caixa 1.2 es formalitza la compatibilitat entre conjunts.

$$X \sim Y \Leftrightarrow X \subseteq Y \text{ o bé } Y \subseteq X$$

Caixa 1.2. *Definició de compatibilitat entre conjunts.*

Ergo si són iguals, són compatibles. Altres exemples de compatibilitat entre

conjunts són els nombres enters i els nombres reals o les cadenes de caràcters i els números de telèfon.

El *cardinal* d'un conjunt és el nombre d'elements que conté. Un nombre enter no negatiu. En notació simbòlica, està clar que li podem dir n, minúscula perquè és un nombre i no un conjunt. I si volem expressar el cardinal en referència al conjunt C, llavors utilitzem la notació de C entre barres, és a dir $|C|$. És fàcil veure que si S és un subconjunt propi de C, i si $n = |C|$ i $m = |S|$, llavors $n > m$.

Com s'ha dit més amunt, els elements d'un conjunt han de ser diferents entre ells, i això implica que han de ser identificables. Aquesta deducció resulta profundament constructiva. Els conjunts són conjunts d'elements identificables.

Des d'un punt de vista molt més filosòfic, es podria dir que hi ha alguna relació entre la identificació i el número u.

1.1.1 Abstracció

La capacitat de percebre característiques comunes en objectes o fenòmens i anomenar-ho amb una paraula és l'abstracció. Com és una cadira? Hi ha cadires d'una pota, o tres o quatre, amb respatller i sense. De fusta, de metall i de plàstic... en fi, hi ha tantes variacions de cadires que sembla mentida que ens entenguem quan utilitzem la paraula cadira. Ja ho veieu. Cada paraula ja és una abstracció.

Es diu que un concepte és més abstracte que un altre quan el primer designa un conjunt del qual el segon n'és subconjunt. Aquest procés mental pot arrossegar pèrdua d'informació. El títol d'un text és una abstracció, ja que sota un mateix títol poden existir varis textos. L'abstracció sintetitza l'essència. I també una abstracció de gat és animal, i una abstracció de cadira és moble. De vegades, però, no és tan senzill. Ja es veu que quant més concret és un concepte, menys abstracte. L'abstracció és la relació més vertical entre paraules. L'índex d'un llibre, que materialitza una aproximació vertical al contingut, també és una abstracció.

En la programació orientada a objectes, la idea d'abstracció hi és implementada. I no és per casualitat que tots els llenguatges que es desenvolupen sobre aquesta tecnologia concloguin en la idea d'*objecte* com a terme al màxim nivell d'abstracció. Però això, tot, en el fons, són objectes.

L'abstracció és una operació mental magníficament complicada, que produeix satisfacció de resoldre. No és una operació calculable, ja que precisa d'una riquesa terminològica que és difícil estructurar. En qualsevol cas, vagi per endavant que és una de les feines més boniques de les que se n'ocupa una persona dissenyadora de bases de dades.

El procés que es realitza quan es defineix una abstracció es pot anomenar inferència. Hi ha lèxics que enriqueixen les semàntiques i aquest n'és un cas. Comprendre què vol dir una inferència ajuda a entendre una abstracció. Les abstraccions dels conceptes s'infereixen, per definició.

1.1.2 Descripció de Conjunts

Descriure un conjunt significa donar les dades necessàries per tal que l'interlocutor o lector comprengui sense cap ombra d'incertesa, quins són els elements que hi pertanyen. Bàsicament hi ha dues maneres de fer-ho. Una manera és enumerant els elements, i l'altra seleccionant d'entre els elements d'un conjunt més gran els que compleixin un predicat. Això vol dir definir un conjunt com la col·lecció d'aquells elements que compleixin una expressió avaluable com a certa o falsa.

Pel que s'ha vist fins ara, és ben clar que en cap moment s'ha fet menció de l'ordre dels elements dins el conjunt. Estructurem conceptes. L'ordenació dels elements d'un conjunt no forma part de la definició del conjunt. Per això, en llenguatge formal anotem amb claus, {}, la definició dels conjunts. Més concretament, la definició d'un conjunt, en qualsevol de les dues maneres de descriure'l, queda entre els símbols de clau d'obertura, {, i de tancament,}.

Filosòficament, es percep alguna relació entre l'ordenació i el número dos, ja que per a poder definir un ordre entre els elements d'un conjunt cal poder respondre quin dels dos elements és menor de cada parella d'elements possibles, que per cert, són $n(n-1)/2$. És fàcil de veure, si tenim n elements i cadascun pot fer parella amb els $n-1$ restants, llavors tenim $n(n-1)$ possibles parelles, però com que no considerem l'ordre entre les parelles, ens en queden la meitat. Un nombre enter segur, ja que o bé n o bé $n-1$ serà parell.

Per enumeració dels seus elements

A la manera més primitiva de descriure un conjunt en diem enumeració, i consta d'una seqüènciació dels elements. Per exemple, $C = \{e_1, e_2, e_3\}$. Llavors, encara que no volguem, si descrivim un conjunt per enumeració estem establint un ordre entre els seus elements. Com a conseqüència, el que estem descrivint és una seqüència, cosa que és més concreta que un conjunt perquè a més, té un ordre. Totes les seqüències són conjunts, però no tots els conjunts són seqüències. Per això fem servir les claus per denotar els conjunts.

Quan es pretén indicar que l'ordre de la seqüència és important, llavors enlloc de claus es fan servir parèntesis. L'expressió $C = (e_1, e_2, e_3)$ indica aquests elements en aquest ordre.

És clar que l'avantatge de descriure un conjunt per enumeració és l'evidència. Els elements que composen el conjunt que es descriu els tenim davant dels nostres ulls. I també és fàcil de veure que mitjançant l'enumeració només podem descriure conjunts finits. És a dir, que el cardinal sigui un número concret. Hi ha moltes disciplines que treballen amb conjunts d'infinits elements. I per tant, aquesta manera de descriure un conjunt no els hi serveix de massa.

A partir de propietats dels seus elements

Una altra manera de descriure un conjunt és establint un criteri que han de satisfer els elements d'un conjunt més gran per pertànyer al que s'està definint. És fàcil comprendre un conjunt definit per enumeració. En canvi, definir un conjunt com el subconjunt d'elements d'un altre conjunt més gran que satisfacin una condició requereix més intel·ligència per part del receptor de la definició, ja que li cal interpretar els criteris que defineixen la pertinença.

El conjunt de totes les persones que porten ulleres és un conjunt definit a partir de propietats dels seus elements. Està ben clar. El fet de no tenir la capacitat d'enumerar cada un dels elements del conjunt en qüestió no impedeix que poguem parlar-ne. Podem dir coses. Per exemple, hi ha menys persones que porten ulleres vermelles que persones que porten ulleres. O sigui, un subconjunt del conjunt de les persones que porten ulleres és el de les persones que porten ulleres vermelles. I això, també vol dir que si porta ulleres vermelles, porta ulleres.

Fixeu-vos bé amb l'estructura deductiva del paràgraf anterior. *Si porta ulleres vermelles implica que porta ulleres.* Hi ha alguna cosa espiritual en el fet d'associar el concepte de subconjunt, que és un sustantiu, amb la implicació que té com a conseqüència, que és un verb.

Però a part d'aquest fet, que per ell mateix ja és desconcertant, també aquí amagat hi ha el secret del perquè la teoria de conjunts té el prestigi que té, i per què hi ha tantes disciplines que deriven d'aquesta. Hi ha moltes persones que quan se'ls hi diu que la teoria de conjunts és molt important no entenen per què. Doncs aquí ho teniu. La manera de deduir que tenim els humans es pot explicar d'una manera gràfica. La teoria de conjunts té la transcendència que té perquè serveix per entendre com deduïm. I això és molt important.

Que un element pertanyi a un subconjunt implica que pertany al conjunt, si és un gat és un animal. I a la inversa, si no pertany a un conjunt, no pot pertànyer a cap dels seus subconjunts, si no és un animal, no pot ser un gat.

Per qualsecol conjunt d'objectes C, i subconjunt $S \subseteq C$, podem formular sistemàticament la proposició *tot* S és C però *no tot* C és S. O sigui, tota cadira és moble però no tot moble és cadira. O tot gat és animal però no tot animal és gat. I de manera dual, en el món de les accions, per qualsevol conjunt d'accions

C, i subconjunt $S \subseteq C$, podem formular sistemàticament la proposició *sempre* que es C es S, però *no sempre* que es S es C. O sigui, sempre es canta es parla, pero no sempre que es parla es canta. O sempre que es llegeix es mira, però no sempre que es mira es llegeix. És interessant observar la transformació de *tot* a *sempre* en la dualitat espai temps.

Descriure un conjunt per mitjà de les propietats dels elements que el composen és certament una manera més complicada que l'anterior. Tot i així, té l'avantatge de poder parlar de conjunts de molts elements, o infinits. I per contra, no és una definició constructiva en el sentit que requereix d'un conjunt més gran al qual es fa referència en la definició, que en el cas de l'exemple seria el conjunt de totes les persones. De totes maneres, com que aquest requeriment ve donat i no acostuma a representar cap esforç partir d'un univers de referència, aquesta segona manera de descriure conjunts és la més àmpliament utilitzada.

1.2 Operacions entre Conjunts

És del tot gratificant poder establir regles que operin amb conceptes tan abstractes com són els conjunts. Les regles que siguem capaços de formular en aquest nivell d'abstracció, gairebé inimaginable de superar, ens resultaran útils per totes les teories que se'n derivin, que són la majoria.

Aquest és l'enorme valor que tenen uns postulats tan senzills i tan convencionals com els que segueixen.

Les operacions entre conjunts són operacions com la suma o el producte, excepte que a la suma, per exemple, se li donen dos o més números d'entrada i ens retorna un número de sortida. Aquestes d'ara, en canvi, són operacions a les quals els hi donem d'entrada dos o més conjunts i de sortida també ens retornen un conjunt.

En endavant, direm C i D a dos conjunts qualssevol. Pels exemples gràfics, s'utilitzaran els dos conjunts que es mostren a la Figura 1.1.

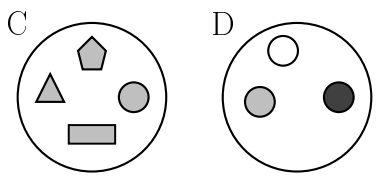


Figura 1.1: *Conjunts C i D utilitzats en els exemples.*

1.2.1 Unió

En la Figura 1.2 es mostra la imatge mental que convé retenir per la noció d'unió de conjunts.

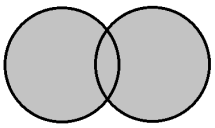


Figura 1.2: *Mnemograma per la unió de conjunts.*

El conjunt resultant de la unió de dos o més conjunts està composat pels elements que pertanyen a qualsevol dels conjunts donats d'entrada. La unió és una operació commutativa. Que els conjunts d'entrada es puguin commutar significa que la unió entre dos conjunts C i D es el mateix conjunt que la unió entre D i C.

Per a les representacions escrites d'aquesta idea s'utilitza el símbol $\cup$ en notació infixa, o sigui, entre els conjunts que es donen d'entrada a l'operació. Així doncs, en llenguatge formal definim la unió dels conjunts C i D com s'indica en la Caixa 1.3.

$$C \cup D = \{x \mid x \in C \vee x \in D\}$$

Caixa 1.3. *Definició de la unió de conjunts.*

Uns comentaris respecte aquesta definició per aclarir terminologia.

- Llegim la barra vertical dient "tals que", o "tal que", ja que en llenguatge formal no hi ha singulars ni plurals. En altres nomeclatures s'utilitza els dos punts enlloc de la barra vertical. En llenguatge formal, $\mid$ i $:$ són símbols sinònims.
- El símbol $\vee$ vol dir "o" en llenguatge formal de predicats. De fet, està íntimament relacionat amb el símbol mateix de la unió de conjunts. Fem servir $\cup$ pels conjunts, i $\vee$ pels predicats, però com es veurà més endavant, en essència són el mateix.
- Tota ella, doncs, es podria llegir com "La unió del conjunt C i el conjunt D és el conjunt format pels elements x tals que x pertany a C o x pertany a D" (en llenguatge formal es troben a faltar els pronoms).
- La definició conté dues expressions. La de la dreta del signe "=", i la de l'esquerra. És com el sintagma nominal i el sintagma verbal d'una frase. El signe d'igual aquí fa el paper del verb.

- A les variables que estan en expressions on s'utilitza tot el seu domini, com la x en l'expressió de la part dreta de la Caixa 1.3, se'ls hi diu variables *lligades*, o *tancades*, en l'expressió. Se les reconeix perquè són les que si canviéssim el seu nom en totes les seves aparicions, enlloc d'x l'anomenéssim y per exemple, el sentit de la definició global seguiria sent correcte. Són com variables locals en els llenguatges de programació.

De manera gràfica, en la Figura 1.3 es mostra la unió dels conjunts de la Figura 1.1.

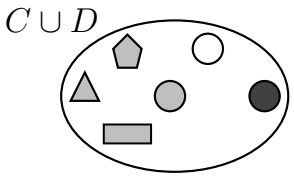


Figura 1.3: *Conjunt Unió $C \cup D$ dels conjunts de la Figura 1.1.*

Noteu que en aquest exemple s'il·lustra el fet que per definició un conjunt no pot tenir elements repetits. Això ha fet que hagi desaparegut un cercle gris. I també un exemple textual per no deixar dubte. Si $C = \{e_1, e_2, e_3\}$ i $D = \{e_3, e_4\}$, llavors $C \cup D = \{e_1, e_2, e_3, e_4\}$.

Tot seguit, s'expressa en llenguatge formal algunes propietats de la unió de conjunts. Només per fer gimnàs. Podríem pensar-ne altres.

- $C \cup C = C$.
- $C \cup \varnothing = C$.
- $S \subseteq C \Rightarrow S \cup C = C$.
- $|C \cup D| \leqslant |C| + |D|$.

1.2.2 Intersecció

En la Figura 1.4 es pot veure una idea gràfica de la intersecció de conjunts.

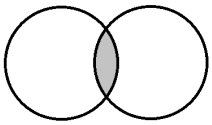


Figura 1.4: *Mnemograma per la intersecció de conjunts.*

El conjunt resultant de la intersecció de dos o més conjunts està composat pels elements que pertanyen a tots els conjunts donats d'entrada. La intersecció també és una operació commutativa.

Per a les representacions escrites d'aquesta operació s'utilitza el símbol $\cap$. Llavors, en llenguatge formal definim la intersecció dels conjunts C i D com es presenta en la Caixa 1.4.

$$C \cap D = \{x \mid x \in C \wedge x \in D\}$$

Caixa 1.4. *Definició de la intersecció de conjunts.*

Aquesta definició es podria llegir com "la intersecció del conjunt C i el conjunt D és el conjunt format per tots els elements x tals que x pertany a C i x pertany a D". El símbol $\wedge$ vol dir "i" en lleguatge formal de predicats. I com en el cas de la unió, està íntimament relacionat amb el símbol mateix de la intersecció de conjunts. Fem servir $\cap$ pels conjunts, i $\wedge$ pels predicats.

Altre cop, de manera gràfica, en la Figura 1.5 es mostra la intersecció dels conjunts de la Figura 1.1.

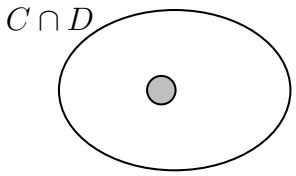


Figura 1.5: *Conjunt Intersecció $C \cap D$ dels conjunts de la Figura 1.1.*

Un exemple textual. Si $C = \{e_1, e_2, e_3\}$ i $D = \{e_3, e_4\}$, llavors $C \cap D = \{e_3\}$.

I també com abans, expressem en llenguatge formal algunes propietats d'aquesta operació. Per divertir-nos. Penseu-ne altres...

- $C \cap C = C$.
- $C \cap \varnothing = \varnothing$.
- $S \subseteq C \Rightarrow S \cap C = S$.
- $|C \cap D| \leq min(|C|, |D|)$.

1.2.3 Diferència

La imatge associada a la diferència de conjunts és a la Figura 1.6.

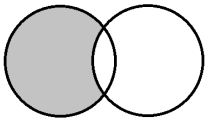


Figura 1.6: *Mnemograma per la diferència de conjunts.*

De vegades, tan sols amb aquestes imatges n'hi ha prou per inspirar raonaments vàlids.

El conjunt resultant de la diferència de dos conjunts està composat pels elements que pertanyen al primer i no pertanyen al segon dels conjunts donats d'entrada. La diferència de conjunts és una operació no commutativa.

Per a les representacions escrites d'aquesta idea s'utilitza el símbol $\setminus$. En llenguatge formal definim la diferència del conjunt C menys el conjunt D com s'introdueix en la Caixa 1.5.

$$C \setminus D = \{x \mid x \in C \wedge x \notin D\}$$

Caixa 1.5. *Definició de la diferència de conjunts.*

Aquesta definició es diu "la diferència del conjunt C menys el conjunt D és el conjunt format per tots els elements x tals que x pertany a C i x no pertany a D".

En la Figura 1.7 es pot observar la diferència entre els conjunts C i D.

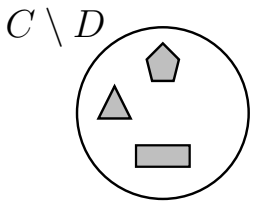


Figura 1.7: *Conjunt Diferència $C \setminus D$ dels conjunts de la Figura 1.1.*

I per l'exemple textual, si $C = \{e_1, e_2, e_3\}$ i $D = \{e_3, e_4\}$, llavors $C \setminus D = \{e_1, e_2\}$.

La diferència entre conjunts ens permet introduir la noció de *complementari*. Si S és un subconjunt de C, llavors el complementari d'S, que descrivim amb $\bar{S}$, és igual a $C \setminus S$. O sigui els altres de C.

Amb tot, podem dir coses com

- $C \setminus D = C \setminus \{C \cap D\}$.
- $S \cup \bar{S} = C$.
- $S \cap \bar{S} = \varnothing$.

1.2.4 Producte Cartesià

El producte cartesià de dos conjunts és una operació essencialment diferent de les que s'han vist fins ara. Perquè en les anteriors, els elements del conjunt resultant eren dels mateixos elements dels conjunts participants en l'operació.

El producte cartesià de dos conjunts és un conjunt format per uns elements que són parelles. Cada element del producte cartesià és una parella formada per un element del primer conjunt d'entrada, i un element del segon. O sigui, que per cada combinació formada per un element del primer conjunt i un element del segon tenim un element del producte cartesià. Que sigui commutativa recau en que ho sigui l'aparellament dels elements.

Observeu la imatge que hauria de servir per recordar un producte cartesià entre conjunts, a la Figura 1.8. Considereu que cada una de les dues línies gruixudes representa un dels dos conjunts. Cada marca que hi ha en les línies gruixudes significa un element. Llavors en el pla tenim un punt associat a cada parella d'elements dels conjunts inicials. És notable que per definir el producte cartesià cal seqüencialitzar els elements segons algun ordre.

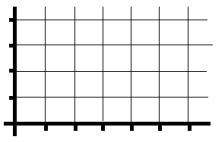


Figura 1.8: *Mnemograma pel producte cartesià de conjunts.*

És interessant observar que el fet de posar els elements d'un conjunt en una disposició seqüencial se suporta sobre la base de que siguin identificables, ja que una seqüència requereix d'un ordre, i per tant, també requereix una operació de comparació entre els elements del conjunt. En aquest extrem hi ha alguna cosa que grinyola. Un conjunt no té ordre per definició, però en canvi, a l'hora de calcular un producte cartesià, se suposa que els conjunts d'entrada a l'operació

es poden ordenar... Bé, en qualsevol cas, com que sempre treballarem amb conjunts ordenables, la cosa no té més transcendència.

Per a les representacions escrites d'aquesta idea s'utilitza el símbol $\times$. El producte cartesià dels conjunts C i D es defineix formalment com es mostra en la Caixa 1.6.

$$C \times D = \{xy \mid x \in C \wedge y \in D\}$$

Caixa 1.6. *Definició del producte cartesià de conjunts.*

En la Figura 1.9 es mostra el producte cartesià dels conjunts d'exemple.

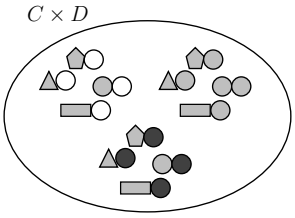


Figura 1.9: *Conjunt Producte Cartesià $C \times D$ dels conjunts de la Figura 1.1.*

I seguint amb l'exemple textual si $C = \{e_1, e_2, e_3\}$ i $D = \{e_3, e_4\}$, llavors $C \times D = \{\{e_1, e_3\}, \{e_1, e_4\}, \{e_2, e_3\}, \{e_2, e_4\}, \{e_3, e_3\}, \{e_3, e_4\}\}$. Fixeu-vos-hi bé, perquè cal saber descriure el resultat d'aquesta operació.

Encara un altre exemple. El conjunt dels nombres enters del 0 al 9 producte cartesià amb ell mateix és el conjunt dels nombres del 00 al 99. O sigui, totes les parelles possibles formades per un element del 0 al 9 i un altre element del 0 al 9. Aquesta reflexió és interessant. Si tenim un conjunt, el producte cartesià d'ell per ell mateix és com si tractéssim els elements del conjunt com si fossin dígits d'un sistema de numeració que enlloc de tenir 10 dígits bàsics en tindria tants com elements hi hagi al conjunt. I llavors el producte cartesià és com comptar fins a cent. Per qualsevol conjunt C, el conjunt de totes les parelles possibles d'elements de C és $C \times C$.

Algunes propietats del producte cartesià de conjunts són

- $C \times \varnothing = \varnothing$.
- $|C \times D| = |C|\ |D|$.

1.3 Lògica de Predicats

Parlar de la teoria de conjunts en un llibre de bases de dades es pot entendre. Al cap i la fi, les dades que es guarden a les bases de dades són conjunts, i per tant es veu una relació ben clara. Ara bé, la lògica de predicats sembla una mica més allunyada del tema.

Doncs bé, la lògica de predicats està íntimament relacionada amb l'Àlgebra de Boole per un costat, cosa que l'apropa a la computació i per tant a les bases de dades. A més, en capítols posteriors caldrà tenir clares algunes qüestions suposadament conegudes pel lector, i per això aquí en fem un repàs utilitzant un llenguatge comú a tots els àmbits com és la lògica de predicats.

1.3.1 Càlcul de Predicats

Un predicat és una expressió que pot ser avaluada com a certa o falsa.

Per exemple, "avui fa sol". O sigui, que la noció de predicat és anàloga a la d'expressió booleana en els llenguatges de programació. Aquesta expressió està formada de proposicions que fan el paper de variables booleanes, i d'operadors que es corresponen amb les operacions lògiques que es descriuen tot seguit. La definició d'aquestes operacions prové del càlcul proposicional d'ordre zero, que és una teoria que consta de nou principis que constitueixen els principis bàsics de la deducció natural.

Per definir el funcionament d'una operació s'utilitza una taula, que és la definició més detallada possible per qualsevol relació. S'anomenen taules de veritat pels valors del seu contingut. A les columnes de l'esquerra s'hi posa totes les possibles combinacions dels valors de les proposicions, i a la columna de la dreta s'hi posa la resposta de l'operació per la combinació de valors corresponent a la fila.

Les operacions de la lògica de predicats i les de l'Àlgebra de Boole són clavades. La diferència més notòria és que els possibles valors per una proposició (i per tant per un predicat) són *cert*, `C`, i *fals*, `F`. En canvi els valors possibles d'una variable booleana són {0,1}. Una altra diferència és que l'ús aritmètic de variables booleanes com la suma o el producte, o la mateixa propietat associativa de la suma respecte el producte és legítim. En canvi, amb proposicions no fem aquestes coses, perquè el càlcul proposicional és abans de la teoria de nombres. I per tant, no té sentit utlitzar una suma de proposicions. De totes maneres, la línia que separa ambdues teories és molt fina, i hi ha continguts, com per exemple les lleis de De Morgan, que ballen entre els dos móns.

Que dues proposicions siguin independents significa que cap d'elles condiciona el valor de l'altra. Això vol dir que es pot produir qualsevol combinació

de valors entre elles. Les proposicions "això pesa més d'un quilo" i "això pesa més de dos quilos" no són independents. Les proposicions "sóc fuster" i "em dic Joan" són independents.

Tot seguit s'introdueixen les tres operacions del càlcul de predicats.

Operació de negació lògica

L'operació de negació té com operand d'entrada una sola proposició. L'operador corresponent s'escriu $\neg$ com a prefix del predicat que es nega, i es diu "no". És a dir, $\neg p$ es llegeix "no p".

Sigui p una proposició. Llavors, la taula de veritat que caracteritza l'operació de negació lògica es mostra en la Taula 1.1.

p	$\neg p$
F	C
C	F

Taula 1.1: *Taula de veritat de la negació lògica.*

La propietat més rellevant que se'n desprèn del comportament d'aquesta operació és el fet que $\neg\neg p = p$. Aquest fet en molts llenguatges naturals porta greus problemes pel fenòmen de la doble negació... no he dit res. De fet, qualsevol pregunta formulada utilitzant la paraula *no* acostuma a conduir a confusions. Qualsevol programador, o analista, hauria d'evitar les preguntes negades. No s'entén? Doncs això.

I per la mateixa raó és de molt mal programador anomenar amb identificadors que comencin amb *no* les variables booleanes. Difícilment justificable, ja que sempre es pot ajustar el codi per deixar la variable en forma directa, no negada. I exactament el mateix per les sentències alternatives completes. Les que tenen *if then else*. Es bona pràctica posar les expressions de l'*if* en positiu.

Operació lògica disjuntiva

Aquesta operació també és coneguda com "o lògica". Està emparentada amb la unió de conjunts, vista en la Secció 1.2.1.

L'operació disjuntiva té com operands d'entrada dos predicats. Per la unió de predicats s'utilitza l'operador $\vee$ que es llegeix "o". Això és, $p \vee q$ es llegeix "p o q".

Si p i q són dues proposicions independents entre elles, llavors la taula de veritat que defineix la disjunció lògica és la que es pot veure en la Taula 1.2.

p	q	$p \vee q$
F	F	F
F	C	C
C	F	C
C	C	C

Taula 1.2: *Taula de veritat de la disjunció lògica.*

És a dir l'o lògica entre dues proposicions és certa si qualsevol de les dues ho és.

És una operació commutativa, com es pot deduir a partir de les files segona i tercera de la taula. A més és associativa, de manera que si p, q, i r són tres proposicions, llavors $p \vee q \vee r = (p \vee q) \vee r = p \vee (q \vee r)$. Això fa que la definició, més que per dos operands, valgui per dos o més.

Algunes propietats d'aquesta operació ens ajudaran a veure el parentiu que té amb la unió de conjunts.

- $p \vee \mathtt{F} = p$.
- $p \vee \mathtt{C} = \mathtt{C}$.
- $p \vee p = p$.
- $p \vee \neg p = \mathtt{C}$.

Operació lògica conjuntiva

Aquesta operació també és coneguda com "i lògica". Està emparentada amb la intersecció de conjunts, vista en la Secció 1.2.2. L'operació conjuntiva també té com operands d'entrada dues proposicions. Utilitza el símbol $\wedge$, que es llegeix "i". O sigui, $p \wedge q$, que es pronuncia "p i q".

La taula de veritat que defineix la conjunció lògica entre dues proposicions independents p i q es pot veure en la Taula 1.3.

p	q	$p \wedge q$
F	F	F
F	C	F
C	F	F
C	C	C

Taula 1.3: *Taula de veritat de la conjunció lògica.*

És a dir la i lògica entre dues proposicions és certa si ho són les dues. També és commutativa i associativa.

És important notar que els termes disjunció i conjunció són perillosos perquè és extremadament fàcil confondre'ls. És una confusió molt normal. Això és perquè els símbols de la unió $\cup$ de conjunts, l'operació $\vee$ entre predicats, inclús les mateixes paraules "o", i "unió", s'assemblen. Per altra banda, amb la intersecció passa el mateix. O sigui, "intersecció" s'assembla a "i", a $\wedge$, i a $\cap$. En canvi, amb els noms de les operacions, va al revés, la disjunció, que té una "i", és l'operació "o", i la conjunció que té una "o", és l'operació "i", o sigui, que alerta amb els noms de les operacions lògiques. Aquesta confusió és tant habitual que hi ha qui anomena les operacions lògiques amb els noms de les operacions de la teoria de conjunts.

Lleis d'Augustus de Morgan

Tanquem el tema del càlcul de predicats enunciant les lleis de De Morgan. Aquí es mostra en tota la seva plenitud la dualitat entre la unió i la intersecció de conjunts, o entre les operacions lògiques disjuntiva i conjuntiva.

Les lleis de De Morgan es postulen sobre àmbits diversos, cosa que reflecteix l'analogia entre ells. El seu interès rau en el fet de relacionar formalment les operacions de la lògica de predicats, o de la teoria de conjunts. Diuen que la negació d'una conjunció és una disjunció, i a la inversa.

1. $\neg(p \vee q) = \neg p \wedge \neg q$.
2. $\neg(p \wedge q) = \neg p \vee \neg q$.

Caixa 1.7. *Lleis de De Morgan.*

Amb tot el rigor, la negació de la disjunció és la conjunció de les negacions. I a la inversa, la negació de la conjunció és la disjunció de les negacions.

Suposem que p vol dir "plou". I q vol dir "vaig a la platja". La disjuntiva seria que "plou o vaig a la platja". I per negar-la hauríem de dir "no plou i no vaig a la platja". Té una lògica aclaparadora. Si a o b és mentida, llavors a és mentida i b també es mentida, perquè si una de les dues coses fos veritat, llavors una o l'altra també seria veritat.

Les lleis de De Morgan són profusament utilitzades en multitud de demostracions en la lògica matemàtica. I també en l'electrònica digital. Utilitzant aquestes lleis, sóm capaços de fabricar tots els tipus de portes lògiques dels circuits, a partir de tan sols un tipus, la conjunció negada, o porta *nand*.

1.3.2 Relació amb la Teoria de Conjunts

Com s'ha introduït en la Secció 1.1.2, hi ha una correspondència entre dos fets que semblen pertànyer a universos diferents. Que una proposició n'impliqui una altra, p implica q, vol dir que el conjunt de veritats que anomenem P és un subconjunt del conjunt de situacions en les que Q és veritat. Aquesta associació entre la teoria de conjunts i la lògica de predicats és un pilar fonamental sobre el que se sustenta gran part del coneixement analític. I com que sabem que si un element pertany a un subconjunt ha de pertanyer al conjunt, podem afirmar el que formalment estableix la Caixa 1.8.

$$p \rightarrow q \Leftrightarrow \bar{P} \cup Q$$

Caixa 1.8. *Relació entre la lògica de predicats i la teoria de conjunts.*

L'equivalència de la Caixa 1.8 es llegiria com, p implica q vol dir el mateix que no p o q. Per entendre'ns, suposem que p vol dir "és bomber", i q vol dir "és persona". És clar que la part esquerra de l'expressió és compleix ja que efectivament si és bomber, és persona. La part de la dreta, que vol dir el mateix dit d'una altra manera, diu que una de dues, o no és bomber, o és persona. Observeu la utilització del complementari de P a la part dreta de l'expressió. Complementari s'associa a negació.

En aquest primer capítol introductori s'ha mostrat les nocions més elementals de la teoria de conjunts, les operacions emblemàtiques d'aquesta teoria, en les quals s'hi ha inclòs el producte cartesià, així com de la lògica de predicats. Aquests conceptes són intensament utilitzats en els capítols següents, i per això, encara que siguin conceptes senzills haurien de quedar sòlidament fonamentats.

Capítol 2

Anàlisi del Projecte

El procés d'anàlisi és un procés d'absorció de coneixement. En cap cas es tracta d'una actitud creativa. És una postura observadora. L'analista ha de tenir una mentalitat esponjosa, molt preparada per comprendre tot allò que se li explica amb la finalitat d'immergir-se en l'univers de l'aplicació. En aquest capítol es toquen varis aspectes que cal considerar.

2.1 Sistemes Gestors de Bases de Dades

Un Sistema Gestor de Bases de Dades, SGBD, o *Database Management System*, DBMS, és un programa informàtic que gestiona bases de dades.

En les bases de dades relacionals, una informació elemental es representa en una cadena de caràcters, per exemple, que es guarda com a valor d'un atribut per un element concret d'una relació concreta que forma part d'una base de dades concreta. Aquesta és l'estructura jeràrquica. Normalment un gestor de bases de dades gestiona varies bases de dades alhora, i a totes elles les anomenem clúster. Un *clúster* és una instal·lació d'un sistema gestor de bases de dades

A grans trets, els programes informàtics es poden agrupar en tres categories. Els programes *batch*, els *serveis*, i els programes d'*interfície*.

Els programes batch són molt anteriors als altres dos tipus. Originàriament, a part del sistema operatiu, eren l'únic tipus de programa que hi havia. Es caracteritzen perquè comencen i acaben. No tenen gairebé interfície. Sovint corren de fons, sense intervenció de l'usuari, a partir d'una programació de tasques, o formant part de processos més grans. Programes de backup o de formateig d'arxius són programes batch. La seva naturalesa rau en la màquina

de Turing, aquella que tenia una cinta d'entrada i una estat intern i segons l'entrada i l'estat actual produïa una sortida i un canvi d'estat.

Els serveis són un tipus de programes activats per altres programes, no per usuaris finals. Antigament se'ls anomenava programes residents a memòria. És habitual que s'arrenquin cada cop que s'engega la màquina, o cada cop que entra un nou usuari. Llavors es queden a memòria i esperen peticions per algun mecanisme. En aquest tipus de programes és on s'emmarquen els SGBDs. La seva manera de fer va adquirir protagonisme amb l'emergència contundent de les xarxes. I a partir del moment que es van dedicar a escoltar peticions d'altres ordinadors es van anomenar serveis. És inqüestionable que l'hegemonia que hi ha avui dia amb la preponderància dels ordinadors petits en xarxa ha eliminat les arquitectures basades en supercomputadores. En altres paraules, la unió fa la força. Avui dia les xarxes manen. En definitiva, l'autèntica intel·ligència resideix en els actors petits, però ben comunicats. Les altes freqüències.

Els d'interfície, GUI de *Graphic User Interface*, o sigui de diàleg, són els que han evolucionat més des del principi de la informàtica. Dins aquest àmbit hi ha un salt històric important marcat per l'aparició de pantalles gràfiques, o sigui que podien pintar un sol punt del monitor. Això, introduït per Steve Jobs a mitjans dels vuitanta, marca un canvi de proporcions incommensurables. El diàleg home màquina perd la sincronicitat, ja que fins llavors les coses es deien d'una en una. La interacció doncs, passa d'una a dues dimensions. De línia de comandes a interfície gràfica. El terme *línia* ha de fer pensar en una dimensió. El terme *gràfica* en dues. Apareixen les finestres, i el ratolí. Ja es poden dir dues coses a l'hora. A partir d'això, els programes comencen a ser conduïts per casos, de manera que tenen un bucle infinit en el seu nucli, i fins que l'usuari no diu que es tanqui, el programa no es para.

Els programes que actualment no tenen interfície gràfica, o sigui ni ratolí ni finestres, funcionen per mitjà de línia de comandes. Es tracta de programari no dirigit a usuaris finals. O sigui que els seus usuaris són o bé tècnics informàtics, o bé altres programes que fan les feines d'interfície pels usuaris finals.

De tot plegat, se n'ha de treure la certesa fonamental que un gestor de bases de dades no contempla cap interacció amb l'exterior. No espera cap diàleg amb usuaris finals. Això és important a l'hora d'estructurar la feina. En el desenvolupament d'una base de dades no s'ha de tenir en compte l'aspecte de la interfície.

2.1.1 Arquitectura Client-Servidor

La idea d'arquitectura *client-servidor* significa que el SGBD com a servei s'executarà en l'ordinador, o *host*, servidor. Per això també se li diu *back-end*. Perquè és l'extrem llunyà de la connexió, respecte els usuaris.

En canvi, l'aplicació d'interfície que es dedicarà a traduir les peticions d'aquests usuaris, es diu *client*, o *front-end* (en anglès la paraula *end* també es pot entendre com extrem).

En la Figura 2.1 es pot veure els dos components, i el medi que els uneix, que normalment serà internet, intranet, o qualsevol altre tipus de connexió.

Figura 2.1: *Arquitectura Client-Servidor.*

Microsoft Access va introduir la primera interfície gràfica per una base de dades. En certs aspectes va ser un èxit, prova n'és la gran quantitat d'aplicacions GUI que després li han seguit la línia. Tot i així, també ha tingut la seva part corrosiva. Les persones que aprenen bases de dades amb aquest sistema perden les nocions de client i servidor. I no distingeixen quines de les opcions dels menús de l'aplicació formen part del nucli de l'SGBD, i quines són utilitats addicionals afegides, que no tenen res a veure, o ben poc, amb la base de dades real.

2.2 Principi d'Independència de les Dades

Una diferència important entre les aplicacions que gestionen bases de dades i les que realitzen altres tasques és la capacitat de reconstruir-se des del no res.

Els programes de càlcul poden recompilar-se i reconstruir-se els cops que faci falta perquè no tenen cap més compromís amb l'exterior que les dades que gestionen en les seves interfícies. Un inconvenient de les aplicacions de gestió de bases de dades doncs, és que el compromís amb l'exterior no només és amb les dades que introdueix l'usuari, sinó que també amb l'estructura de la base.

Quan una aplicació de gestió de bases de dades es posa en producció, ja mai més pot tornar a començar de zero, perquè un cop hi hagi dades ja no podrem rectificar les estructures amb facilitat. Per això hi ha el que es diu principi d'independència de les dades, i de fet és una de les raons que justifiquen l'existència dels SGBDs. Els canvis en l'estructura de la base no han de provocar canvis en els programes clients que en fan ús.

L'estat desitjable al que s'hauria d'arribar és el de poder efectuar des de la línia de comandes de la consola de l'SGBD la totalitat de les accions que finalment faran les aplicacions client, sense finestres ni colors, ni fonts de text. En definitiva, oferir unes determinades funcions com interfície de la base de

dades. I han de ser aquestes mateixes funcions les que utilitzin les aplicacions d'interfície. El concepte és anàleg al d'una classe en java. L'estat desitjable és la implementació d'un conjunt de mètodes públics.

En la Figura 2.2 es mostra la clàssica estructura de ceba que il·lustra el funcionament de les aplicacions de bases de dades. En el nucli hi ha la circuiteria, en la que se suporta la microprogramació, el llenguatge assemblador, el sistema de fitxers,... i tot el que composa el sistema operatiu. I sobre aquesta capa, els SGBDs.

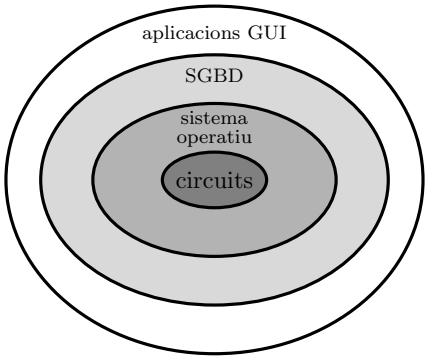


Figura 2.2: *Model en forma de ceba d'un sistema informàtic.*

Finalment, els SGBDs són els que suporten les aplicacions d'usuari final.

Que en la Figura 2.2 una capa se suporti sobre una altra significa que la interna ofereix operacions primitives o elementals a l'externa.

En definitiva, és com si la base de dades fos una classe d'un llenguatge de programació, que ofereix un conjunt d'operacions públiques com afegir, modificar, consultar o esborrar qualsevol dels objectes del domini, a més d'aquelles consultes que s'hagi demanat en la definició de requeriments.

2.2.1 Arquitectura a Tres Nivells

Així com l'arquitectura client servidor té un lligam directe amb allò material, és una arquitectura física, també podem parlar d'una arquitectura lògica, o patró arquitectònic al qual convé ajustar-se a l'hora d'implementar una base de dades.

Aquesta forma de segmentar els mòduls que integren un SGBD resideix en el costat del servidor. Els tres nivells es diuen nivell físic, lògic, i d'aplicació, tal com es pot contemplar en la Figura 2.3. En el nivell físic hi ha tot allò que té que veure amb la interacció entre l'SGBD i el sistema operatiu, així com la part del SGBD que suporta les bases de dades que finalment utilitzaran les aplicacions.

Això inclou les metadades, l'intèrpret del llenguatge estructurat de consultes que normalment disposa també d'un optimitzador de consultes, i anant més enllà podríem parlar de gestors de concurrència, de replicació, de seguretat, així com molts altres mòduls que en la Figura 2.3 no apareixen perquè queden fora de l'abast que es pretén exposar en aquest llibre.

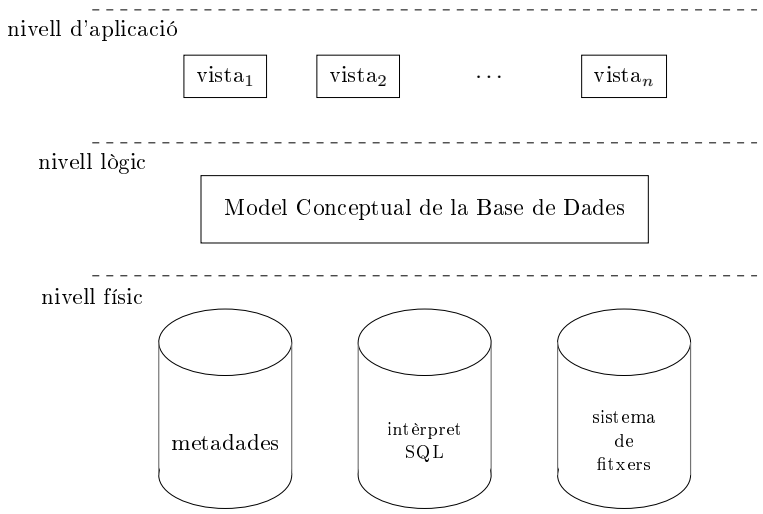


Figura 2.3: *Arquitectura a tres nivells.*

Una característica important del nivell físic és el fet de ser transparent per qui desenvolupa una base de dades. No hi té cap interacció. I aquí es dóna a conèixer perquè sempre convé saber què hi ha darrera les eines que s'estan utilitzant, però no té més impacte en el disseny i implementació de les bases de dades.

El model conceptual d'una base de dades forma el nivell lògic en una arquitectura a tres nivells, tal com s'il·lustra en la Figura 2.3. El nivell lògic és la base de dades que implementem els usuaris dels SGBDs. Aquest nivell materialitza la independència de les aplicacions respecte el suport físic, de manera que el model conceptual ha de ser capaç d'absorvir els possibles canvis en la forma física d'emmagatzemar les dades sense que aquests canvis transcendeixin al nivell d'aplicació.

El nivell de vistes, o d'aplicació representa el compendi d'utilitats que la base de dades proporciona a les aplicacions d'interfície, que són les que en última instància utilitzarà l'usuari final.

2.3 Immersió en el Domini

Gairebé es podria dir que és impossible fer una base de dades sense aprendre alguna cosa. En els primers instants, quan el projecte comença a concebir-se, abans que res, és important acumular el màxim coneixement de tot el que l'afecti.

Forma part del domini del problema tota aquella informació externa que pugui condicionar el contingut de la base de dades. En particular, dades relatives a la legislació vigent sobre el tema. La dissenyadora d'una base de dades, que en la fase inicial ha de fer d'analista, ha de ser una persona que tingui interès en tot allò que pugui afectar les restriccions que imposa la base sobre les dades.

Portat a l'extrem, si parléssim d'una base de dades per gestionar la lliga de futbol convindria que d'alguna manera contingués el coneixement del reglament oficial. Així ens podria prohibir que en un partit juguessin dotze jugadors, per exemple. Això ens ajudaria a mantenir la integritat de la base.

En el moment de realitzar el model de domini, la persona dissenyadora d'una base de dades ha d'estar disposada a aprendre nova terminologia de l'entorn on està a punt d'involucrar-se. Com s'utilitzen els mots, i quins conceptes van associats a quins altres. Una curiositat que es produeix és que dins d'àmbits concrets, els adjectius s'utilitzen com sustantius. Cal fer-ho. Per exemple, "central" en l'entorn del futbol vol dir una cosa molt diferent que en l'entorn de la fusteria. De bell antuvi, desconfiarem d'una analista que li costin aquest tipus d'assimilacions.

Qui dissenya una base de dades s'ha de preguntar què depèn de què. Per exemple, el cognom d'una persona no depèn del municipi on viu. En canvi, la comarca on viu sí que depèn del municipi. Dit així, sembla extrany... que la comarca depèn del municipi... Però si tenim al cap els diferents valors de municipis i comarques que pugui haver-hi, llavors està clar que sempre que el municipi sigui per exemple *Sitges*, la comarca serà el *Garraf*. Vist des d'aquesta perspectiva, o sigui, entès com a valors que poden prendre unes variables, sí que té sentit dir que la comarca depèn del municipi. En canvi, al revés diríem que el municipi vé restringit per la comarca. Aquestes relacions de pertinença són fonamentals pel disseny de bases de dades.

Una dada no depèn d'una altra si qualsevol combinació de valors possibles entre les dues té algun sentit. Dependència vol dir que un valor condicioni un altre. Exactament igual que entre proposicions en el capítol anterior.

Per arribar tan lluny com sigui possible en l'estudi de l'entorn, és molt convenient comportar-se en una actitud humil en el diàleg amb el professional que després de tot serà l'usuari de l'aplicació. Cal observar com diu les paraules, i les preposicions. Fer-li preguntes com si qualsevol element d'un conjunt pot estar relacionat amb qualsevol d'un altre.

De vegades ho tindrà clarissim. I de vegades no. Quan li preguntem si és necessari disposar d'una informació secundària per poder existir una de primària no sabrà què respondre i dirà que mai s'ho havia plantejat. A voltes aquests dubtes es propaguen fins a la interfície de l'aplicació i s'acaben resolent amb caselles d'activació col·locats en pestanyes d'opcions d'algún botó de propietats.

De tot plegat se n'extrau un document al que s'acostuma a anomenar *definició de requeriments*.

2.4 Definició de Requeriments

Una definició de requeriments és l'objectiu principal que ha d'aconseguir trenar l'analista que té l'encàrrec de realitzar una aplicació amb una base de dades per gestionar un projecte. Com és lògic, però, els professionals de l'àmbit que correspongui no han sentit parlar d'aquest document, de manera que l'analista abans de demanar-lo, cosa que ha de fer inqüestionablement, ha d'explicar en què consisteix. Això porta a un document amb coneixements emesos pel professional usuari, o client, però en canvi estructurat sota les directrius de l'analista.

Tot plegat provoca que el diàleg entre client i analista en els moments inicials sigui intens. L'analista n'ha de treure el màxim profit.

L'estructura d'una bona definició de requeriments hauria de passar pels següents punts.

- Quina és l'organització interessada en el desenvolupament del projecte. Quina és la seva activitat. Quants anys porta fent aquesta tasca. També interessa molt si l'aplicació que es proposa en té alguna de predecessora, i en cas negatiu, com es realitzava la feina que ara es pretén automatitzar.
- Conceptes que ha de manegar la base de dades. Com s'identifiquen.
- Impacte del temps en els diferents objectes. Distingir informació històrica de la funcional.
- Tipus d'usuaris i interessos de cada un d'ells.
- Aplicacions que se li pretenen donar. Informes que es desitja obtenir. Accions que es puguin realitzar.

2.4.1 Informació Estadística

Malgrat sigui més complicat obtenir-les, les estadístiques que ens permeten preveure com de sovint es produiran casos que requereixin informacions addicionals són extremadament importants de cara el disseny.

Això és, imaginem que volem guardar les dades d'un conjunt de persones, i en el cas que la persona sigui una dona ens interessaria registrar si ha tingut fills, i qui són, si també els tenim guardats entre les persones de la base de dades. Llavors, la proporció de persones inserides a la base que siguin mares en condiciona l'estructura.

Si gairebé totes les persones són mares, podem guardar la informació juntament amb la que guardem per les persones en general, deixant les dades corresponents buides quan la persona no sigui una mare. I com que estem suposant que la major part ho seran, desaprofitaríem poc espai, i per tant seria una solució admissible.

En canvi, si en la base de persones s'insereixen mares només en casos excepcionals, llavors seria millor guardar-se la informació relativa a ser mare en un lloc apart, de manera que no malaguanyaríem espai per les persones que no ho fossin.

És ben clar que la proporció de registres que requereixin informació addicional determina el lloc on guardar-la.

2.5 Trajectòria de l'Error

No és una bona cosa resoldre confusions que l'usuari final pugui tenir. Per exemple, imaginem una base de dades de receptes de cuina. Les receptes de cuina tenen uns identificadors confosos. Hi ha més d'una recepta amb el mateix nom, com *macarrons a la bolonyesa*. Llavors, no és una missió de la persona dissenyadora discernir-les. L'actitud més adient que hauria de realitzar una aplicació és la de retornar l'error en la identificació a l'usuari. Una base de dades té mecanismes per distingir qualsevol tipus d'objecte. Però tot i així, no és convenient solucionar problemes prenent decisions per iniciativa pròpia. És a dir, la base de dades ha de reproduir les confusions de l'usuari final. Això fa que aquest entengui millor el sistema. Qualsevol usuari entén perfectament que no pot anomenar igual a coses diferents. Cal deixar que sigui ell qui decideixi la manera d'identificar els objectes. Que sigui ell qui, a falta de més imaginació, acabi anomenant *marcarrons a la bolonyesa 1* i *macarrons a la bolonyesa 2* a les seves receptes.

En la majoria de casos, i no només en l'entorn de bases de dades sinó de qualsevol tipus d'aplicació, el tractament més segur enfront d'un error és comunicar-lo. Sense prendre cap decisió per solventar-lo.

2.6 Exemple per un Club Esportiu

Com a fil conductor al llarg dels propers capítols es proposa desenvolupar una petita aplicació per gestionar un club esportiu. La finalitat és més il·lustrativa que funcional. Això fa que moltes dades s'hagin obviat.

Es tracta de mantenir la informació sobre quins esports fa cada soci, així com de comptabilitat en general per l'empresa, o sigui informació dels treballadors i dels seus caps.

2.6.1 Requeriments per al Club Esportiu

A un tipus d'usuari li diem rol. De tots els usuaris es registra el número de passaport, nom, cognom, una adreça de correu electrònic, i la seva ciutat. A més, per cada persona podem guardar un nombre indeterminat de números de telèfon. Per informes estadístics, i per poder fer estudis d'ampliació del club, convé guardar-se el nombre d'habitants de les ciutats i la comarca.

En la Secció 7.3 es veurà la manera de crear-los i donar-los-hi permisos per poder fer les accions que s'enumeren a continuació.

Soci

De cada soci es registra la seva data d'alta al club, que interessarà a l'hora de fer descomptes pels socis més antics. Les accions pròpies dels socis s'enumeren tot seguit.

1. Consultar la informació dels esports, que integra:
   - Preu mensual que costa la pràctica de l'esport.
   - Quota que paguen ells en concret si hi estan inscrits, ja que el club pot fer ofertes a determinats socis.
   - Quants jugadors falten per formar un nou equip, si és un esport amb varis participants per equip.
2. Posar-se en contacte amb altres socis. Això significa que el club manté una relació de quins socis coneixen a quins altres.
3. Inscriure's o donar-se de baixa a qualsevol esport.

Treballador

A l'empresa hi ha tres departaments. Els treballadors poden ser comercials, administratius, o bé preparadors físics que anomenem entrenadors.

Els comercials poden consultar els correus electrònics dels socis.

Els administratius poden registrar el pagament de nòmines dels treballadors. I també poden fet altes i baixes de socis, i modificacions de les seves dades.

Els entrenadors poden veure la quantitat de socis que practiquen cada esport.

Administrador

L'administrador pot fer altes, baixes, i modificacions de tots els objectes de la base de dades. Això contempla els socis, treballadors, ciutats, comarques i esports. Pels socis i treballadors haurà de crear un usuari amb els permisos corresponents per poder fer el que es requereix més amunt.

En aquest capítol s'ha vist qüestions diverses relatives a com s'ha de plantejar un projecte per una aplicació de gestió d'una base de dades. En particular, s'ha mostrat conceptes clàssics de l'arquitectura de l'aplicació. S'ha fet una breu reflexió de l'impacte que pot tenir l'estadística en el model, i s'ha presentat el concepte de definició de requeriments. Finalment, s'ha introduït un projecte senzill que s'utilitzarà en endavant com exemple en cada fase de la realització.

Capítol 3

Disseny de Bases de Dades

L'objectiu final del procés de disseny d'una base de dades consisteix en l'elaboració d'un diagrama, amb la documentació associada, que anomenem *model Entitat Relació*, model ER en endavant. Per l'extensió que requereix exposar tot el que involucra els models ER, el Capítol 4 complet es dedica a detallar-ne els conceptes.

Que quedi clar. Fer un disseny d'una base de dades vol dir fer un model ER, amb la seva documentació.

Fem un incís per parlar del terme *diagrama*. Utilitzem aquest mot per indicar que té una estructura no seqüencial. La paraula diagrama ha de fer pensar en descripció en més d'una dimensió. Comença com *diagonal*, com si tinguessin alguna cosa en comú. Diagrama vol dir una informació plana, més que lineal. Calen dues dimensions com a mínim per poder descriure un diagrama. O sigui, una representació que es desenvolupa tant en horitzontal com en vertical, com un mapa.

No estaria de més que us paréssiu a pensar amb les banderes dels països del món. Classificar-les en lineals i planes. Les banderes lineals només varien en una direcció. És el cas de la bandera catalana, l'espanyola, l'alemana o la francesa. En canvi, la bandera anglesa és una bandera plana, ja que es defineix en dues dimensions. Com la ikurrinya, l'australiana, o la japonesa. Observeu que podem fer banderes lineals tan llarques com volguem, en canvi, amb les planes no, cosa que té més transcendència del que ara pot semblar, i que no queda tan sols en una simple anècdota.

Però bé, tornem al tema. Fem servir diagrames a l'hora d'explicar el disseny d'una base de dades per no comprometre'ns amb cap ordre entre els elements que composen la idea que volem expressar. Així, apareix la noció de graf. Per això convindria en aquest punt aclarir alguns extrems d'aquesta terminologia.

Utilitzem el mot *graf* per indicar un conjunt de punts, o *nodes*, que poden relacionar-se d'alguna manera que indiquem per mitjà de línies que els uneixen. Seria bo que a partir d'aquesta indicació, us féssiu una idea de l'abstracció que aglutina aquesta estructura mental.

Si les línies que uneixen els nodes d'un graf tenen direcció, o sigui són fletxes, llavors es diuen *arcs*. I si no, *arestes*. Podem dir que els arcs o les arestes uneixen, connecten, associen, vinculen, lliguen o enllacen els seus dos nodes. El *grau* d'un node en un graf és el nombre de veïns que té.

En la teoria de grafs, es distingeixen dos tipus de grafs. Grafs *dirigits*, i grafs *no dirigits*. Els dirigits són els que les línies que uneixen parelles de nodes són fletxes, o sigui els nodes són units per arcs. A voltes, per diferenciar-los dels no dirigits, a aquests se'ls anomena *dígrafs*. Així s'indica que les connexions tenen direcció. Els grafs dirigits són els que ens interessen aquí, o sigui els dígrafs, encara que els anomenarem grafs perquè no indueix a confusió. Tots els que utilitzarem seran dirigits. Els que els arcs connecten un node predecessor amb un successor, és a dir, els que la parella de nodes que formen cada unió és una parella ordenada i en la què direm que un node *apunta* un altre. En la Figura 3.1 es mostra la idea de graf que cal retenir.

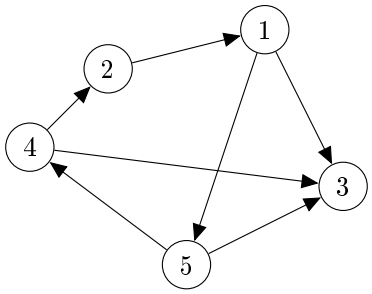


Figura 3.1: *Graf de 5 nodes i 7 arcs*

A partir d'aquestes definicions no hauria de ser difícil imaginar-se la definició de *camí en un graf* com una seqüència de nodes tals que existeixen les arestes que els uneixen, consecutivament. Per exemple, en el graf de la Figura 3.1 un camí podria ser 1-5-3.

En els graf dirigits, com ara el de la Figura 3.1, és poc clar si el concepte de camí requereix que la seqüència d'arcs tinguin la mateixa direcció que la seqüència de nodes que visita. El sol fet de parlar de grafs dirigits fa que les definicions es compliquin. Per exemple, per un graf no dirigit podem dir fàcilment si és *connexe* o no. Un graf no dirigit connexe és aquell que per qualsevol parella de nodes hi ha algun camí que porti de l'un a l'altre. Té gràcia que definim la connectivitat de grafs quan en la vida real, a un graf no connexe, o sigui per exemple amb dues components connexes ho anomenaríem dos grafs. Bé, això pel que fa a grafs no dirigits.

En canvi, pel cas de grafs dirigits cal distingir el concepte de connectivitat en tres possibilitats. Un graf dirigit pot ser *fortament* connexe, *unilateralment* connexe, o *dèbilment* connexe. Això vol dir, en el primer cas que per qualsevol parella de nodes hi ha un camí per anar de l'un a l'altre i també de l'altre a l'un. En els grafs unilateralment connexes per qualsevol parella de nodes existeix un camí que va de l'un a l'altre, o bé de l'altre a l'un. I un graf dirigit dèbilment connexe és aquell que si ignorem les direccions de les fletxes, és connexe.

En la Figura 3.2 hi ha exemples de les tres possibilitats. Observeu com la connectivitat més forta es produeix en el cas *(a)*, seguit del cas *(b)* on podem imaginar-nos la noció de node *font* i node *pou*, i el cas *(c)* on queda ben clar que el node 1 és un node font, i els altres dos són nodes pou.

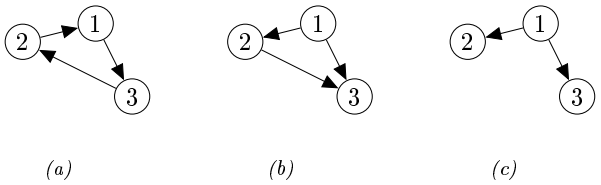


Figura 3.2: *(a) Fortament connexe; (b) Unilateralment connexe; (c) Dèbilment connexe.*

I apretant una mica més, podem endevinar la definició de *cicle*, com a camí que acaba en el mateix node que comença. I mira quina cosa, els cicles comencen i acaben en qualsevol dels seus nodes... Hauria de veure's molt clar que un cicle en el graf de la Figura 3.1 podria ser 5-4-2-1-5.

Un *arbre* és un graf no dirigit, connexe i acíclic, o sigui, que no té cap cicle. Aquest terme, malgrat estar definit en grafs no dirigits s'utilitza sovint pels dirigits ignorant les direccions dels seus arcs. En la Figura 3.3, es mostra l'aspecte que tenen.

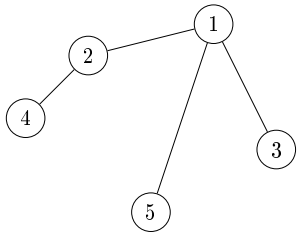


Figura 3.3: *Graf acíclic connexe, o sigui arbre.*

En la teoria de grafs, precisament perquè no els sabem gestionar fàcilment, existeixen una enorme quantitat de definicions. Penseu que ni tan sols tenim cap sistema d'enumeració que ens permeti associar un graf a un número. Penseu en això i intenteu inventar-lo. Alguna manera d'identificar els grafs en ordre. Per exemple, podríem dir que el graf número 1 és el graf format per un node i cap aresta. El 2, podria ser el de dos nodes aïllats, és a dir, també sense cap aresta.

El graf número 3 podria ser el que té dos nodes i una aresta que els uneix, però llavors, el graf amb tres nodes aïllats no està clar quina posició ocuparia en la seqüència... bé, deixem-ho córrer.

Finalment, per encarar el disseny de bases de dades cal el concepte de node *terminal*. és a dir, el que tan sols està connectat a un altre node del graf. Tenint en compte que, com s'ha dit, estem parlant de grafs dirigits, els nodes terminals poder ser fonts o pous, segons si l'únic arc que els connecta a la resta del graf va cap a ells, o surt d'ells. En els dos casos parlarem de nodes terminals, encara que les funcions que realitzin siguin molt diferents.

Un cop es dóna per correcte un disseny, el model ER ha de ser transformat o traduït al model relacional per començar la implementació.

3.1 Models de Dades

Els models de dades s'acostumen a presentar com a grafs. Cada node simbolitza un concepte de l'univers de l'aplicació. Un concepte que en definitiva unifica un grup heterogeni de dades relatives a un mateix tipus d'objecte.

No estem en un context massa rigorós. De models de dades n'hi ha de molts tipus. Qualsevol es pot inventar el model que li resulti convenient pel que pretén expressar. Un diagrama de classes, en l'entorn de la programació orientada a objectes, o el mateix model entitat relació en el que ens disposem a immergir, són diferents expressions de models de dades.

Els conceptes centrals que estructuren el domini acostumen a ser estables, Els terminals, en canvi, és més probable a continguin informació històrica. És important comprendre profundament la diferència entre dades que guarden temps, i dades que es mantenen constants. Grans bases de dades tenen com a concepte central de referència el rellotge o el calendari. És en aquests escenaris on interessa considerar l'índex de volatilitat i d'activitat de cada objecte. També hi ha bases de dades que registren informació sobre aquests lligams. L'índex d'activitat d'una base de dades ens explica com de sovint és consultada. L'índex de volatilitat, com de sovint neixen i moren els registres. Tot plegat condiciona de manera directa les estratègies pel sumministrament d'informació que finalment és el motiu pel que volem la base de dades. Un valor que varia molt sovint, molt, no convé calcular-lo quan se sol·llicita, cal tenir-lo actualitzat en tot moment.

De fet, tan important són els índexos d'activitat i volatilitat de cada una de les parts de les bases de dades, que si els coneguéssim abans de començar a implementar-la en condicionarien directament el disseny.

3.2 Domini del Problema

En general, tots els models de dades tenen per finalitat il·lustrar els conceptes bàsics del domini de l'aplicació, així com les relacions que hi pugui haver entre ells a nivell de referències que es puguin produir. D'entre tots els diagrames que poden incluir-se en l'especificació tècnica d'una aplicació el més abstracte és el *model de domini*. És un diagrama en el que apareixen les nocions essencials que el projecte ha de gestionar. És bàsicament introductori. Introdueix el projecte de manera vertical, com l'índex d'un llibre. o el mapa d'un territori.

En la Figura 3.4 es mostra un exemple pel model de domini pel club d'esports.

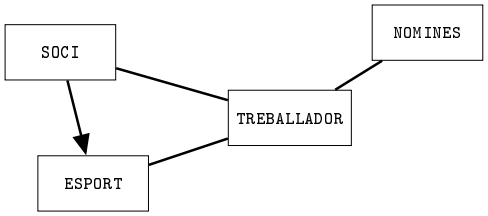


Figura 3.4: *Exemple de model de domini pel club d'esports.*

Noteu que tan sols apareixen els conceptes fonamentals que han de ser gestionats per la base de dades que finalment el pogués implementar. L'arc de `SOCI` a `ESPORT` podria representar que un soci està inscrit en un esport. Aquestes coses que no queden clares s'han de documentar amb els diagrames. És necessari acostumar-se a pensar que un diagrama va sempre amb una documentació associada.

Tot i que els models de domini són uns diagrames utilitzats en altres disciplines, com ara l'enginyeria de projectes, encaixen perfectament en el desenvolupament de bases de dades. Es tracta del punt d'inici a partir del qual més endavant construirem el disseny.

Un cop assumit un concepte pel domini, llavors utilitzem el verb ser quan enumerem els seus atributs. Diem, per exemple, una soci *és* un nom, un cognom, una data d'alta i una ciutat. I per tant, podrem respondre a preguntes com de quina ciutat és un soci.

3.3 Màximes de Qualitat d'un Disseny

És important que un dissenyador o administrador de bases de dades estigui familiaritzat amb la terminologia d'aquestes màximes.

3.3.1 Prohibir la Redundància

Prohibir la redundància vol dir que de cap manera s'ha de guardar una mateixa informació en llocs diferents, a no ser que sigui per motius rigorosament estructurals.

La conseqüència de no respectar aquesta màxima és donar peu a la inconsistència, i per tant la manca d'integritat. Que una base de dades tingui errors d'integritat és com dir que un barco sigui el titànic.

I atenció, no només cal evitar la redundància en les dades, sinó que també en les relacions entre elles. Això és important, i marca una diferència entre un bon dissenyador i un no tant bo. Llegiu amb atenció el paràgraf següent.

Si un concepte del domini, diguem-li C_1, es relaciona amb un altre objecte del domini, C_2, que està fragmentat en parts, o sigui que conté una col·lecció d'elements d'un altre concepte del domini, C_3, llavors a partir del moment que C_1 es relacioni amb C_3, en cas que calgui, treurem la relació entre C_1 i C_2 del disseny.

Per exemple, si C_1 representa un estudiant, C_2 representa un curs, o una assignatura, i C_3 les classes, o sigui les sessions del curs C_2, llavors en principi podríem pensar en relacionar un estudiant amb un curs quan es matricula. Però a partir del moment que volguem tenir alguna informació de l'estudiant respecte cada sessió, posem pel cas que voler saber si hi ha assistit o no, llavors ja no cal crear la relació de quan es matrícula, ja que podem actuar pensant que si un estudiant va a una classe d'una assignatura és que hi és matriculat. No és una llei en ferm. Pot ser que ens interessi els estudiants matriculats en un curs encara que no assisteixin a cap sessió. En qualsevol cas, inclús en el cas en que acceptem que pugui estar relacionat amb un curs, i a més a més amb cada una de les seves classes, cal ser conscients que ho fem per motius de correcció de possibles errors.

L'únic avantatge que ens proporciona la redundància és el fet de poder verificar la coherència que ha de portar associada. En definitiva, tan sols és útil pel control d'errors.

Reflexionant sobre l'impacte que té aquesta màxima en el disseny de bases de dades, es conclou que sempre que es pugui s'han d'evitar els cicles en el graf del model. I en qualsevol cas, cal analitzar molt bé els cicles en un model ER. En definitiva, els models ER en forma d'arbre preserven la unicitat dels conceptes evitant les redundàncies, cosa que els converteix en models fortament recomanables.

3.3.2 Prohibir Nuls Estructurals

No és correcte que sistemàticament algun atribut d'alguna entitat valgui nul. Per exemple, si ens guardéssim per cada persona quin esport fa, i la quantitat de persones que no fan cap esport fos significativa, llavors utilitzar un atribut deixant-lo nul quan la persona no fa esport seria incorrecte. Per això és interessant, en l'anàlisi, tenir tota la informació possible respecte les qüestions estadístiques.

3.3.3 Unificar Conceptes

Sempre que en una base de dades tinguem algun tipus d'objecte que representi un conjunt de dades lògicament relacionades caldrà fer-li referència quan en alguna altra part de la base de dades s'utilitzi el concepte. Per exemple, imaginem que en una base de dades tenim una taula amb informacions de cotxes, com pot ser el fabricant, el model, la cil·lindrada, etcètera. A més també hi tenim persones i cada persona pot tenir només un cotxe. llavors, agregar una dada a la persona amb el model del seu cotxe, per exemple, és incorrecta. Cal adonar-se'n que pel mateix preu (o sigui l'espai que ocupa aquesta dada) podem tenir guardat per cada persona algun identificador del cotxe que té, i per tant, no només tenir el model del cotxe, sinó tota la resta d'informació.

Aquest error és greu. I molt fàcil de detectar perquè en el model ER es pot veure que un atribut d'una entitat s'anomena igual que una altra entitat. I això és inadmissible.

En aquesta part, de disseny, s'ha fet una introducció al que en el proper capítol s'entrarà en profunditat. S'ha establert criteris de qualitat dels dissenys, i en definitiva, s'ha esmentat allò que formant part del disseny de bases de dades no té un impacte directe en el model entitat relació.

Capítol 4

Model Entitat Relació

Els diagrames o models entitat relació són segurament els models de dades més extesos. Són grafs dirigits en el sentit que s'explica en la introducció del Capítol 3. Les entitats són els nodes i les relacions els arcs.

Els dos termes que li donen nom són termes que han de fer reflexionar.

Una entitat en el model ER representa un conjunt d'elements dels que ens interessa mantenir una mateixa agrupació de dades. No és senzill, estem parlant d'un conjunt d'agrupacions. Cada element d'una entitat, doncs, és una agrupació de dades heterogènies en el sentit que tenen diferents semàntiques, i en conseqüència poden ser de diferents tipus.

És una definició molt semblant a les antigues estructures en llenguatges de programació d'alt nivell com fortran, pascal, o c. I una definició que es diferencia de l'actual concepte de classe en java en la carència de mètodes procedimentals. Representa el mateix una classe de java sense cap mètode que una antiga estructura en llenguatge fortran, per exemple, o que una entitat en el disseny d'una base de dades.

Una relació és un predicat, tal com s'ha descrit en la Secció 1.3, sobre una parella d'entitats, o més.

O sigui, la construcció d'un model ER és com un la d'un graf, primer són les entitats i després les relacions.

Quan un model es considera satisfactori, és a dir, es preveu com es podrà resoldre qualsevol demanda que l'usuari hagi formulat en la definició de requeriments, llavors es passa a la implementació, per la qual cosa caldrà traduir el model ER a un model relacional, procediment que es veu en detall en la Secció 5.3.4.

Els destinataris dels models ER som els humans. Els models ER els fem els humans, i per als humans. Tenen una finalitat que en última instància és documental. El model ER ha de formar part de la documentació tècnica de qualsevol base de dades.

Per altra banda, el fet que els destinataris siguem els mateixos humans, fa que no ens hagi d'extranyar que hi hagi part del que s'expressa en aquests models que després no tingui traducció al model relacional, és a dir a la implementació. En un disseny es pretén expressar les idees sense preocupar-se de la implementació.

Els diagrames ER tenen el seu propi llenguatge. De fet, són un llenguatge. Per això, disposen d'un compendi de símbols gràfics cadascún dels quals va associat a una semàntica de cara allò que modelen, i al mateix temps a una funció que de vegades és merament descriptiva, i altres estructural. Amb aquesta col·lecció de símbols s'estableix llavors una gramàtica amb la que s'expressa la planificació dels espais que finalment allotjaran les dades.

Els elements que constitueixen un model ER es presenten tot seguit.

4.1 Entitats

Les entitats representen el concepte més bàsic del disseny. Cada entitat té un nom, que indica quin tipus d'elements la composen. És a dir, una entitat representa un conjunt d'elements, tots ells caracteritzats amb diferents valors d'uns mateixos tipus de dades.

Els elements d'una entitat comparteixen els seus atributs, no els valors en cada un d'ells. Cal entendre doncs, que una entitat és una agrupació de dades que ens interessen, que van lligades entre elles, i no depenen de cap altra consideració.

La seva representació gràfica és un rectangle, com es mostra en la Figura 4.1.

Figura 4.1: *Representació d'una entitat en un model ER.*

Totes les bases de dades parteixen d'un disseny on hi ha una o més entitats.

Probablement l'entitat que apareix a més bases de dades és l'entitat `PERSONA`. En la major part de les bases que ens envolten hi és present una entitat que guarda dades relatives a persones.

Així doncs, un exemple d'ús de la representació de la Figura 4.1 podria ser la que es mostra en el Model 4.1.

Model 4.1. *Exemple d'una entitat en un model* ER.

S'entén que el Model 4.1 representa una conjunt de persones.

Les pautes d'estil que s'utilitzen en aquest llibre es troben resumides en l'Apèndix A. Pel que fa als noms o identificadors per designar entitats en els models ER utilitzarem substantius en singular. És a dir, una sola paraula, que escriurem en majúscules. Excepcionalment si ens calen dues paraules per designar una entitat, podem usar guions_baixos. Fer cas d'aquestes directrius ens ajudarà a interpretar els models. Al llegir una entitat en un diagrama utilitzem l'article indeterminat. El cas del Model 4.1 representa una persona, que pot ser qualsevol. Aquesta és la interpretació correcta.

4.2 Atributs

Cadascuna de les dades que convingui agregar a una entitat serà un atribut, que també podem anomenar *camp*.

Els atributs són les partícules elementals de les bases de dades, en el sentit que és la unitat mínima de transferència, tant de lectura com d'escriptura.

En principi, és raonable pensar que dues entitats amb els mateixos atributs haurien de ser considerades una mateixa entitat, encara que aquest és un aspecte que es veurà més endavant. Una entitat és una agrupació lògica d'atributs, entenent lògica com natural, dictada pel sentit comú.

No és correcte dir que un atribut pertany a una entitat. És correcte dir que un atribut estructura una entitat, ja que els elements d'una entitat no són els atributs. Una entitat és un conjunt d'elements que comparteixen uns mateixos atributs.

Podem suposar que si no fos pels atributs una entitat no tindria raó de ser. Més endavant ens trobarem altres factors que conduiran a l'existència d'entitats sense atributs, però de moment, resulta útil entendre que els atributs justifiquen l'existència de les entitats.

En el diagrama s'expressen amb una el.lipse que s'uneix a l'entitat que estructuren, tal com es pot veure a la Figura 4.2.

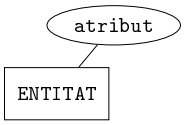


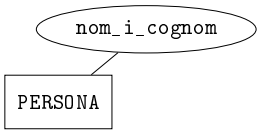
Figura 4.2: *Representació d'un atribut en una entitat d'un model* ER.

En general els noms dels atributs seran sustantius o, pel cas d'atributs booleans, sovint ens trobarem amb adjectius. Si volem anomenar un atribut amb més d'una paraula utilitzarem guions baixos. Escriurem els noms dels atributs en minúscules.

Un exercici freqüent en el disseny de bases de dades és trobar un terme que englobi un conjunt de possibilitats, o sigui l'abstracció. Freqüent i molts cops difícil. Això està relacionat amb la capacitat de considerar el cas que tenim al davant com un cas particular d'un conjunt més ampli. A una variable que pugui valer *moto*, *cotxe*, o *camió*, no cal tenir massa imaginació per anomenar-la *vehicle*. Però en altres casos això no és tan senzill. Massa sovint hi ha atributs que es diuen *tipus*, cosa dolenta, perquè no aporta cap contingut semàntic i es presta a confusió. Sempre que un atribut s'anomeni tipus entendrem que es refereix al tipus del que sigui el nom de l'entitat. És a dir, si una entitat es diu E, i té un atribut que es diu tipus, llavors mentalment a l'hora de llegir direm "tipus d'E". I acceptarem aquest nom per l'atribut si això reflecteix la seva semántica en el domini on ens trobem. En qualsevol cas, els atributs anomenats tipus cal documentar-los per regla general.

Tot seguit, recuperant l'exemple del Model 4.1, ens plantejarem un dilema que ens ajudarà a establir l'existència d'atributs junts, o separats.

Si pel cas volguéssim guardar el nom i el primer cognom de les persones, una aproximació inicial en l'exemple del Model 4.1 seria afegint un atribut `nom_i_cognom`, com en el Model 4.2.



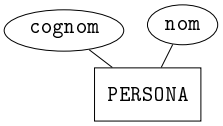
Model 4.2. *Exemple de l'ús d'atributs en l'entitat del Model 4.1.*

Fent-ho així n'hi hauria prou per poder guardar el nom i el cognom de les persones. Requeriment satisfet.

Això no obstant, el Model 4.2 té greus limitacions. Si es demana un llistat dels que es diguin *Garcia* de cognom, costaria obtenir-lo. En general, no ho tindríem fàcil per cap tractament que es basés en el cognom. Ni tan sols obtenir les

persones ordenades per cognom. La raó és senzilla. Els valors dels atributs són unitats de comparació, i per tant, els tractaments que necessitin exclusivament una part del valor ralentitzaran la resposta, perdent eficiència.

Molt probablement resulti millor la solució de guardar el nom i el cognom com atributs independents. És el cas del Model 4.3.



Model 4.3. *Exemple més acurat de l'ús d'atributs en l'entitat del Model 4.1.*

L'avantatge que té el Model 4.3 respecte el 4.2 és que resultarà més senzill obtenir una llista de les persones ordenades per cognom, per exemple. En general, el Model 4.3 ofereix més possibilitats operatives a partir dels cognoms. No només ordenar-los, sinó qualsevol altre tractament. Per la major part de les aplicacions, el Model 4.3 és millor que el 4.2.

En conclusió, el Model 4.2 és més senzill, però no tant potent. Tot i així pot ser útil, i molts cops basta. És més. Moltes bases de dades es guarden les dades fragmentades de maneres injustificables, com separar el bloc, l'escala, el pis i la porta d'una adreça postal. Això no cal, i complica el model. Com a molt, una adreça ha de discernir entre el carrer i el número. No perquè ens interessi el número (seria molt extrany una consulta a la base de dades que afectés a totes les persones que visquin en el número 30 del seu carrer). No és això. El que efectivament sí que ens pot interessar és el nom del carrer, ja que pot ser que volguem enviar alguna cosa a tothom que visqui en un carrer concret, per exemple.

Per altra banda, a l'hora d'avaluar si val la pena la introducció d'un nou atribut, un criteri de qualitat útil és: Si no som capaços de trobar valors d'exemple d'un atribut en menys d'un segon vol dir que l'atribut no té sentit. Els atributs d'una entitat són anàlegs a les variables d'un llenguatge de programació. No respectar aquesta regla és un error que cometen molts aprenents.

Per exemple, si tinguéssim una entitat que es digués `VIVENDA` que pogués ser un *pis*, una *casa* o una *masia*, llavors en cap cas hauríem de posar tres atributs que es diguessin `pis, casa,` i `masia`. Imagineu-vos-ho. Quins valors podrien prendre? Si penseu en variables booleanes, malament, per la segona màxima de la Secció 3.3, nuls estructurals. Com a molt, un atribut que es digués `tipus`, i que pogués prendre aquests tres possibles valors. Efectivament es referiria al tipus de vivenda, que és el nom de l'entitat. De totes maneres, si el valor d'aquest atribut condicionés l'existència d'altres dades addicionals, llavors hauríem de pensar en una especialització. Aquest concepte és tracta a la Seccio 4.14.1.

4.3 Atributs Clau

Totes les entitats han de tenir un conjunt d'atributs clau, encara que gairebé sempre amb un sol atribut n'hi ha prou.

Tenint en compte que una entitat representa un conjunt d'elements, i que els elements d'un conjunt han de ser diferents, per cada entitat d'un model ER haurem de garantitzar que tots els elements siguin diferents. Això ho fem establint una clau. Una clau d'una entitat és un conjunt d'atributs que per definició tindrà una combinació de valors diferents en cada element de l'entitat.

O sigui, si la clau consta d'un sol atribut, llavors els valors d'aquest atribut han de ser diferents per cada element de l'entitat, és a dir, no es poden repetir. I en canvi, si la clau consta de més d'un atribut, llavors els valors de qualsevol dels atributs que formen la clau es poden repetir sense cap problema. El que no es pot repetir en aquest segon cas és la combinació de valors dels atributs que formen la clau. La combinació.

Amb aquesta definició assegurem que cada element d'una entitat sigui diferent dels altres, i per tant, que l'entitat mateixa segueixi sent un conjunt i no un multiconjunt quan es produeixin noves insercions. Com que els valors de la clau d'una entitat són diferents per cada element, la clau serveix per identificar l'element dins l'entitat. Això marca la diferència entre els atributs clau i els altres, no és el fet que siguin atributs amb valors no repetits, és el fet que els usem per identificar.

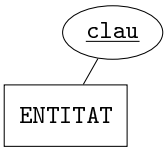


Figura 4.3: *Representació d'un atribut clau en una entitat d'un model ER.*

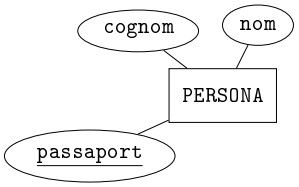
Els atributs que formen part de la clau es representen subratllats en un model ER, Figura 4.3.

En endavant anomenarem atributs *estructurals* als que els valors que puguin tenir vinguin condicionats per altres continguts de la base de dades, com és el cas dels atributs clau, ja que no poden tenir un valor que ja existeixi en la mateixa entitat, i per tant el valor de la clau ve condicionat per algun contingut de la base de dades. Als altres atributs els anomenarem *descriptius*. Aquesta classificació dels atributs en estructurals i descriptius es veu en més detall a la Secció 5.3.

Bé, amb el Model 4.3 de la pàgina 53 en tindríem prou per fer una base de dades que guardés el nom i el cognom d'un conjunt de persones. De tota manera hauríem de tractar el problema de com distingir diferents persones que

tinguessin el mateix nom i cognom. Una possible solució seria establir el nom com atribut clau, o el cognom, o la combinació dels dos valors. Però és clar que si poséssim el nom com a clau, no admetríem persones amb el mateix nom, i amb el cognom o amb la combinació d'ambdós tampoc admetríem les repeticions corresponents. El que ha de fer un dissenyador de bases de dades és observar la realitat, i ser capaç d'adonar-se'n de quin criteri s'utilitza en el món real per distingir les persones d'aquesta base de dades.

Ara, com atribut clau, hi afegim el número de passaport, ja que, per una banda, sembla la manera més natural d'assegurar-nos que tots seran diferents i per una altra, perquè els usuaris de la base de dades consideraran natural aquesta manera d'identificar les persones. I per tant, no s'extranyaran quan, si es donés el cas, no permetéssim donar d'alta un número de passaport que ja existeix. Assumiran l'error, i miraran d'arreglar-lo com sigui. Una de les característiques que ha de tenir una bona base de dades és fer comprendre els errors que es puguin produir a l'usuari final. I allò més desitjable, és que aquests usuaris assumeixin els errors com a seus.



Model 4.4. *Representació d'una entitat completa en un model ER.*

4.4 Atributs Multivalorats

De vegades interessa poder guardar més d'un valor sota un mateix concepte d'atribut. En altres paraules, per cada element d'una entitat, enlloc de tenir tan sols un valor de cada atribut, per algun atribut concret en tenim varis. Aquesta és la idea del que representa un atribut multivalorat. El símbol que s'utilitza en un model ER per expressar que un atribut és multivalorat és traçant l'el·lipse en doble línia, tal com hi ha en la Figura 4.4.

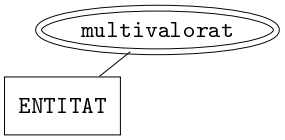
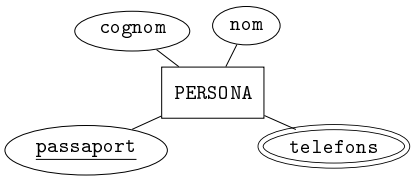


Figura 4.4: *Representació d'un atribut multivalorat en una entitat d'un model ER.*

Bé, en l'últim paràgraf de la introducció del Capítol 3, precisament quan es defineixen els nodes terminals, es diu que segons si són fonts o pous fan funcions molt diferents. Els atributs multivalorats són nodes terminals font en el graf dirigit d'un model ER. Així doncs alerta, aquí es presenta el cas en que són fonts. O sigui, la fletxa surt d'ells.

Aquesta excepció que fem per poder interpretar que un atribut es correspon en un node del graf és la causa de que en la representació de la Figura 4.4 no hi hagi cap fletxa. De totes maneres, pels atributs multivalorats, convé imaginar-se-la.

Per l'aplicació del club, diuen els requeriments de la Secció 2.6.1 que cal admetre varis telèfons per persona. Llavors, un model com el 4.5 podria valer.



Model 4.5. *Exemple d'atribut multivalorat en el Model 4.4.*

Noteu que no fem cap distinció entre els telèfons associats a una mateixa persona. El que expressem precisament és que no volem guardar cap informació addicional amb cada número de telèfon d'una persona. Si es volgués guardar la categoria, feina, particular, etc... per exemple, llavors no estaríem parlant d'un atribut multivalorat. De fet, un atribut multivalorat es caracteritza per això, perquè no distingeix entre els valors associats a un mateix element de l'entitat.

Un atribut multivalorat associa col·leccions a elements d'entitats, no els hi associa seqüències. Aquestes col·leccions d'elements acostumen a ser disjuntes. El Model 4.5 significa que cada persona pot tenir molts telèfons, però per regla general un telèfon només pertany a una persona. Això és així perquè amb atributs multivalorats no tenim cap control de les repeticions. Per tant, si algun dia un número compartit fos modificat per les raons que sigui, al ser un atribut multivalorat es requeriria actualitzar les aparicions d'una en una. Si la quantitat de números de telèfon compartits entre varies persones fos gran, per exemple de més del cinquanta per cent, llavors no convindria moderlar-lo amb un atribut multivalorat. Convindria fer-ho d'alguna manera que l'SGBD pogués tenir constància que aquell número és el mateix per varies persones.

Pels noms dels atributs multivalorats utilitzarem el plural, igual que fem pels atributs que són números enters. La diferència resultarà evident en qualsevol cas per la doble el·lipse. És fàcil comprendre que pels atributs enters estem dient quants, i pels multivalorats, quins.

4.5 Atributs Compostos

En molts casos pot interessar tractar els atributs de forma jeràrquica. Això vol dir poder disposar dels valors d'un sol atribut concebut com un tot, encara que realment estigui fet de parts, de les que també ens interessa poder-ne disposar. Els atributs compostos són els que contenen atributs més petits.

Per entendre'ns, tractar o disposar d'un atribut significa poder filtrar tots els elements de l'entitat segons el valor d'aquell atribut, per exemple, o també poder ordenar segons aquell camp.

En la Figura 4.5 es pot veure la forma que adopta un atribut compost en el diagrama.

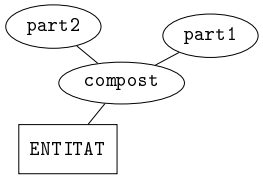


Figura 4.5: *Representació d'un atribut compost en una entitat d'un model* ER.

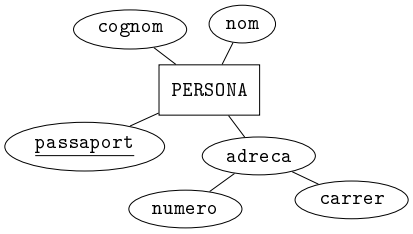
Els atributs més petits de tots són els atributs atòmics. Un atribut és *atòmic* si es correspon amb un tipus de dades primitiu, com són les cadenes de caràcters, enters, booleans, o números amb decimals. O sigui, que de la mateixa manera que quan parlem de subconjunt ens imaginem un subconjunt propi, quan parlem d'un atribut ens imaginem un atribut atòmic. Els normals són els atòmics. El nom, el cognom,...

Anticipem aquí que no tindrem manera per expressar la idea d'atribut compost en la implementació. I malgrat tot, sí que existeix en el disseny. Potser cal recordar que com s'ha dit en la introducció d'aquest capítol, el model ER ha de ser un document fet pels humans i per als humans, o sigui que quan dissenyem procurem abstraure'ns de com s'haurà d'implementar el que estem planificant, encara que honestament, quan es dominen els conceptes de la implementació es produeix una interacció incontrolable que els dissenyadors difícilment podem controlar. Dissenyem pensant en la implementació... però bé, tampoc està clar que això representi un inconvenient.

En qualsevol cas, els atributs compostos són un concepte. I si cal implementar un comportament que reflecteixi fidelment aquesta idea sempre queda el recurs d'incorporar procediments i vistes per poder veure l'entitat segons els diferents nivells dels atributs compostos que l'integrin.

Posem per cas que interessa poder tractar les adreces de les persones, però també es necessita fer tractaments segons els carrers, per qüestions de distribució logística, o per saber a quin carrer convé posar una valla publicitària...

El Model 4.6 respon aquest requeriment.



Model 4.6. *Exemple d'atribut compost en el Model 4.4.*

El número de l'adreça és un atribut residual. No és que interessi poder fer operacions, com ara comparacions o seleccions amb els valors d'aquest atribut. És que com que efectivament sí que cal fer-ho amb els altres camps, aquest ens queda com a residu.

4.6 Atributs Calculats

Els atributs calculats, o *derivats*, són atributs els valors dels quals per cada element de l'entitat són autogestionats. No són dades que hagi d'introduir l'usuari, sinó valors que l'SGBD pot obtenir a partir d'altres valors presents al seu abast.

Normalment serveixen per mantenir informacions estadístiques.

Els atributs calculats tenen una importància emergent. En l'actualitat de l'any 2014 està augmentant espectacularment la quantitat de recursos que es dediquen a la mineria de dades, tema del qual en [9, 4] se'n parla en profunditat. Clar, d'alguna manera es pot entendre que les bases de dades "saben" coses que nosaltres els humans no sabem.

Imagineu una base de dades d'una perruqueria de gossos en la que resulta que tots els caniches no tornen després de la primera visita. Aquesta és una informació rellevant per l'amo de la perruqueria, i de fet està continguda dins la base de dades, però en canvi no tenim eines per fer-la aflorar.

O un altre exemple. L'exemple típic que es posa quan s'explica el que és la mineria de dades. Una cadena de supermercats nordamericana va descobrir que els divendres es venien quantitats excepcionals de cerveses i de bolquers per criatures. Llavors va posar de costat ambdós productes i les vendes es van disparar. Tot plegat fa pensar que els causants eren pares separats que els hi tocava cuidar-la el cap de setmana.

Els atributs derivats són l'origen de la mineria de dades, és a dir, de l'explotació dels continguts de les bases de dades.

Es representen amb línia discontínua tal com s'exposa en la Figura 4.6.

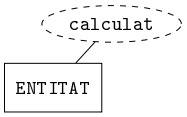


Figura 4.6: *Representació d'un atribut calculat en una entitat d'un model* ER.

Noteu que en aquest moment no se'ns passa pel cap parar-nos a preguntar qui ni quan es calcularan els valors d'aquests atributs. Tan sols establim la intenció de que algú ho farà en algun moment. Estem en l'etapa de disseny.

A l'hora de definir un atribut com a calculat s'està fent una gestió molt meticulosa del temps. Per això convé tenir en compte quatre consideracions que ens ajudaran en la decisió.

- L'eficiència del procediment de càlcul és fonamental. Pel cas que calcular el valor d'un atribut derivat es pugui utilitzar un algorisme d'eficiència $O(1)$, o sigui que trigui com una assignació, una suma o una resta, llavors no val la pena malaguanyar l'espai per guardar-lo. Simplement es pot calcular cada cop que es requereixi el seu valor. Aquesta feina la pot fer el mateix programa client, o es pot encapsular en el SGBD.

- Partim doncs de que l'eficiència de l'algorisme és com a mínim $O(n)$, sent n el nombre d'elements a la base de dades. És a dir, suposem que per actualitzar el valor d'un atribut derivat cal recórrer completament els elements d'alguna entitat. Llavors, la nostra administració del temps és el tema crític. Què preferim?

  - Que el SGBD trigui molta estona quan ens interessi obtenir el valor de l'atribut, o bé
  - Que cada cop que es modifiqui alguna dada que impacti en el valor de l'atribut es trigui una miqueta gairebé inapreciable més de temps en actualitzar el valor derivat.

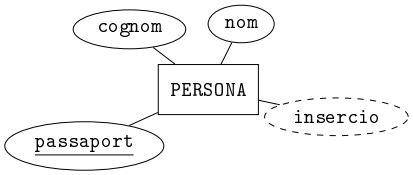
  La decisió ve condicionada per l'estadística. Si hem de consultar el valor molt de tant en tant triarem la primera opció, i per tant no ens caldrà tenir cap atribut derivat. En canvi si el volem tenir disponible molt sovint, llavors la segona, que sí que requereix un atribut calculat.

- Un atribut derivat està absolutament justificat quan sense ell no podríem obtenir alguna informació. Un cas molt freqüent és el de guardar les dates d'inserció dels elements en les entitats.

- Es diu que una entitat d'una base de dades té un alt índex de volatilitat si té moltes altes i baixes d'elements. Un altre cas en el que un atribut derivat queda totalment justificat és quan depèn de valors que varien tota l'estona. Convé tenir atributs derivats si a l'hora de calcular-los els operands resideixen en entitats d'aquesta mena.

Dins la disciplina de les bases de dades, a més dels tipus primitius de qualsevol llenguatge de programació també hi ha el tipus *data*. Això es veurà en profunditat en la Secció 6.2.4. Com que per norma d'estil no posem els noms dels tipus en els atributs, en aquest l'anomenem simplement amb un mot que indiqui què significa la data.

Aquest és el cas de l'exemple del Model 4.7.



Model 4.7. *Exemple d'atribut calculat en el Model 4.4.*

Un exemple no tant típic seria recuperant el Model 4.6 on hi apareixia l'adreça de les persones, i afegint-hi un camp enter calculat que comptabilitzés els canvis d'adreça que ha efectuat cada persona des que va ingressar en la base de dades. Noteu que si no fos per l'existència d'un atribut calculat, no podríem saber-ho de cap manera.

4.7 Relacions

Les relacions en un model ER lliguen dues o més entitats.

Tota l'estructuració de les bases de dades prové de les relacions entre entitats. Per això són tan importants.

El grau d'una relació és el nombre d'entitats que relaciona, cosa que coincideix amb la definició de grau d'un node en un graf. Anomenem relació *binària* la de grau dos, o sigui que relaciona dues entitats. Si en relaciona tres, relació *ternària*. I si en relaciona més de tres, tot i que podríem dir quaternària o el que fes falta, més aviat podríem assegurar que la base de dades està mal dissenyada. No és una llei en ferm, però sí que molt habitual. És excepcional trobar una relació amb més de tres entitats en un model ER. I quan hi és, fàcilment és incorrecta.

Al llarg d'aquesta secció suposem que tenim una relació de nom R, que relaciona les entitats E_1 i E_2.

A l'hora que s'estableix una relació entre dues entitats és convenient pensar bé en quin tipus de participació ha de tenir l'entitat a la relació. La participació d'una entitat en una relació pot ser parcial o total. Que una entitat E_1 tingui *participació parcial* en una relació R significa que no tots els elements d'E_1 estan relacionats amb algun element d'E_2 necessàriament. En canvi, *participació total* vol dir que tots els elements d'E_1 han d'estar relacionats amb algun element d'E_2 forçosament.

La participació d'una entitat en una relació és una qüestió de tercer ordre en el moment del disseny. Del que es preocupa qui dissenya és d'establir les relacions com a tema primordial. Després analitzar cardinalitats, que veurem tot seguit. I en tercera posició, per ordre d'importància, s'analitzen i s'estableixen les participacions... si es fa. Això condueix a que en els models ER se li doni molt poca importància al tema de les participacions, i en molts casos no s'expressa. És considera una qüestió menor. I és una llàstima, perquè la informació que aporten és força útil.

La participació parcial d'una entitat en una relació s'expressa amb línia simple en el diagrama. La total amb línia doble. En la Figura 4.7 la línia simple d'E_1 a R significa que pot haver-hi elements d'E_1 que no estiguin relacionats amb cap element d'E_2. En canvi, la línia doble d'E_2 a R vol dir que tots els elements d'E_2 estan relacionats amb algun element d'E_1.

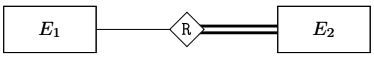


Figura 4.7: *Participació parcial d'E_1 i total d'E_2 en la relació R.*

Per entendre'ns, si E_1 fos PAIS, E_2 fos PERSONA, i R fos ES, el model de la Figura 4.7 permetria afegir països encara que no hi hagués cap persona d'aquell país. En canvi, no s'hi podria afegir una persona si no se sapigués de quin país és.

La doble línia, que en un model ER s'utilitza per indicar que un atribut és multivalorat, també es fa servir per indicar participació total d'una entitat en un atribut. Tot i així hi ha una lleugera diferència. En un atribut multivalorat la doble línia vol dir "molts". En una participació total vol dir "tots".

4.7.1 Dependències d'existència

Quan una entitat té participació total en una relació es produeix una *dependència d'existència*. És a dir, l'entitat no pot contenir cap element mentre no existeixi el corresponent element relacionat en l'altra entitat de la relació. Això

vol dir que tan sols en casos patològics pot succeir que les dues entitats d'una relació tinguin participació total en la relació. Penseu-hi. Sembla impossible. Si tant E_1 com E_2 tenen participació total en R, llavors no pot existir cap element a E_1, perquè per poder existir hauria de tenir un element lligat amb ell per la relació. Però com que E_2 tampoc podria tenir cap element mentre no existís el corresponent en E_1, ens trobem davant d'un peix que es mossega la cua.

En els SGBDs hi ha mecanismes per poder trencar aquest tipus de bucles. Són mecanismes senzills que es limiten inhibir la verificació de restriccions fins a nova ordre. És a dir, inhibiríem el control, ompliríem les dades, i restauraríem la verificació de restriccions altre cop. De totes maneres, això són casos excepcionals que no succeeixen sovint.

Com a conseqüència, acostumem entendre que quan una entitat participa totalment en una relació, l'altra no.

Quan una entitat E_2 participa totalment en una relació R vol dir que no acceptem elements a E_2 si no estan relacionats amb almenys un element d'E_1, i en aquest cas diem que E_2 té una dependència d'existència d'E_1. Els SGBDs també tenen mecanismes per dir que quan s'esborri, o es canvïi, un element d'E_1 s'esborrin, o s'actualitzin, automàticament tots els elements d'E_2 relacionats amb ell.

4.7.2 Cardinalitat d'una relació binària

La *cardinalitat* d'una relació binària en un model ER es defineix com el nombre d'elements d'una entitat que poden estar associats a un cert element de l'altra entitat de la relació. El terme cardinalitat és nou. En la Secció 1.1 havíem vist el cardinal d'un conjunt. Ara parlem de la cardinalitat d'una relació. Són coses diferents. La noció de cardinalitat reflecteix relacions de pertinença d'elements d'una entitat en elements de l'altra.

Hi ha tres tipus de cardinalitats per relacions binàries, 1:1, 1:N, i M:N, i es pronuncien simplement amb els mots "u u", "u ena", i "ema ena". Tant important és la cardinalitat d'una relació que direm directament relació 1:1, o 1:N, o M:N, enlloc de dir relació de cardinalitat 1:1, o 1:N, o M:N.

La transcendència dels noms de les relacions en un model ER depèn de la seva cardinalitat. Totes aquelles paraules que en la transformació del model ER al model relacional hagin de desaparèixer ens les estalviarem, tal com s'indica en les normes d'estil de l'Apèndix A. En concret, la majoria dels noms de relacions 1:1 o 1:N desapareixeran.

En les tres properes seccions s'explica amb més detall cada una de les tres cardinalitats.

4.8 Relacions 1:1

Les relacions binàries amb cardinalitat 1:1 permeten cada element d'E_1 associar-se amb un sol element d'E_2, o cap, i cada element d'E_2 associar-se amb un sol element d'E_1, o cap.

Es representen com es mostra en la Figura 4.8.

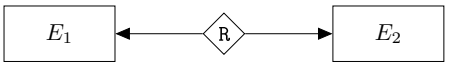


Figura 4.8: *Relació 1:1 en un model ER.*

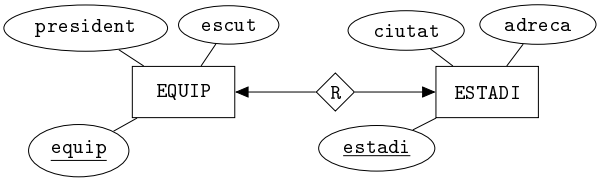
Les fletxes sempre surten de la relació cap enfora, i signifiquen "només un, o cap". Per forçar que vulgui dir "un", cal una participació total, posant línia doble enlloc de línia simple.

Es pot evitar dibuixar el rombe, i el nom de la relació. Llavors, segons E_1 la relació s'anomenaria "E_2", i viceversa. Per tant, no s'introduiria cap ambigüitat. Tot i així, això no obstant, les relacions 1:1 són excepcionals. La seva raresa és fruit de que gairebé sempre són supèrflues i poden suprimir-se del diagrama agregant els atributs de les dues entitats en una de sola que pot anomenar-se amb qualsevol dels noms de les dues entitats agregades.

La justificació de que en un model ER hi aparegui una relació 1:1 passa per l'estadística. Té sentit fragmentar la informació que lògicament hauria d'anar agregada per un dels dos motius següents:

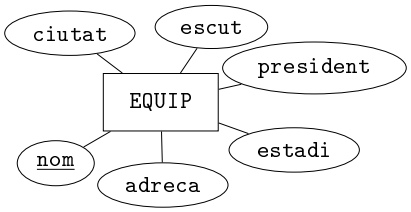
- Quan una part dels dos fragments tingui uns índexos d'activitat molt diferents als de l'altra. És a dir, quan s'hagin de produir moltes transferències de dades d'una de les dues entitats inicials.
- Quan les dues entitats són grans, és a dir tenen molts atributs, i l'associació no és indispensable, o sigui un dels dos uns és zero per molts elements d'algun dels dos costats.

En el Model 4.8 hi ha un exemple de relació entre equips de futbol i estadis.



Model 4.8. *Exemple de relació 1:1 en un model ER.*

Com s'ha dit, davant d'un model com el 4.8, el que hem de fer com a dissenyadors, és aplicar-li la simplificació que es mostra en el Model 4.9.



Model 4.9. *Simplificació del Model 4.8.*

Noteu que en el moment de fer aquesta agregació estem relaxant una restricció que en el Model 4.8 hi és present, i en canvi en la solució simplificada del Model 4.9 ha desaparegut. Es tracta de l'atribut clau de l'entitat ESTADI. En altres paraules, els noms de l'estadi de cada equip en el Model 4.8 no poden ser repetits, perquè són clau. I en canvi en el model simplificat 4.9, sí. Però bé, això no ens ha de preocupar, ja que posteriorment, en el moment de la implementació podrem imposar la unicitat als valors del nom de l'estadi.

Això serveix per entendre que el fet que en un model ER hi subratllem els atributs clau és per indicar que a partir d'aquells valors es podrà identificar els elements de l'entitat en qüestió. Això implica que tots els valors siguin diferents, però el propòsit de subratllar-los no és dir que han de ser diferents, sinó que els identificarem per mitjà d'aquests atributs.

I encara una altra diferència entre els models 4.8 i 4.9 prendria importància si existissin molts equips sense estadi. En aquest cas, també seria justificable el primer dels dos dissenys, ja que el segon passaria a tenir una quantitat excessiva de valors nuls estructurals, cosa que va en contra de la màxima segona de la Secció 3.3.

4.9 Relacions 1:N

Les relacions 1:N són la plasmació gràfica més clara del que significa una referència en una base de dades. Aquest tipus de cardinalitat acostuma a representar una relació de pertinença. Es diria que hi ha varis elements d'E_2 que pertanyen a un únic element d'E_1.

Les relacions 1:N permeten cada element d'E_1 associar-se amb varis elements d'E_2, però per cada element d'E_2 es permet associar amb un únic element d'E_1 o cap. Com en el cas 1:1, de vegades l'1 de la relació pot ser zero, cosa que podem impedir fent que la participació del costat N sigui total amb doble línia.

En un model ER, les relacions 1:N es representen com en la Figura 4.9.

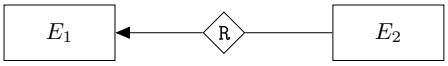


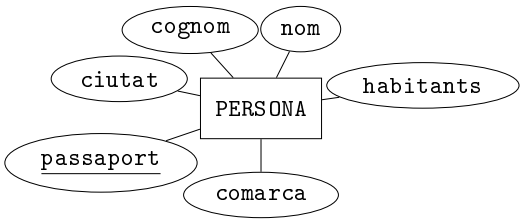
Figura 4.9: *Relació 1:N en un model ER.*

Que quedi clar, la fletxa apunta cap el que només pot ser un.

No distingim cap ordre en les entitats relacionades. És a dir, no definim les relacions N:1, perquè com es diu en la introducció d'aquest capítol, expressar un model en un diagrama serveix per no comprometre's en cap ordre entre les entitats. Els nodes d'un graf no tenen cap posició relativa preestablerta, i per tant, si ens trobem amb una relació N:1, girant el full de paper ja la tenim 1:N.

Tant el rombe com el nom de la relació es pot suprimir sense cap problema, sempre que només hi hagi una sola relació 1:N entre aquesta parella d'entitats. Al no dibuixar ni el símbol ni el nom de la relació s'entendria com que el nom de la relació és "E_1", que és l'entitat apuntada, o sigui la del costat 1 de la cardinalitat.

Ara hauria de venir aquí un exemple de relació 1:N. Aprofitem però l'avinentesa per il·lustrar una forma en la que es procedeix per decidir crear noves entitats. Per observar que l'establiment d'una nova entitat no sempre és fruit de l'anàlisi, sinó que de vegades és automàtica. Cal que imaginem que ens equivoquem a l'hora d'assignar atributs a una entitat, i veurem com anem a parar a la creació de la nova. Per fer-ho, recuperem l'exemple del Model 4.4 de la pàgina 55 on hi havia tan sols persones amb nom, cognom i el camp clau amb el passaport. Suposem que volem guardar més atributs per cada persona. Afegim la ciutat on viu, el seu número d'habitants, i la comarca. Ens quedaria un disseny semblant al del Model 4.10.



Model 4.10. *Representació d'una entitat amb atributs incorrectes en un model ER.*

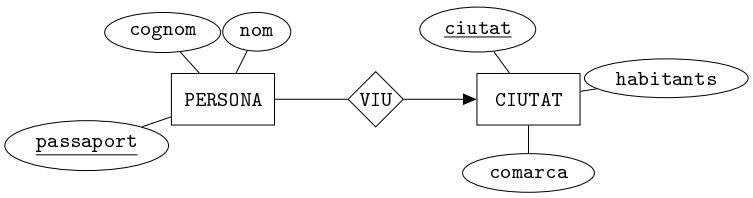
El plural de l'atribut `habitants` és per indicar que és una quantitat. Un nombre enter. El nom de l'atribut podria ser `numero_d_habitants` però per qüestions d'estil no posem el nom dels tipus dels atributs, per tant no hi posem "numero".

Noteu que afegint els atributs tal com s'indica en el Model 4.10, llavors per cada persona de *Lleida* que s'insereixi és repetiria tota la informació que només depèn de la ciutat. O sigui `comarca` valdria *Segrià*, i `habitants`, 139809. Això contradiu la primera màxima de la Secció 3.3, ja que estaríem introduint redundància.

Així doncs, el Model 4.10 és incorrecte. I l'explicació del per què, és que ens trobem davant del que es defineix com una *dependència funcional*. És a dir, que en una entitat el valor d'un atribut depengui del valor d'un altre atribut de la mateixa entitat. És un error greu de disseny, que cal evitar de totes totes.

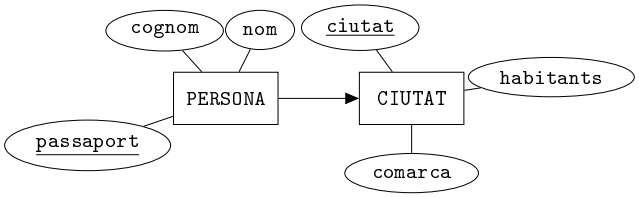
Precisament en aquest punt es posa en pràctica allò que es deia en la introducció del Capítol 2. Què depèn de què.

La forma correcta de procedir és agregant els atributs correlacionats en una nova entitat. Pel cas de l'exemple tindríem una primera proposta en el diagrama del Model 4.11.



Model 4.11. *Model correcte per l'exemple del Model 4.10.*

I tal com s'ha indicat més amunt, en aquest cas es pot suprimir el símbol i el nom de la relació. Llavors, enlloc d'anomenar-la `VIU` l'estaríem anomenant `CIUTAT`, sempre en el benentès que la ciutat d'una persona fa referència on viu, extrem que hauria de quedar documentat. El nou disseny és el del Model 4.12.

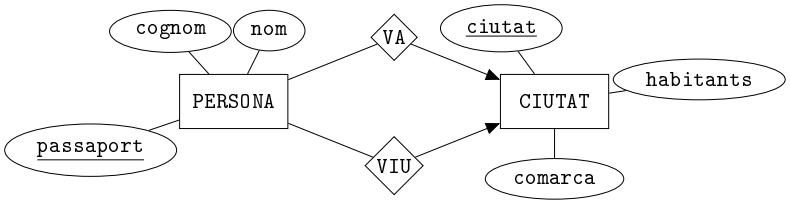


Model 4.12. *Model correcte simplificat per l'exemple del Model 4.10.*

Convé fer aquest tipus de simplificacions perquè ens recorden que una relació 1:N vista des del costat N és com un meta atribut, cosa que encaixa en la filosofia.

Però alerta. Aquesta simplificació és possible perquè no introdueix ambigüitat. Això és gràcies a que entre les relacions PERSONA i CIUTAT tan sols hi ha una relació en el model. En canvi, si hi haguessin dues relacions 1:N entre el mateix parell d'entitats, llavors sí que el nom seria necessari.

Suposem, per exemple, que en la mateixa base de dades de l'exemple es volgués guardar addicionalment la ciutat on va a treballar cada persona. En el Model 4.13 s'ha afegit al disseny del Model 4.11 la nova relació VA per satisfer aquest nou requeriment.



Model 4.13. *Exemple del Model 4.11 amb una relació nova.*

No podem prescindir dels noms de les relacions del Model 4.13, malgrat les seves cardinalitats siguin 1:N, ja que pel fet de lligar les mateixes entitats, introduiríem ambigüitat. Estaríem pretenent que les dues relacions es diguessin CIUTAT.

4.9.1 Entitats de suport a la interfície

La definició d'una entitat com *de suport a la interfície* està relacionat amb l'entrada i sortida de dades, és a dir amb la interfície i per tant amb el programa client. Per això no forma part essencial del disseny de bases de dades. De totes maneres s'introdueixen aquí perquè són un cas molt habitual de les relacions 1:N.

Les entitats de suport són fruit d'atributs descriptius, no estructurals, amb els que volem mantenir un control de possibles errors en el format dels valors. En altres paraules, són noves entitats nascudes a partir de voler uniformar els valors d'un atribut descriptiu. Per això és correcte dir que l'atribut *se segrega* en una entitat de suport.

Aquestes entitats satisfan dues condicions.

- Tan sols tenen un camp clau. No tenen atributs descriptius.
- Són nodes pou en el model ER, és a dir, nodes terminals que la fletxa apunta cap a ells.

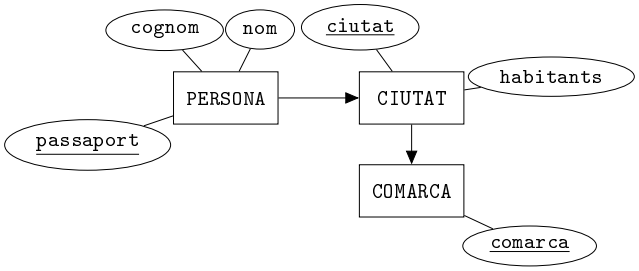
Al principi del Capítol 3, al final de la introducció, es defineixen els nodes terminals en un graf dirigit com és un model ER. I també es diu que els nodes del graf són les entitats del model. Es veu la diferència entre nodes terminals font, i nodes terminals pou i també s'anticipa que tenen funcions diferents. Després, en la Secció 4.4 es veu com els atributs multivalorats, tot i ser atributs i no entitats, fan el paper de nodes terminals font en un model ER. I ara aquí, es tanca l'explicació tot dient que les entitats de suport fan de nodes terminals pou.

Tornem a l'exemple del Model 4.12, de persones que viuen a ciutats. Com que el nom de la ciutats és clau, confiem que no es produiran errors de lletreig. Ara bé, la comarca no és clau, i és probable que sense adonar-s'en l'usuari de l'aplicació hagués introduït per error alguna ciutat amb la comarca *Segria* sense accent, enlloc de *Segrià* com ha de ser. Els errors de lletreig en la introducció de dades són molt perillosos perquè no hi ha cap nivell en l'arquitectura de l'aplicació que se'n responsabilitzi. És a dir, ningú ens avisarà mai de que existeix un error d'aquest tipus.

Això provocarà que el dia que es faci una consulta per saber quantes persones hi ha a la base que siguin de la comarca del *Segrià* se'ns estarà responent una quantitat equivocada. I això tampoc ho detectarà ningú. En definitiva, cal prendre mesures per evitar aquests errors.

Davant de casos així, les entitats de suport no és que els resolguin, però sí que serveixen per tenir un major control, i és molt habitual el seu ús.

En el Model 4.14 es pot veure l'exemple modificat havent-hi segregat l'entitat de suport COMARCA.



Model 4.14. *Entitat de suport* COMARCA.

Amb el disseny del Model 4.14 es té un millor control perquè existint aquesta entitat, és fàcil imaginar que la introducció dels valors de comarques es faci via menú desplegable. De manera que per l'usuari no només li resulti més còmode seleccionar el valor quan ja existeix, sinó que a més, s'ho hagi de pensar dues vegades a l'hora d'introduir un nou nom de comarca, ja que la primera vegada serà quan tingui el menú davant i vegi que no existeix la comarca que vol entrar.

Aquesta és una bona pràctica i es recomana utilitzar-la. De totes maneres, si això ens interfereix el model amb un excés d'entitats, pot quedar documentat de manera textual, tan sols informant que tal atribut se segregarà en una entitat de suport.

És normal que a part de les taules que s'infereixin a partir d'un model ER les bases de dades utilitzin moltes altres taules independents del disseny per moltes altres utilitats. Per exemple, per guardar opcions de configuració del programa client pot haver-hi una taula que guardi parelles `nom = valor`. A més, quan les opcions son personalitzables, llavors es comú guardar també una taula addicional d'usuaris. En fi, en aquest aspecte no ens hem de preocupar. No hi ha cap problema en crear taules auxiliars per fer càlculs i per la gestió de l'aplicació.

4.9.2 Entitats Febles

Per introduir el concepte d'entitat feble cal recuperar l'esperit analític i la vocació de fer de mirall de la realitat. Una entitat feble és rigorosament fruit de la forma d'identificar els elements de les entitats en la vida real.

Aquesta és precisament la característica més important que tenen les entitats febles. Per identificar els seus elements cal identificar prèviament l'element relacionat. Això es fa per mitjà d'una relació que rep l'adjectiu de relació *identificadora*. I l'entitat relacionada via relació identificadora es diu entitat *forta*, encara que de vegades també se li digui entitat identificadora a tot plegat, que de fet s'ha d'entendre com un concepte monolític.

Així doncs, una entitat feble ha de tenir forçosament una única entitat forta, que li proporciona part de la identificació dels seus elements per mitjà de la relació identificadora. Però com que és una entitat, cal poder identificar els seus elements completament, a diferència dels atributs multivalorats que per això mateix no són una entitat.

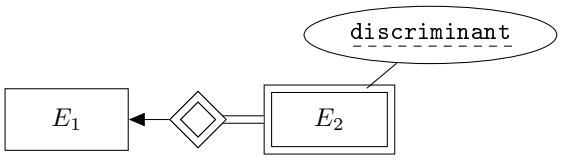


Figura 4.10: *Entitat feble E_2 identificada via E_1.*

Tot plegat fa que s'afegeixi un atribut que es diu *discriminant* a l'entitat feble, per distingir dintre dels elements relacionats amb un mateix element de l'entitat forta. En un model ER, el discriminant se subratlla en línia discontínua, cosa que té la seva raó de ser, ja que un discriminant es pot entendre com "mitja clau".

El fet de ser entitats habilita les entitats febles per poder relacionar els seus elements amb altres entitats a part de la identificadora, capacitat que no tenen els elements d'un atribut multivalorat.

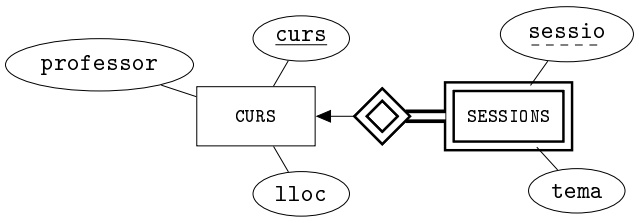
Per tot el que s'ha dit, es pot comprendre que una entitat feble és una extensió d'un atribut multivalorat. Li afegeix una distinció entre els valors. Per això el símbol gràfic utilitzat per representar una entitat feble en un model ER és el mateix que pels atributs multivalorats, com es mostra en la Figura 4.10.

Noteu que la relació identificadora d'una entitat feble ve indicada pel doble rombe que forma part del mateix símbol que expressa que una entitat és feble. I no cal que li donem nom. És important que sempre que es dibuixi un rectangle en doble línia, cosa que vol dir entitat feble, es dibuixi automàticament la doble línia que indica participació total, i el doble rombe que indica quina és l'entitat identificadora. Aquestes tres parts formen un sol símbol. No té sentit el doble rombe sense el doble rectangle, ni el doble rectangle sense el doble rombe. I precisament per això, perquè una entitat feble sempre ha de tenir una entitat forta identificadora, millor recordar tota l'estructura de doble línia com un símbol compacte.

A l'hora de decidir si en un model hi introduïm una entitat feble hi ha una regla d'or. I cal ser-li fidels. Si per identificar els elements de l'entitat en la vida real utilitzem la identificació d'una altra entitat, llavors convé introduir-la en el disseny com entitat feble. Per respondre's aquesta qüestió no cal ser ni analista ni dissenyador ni res. Cal simple observació.

Que quedi clar. Utilitzarem una entitat feble per disposar de les parts d'un tot sempre que per identificar cada una de les parts ens calgui la identificació del tot que les conté. Aquesta és una regla de disseny que distingeix un bon disseny d'un que no ho és.

L'exemple del Model 4.15 és d'un disseny amb una entitat que representa cursos, o assignatures, formats per un conjunt de sessions, o classes, que s'identifiquen numerant-les de l'1 en endavant.



Model 4.15. *Exemple d'ús d'una entitat feble.*

També es podria fer amb el dia, si en cap cas es poguessin fer dues classes

el mateix dia. Observant-lo amb esperit crític cal preguntar-se si realment per poder parlar d'una sessió d'un curs cal necessàriament dir de quin curs es tracta. La resposta sembla ser que efectivament sí. I per tant el model sembla ser correcte. L'atribut discriminant `sessio` ha de servir per distingir les sessions d'un mateix curs. Encara que costi de recordar, dins l'entitat SESSIONS hi ha barrejats sense cap ordre totes les sessions de tots els cursos. Una entitat feble és un conjunt de conjunts d'elements.

I per això, a l'hora d'identificar un element d'una entitat feble, en primer lloc cal identificar el conjunt, és a dir l'element de la identificadora, que el conté.

Altres exemples són els bitllets per un vol d'avió. L'entitat forta seria VOL, i la feble BITLLETS. Efectivament, no es pot identificar un bitllet d'avió sense dir de quin vol es tracta. O bé el típic exemple en el que l'entitat forta és FACTURA, i la feble LINIES, entenent que una línia d'una factura no pot ser identificada si no sabem de quina factura parlem.

Anomenarem les entitats febles amb sustantius en plural. I per descomptat, la participació d'una entitat feble en la relació identificadora és total. És més, entre una entitat feble i la corresponent identificadora hi ha una dependència d'existència. Només faltaria. Si no calgués que un element de l'entitat feble estigués relacionat amb algun de la identificadora, llavors no el podríem identificar. I en una base de dades, els elements de totes les entitats han de poder ser identificats.

4.9.3 Classificació de les Relacions 1:N

Tanquem el tema de les relacions 1:N amb una sinopsis que ens ajudarà a comprendre els diferents nivells de compromís que pot haver-hi entre les dues entitats relacionades en una relació de cardinalitat 1:N.

Recordem que el fet que una relació 1:N sigui amb dependència d'existència és una decissió del disseny. Depèn de si ens interessa mantenir els elements del costat N quan s'esborra element associat al costat 1.

O, vist d'una altra manera, depèn de si acceptem elements a l'entitat del costat N sense tenir informació de l'element que li correspon al costat 1, o no. En definitiva, és el mateix concepte el de dependència d'existència que el de participació total.

Portat a l'exemple del model de persones que viuen a ciutats, en el Model 4.12 de la pàgina 66, noteu que segons quin ús se li hagi de donar a la base de dades pot interessar que quan s'esborri una ciutat, les persones que hi viuen segueixin presents a la base de dades (sense tenir informació d'on viuen), o no. És a dir, que PERSONA participi total o parcialment a la relació CIUTAT.

La participació total significa dependència d'existència. O sigui, que ens interessa tant saber d'on és una persona, que si s'esborra la ciutat on viu, també eliminem la persona. En altres paraules, si no sabem on viu, prohibim la inserció de la persona.

Un cop introduïdes les diferències entre els nivells de compromís en les relacions 1:N fora bo que no hi hagués problema en comprendre la Figura 4.11.

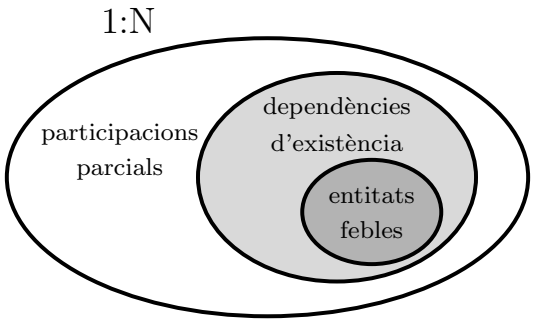


Figura 4.11: *Relació jeràrquica entre relacions 1:N.*

De la imatge de la Figura 4.11 se'n treuen qualsevol d'aquestes conclusions.

- Qualsevol relació identificadora és una relació amb dependència d'existència, ja que si no existeix qui pugui identificar uns elements d'una entitat, no poden existir aquests elements.
- Hi ha relacions 1:N que no són amb dependència d'existència. Són relacions en les que l'entitat del costat N hi participa parcialment. Per exemple, en el disseny del Model 4.12, cas que ens interessi tenir una persona a la base de dades encara que no sabem on viu.
- Hi ha relacions amb dependència d'existència que no són identificadores. Ja que els elements de l'entitat del costat N tenen la seva pròpia clau, i malgrat per poder existir necessiten que existeixi un element relacionat en l'entitat del costat 1, pel fet de poder-se identificar per ells sols, la relació que és amb dependència d'existència no és una relació identificadora.
- I per descomptat, una relació identificadora és una relació 1:N.

4.10 Relacions M:N

Les relacions binàries amb cardinalitat M:N permeten cada element d'E_1 associar-se amb molts elements d'E_2, així com cada element d'E_2 associar-se amb molts d'E_1.

Com analistes, la primera qüestió que ens ve al cap és per què no fem totes les relacions M:N ja aquest tipus de relacions engloben les altres. I la resposta és que l'inconvenient que tindria fer totes les relacions d'una base de dades M:N seria la complicació innecessària tant del disseny com de la implementació.

Dit això, recuperem amb gran interès el producte cartesià explicat en la Secció 1.2.4. Perquè és del que estem parlant. Podem entendre una relació M:N entre dues entitats E_1 i E_2 com una matriu amb els elements d'E_1 un per fila, i els d'E_2 un per columna. El contingut de la matriu llavors serien booleans, que ens dirien si l'element d'E_1 corresponent a la fila està relacionat amb l'element d'E_2 corresponent a la columna.

Aquestes relacions es representen com es mostra en la Figura 4.12.

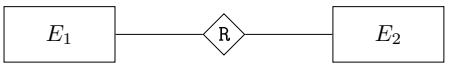


Figura 4.12: *Relació M:N en un model ER.*

Mai podrem prescindir del nom d'una relació M:N. És més, una relació amb aquesta cardinalitat haurà de ser considerada com un nou node del graf que és el model. La raó és filosòfica. Així com per les relacions de cardinalitat inferior no es produeix un efecte multiplicatiu, ni per tant d'augment de l'espai necessari per guardar els valors relacionats, aquí això sí que efectivament passa.

Si diem que E_1 té $n = |E_1|$ elements i E_2 en té $m = |E_2|$, llavors és clar que el nombre d'elements que poden haver-hi relacionats per mitjà d'una relació M:N és, com sabem pel producte cartesià de dos conjunts, el producte de la quantitats d'elements que tingui una entitat multiplicat per l'altra és a dir $|E_1 \times E_2| = nm$. Aquesta és la quantitat de parelles d'elements que poden estar relacionats. I normalment és superior a n i a m, i per tant, per guardar les parelles d'elements que estiguin relacionats ens caldrà un espai addicional. Aquest espai l'aconseguim així, considerant la relació com un nou node del graf, cosa que fa que s'assemblin a les entitats.

A més, aquestes relacions poden tenir atributs, cosa que fa que encara s'assemblin més a les entitats. O sigui, que si cal entendre una relació d'aquesta cardinalitat com un nou node en el graf i a més pot tenir atributs, la qüestió que ens hauria d'interessar és quina característica distingeix les relacions M:N de les entitats.

La resposta és l'existència d'una clau. La diferència entre una relació M:N i una entitat és que una relació M:N no té clau per naturalesa. És a dir, l'usuari final no ens podrà dir com identifica els elements d'una relació.

Posterior al disseny, en la implementació, sovint es defineix com clau d'una relació M:N la parella de claus dels elements relacionats. Tot i així, la finalitat amb la que s'estableix aquesta definició més que per identificar cada element de

la relació és simplement per significar que no s'admetran repetits. En qualsevol cas, en un disseny no s'admet establir identificació en les relacions. És més, en el moment que posem un camp clau efectivament identificador en una relació l'estem convertint en entitat. Aquest és un extrem en el que altre cop es demostra l'enginy del dissenyador. Trobar paraules per transformar en entitats les relacions M:N requereix certa habilitat.

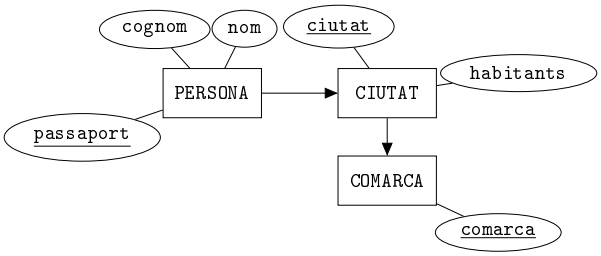
Quan ens preguntem si una relació és M:N, cal dir dues frases, en veu alta millor,

Un E_1 pot relacionar-se amb molts E_2...
...i un E_2 pot relacionar-se amb molts E_1.

Encara que sembla fàcil, segur que trigarem més del que sembla a formular-les. Perquè per dir-les, cal concentrar-se dos cops. S'està expressant una dualitat, i això costa.

Vist des de la perspectiva de l'anàlisi matemàtica, una relació binària M:N representa una funció booleana de dues variables independents. $f = f(x, y)$, sent $f \in \{0, 1\}$, i definida sobre el conjunt que resulta del producte cartesià entre les entitats, és a dir sobre qualsevol parella tal que $x \in E_1$ i $y \in E_2$. I el fet que la funció tingui una resposta booleana, pot ser u o zero, significa que els elements x i y estan relacionats, o no.

Prendrem el disseny del Model 4.16 com a punt de partida per l'aplicació del club esportiu que ens acompanyarà per gran part dels exemples que es veuran en aquest i en propers capítols. En la Secció 2.6.1 s'ha donat una definició de requerimens molt breu però prou clara del que ha de suportar aquesta aplicació.



Model 4.16. *Model inicial abans d'introduir una relació M:N.*

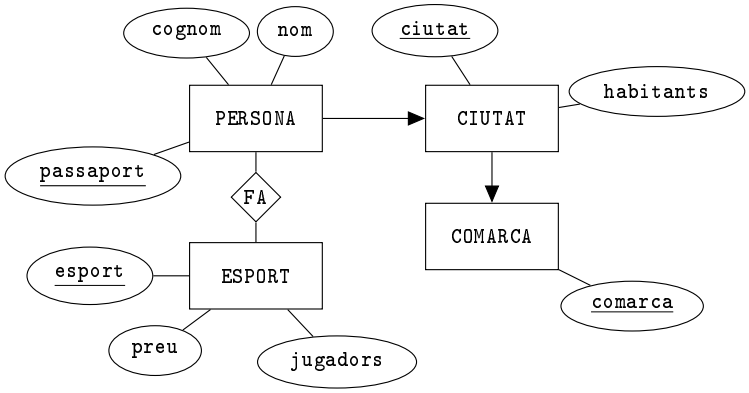
Resulta que cal guardar quins esports practica cada persona, que en definitiva són els socis del club. Per això, els socis paguen un preu mensual per cada esport que vulguin practicar.

També interessaria poder formar equips entre els socis, i per això interessaria disposar de quants jugadors formen un equip de cada esport.

Qui dissenya la base de dades pensa, d'entrada, com s'identifica cada esport en el club. Pregunta, i li diuen que amb el nom. O sigui, segur que no hi pot haver dos esports que es diguin igual. Després, destil·la d'entre els requeriments quines són les dades que van lligades a cada esport, i observa que són el preu, i els jugadors que composen un equip. L'entitat esport està clara. Ara ens plantegem la relació que té amb les persones i pensem, amb cura, formulant aquestes dues frases en veu alta.

Una persona pot fer molts esports...
...i un esport pot ser fet per moltes persones.

Sempre resulta convenient preguntar-li a l'usuari final de l'aplicació si hi està d'acord, I un cop ho tinguem clarissim, llavors establim la relació M:N tal com s'il·lustra en el Model 4.17.



Model 4.17. *Exemple de relació M:N en un model ER.*

La relació FA guarda informació de quins esports fa cada persona, així com de quines persones fan cada esport. Noteu que aquesta última afirmació ve estructurada per les paraules "quins" i "cada". O sigui, la relació R guarda *quins* E_1 per *cada* E_2, i *quins* E_2 per *cada* E_1.

En la documentació que acompanyés aquest model ER hi hauria de figurar les explicacions de que l'atribut `jugadors` és el nombre de jugadors per equip, plural que significa nombre enter. I també que el `preu` és un número amb decimals, pels cèntims. Noteu finalment que els atributs de la nova entitat esport són les dades que depenen exclusivament de l'esport.

En els requeriments de la Secció 2.6.1 es demana que cada persona tingui una adreça de correu electrònic, i varis números de telèfon. Més endavant caldrà afegir-los al Model 4.17 per ajustar-lo a l'aplicació per al club esportiu.

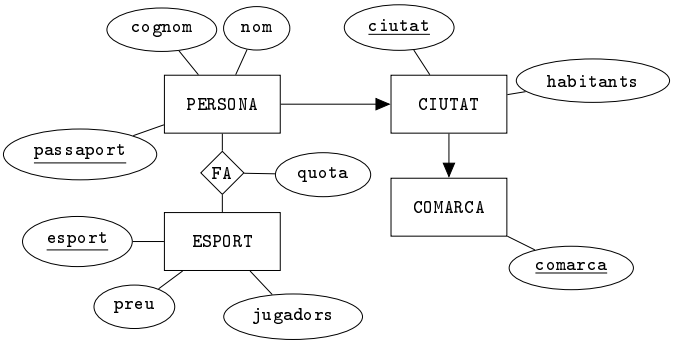
4.10.1 Atributs en relacions M:N

Tal com es diu en els requeriments, el preu dels esports ha de dependre de cada persona perquè el club esportiu vol fer ofertes com que el primer mes és gratuït, o que el que es paga depèn de l'edat del socis, llavors amb guardar el preu no n'hi ha prou. Cal un nou atribut, `quota`, que depèn a més de l'esport, de la persona. Així ens reservem la possibilitat de que per un mateix esport, diferents persones paguin diferents quantitats. Si després resultés que tots paguen el mateix podríem posar el mateix valor en tots els registres, que coincidiria amb el preu de l'esport. Però si en algun moment això es convertís en una regla, llavors estaríem introduint redundància. La qüestió que cal formular-se per adonar-se'n d'aquestes situacions està relacionada amb la interpretació de l'anàlisi matemàtica feta més amunt.

Què ens cal per poder parlar d'una quota?.

I observem que ens cal una persona i un esport.

Un cop convençuts que el raonament és correcte, llavors podem afegir atributs a la relació, tal com es mostra en el Model 4.18.



Model 4.18. *Exemple d'atributs en una relació M:N d'un model ER.*

Queda clar, l'atribut `quota` de la relació `FA` vol dir que cada persona paga una quantitat concreta per cada esport.

Aquest tipus d'atributs associats a relacions M:N són molt habituals. Si el nombre d'atributs d'una relació M:N augmenta excessivament, llavors cal considerar la possibilitat de fer una agregació, donant clau primària a la relació `fa`, i amb una mica d'imaginació, trobant un substantiu que li doni nom.

4.11 Autorelacions

Anomenem *autorelació* la relació d'una entitat amb ella mateixa.

Amb les autorelacions s'introdueix també el concepte de rol. Un *rol* en un disseny és una paraula que ens diu quin és el paper que juga una entitat en una relació. En el model es representen amb el nom del rol al costat de la línia que uneix l'entitat amb la relació. I no té res a veure amb els rols de la base de dades que són els tipus d'usuari.

Tot el que s'ha dit pel que fa a les relacions binàries entre entitats en un model ER és igualment vàlid quan les dues entitats de la relació són la mateixa, excepte pel fet que en aquest cas cal augmentar un nivell els objectes identificats. És a dir, tant les autorelacions 1:1 com les 1:N ara sí que necessitaran que els hi posem un nom identificador. I pel cas de les M:N, encara més. Per les M:N cal usar rols forçosament.

Seguidament es veuen en detall les autorelacions per cada tipus de cardinalitat.

4.11.1 Autorelacions 1:1

Amb una autorelació 1:1 podem representar des d'una simple seqüència, com representa la relació successor pels números naturals, fins una relació d'equivalència que particioni els elements d'una entitat en classes d'equivalència. Per comprendre la capacitat expressiva que tenen aquest tipus de relacions, la millor manera és imaginant un graf dirigit on tots els nodes tenen un arc entrant i un de sortint, que poden ser el mateix.

En la Figura 4.13 els nodes del graf representen els elements d'una entitat amb participació total en la relació que ve indicada pels arcs. Noteu que aquesta figura presenta tan sols una possibilitat entre una enorme quantitat que van des d'un cicle que connectés tots els elements, fins una relació que els agrupés per parelles. Totes aquestes possibilitats mantenint la participació total.

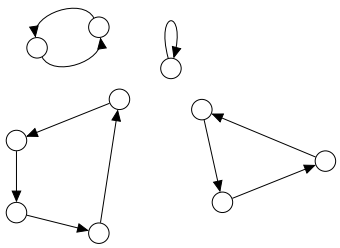


Figura 4.13: *Potència expressiva de les autorrelacions 1:1.*

En cas que l'entitat tingués participació parcial en l'autorelació 1:1, llavors obrim encara més el ventall de possibilitats de representació incloent nodes terminals i nodes aïllats.

El símbol que s'utilitza per les autorelacions 1:1 és el de la Figura 4.14.

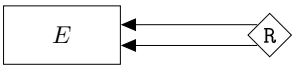
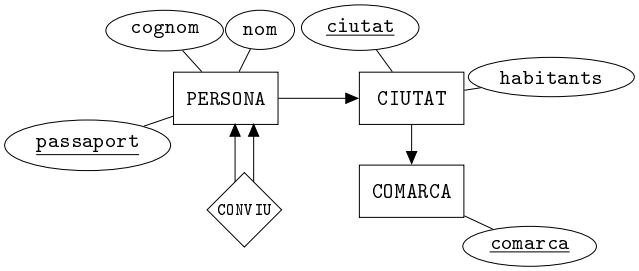


Figura 4.14: *Autorelació 1:1 en un model ER.*

Tot seguit es mostra un exemple d'aquest tipus de relacions, a partir del disseny del Model 4.16 de la pàgina 74, on hi ha persones que viuen en ciutats que estan dins de comarques.

Posem per exemple que interessa guardar per cada persona qui és la seva parella. És a dir, per cada persona volem saber les dades de la persona amb qui conviu, sempre que sigui una altra persona de la base de dades. Consti que no es pretén modelar la realitat sinó exemplificar una autorelació 1:1.

L'aspecte que tindria en el diagrama la relació CONVIU es pot observar en el Model 4.19.



Model 4.19. *Exemple d'autorelació 1:1 en el Model 4.16.*

Per descomptat que la participació de l'entitat PERSONA a la relació CONVIU és parcial, perquè volem poder inserir persones que visquin soles.

Fem una anàlisi breu per adonar-nos-en que si fos una paticipació total, es produiria un cas de dependències d'existència creuades, tal com s'ha vist en la Secció 4.7.1. És a dir, no podríem inserir persones perquè hauria d'existir la seva parella, però tampoc podríem inserir la parella de la persona, perquè hauria d'existir la seva parella, que és la persona que estem pretenent inserir. Un peix que es mossega la cua.

Ja s'ha dit que els mecanismes per trencar aquest tipus de cicles en els SGBDs, posposen la verificació de les restriccions d'integritat fins a nou avís. O sigui, permeten violar les restriccions transitòriament.

4.11.2 Autorelacions 1:N

A diferència de les relacions entre entitats diferents, amb les autorelacions succeeix que qualsevol fenòmen que pugui representar-se amb una autorelació 1:1 tambe pot ser representat amb una autorelació 1:N sense cap més esforç que restringir repetits. En altres paraules, no hi hauria cap problema en utilitzar una autorelació 1:N per explicar les relacions de la Figura 4.13. Com a conseqüència les autorelacions 1:1 queden arraconades, i no s'utilitzen gairebé mai, ja que la diferència amb les autorelacions 1:N es limita a establir una restricció d'unicitat.

Les autorelacions 1:N amplien la capacitat expressiva de les 1:1. Serveixen particularment per descriure jerarquies entre els elements d'una entitat, cosa que no pot representar-se amb autorelacions 1:1, degut a la restricció d'unicitat. Dos no poden apuntat al mateix.

Una jerarquia es pot representar en un *vector de predecessors*, cosa que s'il·lustra en la Figura 4.15. En la part esquerra es mostra una estructura jeràrquica, o sigui un arbre. En la part central, hi ha una seqüenciació per nivells del nodes mantenint l'estructura arborescent, imatge que convé retenir com a concepte de vector de predecessors. I en la part dreta, es pot veure el contingut del vector de predecessors pròpiament dit, on cada casella conté l'índex del node pare. Noteu que aquesta estructura té un -1 com a pare del node rel. De la mateixa manera podrien haver-hi varies rels convertint la representació en un conjunt d'arbres disjunts. És a dir, un bosc.

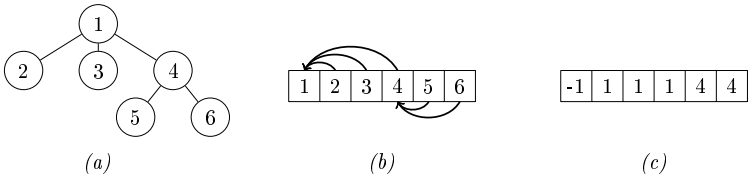


Figura 4.15: *Vectorització d'un arbre. (a) Arbre; (b) Redisposició seqüencial; (c) Vector de predecessors.*

Fixeu-vos en que un vector de predecessors és una estructura recursiva, i per tant els tractaments per recórrer la jerarquia que hi ha per sobre un node cal plantejar-los recursivament. En definitiva, una jerarquia és una relació 1:N entre els elements d'un mateix conjunt, ja que cada node pot tenir molts fills, però només un pare.

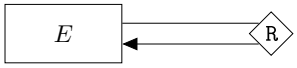


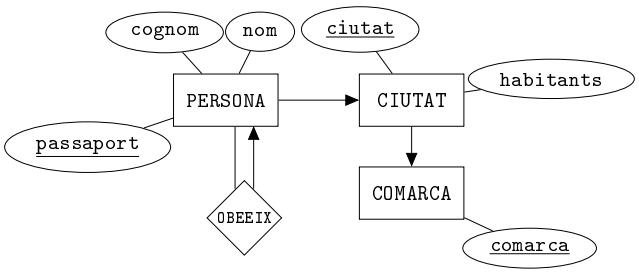
Figura 4.16: *Autorelació 1:N en un model ER.*

El símbol gràfic corresponent a una autorelació 1:N es veu a la Figura 4.16.

Un cop més recuperem el Model 4.16 per posar l'exemple d'autorelació 1:N. Aquest cop suposarem que aquell disseny de persones, ciutats i comarques és per gestionar el personal del club esportiu, de manera que les persones del model són els treballadors.

A més, com es diu en els requeriments de la Secció 2.6.1 hi ha caps d'àrees que tenen treballadors al seu càrrec. És a dir, els treballadors tenen un cap que els mana. O sigui, un cap al que han d'obeir, per molt lleig que soni.

Llavors, reflectim una autorelació `OBEEIX` amb cardinalitat 1:N en el model.



Model 4.20. *Exemple d'autorelació 1:N en el Model 4.16.*

L'autorelació del Model 4.20 es llegeix com que cada persona obeeix a una persona com a màxim, però pot manar-ne a moltes. Deixem-ho aquí.

4.11.3 Autorelacions M:N

Les autorelacions M:N expressen dependències entre elements d'una mateixa entitat, de manera que un element pot ser relacionat amb molts, i al mateix temps, molts elements es poden relacionar amb l'un. Això introdueix una confusió en el model que cal desambigüitzar amb els rols.

Es pot intuir que la potència expressiva d'aquesta eina és gran. De fet, estem parlant del tipus de relació que hi ha entre els nodes d'un graf, expressada mitjançant les arestes que els uneixen. La capacitat de representació d'una autorelació M:N és la mateixa que la d'un graf com estructura. O sigui, extremadament abstracte. Tant és així, que com exemple podem imaginar una entitat `NODE` amb una autorelació que s'anomenés `CONNECTA` que vindria a representar el conjunt d'arestes d'un graf.

Seguint en aquesta línia, si el graf fos dirigit, els rols podrien ser `origen` i `desti`. I si fos un graf no dirigit, llavors tan sols podríem anomenar els nodes amb uns noms de rols que no tindrien massa contingut semàntic, $\texttt{node}_1$ i $\texttt{node}_2$,

de manera que reflectiríem l'ambigüitat d'un graf no dirigit en els rols. No és feina del dissenyador distingir entre conceptes que en la realitat es confonen.

De tota manera, és una norma obligatòria en un model ER que contingui una autorelació M:N la presència de rols que etiquetin les línies que hi ha entre l'entitat i l'autorelació.

Gràficament es representen com s'il·lustra en la Figura 4.17.

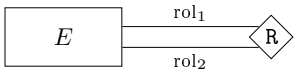
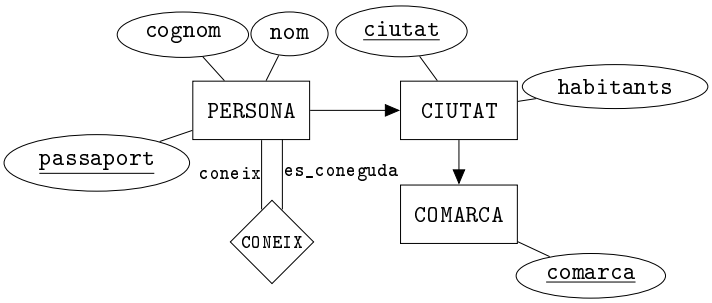


Figura 4.17: *Autorelació M:N en un model ER.*

Seguim el fil dels exemples que s'han anat donant en aquestes últimes seccions. En el club esportiu es vol saber quines persones coneixen a quines altres. Ve a ser una mena de xarxa social interna, amb l'objectiu específic de formar equips. Això ens portarà a un model més realista que en l'exemple de les autorelacions 1:1, on se suposava que cada persona només podia conviure amb una sola persona.

Cal modelar el fet que una persona en pot conèixer moltes. I també pot ser coneguda per moltes altres.



Model 4.21. *Exemple d'autorelació M:N en el Model 4.16.*

En el disseny del Model 4.21 es mostra l'autorelació M:N que explica que unes persones coneixen unes altres persones. Els rols `coneix` i `es_coneguda` són necessaris en el model, ja que transcendeixen al model relacional un cop acceptat el disseny, a l'hora d'implementar-lo.

4.12 Càlcul Automàtic de les Cardinalitats

Fàcilment es pot automatitzar la tasca d'establir la cardinalitats de les relacions. O sigui, és una cosa simple, però s'ha de tenir clar. Més que tenir utilitat funcional, el codi que es mostra en aquesta secció serveix per aclarir la manera de procedir a l'hora d'establir les cardinalitats d'un model ER.

Tot i així, l'algorisme de la Caixa 4.1 tan sols serveix per bases senzilles, ja que considera que entre entitats hi pot haver una relació com a molt, no més d'una. És a dir, el pseudocodi que es mostra tot seguit no serviria pel disseny del Model 4.13, on entre persones i ciutats hi ha dues relacions.

```
funcio establir_cardinalitat(E1,E2) retorna string {
    quants12 = llegir("Quants " + E1 + " té " +
                                "un únic " + E2 + "(CAP, 1, o N) ?");
    si (quants12 = "CAP") retorna nul;
    si (quants12 = "1") {
       quants21 = llegir("I quants " + E2 + " té " +
                                "un únic " + E1 + " (1, o N) ?");
    }
    sino {
       quants21 = llegir("I quants " + E2 + " té " +
                                "un únic " + E1 + " (1, o M) ?");
    }
    retorna quants12 + ":" + quants21;
}

procediment establir_cardinalitats(BD) {
    n = |BD|;
    per cada i en 1..n {
       per cada j en 1..i {
          cardinalitat = establir_cardinalitat(BD[i], BD[j]);
          si (cardinalitat) {
             escriure("La relació entre " + i + " i " + j +
                                " té cardinalitat " + cardinalitat);
          }
       }
    }
}
```

Caixa 4.1. *Pseudocodi per establir les cardinalitats d'una base de dades BD.*

El procediment en pseudocodi `establir_cardinalitats`, en plural, de la Caixa 4.1 rep per paràmetre d'entrada una col·lecció que anomena *BD*, d'entitats. No retorna res, ja que és un procediment i no una funció. El resultat queda al canal de sortida. Fa la meitat del recorregut del conjunt $BD \times BD$, és a dir, recorre la meitat del producte cartesià del conjunt d'entrada per ell mateix. Això és, tracta cada parella possible d'entitats un sol cop, incluint-hi les d'una entitat i ella mateixa. Observeu que com que el bucle intern ve limitat per la

variable del bucle extern, en el cos interior als dos bucles es tracten totes les parelles (i, j) tals que $i \geq j$. Així ens assegurem que només tracti un cop cada parella.

Llavors, per cada parella possible d'entitats fa una crida a una funció que es diu `establir_cardinalitat`, en singular, que rep per paràmetres dues entitats, i retorna un string que pot valer "1:1", "1:N", o "M:N", o bé pot ser nul, en cas que entre les dues entitats actuals no hi hagi cap relació.

La funció `establir_cardinalitat` no fa res més que fer reflexionar a l'usuari preguntant-li quina relació de cardinalitat hi ha entre les entitats que se li han passat com a paràmetres d'entrada. I ja que en el codi de la Caixa 4.1 no hi ha cap control d'errors, almenys en la mateixes preguntes se li mostren les respostes possibles.

La primera pregunta és quants elements de la primera entitat es poden relacionar amb un únic element de la segona. La resposta de l'usuari es recull en la variable local `quants12`. Hi ha tres possibilitats per aquesta primera resposta. "CAP","1", o "N".

- Si l'usuari diu que cap, llavors no hi ha relació entre les entitats, i es retorna nul.
- I si no, cal una segona pregunta que és la inversa de la primera. És a dir, quants elements de la segona entitat es poden relacionar amb un únic element de la primera. Les respostes possibles són dues, que també depenen de la primera resposta, com es pot veure. En qualsevol dels dos casos, les respostes admissibles també s'indiquen entre parèntesis.

Quedi clar que el codi de la Caixa 4.1 és orientatiu. Bàsicament es pretén donar entendre que un cop decidits els conceptes que esdevindran entitats en el model ER, llavors per cada parella possible s'ha d'analitzar si hi ha una relació, i en cas que sí, de quina cardinalitat. No es tracta d'una panacea, ja que ens dirà relacions redundants, perquè ens les dirà totes. Per exemple, utilitzant-lo amb el disseny del Model 4.16 ens diria que entre `PERSONA` i `COMARCA` hi ha una relació 1:N, bé, en rigor ens diria N:1. I realment aquesta relació no ha d'estar en el model perquè és absorvida per les altres dues. Però bé, aquest darrer raonament també ha de formar part de l'anàlisi, i per tant, és cosa bona que ens el plantegem.

4.13 Relacions Ternàries

Donades tres entitats E_1, E_2, i E_3, i una circumstància que les involucra totes tres, podem establir una relació ternària que les lligui. És clar que si les relacions M:N en un model ER provoquen l'existència de nous nodes en el graf, encara més

les relacions ternàries. Aquest tipus de relacions són extremadament concises, i donen molta informació amb tota la coherència necessària, encara que mal utilitzades suposen la generació de nuls estructurals.

Des del punt de vista de l'anàlisi matemàtica, una relació ternària vindria a ser una funció lògica que depengués de tres variables independents. És a dir $f = f(x, y, z)$, sent x un element d'E_1, y un d'E_2, i z d'E_3. Així doncs, el domini on està definida f es pot representar en un espai tridimensional, o sigui com un cub, encara que més aviat un paral·lelepípede, en el que cada un dels tres eixos es correspon amb una disposició seqüencial dels elements de cada una de les tres entitats. És a dir, el conjunt resultant del producte cartesià entre les tres entitats, $E_1 \times E_2 \times E_3$. I això vol dir que cada element de la relació ternària és algun dels trios possible format per un element de cada entitat.

La frases que hem de pronunciar, en veu alta, abans d'establir una relació ternària entre tres entitats ha de tenir tres cops la paraula "qualsevol".

Qualsevol E_1 pot estar relacionat amb qualsevol E_2 i qualsevol E_3.

Convé a més, repetir aquesta afirmació amb els diferents ordres entre les entitats. O sigui, afegir

I qualsevol E_2 pot estar relacionat amb qualsevol E_3 i qualsevol E_1, i també qualsevol E_3 pot estar relacionat amb qualsevol E_1 i qualsevol E_2.

Formular aquestes consideracions en veu alta fa que reflexionem si efectivament el fenòmen que pretenem modelar s'ajusta a una relació ternària.

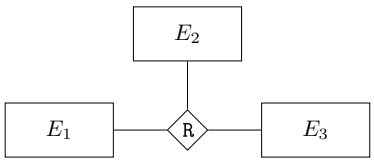


Figura 4.18: *Relació ternària en un model ER.*

La representació d'una relació ternària en un model ER és la que es pot veure en la Figura 4.18.

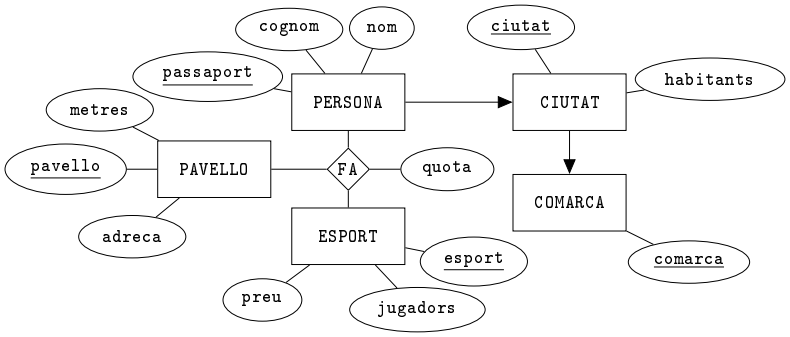
Reprenem l'exemple del Model 4.17 pel club esportiu on hi havia persones que vivien en ciutats que estaven en comarques, i que podien fer diversos esports. Bé, ara entren a escena els diferents pavellons del club. Són pavellons

polivalents que en qualsevol d'ells s'hi pot fer qualsevol esport. I suposem que resulta interessant saber a quins pavellons fa cada esport cada persona. Noteu l'estructura *quins...cada...cada*.

El raonament que ens condueix a establir-hi una relació ternària passa per les següents reflexions.

> *Qualsevol persona pot fer qualsevol esport en qualsevol pavelló, qualsevol esport es pot fer en qualsevol pavelló per qualsevol persona, i qualsevol pavelló pot ser utilitzat per qualsevol persona per fer qualsevol esport.*

I si l'usuari final ho aprova, podem tirar endavant amb el Model 4.22.



Model 4.22. *Exemple de relació ternària afegit al Model 4.18.*

4.13.1 Cardinalitats en les relacions ternàries

La relació ternària del Model 4.22 es diu que té una cardinalitat M:N:P. Fixeu-vos que el plantejament que s'ha fet per introduir-la en el model s'ha basat en repetir tres cops la paraula "qualsevol".

En les relacions binàries les cardinalitats condicionen el model. Al llarg de les seccions anteriors s'ha vist com podíem fusionar les relacions 1:1, o com fer desaparèixer del diagrama una relació 1:N. I també com una M:N causava la introducció d'un nou node en el model. Per les relacions ternàries tot això és molt diferent. La capacitat expressiva d'una relació ternària és tal que en qualsevol cas generarà un nou node en el graf que és el model ER. I per tant, tot el que sigui cardinalitats deixarà de condicionar l'espai utilitzat.

Les cardinalitats en les relacions ternàries no transcendeixen a la implementació. És a dir, qualsevol restricció de participació s'haurà de controlar a

partir de la lògica que finalment, després d'haver implementat el model, afegim a la base de dades en forma de procediments per mitjà dels llenguatges que el SGBD ens posi a l'abast. Per això, no se li dóna tanta importància a les cardinalitats de les relacions ternàries, ja que sempre es poden modificar en última instància.

Amb una finalitat estrictament documental, admetrem simesnó que aparegui una fletxa com a molt en una entitat ternària, tal com es dibuixa en la Figura 4.19.

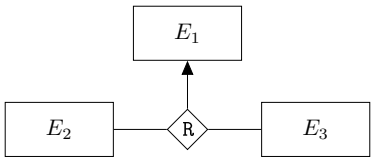


Figura 4.19: *Relació ternària amb cardinalitat 1:M:N.*

El model de la Figura 4.19 significa que cada parella formada per un element de l'entitat E_2 i un de l'E_3, pot ser relacionada amb un únic element de l'entitat E_1. És a dir, que no s'admeten repeticions de parelles formades per un element d'E_2 i un d'E_3. Això cal documentar-ho necessàriament.

Més d'una fletxa en una relació ternària en un model ER està prohibit. Degut a la confusió de la semàntica que podria voler indicar no es considera convenient usar-ho en un model. Per això, si en una relació ternària hi ha més d'una fletxa es tracta d'un error. No hi ha cap interpretació oficial.

4.13.2 Relació ternària amb tres relacions binàries

Sovint hi ha qui es planteja tres relacions binàries M:N enlloc d'una sola relació ternària. És a dir, el model de la Figura 4.20 enlloc del de la 4.18.

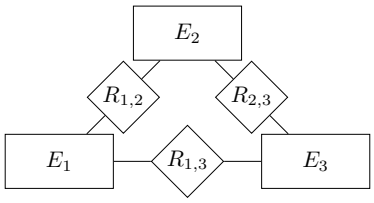


Figura 4.20: *Alternativa incorrecta a una relació ternària.*

Són coses molt diferents.

El model de la Figura 4.18 és molt més concís, ja que entre altres coses demana que existeixi un tercer element per poder-ne relacionar qualssevol dos.

El 4.20 és molt menys restrictiu, i en moltes ocasions, quan l'utilitzem enlloc d'una ternària, introduirem confusió a la base de dades.

La diferència entre ambdos models és considerable, i cal saber quin tipus de relació es necessita en cada cas. Essencialment, el contrast està en que en el model 4.18 hi ha una sola relació, i per tant tot va lligat. En el 4.20 les relacions són independents. O sigui, si la possibilitat de que E_1 és relacioni amb E_2 no té res a veure amb l'existència d'un element d'E_3 que hi participi, llavors sí que el model 4.20 està utilitzat correctament. De fet, això no té res que veure amb les relacions ternàries.

4.14 Model Entitat Relació Estès

L'antiguitat que té tota la teoria de bases de dades és una garantia de la seva robustesa. El model ER com a mètode de disseny de bases de dades és igual avui a data de 2014 que fa quaranta anys. I això és fantàstic, perquè vol dir que al llarg de tots aquests anys ha demostrat reiteradament la seva funcionalitat.

I no només això. L'èxit dels diagrames entitat relació en les aplicacions de dades ha estat tan rotund que ha creat escola. Els diagrames de classes que s'utilitzen intensament en la programació s'han servit d'algunes de les propostes del model ER. Especialment pel que fa a les cardinalitats. També, alguns dels artefactes que es defineixen en enginyeria de projectes en UML tenen la seva rel en el model ER.

Tot i així, hi ha hagut un moment de la història, l'únic, en el qual el disseny de bases de dades s'ha alimentat d'una nova tecnologia. La programació orientada a objectes, que encara no té trenta anys. I la cosa bona que té la programació orientada objectes, i que en un principi era una carència dels models ER i per això finalment ho va incorporar, és la capacitat d'allotjar dades de manera condicional. Condicionades a un valor d'alguna dada previament allotjada. Estructures que permetin guardar si una persona és un bomber o un taxista, i que a més, en cas que sigui bomber, es pugui guardar si és caporal, o ras, és a dir el càrrec. I si és taxista, guardar el número de licència de taxi. La solució antiga, amb un model ER, establiria les dues entitats `BOMBER` i `TAXISTA`, i en cada una d'elles hi col·locaria els atributs necessaris per guardar el que es demana. Llavors, però, hi ha una sèrie de dades com el nom, el cognom, o el número de passaport, que estarien en les dues entitats, i que, per qüestions de disseny es guardarien separades malgrat que lògicament fan referència a un mateix tipus de concepte, les persones.

De vegades interessaria definir restriccions d'unicitat entre tots els valors no tan sols d'un atribut d'una taula, sinó també entre els de dos atributs de dues taules. Per exemple, suposant que tenim la forma antiga de dues taules, `BOMBER` i `TAXISTA`, i que a més no es permetés l'existència de bombers que fossin també

taxistes, llavors ens agradaria poder dir que el número de passaport ha de ser diferent no només entre tots els bombers, sinó també entre els bombers i els taxistes. Amb les eines que s'ha donat fins ara, això no és possible.

4.14.1 Especialització i Generalització d'Entitats

Així doncs, una aportació molt important que la programació orientada a objectes va fer al disseny de bases de dades és la capacitat de considerar un tipus d'objecte com una extensió d'un altre. Les herències.

Es representen en una estructura com la de la Figura 4.21.

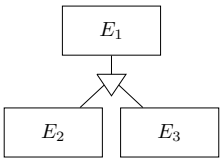
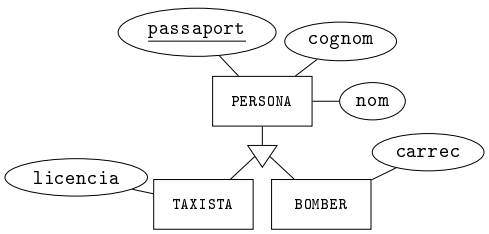


Figura 4.21: *Especialització o generalització en un model ER.*

A primer cop d'ull, s'observa que el model 4.21 és el primer en el que les línies que uneixen entitats no representen estrictament relacions. En la Figura 4.21 hi ha tres entitats. E_1 s'anomena entitat *genèrica*, o *mare* si es vol usar la terminologia de la programació orientada a objectes. Tant E_2 com E_3 són les entitats *especialitzades*, o *derivades* en aquell vocabulari.

El símbol de la Figura 4.21 significa que qualsevol atribut que s'afegeixi a E_1, és com si s'afegís automàticament a E_2 i a E_3. D'aquests atributs, els d'E_1, en diem atributs *genèrics*. Anomenem atributs *específics* als que no són genèrics, és a dir, els que es troben exclusivament a les entitats especialitzades, E_2 i E_3. A partir d'aquesta estructura cada cop que s'insereixi un element a qualsevol de les entitats especialitzades també s'estarà afegint de retruc a la genèrica. Però no al contrari és clar.

Un exemple senzill que il·lustra aquests conceptes és al Model 4.23.



Model 4.23. *Exemple d'especialització o generalització en un model ER.*

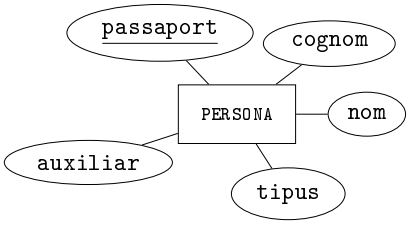
Segons aquest model, per les persones que no siguin taxistes ni bombers ens guardarem nom, cognom, i passaport. Pels taxistes, a més ens guardarem la licència. Observeu que enlloc ens guardem el fet que la persona sigui taxista. Ho sabrem perquè té licència. I pels bomber a més de les dades de qualsevol altra persona, tindrem el càrrec.

Noteu que dins l'àmbit del disseny de bases de dades es distingeix entre dos conceptes que en última instància van a parar a una mateixa representació.

Per un costat, es considera una especialització quan una entitat pot ser de diferents tipus. I segons de quin tipus sigui desitjaríem guardar altres dades addicionals, els atributs específics. Per decidir crear una especialització, a més ha de succeir que els tipus específics d'entitats es presentin en proporcions semblants. És a dir, seguint amb l'exemple, si de les persones que ens interessa guardar hi hagués un 95% de taxistes, llavors gairebé valdria la pena, trencant la màxima de nuls estructurals, guardar el número de licència per totes les persones deixant-lo nul en el 5% de les persones restants.

I d'altra banda, el procés de creació d'una generalització és a la inversa. I més senzill. D'entrada a partir del model que tenim, observem que hi ha un conjunt d'atributs presents a varies entitats. I llavors decidim crear una nova entitat amb aquests atributs comuns. Això requereix inventar-se un nom per aquesta nova entitat.

Hi ha mals dissenyadors que, creient-se molt espabilats, pensen que la solució que es mostra en el Model 4.24 és bona, inclús millor que la del Model 4.23 ja que s'estalvia una entitat. Per entendre aquesta proposta s'expliquen dient que l'atribut `tipus` guardarà un dels valors *bomber* o *taxista*. I llavors, si `tipus` val *bomber*, l'atribut `auxiliar` contindrà el valor del `carrec`. I si val *taxista*, llavors l'`auxiliar` guardarà la `licencia`. I ho expliquen orgullosos, amb un somriure.



Model 4.24. *Bunyol inadmissible en un model* ER.

L'existència de l'atribut `tipus` encara. Es pot justificar en els casos que hi hagi més d'un tipus d'entitats específiques de les que no ens interessa guardar informació addicional. Quan això passa, es pot posar aquest atribut, tot i que introdueix redundància per les entitats específiques que sí que tenen atributs específics.

Ara bé, quedi clar que la idea "intel·ligent" de guardar el tipus de persona de manera explícita en un atribut quan no fa falta, i, a més, un atribut auxiliar que segons el tipus guardi la licència o el càrrec, és un nyap que suposa un insult al model introduint una dependència funcional palmària.

Recordeu de la Secció 4.9 que una dependència funcional en una entitat es produeix en el moment en que el valor d'un atribut condiciona el valor d'un altre dins la mateixa entitat. Tot plegat resulta en una complicació incòmode que forçaria qualsevol tractament de l'entitat a considerar casos. És per tant un greuge a la legibilitat, a més a més. Les dependències funcionals violen les formes normals per les taules relacionals. En aquest llibre no es descriu la teoria de les formes normals, però és respecta. Per qui estigui interessat pot trobar la descripció minuciosa a [10].

Cal treure's del cap idees com que si aquest atribut val no se què, llavors aquest altre contindrà això, i si no allò. Això són dependències funcionals i per tant dissenys tan inadmissibles com el del Model 4.10.

Especialitzacions o generalitzacions completes

Quan una especialització, o generalització, estableix que qualsevol element de l'entitat genèrica ha d'estar també en alguna de les entitats especialitzades rep el distintiu de *completa*. O sigui, una especialització completa no permet que hi hagi elements en l'entitat de dalt que no apareguin en cap de les de baix. Això es pot indicar en el diagrama fent doble la línia de la connexió superior del triangle, que vindria a dir que la participació de l'entitat genèrica en el conjunt de les especialitzades és total. Hi ha certa analogia amb els llenguatges de programació orientats a objectes on l'entitat genèrica es diria classe abstracta.

Especialitzacions o generalitzacions disjuntes

Que una especialització, o generalització, sigui completa no impedeix que un element de l'entitat genèrica pugui estar en totes les especialitzades. És a dir, és possible que hi hagi un taxista bomber, pel cas de l'exemple. Quan es vol impedir que un mateix element de l'entitat superior aparegui en més d'una de les entitats inferiors, llavors es diu que l'especialització, o generalització, és *disjunta*, i s'indica en el diagrama amb el terme "disjunta" tocant a la línia de la connexió superior del triangle, és a dir sota la dreta de l'entitat genèrica.

4.14.2 Agregació d'Entitats

Al llarg de tot aquest capítol s'ha insistit repetidament que la diferència entre una entitat i una relació M:N és l'existència o no d'un atribut clau que n'identifiqui cada element. Els atributs clau no només tenen valors diferents per cada element d'una entitat, sinó que precisament perquè identifiquen aquests elements, també són utilitzats en les relacions on participa l'entitat per ser aparellats amb elements clau d'entitats relacionades. És el que en programació orientada a objectes es declararia com a variable membre pública d'una classe. El que es coneix des de fora. Això vol dir que l'essència que realment caracteritza una entitat és la capacitat de ser relacionada amb altres entitats. Una relació que relacioni relacions és un concepte absolutament prohibit, que tomba qualsevol disseny. No pot ser.

Malgrat tot, a voltes resulta que ens convindria fer alguna cosa com una relació entre relacions. Bàsicament, la nostra intenció és poder relacionar qualsevol element d'una relació amb alguna altra cosa. I com que això no pot ser, precisament perquè les relacions no tenen clau, existeix el concepte d'agregació. Fer una agregació és introduir una clau en una relació per transformar-la en entitat, per la qual cosa convé canviar-li el nom, del verb que pogués tenir al substantiu propi de les entitats. Això es fa en relacions M:N i relacions ternàries.

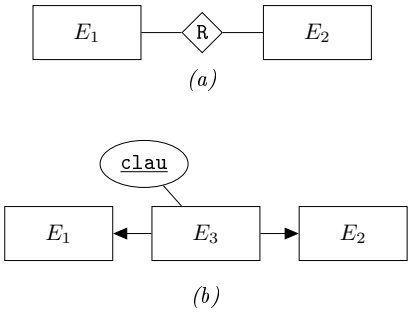


Figura 4.22: *Agregació d'una relació. (a) Relació M:N. (b) Entitat agregada.*

Les agregacions no tenen cap símbol per representar-se en un model ER, encara que de totes maneres, els dissenyadors experimentats adquireixen la capacitat de reconèixer-les.

En la Figura 4.22 es pot veure com es fa una agregació pel cas d'una relació binària M:N. En la part superior hi ha la relació abans de ser agregada. És a dir, abans de descobrir la necessitat de relacionar-la amb altres entitats del model. En la inferior s'ha transformat la relació `R` en una nova entitat E_3 afegint-hi una clau primària.

En definifiva, fer una agregació vol dir transformar una relació M:N en una entitat en dues relacions 1:N, en les quals la nova entitat és al costat N de les dues.

En aquest capítol s'ha introduït tot un llenguatge gràfic que serveix per modelar dissenys de bases de dades. Els conceptes primordials d'aquest llenguatge són el d'entitat i el de relació. Una entitat ve caracteritzada per un col·lecció d'atributs que li donen estructura entre els quals n'hi ha que formen la clau. Les entitats tenen clau. Les relacions no. Una relació ve caracteritzada per les agrupacions, siguin parelles o trios, de claus de les diferents entitats que relaciona.

Capítol 5

Àlgebra Relacional

Com diu el preàmbul, reposar continguts teòrics en el mètode matemàtic fa més robustos aquests continguts. Bé, doncs l'àlgebra relacional és una teoria matemàtica que va néixer juntament amb les bases de dades, i en regeix el seu comportament.

El pròposit d'aquest capítol és adquirir la capacitat d'actuar, amb paper i llapis, de la mateixa manera que ho fa un SGBD de manera automàtica. Això vol dir saber descriure les dades, i saber calcular amb elles.

Aquest capítol comença amb una introducció a la noció d'àlgebra en la Secció 5.1. Després, malgrat formar part d'un mateix contingut teòric, l'àlgebra relacional es fragmenta en dues parts. Primer, en la Secció 5.2, la que té que veure amb l'espai de dades, o sigui els objectes, que a partir d'aquí en endavant seran les relacions. I, abans d'entrar en la segona, es descriu la transformació del disseny de la base de dades pel club esportiu al seu model relacional en la Secció 5.3. És a dir, tanquem definitivament tot al que afecta a l'espai abans d'entrar en les operacions de l'àlgebra, en la Secció 5.4. Les operacions viuen al món del temps.

Amb tot, el que en aquest capítol es presenta són unes nocions bàsiques de l'àlgebra relacional, però no és el propòsit aquí extendre's massa en la direcció més teòrica. Particularment, es troba a molt faltar la part de formes normals de les relacions.

Per qui estigui interessat en els fonaments i desitgi aprofundir-hi, hi ha la refèrencia ineludible de Silberschatz, [10]. Tant la sintaxi com tot el que afecta la teoria d'àlgebra relacional d'aquest capítol va ser introduït en aquesta obra.

5.1 Àlgebres

En el seu sentit més genèric l'Àlgebra estudia els comportaments i el potencial expressiu d'unes estructures en les que bàsicament hi ha definits uns valors, així com unes operacions internes per treballar amb aquests valors.

Una operació és *interna* a un conjunt de valors quan tant els operands d'entrada com el valor resultant de sortida de l'operació pertanyen al conjunt. Hi ha molts exemples, com ara la suma, que és una operació interna pels números naturals, i també pels números parells, però no pels números senars. La comparació, en canvi, no és una operació interna ni pels naturals ni pels enters, ni per cap conjunt de nombres, perquè encara que d'entrada li donem els nombres que sí que són del conjunt en qüestió, ens retorna un valor lògic, és a dir cert o fals, cosa que no és un nombre.

L'exemple més concís del que és una àlgebra el tenim a l'Àlgebra de Boole, on els valors possibles són el zero i l'u, i les operacions la negació, el producte i la suma lògica, en la qual u més u, fa u. És a dir, la suma, en l'àlgebra booleana es defineix com la unió de conjunts de la Secció 1.2.1, o l'operació disjuntiva del càlcul de predicats que s'ha vist a la Secció 1.3.1.

En l'àlgebra relacional, en canvi, els valors enlloc del zero i l'u, són relacions, cosa que les converteix en el concepte central de la teoria.

5.2 Relacions

Un dels aspectes que segurament introdueix més confusió a l'hora d'adquirir la terminologia pròpia de les bases de dades és el canvi semàntic que va sofrint el terme *relació* al llarg dels diferents capítols. En el Capítols 3 i 4, s'ha utilitzat bàsicament com un arc del graf del model ER, tot i que també, per cardinalitats complexes, s'ha vist relacions que generen nodes.

Aquí és diferent. Dins l'àlgebra relacional reprenem el concepte de conjunt vist en el Capítol 1 per les relacions. Les relacions són conjunts, o si voleu llistes, per imaginar una cosa més concreta. I recordeu que ja en la Secció 1.1 s'insisteix en que per definició un conjunt no té elements repetits. És a dir, en l'àlgebra relacional per definició les relacions no tenen tuples iguals.

El nucli d'una base de dades és un conjunt de relacions. Allò més important i més independent de qualsevol altra cosa. Les bases de dades comencen declarant el conjunt de relacions que les constitueixen.

Per les persones més joves pot resultar sorprenent sentir utilitzar la paraula relació com a sinònim de llista. El primer sentit que li donem, d'infants o

adolescents, és el de que una persona està relacionada amb el lloc on viu, o els germans estan relacionats perquè són fills d'una mateixa mare... una semàntica propera al concepte de dependència. Després, arriba un dia que sentim algú dient que té la relació de totes les persones d'un dinar o d'una reunió. I pensem, com?, una relació també és una llista?. Quan vas veient que efectivament el terme té un sentit que fins llavors no havies descobert, et preguntes si realment és un sentit nou, o l'antic era un concepte prou genèric com per incorporar aquest nou ús del terme. Perquè clar, les coses que surten en una llista estan relacionades pel sol fet d'estar a la llista...

De totes maneres, en els diccionaris es diferencien aquestes dues accepcions. Mirant [8] el sentit de llista apareix en el segon lloc. L'entrada té moltes accepcions i usos. Les tres primeres diuen així,

> **relació** [*s. XIV; del ll. relatio, -ōnis, íd.*]
>
> *f* **1** *1 Acció de relatar o referir allò que hom ha sentit, ha vist, etc. Una relació detallada dels fets. Relació oral, escrita. (...)*
> **2** *Llista o enumeració de noms, adreces o altres indicacions especialment ordenats.*
> **3** *1 Lligam, referència, connexió, que hom percep o imagina entre dues o més coses. Una relació de semblança. La filosofia i la teologia tenen una relació molt estreta. Entre aquests fets no hi ha cap relació.*

...i segueix amb varis usos de diferents àmbits.

Respecte aquesta entrada del diccionari, sembla que la primera accepció sigui un cas particular de la segona. En qualsevol cas, la primera és útil pel fet de mostrar que la rel de relació és la mateixa que la de relatar. Interessant. Respecte les altres dues, clarament la segona és la que permet considerar-la un sinònim de llista, i és l'ús que se'n fa en l'àlgebra relacional. I observeu que curiosament, la tercera accepció és la que utilitzàvem en el capítol de disseny amb els models ER.

Bé. Deixem enrere el circumloqui sobre les relacions per posar els punts sobre les is. A partir d'aquí, d'aquest capítol en endavant, ens trobarem en un entorn més categòric que en capítols anteriors. Per això cal descriure amb rigor la definició de relació.

5.2.1 Relació

Una relació és un conjunt d'uns elements que anomenem *tuples*. Un tupla és un element amb un nombre fix d'*atributs*. El valor d'una tupla en un atribut és una dada, o una informació elemental.

El terme tupla, no s'utilitza massa a la pràctica. Normalment es parla de registres o files, però no de tuples. Tot i així, té una raó de ser. Una tupla marca diferències amb un registre en el sentit que un registre té un enfoc més tecnològic. Més físic. I el mot fila, que també fa que tupla no s'utilitzi, té una definició que fa referència a la relació a la qual pertany. Així doncs, la paraula tupla ens ha de servir per la part estrictament formal de l'àlgebra. El subjecte al que pretenem referir-nos és un element de la relació entesa com a conjunt. Això permet el desenvolupament de l'àlgebra relacional, i més encara del càlcul relacional per tuples.

Així doncs, una relació es pot representar com una matriu de dades. Una matriu plana, o sigui de dues dimensions. Files per columnes, vaja. A les files, que són els elements del conjunt, se les anomena tuples dins l'àmbit de l'àlgebra relacional, i registres dins l'àmbit de la programació de les bases de dades. A les columnes se les anomena atributs o camps, tant dins l'àmbit de l'àlgebra relacional com dins la programació.

En la Figura 5.1 es mostra la terminologia pròpia de l'àlgebra relacional.

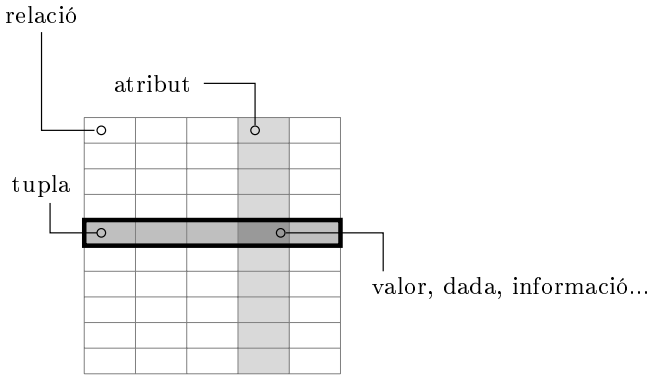


Figura 5.1: *Una relació és un conjunt de tuples.*

Tuples

Una tupla és una agrupació de valors de tipus heterogenis definits en un ordre concret, cosa que permet que associem aquesta noció a les estructures de dades `RECORD` del fortran, o `struct` del llenguatge c. En java, en canvi, una tupla vindria a ser una classe sense cap mètode. És a dir, un seqüència d'atributs de diversos tipus.

Regla 1 (de les bases de dades relacionals)
Les tuples d'una relació tenen un número fix d'atributs, cada un dels quals té una mida màxima.

Un tupla és un element amb un nombre fix d'atributs. Insistim, perquè tot arrenca d'aquí. Un nombre fix perquè puguem moure grans quantitats de tuples sota control rigorós. Un nombre fix per poder accedir ràpidament a l'enèsima tupla multiplicant n per la mida de la tupla. Un nombre fix perquè malgrat els humans representem les dades en superfícies planes de dues dimensions, a l'hora d'afegir i treure dades, resulta més senzill utilitzar-ne només una.

És tal la transcendència d'aquesta limitació, que condiciona la teoria completa de les bases de dades relacionals, àdhuc l'àlgebra relacional que aquí ara encetem.

El nombre de columnes és fix. En canvi, el nombre de files pot variar molt sovint, ja que els elements del conjunt que és la relació són les files. I recordeu de la Secció 1.2 que podem fer unions i diferències per afegir i treure elements dels conjunts.

Donem una volta més a la qüestió, des d'una visió més tecnològica. Tenim un conjunt de registres, o files, que tots tenen la mateixa amplada. Això ho fem així perquè ens resulti més fàcil poder accedir-hi. Com ja s'ha dit, si cada fila ocupa un espai fix, llavors podem accedir ràpidament a la fila n multiplicant n per la mida d'una fila. I és una prioritat poder accedir ràpidament als registres.

Això resulta interessant. Per un costat, entre línies es pot llegir una regla que sembla natural. S'està dient que els arxius amb registres de longitud variable no poden tenir accés directe. I no és veritat. Existeixen arxius amb longitud de registre variable i accés directe, els fitxers indexats o calculats. Però suportar els SGBDs en aquests tipus de fitxers complicaria excessivament la gestió de les dades. En fi, sembla raonable associar registres de longitud variable amb accés seqüencial, com els xml.

I per altra banda, també entre línies es torna a parlar d'allò de les banderes lineals i les banderes planes de la introducció del Capítol 3. Allà es deia que podem fer les banderes lineals tan llargues com volguem, i les planes no. I aquí en tenim la prova. Fixeu-vos-hi bé. Com s'ha dit més amunt, guardem la informació en taules de dues dimensions, però l'estructurem de tal manera que tan sols li permetem créixer indefinidament en una de les dues dimensions. És com les banderes. Una relació és com una bandera lineal, en vertical.

Bé, ara anem al detall. Fins aquí hauria d'estar clar que el nombre de dades que té una relació és el nombre d'atributs multiplicat pel nombre de tuples. Per poder treballar amb aquestes dades, primer de tot cal identificar cada cel·la de la matriu. I per identificar una cel·la cal donar columna i fila.

La identificació de les columnes és fàcil. S'anomena amb un mot diferent a cada una i ja està, identificades. Ergo, cada relació que és una matriu, a més de tenir un nom com a tal, també ha de tenir un nom diferent per cada columna. Noteu que una de les dues dimensions que té una relació, l'horitzontal, s'estableix en el moment de la definició de la relació. En canvi, el nombre de

files és variable, sempre es poden afegir o treure tuples d'una relació. Per les files, doncs, com que no sabem quins nous valors poden venir, l'única cosa que podem fer és establir una restricció que forci que, al menys en el valors d'algunes columnes concretes, totes les files tinguin valors o combinacions de valors diferents.

De la capacitat d'establir aquesta restricció neix la primera diferència entre un SGBD i un sistema d'arxius d'un sistema operatiu. Tenir coneixement del contingut de les dades és la primera característica distintiva dels SGBDs, cosa que cal per poder dictar una restricció que impedeixi la repetició de segons quins continguts.

En síntesi, observeu que la identificació de les files és un ordre d'abstracció més complicat que la de les columnes, ja que, en files, l'única cosa que podem fer en el moment de la creació de la relació és establir que per algun atribut concret totes les tuples que s'afegeixin a la relació pels segles dels segles siguin diferents. Això és una norma, un restricció, un llei... En canvi per les columnes és més fàcil. Som nosaltres, els dissenyadors en el moment de la creació, que establim els noms. I som nosaltres que sabem que han de ser tots diferents.

Càlcul relacional per tuples

Del càlcul relacional per tuples no en parlarem massa. Simplement enunciem que és una eina alternativa a l'àlgebra relacional, una mica més teòrica, i que el que diferencia els dos mètodes és que, així com l'àlgebra relacional explica el procediment que cal seguir a partir de les dades inicials per obtenir els resultats, el càlcul relacional per tuples és limita a definir els conjunts de tuples que es pretenen aconseguir en el resultat per mitjà de la lògica de predicats vista en la Secció 1.3. Aquests predicats s'expressen a partir dels atributs de les tuples, constants, i operadors de comparació. De fet, ja es veu que el càlcul relacional per tuples és més abstracte, ja que es limita a tractar les tuples com elements dels conjunts que són les relacions. L'àlgebra relacional és més propera a la computació, ja que explica el procediment en la seva formulació.

Dominis

Un domini és el conjunt de valors que pot prendre un atribut.

Noteu que els dominis no depenen de res. En cert sentit, matitzem aquí allò dit en el tercer paràgraf d'aquesta mateixa Secció 5.2. Allà es diu que les bases de dades comencen declarant les relacions que contenen, i per ser més fins, s'hauria de dir que abans de crear una relació s'han de crear els dominis dels seus atributs. Això no obstant, en la major part dels casos les relacions usen dominis predefinits.

De tota manera, queda clar que un domini no depen de ningú, i una relació depèn dels seus dominis. Això converteix la definició de dominis, quan cal, en la primera tasca de la definició d'una base de dades.

La inferència del domini a partir d'un atribut donat no és una qüestió massa estricta, ja que un mateix atribut pot tenir dominis més o menys restrictius. Un nom de persona pot tenir per domini les cadenes de caràcters, o bé pot definir-se restringit als noms que figurin a una relació concreta. Igualment, l'edat de les persones pot implentar-se en un enter, però també en un enter restringit a ser inferior a dos cents.

Els tipus de dades primitius de qualsevol llenguatge de programació serveixen per implementar qualsevol domini. I a partir d'aquí, establir-hi restriccions. El valor especial *nul* pertany a tots els dominis, i significa absència de valor.

Estrictament, dins l'àlgebra relacional, els tipus dels atributs no ens afecten. Aquí es parteix de l'existència de conjunts de valors que anomenem dominis, D_i per $i = 1, ..., n$, sent n el nombre d'atributs de la relació. Cada domini conté els valors possibles que pot prendre cada atribut. Així doncs, si diem A_i per $i = 1, ..., n$ als atributs d'una relació, llavors $A_i \subseteq D_i$, per $i = 1, \ldots, n$. Ens podem referir a un atribut com al conjunt de valors que pren en les tuples de la relació.

Definició formal de relació

Arribats aquest punt, se suposa que es té una idea ben clara del que significa una relació en l'estructura d'una base de dades. Ara bé, per poder desenvolupar una àlgebra cal rigor en la definició. I tot plegat ens porta a establir una definició de relació que se suporta única i exclusivament en la definició de conjunt.

La definició formal d'una relació utilitza la noció d'esquema de la relació.

L'*esquema d'una relació* en l'àlgebra relacional és la llista ordenada dels noms dels seus atributs, encara que sovint posem el nom de la relació abans de la llista i a l'expressió completa li diem també esquema de la relació. Com que la llista d'atributs se suposa ordenada, ha d'anar entre parèntesis, tal com mostra la Caixa 5.1.

$$r(A_1, A_2, \ldots, A_n)$$

Caixa 5.1. *Esquema d'una relació en l'àlgebra relacional.*

Encara que siguin conjunts, a les relacions les designarem amb minúscules.

Els esquemes de relació són fonamentals per comprendre les operacions de l'àlgebra relacional. Encara que sembli mentida, una relació és a un número enter el mateix que els atributs de la relació a cada una de les xifres. Aquesta percepció tan desconcertant es consolidarà més endavant amb l'operació del producte cartesià.

A la Caixa 5.2 hi ha un exemple d'esquema de relació.

```
persona(ciutat,cognom,nom,passaport)
```

Caixa 5.2. *Exemple d'esquema de relació en l'àlgebra relacional.*

Estrictament, una relació és un subconjunt del producte cartesià entre els dominis dels seus atributs. No és una definició complicada. És una manera de parlar. Per això, veiem-ho a poc a poc.

Probablement recordeu que en l'àmbit de l'anàlisi matemàtica és notòria la presència de les funcions de variables reals. Tothom sap que en un espai euclidi podem representar la funció x^2, que és la típica paràbola. La Figura 5.2 mostra la corba $f(x) = x^2$.

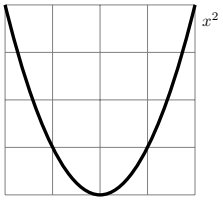


Figura 5.2: *Representació de la paràbola que descriu la funció* $f(x) = x^2$.

Doncs bé. Podem definir cada funció de variable real com un subconjunt de punts del pla, o el que és el mateix, com un subconjunt del pla. És a dir, dient $\mathbb{R}$ al conjunt dels nombres reals, podem fer referència a la corba de la Figura 5.2 com al subconjunt $\{(x, y) \in \mathbb{R} \times \mathbb{R} \mid y = x^2\}$. És extrany, no?

Per altra banda, l'anàlisi matemàtica determina que una funció real de variable real com la de la Figura 5.2 no pot tenir més d'una resposta per una x determinada, condició que la paràbola satisfà per descomptat. Això vol dir que la funció representada en el pla només pot tallar una línia vertical en un sol punt, cosa que es pot comprovar desplaçant horitzontalment un foli sobre la Figura 5.2. De fet, per representar una circumferència cal definir dues funcions en el pla. Una per la meitat de sobre, i l'altra per la de sota.

Però en canvi, si en rigor acceptéssim que per definició una funció real de variable real és un subconjunt de punts del pla, la restricció de que en vertical

només podem tallar la corba en un punt no quedaria plasmada. I per tant, aquesta definició pel concepte de funció real de variable real és insuficient. O sigui, podem definir cada funció amb un subconjunt del pla, però no la definició de funció com a concepte genèric, que cal fer d'alguna altra manera.

Bé, en qualsevol cas, portar a escena la definició de funció com a subconjunt de punts del pla es justifica perquè la definició formal de relació, amb tot el rigor, té un aspecte molt semblant, tal com es pot veure a la Caixa 5.3.

Donats els n dominis $D_1, D_2, \ldots, D_n$,

diem que $r = r(A_1, A_2, \ldots, A_n)$ és una relació

$$\Leftrightarrow$$

$$r \subset D_1 \times D_2 \times \cdots \times D_n,$$

sent $A_i \subseteq D_i$, per $i = 1, \ldots, n$.

Caixa 5.3. *Definició formal d'una relació.*

Dit en paraules, una relació és un subconjunt del producte cartesià dels dominis dels atributs que la composen.

Noteu que per $n = 1$ i D_1 igual al conjunt de tots els números, la definició contempla que una relació sigui tan sols un número.

Compatibilitat entre relacions

La compatibilitat entre relacions fa referència a la compatibilitat dels seus esquemes. A la Secció 1.1 és diu que dos conjunts són compatibles si un d'ells és subconjunt de l'altre, i no necessàriament subconjunt propi. O sigui, poden ser iguals. La compatibilitat es denota amb el símbol $X \sim Y$. És a dir $X \sim Y \Leftrightarrow X \subseteq Y \vee Y \subseteq X$.

Els esquemes de dues relacions són compatibles si prenent els atributs per parelles segons el seu ordre, els dominis corresponents són compatibles.

La idea de compatibilitat de relacions s'espressa formalment en la Caixa 5.4.

Donades dues relacions,

$r = r(A_1, A_2, \ldots, A_n)$, sent $A_i \subseteq D_i^A$ per $i = 1, \ldots, n$

$s = s(B_1, B_2, \ldots, B_m)$, sent $B_j \subseteq D_j^B$ per $j = 1, \ldots, m$

diem que r i s són compatibles

$\Leftrightarrow$

$n = m$, i $D_i^A \sim D_i^B$, per $i = 1, \ldots, n$.

Caixa 5.4. *Definició de compatibilitat entre les relacions r i s.*

5.2.2 Visió Cartesiana d'una Relació

Les coordenades cartesianes són una bona eina per representar conceptes ortogonals. Ortogonal vol dir perpendicular. La rel essencial de la seva utilitat tan estesa està en el fet que horitzontal és perpendicular a vertical. O sigui, moventnos en horitzontal, mai de la vida pujarem ni baixarem. O sigui, l'alçada en la que estem no depèn de quin moviment horitzontal fem, de tots els possibles. Ortogonal o perpendicular, doncs, també vol dir independent.

I això és important. Podem representar conceptes independents en diferents eixos de coordenades cartesianes, ja que la perpendicularitat dels eixos reflecteix la independència dels conceptes.

Les coordenades cartesianes tenen multitud d'usos, a part del de representar funcions. I tot seguit se'n mostra un de molt interessant.

En estadística s'utilitzen els diagrames bivariants per representar valors de variables aleatòries que en principi són independents. Per exemple, el nivell d'anglès i l'edat d'una població.

La hipòtesis que ens plantegem és,

Hi ha alguna dependència entre el nivell d'anglès i l'edat d'una població?

Llavors prenem una mostra de 400 persones seleccionades a l'atzar i els hi preguntem quina edat i nivell d'anglès tenen amb un examen que puntuem del 0 al 100.

Tot plegat ho representem com en el Diagrama 5.1, en el qual cada persona de la mostra està representada amb un punt. Observeu que en aquest moment, fent aquesta representació, estem fent ús de la suposada independència entre les dues variables. Si finalment resultés que efectivament una variable impactés

a l'hora d'explicar l'altra, en el diagrama apareixerien distribucions de punts reconeixibles. En el cas més senzill podria ser una distribució lineal.

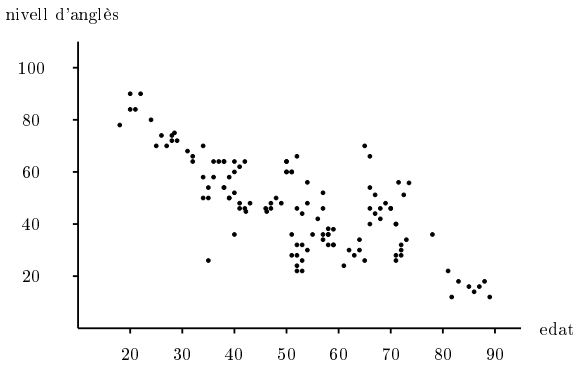


Diagrama 5.1. *Diagrama bivariant com exemple d'ús de les coordenades cartesianes.*

Si ara diem x a l'edat, i y al nivell d'anglès, podríem buscar la recta $y = mx+n$ que lligui les dues variables. Això és trobar les m i n tals que minimitzin la desviació de les y_i's estimades, que valen $mx_i + n$, menys les y_i's observades. O sigui, resoldre

$$\min_{m,n} \sum_{i=1}^{400} (mx_i + n - y_i)^2.$$

En fi, a partir de regressions com aquesta, i no només amb comportaments lineals sinó amb qualssevol que plantegi una hipòtesi, podem explotar el contingut d'una base de dades inferint nous coneixements, tasca que fa la mineria de dades usant mètodes estadístics, vegeu [9, 4].

Molt bé. L'exemple d'ús de coordenades cartesianes s'ha mostrat per un cas bidimensional, és a dir, a partir de punts del pla cartesià. I a més, per dues variables quantitatives, és a dir contínues, cosa que fa que la representació sigui més consistent perquè qualsevol punt del pla té una interpretació real.

De totes maneres, la idea d'independència entre les variables no implica que hagin de ser quantitatives. I si enlloc de qualsevol punt del pla només tingués sentit els punts pertanyents a la malla de nombres enters, la manera de representar la informació seguiria sent útil. Per tant, igual que en dues dimensions podem representar-ne tres en l'espai. I per una relació sense camps numèrics, podem utilitzar l'ordre alfabètic, com en l'exemple que es mostra tot seguit.

En la Secció 5.2.1 s'ha vist que per definir una relació calen prèviament els dominis. La definició de la relació r s'exposa a la Caixa 5.5.

Donats els tres dominis

D_1 = {A,B,...,Z} majúscules de l'alfabet anglès.
D_2 = {1,2,...,10} enters entre u i deu.
D_3 = $\{\alpha, \beta, \ldots, \omega\}$ minúscules de l'alfabet grec.

definim la relació

$$r = r(A_1, A_2, A_3),$$

sent $A_i \subseteq D_i$, per $i = 1, 2, 3$.

Caixa 5.5. *Definició de la relació per la visió cartesiana.*

En la Figura 5.3 es presenta una instància per aquesta relació. Una instància d'una relació és el contingut que pot tenir en un instant, però també es pot entendre com una traducció de l'anglès *instance*, que vol dir exemple.

En la Figura 5.3*(a)* s'hi ha afegit una columna amb un to més clar amb els noms de les tuples. Aquesta columna no forma part de la relació, d'aquí el to clar. Tan sols es mostra per facilitar la identificació en el diagrama de la dreta.

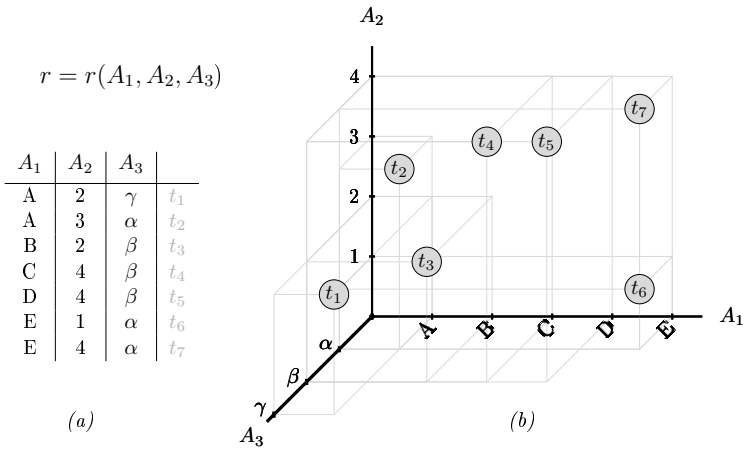


Figura 5.3: *Instància de la relació de la Caixa 5.5: (a) Descripció tabular. (b) Descripció cartesiana.*

Observeu-ho bé. A partir de les tuples, que són els punts etiquetats resseguiu per les línies fins arribar als eixos per comprovar que conté la mateixa informació que la relació.

Representacions geomètriques com la de la Figura 5.3 obren possibilitats a l'hora d'operar amb relacions. Això és perquè l'analogia entre la taula i la

visió cartesiana és robusta. Tant, que a partir d'aquí disposem d'una eina amb la qual reproduir gràficament cada operació sobre relacions que definim en l'àlgebra relacional, cosa força interessant.

Per altra banda, la limitació de no poder representar més de tres columnes d'una relació no ha de suposar cap problema. Podem comprendre una distribució espaial de les dades tinguin el nombre de components que tinguin.

5.3 Model Relacional d'una Base de Dades

El model relacional d'una base de dades és la descripció textual més concisa de l'estructura d'una base de dades. La descripció d'una base de dades en un model relacional és un punt d'articulació en el projecte. Això vol dir que el camí que ens porta de la idea inicial a la funcionalitat final ha de passar necessàriament per aquest estat. Es tracta del primer enfoc automàtic del projecte. Comencem a tocar de peus a terra. Arribats aquest punt, cal saber fer-se una idea precisa de com resoldre cada una de les sol·licituds de la definició de requeriments inicial mirant el model ER.

Venim del model ER, i anem cap a la implementació en llenguatge estructurat de consultes. És a dir, igual com un model ER s'implementa en un model relacional en la transició de la primera a la segona etapa del projecte, el que es descriu en aquest capítol serveix d'origen a partir del qual implementarem la base de dades en una segona transició que anirà del model relacional a l'SQL.

Tant el que eren entitats com relacions M:N en el model ER, aquí són relacions. Tancada la fase del disseny, deixem enrera el terme entitat, i canviem el sentit del terme relació.

El model relacional d'una base de dades és, senzillament, la llista dels esquemes de les seves relacions.

No s'ha de confondre amb un diagrama d'esquemes, que és un concepte ideat molts anys més tard, i aquí es mostra en la Secció 5.3.5. Aquesta confusió és impressionantment freqüent. A molts alumnes els hi costa entendre que un model sigui una llista textual. Deu ser perquè els estudiants associen model a diagrama, quan és un concepte infinitament més ampli.

El model relacional contribueix al desenvolupament del projecte en tres aspectes molt importants.

- Estableix ordre entre els atributs de les entitats.
- Transforma tant les entitats com les relacions del model ER en un únic concepte de relació del model relacional.

- Incorpora un ordre en la col·lecció de relacions que constitueixen la base de dades.

Això vol dir que a part d'establir un ordre en els atributs de les entitats, transforma les fletxes en alguna cosa textual, per una banda, i que seqüencialitza el diagrama per una altra.

Escriurem els noms de les relacions d'un model relacional en minúscules, com es diu a l'Apèndix A. Això ens ajudarà per distingir-les de les entitats del model ER on han estat originades.

Si us fixeu en l'esquema de la Caixa 5.2 de la pàgina 100, observareu que no s'hi expressa la condició de clau per l'atribut `passaport`. Això és perquè la definició d'esquema d'una relació pertany a l'àlgebra relacional, i no al model relacional. Dins l'àlgebra, les tuples de les relacions són diferents entre elles per definició, i no en saben res de claus ni hi ha cap problema d'identificació. L'àlgebra relacional és una teoria matemàtica. El model relacional és la manera de posar en pràctica la teoria. Posem en pràctica la teoria per poder implementar bases de dades. És en aquesta materialització de l'àlgebra quan introduïm el concepte de clau.

5.3.1 Atributs estructurals i atributs descriptius

A partir del concepte de clau, dins el model relacional tenim que tant les claus primàries com les claus foranes són atributs *estructurals*. Els atributs que no són estructurals són atributs *descriptius*. És interessant discernir entre aquests dos tipus d'atributs perquè l'única redundància que admetem en la base de dades és amb atributs estructurals. Aquesta redundància no només és admissible. És necessària perquè és el que efectivament implementa l'estructura, que es basa en repeticions de valors en diferents relacions.

Noteu que pel que fa a la integritat, no hi ha cap problema amb eliminar de la base de dades qualsevol atribut descriptiu. En canvi, segons quin atribut estructural s'elimini, s'ensorra la base de dades completa.

5.3.2 Claus Primàries

Establir una *restricció d'existència* per un atribut d'una relació vol dir no permetre afegir tuples a la relació si no tenen un valor en aquell atribut. O sigui, els atributs amb restricció d'existència han de tenir valors diferents de nul en totes les seves tuples. I es diuen atributs *requerits*, en el benentès que el que és requerit és el valor en cada tupla de la relació per aquest atribut.

Establir una *restricció d'unicitat* per un atribut d'una relació vol dir no permetre afegir tuples a la relació si ja existeix alguna tupla amb el mateix valor en l'atribut que el que s'està pretenent inserir. I és diuen atributs *únics*, en el benentès que el que és únic és el valor per aquest atribut entre totes les tuples de la relació.

Sobretot alerta. Que quedi clar que no hi ha redundància entre els dos conceptes. Restringir la unicitat no implica restringir l'existència. Això seria com considerar que dos valors nuls són el mateix. Aquest és un tema delicat que es veu en profunditat en la Secció 6.4.8. En qualsevol cas, pel que fa a les restriccions d'unicitat i existència, es considera que dos valors nuls són diferents. I per tant, es pot establir que en un atribut que admeti valors nuls, tots els valors diferents de nul siguin diferents entre ells.

Una clau primària, o principal, és un subconjunt dels atributs d'una relació sobre el qual s'estableixen les restriccions d'existència i d'unicitat amb la intenció d'identificar les tuples de la relació on viuen.

Per escriure un model relacional a partir d'un model ER, en els esquemes de les relacions posarem les claus primàries al principi de la llista d'atributs, subratllades, com en el Model 5.1. Fent això estem definint un ordre parcial entre els atributs.

```
persona(passaport,ciutat,cognom,nom)
```

Model 5.1. *Exemple d'esquema de relació.*

Quan se subratlla una clau formada per més d'un atribut, cal subratllar també les comes que els separen. És una sola línia que subratlla varis atributs. Això reflecteix que és una sola clau primària, com en el Model 5.2.

```
persona(nom,cognom,ciutat)
```

Model 5.2. *Exemple d'esquema de relació amb clau primària binomial.*

Un atribut pot ser requerit i únic, i no per això ser clau primària. Fa falta la intenció. Noteu que els adjectius primària o principal tenen certa connotació subjectiva, de criteri. Al dissenyador li cal sotmetre's al domini del problema a l'hora d'establir les claus primàries. Una submissió abjecta. L'usuari final de l'aplicació, que resideix en el domini del problema, sap millor que ningú el sistema que té per identificar els objectes del seu domini. Per això, quan el

dissenyador estableix les claus primàries de les entitats, ha de reflectir la manera com l'usuari identifica els elements, i en tot cas fer aflorar les inconsistències que pugui haver-hi en la identificació.

Mirant-ho des de la perspectiva de l'anàlisi matemàtica, s'entén que la clau primària és com una variable independent comuna a una col·lecció de funcions que són els altres atributs de l'entitat. Vist així, es pot entendre que es digui que el nom d'una persona depèn del número de passaport, encara que sembli extrany. Realment es vol dir que fem dependre el nom del número de passaport, cosa més entenedora. És interessant la perspectiva analítica perquè té continuïtat en tot allò que té que veure amb la implementació de les relacions del model ER.

En l'exemple de la Taula 5.1 es mostra una instància de l'entitat PERSONA que es veurà en el Model 5.3 de la pàgina 111, amb l'atribut mail ja afegit, que és el que s'anirà desenvolupant per l'aplicació del club esportiu introduïda en la Secció 2.6.1. En aquest punt, tan sols cal observar que efectivament els valors de la clau primària, `passaport`, són tots ells diferents.

passaport	nom	cognom	mail
27673812M	Carme	Peralta	carmep@ionos.cat
X3478937A	Carles	Sanàbria	carsan1994@gmail.com
47548338K	Anna	Sanàbria	annasanabria@gmail.com
45493393Z	Jesús	Hortesa	rexstat143@rediris.es
C00001549	Michael	Bros	mbros1989@aol.com
294394950	Klauss	Stallman	kstallman@dvw.tum.de

Taula 5.1: *Instància de l'entitat* PERSONA *de l'exemple del Model 5.3.*

5.3.3 Claus Foranes

En rigor, una clau forana, o externa, és un subconjunt d'atributs d'una relació el domini dels quals és el conjunt de valors que conté una clau primària d'alguna altra relació de la mateixa base de dades.

En paraules més prosaiques: Quins valors pot prendre aquest atribut? Els que hi hagi allà.

Les claus foranes constitueixen sens dubte el concepte més nucleic de la lògica de les bases de dades relacionals. Tenen la mateixa transcendència a nivell estructural que les mateixes relacions com a concepte primogeni. Són les fletxes del model ER implementades en una màquina.

És més encara, la raó per la que es defineixen claus primàries és perquè puguin ser apuntades des de claus foranes. I això, precisament, és el que diferencia una clau primària d'un simple atribut requerit i únic. Que la clau primària, a més de ser requerida i única, és apuntada des d'altres relacions.

Això representa un pas de gegant en la implementació de les relacions 1:N del model ER. És important.

Regla 2 (de les bases de dades relacionals)
Les claus foranes implementen les relacions 1:N d'un model ER.

Quan una relació apunta a una altra, és a dir, té una clau forana cap a l'altra, es diu que les dues columnes, clau primària de l'una i clau forana de l'altra, són columnes *vinculades*. És fortament recomanable que s'anomenin igual per poder treure el màxim profit de les operacions que més endavant es veuran.

Observeu que la Regla 2 és conseqüència immediata da la Regla 1, que deia que les tuples tenen un nombre fix de columnes, ja que gràcies a l'1 de les relacions 1:N no cal més que un atribut addicional a l'entitat del costat N per representar les relacions 1:N del model ER, i per tant, el nombre d'atributs segueix sent fix per totes les entitats. I no només això, sinó que jugant amb les restriccions que imposem a l'atribut que és clau forana podem traduir més d'un concepte dels vistos en el Capítol 4.

- Una restricció d'existència en una clau forana implementa una participació total en la relació del model ER.
- Una restricció d'unicitat en una clau forana implementa una relació 1:1 del model ER enlloc d'una 1:N.

Atenció al fet que l'entitat del costat 1 de la relació no resulta modificada en res per implementar una relació 1:N. És a dir, ni s'assabenta que algú l'està apuntant. Igual que una, podrien estar-la apuntant cinquanta relacions que això no modificaria la seva estructura.

En els esquemes de les relacions, les claus foranes es col·loquen al final de la llista d'atributs, i sempre s'anomenaran amb el mateix nom que la clau primària a la que apunten.

Refinem doncs l'ordre parcial dels atributs en els esquemes que hem començat a establir en la secció anterior. I així com a les claus primàries les subratllem, a les claus foranes no. Així doncs, l'aspecte de les claus foranes en l'esquema de la relació fa que es confonguin amb atributs descriptius. Efectivament però, es poden diferenciar perquè el nom de les claus foranes en una relació coincideix amb el nom d'alguna clau primària d'una altra relació de la base de dades.

Quan en una relació no hi ha clau primària, les claus foranes es posen al principi del seu esquema.

En la Figura 5.4 es pot observar la idea de la implementació d'una relació 1:N del model ER amb claus foranes en el model relacional.

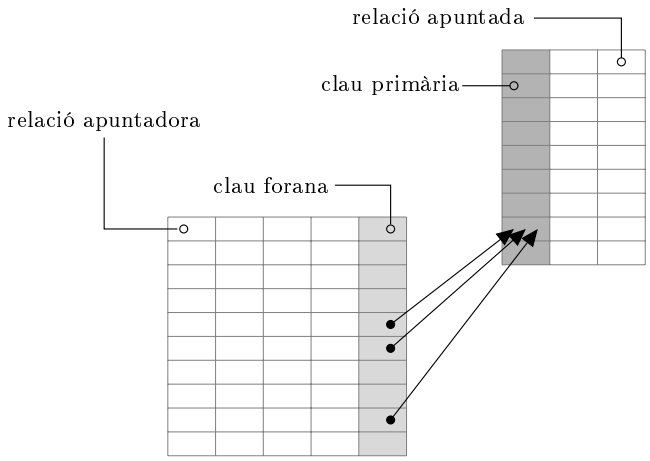


Figura 5.4: *Implementació amb claus foranes en el model relacional d'una relació 1:N del model ER .*

Tot plegat fa que la relació mostrada en la Taula 5.1 no fos completa. Hi faltava la clau forana que sí que es mostra en la Taula 5.2.

passaport	nom	cognom	mail	ciutat
27673812M	Carme	Peralta	carmep@ionos.cat	Cadaqués
X3478937A	Carles	Sanàbria	carsan1994@gmail.com	Badalona
47548338K	Anna	Sanàbria	annasanabria@gmail.com	Badalona
45493393Z	Jesús	Hortesa	rexstat143@rediris.es	Castelldefels
C00001549	Michael	Bros	mbros1989@aol.com	San Francisco
294394950	Klauss	Stallman	kstallman@dvw.tum.de	Berlín

Taula 5.2: *Instància completa de l'entitat* PERSONA *de l'exemple del Model 5.3.*

5.3.4 Transformació del Model ER al Model Relacional

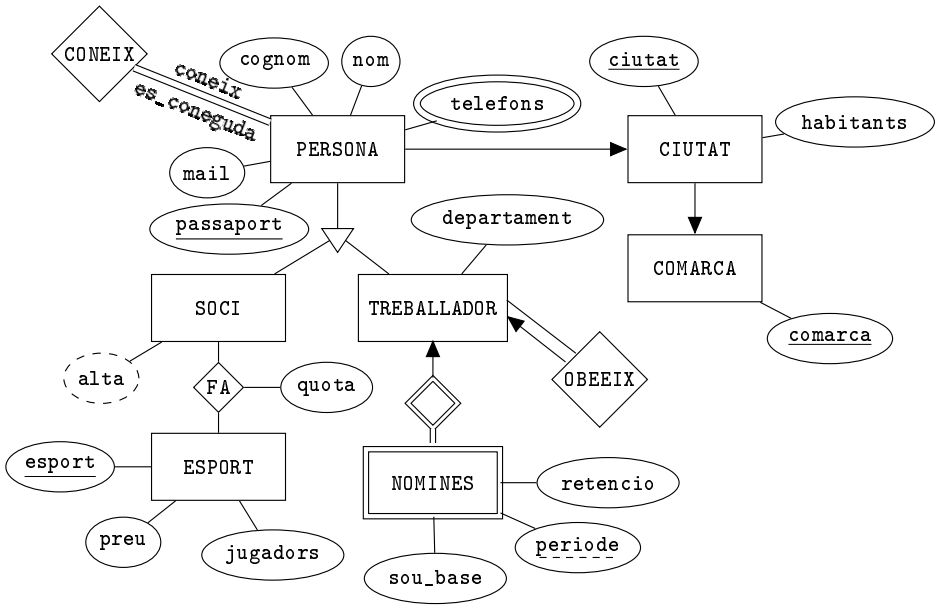
L'ordre que un model relacional imposa en la col·lecció d'esquemes de relacions que el composen, és un ordre parcial. Això és, a partir d'un mateix model ER, hi ha diferents ordenacions vàlides a l'hora de llistar les relacions del model relacional corresponent.

En termes estrictament tècnics, cal obtenir alguna ordenació topològica del graf dirigit que és el model ER. Una ordenació topològica d'un graf acíclic dirigit és una seqüènciació dels nodes tal que qualsevol predecessor aparegui en la seqüència abans que els seus successors. Per això, en general tenim diferents possibilitats per un mateix graf. I si això costa d'entendre, llavors només cal mantenir el principi de l'ordre parcial.

Només es pot apuntar a relacions que ja existeixin.

Això pel que fa l'ordre de declaració de les relacions. Mantenir el principi farà que abans definim les entitats que les relacions del model ER. I primer les entitats apuntades que les apuntadores. Per tant, és correcte començar per les entitats de suport.

El disseny del Model 5.3 és per la base de dades del club esportiu que s'ha introduït a la Secció 2.6.1. Gestiona les dades d'una sèrie de persones que són socis del club o bé treballadors. És necessari el número de passaport de les persones per poder-les donar d'alta a la base de dades. A més del nom, el cognom, i l'adreça de correu electrònic, de les persones també es guarda varis telèfons per cada una d'elles, en quina ciutat viuen, quants habitants té la ciutat, i la comarca on es trobi. Per altra banda, també és capaç d'establir amistats entre les persones que guarda. Les persones poden ser socis, o bé treballadors, o les dues coses a l'hora. Pels socis es guarda automàticament la data d'alta. I pels treballadors es guarda el departament, que pot ser *administració*, *comercial*, o *entrenador*.



Model 5.3. *Model ER per l'aplicació del club esportiu.*

Dels esports es coneix el nombre de jugadors necessaris per formar un equip i el preu mensual. De cada soci, es guarda quins esports practica i quina quota paga per cada un, que pot variar i preval respecte el preu. I per cada treballador, es guarda les dades del seu cap, així com l'historial de nòmines. Una nòmina ve identificada pel treballador a qui correspon més un període que conté el mes i l'any. Els valors possibles d'aquest atribut són dates. La base guarda la data de pagament de cada nòmina a cada treballador. A més, una nòmina també és un sou base i una retenció que és un percentatge.

Molt bé. Tot seguit es farà la transformació del disseny del Model 5.3 a un model relacional.

Comencem.

D'entrada, els esquemes de les entitats que siguin apuntades però no apuntin a cap altra entitat. Després, les que apuntin a les primeres, i així anar fent. Per tant, podem començar perfectament per la relació `comarca`, i després `ciutat`, tal com es mostra en el Model 5.4.

```
comarca(comarca)
ciutat(ciutat,habitants,comarca)
...
```

Model 5.4. *Primeres relacions de la versió relacional de l'exemple 5.3*

El fet que l'atribut `comarca` de la relació `ciutat` es digui com la clau primària d'una altra relació significa que és una clau forana que l'apunta. Per tant, les tuples de la relació `ciutat` han de tenir com a valors possibles de l'atribut `comarca` els que existeixin en la clau primària de la relació apuntada, és a dir, els valors de l'atribut `comarca` de la relació `comarca`.

Ara podem seguir amb la traducció de l'entitat PERSONA, ja que no apunta a cap relació que encara no haguem definit.

```
comarca(comarca)
ciutat(ciutat,habitants,comarca)
persona(passaport,nom,cognom,mail,ciutat)
...
```

Model 5.5. *Incorporació de la relació* `persona`.

Observeu en el Model 5.5 dues absències notables. L'autorelació CONEIX, que com que és M:N, provocarà una nova relació en el model relacional, i per tant no la tenim en compte en aquest moment. I per altra banda, els atributs multivalorats, com és el cas de `telefons` tampoc no es tenen en compte en l'esquema de la relació corresponent a l'entitat que els conté.

Bé, Un cop traduïda l'entitat PERSONA, procedim amb el que tan sols depèn d'ella. Primerament l'atribut multivalorat, que és més fàcil.

```
comarca(comarca)
ciutat(ciutat,habitants,comarca)
persona(passaport,nom,cognom,mail,ciutat)
telefons(passaport,telefon)
...
```

Model 5.6. *Transformació d'un atribut multivalorat.*

Les tuples de la relació `telefons` ens diran que tal persona té tal telèfon. El número de passaport d'una persona hi apareixerà tantes vegades com telèfons tingui.

Observant el Model 5.6 tenim la primera relació que apareix en un model relacional i no prové d'una entitat del model ER. Cosa nova. Per això no té clau primària, perquè no és una entitat. En canvi, sí que té una clau forana, `passaport` que pot prendre com a valors els números de passaport de les persones que hi hagi a la base de dades. Noteu que quan en una relació no hi ha clau primària, les claus foranes es posen al principi del seu esquema.

Com que no es permetrà que el mateix telèfon el tinguin dues persones, encara que en Model 5.6 no es mostri, caldrà una restricció d'unicitat per aquest atribut. A més, un registre en la taula `telefons` que no contingui un valor per la columna `telefon` no té sentit. Així doncs, `telefon` és un atribut requerit i no repetit, però no per això clau primària, ja que no s'apunta des d'enlloc.

La transformació de l'autorelació `CONEIX` segueix desenvolupant el model relacional com s'indica en el Model 5.7. Com en el cas anterior, la relació `coneix` tampoc inclou clau primària perquè no és una entitat.

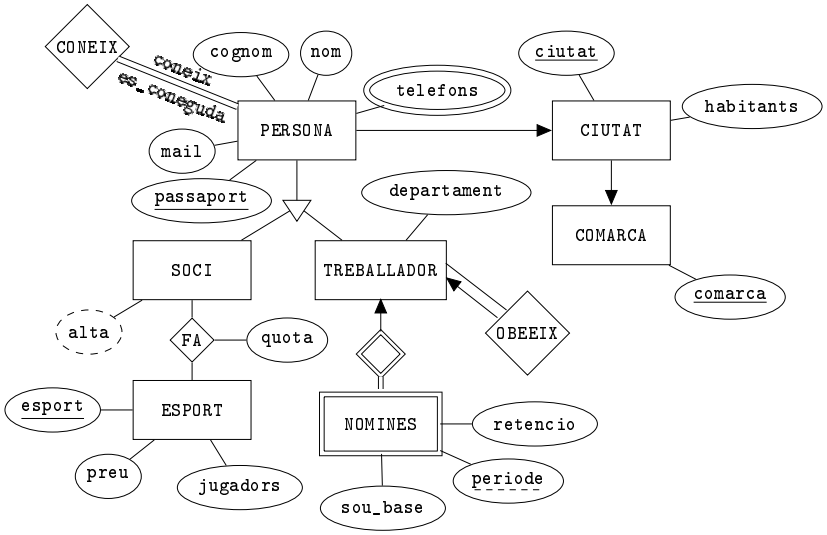
```
comarca(comarca)
ciutat(ciutat,habitants,comarca)
persona(passaport,nom,cognom,mail,ciutat)
telefons(passaport,telefon)
coneix(coneix,es_coneguda)
...
```

Model 5.7. *Transformació d'una autorelació M:N al model relacional.*

Atenció, les claus foranes de `coneix` violen la norma de que les claus foranes s'anomenen amb el nom de la clau primària de relació a la que apunten. Està justificat. Són rols. Per les autorelacions M:N farem aquesta excepció. Els dos atributs de la relació `coneix` són claus foranes que apunten a la relació `persona`. Cada tupla d'aquesta relació ens dirà tal persona coneix tal altra. I

per dir-ho utilitzarà els números de passaport. Un persona que aparegui molts cops a l'esquerra és una persona que coneix a molta gent. Una persona que no aparegui cap cop a la dreta vol dir que és una persona a qui no coneix ningú, pobra.

Arribats aquest punt, ja s'ha transformat bona part del Model 5.3, que es repeteix en el Model 5.8 per agilitzar el procediment en el que estem immersos.



Model 5.8. *Model ER pels exemples, repetició del 5.3.*

Ataquem l'especialització. Per transformar-la al model relacional, afegim a cadascuna de les entitats especialitzades una clau forana que apunti a la clau de l'entitat genèrica. A més, alerta, aquesta clau forana també és clau principal. Un fenòmen que no havíem vist fins ara, i que és la manera d'implementar una especialització o generalització. Total, que afegim les dues entitats com es pot veure en el Model 5.9.

```
comarca(comarca)
ciutat(ciutat,habitants,comarca)
persona(passaport,nom,cognom,mail,ciutat)
telefons(passaport,telefon)
coneix(coneix,es_coneguda)
soci(passaport,alta)
treballador(passaport,departament,obeeix)
...
```

Model 5.9. *Transformació d'entitats especialitzades i d'autorelació 1:N.*

Respecte les dues incorporacions del Model 5.9 se'n desprenen tres comentaris.

- Com es pot observar, que l'atribut `alta` de la relació `soci` sigui calculat no impacta en el model relacional. Es tracta com un atribut qualsevol. El fet de ser calculat, doncs, també ha de constar en la documentació del model relacional.
- Per altra banda, el fet que els atributs `passaport` de les relacions corresponents a les entitats especialitzades siguin claus primàries, a part de foranes, garantitza la unicitat d'aquests atributs en la relació, i per tant, tot i ser clau forana estan implementant relacions 1:1, com ha de ser, enlloc d'implementar relacions 1:N.
- També es notable l'aparició de la clau forana `obeeix`, que malgrat ser això, una clau forana, apunta a la mateixa relació `treballador`, i per tant contindrà el número de passaport del treballador que sigui el cap del que representa cada tupla. En aquest sentit, és gratificant que les autorelacions 1:N, igual que hem vist amb les M:N, s'implementen exactament igual que les relacions entre entitats diferents.

La relació FA és M:N. O sigui, com dues relacions 1:N. Per tant, apunta a SOCI i a ESPORT. Això fa que abans de descriure l'esquema de la relació FA haguem de descriure el de l'entitat ESPORT tal com diu el principi de l'ordre parcial. Només es pot apuntar a relacions existents. Total, que la transformació continua amb dues relacions més, tal com es mostra en el Model 5.10.

```
comarca(comarca)
ciutat(ciutat,habitants,comarca)
persona(passaport,nom,cognom,mail,ciutat)
telefons(passaport,telefon)
coneix(coneix,es_coneguda)
soci(passaport,alta)
treballador(passaport,departament,obeeix)
esport(esport,preu,jugadors)
fa(passaport,esport,quota)
...
```

Model 5.10. *Transformació de relacions M:N prèvia descripció de les entitats relacionades.*

Per definició de l'estructura, les dues claus foranes de la relació `fa` haurien de ser requerides, ja que no té sentit un element en aquesta relació si no apunta a una parella de valors existents. És normal que quan dos elements de dues entitats, entre les que hi ha una relació M:N, es relacionen, que no tingui sentit

la duplicitat de l'element de la relació. O sigui, la dada de que un soci fa un esport queda registrada només un cop. Això és especialment cert quan no hi ha atributs en la relació. Llavors, sovint es declara la parella de claus foranes de la relació `fa` com una clau primària binomial. Estrictament no és correcte, ja que tan sols es fa així per restringir la unicitat i l'existència. I per fer això no cal declarar-los com a clau primària. Per altra banda, com que fent-ho així no és menys eficient ni ocupa més espai, també és admissible. Però llavors, cal tenir molt clar que s'està fent aquest ús alternatiu del concepte de clau primària. L'ús genuí de les claus primàries és el de ser apuntades.

La relació `esport` no requereix cap comentari addicional. Noteu que al no apuntar cap altra relació en el diagrama s'hagués pogut posar al principi del model relacional sense cap problema. La relació `fa` ens vé a dir, per cada tupla, el soci tal paga tal quota per tal esport. A partir d'això suposarem que el soci practica l'esport, encara que en la realitat sigui tan diferent. No té clau principal perquè no prové de cap entitat, i conté dues claus foranes a soci i esport.

Atenció al tema de les participacions totals. Així com implementar una participació total d'una entitat en una relació 1:N és tan senzill com fer requerida la clau forana, pel cas de les M:N no és tan fàcil. L'única manera que tenim de garantir una participació total d'una entitat en una relació M:N és per mitjà de lògica. O sigui, afegint programes especialitzats en la base de dades que no permetin inserir un element en l'entitat si no s'insereix també algun element a la relació que l'involucri.

Finalment afegim l'entitat feble NOMINES. El fet que l'entitat identificadora sigui TREBALLADOR vol dir que no podem parlar d'una nòmina si no sabem de quin treballador és. I el discriminant, `periode`, servirà per distingir entre les nòmines d'un mateix treballador. Observeu que per implementar entitats febles es produeix un fenòmen que fins ara no havia aparegut. Una clau forana forma part d'una clau primària. És el cas de l'atribut `passaport` de la taula `nomines`.

En el Model 5.11 es mostra la versió relacional del disseny del Model 5.3.

```
comarca(comarca)
ciutat(ciutat,habitants,comarca)
persona(passaport,nom,cognom,mail,ciutat)
telefons(passaport,telefon)
coneix(coneix,es_coneguda)
soci(passaport,alta)
treballador(passaport,departament,obeeix)
esport(esport,preu,jugadors)
fa(soci,esport,quota)
nomines(passaport,periode,sou_base,retencio)
```

Model 5.11. *Model Relacional del disseny del Model Entitat Relació 5.3.*

L'extensió de relacions binàries M:N a relacions ternàries no té més secret. En lloc de dues, serien tres claus foranes. Depenent de les restriccions d'existència o d'unicitat de cada una de les claus controlaríem les cardinalitats de la relació. S'ha deixat enrera la inclusió de pavellons per agilitzar tot el procediment, considerant que a part del fet de ser tres claus foranes, no introduïa cap altre nou concepte.

Igualment s'ha abandonat el concepte d'atributs compostos perquè s'implementen només els atòmics ignorant el nom de l'atribut compost.

Bé, un cop esmicolat el procediment de traducció a partir del disseny cap al model relacional, fem una sinopsis de com han quedat les claus de les relacions del model relacional depenent del tipus d'entitats en el model ER que les han originat.

model ER	**claus del model relacional**
entitat	clau primària en un sol atribut
relació 1:N	clau forana
relació M:N	parella de claus foranes
atribut multivalorat	clau forana i cap clau primària
especialització	la clau primària és clau forana
entitat feble	clau primària formada per dos atributs, el primer dels quals és clau forana

Taula 5.3: *Sinopsis per la transformació del model ER al model relacional.*

5.3.5 Diagrama d'Esquemes d'una Base de Dades

Els diagrames de classe utilitzats en la programació orientada a objectes resulten unes eines de gran utilitat. Per altra banda, hi ha un isomorfisme prou intuitiu entre una relació en una base de dades i una classe d'objectes en la programació amb llenguatges d'alt nivell.

Com en tot, per comprendre l'impacte de cada definició convé conèixer la història. La programació orientada a objectes és dels anys noranta. El model ER dels setantes. O sigui, tant el model ER com el model relacional són considerablement anteriors als diagrames d'esquemes. Per això, en la descripció de les etapes del projecte, no hi considerem el diagrama d'esquemes com un pas imprescindible.

El diagrama d'esquemes d'una base de dades se situa en un punt intermig entre el model ER i el model relacional. Aquest tipus de diagrames van aparèixer amb l'Access de Microsoft a principis dels noranta, en el que va ser la primera interfície gràfica d'aplicació per a bases de dades, com es diu en la Secció 2.1.1. Tot i tenir greus mancances en la capacitat de representació el seu èxit ha estat notable. La prova és que actualment existeixen aplicacions d'interfície gràfica per tots els SGBDs, tot i que són programes absolutament independents del gestor.

Un exemple d'un esquema de relació, com es descrivia en el Model 5.1, era `persona(passaport,nom,cognom,mail,ciutat)`. Bé, doncs si tenim clar el que és un esquema de relació, llavors un diagrama d'esquema de relació és el mateix, però posat dins una caixa en vertical. El nom de la relació es posa dalt del rectangle com a títol, i sota, en format llista, els atributs començant per les claus primàries en negreta. El sol fet d'utilitzar la negreta reflecteix que és una eina posterior a les altres. O sigui, malgrat ser un instrument per la documentació entre humans, va néixer després de les aplicacions d'interfície. L'aspecte d'un diagrama d'esquema de relació entès de forma genèrica es mostra en el Diagrama 5.2.

persona
passaport
nom
cognom
mail
ciutat

Diagrama 5.2. *Exemple de diagrama d'esquema de relació.*

Amb aquests diagrames no hi ha una convenció prou estesa sobre la sintaxi com per poder establir les decisions del disseny amb el rigor que sí que ho permet un model ER o el mateix model relacional. És a dir, que això de la negreta tampoc és per posar les mans al foc.

Cal entendre doncs que un diagrama d'esquemes de relació no és una eina estrictament necessària, ja que durant vint anys es van fer bases de dades sense l'ús de diagrames d'esquemes. Com es deia en els primers paràgrafs d'aquest capítol, un model ER és una descripció orientada als humans, i un model relacional és el punt de partida per fer-li entendre a la computadora. Un diagrama d'esquemes és una eina auxiliar que no té per què aparèixer en la documentació del projecte, encara que quan es pretenen exposar algunes condicions sense massa rigor, efectivament resulti una eina prou útil.

És ben clar que la nomenclatura utlilitzada en els models ER té el valor que té majorment pel fet de ser convecionalment admesa. Això és una característica pròpia de qualsevol llenguatge humà. Tot i així, no hi ha una organització internacional que imposi l'ús de cap simbologia concreta. Això ha provocat que tot allò més complicat de representar gràficament hagi estat contaminat d'alternatives que han tingut més o menys èxit. Aquells que algun cop hagin programat aplicacions de disseny gràfic, s'hauran trobat que la complexitat de dibuixar una fletxa és superior a la de dibuixar un text. Això és així perquè un text sempre s'escriu igual, en canvi la forma que té una fletxa depèn de l'orientació de la línia sobre la que se suporti. Això mateix passa amb el sistema de Microsoft Access. De manera que els dissenyadors d'aquests tipus d'aplicacions es prenen la llibertat de redefinir els símbols que costa dibuixar automàticament introduïnt dialectes al model ER que després cal un esforç, menor però addicional, per poder interpretar.

És freqüent en moltes aplicacions d'interfície que enlloc de fletxes es limitin a escriure textualment dels cardinalitats al costat de les línies que uneixen les entitats.

Tot plegat desemboca en una eclosió de diferents versions de models ER. Hi ha qui per declarar una cardinalitat com 1:N, per exemple, no posa fletxa en el costat 1, i enlloc d'això es posa una mena de forquilla al costat N.

Bé, tampoc és la intenció d'aquest llibre adoctrinar sobre la manera de descriure els dissenys de bases de dades, de manera que les diferents simbologies que s'utilitzen s'han de considerar totes correctes. La que aquí s'ha mostrat és la primera que es va difondre. Ara bé, tan exteses estan les formes alternatives, que no hi ha motius per considerar-les incorrectes, ja que la seva proliferació és una demostració de la seva utilitat.

Sense més explicacions, en el Diagrama 5.3 es pot veure el diagrama d'esquemes de relació del disseny del Model 5.3. El Microsoft Access el faria semblant, però amb ∞'s als llocs on comencen i 1's als lloc on hi ha les puntes de les fletxes.

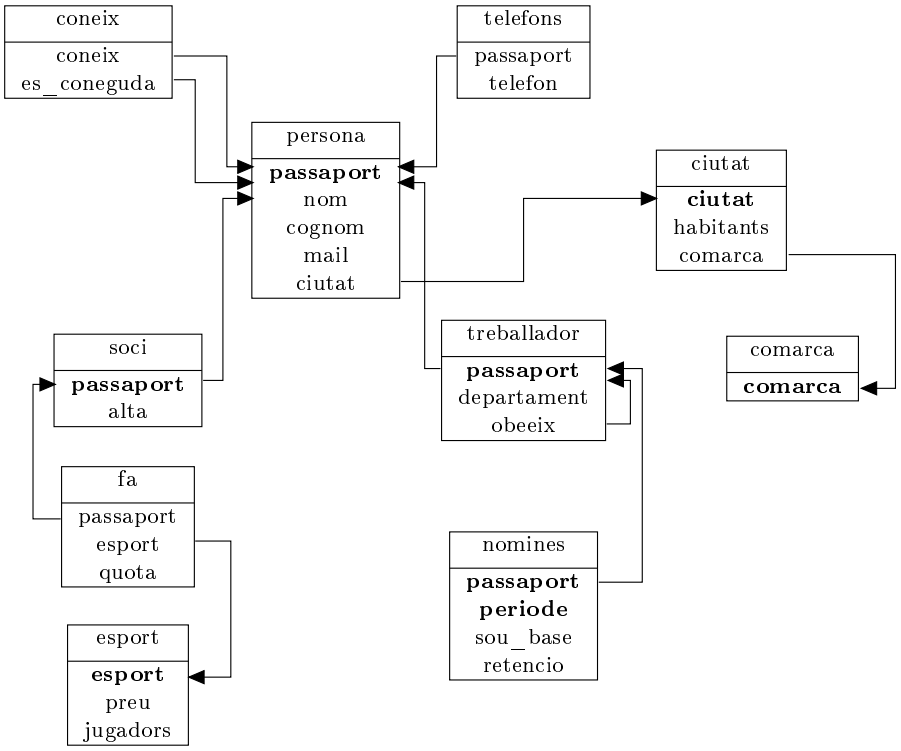


Diagrama 5.3. *Diagrama d'esquemes de relació del Model ER de l'exemple.*

5.4 Operacions Bàsiques amb Relacions

Fins aquí s'ha vist la manera d'emmagatzemar les dades. Amb relacions, claus primàries i claus foranes.

D'aquí en endavant s'exposa la manera de calcular les relacions resultants per les sol·licituds que es puguin plantejar. Donar respostes, obtenir expressions relacionals.

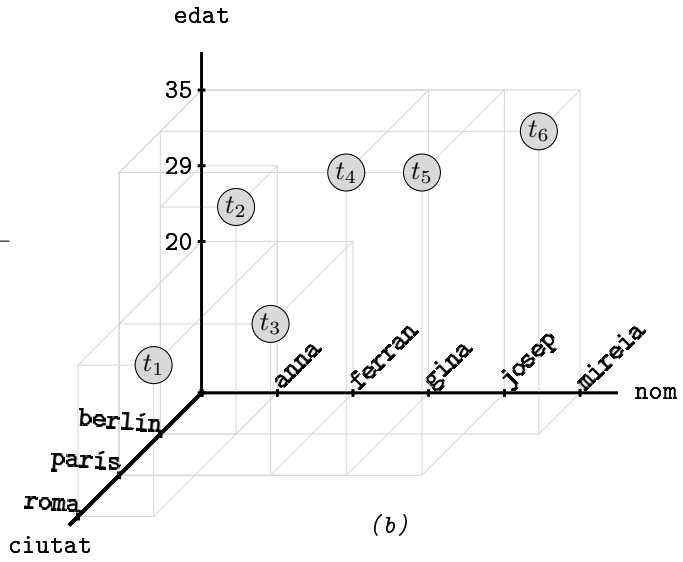
Diem que una operació és bàsica quan el resultat que obté no es pot obtenir per mitjà de cap altra operació. Són sis. La *selecció*, que obté la relació formada per les tuples de la relació donada que compleixen una condició. La *projecció* ens retorna una relació formada per les columnes de la relació d'entrada que se li demanen. El *producte cartesià*, que ens retorna una relació amb totes les possibles parelles entre tuples de les dues relacions d'entrada. La *unió*, que agrupa les tuples de dues relacions i ens en torna una. La *diferència*, que elimina totes les tuples de la primera relació d'entrada que estiguin a la segona. I el *renomenament*, que permet canviar el nom i el nom dels atributs a qualsevol expressió relacional.

En la Figura 5.5 es presenta una nova relació `persona` per als exemples que es veuran tot seguit.

`persona(nom,edat,ciutat)`

nom	edat	ciutat	
anna	20	roma	t_1
anna	29	berlín	t_2
ferran	20	París	t_3
gina	35	París	t_4
josep	35	París	t_5
mireia	35	berlín	t_6

(a)



(b)

Figura 5.5: *Relació* `persona` *dels exemples. (a) Descripció tabular. (b) Descripció cartesiana.*

5.4.1 Operació de Selecció

La nomenclatura utilitzada per l'operació de selecció s'indica a la Caixa 5.6,

$$\sigma_p(r)$$

Caixa 5.6. *Expressió relacional d'una selecció.*

on p és un predicat que se li proporciona com argument, i r és la relació d'entrada. Aquesta expressió es pronuncia "sigma sub pe de erra" o simplement "sigma pe de erra", i fa servir la lletra essa grega com a mnemotècnic de "selecció".

Retorna la relació formada per les tuples de la relació d'entrada que satisfan el predicat donat. L'esquema de la selecció és el mateix que el de la relació que se li dóna. Aquesta operació serveix per establir criteris.

exemple 5.1. *Obtenir la relació de persones que viuen a* parís.

solució $\sigma_{\texttt{ciutat='parís'}}(\texttt{persona})$

En la Figura 5.6 s'il·lustra la solució de l'exemple 5.1. A l'esquerra, Figura 5.6*(a)*, s'ha deixat en to més clar les tuples excloses de la solució.

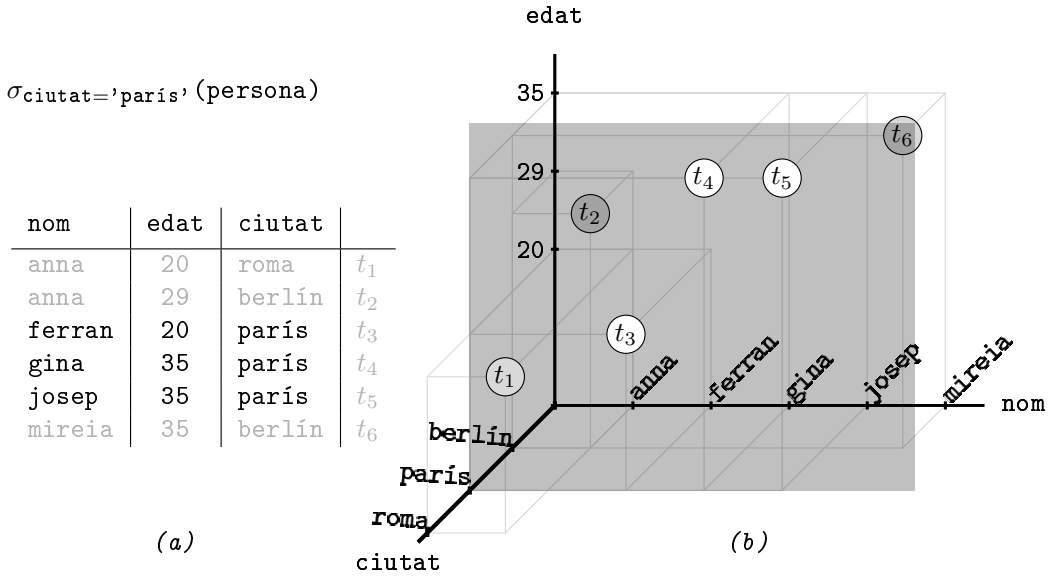


Figura 5.6: *Selecció de persones de* parís. *(a) Descripció tabular. (b) Descripció cartesiana.*

Segurament, allò més interessant de la visió cartesiana és l'analogia entre el predicat `ciutat = 'parís'` i el subespai format per aquest mateix pla. De manera que les tuples seleccionades, i que per tant formen part de la relació resultant, són les que s'ubiquen en el pla definit pel predicat. Això és molt gràfic. Noteu que el fet que el predicat estigui format per tan sols una proposició, fa que reduïm l'espai de la relació en una dimensió.

A continuació, el cas d'un predicat conjuntiu de dues proposicions.

exemple 5.2. *Persones de* 35 *anys que viuen a* parís.
solució $\sigma_{\texttt{ciutat='parís'} \wedge \texttt{edat=35}}(\texttt{persona})$

En la Figura 5.7 es dibuixa la solució de l'exemple 5.2

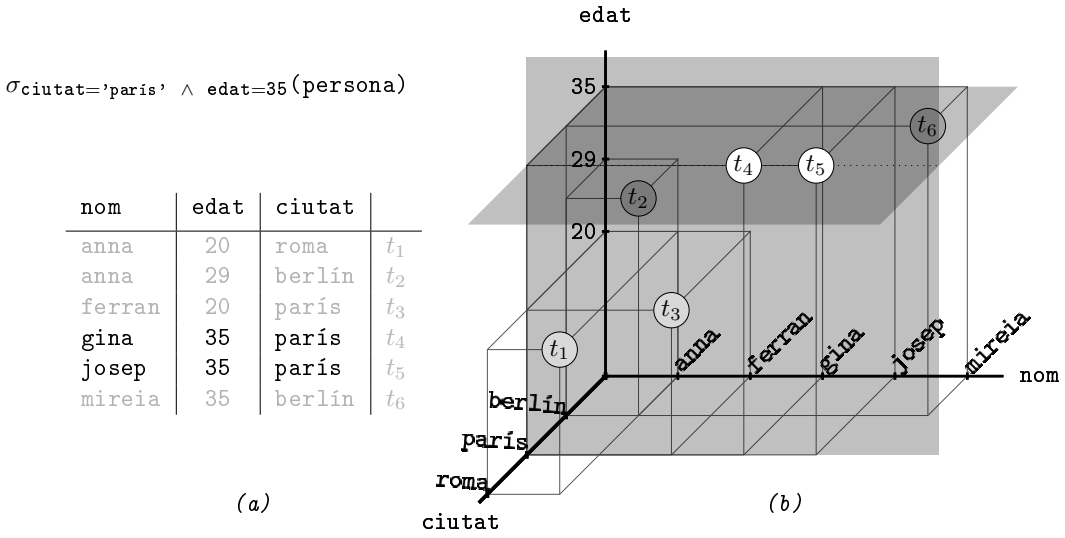


Figura 5.7: *Selecció de persones de* 35 *anys, de* parís. *(a) Descripció tabular. (b) Descripció cartesiana.*

Estem reduint molt més l'espai de selecció. D'un pla passem a un línia. La línia d'intersecció entre els dos plans donats per les proposicions, per tant, en la selecció resultant tan sols hi apareixeran les tuples que s'ubiquen en aquesta línia. Com es pot veure, el subconjunt de tuples seleccionades consta de les dues tuples, t_4 i t_5.

Tot plegat està molt relacionat amb l'anàlisi matemàtica i la resolució d'equacions lineals. És fàcil imaginar-se predicats que provoquessin relacions resultants buides. O sigui, si els dos plans de la Figura 5.7*(b)* enlloc de perpendiculars fossin paral·lels, llavors no hi hauria intersecció entre ells, i per tant el resultat de la selecció seria el conjunt buit, bé, una relació buida. Però clar, ¿què vol dir que els dos plans siguin paral·lels? Doncs vol dir que demanem una cosa impossible, com per exemple persones que tinguin *35* i *29* anys al mateix temps, o millor dit, en la mateixa tupla. I ja es veu que això és impossible.

Si en canvi considerem el predicat disjuntiu, amb la unió enlloc de la intersecció, persones que tinguin *35* anys o que visquin a *parís*, llavors les tuples seleccionades no seran només les de la línia d'intersecció entre els dos plans, sinó qualsevol tupla que s'ubiqui en un o altre pla. Això es mostra en la Figura 5.8, on es pot observar que la relació resultant consta de quatre tuples. A més de les dues de la intersecció han estat seleccionades t_3 per ser de *parís*, i t_6 per tenir *35* anys.

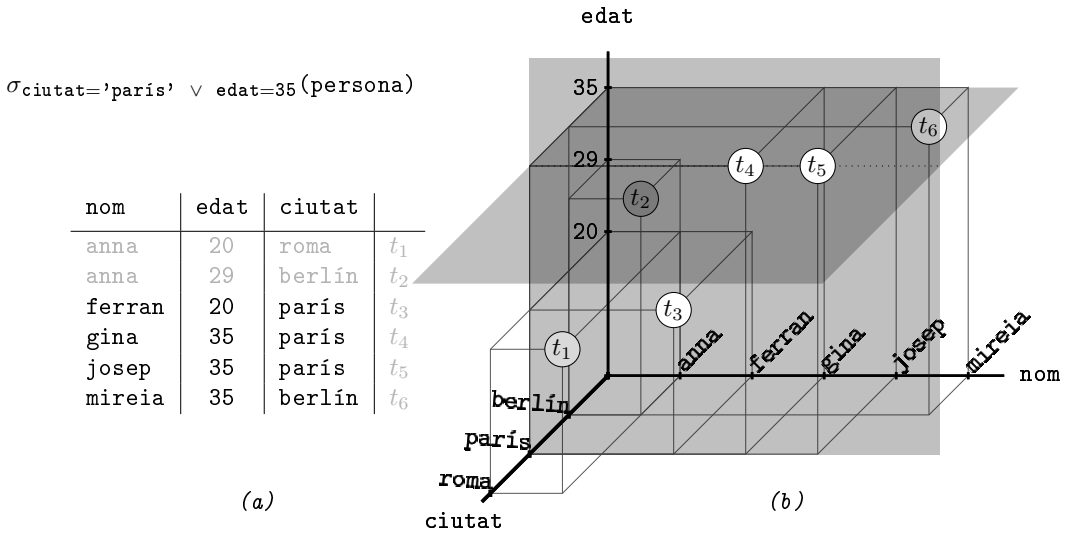


Figura 5.8: *Selecció de persones de* 35 *anys, o de* parís. *(a) Descripció tabular. (b) Descripció cartesiana.*

5.4.2 Operació de Projecció

El símbol de l'operador de projecció s'exposa a la Caixa 5.7,

$$\Pi_{A_1,A_2,...,A_k}(r)$$

Caixa 5.7. *Expressió relacional d'una projecció.*

sent $A_1, A_2, ..., A_k$ un subconjunt dels atributs d'r que és la relació d'entrada a l'operació. Aquesta expressió es pronuncia "pi sub a u, a dos, a ca, de erra", i fa servir la lletra pi grega com a mnemotècnic de "projecció".

De la relació d'entrada, el resultat d'una projecció tan sols conté els atributs presents en l'argument que se li dóna. Per tant, la projecció té per esquema

exactament l'argument. Aquesta operació serveix per especificar els atributs resultants d'una consulta.

En principi, la relació resultant tindrà tantes tuples com tingui la relació donada. Això no obstant, hi ha la possibilitat que les tuples de la relació d'entrada es diferencïin entre elles en camps que no són demanats en l'argument. Llavors, en la relació de sortida serien tuples iguals. Si això passés el nombre d'elements de la relació resultant es veuria reduït.

exemple 5.3. *Obtenir l'edat i la ciutat de totes les persones.*
solució $\Pi_{\texttt{edat,ciutat}}(\texttt{persona})$

En la Figura 5.9 es pot observar de manera gràfica la solució de l'exemple 5.3.

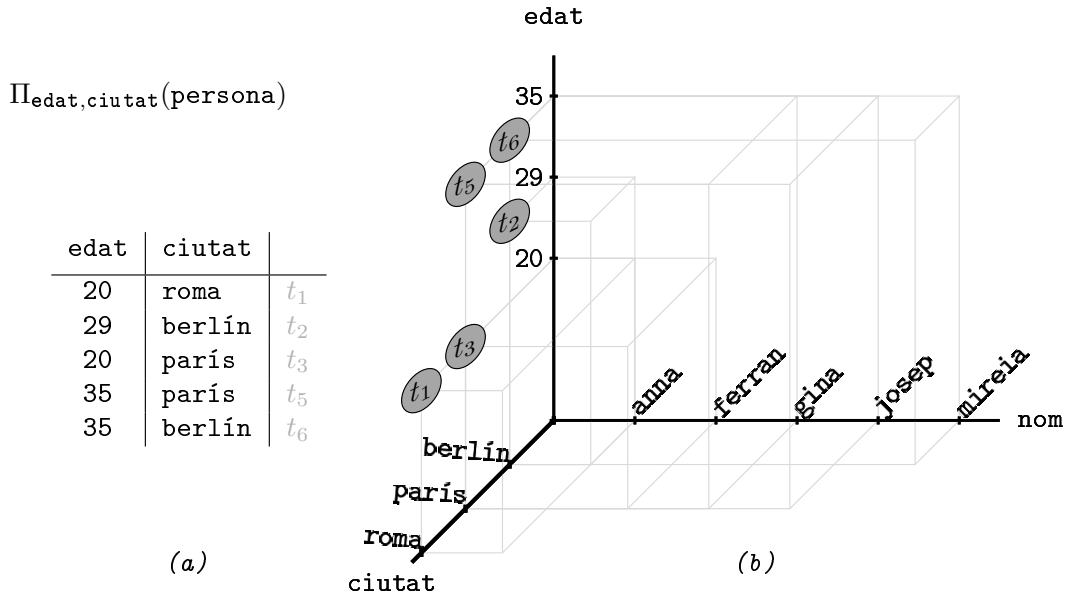


Figura 5.9: `edat` *i* `ciutat` *de tothom. (a) Descripció tabular. (b) Descripció cartesiana.*

La tupla t_4 ha desaparegut. L'atribut que la diferenciava de t_5 no ha estat projectat, i per tant en la relació resultant $t_4 = t_5$. L'exemple hagués estat igual de vàlid si hagués desparegut t_5.

Noteu també, a partir de la Figura 5.9*(b)*, que per imaginar el que fa la projecció convé imaginar-se la relació inicial de la Figura 5.5 enfocada des de la dreta, perpendicular al pla (`edat`,`ciutat`). Les ombres que dibuixen les tuples inicials en aquesta paret són la relació resultant. D'aquí el nom de projecció.

Bé, tot això havent projectat dos atributs dels tres de la relació inicial. Desgranem el cas d'un sol atribut.

exemple 5.4. *Obtenir les edats de totes les persones.*
solució $\Pi_{\texttt{edat}}(\texttt{persona})$

En aquest cas, hauríem d'imaginar que enfoquem des del davant un pèl a l'esquerra la Figura 5.9*(b)*, de manera que tot quedés concentrat contra l'eix vertical. Les ombres finals marcarien els diferents valors que hi ha per l'`edat` en la relació inicial.

En la Figura 5.10 es manifesta aquest cas. Han desaparegut varies tuples. Només queda una representant de cada edat.

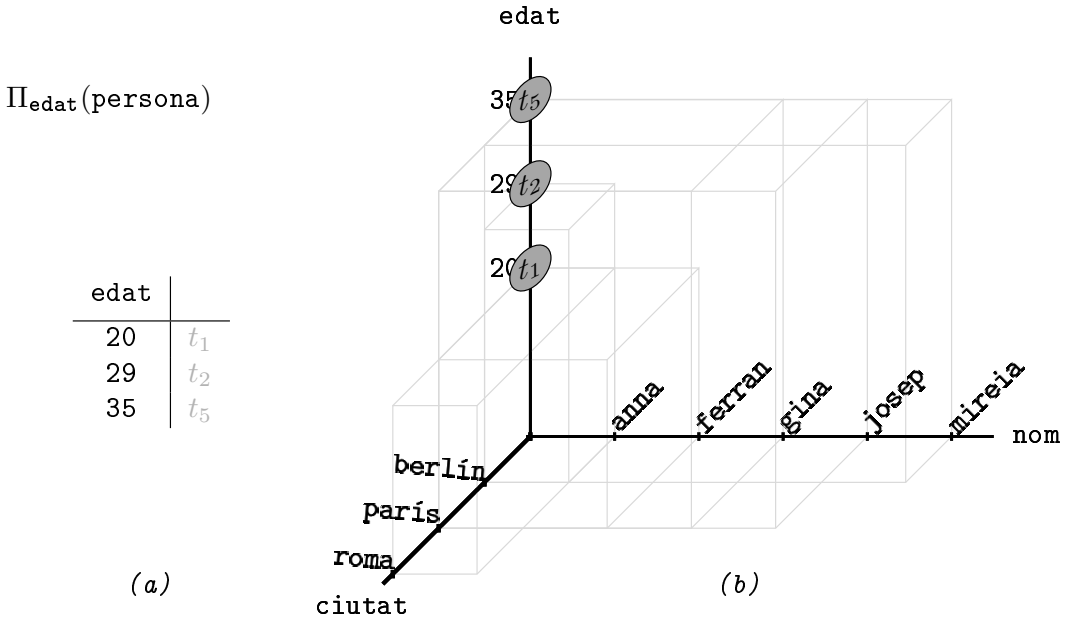


Figura 5.10: `edat` *de totes les persones. (a) Descripció tabular. (b) Descripció cartesiana.*

Una característica important de la projecció és que permet fer reordenacions en els esquemes de les relacions. Per exemple, la relació

$$\Pi_{\texttt{ciutat,nom,edat}}\ (\texttt{persona})$$

és la mateixa que persona, però amb els atributs en un altre ordre.

Composició d'operacions

Arribats aquest punt podem resoldre consultes més complicades. Això ens particiona les dades en dos grups. Els valors que serveixen per establir criteris, i els noms d'atribut que han d'encapçalar les columnes de la relació resultant, o sigui el seu esquema.

exemple 5.5. *Obtenir les edats i les ciutats de les persones que es diuen* anna.
solució $\Pi_{\texttt{edat,ciutat}}(\sigma_{\texttt{nom='anna'}}\ (\texttt{persona}))$

En la visió cartesiana de l'exemple 5.5 impresa en la Figura 5.11 hi ha dues etapes. Primerament, se seleccionen les tuples del pla `nom = 'anna'`, que apareixen en to més clar. I tot seguit, es projecten al pla (`edat`,`ciutat`). En total, queden les dues tuples que apareixen en blanc, encara que ombrejat pel pla, en la Figura 5.11*(b)*, que són t_1 i t_2 tal com s'indica en la descripció tabular de la Figura 5.11*(a)*.

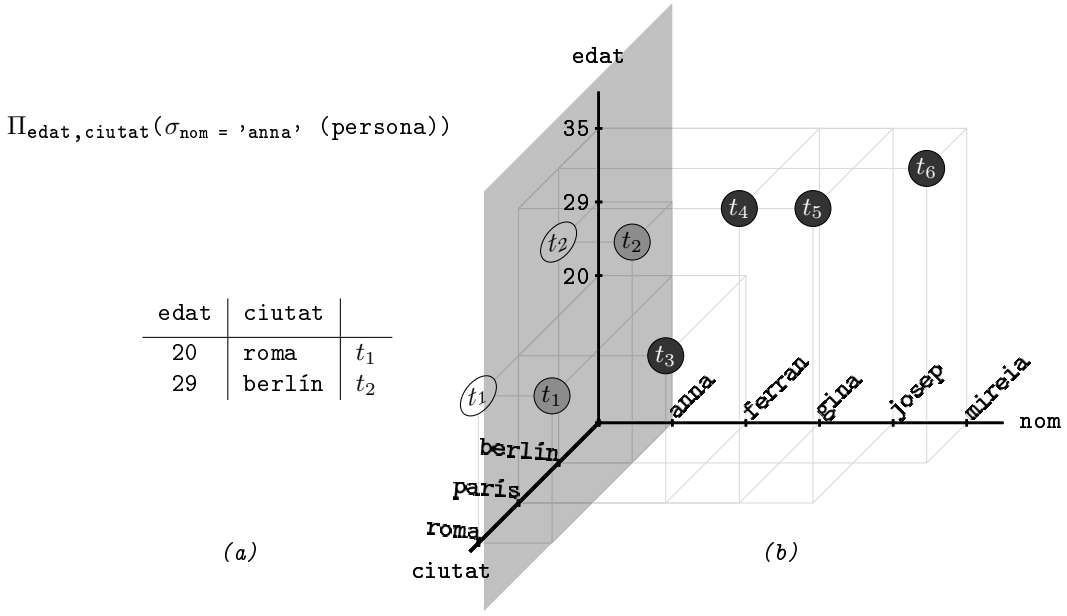


Figura 5.11: `edat` *i* `ciutat` *de les persones que es diuen* anna. *(a) Descripció tabular. (b) Descripció cartesiana.*

5.4.3 Producte Cartesià de Relacions

Per al producte cartesià es fa servir el mateix operador que per conjunts, com s'indica en la Caixa 5.8,

$$r \times s$$

Caixa 5.8. *Expressió relacional del producte cartesià entre dues relacions.*

sent r i s dues relacions amb esquemes qualssevol.

Sens dubte, aquesta és l'operació més habitual en les bases de dades relacionals. El producte cartesià entre dues relacions és una nova relació en la qual cada tupla és una parella possible formada per un element de cada una de les dues relacions d'entrada. Per tant, hi ha tants elements com parelles possibles, el producte de cardinals. De fet, aquesta definició no té res de nou respecte la mateixa operació definida entre conjunts, llevat la forma d'aparellar els elements. Els elements d'r i s s'aparellen definint les tuples del producte amb els atributs d'r i d's. Ergo l'esquema del producte cartesià és la unió dels esquemes de les relacions d'entrada. En cas que hi hagi algun atribut que es digui igual en r i en s, caldrà posar per prefix el nom de la relació, anomenant als atributs homònims $r.A$ i $s.A$, cosa que passarà amb les claus foranes. O també es pot utilitzar operacions de canvi de nom pels atributs. L'operació de renomenament es veu a la Secció 5.4.6.

En l'exemple de la Figura 5.12 s'utilitzen dues noves relacions d'entrada. De fet, la primera, `persona`, coincideix amb les tres darreres tuples de la relació de les seccions anteriors. La segona és nova, `ciutat(ciutat,país)`. Suposem que la columna `ciutat` de la relació `persona` fa el paper de clau forana a la relació `ciutat`. En la Figura 5.12*(a)* es presenta el contingut de les dues relacions d'entrada. A sota a l'esquerra, Figura 5.12*(b)*, la relació resultant, `persona` × `ciutat`.

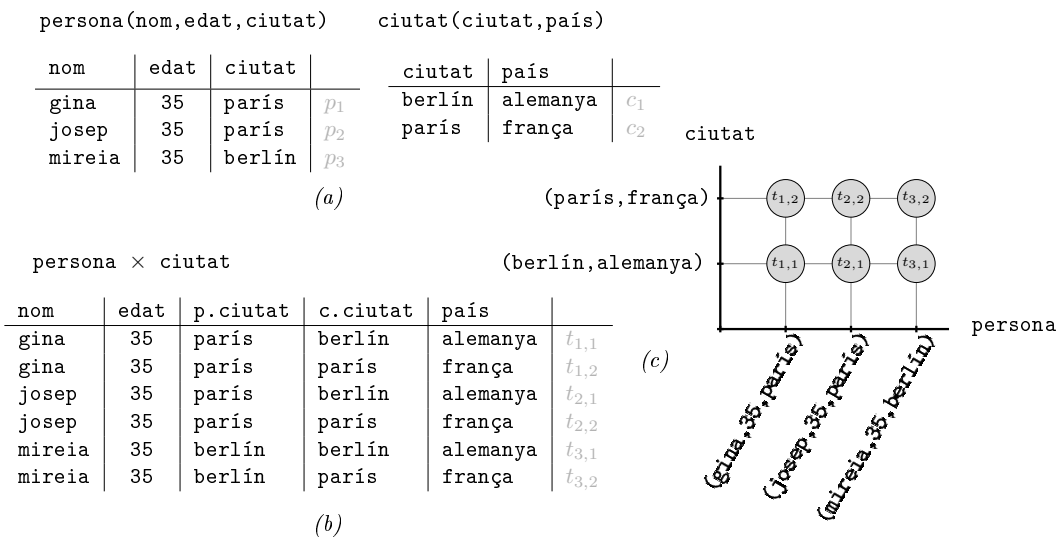


Figura 5.12: *Producte cartesià. (a) Relacions d'entrada. (b) Descripció tabular. (c) Descripció cartesiana.*

Filòsoficament, tal com es deia en la Secció 5.2.1, noteu que hi ha cert paral·lelisme entre el producte cartesià de relacions, on l'esquema és la unió dels esquemes de la relacions inicials, i la multiplicació de números enters, que el resultat té tantes xifres com la suma dels números de xifres dels dos números que es multipliquen. Estem parlant de logaritmes, i el logaritme del producte és la suma dels logaritmes.

Aquesta analogia encara va més lluny. Fixeu-vos que per omplir la taula de la Figura 5.12*(b)* es tracten els valors dels atributs de cada relació com si fossin xifres d'un sistema d'enumeració.

Per distingir els atributs `ciutat` en la relació final s'ha utilitzat com a prefix la inicial del nom de les relacions d'entrada.

Ara, en cada eix de la Figura 5.12*(c)* s'hi repesenta una relació completa establint un ordre entre totes les seves tuples, no com els exemples anteriors en els que s'hi representava un atribut. I també en la Figura 5.12*(c)* es pot comprovar que el cardinal del producte cartesià $|\texttt{persona} \times \texttt{ciutat}|$ és igual al producte dels cardinals de les relacions que el composen, $|\texttt{persona}| * |\texttt{ciutat}|$. O sigui, 3 * 2 = 6.

exemple 5.6. *A quin país viu la* gina*?*

solució $\Pi_{\texttt{país}}\ (\ \sigma_{\texttt{nom='gina'} \wedge \texttt{p.ciutat=c.ciutat}}\ (\texttt{persona} \times \texttt{ciutat}))$

Atenció a l'exemple 5.6. Són tres passes.

- Calcular totes les parelles possibles.
- Seleccionar aquelles que les columnes vinculades coincideixin en valor i al mateix temps satisfacin el criteri especificat en l'enunciat.
- Projectar els atributs que es demanin.

En definitiva, el producte cartesià entre dues relacions vinculades serveix per disposar d'una relació amb la unió de tots els atributs en la qual només tenen sentit aquelles files on les columnes vinculades coincideixin en valor.

Aquesta història es resumeix molt barroerament en cinc vinyetes, encara que dient mentides. A veure si les descobriu. És la tira de la Figura 5.13. La historieta comença a la vinyeta 1, amb una relació representada per un conjunt de punts. La línia horitzontal. A la segona, hi ha dues relacions.

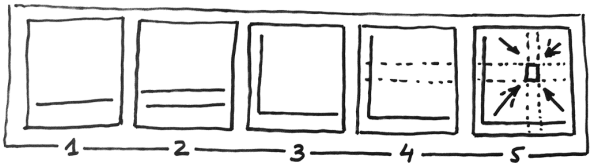


Figura 5.13: *La historieta de l'àlgebra relacional.*

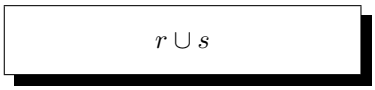
A la tercera, el producte cartesià. Després, a partir del resultat de la tercera, seleccionem algunes files en la quarta, i les projectem en la cinquena obtenint el resultat del quadradet petit.

Potser no us en heu adonat, però l'explicació de la historieta és un raonament fal·laç. La trampa està en la interpretació del pla, que en la tercera vinyeta representa el pla cartesià format per totes les parelles possibles d'elements de les dues relacions, o sigui, que una fila representa un element d'una d'elles, i una columna representa un element de l'altra. En canvi, a la quarta vinyeta el pla representa la relació resultant del producte cartesià. Com si una fila fos un element i una columna un atribut.

En qualsevol cas, tot plegat resulta una imatge prou mnemotècnica.

5.4.4 Unió de Relacions

La unió de relacions és com la unió de conjunts, i utilitza el símbol de la Caixa 5.9,



Caixa 5.9. *Expressió relacional de la unió de dues relacions.*

sent r i s dues relacions.

La unió resultant de dues relacions és una relació amb les tuples de les dues.

L'operació d'unió entre relacions només està definida entre relacions compatibles, tal com s'ha definit en la Secció 5.2.1. Lògicament, l'esquema de la unió prendrà el domini més gran, o sigui menys restrictiu, entre els esquemes de les relacions d'entrada per cada un dels atributs. Tot i així, gairebé sempre els esquemes són exactament iguals, o siqui que l'esquema de la unió és igual als esquemes de les relacions entrants.

En essència, serveix per la inserció de noves tuples en relacions existents. Podem afegir elements a les relacions utilitzant l'operació d'unió. En els casos més senzills inserirem tuples constants, com en l'exemple 5.7.

exemple 5.7. *Afegir en* joan*, de* 25 *anys, que viu a* atenes *a la relació* `persona`.

solució `persona` $\cup$ {(`'joan'`,25,`'atenes'`)}

Noteu que en l'expressió relacional que resol l'exemple 5.7 el segon operand de l'expressió és una relació, i per això les claus d'obertura i tancament, composada d'una sola tupla, i per això els parèntesis.

En la Figura 5.14 s'ha incorporat aquest nou element al conjunt, identificant-lo com a t^*.

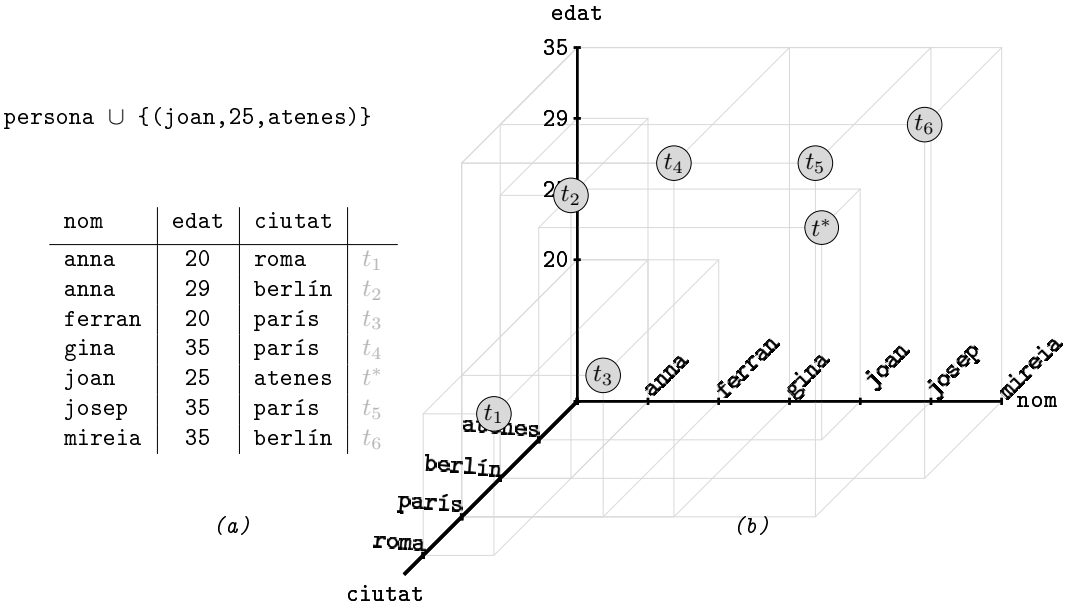


Figura 5.14: *Inserció de la tupla* ('joan',25,'atenes'). *(a) Descripció tabular. (b) Descripció cartesiana.*

5.4.5 Diferència de Relacions

Per la diferència entre dues relacions s'utilitza el mateix símbol que per la resta entre dos números enters, com s'indica en la Caixa 5.10,

$$r - s$$

Caixa 5.10. *Expressió relacional de la diferència de relacions.*

sent r i s dues relacions compatibles, com en el cas de la unió.

La relació resultant de la diferència de la relació r menys la relació s és la relació de les tuples d'r que no estan a s. L'esquema de la diferència és l'esquema de la primera de les dues relacions d'entrada, tot i que normalment coincideixen.

Noteu doncs que totes aquelles tuples de la relació s que no pertanyin a r no tenen cap impacte en la relació resultant.

exemple 5.8. *Eliminar totes les persones de* 35 *anys de la relació* persona.
solució persona - $\sigma_{\texttt{edat=35}}$ (persona)

L'expressió solució de l'exemple 5.8 treu totes les tuples del pla horitzontal corresponent a l'edat de *35* anys, tal com s'il·lustra a la Figura 5.15.

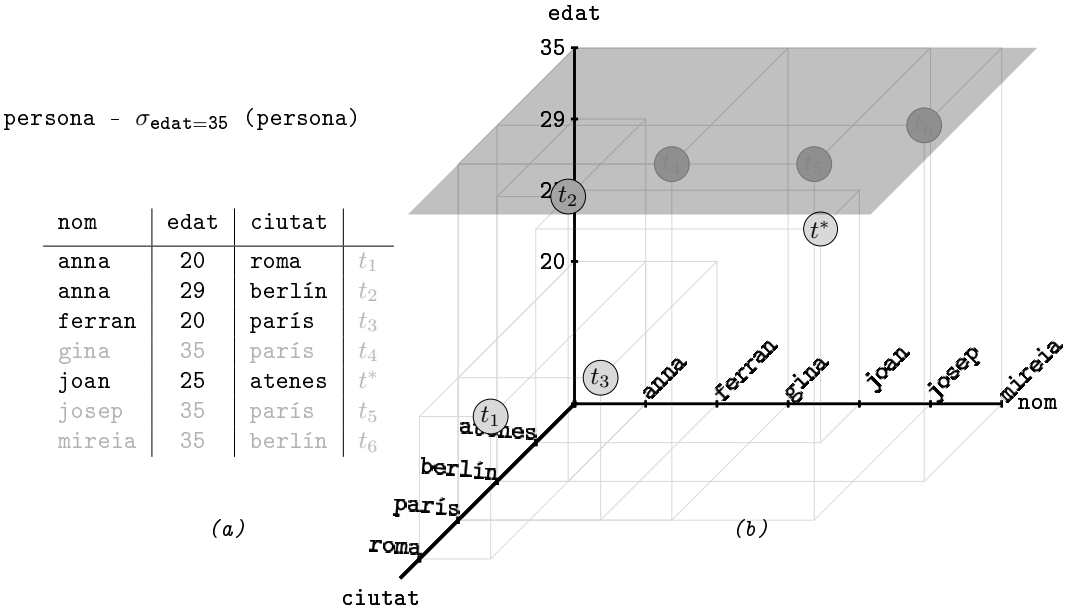


Figura 5.15: *Eliminació de les persones de* 35 *anys. (a) Descripció tabular. (b) Descripció cartesiana.*

5.4.6 Renomenament de Relacions

L'operació de renomenament de relacions s'expressa amb la lletra grega ro, com diu la Caixa 5.11,

$$\rho_{x(A_1,A_2,...,A_n)}(E)$$

Caixa 5.11. *Expressió relacional per al renomenament de qualsevol expressió relacional.*

on x és el nou nom per l'expressió E, i $A_1, A_2, ..., A_n$ és un argument optatiu pel nom pels atributs que assignem a la relació resultant d'avaluar l'expressió relacional E, o sigui, l'esquema d'x.

Si es proporciona l'argument optatiu $A_1, A_2, \ldots, A_n$, l'esquema del renomenament és l'argument donat. Si no, és el mateix esquema de l'expressió relacional que rep com a relació d'entrada, llavors els noms dels atributs són els que tinguin les relacions involucrades en l'expressió relacional E, amb el prefix x. En aquesta definició es posa E enlloc d'r per indicar que és freqüent l'ús del renomenament a partir d'una expressió.

Hi ha dos usos diferenciats d'aquesta operació.

Per un costat podem usar-la tan sols amb la finalitat de canviar els noms dels atributs d'una relació. Aquest ús és particularment útil per productes cartesians de relacions amb atributs homònims.

exemple 5.9. *Renomenar els atributs de la relació* `persona`*, o sigui* `nom`*,* `edat` *i* `ciutat`*, a* `nom`*,* `anys`*, i* `lloc`*.*

solució $\rho_{\texttt{persona(nom,anys,lloc)}}(\texttt{persona})$

En la Figura 5.16 es mostra la solució de l'exemple 5.9.

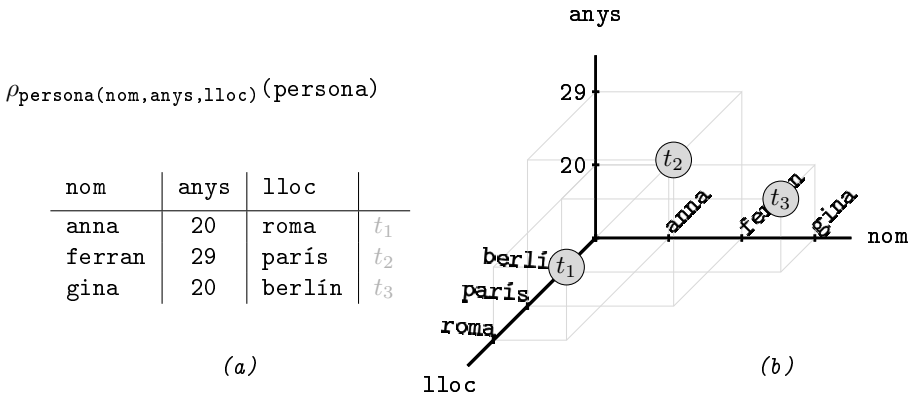


Figura 5.16: *Renomenament dels atributs de la relació* `persona`*. (a) Descripció tabular. (b) Descripció cartesiana.*

I per altra banda, el renomenament té un ús bastant més críptic. És el cas en que es pretén referenciar dos cops una mateixa relació en una composició d'operacions.

Podem assignar un nom a una expressió del nucli d'una composició d'operacions, per utilitzar-lo des de les operacions més exteriors, o sigui finals, d'aquesta mateixa composició. Quan es tracta de comparar tots els elements d'una relació entre ells, per exemple. En general, qualsevol tractament que des del punt de vista de la programació requerís un bucle dins d'un bucle. I això vol dir, operacions que involucrin totes les parelles possibles de tuples d'una mateixa relació. Són consultes més complexes.

Per l'exemple 5.10 es parteix de la relació `persona(nom,anys,lloc)` de la Figura 5.16.

exemple 5.10. *Obtenir el nom de la persona més gran.*

solució $\Pi_{\texttt{nom}}(\texttt{persona}) - \Pi_{\texttt{x.nom}}(\sigma_{\texttt{x.anys}<\texttt{y.anys}}(\rho_{\texttt{x}}(\texttt{persona}) \times \rho_{\texttt{y}}(\texttt{persona})))$

Desgranem la solució de l'exemple 5.10. Ataquem d'entrada el segon operand de la diferència,

$$\Pi_{\texttt{x.nom}}(\sigma_{\texttt{x.anys}<\texttt{y.anys}}(\rho_{\texttt{x}}(\texttt{persona}) \times \rho_{\texttt{y}}(\texttt{persona})))$$

Això és una relació d'un sol atribut anomenat `nom`. Està formada per les persones de la relació tals que existeix algú més gran que elles. Pam a pam. Comencem per observar quin és el contingut de la relació que es correspon al producte cartesià

$$\rho_{\texttt{x}}(\texttt{persona}) \times \rho_{\texttt{y}}(\texttt{persona})$$

que es mostra en la Taula 5.4.

x.nom	x.anys	x.lloc	y.nom	y.any	y.lloc
anna	20	roma	anna	20	roma
anna	20	roma	ferran	29	parís
anna	20	roma	gina	20	berlín
ferran	29	parís	anna	20	roma
ferran	29	parís	ferran	29	parís
ferran	29	parís	gina	20	berlín
gina	20	berlín	anna	20	roma
gina	20	berlín	ferran	29	parís
gina	20	berlín	gina	20	berlín

Taula 5.4: *Contingut de la relació* $\rho_{\texttt{x}}(\texttt{persona}) \times \rho_{\texttt{y}}(\texttt{persona})$.

Altre cop es pot observar que la manera d'omplir la relació de la Taula 5.4 és com si les tuples de la relació persona fossin xifres d'un sistema d'enumeració, i en la relació hi haguéssim posat tots els números de dues xifres d'aquest sistema. Dit això, observem que fa la selecció

$$\sigma_{\texttt{x.anys}<\texttt{y.anys}}(\rho_{\texttt{x}}(\texttt{persona}) \times \rho_{\texttt{y}}(\texttt{persona})).$$

És a dir, ens quedem amb les dues tuples que $\texttt{x.anys} < \texttt{y.anys}$ de la Taula 5.4. En la Taula 5.5 s'ha deixat en to més clar les tuples eliminades de la solució.

x.nom	x.anys	x.lloc	y.nom	y.any	y.lloc
anna	20	roma	anna	20	roma
anna	20	roma	ferran	29	parís
anna	20	roma	gina	20	berlín
ferran	29	parís	anna	20	roma
ferran	29	parís	ferran	29	parís
ferran	29	parís	gina	20	berlín
gina	20	berlín	anna	20	roma
gina	20	berlín	ferran	29	parís
gina	20	berlín	gina	20	berlín

Taula 5.5: *Selecció* $\sigma_{\texttt{x.anys}<\texttt{y.anys}}(\rho_{\texttt{x}}(\texttt{persona}) \times \rho_{\texttt{y}}(\texttt{persona}))$.

I a partir d'aquí, ja és molt senzill. Projectem els dos noms, *anna* i *gina*, com es veu en la Taula 5.6 que conté els noms de les persones tals que hi ha algú més gran que elles. O sigui, totes les persones excepte la més gran.

nom
anna
gina

Taula 5.6: *Projecció* $\Pi_{\texttt{nom}}(\sigma_{\texttt{x.anys}<\texttt{y.anys}}(\rho_{\texttt{x}}(\texttt{persona}) \times \rho_{\texttt{y}}(\texttt{persona})))$

Finalment, restem de tots els noms, els que apareguin a la relació de la Taula 5.6. En la Figura 5.17 es mostra la part final del procediment de la consulta de l'exemple 5.10.

nom
anna
ferran
gina

$-$

nom
anna
gina

$=$

nom
ferran

Figura 5.17: *Última etapa de la resolució de l'exemple 5.10.*

El problema de la persona més gran s'ha resolt utilitzant el concepte de complementari. És a dir, s'ha calculat primer la relació de tots excepte el més gran, i després s'ha pres la relació complementària respecte la relació inicial. No hi ha altra manera. Si la selecció hagués agafat les tuples tals que existeixi algú més gran que tots els altres, hagués sortit una relació buida. I si s'hagués demanat per més gran o igual, s'haguéssin seleccionat totes les tuples.

L'exemple 5.10 és complicat. Serveix per mostrar la capacitat expressiva de les operacions bàsiques que aquí tanquem. En endavant, ens aprovisionarem d'operacions més sofisticades que ens permetran seleccionar directament un màxim.

5.5 Operacions Addicionals

Les operacions addicionals són operacions que no contribueixen al potencial expressiu de l'àlgebra. La demostració matemàtica d'aquesta afirmació consisteix en expressar cada una d'aquestes operacions en termes d'operacions bàsiques, com efectivament es fa tot seguit. Aquestes operacions s'han d'entendre com eines auxiliars que fan les consultes més concises, i per tant més legibles.

En primer lloc hi ha la *intersecció*, que es pot construir a partir de la diferència. Després, la *reunió natural* que aglutina producte cartesià, selecció i projecció. Una generalització de la reunió natural és la reunió interna, que dóna pas a les altres tres operacions de reunió, les externes. També es considera la *divisió* d'operacions, que es defineix en base al producte cartesià, i finalment, per facilitar la descripció de consultes, s'inclou una instrucció d'assignació. L'*assignació* es presenta com a última eina auxiliar perquè els resultats de les consultes són expressions, no instruccions. Les assignacions permeten fer passos intermitjos, però en qualsevol cas la consulta final ha de resultar ser una expressió com en tots els exemples anteriors.

Les relacions `persona(nom,edat,ciutat)` i `ciutat(ciutat,país)` que es mostren a la Taula 5.7 serviran pels exemples d'aquesta secció.

nom	edat	ciutat
gina	35	parís
josep	35	parís
mireia	35	berlín
pere	23	roma

ciutat	país
berlín	alemanya
parís	frança
lisboa	portugal

Taula 5.7: *Relacions* `persona` *i* `ciutat`.

5.5.1 Intersecció de Relacions

El símbol de la intersecció de relacions, com el de la unió, és igual que per conjunts, com es pot veure en la Caixa 5.12,

$$r \cap s$$

Caixa 5.12. *Expressió relacional de la intersecció de relacions.*

sent r i s compatibles com en la unió o la diferència.

El resultat de la intersecció de relacions és la relació de tuples que estiguin a r i a s, com ja ens podíem imaginar. Respecte el seu esquema, allò mateix que s'ha dit de la unió. O sigui, la combinació de dominis menys restrictiva pels atributs. Tot i així, la major part de les vegades els esquemes de les relacions d'entrada coincideixen, i llavors també amb el de la intersecció.

Una tupla està a r i a s si tenen els mateixos valors en tots i cadascún dels atributs.

No és una operació bàsica, com es demostra en la Caixa 5.13,

$$r \cap s = r - (r - s)$$

Caixa 5.13. *La intersecció no és una operació bàsica.*

Ja se sap, la intersecció entre dos conjunts, o també dues mentalitats, és el que en queda després d'eliminar les diferències.

És curiós que sent una operació commutativa es pugui expressar amb un aspecte tan poc commutatiu com el de la Caixa 5.13. Però tot i així, recordar els mnemogrames vistos per aquestes operacions quan es definien entre conjunts, en la Secció 1.2, facilita la comprensió de la igualtat de la Caixa 5.13.

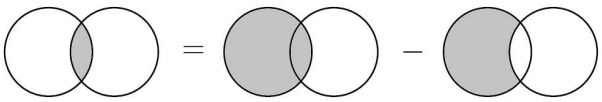


Figura 5.18: *La intersecció és un menys la diferència.*

En la Figura 5.18 es pot contemplar el que podríem anomenar una operació entre ideogrames. És important tenir present que aquesta forma d'expressar-se no té cap rigor, i tan sols serveix per facilitar la comprensió de la idea de llimar diferències.

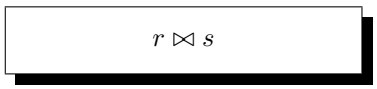
L'enunciat de l'exemple 5.11 ja s'ha vist en la Secció 5.4.1. Aquí es resol per mitjà de la intersecció de relacions.

exemple 5.11. *Obtenir els noms de les persones de* París, *de* 35 *anys.*

solució $\Pi_{\texttt{nom}}(\sigma_{\texttt{edat=35}}(\texttt{persona})) \cap \Pi_{\texttt{nom}}(\sigma_{\texttt{ciutat='París'}}(\texttt{persona}))$

5.5.2 Reunió Natural

L'operació de reunió natural entre dues relacions s'indica amb el signe de la Caixa 5.14,



Caixa 5.14. *Expressió relacional de la reunió natural entre dues relacions.*

on r i s són dues relacions, suposadament amb alguns atributs homònims. L'expressió de la Caixa 5.14 és pronuncia "erra reunió natural essa", encara que sovint s'usa l'anglicisme *natural join*, "erra natural join essa". Al símbol $\bowtie$, també hi ha qui li diu corbatí. Té gràcia, "erra corbatí essa".

La reunió natural retorna la selecció de les tuples del producte cartesià que tinguin valors coincidents en les columnes homònimes.

Aquestes columnes homònimes resulten per tant equivalents, i la reunió natural tan sols les projecta un cop.

Això significa que l'esquema de la reunió natural tindrà la unió, en el sentit més rigorós, dels esquemes de les relacions d'entrada. Numèricament, l'esquema tindrà la suma de les quantitats d'atributs de les relacions d'entrada menys la quantitat d'atributs comuns, és a dir homònims. Pel cas de l'exemple, `persona` té tres columnes, `ciutat` dues, i tenen una columna homònima. L'esquema de la reunió natural tindrà per tant quatre columnes.

Tornem a l'exemple 5.6 de la pàgina 128 on la consulta era a quin país vivia la *gina*. L'expressió que solucionava la consulta era

$$\Pi_{\texttt{país}} \left(\sigma_{\texttt{nom='gina'} \wedge \texttt{p.ciutat=c.ciutat}} (\texttt{persona} \times \texttt{ciutat}) \right).$$

És ben clar que el predicat de la selecció,

$$\texttt{nom} = \texttt{'gina'} \wedge \texttt{p.ciutat} = \texttt{c.ciutat},$$

es composa de dues proposicions. La primera, `nom = 'gina'`, és conseqüència directa de l'enunciat de la consulta, a quin país viu la *gina*. Però la segona no. No depèn directament de la consulta. Hi ha moltes altres consultes que també requeririen la proposició `p.ciutat = c.ciutat` en el seu predicat. En definitiva, és una proposició interna. Si enlloc del producte cartesià utilitzem la reunió natural, ens estalviarem afegir aquesta proposició en totes les consultes.

D'aquesta manera podem reformular la solució de l'exemple 5.6 com es pot veure en l'exemple 5.12.

exemple 5.12. *A quin* `país` *viu la* gina*?*

solució $\Pi_{\texttt{país}}$ ($\sigma_{\texttt{nom='gina'}}$ (`persona` $\bowtie$ `ciutat`)).

Independentment que la millora sigui poc notable a nivell lexicogràfic, sí que és interessant a nivell filosòfic, ja que en la solució de l'exemple 5.12 tots els arguments, el de la projecció i el de la selecció, formen part de l'enunciat del problema, o sigui que són específics d'aquesta consulta.

Observeu doncs que la segona proposició del predicat de la selecció, `p.ciutat` = `c.ciutat`, queda assumida en el fet que les columnes es diguin igual, i per tant reflectida en l'estructura del model.

A la Taula 5.8 hi ha el contingut de la reunió natural de les relacions de la Taula 5.7.

nom	edat	ciutat	país
gina	35	parís	frança
josep	35	parís	frança
mireia	35	berlín	alemanya

Taula 5.8: *Reunió natural* `persona` $\bowtie$ `ciutat` *amb els valors de la Taula 5.7.*

La reunió natural ens elimina les files que no tenien sentit en la composició selecció de producte cartesià, a més d'una de les dues columnes equivalents.

A partir de la relació resultant d'aquesta operació, podem projectar qualsevol atribut de les dues relacions i també seleccionar a partir de qualsevol altre atribut de les dues relacions. Ja es veu que la potència d'aquesta operació és extraordinària.

Reunió interna

L'operació de reunió interna, o reunió zeta, entre dues relacions es denota amb el mateix signe que la reunió natural, amb un predicat simbolitzat per la lletra zeta minúscula grega θ, que es pronuncia zeta. En la Caixa 5.15 es mostra l'expressió.

$$r \bowtie_\theta s$$

Caixa 5.15. *Expressió relacional de la reunió interna entre dues relacions.*

El resultat de la reunió interna és la selecció de les files del producte cartesià que satisfacin el predicat θ. És a dir,

$$r \bowtie_\theta s = \sigma_\theta(r \times s).$$

Es tracta d'una generalització de la reunió natural que permet associar tuples de les dues relacions encara que les columnes no es diguin igual.

L'esquema de la reunió interna no agrega columnes homònimes, té tants atributs com la suma de les quantitats d'atributs de les relacions d'entrada. Això significa que si hi ha columnes homònimes hi haurà títols repetits en la relació resultant, i per tants si cal referenciar-los posteriorment caldrà utilitzar renomenaments.

La reunió interna s'utilitza en particular per establir les proposicions dels predicats que lliguen atributs estructurals. És a dir, claus foranes i claus principals, quan per les raons que sigui no es poden dir igual. Cal tenir en compte que no sempre es pot anomenar les claus foranes amb el nom de la clau primària a la qual apunten. Per exemple, tres casos.

- Quan hi ha dues relacions 1:N entre la mateixa parella d'entitats en el model ER, com en l'exemple del Model 4.13, on entre persona i ciutat hi havia dues relacions 1:N.
- En una autorelació 1:N, com en el cas de la clau forana `obeeix` del Model 5.11.
- Quan hi ha una autorelació M:N, con en la relació `coneix`, del mateix Model 5.11.

En aquestes situacions no es pot utilitzar la reunió natural. Llavors s'acostuma a fer amb la reunió interna, que gairebé és una notació alternativa a la selecció d'un producte cartesià. Ajuda a justificar la seva definició, a més, que la reunió interna serveix de base per les reunions externes que es veuran més endavant.

El contingut de la reunió interna entre les relacions de la Taula 5.7 és a la Taula 5.9.

nom	edat	ciutat	ciutat	país
gina	35	parís	parís	frança
josep	35	parís	parís	frança
mireia	35	berlín	berlín	alemanya

Taula 5.9: *Reunió interna* `persona` $\bowtie_{\texttt{p.ciutat=c.ciutat}}$ `ciutat` *amb els valors de la Taula 5.7.*

La columna ciutat, doncs, apareix repetida. Això és així perquè en la majoria dels casos que s'utilitza la reunió interna és per després projectar alguns atributs concrets.

Els dos propers exemples fan referència al Model 5.12. Hi ha treballadors que pertanyen a departaments, en els que manen uns caps. I també persones que coneixen a persones.

```
persona(passaport,nom,cognom,ciutat)
coneix(coneix,es_coneguda)
treballador(passaport,departament,obeeix)
```

Model 5.12. *Fragment Model 5.11 relacional.*

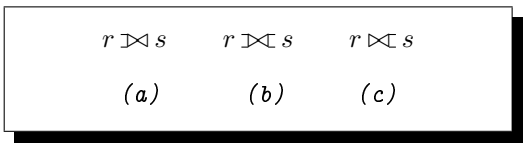
exemple 5.13. *Obtenir els noms dels caps dels treballadors.*
solució $\Pi_{\texttt{nom}}(\rho_x(\texttt{persona}) \bowtie_{\texttt{x.passaport=obeeix}} \texttt{treballador})$

En l'exemple 5.13 es calcula `persona` $\times$ `treballador`, donant set columnes, i llavors se seleccionen els noms de totes les persones que apareixen en el valor de l'atribut `obeeix` d'alguna tupla de `treballador`. El renomenament s'hagués pogut evitar posant `persona.passaport` en l'argument θ enlloc d'`x.passaport`. L'exemple 5.14 és molt semblant. Aquest cop amb la relació `coneix`.

exemple 5.14. *Noms de les persones conegudes per algú.*
solució $\Pi_{\texttt{nom}}(\rho_x(\texttt{persona}) \bowtie_{\texttt{x.passaport=coneguda}} \texttt{coneix})$

Reunions externes

Hi ha tres operacions de reunió externa. Per la dreta, per l'esquerra i completa. L'esquema de totes és el mateix que el de la reunió interna. Es diferencien en la quantitat de tuples de la relació resultant. En la Caixa 5.16 es mostren els signes que utilitzen. A l'esquerra, Caixa 5.16*(a)*, la reunió externa per l'esquerra, *left outer join*. En la Caixa 5.16*(b)* la reunió externa completa, *full outer join*, i en la Caixa 5.16*(c)* la reunió externa per la dreta, *right outer join*,



Caixa 5.16. *Expressió relacional de les reunions externes. (a) per l'esquerra. (b) completa. (c) per la dreta.*

sent r i s dues relacions.

Donada una reunió interna entre dues relacions r i s, o sigui, una relació formada pels atributs d'r i els d's, i per les tuples que satisfacin el predicat θ, la idea de les reunions externes és útil quan es vol mantenir la informació de totes les tuples d'alguna de les relacions d'entrada, encara que no hagin estat aparellades amb cap tupla de l'altra relació d'entrada. És a dir, si les columnes que es vinculin en el predicat θ fan que alguna de les tuples d'una relació desaparegui del resultat perquè no té tupla relacionada en l'altra, llavors fem que aparegui en el resultat aparellada amb valors nuls com a tupla relacionada. Això fa que haguem d'indroduir nuls.

La relació resultant d'una reunió externa per l'esquerra consta de totes les dades de la primera de les dues relacions d'entrada, més les de la segona que estiguin vinculades amb alguna de la primera. Pels valors de la Taula 5.7 la relació resultant es mostra en la Taula 5.10.

nom	edat	ciutat	ciutat	país
gina	35	París	París	França
josep	35	París	París	França
mireia	35	Berlín	Berlín	Alemanya
pere	23	Roma	Roma	null

Taula 5.10: *Reunió externa per l'esquerra, left outer join, entre* `persona` *i* `ciutat` *de la Taula 5.7.*

Amb una reunió externa completa no es perd cap dada de les relacions d'entrada. Això fa que quedin valors nuls en totes les columnes no vinculades, com es pot veure la Taula 5.11.

nom	edat	ciutat	ciutat	país
gina	35	París	París	França
josep	35	París	París	França
mireia	35	Berlín	Berlín	Alemanya
pere	23	Roma	Roma	null
null	null	Lisboa	Lisboa	Postugal

Taula 5.11: *Reunió externa completa, full outer join, entre* `persona` *i* `ciutat` *de la Taula 5.7.*

I la reunió externa per la dreta és la versió simètrica de la de l'esquerra. Per això té totes les tuples de la segona relació. En la Taula 5.12 es pot observar el resultat de la reunió externa per la dreta de les relacions de la Taula 5.7.

nom	edat	ciutat	ciutat	país
gina	35	parís	parís	frança
josep	35	parís	parís	frança
mireia	35	berlín	berlín	alemanya
null	null	lisboa	lisboa	postugal

Taula 5.12: *Reunió externa per la dreta, right outer join, entre* `persona` *i* `ciutat` *de la Taula 5.7.*

5.5.3 Divisió de Relacions

El signe per l'operació de divisió entre dues relacions és el de la Caixa 5.17

$$r \div s$$

Caixa 5.17. *Expressió relacional de la divisió entre dues relacions.*

sent r i s dues relacions tals que els atributs d's són un subconjunt propi dels atributs d'r. A r se li diu relació divivend, a s se li diu relació divisora, i al resultat se li pot dir relació quocient.

La relació resultant d'una divisió té per esquema els atributs d'r que no estan a s. El seu contingut són les parts corresponents de les tuples d'r que, en r, estiguin combinades amb cada una de les tuples d's.

Aquesta operació serveix per consultes que demanin un element d'una relació que estigui relacionat amb tots els elements d'una relació corresponent a una expressió relacional que es construeix en la mateixa consulta.

Per entendre'ns, mirem el cas més senzill, r de dos atributs i s d'un de sol. Diguem $r = r(A, B)$, i $s = s(B)$. Llavors el resultat d'r dividit per s tindrà només la columna A, i tindrà un fila per cada valor d'A que en r aparegui juntament amb totes i cadascuna de les tuples d's.

Probablement, la manera més senzilla de recordar-ho és que el resultat de la divisió és aquella relació tal que multiplicada per s en resulti una part gran d'r, ja que, atenció, aquelles tuples d'r en les quals el valor de l'atribut A no estigui combinat amb cada una de les tuples d's no tenen cap impacte en la divisió.

No és una operació bàsica, tal com es demostra en la Caixa 5.18 per uns esquemes d'r i s genèrics.

Si $r = r(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$
i $s = s(B_1, B_2, \ldots, B_m)$

$$r \div s = \Pi_{A_1, A_2, \ldots, A_n}(r \bowtie s)$$

Caixa 5.18. *La divisió no és una operació bàsica.*

.

L'exemple 5.15 és planteja amb l'exemple del club esportiu, del Model 5.11.

exemple 5.15. *Noms dels socis que fan tots els esports que hi ha al club.*

solució $\Pi_{\texttt{nom}}(\texttt{persona} \times (\Pi_{\texttt{passaport,esport}}(\texttt{fa}) \div \Pi_{\texttt{esport}}(\texttt{esport}))$

Per pensar la solució de l'exemple 5.15 la primera cosa és adonar-se que per resoldre la consulta cal obtenir en algun lloc la relació de tots els esports, que farà el paper de divisora és a dir, d's. La relació de tots els esports és $\Pi_{\texttt{esport}}(\texttt{esport})$.

Un cop disposem de la relació divisora, ens hem de preocupar d'aconseguir la relació dividend, és a dir, la que tingui en alguna columna l'atribut que serveixi per conseguir la sortida, i en alguna altra columna els valors de la relació divisora, que en l'exemple 5.15 és `fa`.

En aquest mateix exemple 5.15 l'atribut que obtenim de la divisió és `passaport`. Per això acabem multiplicant-lo per persona i així resoldre el nom tal com es demana.

5.5.4 Assignació de Relacions

El símbol que representa una assignació és a la Caixa 5.19

$$r \leftarrow s$$

Caixa 5.19. *Expressió per l'assignació de relacions.*

sent s una relació, i r a partir d'aquest moment, també.

El resultat d'aquesta operació és una nova relació amb el mateix contingut que la donada d'entrada.

Utilitzant aquesta operació considerarem que les relacions són variables que poden prendre per valor altres relacions, o més en general, qualsevol expressió relacional, és a dir, qualsevol resultat d'una operació de l'àlgebra. Un cop assignat un resultat a una relació es pot fer servir de nou com relació d'entrada a altres operacions, modularitzant així la feina de descripció d'una consulta a la base de dades.

En la Caixa 5.20 s'assigna una relació d'una única tupla amb els valors (*'jordi',24,'amsterdam'*) a la relació `persona(nom,edat,ciutat)` dels exemples anteriors. Observeu que els valors de la tupla pertanyen als dominis de la relació. Hi ha una correspondència posicional. O sigui, *josep* és el `nom`, el *24* és l'`edat`, i *amsterdam* la `ciutat`.

```
persona ← {('josep',24,'amsterdam')}
```

Caixa 5.20. *Assignació d'una relació amb una sola tupla a la relació* `persona`.

Els parèntesis estableixen que els valors van en aquest ordre. I l'expressió {(*'josep',24,'amsterdam'*)} és una relació que tan sols conté una tupla. O sigui que cada una de les tuples sí que està ordenada interiorment i per això es descriu amb parèntesis. En canvi la relació és un conjunt, i com a tal no ordenat, de tuples, cosa que s'expressa amb les claus. En definitiva, la relació donada com operand d'entrada en la Caixa 5.20 és un conjunt definit per enumeració segons allò vist en la Secció 1.1.2.

Tot seguit es desgrana en etapes l'exemple 5.16 de la Secció 5.4.6, que es definia sobre la relació `persona = persona(nom,anys,lloc)`. En el exemple 5.16 és notori el fet que després d'una sèrie d'assignacions, s'acaba donant una expressió com a solució.

exemple 5.16. *Obtenir el nom de la persona més gran.*

solució

$$r_1 \leftarrow \Pi_{\texttt{nom}}(\texttt{persona})$$
$$x \leftarrow \texttt{persona}$$
$$y \leftarrow \texttt{persona}$$
$$r_2 \leftarrow \sigma_{\texttt{x.anys}<\texttt{y.anys}}(x \times y)$$
$$r_1 - r_2$$

5.6 Àlgebra Relacional Estesa

Considerem que les operacions següents formen part de l'àlgebra relacional estesa pel fet de tenir en compte els tipus dels atributs.

5.6.1 Projecció Generalitzada

Podem generalitzar l'operació de projecció vista en la Secció 5.4.2 amb la notació de la Caixa 5.21

$$\Pi_{F_1(A_1),F_2(A_2),\ldots,F_n(A_n)}(r)$$

Caixa 5.21. *Projecció generalitzada dels atributs d'una relació.*

L'operació de projecció generalitzada retorna la relació formada per les transformacions segons les funcions $F_1, F_2, \ldots, F_n$ dels atributs de la relació d'entrada.

exemple 5.17. *A partir de la relació* `fa = fa(passaport,esport,quota)`*, obtenir la quantitat d'impostos, 21%, que paga cada soci per cada esport.*

solució $\Pi_{\texttt{passaport,esport,0.21*quota}}$ (`fa`)

exemple 5.18. *A partir a la relació* `persona(nom,edat,ciutat)`*, per cada persona de parís, inserir-ne una altra amb el mateix nom, però 5 anys més gran.*

solució `persona` $\cup\ \Pi_{\texttt{nom,edat+5,ciutat}}(\sigma_{\texttt{ciutat='parís'}}$ (`persona`))

D'aquests exemples se'n desprèn que les funcions de la projecció generalitzada poden involucrar atributs de la relació donada, i també valors constants, cosa que ens obre la possibilitat d'utilitzar l'àlgebra per fer operacions aritmètiques, com en l'exemple 5.19

exemple 5.19. *Calcular la suma de 2 + 3.*

solució Π_{2+3}

on com es pot observar ni tan sols li passem cap relació d'entrada.

De la mateixa manera, una nova versió per l'assignació de la Caixa 5.20 s'exposa en la Caixa 5.22.

$$\texttt{persona} \leftarrow \Pi_{\texttt{'josep',24,'amsterdam'}}$$

Caixa 5.22. *Nova versió de l'assignació de la Caixa 5.20.*

Aquestes projeccions d'atributs constants en el valor de totes les tuples del resultat potser queden més clares veient com projectar la constant en més d'una tupla.

Posem per cas, l'expressió

$$\Pi_{\texttt{nom},200}(\texttt{persona})$$

retornaria una relació formada de tantes files com tingui la taula persona, i de dues columnes. Una amb el nom, i la segona constant igual a 200 en tota la taula. És la relació de la Taula 5.13.

nom	?
gina	200
josep	200
mireia	200
pere	200

Taula 5.13: *Resultat de la projecció d'un atribut constant.*

Noteu que la relació resultant té un atribut amb nom desconegut. O sigui, per reutilitzar-la caldria donar-li algun nom amb l'operació del canvi de nom.

5.6.2 Valors Nuls

Un valor *null* és un valor inexistent en una relació. O sigui en alguna tupla i per algun atribut d'una relació. Això és ambigu, perquè tant pot ser que el valor sigui inexistent perquè no es coneix però podria estar disponible més endavant, com ser *null*s estructurals que mai podran adquirir un valor.

En qualsevol cas, com s'ha vist en seccions anteriors, els valors dels atributs es fan servir en les comparacions de les seleccions. I per tant, cal deixar ben clar els resultats que esperem d'aquestes comparacions amb valors nuls.

La filosofia és posposar la interpretació de *null* com a fals lògic tant com es pugui. És a dir, mentre poguem operar amb el valor *null*, si tenim la sort que desaparegui en els càlculs intermitjos, millor. I si no, si al final de l'avaluació d'un predicat ens resulta que val *null*, llavors considerarem que és fals.

El llenguatge estructurat de consultes, SQL, té un mecanisme per discernir si el resultat d'una expressió booleana és fals realment, o és fals degut a que l'avaluació ha donat com a resultat un valor nul.

Qualsevol comparació d'un atribut amb *null* donarà *null*. És a dir, si en un predicat hi ha proposicions que un cop avaluades acaben calculant si $A < null$, $A = null$, o $A > null$, llavors el resultat d'aquestes proposicions és *null*. I per tant si són l'única proposició del predicat, fals. Observeu que en particular, el predicat $null = null$ és fals, com també ho és $null \neq null$.

Amb l'SQL podem preguntar si el valor d'un atribut és nul. Però alerta, no amb el comparador d'igualtat. Fer-ho amb el comparador d'igualtat és un error freqüent. I clar, sempre torna fals, encara que el valor comparat no sigui nul.

El valor *null* per les operacions aritmètiques es comporta d'una manera absorvent. És a dir, l'expressió $5 + null$ és *null*.

Lògica de tres valors

Per descomptat que sempre que els valors nuls per les proposicions puguin ser absorvits per l'Àlgebra de Boole, millor. Les taules de veritat per aquesta lògica de tres valors són.

p	q	$p \wedge q$
fals	null	fals
cert	null	null

(a)

p	q	$p \vee q$
fals	null	null
cert	null	cert

(b)

p	$\neg p$
null	null

(c)

Taula 5.14: *Taules de veritat amb valors nuls. (a) Conjunció. (b) Disjunció. (c) Negació*

És a dir, per la conjunció lògica, el valor nul es comporta com si fos `fals`. Per la disjunció, `cert`. I com a cosa nova, `no` *null*, és *null*.

Filosòficament, des d'un punt de vista més essencial, una de les regles que sembla més qüestionable de la deducció natural és que fals sigui igual a fals.

5.6.3 Funcions d'Agregació

Les funcions d'agregació són funcions a les quals se les alimenta amb una col·lecció de valors, i ens en retornen un de sol, com el recompte, la suma, la mitja, o la desviació tipus, que vé a ser la mitja de distàncies a la mitja.

És cosa bona interpretar el terme *agregar* com una unió irreversible. Com una fusió. Quan en l'operació booleana de suma lògica u més u fa u, llavors més que una suma, l'operació es comporta com una agregació. I també en aquest sentit, quan els organismes administratius d'estadística volen preservar la confidencialitat de la informació, donen les dades agregades.

Aquesta mateixa interpretació és la que es fa en el model ER quan incorporem una clau primària a una relació M:N convertint-la en entitat. En aquest cas estem agregant una parella, o més, de claus foranes en una sola clau.

És crucial adonar-se'n que una agregant un atribut es redueix el nombre de files d'una relació per l'atribut agregat a una sols tupla.

El format de les funcions d'agregació, en la seva versió més senzilla, s'indica en la Caixa 5.23

$$\mathcal{G}_{f(A)}(r)$$

Caixa 5.23. *Expressió relacional mínima per les funcions d'agregació.*

en el benentès que l'atribut A pertany a l'esquema d'r. El símbol $\mathcal{G}$ es diu G cal·ligràfica, i fa de mnemotècnic pel terme *grup*, o *agrupació*. S'utilitza passant-li com argument la funció d'agregació que volem calcular, i entre parèntesis l'atribut sobre el que volem fer el càlcul.

La relació resultant d'operar amb funcions d'agregació són d'una sola tupla i un sol atribut, sense nom.

Per incorporar aquestes funcions a l'àlgebra cal distingir atributs numèrics, i atributs textuals, per això estem dins l'àlgebra estesa.

Les funcions definides sobre atributs textuals són el recompte, el mínim i el màxim. El recompte retorna el nombre de valors diferents en l'atribut donat a l'operació com a paràmetre de la funció. El mínim i el màxim retornen el primer i l'últim valor ordenats alfabèticament segons la taula ASCII, que es mostra en l'Apèndix B. Els noms d'aquestes funcions són `count(A)`, `min(A)`, i `max(A)`, sent `A` l'atribut sobre el que es calculen.

Les funcions definides sobre atributs numèrics són les de les textuals, i a més les aritmètiques, o estadístiques. La suma i la mitjana. Els seus noms són `sum(A)`, `avg(A)`.

Veiem exemples a partir de les relacions del Model 5.13 per al club esportiu.

```
persona(passaport,nom,cognom,mail,ciutat)
soci(passaport,alta)
esport(esport,preu,jugadors)
fa(passaport,esport,quota)
```

Model 5.13. *Fragment del Model 5.11 per als exemples de les funcions d'agregació.*

exemple 5.20. *Quants socis hi ha al club?*
solució $\mathcal{G}_{\texttt{count(*)}}(\texttt{soci})$

La solució de l'exemple 5.20 comptaria el nombre de socis del club. L'ús de l'asterisc enlloc de l'atribut significa que es vol comptar el nombre de tuples de la relació donada.

En els exemples 5.21, i 5.22 El producte cartesià serveix per només tenir en compte els números de passaport que es corresponguin a socis.

exemple 5.21. *Quants socis hi ha al club que es diguin* joan*?*
solució $\mathcal{G}_{\texttt{count(*)}}(\sigma_{\texttt{nom='joan'}}(\texttt{persona}) \times \texttt{soci})$

exemple 5.22. *Com es diuen el primer i l'últim soci, alfabèticament, del club?*
solució $\mathcal{G}_{\texttt{min(nom),max(nom)}}(\texttt{persona} \times \texttt{soci})$

exemple 5.23. *Quants diners entren cada mes al club?*
solució $\mathcal{G}_{\texttt{sum(quota)}}(\texttt{fa})$

En els exemples anteriors es fa l'agregació en base a la relació donada completa. I en qualsevol d'ells el resultat és una sola tupla.

Hi ha casos que interessa agregar resultats segons els valors concrets d'un atribut, o d'un conjunt d'atributs, que en diem *criteri d'agregació*. Alerta, això provocarà que per cada valor del criteri d'agregació es crearà una tupla a la relació de sortida. En el fons, estem segmentant el resultat de l'operació agregada que es faria sobre la relació completa, segons els diferents valors del criteri.

Cal ser conscients d'això. Si pretenem afegir atributs addicionals en el resultat d'una funció d'agregació, els càlculs es realitzaran per cada un dels valors, o cada combinació de valors, d'aquests atributs addicionals. Per això, aquests atributs no agregats han de ser agrupats. És una norma tan obligatòria, que com es veurà en SQL es converteix en norma sintàctica. No es pot fer d'altra manera.

Per agregar valors segons criteris d'agregació, cal posar aquests atributs abans de la $\mathcal{G}$. En la Caixa 5.24 es mostra la versió més sofisticada d'una consulta d'agregació.

$$A_{k+1}, A_{k+2}, \ldots, A_n \ \mathcal{G}_{f_1(A_1),f_2(A_2),\ldots,f_k(A_k)}(r)$$

Caixa 5.24. *Expressió relacional per les funcions d'agregació.*

Els atributs $A_{k+1}, A_{k+2}, \ldots, A_n$ formen el criteri d'agregació. De manera que es pot entendre que un valor del criteri d'agregació és una combinació de valors dels atributs que el formen.

L'expressió de la Caixa 5.24 obtindrà una tupla per cada valor del criteri d'agregació, que és cada combinació de valors dels atributs $A_{k+1}, A_{k+2}, \ldots, A_n$. Per tant, segons els atributs que s'utilitzin com a criteri d'agregació variarà el nombre de tuples en la relació resultant.

En els primers k atributs, la tupla ha de tenir el resultat d'agregar els diferents valors de cada A_i amb la corresponent funció f_i, per $i = 1, \ldots, k$.

exemple 5.24. *Quant ingressa el club per cada esport?*

solució $\texttt{esport}\ \mathcal{G}_{\texttt{sum(quota)}}(\texttt{fa})$

En l'exemple 5.24 el criteri d'agregació és `esport`. La relació resultant tindrà tantes tuples com esports, fets per algun soci, que hi hagi en el club. Si hi ha algun esport que no el fa ningú se suposa que el club no ingressa ni un ral per aquest esport.

exemple 5.25. *Quant paga cada soci?*

solució $\texttt{passaport}\ \mathcal{G}_{\texttt{sum(quota)}}\ (\texttt{fa} \bowtie \texttt{soci})$

En l'exemple 5.25 s'agrega segons el passaport sumant la quota de tots els esports que faci.

En l'SQL s'ha pres la decisió d'ignorar en tots els sentits els valors nuls a l'hora d'interpretar les funcions d'agregació. Així com l'expressió `5` + *null* és *null*, si fem la suma agregada d'un atribut numèric, A, en una relació r de dues tuples amb un `5` a la primera i nul a la segona, llavors la $\mathcal{G}_{sum(A)}(r)$ retornarà una tupla amb el valor de l'atribut igual a `5`.

En aquest capítol hi ha hagut dues parts. Primer, tot allò que fa referència a la creació de les bases de dades. Això ha fet que ens desviéssim una mica de l'àlgebra relacional per assentar el concepte de model relacional com a punt de partida en la implementació de les bases de dades. Després, s'ha recuperat el fil de l'àlgebra relacional, però ja no amb la intenció de veure la manera de definir les relacions sinó focalitzant-se en la manera de treballar amb les relacions definides segons la primera part. De la primera part, el concepte més important es sens dubte el d'esquema de relació. De la segona, les operacions, el producte cartesià, la reunió interna, i la projecció.

Capítol 6

Llenguatge Estructurat de Consultes

El llenguatge estructurat de consultes, o SQL d'*Structured Query Language*, és sens dubte un dels èxits d'estandardització més notables en la història de la informàtica. Segurament el més antic dels llenguatges de programació que se segueixen utilitzant profusament en l'actualitat. És dels anys setanta, anterior a la programació orientada o objectes. Això fa que hi hagi coses que no s'entendrien si no es posen en context. És un llenguatge que va aparèixer quan la programació en llenguatges d'alt nivell era majoritàriament imperativa. Quan va començar, va impressionar la comunitat científica. I aquesta intenció pretenciosa es reflecteix en moltes de les seves característiques. Així mateix, tot plegat ha provocat l'aparició de diferents dialectes, de manera que en moltes qüestions és difícil averiguar si el llenguatge que s'està utilitzant és estàndar completament, o s'està fent ús de característiques específiques de l'SGBD.

En tot aquest llibre no es considera cap aplicació d'interfície client per desenvolupar la base de dades. No convé. Fan coses que cal que aprenguem a fer nosaltres com a responsables de l'aplicació. Prenen moltes decisions de maneres genèriques, que embruten el codi, i en cap cas l'optimitzen. Com es deia al final del preàmbul, desconfiarem de les tendències, i procurarem mirar-les des d'un punt de vista crític.

Bàsicament l'SQL es divideix en dos llenguatges. El llenguatge que es dedica a la definició dels espais i a l'establiment de les restriccions, i el que es focalitza en l'explotació, o sigui inserció, modificació, esborrat, o consulta de les dades.

El capítol obre amb una introducció a l'entorn de treball PostgreSQL, que servirà per poder posar en pràctica els dos llenguatges que després es presenten.

6.1 Entorn de Treball

Com que el que queda per endavant serà una pila de proves, és important, encara que no imprescindible, tenir instal·lat el SGBD PostgreSQL, [7]. El codi que es presenta al llarg de les seccions restants es pot escriure amb qualsevol editor de text, dins la carpeta del projecte.

Recordeu del Capítol 2, que es treballa sobre una arquitectura client servidor tal com es mostra en la Figura 6.1. De totes maneres, per poder fer els experiments amb un sol ordinador, podem aconseguir-ho muntant una màquina virtual, o també connectant-nos directament al *localhost*.

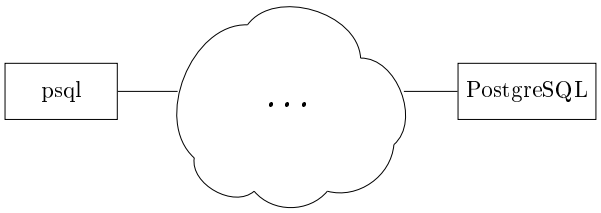


Figura 6.1: *Arquitectura de l'entorn de treball*

És a dir, la gestió de bases de dades utilitza com a mínim dos programes.

Un és el propi SGBD que resideix al servidor, i com a servei que és, està sempre infatigablement actiu en memòria principal esperant amb paciència sol·licituds pels ports de comunicació, que amb el PostgreSQL per defecte és el 5432. L'ordinador servidor no s'atura mai, o gairebé mai. Quan es mesura la disponibilitat de les bases de dades és fa amb nous. Un SGBD amb una disponibilitat 99 vol dir que està actiu el 99% del temps. I els més ben preparats en aquest sentit tenen quatre nous, és a dir que estan un 99.99% del temps actius. Un SGBD amb una disponibilitat 9999 es para cinc minuts al més com a màxim, per tasques de manteniment. Observeu que la mesura de disponibilitat és logarítmica, o sigui que si millora un 9 vol dir que el temps de no disponibilitat es divideix per deu.

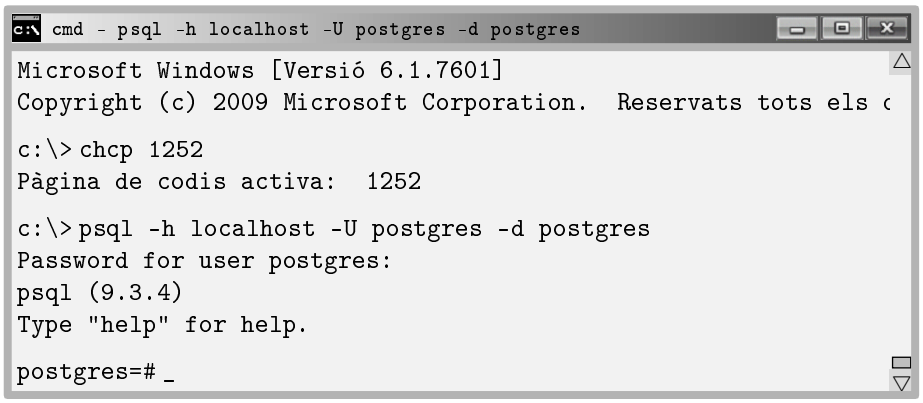
L'altre és el programa client, que des de qualsevol ordinador pot connectar-se donant la ip del servidor, com si truqués per telèfon. El programa que s'utilitzarà per connectar-se al servei es diu `psql`, que és el client natiu de postgres. És un programa en línia de comandes, o sigui sense interfície gràfica. Així ens concentrarem en l'SQL, i prendrem consciència de què es pot fer, i què no cal.

Un cop descarregat de [7] i instal·lat, podem comprovar que tot està bé abans de començar invocant directament el client amb la comanda

```
psql -h localhost -U postgres -d postgres.
```

És necessari que des de qualsevol lloc es pugui usar el `psql`, cosa que en windows significa tocar el path. Si en les sessions interactives interessa poder veure els caràcters de vuit bits, cosa que no té més transcendència, es poden ajustar els codis de pàgina fent `chcp 1252`.

Tot plegat es mostra en la Pantalla 6.1. Les pantalles d'exemple han estat capturades sobre el sistema operatiu Windows de Microsoft. Això no obstant, els exemples es poden reproduir sobre terminals de sistemes línux, ja sigui Ubuntu, Debian, Red Hat... etcètera.



Pantalla 6.1. *Inici de sessió.*

Aquesta comanda arrenca el client `psql`, que es connecta a l'ordinador servidor, altrament dit *host*, segons el paràmetre `-h`, amb l'usuari que digui el paràmetre `-U`, en aquest cas l'usuari *postgres*, a la base de dades que indiqui el paràmetre `-d`, que en aquest cas també és *postgres*. Aquest usuari es crea durant la instal·lació, i la base de dades postgres també. En PostgreSQL, sempre que un usuari està connectat, ho està en alguna base de dades del clúster. En canvi en MySQL, no.

Tots els paràmetres tenen el seu valor per defecte. Si no es dóna el host, `psql` prèn *localhost* com a valor per defecte. Si no es dóna l'usuari, `psql` pren l'usuari del sistema operatiu. I si no es dóna el tercer paràmetre, llavors el `psql` pren per defecte com a nom de la base de dades el nom de l'usuari.

Si en la instal·lació hem donat contrasenya per l'usuari postgres, cal entrar-la tot seguit.

Diem *prompt* al símbol que els programes en línia de comandes com el `psql` posen a principi de línia indicant que esperen alguna entrada de dades. Pel cas que ens ocupa, el prompt es composa de tres parts. Com es pot veure a la Pantalla 6.1, és `postgres=#`. Això vol dir:

- `postgres` és el nom de la base de dades actualment connectada.
- `=`, el signe d'igual significa que l'entrada actual no ha començat. Proveu a obrir un parèntesi i intro. El prompt canvia a `postgres(#` indicant que tenim un parèntesi obert en aquesta comanda, i mentre no el tanquem, s'estaran produint errors en qualsevol entrada. El mateix passaria si enlloc del parèntesi polsem un apòstrof per exemple. Ara bé, si teclejem qualsevol cosa que no comenci amb \, i premem intro, llavors el prompt es transforma en `postgres-#`, o sigui un guió enlloc de l'igual. Això significa que està esperant un punt i coma per acabar el que se suposa que és una comanda SQL. Premeu ; i després d'un missatge d'error rescuperareu l'estat normal.
- `#`, el coixinet indica que l'usuari connectat té permisos de superusuari, com en els sistemes línux. Pels altres tipus d'usuari, altrament dit per altres rols, el prompt té un signe de major enlloc del coixinet, `postgres=>`.

Un cop connectats al servidor com es mostra en la Pantalla 6.1, tenim dos llenguatges possibles per utilitzar. Una de dues,

- O bé comencem la comanda amb una contrabarra, \, o sigui emetem una ordre dirigida al client, `psql`.
- O bé acabem la comanda amb un punt i coma, ;, que llavors el `psql` l'enviarà al PostgreSQL tot entenent que es tracta d'una comanda SQL.

O sigui una o l'altra, però no les dues. O comença amb la barra, o acaba amb el punt i coma. Proveu a consultar l'ajuda del `psql`, amb la comanda `\?` i l'ajuda de l'SQL fent `\h`. Si us en voleu anar a dormir, premeu `\q`.

6.1.1 Familiarització amb l'SGBD

Estem en situació de tocar les primeres tecles. Fem-ho.

Familiarització amb el `psql`

Podeu consultar les bases de dades que hi ha al clúster fent `\l`, que tot just instal·lat el PostgreSQL hauria de donar la base de dades actual anomenada *postgres*, i les dues plantilles que postgres utilitza. La *template0* és una base de dades buida, a partir de la qual es crea la plantilla *template1* que ha de servir d'origen quan es creen noves bases de dades dins aquest clúster.

També es pot consultar les relacions existents en la base de dades, amb `\d`, o els usuaris prement `\du`. La `d` d'aquestes consultes vol dir *display*. Tot això ho diu l'ajuda del `psql`.

Familiarització amb l'SQL

Un cop connectats i amb el cursor en la línia de comandes del PostgreSQL, resulta convenient familiaritzar-s'hi provant algunes consultes.

exemple 6.1. *Quant sumen 2 + 3 i quant multipliquen?*

solució
```
SELECT 2 + 3, 2 * 3;
```

Observeu que el PostgreSQL sempre retorna qualsevol informació en forma de taula, posant-hi `?column?` per defecte als noms dels atributs. Proveu a posar-li títols renomenant les columnes amb l'operador corresponent, `AS`, com s'indica tot seguit.

solució
```
SELECT 2 + 3 AS suma, 2 * 3 AS producte;
```

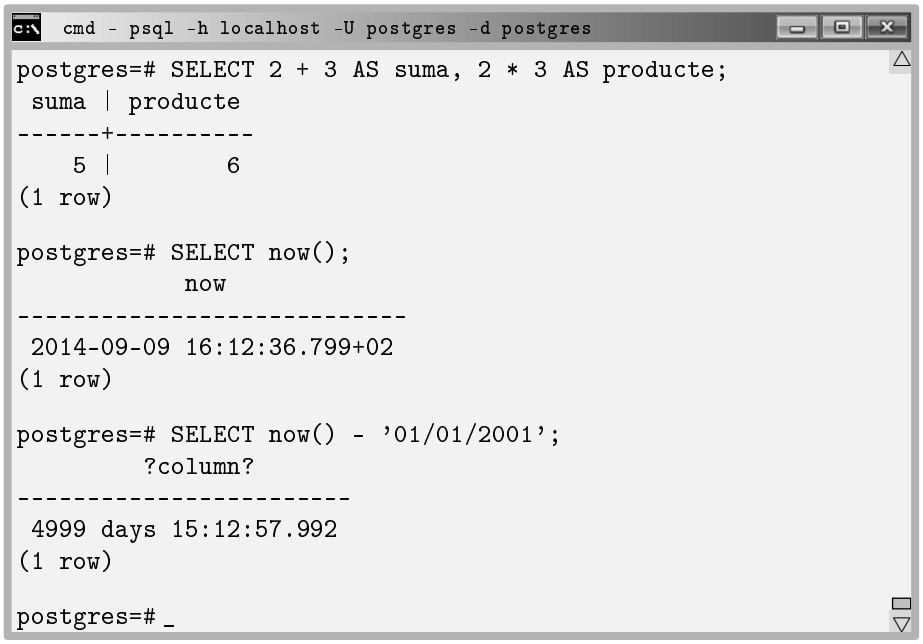
exemple 6.2. *Quina hora és?*

solució
```
SELECT now();
```

exemple 6.3. *Quants dies heu viscut?*

solució
```
SELECT now() - '01/01/2001';
```

Clar que la solució de l'exemple 6.3 és correcte per una persona que hagi nascut l'1 de gener de l'any 2001.



Pantalla 6.2. *Familiarització amb l'SQL de PostgreSQL.*

6.2 Llenguatge de Definició de Dades

El Llenguatge de Definició de Dades, o DDL de *Data Definition Language*, és la part de l'SQL que s'utilitza per construir una base de dades. I això vol dir proporcionar les dades necessàries per poder fer l'explotació.

6.2.1 Multiconjunts

L'SQL tracta les relacions com multiconjunts. O sigui, admet que les relacions tinguin elements repetits. Aquesta és la diferència més important entre l'àlgebra relacional i el llenguatge de consultes.

I per què?

Bé, igual que qualsevol magnitud numèrica dóna més informació que la simple existència de la magnitud, així com un valor enter sempre guardarà una informació més precisa que un booleà, convertim les relacions en comptadors. Enlloc de guardar-nos que un cosa passa, ens guardem quants cops passa la cosa. No és que ho haguem de fer sempre, però sí que ens reservem la possibilitat de fer-ho. L'àlgebra relacional no sap de magnituds numèriques, i per això no entra en aquestes distincions.

6.2.2 Taules

En endavant, utilitzarem el terme *taula* per indicar aquelles relacions físiques que guarden les dades reals d'una base de dades. Cada taula té com a mínim, un nom i un conjunt d'atributs, dels que a part del seus noms també hi ha constància dels seus dominis, que vol dir els tipus de valors que poden emmagatzemar.

Sembla doncs que es tracti del mateix concepte de relació que s'ha estat utilitzant en el Capítol 5. Però no és el cas.

Qualsevol expressió relacional es pot avaluar resultant-ne una relació, però no una taula. És a dir, el fet que la relació es guardi físicament en algun lloc fa que li diem taula. Observeu que en definitiva estem parlant de la persistència d'una relació. El fet d'emmagatzemar-se a disc sembla que no tingui més transcendència. Però sí que en té, igual que els fonaments dels edificis o les rels dels arbres, la part que toca terra serveix per suportar la lògica que veuen els nostres ulls.

I per tot plegat, entenem que la importància d'una relació va directament lligada a la seva persistència, cosa que estableix una classificació en les categories de les relacions, en funció de la seva volatilitat.

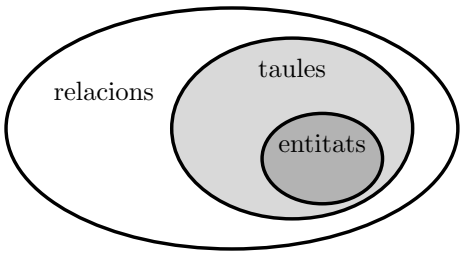


Figura 6.2: *Classificació de les relacions d'una base de dades.*

A partir del dibuix de la Figura 6.2 es pot deduir que

- totes les entitats d'un model ER s'implementen en taules, que són relacions que en particular tenen suport físic,
- totes les taules, entre les que també hi ha les restants del model relacional, són relacions que es guarden en suport físic.
- les consultes que es poden guardar en una base de dades són relacions, però no tenen un suport físic per les dades, es guarda l'expressió relacional que obté el resultat de la consulta, no la col·lecció de dades.

Les taules que constitueixen un base de dades són les que apareixen en el seu model relacional.

6.2.3 Metadades

Metadades es podria interpretar com alguna cosa que va més enllà de les dades. De fet, el prefix *meta*, a part de "més enllà", sovint convé entendre'l com autoreferència. Metamorfosi és quelcom que va més enllà de la forma, metallenguatge és aquella part del llenguatge que parla del mateix llenguatge. D'alguna manera, és com si anar més enllà tingués que veure amb l'autoreferència. I metadades vol dir aquelles dades que contenen informació sobre les dades. Les metadades en sí, són una base de dades. La base de dades de la base de dades.

Formen part de les metadades d'una base de dades, els noms de les relacions, els noms i els dominis dels atributs, les restriccions de clau primària i clau forana, restriccions d'existència i d'unicitat, i altres coses.

Els SGBDs guarden tota aquesta informació de la mateixa manera que ho guarden tot. És a dir, en taules. De vegades a les metadades se li diu catàleg de la base de dades.

Qui desenvolupa una base de dades no té cap interacció directa amb les metadades. Podria consultar-hi, per exemple el tipus d'un atribut d'una taula, però com que aquesta informació també la té al codi de creació de la taula, cosa que ha fet ella mateixa, acostuma a consultar abans el codi font.

Tot allò que forma part de les metadades passa automàticament a considerar-se objectes de la base de dades, i per tant, en PostgreSQL tenen un identificador d'objecte, OID. Això també vol dir que pot tenir un nom. És a dir, cada restricció, per exemple, pot tenir un nom.

6.2.4 Dominis dels Atributs

Per la creació de les taules corresponents al model relacional cal establir un tipus per cada atribut. Tot seguit es veu les diferents possibilitats per fer-ho.

Qualsevol persona que hagi desenvolupat aplicacions en llenguatges tipificats té consciència del que són els tipus de dades d'un llenguatge de programació. I convé tenir present que, en el fons, són les mides de les dades, en bytes. Aquesta és una de les abstraccions més notables en la pila de conceptes que constitueixen una màquina computadora. A una cosa que estrictament es correspon a una mida, se li diu tipus. És força interessant. És a dir, quan en un programa es diu que una variable és entera, al compilador tan sols l'interessa que ocupa quatre bytes.

Tipus primitius

Es diuen tipus primitius els que no tenen cap tractament intern associat, de manera que es corresponen senzillament amb l'espai que se'ls hi assigna, i qualsevol ús que se'n faci serà per mitjà de procediments implementats en algun llenguatge de programació.

- `BOOLEAN`
  Els atributs de tipus booleà poden valer *cert*, o *fals*. Ocupen un byte. I de fet, no acostumen a ser massa propensos a ser emmagatzemats en bases de dades, ja que normalment pel mateix preu podem guardar alguna informació més acurada. Tenir qualsevol valor significa existir, i per tant sempre és millor guardar el valor, que la simple existència d'ell.

- `CHARACTER`
  Aquest tipus també es pot anomenar `CHAR`. Rarament s'utilitzen atributs d'aquest tipus. Normalment el tipus caràcter és usat com element de les seqüències que són les cadenes de caràcters. Poden valdre qualsevol caràcter de la taula ASCII. Per tant ocupen un byte, però cal tenir en compte que dels 256 valors que poden prendre, només els 128 primers són

estàndar. I d'aquests, els 32 primers no són imprimibles, com ara el retorn del corró, el salt de línia o la tabulació. La taula de codis imprimibles estàndar es mostra a l'Appèndix B.

- `INTEGER`
  Números enters en l'interval $[0..2^{32}-1]$, ja que normalment ocupen quatre bytes, i igual que amb tres xifres decimals podem representar mil números, 10^3, amb 32 xifres binàries podem representar-ne 2^{32}. Això vol dir que si l'aplicació que els tracta els interpreta sense signe, el valor pot ser com a mínim 0 i com a màxim $2^{32}-1$. En canvi, si l'aplicació els interpreta amb signe, tenim la meitat del nombres positius que teníem. Llavors es podrà representar nombres enters en l'interval $[-2^{31}..2^{31}-1]$.

- `DECIMAL`
  Quan interessa emmagatzemar variables contínues, com ara pesos, o àrees en metres quadrats, llavors utilitzem aquest tipus de dades. Podem establir la quantitat de xifres que volem en total, i quantes darrera la coma. És a dir, és un tipus de dades parametrizat amb dues quantitats. Per exemple, el màxim valor de tipus `DECIMAL(5,2)` és el 999.99, ja que utilitza cinc xifres de les quals dues són decimals. El fet que un tipus de dades es preocupi de la seva representació impresa no és propi dels SGBDs, que en principi no tenen cap consideració relativa a l'entrada i sortida. Però bé, és una qüestió atàvica de l'SQL, i així és. Enlloc d'aquest tipus, també es pot utilitzar el `DOUBLE PRECISION`.

Tipus alfanumèrics

Els tipus alfanumèrics són seqüències de caràcters. Un valor constant d'aquest tipus va entre cometes simples o apòstrofs. Com ara 'comercial'. La concatenació de cadenes de caràcters utilitza dues barres verticals, *AltGr*+1, el símbol ||. O sigui, 'hola,'||' què tal' és igual a 'hola, què tal'.

L'SQL té l'operador `LIKE`, que usa patrons amb dos caràcters comodí. El guió baix i el tant per cent. El guió baix serveix per sustituir exactament una lletra. El tant per cent per sustituir-ne *n*, que pot ser no-res. Aquest operador retorna un booleà, cert si l'atribut encaixa al patró. Per exemple, *A* `LIKE` 'B_', retornarà cert sempre que el valor de l'atribut *A* sigui de dues lletres i comenci amb B majúscula. *A* `LIKE` '%*pepito*%' retornaria cert si el valor de la cadena *A* contingués la subcadena *pepito*.

- `CHARACTER(n)`
  També es pot anomenar `CHAR(n)`. Representa una seqüència de caràcters de longitud fixa que si ocupa menys d'*n* caràcters s'omple amb blancs per l'esquerra. Si n'ocupa més, es produeix un error.

- `CHARACTER VARYING(n)`
  També es pot anomenar `VARCHAR(n)`. Seqüència de caràcters de longitud

variable, sent n un paràmetre optatiu que si es dóna indica la màxima. I per defecte, si es declara un atribut d'una taula com `VARCHAR` sense el paràmetre, llavors la longitud màxima és `255`.

- `TEXT`
  Seqüències llargues de caràcters. En principi, la declaració d'aquest tipus fa que el valor es guardi en un espai diferent a la taula on pertany. De manera que a la taula tan sols hi ha una referència a algun lloc on es guarden els valors d'aquest tipus. En PostgreSQL però, no és el cas, de manera que és sinònim de `VARCHAR`.

Tipus temporals

Com diu la introducció del capítol, des del començament dels seus temps, l'SQL va marcar diferències respecte els altres llenguatges de programació per la seva proximitat extraordinària al llenguatge humà. En aquest sentit va incorporar tipus de dades per representar valors temporals juntament amb una sèrie d'operacions. Per treballar amb constants d'aquest tipus cal posar els valors entre cometes simples.

Els tipus de dades relacionats amb unitats de temps es detallen tot seguit.

- `DATE`
  Per representar dates. Si un valor d'aquest tipus val `'30/12/2014'` i li sumem `15`, llavors valdrà `'14/01/2015'`.
- `TIME`
  Per representar hores, minuts, segons, i milèsimes de segon. Un valor, `'13:15:12.123'`.
- `TIMESTAMP`
  Unió, o concatenació dels dos tipus anteriors. És a dir, un valor podria ser `'2014-12-30 15:15:12.123'`. A més, existeix la possibilitat de declarar-lo com a `TIMESTAMP WITH TIME ZONE`, que vol dir guardar addicionalment, dins la mateixa estructura de dades, un enter amb signe de `-12` fins a `+12`, que significa el canvi de fus horari respecte el meridià de greenwich. Un valor amb aquesta darrera variant és `'2014-12-30 15:15:12.123+02'`. Aquest tipus de dades és excepcional, perquè depèn del lloc on es representi pot variar el valor numèric.
- `INTERVAL`
  Representa la diferència entre dos dels valors dels tipus anteriors. Utilitza unitats textuals, com per exemple '`3` days'. És interessant per les operacions que proporciona.

És notori doncs que podem representar els dies en format `'21/01/2015'`, o bé `'2015-01-21'`. El primer dels formats perquè és al que estem acostumats. El segon

perquè és un format amb l'avantatge que si treiem els guions la xifra resultant és creixent amb el temps. És a dir, quan més cap el futur més gran resulta el número que en l'exemple seria `20150121`. Tot plegat, però, porta confusió sovint. Així que alerta.

Per qualsevol d'aquests tipus, els SGBDs proporcionen funcions d'extracció de les parts. Podem obtenir fàcilment l'any, el mes o el dia d'una data, i també saber a quin dia de la setmana correspon.

Tipus autoincrementals

De tipus autoincrementals només n'hi ha un. En cada SGBD té el seu propi nom. L'Access li diu *autonumérico*, el MySQL *autoincrement*, el PostgreSQL *serial*,...

El funcionament dels atributs declarats com autoincrementals és simple. El valor pròpiament dit és un enter positiu. I per cada nova inserció s'incrementa en una unitat. Llavors, la diferència entre tuples d'una relació que contingui un atribut autoincremental ve garantitzada pel propi SGBD, fent-lo inútil. La primera virtut d'un SGBD és el control d'existència i unicitat d'alguns dels seus valors. Si utilitzem atributs autoincrementals, estem malaguanyant aquest avantatge.

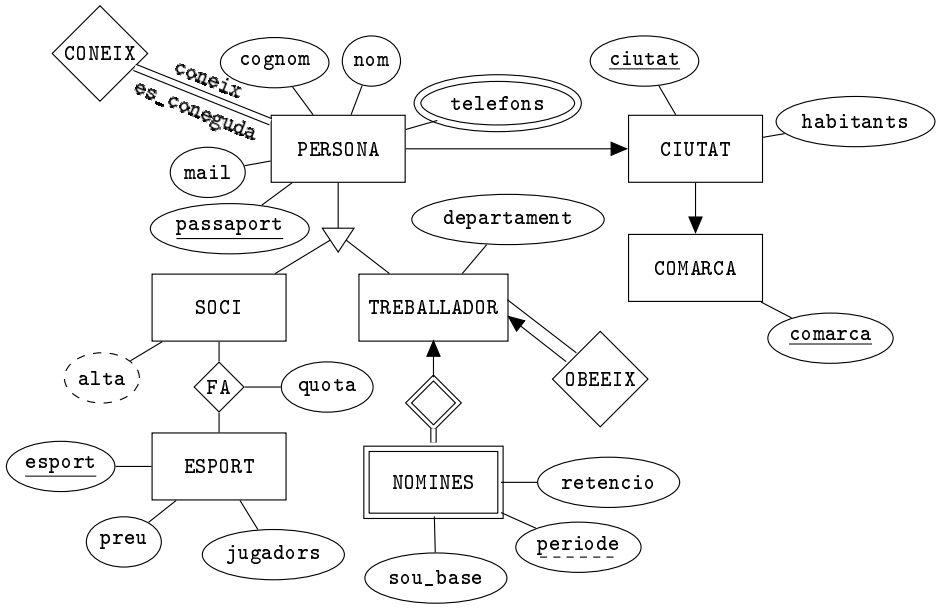
Si un atribut és de tipus autoincremental, llavors necessàriament ha de ser clau primària, i per tant, només pot haver-hi un atribut d'aquest tipus en cada relació. Si no és clau primària, ser autoincremental no té cap sentit. Les claus foranes que apunten claus primàries autoincrementals han de ser números enters.

Com que d'ençà que van sorgir les bases de dades qualsevol aplicació comercial ha fet gala del seu ús, hi ha hagut qui sense introduir-se en el coneixement de les bases de dades relacionals les ha utilitzat com si fossin simples sistemes d'arxius, fent aplicacions mal fetes. I per això serveix essencialment aquest tipus de dades. Si identifiquem a les persones per número de passaport, per exemple, no té cap sentit que el sistema permeti donar dues persones diferents amb el mateix número. Utilitzant autoincrements per claus primàries permetem accions que no tenen sentit. A més a més, en aquestes aplicacions mal fetes les tasques de verificació d'existència o d'unicitat queden en mans del programa client, recodificant la feina que l'SGBD faria amb tota seguretat de manera més eficient, i no és poca. En el món, corren barbaritats ingents d'aplicacions mal programades en aquest sentit, repetint malament moltes tasques, i diluint les responsabilitats de manera desordenada. Aquest és un extrem en que l'intrusisme professional és palmari.

El seu ús tan sols pot justificar-se en agregacions del model ER, sempre que el dissenyador no tingui prou imaginació per identificar la relació entre dues o més entitats amb alguna paraula més semàntica, ja que la característica més important dels atributs autoincrementals és que de semàntica, zero.

6.3 Construcció de la Base de Dades

En aquesta secció s'escodrinya minuciosament la implementació d'una base de dades des del no res. Per això, recuperem el disseny de la base per al club esportiu del Capítol 5 en el Model 6.1.



Model 6.1. *Model ER de la base de dades.*

En el mateix Capítol 5 aquest model ER ha estat tranformat en el model relacional 6.2.

```
comarca(comarca)
ciutat(ciutat,habitants,comarca)
persona(passaport,nom,cognom,mail,ciutat)
telefons(passaport,telefon)
coneix(coneix,es_coneguda)
soci(passaport,alta)
treballador(passaport,departament,obeeix)
esport(esport,preu,jugadors)
fa(soci,esport,quota)
nomines(passaport,periode,sou_base,retencio)
```

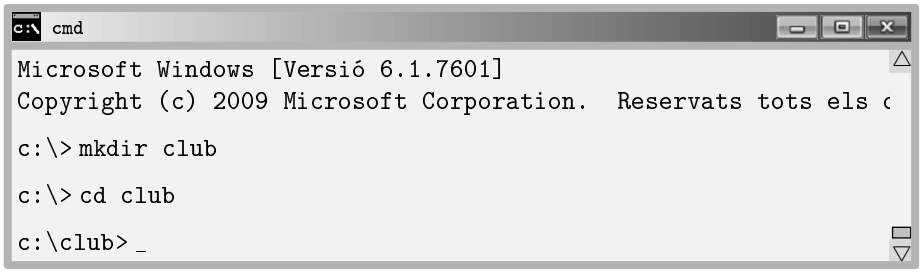
Model 6.2. *Model Relacional del disseny del Model Entitat Relació 6.1.*

Bé, comencem. La primera cosa que es fa és donar una paraula al projecte. L'anomenem `club`, que ja sabem que vol dir club esportiu.

6.3.1 Espai de Treball

Fem un directori nou amb nom `club`, on hi posarem tots els arxius relacionats amb el projecte. Un directori és una carpeta. Per crear-la, podem fer-ho amb qualsevol interfície gràfica del sistema operatiu. Tot i així, com que seguidament haurem d'obrir una terminal i situar-nos dins, també podem crear la carpeta des de la mateixa consola. Utilitzem els termes consola i terminal com a sinònims. En aquest cas teclejarem la comanda *make directory*, `mkdir`. I un cop creat, establirem aquest directori per defecte, o sigui ens hi ficarem dins, amb la comanda de *change directory*, `cd`.

Fins aquí, doncs, tot el que hem fet es pot veure en la Pantalla 6.3.



Pantalla 6.3. *Carpeta del projecte club.*

6.3.2 Escript Principal

Anomenarem *escripts* als arxius de text pla, de caràcters de set bits i amb extensió `sql` que escriurem. Les lletres accentuades, enyes i ces trencades no són caràcters de set bits, són de vuit, i per tant, no utilitzarem aquests caràcters ni en el nom ni en el codi dels escripts. Podran aparèixer, això sí, en cadenes constants de caràcters, ja siguin dades, o missatges que s'enviaran finalment a l'aplicació client perquè els pugui mostrar.

Dins el cicle de vida de l'aplicació, en la fase de desenvolupament remuntarem del no res la base de dades en cada canvi que pugui suposar un impacte en l'estructura. Per això, la primera cosa que fa l'escript principal és eliminar la base de dades del sistema i tornar-la a crear. En la Caixa 6.1 es mostra el contingut de l'arxiu `club.sql`.

```
\echo escript principal per la base de dades club
\echo ------------------------------------------
\c postgres
DROP DATABASE club;
CREATE DATABASE club;
\c club
```

Caixa 6.1. *Contingut inicial de l'arxiu* `club.sql`*, escript principal.*

El contingut de la Caixa 6.1 es justifica tot seguit línia a línia. Per posar-se en situació, cal entendre que executem aquest escript estant connectats a la base `club`, que a part del primer cop, passarà sempre.

- `\echo` *missatge* serveix perquè el `psql` imprimeixi per pantalla el text que segueix al nom de la comanda fins a final de línia. De passada, també es útil com a comentari en el mateix escript. Aquesta és la funció de les dues primeres línies.

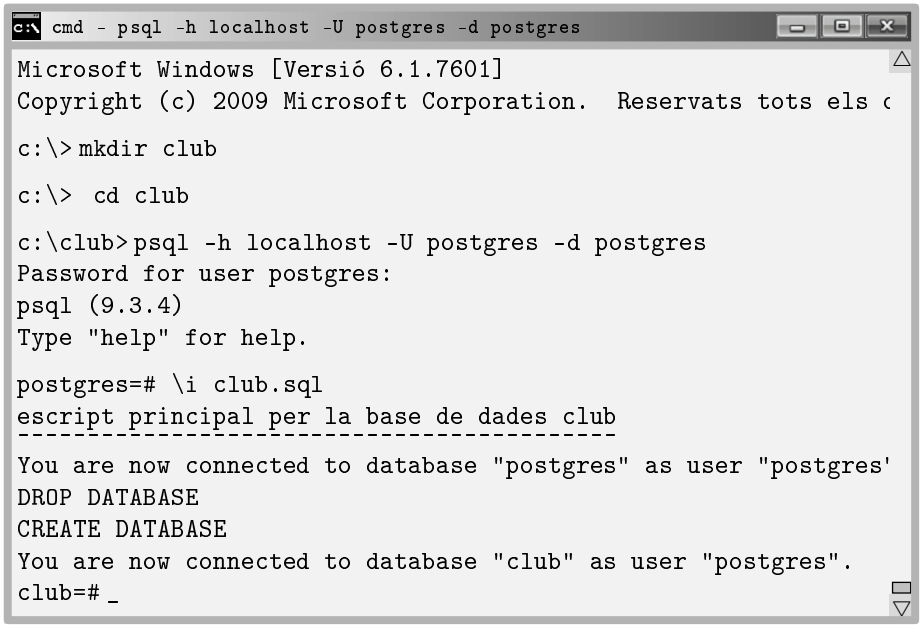
- `\c postgres` serveix per connectar-se a la base de dades *postgres*. La intenció és deixar d'estar connectats a la base actual, `club`, per poder-la esborrar, ja que no es pot borrar la base activa. Observeu doncs l'ús de la comanda `\c` del `psql` per canviar la base actual. Aquesta comanda tanca la sessió i n'obre una altra amb el mateix usuari a la base a la què es connecta. Si s'emet sense cap paràmetre serveix perquè el PostgreSQL ens informi de quin usuari som, i quina és la base activa, que és la que diu al prompt.

- `DROP DATABASE club`; esborra la base de dades `club`, i per tant tot el que hi ha dins, encara que de moment no hi ha res. Aquesta comanda provocarà un error la primera vegada que s'executi, perquè la base `club` no existirà. Cap problema. A més, es pot evitar posant `DROP DATABASE IF EXISTS club`;. És a dir, esborra la base de dades si existeix, en tercera persona del singular del present simple. Però per un sol cop que es produirà l'error, no val la pena afegir codi.

- `CREATE DATABASE club`; crea la base de dades en el clúster. A partir d'aquest moment, quan fem `\l` apareixerà la base `club` com una base més.

- `\c club` torna a establir la base de dades actual en la base `club`. Així les taules que creem en endavant es crearan en aquesta base.

En la Pantalla 6.4 es pot observar la importació de l'escript des de la línia de comandes del `psql`. Això és després d'haver fet la importació per segona vegada. És a dir, no s'ha mostrat l'error que es comenta més amunt.



Pantalla 6.4. *Importació del* `club.sql` *des del* `psql`.

Fem ús de la comanda `\i` per incloure o importar l'escript, que ha d'estar en el mateix directori des del que hem invocat el `psql`.

Cada cop que s'emet una comanda `\c` de connexió a una base, PostgreSQL ens diu qui som i a quina base estem, per això a l'incloure l'escript, que conté dues comandes `\c`, apareix en pantalla dues vegades els missatges corresponents. En mig d'elles, veiem que quan una comanda SQL resulta exitosa, el PostgreSQL ens ho informa repetint el nom de la comanda.

Arribats aquest punt, és el moment de posar-se a crear els objectes de la base de dades `club`.

6.3.3 Creació dels Dominis

Tal com s'ha vist en la Secció 5.2.1, sempre que convingui definir algun domini convé fer-ho abans que res. La definició de dominis té per finalitat deixar la responsabilitat de verificar els valors d'un atribut en mans de l'SGBD. Si un atribut és d'un domini concret, ja ens podem despreocupar, que segur que els valors que conté responen a les restriccions que imposi el domini.

La instrucció `CREATE DOMAIN`, inexistent en MySQL, serveix per crear dominis. Després del nom del domini hi ha el tipus de dades que pretenem restringir. La restricció ve marcada pel mot clau `CHECK` seguit d'una expressió booleana entre

parèntesis amb el mot clau VALUE que fa el paper dels valors del domini a l'hora de definir les restriccions. En el cas de la Caixa 6.2 s'utilitza l'operador lògic IN que es veurà en la Secció 6.4.10.

En l'exemple tenim que l'atribut `departament` de la relació `treballador` pot ser *administració*, *comercial*, o *entrenador*.

```
CREATE DOMAIN domini_departament TEXT
    CHECK (VALUE IN ('administració','comercial','entrenador'));
```

Caixa 6.2. *Domini per l'atribut* `departament` *de la taula* `treballador`.

Si el domini s'utilitza tan sols en una taula, podem crear-lo en el mateix escript on crearem la taula. Si s'utilitzés a més d'una, llavors faríem un escript especial de creació de dominis. Pel cas, tan sols s'utilitza en la taula treballador, així que per aquesta taula en concret, l'escript començarà com s'indica en la Caixa 6.2.

No hi ha massa diferència entre la creació d'un domini definit per enumeració, com el de la Caixa 6.2 i una entitat de suport. O sigui, si a la base de dades per al club li afegíssim una taula anomenada `valors_departament` amb una sola columna, clau primària, que contingués tres registres amb els valors *administració*, *comercial*, i *entrenador*, llavors es podria indicar en la taula treballador que el camp `departament` és clau forana de la taula `valors_departament`. Avantatge. Si algun dia apareixen nous departaments, no cladrà remuntar la base de dades, tan sols fer una nova inserció a la taula `valors_departament`. Inconvenient. Fa que les consultes que involucrin l'atribut `departament` siguin menys eficients.

Per altra banda, un domini no es podrà sustituir per una entitat de suport quan es defineixi amb altres criteris que no siguin per enumeració, com en l'exemple de la Caixa 6.3.

```
CREATE DOMAIN positiu INTEGER
    CHECK (VALUE > 0);
```

Caixa 6.3. *Creació d'un domini pels números positius.*

De la base de dades del club esportiu queda per definir un domini una mica més complicat. És el domini que definim pel camp `mail` de la relació `persona`.

A la Secció 6.2.4 s'ha introduït l'operador LIKE, que utilitza patrons fets amb dos caràcters comodí que són el guió baix i el tant per cent.

- El guió baix significa exactament una lletra, qualsevol.
- El tant per cent en substitueix n, que pot ser cap.

Aquest operador és especialment útil per restringir formats d'atributs textuals. Per exemple, en la Caixa 6.4 es presenta un domini que serviria per verificar les adreces de correu electrònic.

```
CREATE DOMAIN domini_mail TEXT
    CHECK (VALUE LIKE '%_@%_.%__');
```

Caixa 6.4. *Domini per l'atribut* `mail` *de la taula* `persona`.

En la Caixa 6.4 definim el tipus de dades `domini_mail`. Els valors dels atributs d'aquest domini han de ser de tipus text, han de contenir una arroba, i com a mínim amb una lletra abans de l'arroba i una després. Observeu que `%_` vol dir com a mínim una lletra. A més, també ha de contenir un punt i dues lletres o més, al final.

Aprofundir en el tema de la descripció de formats passa per fer un incursió en el món de les expressions regulars. És una norma molt extensa, cosa que impedeix introduir-les aquí, però queda dit que hi ha una nomenclatura estàndar de definició de formats que se'n diu expressions regulars.

6.3.4 Creació de les Taules

Posarem cada taula en un escript amb el seu nom. O sigui, que des de l'escript principal haurem d'incloure tants escripts com taules hi hagi. I tot en el mateix ordre del model relacional.

Actuarem metòdicament. Primer farem l'escript, després provarem a llegir-lo des del `psql`, i quan estiguem segurs que no té errors de sintaxi, llavors crearem una carpeta que pengi de la carpeta principal i que es digui igual que l'escript, o sigui, igual que la taula que implementa. Mourem l'escript a la nova carpeta, i llavors afegirem la línia corresponent a l'escript principal.

Comencem doncs fent un nou arxiu de text anomenat `comarca.sql`, amb el contingut que es mostra en la Caixa 6.5. La comanda d'SQL CREATE TABLE serveix per crear taules.

```
\echo ----- taula comarca

CREATE TABLE comarca (
    comarca TEXT,
    CONSTRAINT comarca_repetida PRIMARY KEY(comarca)
);
```

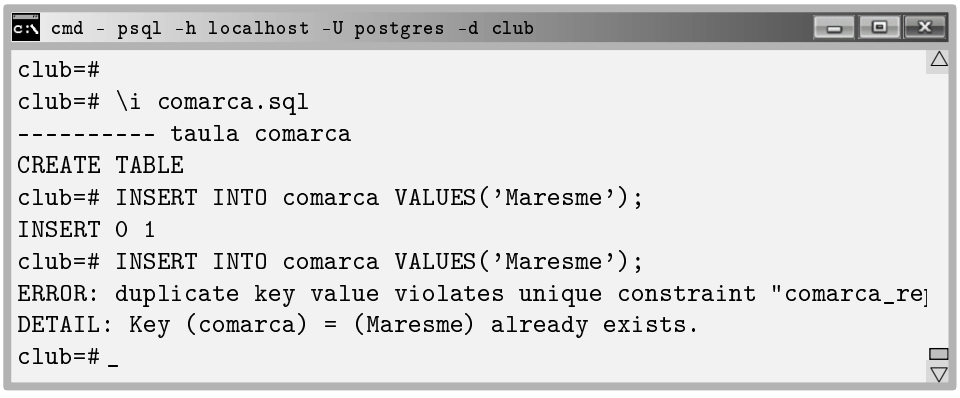
Caixa 6.5. *Arxiu* `comarca.sql`.

Per poder provar que la taula comarca ha estat ben creada cal fer la importació des de la línia de comandes del `psql` amb `\i comarca.sql`.

En la Caixa 6.5, la sintaxis per establir la restricció de clau primària es presenta en la seva forma més completa. Això vol dir que se li ha donat nom a la restricció, `comarca_repetida`. Veiem la seva utilitat amb un experiment de quatre passes, malgrat això signifiqui anticipar la comanda d'inserció, `INSERT INTO`.

1. Importem l'arxiu amb `\i comarca.sql`.
2. Inserim la comarca *Maresme*, amb `INSERT INTO comarca VALUES('Maresme');`.
3. Tornem a inserir la comarca *Maresme* prement fletxa amunt, o tornant-ho a teclejar.
4. Observem el missatge d'error.

Tot plegat es pot veure en la Pantalla 6.5.



Pantalla 6.5. *Prova de clau primària amb nom repetit.*

És a dir, donar nom a una restricció serveix perquè quan es produeixi una violació el nom donat aparegui en el missatge d'error. Això pot servir a voltes si entre el programa client i la base de dades hi ha algún protocol relatiu als codis d'error. Si no es dóna cap nom a la restricció, enlloc de `comarca_repetida` s'anomenaria `comarca_pkey`, de primary key. En endavant acceptarem aquests noms per defecte.

Un cop importat correctament l'escript, creem una carpeta `comarca`, i deixem l'arxiu `comarca.sql` dins. L'estructura de l'espai de treball hauria de ser la que es mostra en la Figura 6.3.

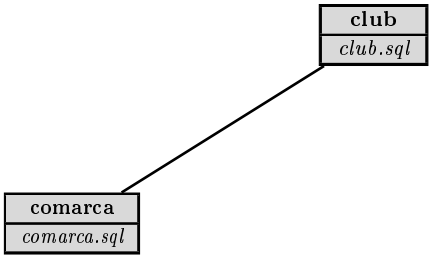


Figura 6.3: *Estructura actual de l'espai de treball.*

La caixa `comarca` representa la carpeta que penja de la carpeta principal. I conté l'arxiu amb el mateix nom que la carpeta i que la taula que implementa. Cada cop que acabem l'escript d'una taula, afegirem una carpeta igual que hem fet ara. I finalment, a l'escript principal afegim la nova línia

```
\i 'comarca\\comarca.sql'.
```

Com que l'escript principal s'executa estant en la carpeta del projecte, cal donar la ruta relativa a aquest directori quan es fa l'import. En `psql` podem usar la doble contrabarra per indicar el separador de directoris.

A continuació, fem un nou escript per la taula `ciutat`.

```
\echo ----- taula ciutat

CREATE TABLE ciutat (
    ciutat TEXT PRIMARY KEY,
    habitants INTEGER,
    comarca TEXT,
    CONSTRAINT comarca_inexistent
         FOREIGN KEY(comarca) REFERENCES comarca(comarca)
);
```

Caixa 6.6. *Primera proposta per l'arxiu* `ciutat.sql`.

A la Caixa 6.6 la declaració de clau primària s'ha fet de la forma més abreviada possible. També s'introdueix una clau forana en la seva forma més completa, donant-li nom. Si no li haguéssim donat nom es diria `ciutat_comarca_fkey`.

Integritat referencial

Introduint una clau forana aprofundim en el conceptes d'integritat referencial. S'estableix que els valors possibles per l'atribut `comarca` de la taula `ciutat` són aquells que hi hagi a la columna `comarca` de la taula `comarca`. Això obre diferents opcions pel que fa a l'actualització. O sigui, si una ciutat apunta a una comarca, cal decidir què fem si la comarca s'esborra o s'actualitza.

- `ON DELETE RESTRICT, o ON UPDATE RESTRICT`

  Si en la Caixa 6.6 haguéssim afegit una última línia com es mostra en la Caixa 6.7, llavors estaríem imposant que no es pugui esborrar una comarca mentre existeixi alguna ciutat que l'apunti. És a dir, que l'SGBD retorni un missatge d'error si s'intenta esborrar la comarca. Si enlloc de `DELETE` digués `UPDATE`, la cosa seria igual quan s'intentés modificar el nom d'una comarca. És a dir, l'SGBD ho prohibiria retornant un error.

```
\echo ----- taula ciutat

CREATE TABLE ciutat (
    ciutat TEXT PRIMARY KEY,
    habitants INTEGER,
    comarca TEXT,
    CONSTRAINT comarca_inexistent
         FOREIGN KEY(comarca) REFERENCES comarca(comarca)
         ON DELETE RESTRICT
);
```

Caixa 6.7. *Restricció d'eliminació per la clau apuntada.*

  Aquesta opció és la que hi ha implementada per defecte, és a dir, la Caixa 6.6 i la 6.7 fan la mateixa funció.

- `ON DELETE CASCADE, o ON UPDATE CASCADE`

  De la mateixa manera, en la Caixa 6.7, una altra opció seria estipular l'eliminació, o actualització, en cascada. Això vol dir, en el cas de l'eliminació, que si s'esborra una comarca s'esborrin també totes les ciutats que l'apuntin. I pel de l'actualització, que si es canvia el nom d'una comarca es canvïi també en totes les ciutats que l'apuntin. S'utilitza el terme `CASCADE` en el sentit que, si per exemple s'esborra una comarca, i

com a conseqüència se suprimeixen totes les ciutats que l'apunten, llavors podria passar que si la clau forana de persona a ciutat també es declarés `ON DELETE CASCADE`, llavors a l'eliminar la comarca, a més d'esborrar totes les ciutats d'aquella comarca, també se suprimirien totes les persones d'aquelles ciutats.

- `ON DELETE SET NULL`, o `ON UPDATE SET NULL`

  Aquesta és l'opció que ens interessaria si no fos imprescindible saber de quina comarca són les ciutats. Significa que en cas que s'elimini una comarca, automàticament es posi un valor nul a les ciutats que s'hi ubiquen.

- `ON DELETE SET DEFAULT`, o `ON UPDATE SET DEFAULT`

  A qualsevol atribut se li pot donar un valor per defecte en el moment de la declaració posant el mot clau `DEFAULT` just després del tipus. Si això es fa amb una clau forana, llavors també es pot establir que en cas que el registre apuntat es modifiqui o s'elimini es posi automàticament el valor per defecte en la clau forana.

En qualsevol cas, observeu que encara que sembli que hi ha una analogia entre esborrar i actualitzar, no és ben bé així. Cal tenir en compte que una eliminació és una operació que afecta un registre sencer, en canvi una actualització pot afectar tan sols un camp. Per tant, les normes de comportament pels casos d'actualització només faran efecte quan s'actualitzi el valor de les claus i prou.

Per altra banda, sempre es pot establir una restricció d'existència per qualsevol atribut posant `NOT NULL` just després del tipus de l'atribut. Per tant, una bona implementació hauria de fer coherents les dues restriccions.

És a dir, eliminacions en cascada haurien de ser associades a atributs no nuls, cosa que en capítols anteriors s'havia anomenat participació total de l'entitat que té la clau forana, en la relació 1:N que implementa aquesta clau forana. O a la inversa, si permetem que una clau forana pugui valdre nul per algun registre, llavors lògicament hauríem de sentenciar `ON DELETE SET NULL` per aquesta clau forana. En aquest segon cas estaríem implementant una participació parcial. Tot plegat se sintetitza en la Taula 6.1.

participació	`ON DELETE`	clau forana
total	`CASCADE`	`NOT NULL`
parcial	`SET NULL`	

Taula 6.1: *Coherència en les participacions.*

És a dir, de la Taula 6.1 cal decidir-se entre un fila, o l'altra. Però no opcions creuades, que significarien incoherència. I com sempre que es parla de participacions, val la pena recordar que la decisió depèn de si interessa o no mantenir els registres del costat n quan s'elimini un registre del costat 1, que

pel cas de l'exemple és si volem mantenir les ciutats encara que se suprimeixi la seva comarca. O dit d'una altra manera, si podem donar ciutats d'alta sense saber la comarca on pertanyen.

L'opció per defecte, RESTRICT, no defineix cap tipus de participació. Precisament és l'opció útil quan qui desenvolupa l'aplicació de la base de dades no té les coses clares. No s'ha d'entendre com un problema, simplement es tracta de posposar una decisió. Qui no té les coses clares és l'usuari final, i per tant ningú s'atreveix a prendre cap decisió. L'opció RESTRICT és útil quan es vol deixar en mans del programa client la forma de procedir. Clar, el programa client pot deixar-ho en mans de l'usuari final en cada cas. De manera que no hi ha cap regla vàlida per actuar quan s'esborri o es modifiqui el resgistre de la taula apuntada.

Lògicament, si podem donar d'alta ciutats ignorant la comarca on pertanyen, les hauríem de mantenir existents si la comarca desapareix. Això és el que diu la Taula 6.1.

Per continuar, decidim que la participació serà parcial. Volem mantenir les ciutats encara que no sabem, o no existeixi, la comarca on es troben, ja que poden ser ciutats d'altres països, i llavors no existirà cap comarca.

Per altra banda, si una comarca canvia de nom, per la raó que sigui, llavors volem que totes les ciutats s'actualitzin amb el nou nom. Això passa quan algú ha comès una errada ortogràfica en la introducció de la dada.

L'escript de la taula `ciutat` queda finalment com es mostra en la Caixa 6.8.

```
\echo ----- taula ciutat

CREATE TABLE ciutat (
    ciutat TEXT PRIMARY KEY,
    habitants INTEGER,
    comarca TEXT REFERENCES comarca
         ON DELETE SET NULL
         ON UPDATE CASCADE
);
```

Caixa 6.8. *Arxiu* `ciutat.sql`.

Observeu que en la versió final de l'arxiu `ciutat.sql` s'ha fet ús de les versions més breus, tant per la clau primària com per la clau forana. En concret, s'ha omès el nom de l'atribut a la taula comarca, ja que si estem dient que l'apuntem, l'SGBD suposa que serà a la seva clau primària. O sigui que alerta. Cal entendre bé sentències com `comarca TEXT REFERENCES comarca`. La primera

aparició de `comarca` és l'atribut que s'està declarant en aquesta taula, i la segona és la taula `comarca` declarada en la Caixa 6.5.

Estem immersos en un procés iteratiu per cada taula del model. La manera de procedir que s'està utilitzant es repeteix tantes vegades com taules hi hagi.

- Codificar la creació de la taula en un escript homònim.
- Importar l'escript principal `club.sql` des de la línia de comandes del `psql`, reconstruint així la base des de zero.
- Importar tot seguit el nou escript. Corregir els errors, fins obtenir la resposta `CREATE TABLE` del PostgreSQL.
- Crear una nova carpeta dins la principal del projecte, amb el mateix nom que la taula i l'escript, i guardar-hi el nou escript.
- Afegir la importació amb la nova ruta de l'escript a l'arxiu `club.sql`.

Així doncs, importem de nou el `club.sql`, i com que ja conté la importació de la taula `comarca`, immediatament després fem la importació de `ciutat.sql` que fins ara es troba a la carpeta principal del projecte. Si el PostgreSQL respon `CREATE TABLE`, llavors es repeteix el procediment d'abans, deixant l'estructura com s'il·lustra en la Figura 6.4.

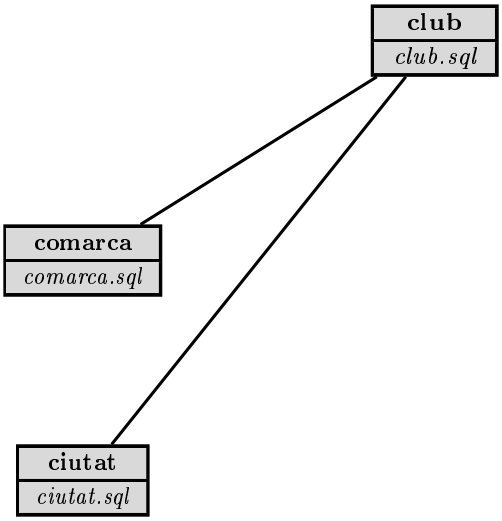


Figura 6.4: *Estructura de l'espai amb la taula* `ciutat`.

Molt bé. Seguim endavant amb la implementació. Comencem la tercera iteració fent un nou escript que ha de ser el que hi hagi en el model relacional del projecte, ja que és en aquest model on s'estableix l'ordre de creació de les taules. Com es pot consultar en el Model 6.2, ara toca la taula `persona`.

En la Caixa 6.9 es crea la taula `persona`.

```
\echo ----- taula persona

CREATE DOMAIN domini_mail TEXT
    CHECK (VALUE LIKE '%_@%_.%__');


CREATE TABLE persona (
    passaport TEXT PRIMARY KEY,
    nom TEXT NOT NULL,
    cognom TEXT NOT NULL,
    mail domini_mail NOT NULL,
    ciutat TEXT REFERENCES ciutat
         ON DELETE SET NULL
         ON UPDATE CASCADE,
    CONSTRAINT mail_ja_assignat UNIQUE(mail)
);
```

Caixa 6.9. *Arxiu* `persona.sql`.

Fent-ho d'aquesta manera estem assumint varies decisions de disseny, d'acord amb la definició de requeriments, que tenen un gran impacte en la base de dades. Aquestes condicions s'enumeren tot seguit.

- Identifiquem cada registre amb el número de passaport. O sigui que prohibim donar d'alta persones si no se sap el seu número de passaport.
- Tampoc acceptem persones si no sabem el nom i el cognom, encara que per aquests atributs es poden repetir els valors.
- L'adreça de correu electrònic de les persones també és necessària. I a més han de ser úniques, no s'admeten repeticions.
- Si a més podem saber la ciutat, millor. Però si la persona no la vol dir, igualment podrà inscriure's en el club. També com abans, si s'elimina una ciutat, mantindrem les persones sense saber d'on són. I si canvia de nom, a les persones de la ciutat en qüestió se'ls hi actualitzarà la ciutat amb el nou nom.

En la Caixa 6.9, la restricció `UNIQUE` serveix per establir unicitat en els valors dels mails. No es permetrà que la mateixa adreça de correu pertanga a més d'una persona. Aquí, la restricció s'implementa en la seva versió més completa, és a dir, donant-li nom. En endavant, però, com amb les claus primàries i foranes, també s'usarà el nom per defecte per les restriccions d'unicitat, que en aquest cas seria `u_mail`.

Claus candidates

La restricció descrita en l'última sentència de la Caixa 6.9 implementa el que normalment és conegut com a clau alternativa. És a dir, un atribut o un conjunt d'atributs formen una clau *alternativa*, o clau *candidata*, en una taula quan es declara una restricció d'unicitat pel conjunt de tots ells, que sovint només és un de sol. Això no impedeix que aquests atributs puguin contenir valors nuls, com ja s'havia dit. La unicitat no implica l'existència perquè no es pot considerar que dos valors nuls siguin equivalents.

El següent pas torna a ser la importació de l'escript principal des de la línia de comandes del `psql`. Seguidament s'importa l'arxiu `persona.sql` que fins aquest moment ha estat en la carpeta principal del projecte. I un cop rectificats els errors sintàctics que pogués haver-hi, cal crear una nova carpeta i moure l'escript dins.

Fet això, actualitzem la línia en l'escript principal. Aquestes alçades, el `club.sql` hauria de tenir l'aspecte que es mostra a la Caixa 6.10.

```
\echo escript principal per la base de dades club
\echo ------------------------------------------
\c postgres
DROP DATABASE club;
CREATE DATABASE club;
\c club
\i 'comarca\\comarca.sql'
\i 'ciutat\\ciutat.sql'
\i 'persona\\persona.sql'
```

Caixa 6.10. *Contingut actual de l'escript principal, arxiu* `club.sql`.

Afegida la línia per la nova taula en l'escript principal ens trobem amb un estat lliure d'errors. Ho podem garantitzar sense ni tan sols provar-ho. Això fa que sigui un bon moment per fer una còpia de seguretat del projecte complet, és a dir de la carpeta principal del projecte amb totes les seves carpetes i arxius.

Tornem a començar. Creada la taula `persona`, podem seguir amb les taules que l'apunten. Com sempre seguint els dictats del model relacional, abordem la taula `telefons`.

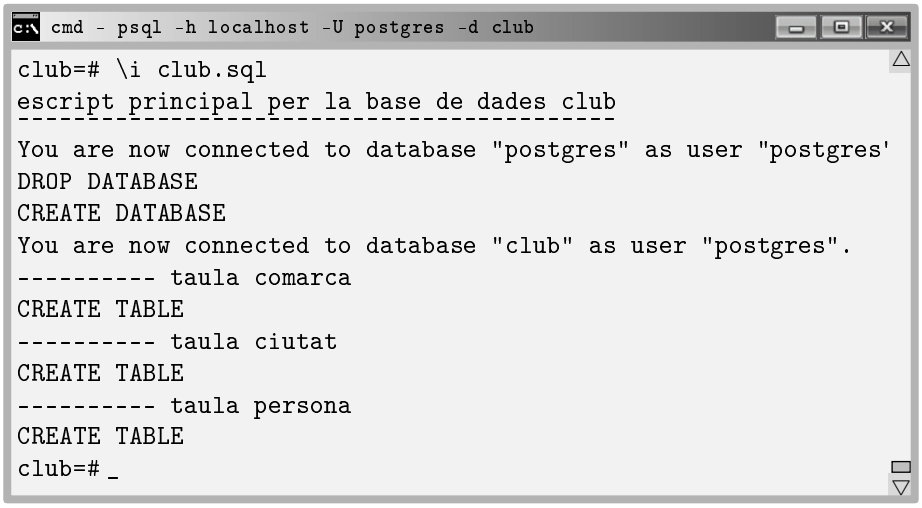
La taula que es mostra en la Caixa 6.11 ve a dir que la persona indicada amb el número de passaport tal, té el telèfon tal. Aquesta informació no sembla que

tingui sentit guardar-la més d'un cop per una persona i un telèfon concret. I per això molts desenvolupadors posarien la parella d'atributs com a clau primària per evitar repeticions. Aquí, s'implementa d'una manera més correcta.

```
\echo ----- taula telefons

CREATE TABLE telefons (
    passaport TEXT NOT NULL REFERENCES persona
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    telefon TEXT NOT NULL,
    UNIQUE(passaport,telefon)
);
```

Caixa 6.11. *Arxiu* `telefons.sql`.

Toca tornar a executar l'escript principal des de la línia de comandes abans d'ensamblar-hi una nou import per aquesta nova taula. En la Pantalla 6.6 es pot observar el diàleg que en aquest moment remunta la base de dades.



Pantalla 6.6. *Importació de l'escript principal a l'SGBD.*

És habitual que les taules que no provenen d'entitats del model ER, i que per tant enlloc de tenir una clau primària tenen una o varies claus foranes, tan sols admetin repetits quan a més de les claus foranes hi hagi algun atribut addicional. És a dir, que més que guardar que dos conceptes estan relacionats, guardem que entre dos conceptes hi ha una col·lecció d'interaccions, cada una de les quals té alguna característica en particular. I si com es deia en el preàmbul d'una

manera filosòfica, el temps estableix diferències entre una cosa i ella mateixa, la característica en particular més comuna que diferencia que dos objectes estiguin relacionats més d'un cop, és el temps. En seguirem parlant.

Tot seguit, cal fer l'import de l'arxiu `telefons.sql` corregint els errors que hi hagi fins que resulti exitós. Es crea la nova carpeta `telefons`, col·locant-hi aquest escript nou, s'afegeix el nou import a l'escript principal, i a per la següent.

En la Caixa 6.12 s'exposa la creació de la taula `coneix`.

```
\echo ----- taula coneix

CREATE TABLE coneix (
    coneix TEXT NOT NULL REFERENCES persona
         ON DELETE CASCADE
         ON UPDATE CASCADE,
    es_coneguda TEXT NOT NULL REFERENCES persona
         ON DELETE CASCADE
         ON UPDATE CASCADE,
    UNIQUE(coneix,es_coneguda)
);
```

Caixa 6.12. *Arxiu* `coneix.sql`.

Les autorelacions M:N tenen doble participació total en l'entitat on són definides. També en aquest cas és clar que no té sentit guardar-se registres amb algun dels dos valors inexistent. Per tant, altre cop es declaren no nuls cada un d'ells. Ni duplicats, i per això la parella com a conjunt, única.

Atenció al temps. Imagineu que enlloc de guardar-nos simplement que dues persones es coneixen volguéssim guardar-nos quan han parlat o s'han vist per última vegada, és a dir, quan han contactat.

Llavors, enlloc de `coneix` l'hauríem d'anomenar `contacta`, significant que una persona contacta l'altra. I caldria també afegir un atribut a l'autorelació que es digués `quan`, i fos de tipus `TIMESTAMP`. Fins aquí, amb la restricció d'unicitat en seguiríem tenint prou per registrar quan va ser l'últim contacte entre la parella de persones.

Però encara hi ha més, si un cop afegit l'atribut `quan` a la relació `contacta` que substituiria `coneix`, treiéssim la restricció d'unicitat, llavors ens quedaria un registre històric de quan s'han vist o han parlat dues persones al llarg dels temps. Al poder-se repetir la parella de persones hi hauria registres de `contacta` que tan sols es diferenciarien en `quan`.

Això és el que es pretenia il·lustrar quan es deia que el temps estableix diferències entre una cosa i ella mateixa. I com a conseqüència, tractats importants de bases de dades distingeixen entre bases de dades temporals, i no temporals, [10], [1]. Les extensions temporals de l'SQL s'aglutinen en un llenguatge anomenat *Temporary Structured Query Language 2*, TSQL2.

Noteu en fi, que havent entès que en un model ER les entitats representen substantius i les relacions verbs, encara s'entén més bé que una relació M:N mantingui un registre de temps.

Abordem tot seguit les entitats de l'especialització del model ER.

Pel tema de les herències en el PostgreSQL hi ha implementada una clàusula `INHERITS`, que podeu consultar en la documentació oficial, [8]. No obstant, presenta problemes a l'hora de distingir un registre heredat, d'un que no ho és. O sigui, utilitzant aquesta eina el PostgreSQL permetria crear una persona amb un número de passaport `12345678A`, i addicionalment un soci diferent de la persona, amb el mateix número `12345678A`. Això fa que si es vol fer servir la clàusula es requereixi de disparadors que controlin aquestes irregularitats. En qualsevol cas, aquí no s'utilitza, i en canvi s'implementa l'herència com sempre s'ha fet en l'SQL, és a dir, amb la clau primària igual a la clau forana.

```
\echo ----- taula soci

CREATE TABLE soci (
    passaport TEXT PRIMARY KEY REFERENCES persona,
    alta TIMESTAMP WITH TIME ZONE NOT NULL
);
```

Caixa 6.13. *Arxiu* `soci.sql`.

En la Caixa 6.13 no hi apareix cap indicació per la clau forana respecte l'actualització o l'eliminació de la clau primària. És a dir, no es diu res de què s'ha de fer amb un soci quan s'elimini la persona corresponent. Això és així perquè, altre cop com a decisió de disseny, no es permetrà que s'esborri una persona que és un soci. Per tant va bé l'opció `ON DELETE RESTRICT ON UPDATE RESTRICT` que és la que hi ha per defecte. El que s'haurà de fer és eliminar el soci. Llavors cal prendre una nova decisió. O acceptem que no hi ha problema en mantenir la informació de la persona corresponent quan s'esborra el soci, o bé caldrà fer un procediment específic per aquesta situació. Una disparador que quan s'elimini un soci esborri també la persona corresponent. Això es veurà a la Secció 7.7.

De moment, aquí es prohibeix l'eliminació d'un soci si s'intenta fer esborrant la persona. I el mateix amb l'actualització. Es podrà modificar un soci, i caldrà

fer disparadors per reflectir les actualizacions a la taula `persona`. I tot plegat, serà igual pel cas dels treballadors.

Observeu també que l'atribut calculat `alta` és del tipus més complet de tots els temporals. Si després interessa tan sols la data ja es podrà fet un truncament sense perdre eficiència. Com que en el mateix instant de la creació d'un soci li donarem valor, es declara com atribut requerit.

A la Caixa 6.14 hi ha l'altra entitat específica de l'herència. En aquest cas, a més, s'usa el domini definit per l'usuari vist en la Caixa 6.2.

```
\echo ----- taula treballador

CREATE DOMAIN domini_departament TEXT
    CHECK (VALUE IN ('administració','comercial','entrenador'));

CREATE TABLE treballador (
    passaport TEXT PRIMARY KEY REFERENCES persona,
    departament domini_departament NOT NULL,
    obeeix TEXT REFERENCES treballador
         ON DELETE SET NULL
         ON UPDATE CASCADE
);
```

Caixa 6.14. *Arxiu* `treballador.sql`.

Com que es considera que tots els treballadors han de pertànyer algun departament, l'atribut corresponent és requerit. Aquesta restricció d'existència s'hagués pogut incloure en la definició del domini, afegint a la condició `AND VALUE IS NOT NULL`, però no és aconsellable perquè acostuma a portar problemes si després volem afegir disparadors en la inserció de la taula.

De la Caixa 6.14 es pot extraure'n també que pot haver-hi treballadors dels quals no se sàpiga qui és el seu cap. Aquesta decisió, un pèl arbitrària, s'ha près per evitar problemes amb els registres dels treballadors el dia que el seu cap plegui. Si s'hagués exigit participació total, llavors a l'hora de donar de baixa un cap, caldria assignar un nou cap a cadascun dels seus treballadors, i això podria representar una feina antipàtica pels usuaris.

Si realment així es desitgés, ho hauríem d'explicar bé a l'usuari. Ara de moment, el que passarà és que no es permetrà esborrar un treballador que sigui cap d'algun altre. Una opció raonable seria que a l'esborrar un cap, els seus treballadors passessin a dependre del que és cap del que s'esborra, que es podria implementar en un disparador.

En la Caixa 6.15 es mostra la taula esport, i la seva comprensió no hauria de suposar cap problema.

```
\echo ----- taula esport

CREATE TABLE esport (
    nom TEXT PRIMARY KEY,
    preu DECIMAL(5,2) DEFAULT 10.0,
    jugadors INTEGER
);
```

Caixa 6.15. *Arxiu* `esport.sql`.

La definició de la taula `fa`, en la Caixa 6.16, es tracta de la implementació clàssica d'una relació M:N. Igual que en el cas de l'autorelació M:N, no sembla tenir sentit guardar-se la informació de que un soci fa un esport més d'un cop. Altre cop convé fer la reflexió d'introduir temps. Afegint dos atributs a la relació podríem mantenir un registre de quan s'ha inscrit un soci en un esport i quan s'ha donat de baixa. Però bé, com ja es diu en el Capítol 2 d'anàlisi del projecte, ha de quedar molt clara en la definició de requeriments la distinció entre informacions que volem tenir en el present, i registres històrics d'una mateixa informació.

```
\echo ----- taula fa

CREATE TABLE fa (
    passaport TEXT REFERENCES soci
         ON DELETE CASCADE
         ON UPDATE CASCADE,
    esport TEXT REFERENCES esport
         ON DELETE CASCADE
         ON UPDATE CASCADE,
    quota DECIMAL(5,2),
    UNIQUE(passaport,esport)
);
```

Caixa 6.16. *Arxiu* `fa.sql`.

Finalment, per acabar amb la construcció de la base de dades `club`, tan sols queda la creació de la taula `nomines`. La definició que es presenta a la Caixa 6.17 té gran interès. L'entitat feble es caracteritza per tenir una clau primària binomial, cosa que imposa declarar la restricció de clau primària en una línia addicional. A més, part d'aquesta clau primària és clau forana de

l'entitat identificadora, `treballador`. Això reflecteix el fet natural d'identificar una nòmina a partir de la identificació del treballador a qui pertany. Pel que fa a l'atribut discriminant, `periode`, guardarem la data de pagament. El sou és requerit. Observeu també que com a retenció s'estableix un valor per defecte. Aquest valor hauria de provenir directament de la definició de requeriments. És a dir, hauria de ser establert per l'usuari final.

```
\echo ----- taula nomines

CREATE TABLE nomines (
    passaport TEXT REFERENCES treballador
         ON DELETE CASCADE
         ON UPDATE CASCADE,
    periode DATE,
    sou_base DECIMAL(6,2) NOT NULL,
    retencio DECIMAL (4,2) DEFAULT 2.00,
    PRIMARY KEY(passaport,periode)
);
```

Caixa 6.17. *Arxiu* `nomines.sql`.

I per fi, ja estem. El model relacional complet és a la Caixa 6.18. Sorprèn l'aparició de la primera línia `\set ON_ERROR_STOP on`.

```
\set ON_ERROR_STOP on
\echo escript principal per la base de dades club
\echo -------------------------------------------
\c postgres
DROP DATABASE club;
CREATE DATABASE club;
\c club
\i 'comarca\\comarca.sql'
\i 'ciutat\\ciutat.sql'
\i 'persona\\persona.sql'
\i 'telefons\\telefons.sql'
\i 'coneix\\coneix.sql'
\i 'soci\\soci.sql'
\i 'treballador\\treballador.sql'
\i 'esport\\esport.sql'
\i 'fa\\fa.sql'
\i 'nomines\\nomines.sql'
```

Caixa 6.18. *Contingut final de l'escript principal, arxiu* `club.sql`.

Per la contrabarra inicial, és una comanda del programa client `psql`. En l'ajuda del `psql`, fent `\?`, s'explica la comanda `\set`, per donar valors a les seves variables d'entorn. La variable del `psql` `ON_ERROR_STOP`, és una variable booleana que com el seu nom indica vol dir que si es produeix algun error a l'executar aquest escript, es pari. Amb aquesta línia al començament no seguirà intentant crear les taules següents després d'un error. Ara, que el nombre d'escripts ja és considerable, si hi hagués algun error desecadenaria un allau de missatges en cascada, ja que qualsevol cosa que impedeixi crear una taula impedeix crear les que depenguin d'ella. I tot plegat resultaria incòmode.

El diàleg amb l'SGBD corresponent a la importació d'aquest escript es mostra en la Pantalla 6.7.

Pantalla 6.7. *Diàleg resultant d'importar l'arxiu* `club.sql` *final.*

L'escript de la Caixa 6.18 s'executa, com tots, en el costat del servidor. Igualment la resposta mostrada en la Pantalla 6.7 també s'emet des del costat del servidor.

I per altra banda, o sigui pel que fa al costat de l'ordinador client, si s'han anat fent les coses com es deia al principi de la construcció de la base de dades hauria d'haver-se creat una estructura de directoris en el disc local. L'estructura completa corresponent al projecte tal com hauria d'estar en aquest moment es mostra en la Figura 6.5. És important que així sigui per poder treballar de manera ordenada en les properes seccions.

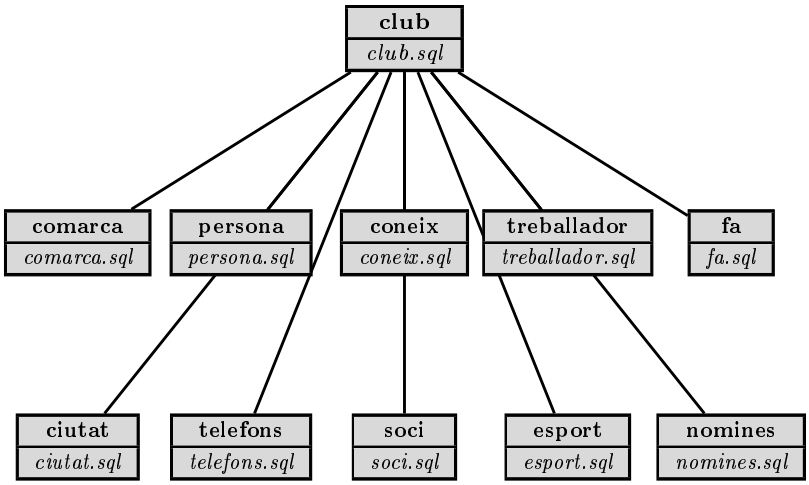


Figura 6.5: *Estructura de directoris del projecte.*

6.4 Llenguatge de Manipulació de Dades

El Llenguatge de Manipulació de Dades, o DML de *Data Manipulation Language*, és la part de l'SQL que s'utilitza per consultar i manipular les dades de la base. Això significa proporcionar les operacions o funcions necessàries per poder fer l'explotació.

Tornem a fer la mateixa transició que en el Capítol 5. Passem d'ocupar-nos d'aspectes relacionats amb l'espai, a accions relacionades amb el temps. El DDL és al concepte de relació en l'àlgebra el que el DML és a les operacions amb relacions. El que farem a partir d'ara ho anomenem consultes. Farem consultes de lectura, que podrem guardar en vistes, i consultes d'actualització.

Escripts d'inserció

Per facilitar les proves dels exemples que es veuran, convé muntar una nova jerarquia d'escripts. Ara però, tots es diran igual, `inserts.sql`. L'estructura quedarà com s'il·lustra a la Figura 6.6.

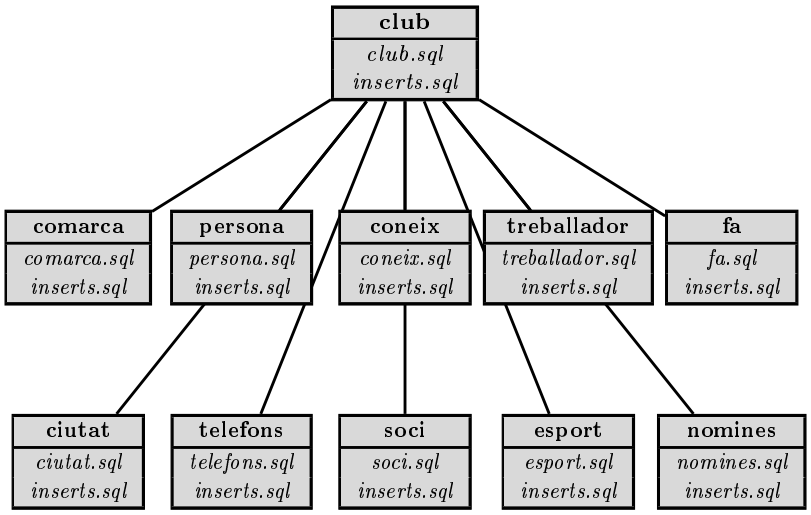


Figura 6.6: *Incorporació d'escripts d'inserció.*

El de la rel, la carpeta del projecte, serà l'escript principal `inserts.sql` i també farà deu imports, de la forma

```
\i 'comarca\\inserts.sql'.
```

Per això, caldrà crear un arxiu de nom `inserts.sql` a cada carpeta del projecte, on hi posarem les dades d'exemple necessàries per poder fer les consultes.

I llavors, afegim una línia al final de l'escript principal per incloure-hi l'escript d'inserts. Si en algun moment interessés tornar a tenir la base buida, no cal res més que treure aquesta última línia de l'escript principal. Les dades d'exemple per totes les taules es mostren a l'Apèndix C, i es poden descarregar a l'adreça que figura al final del preàmbul.

6.4.1 Consultes de Lectura

Anomenem consultes de lectura les que no modifiquen el contingut de la base de dades. Es caracteritzen a més perquè el tipus de valors que retornen són relacions. Hi ha una sola comanda per fer consultes de lectura, el `SELECT`.

En el capítol anterior ens limitàvem a expressar com haurien de formular-se expressions de l'àlgebra relacional que donessin resposta a certes qüestions. Eren expressions teòriques. En endavant no. Ara emetem ordres a l'SGBD, que és qui realitza l'acció. El que abans eren expressions, ara són instruccions que anomenem consultes de lectura. El que abans fèiem nosaltres pensant, ara ho farà l'SGBD computant.

Seguidament es presenta la comanda emblemàtica del llenguatge estructurat de consultes SQL. És una sentència que com a molt pot tenir sis clàusules, encara que en la seva forma més famosa tan sols n'utilitza tres.

Estructura fonamental

L'estructura fonamental d'una consulta de lectura té la forma de la Caixa 6.19,

```
SELECT A_1, A_2, ..., A_k
FROM r
WHERE p;
```

Caixa 6.19. *Estructura fonamental d'una consulta de lectura en SQL.*

sent $r = r(A_1, A_2, \ldots, A_n)$ una relació amb $k < n$, i p un predicat. Aquesta comanda projecta els atributs $A_1, A_2, \ldots, A_k$ dels registres d'r que satisfacin el predicat p.

Així que alerta, la clàusula `SELECT` es correspon amb l'operació de projecció, i no de selecció com podia semblar. És la clàusula `WHERE` la que permet introduir el predicat de selecció de l'àlgebra relacional. I aquestes dues operacions, selecció

i projecció, es realitzen en la relació formada pel producte cartesià de les taules que hi hagi en la clàusula FROM, altrament dit, la relació de la consulta.

En notació coneguda doncs, l'expressió relacional de la Caixa 6.19 és

$$\Pi_{A_1,A_2,\ldots,A_k}(\sigma_p(r)).$$

Això no obstant, l'expressió amb dues taules $r = r(A_1, A_2, \ldots, A_n)$ i $s = s(B_1, B_2, \ldots, B_m)$, sent $k \leq n$ i $\ell \leq m$, resulta més il·lustrativa.

```
SELECT A_1, A_2, ..., A_k, B_1, B_2, ..., B_ℓ
FROM r, s
WHERE p;
```

Caixa 6.20. *Estructura habitual d'una consulta de lectura en SQL.*

La consulta de la Caixa 6.20 és

$$\Pi_{A_1,A_2,\ldots,A_k,B_1,B_2,\ldots,B_\ell}(\sigma_p(r \times s)),$$

expressió que hauria de recordar la tira de la Figura 5.13.

El predicat p, com en l'àlgebra relacional, pot implicar valors constants i atributs de qualsevol de les dues taules. Si hi ha columnes homònimes en r i s presents en el predicat es pot posar el nom de la relació separat per un punt com a prefix del nom d'aquestes columnes, o utilitzar renomenaments.

Dades

Per aconseguir que el PostgreSQL doni algun resultat, primer cal entrar algunes dades. A la Taula 6.2 es pot observar un possible contingut de la relació `comarca`, que és una part de les dades de la taula comarca de l'Apèndix C.

nom
Barcelonès
Alt Empordà
Baix Llobregat
Val d'Aran

Taula 6.2: *Instància de la relació* `comarca`.

L'escript que insereix aquests valors a la base és a la Caixa 6.21. Utilitza un cop més la comanda INSERT INTO, que ja s'ha anticipat a la Secció 6.3.4, per construir relacions enumerant registres de valors constants.

```
\echo ----- inserts taula comarca

INSERT INTO comarca VALUES
('Barcelonès'),
('Alt Empordà'),
('Baix Llobregat'),
('Val d''Aran');
```

Caixa 6.21. *Arxiu* `club/comarca/inserts.sql`.

Observeu que les cadenes de caràcters constants s'escriuen entre cometes simples, o apòstrofs. I també que per escriure el caràcter apòstrof cal posar-lo per duplicat. És el cas de *Val d'Aran*.

A la Taula 6.3 hi ha el que podria valer la relació `ciutat`.

nom	habitants	comarca
Cadaqués	2938	Alt Empordà
Badalona	219708	Barcelonès
San Francisco	805235	
Berlín	3499879	
Rio de Janeiro	6320446	
Castelldefels	63077	Baix Llobregat

Taula 6.3: *Instància de la relació* `ciutat`.

L'escript per establir aquests valors és a la Caixa 6.22.

```
\echo ----- inserts taula ciutat

INSERT INTO ciutat VALUES
('Cadaqués',2938,'Alt Empordà'),
('Badalona',219708,'Barcelonès'),
('San Francisco',805235,null),
('Berlín',3499879,null),
('Rio de Janeiro',6320446,null),
('Castelldefels',63077,'Baix Llobregat');
```

Caixa 6.22. *Arxiu* `club/ciutat/inserts.sql`.

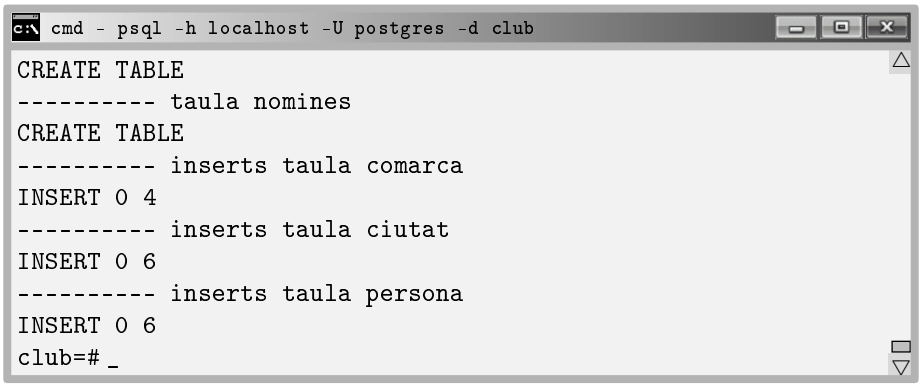
Noteu que les quantitats numèriques, van sense cometes. Igual que un valor nul. Pels booleans caldria usar les constants `true` i `false`, també sense cometes.

I una instància de la taula `persona` per aquests valors podria ser la de la Taula 6.4.

passaport	nom	cognom	mail	ciutat
27673812M	Carme	Peralta	carmep@ionos.cat	Cadaqués
X3478937A	Carles	Sanàbria	carsan1994@gmail.com	Badalona
47548338K	Anna	Sanàbria	annasanabria@gmail.com	Badalona
45493393Z	Jesús	Hortesa	rexstat143@rediris.es	Castelldefels
C00001549	Michael	Bros	mbros1989@aol.com	San Francisco
294394950	Klauss	Stallman	kstallman@dvw.tum.de	Berlín

Taula 6.4: *Contingut de la taula* `persona`.

En la Pantalla 6.8 hi ha la part final del diàleg amb el PostgreSQL quan es reconstrueix la base de dades amb aquestes insercions, és a dir, quan fem un import del `club.sql` actual, que ja inclou l'escript d'insercions.



Pantalla 6.8. *Final del diàleg resultant d'importar l'arxiu* `club.sql` *amb les primeres insercions.*

S'observa en la Pantalla 6.8 que la resposta exitosa de la comanda `INSERT` retorna dues xifres. La primera indica el nombre d'objectes de les metadades afectats, és a dir objectes amb OID. Que aquest primer nombre sigui diferent de zero queda fora de l'abast d'aquest llibre. Aquí, sempre serà zero. La segona és el nombre de files afectades, com es pot endevinar.

Clàusula `SELECT`

La funció de la clàusula `SELECT` és establir els atributs que formen l'esquema del resultat de la relació que s'està requerint. Coincideix amb la projecció de l'àlgebra relacional.

És l'única que forçosament ha d'aparèixer a qualsevol consulta de lectura. Pot anar sola, és a dir sense altres clàusules, com ja s'ha vist en la Secció 6.1.1.

Si en el SELECT li donem un asterisc enlloc de la llista d'atributs, llavors obtindrem tots els atributs de la taula de la clàusula FROM si només n'hi ha una.

Per veure el contingut d'una taula completa utilitzem la consulta més habitual de totes les consultes. En la Caixa 6.23 es presenta la comanda per volcar el contingut de la taula `persona`.

```
SELECT * FROM persona;
```

Caixa 6.23. *Consulta de tot el contingut de la taula* `persona`.

La resposta de PostgreSQL a aquesta consulta es pot veure a la Pantalla 6.9.

```
cmd - psql -h localhost -U postgres -d club
club=#
club=# SELECT * FROM persona;
passaport |  nom   |  cognom   | ciutat
----------+--------+-----------+--------------
27673812M | Carme  | Peralta   | Cadaqués
X3478937A | Carles | Sanàbria  | Badalona
47548338K | Anna   | Sanàbria  | Badalona
45493393Z | Jesús  | Hortesa   | Castelldefels
C00001549 | Mick   | Brown     | San Francisco
294394950 | Klauss | Stallman  | Berlín
(6 rows)

club=# _
```

Pantalla 6.9. *Contingut de la taula* `persona`.

Més genèricament, l'asterisc serveix per demanar tots els atributs que resultin de la concatenació del producte cartesià de les relacions que apareguin a la clàusula FROM. Fixeu-vos-hi bé. No es tracta de la unió d'atributs, sinó de la concatenació. Aquest extrem s'aclarirà en la propera secció.

SELECT amb multiconjunts

Com s'ha insistit anteriorment, la diferència més important entre l'àlgebra relacional i l'SQL és que la primera treballa amb conjunts, i la segona amb multiconjunts. En altres paraules, les relacions en SQL poden tenir elements repetits.

Això és fruit que en l'àlgebra les tuples de les relacions són diferents per definició, en canvi en l'SQL cal establir claus si volem garantitzar aquesta condició.

En l'exemple 6.4 es pretén obtenir el nom de totes les ciutats de les persones de la base. La primera aproximació es limita a obtenir el nom de totes les ciutats de la base de dades.

exemple 6.4. *Llistar les ciutats de les persones de la base de dades.*

solució
```
SELECT nom FROM ciutat;
```

I clarament, no és una bona solució, perquè ens mostra ciutats de les que no és ningú. És el cas de *Rio de Janeiro*. En una segona aproximació podríem pensar en

solució
```
SELECT ciutat FROM persona;
```

que efectivament ens mostra les ciutats de les persones de la base de dades. Això no obstant, cada ciutat apareix tants cops com persones hi ha, com es veu pel cas de *Badalona*.

Quan es fa un `SELECT` d'un camp que no és clau primària, i en particular quan és clau forana resulta molt habitual, poden aparèixer elements repetits en la relació resultant. Així es comporta l'SGBD per defecte. Quan es vulgui evitar aquest efecte, s'utilitza `SELECT DISTINCT`. La solució 6.5 és correcta.

exemple 6.5. *Llistar les ciutats de les persones de la base de dades.*

solució
```
SELECT DISTINCT ciutat FROM persona;
```

Per defecte, un `SELECT` és amb l'opció `ALL`, que vol dir que apareguin els valors repetits. Això és interessant si per exemple es volgués comptar les persones de cada ciutat. Es resoldria comptant quants cops apareix el nom de cada ciutat en una consulta com la de la segona solució de l'exemple 6.4. Per això l'SQL manté la possibilitat de treballar amb relacions que siguin multiconjunts.

Clàusula `FROM`

La clàusula `FROM` és la que indica l'univers a partir del qual fem les dues coses, establim criteris per segons quins atributs, i en sol·licitem d'altres. Per això, s'anomena *relació de la consulta* a la relació resultant del producte cartesià de les taules presents en la clàusula `FROM`.

El nombre de relacions que apareixen en una clàusula `FROM` és l'*ordre* de la consulta, i serveix per mesurar la seva complexitat.

Al computar-se, qualsevol consulta comença amb la relació de la consulta. I això té una incidència més que considerable. Si en una consulta hi apareix una

taula innecessària, el temps de resposta es multiplica per el número d'elements que tingui la taula innecessària. Això és un error molt greu. Cal mantenir al cap que quan afegim una taula a la clàusula FROM estem afegint un bucle que la recorre, és a dir, que tracta un element en cada iteració. Vist d'una altra manera, si fem una selecció amb una sola taula, ens podem imaginar el codi consistent amb un bucle que recorre la taula. Si posem dues taules en el FROM llavors és un bucle dins d'un bucle. I si n'hi posem tres, doncs un bucle que per cada registre de la primera recorre cada registre de la segona, i per cada parella possible de la primera i la segona es recorre el bucle per cada un dels elements de la tercera.

Com que és important, fem un experiment de vuit passes amb la quantitat de registres resultants d'un producte cartesià.

1. Crear una taula, `t`, amb un sol atribut que es digui `n` i sigui un enter, fent `CREATE TABLE t(n INTEGER);`
2. Inserir un nombre qualsevol a la taula `t`, fent `INSERT INTO t VALUES(314);`
3. Mostrar el nom de totes les comarques afegint innecessàriament la taula `t` en la clàusula FROM, fent `SELECT comarca FROM comarca,t;`
4. Observar que hi ha quatre resultats, el nombre de comarques.
5. Inserir un nou nombre qualsevol a `t`, fent dos cops fletxa amunt, o teclejant altre cop `INSERT INTO t VALUES(314);`
6. Mostrar de nou el nom de totes les comarques afegint innecessàriament la taula `t` en la clàusula FROM.
7. Comprendre el resultat.
8. Eliminar la taula `t` fent `DROP TABLE t;`

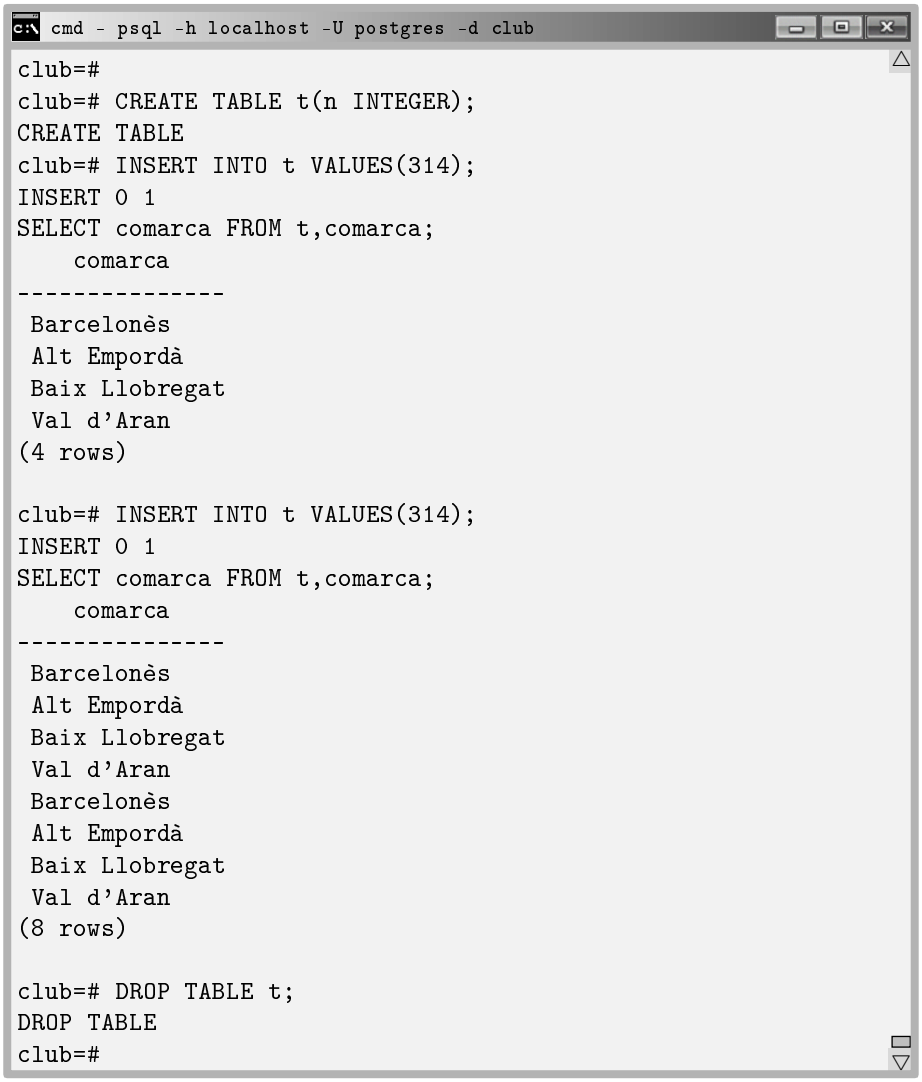
Un incís respecte la novetat d'aquest experiment. La comanda DROP TABLE. Una comanda del DDL. Llevat que pot fer-se en CASCADE, no hi ha res més a dir. Fer un `DROP TABLE t CASCADE;` serviria per eliminar la taula `t` i totes les que puguéssin fer-ne referència amb una clau forana, ja que si es fa un DROP TABLE d'una taula que és apuntada des d'alguna altra, es produeix un error. Ara però, com que no hi ha cap taula que referncïi `t`, afegir l'opció CASCADE no tindria cap efecte.

El que s'hauria de comprendre d'aquest experiment és que per cada nou element que afegim a la taula `t`, el resultat de la selecció posterior treuria un cop més la llista de totes les comarques, o sigui, que la relació resultant s'incrementaria en quatre registres més.

El fet de només projectar un atribut de la taula `comarca` fa que l'experiment resulti més sorprenent. Això és, si enlloc d'inserir dos cops el mateix número a

la taula `t` haguéssim posat dos valors diferents, i a més, enlloc d'un sol atribut els haguéssim projectat tots, amb un asterisc en el SELECT, llavors el resultat s'hauria comprès d'una manera molt més clara, ja que de totes les parelles possibles entre les quatre comarques i els dos elements de `t`, les quatre primeres haurien aparegut associades al primer número, i les quatre segones, al segon.

Tal com s'ha fet, l'experiment exposa més clarament el perill d'afegir taules innecessàries en la clàusula FROM. A la Pantalla 6.10 es pot observar el diàleg d'aquest experiment.



Pantalla 6.10. *Resultat de l'experiment per la cardinalitat del producte cartesià.*

Tot això pel que fa al cardinal del producte cartesià entre les taules que apareixen a la clàusula FROM. Aprofundim seguidament en el tema dels atributs, l'esquema de la relació de la consulta. Les columnes del producte cartesià. El PostgreSQL no té cap problema en repetir noms de columnes en les relacions resultants.

Observem-ho amb una prova senzilla, amb un resultat que repeteixi el nom de la columna `comarca`.

exemple 6.6. *Obtenir totes les combinacions possibles de ciutats i comarques*

solució `SELECT * FROM ciutat,comarca;`

A la Pantalla 6.11 hi ha el resultat de la solució a l'exemple 6.6.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT * FROM ciutat, comarca;
    ciutat      | habitants |    comarca     |    comarca
----------------------+-----------+----------------+-------------
 Cadaqués        |      2938 | Alt Empordà    | Barcelonès
 Badalona        |    219708 | Barcelonès     | Barcelonès
 San Francisco   |    805235 |                | Barcelonès
 Berlín          |   3499879 |                | Barcelonès
 Rio de Janeiro  |   6320446 |                | Barcelonès
 Castelldefels   |     63077 | Baix Llobregat | Barcelonès
 Cadaqués        |      2938 | Alt Empordà    | Alt Empordà
 Badalona        |    219708 | Barcelonès     | Alt Empordà
 San Francisco   |    805235 |                | Alt Empordà
 Berlín          |   3499879 |                | Alt Empordà
 Rio de Janeiro  |   6320446 |                | Alt Empordà
 Castelldefels   |     63077 | Baix Llobregat | Alt Empordà
 Cadaqués        |      2938 | Alt Empordà    | Baix Llobreg
 Badalona        |    219708 | Barcelonès     | Baix Llobreg
 San Francisco   |    805235 |                | Baix Llobreg
 Berlín          |   3499879 |                | Baix Llobreg
 Rio de Janeiro  |   6320446 |                | Baix Llobreg
 Castelldefels   |     63077 | Baix Llobregat | Baix Llobreg
 Cadaqués        |      2938 | Alt Empordà    | Val d'Aran
 Badalona        |    219708 | Barcelonès     | Val d'Aran
 San Francisco   |    805235 |                | Val d'Aran
 Berlín          |   3499879 |                | Val d'Aran
 Rio de Janeiro  |   6320446 |                | Val d'Aran
 Castelldefels   |     63077 | Baix Llobregat | Val d'Aran
(24 rows)

club=#
```

Pantalla 6.11. *Resultat de l'experiment per les columnes del producte cartesià.*

Renomenament

En la Pantalla 6.11, atenent els títols de les columnes de la resposta de l'SGBD es pot comprendre fins a quina frontera pot arribar l'ambigüitat.

Si és pretengués anar més enllà, demanant per la columna `comarca`, enlloc d'asterisc, PostgreSQL ens retornaria un error, explicant que no pot saber a quina columna ens estem referint, si la de la taula `ciutat`, o `comarca`. La consulta de la Pantalla 6.12 no té massa lògica des d'un punt de vista semàntic. L'exemple és estrictament tècnic. En el fons s'està demanant per les comarques que apareguin a la taula `ciutat` tants cops com comarques hi hagi. No té sentit, però il·lustra el tema de l'ambigüitat. Obtindríem la resposta que es mostra.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT comarca FROM ciutat, comarca;
ERROR: column reference comarca is ambiguous
LINE 1:  SELECT comarca FROM ciutat,comarca;
                ^

club=# _
```

Pantalla 6.12. *Error provocat per ambigüitat de la columna* `comarca`.

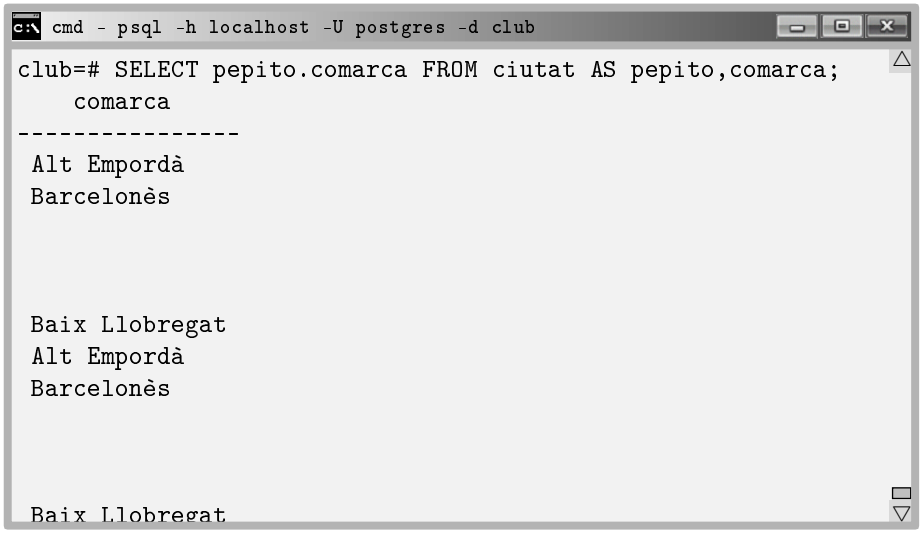
A més, a la Pantalla 6.12 s'hi mostra la forma d'imprimir els errors que té `psql`. Noteu l'accent circumflex que apareix just després de la línia i just en la primera lletra de la paraula que ha provocat l'error. Això és força interessant, i convé prestar-li atenció. Quan pot, PostgreSQL ens indica amb una precisió a la lletra, quin terme és el que ha provocat un error.

Bé, aquest error es podria resoldre, com s'ha dit en la Secció 6.4.1, posant el nom de la taula com a prefix de l'atribut, en el predicat del `SELECT`. És a dir,

`SELECT ciutat.comarca FROM ciutat,comarca;`.

De tota manera, quan els atributs homònims apareixen en els predicats del `WHERE` que tot seguit es veuran, i afecten més d'un atribut comú a les taules del `FROM`, o quan es fa referència dues vegades a la mateixa taula en una consulta, llavors convé usar l'operador `AS`.

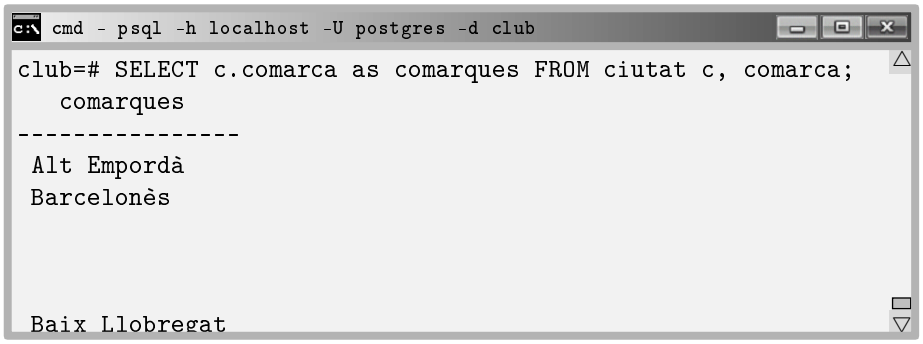
És interessant dominar l'operador de canvi de nom de cara les consultes aniuades que es veuran a la Secció 6.4.9. De moment tan sols comprendre que la consulta de la Pantalla 6.12 s'hagués pogut desambigüitzar de la manera que es mostra en la Pantalla 6.13, encara que només s'hagi capturat una part inicial de la resposta.



Pantalla 6.13. *Fragment inicial de la solució correcta, amb canvi de nom de la taula* `ciutat`*, a* `pepito`.

Fem servir la paraula *pepito* per deixar clar que pot ser qualsevol paraula.

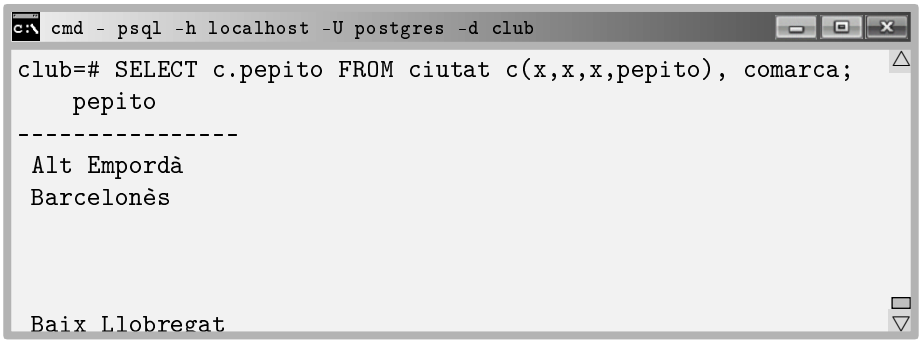
Tant freqüent resulta el renomenament que si no es posa l'operador AS en el FROM, de manera que quedi un espai entre dues paraules, s'entén que la primera és una taula i la segona el nom que rep en aquesta consulta. En la Pantalla 6.14 fem un renomenament a una paraula d'una lletra. És la manera habitual com s'utilitzarà. Davant la possibilitat que s'afegeixin noves columnes a les taules en el futur, convé fer-ho per seguretat.



Pantalla 6.14. *Solució equivalent a la de la Pantalla 6.13 amb canvi de nom de la taula* `ciutat`*, a* `c`.

En la Pantalla 6.14, a més, s'ha renomenat l'atribut, i per això el títol `comarques` de la columna. El mateix efecte s'aconsegueix renomenant l'atribut en el mateix lloc que es dóna el nom `c`, o sigui en el FROM. Això significa renomenar

la taula i cadascun dels seus atributs. La Pantalla 6.15 mostra una solució equivalent a la de la Pantalla 6.14.



Pantalla 6.15. *Canvi de nom dels atributs d'una relació.*

És notable el fet que per renomenar un atribut haguem de donar un nom per cada un dels altres. En la Pantalla 6.15 s'ha anomenat `x,x,x` als altres atributs de la taula `ciutat`. Això va bé per confirmar que sabem de què estem parlant, quants atributs té la taula i quin interessa.

Clàusula WHERE

La clàusula WHERE porta un predicat. Que quedi clar, després de la clàusula hi ha d'anar alguna expressió que pugui ser avaluada com a certa o falsa per cada una de les tuples de la relació de la consulta, que com s'ha dit en la secció anterior, és la relació formada pel producte cartesià de les relacions de la clàusula FROM.

Recordeu del Capítol 1 que un predicat està format de proposicions unides per mitjà d'operacions conjuntives o disjuntives, o sigui de les operacions lògiques AND i OR.

L'exemple més simple de la clàusula WHERE es fa en una consulta d'una sola taula.

exemple 6.7. *Noms de les persones de* Cadaqués.

solució `SELECT nom FROM persona WHERE ciutat = 'Cadaqués';`

I un altre exemple, aquest cop implementant un join. Diem *join* aquelles proposicions que formen part dels predicats, en les quals es demana que valors de columnes vinculades coincideixin. Vinculades no vol dir necessàriament homònimes. Quan s'indroduïa la reunió interna, a la Secció 5.5.2, s'ha precisat aquest extrem.

exemple 6.8. *Noms de les persones de la comarca del* Barcelonès.

solució
```
SELECT p.nom
FROM persona p,ciutat c
WHERE  p.ciutat = c.ciutat
AND c.comarca = 'Barcelonès';
```

La finalitat d'haver declarar els mateixos noms per les claus primàries i les foranes que les apuntin, és a dir, les columnes amb les quals es faran els joins és precisament no haver d'expressar les proposicions corresponents en els predicats. L'exemple 6.8 es podrà simplificar a partir de la Secció 6.4.7.

Finalment, un últim exemple per mostrar la potència de l'estructura fonamental d'una consulta. Per això, caldrà omplir algunes dades de la taula `coneix`. Posem pel cas que entre les persones de la relació donada en la Taula 6.4 de la pàgina 190 hi ha les coneixences que es poden deduir de l'escript que es mostra en la Caixa 6.24.

```
\echo ----- inserts taula coneix

INSERT INTO coneix VALUES
('27673812M','X3478937A'),
('X3478937A','27673812M'),
('27673812M','47548338K'),
('27673812M','294394950'),
('294394950','27673812M'),
('294394950','C00001549');
```

Caixa 6.24. *Arxiu* `club/coneix/inserts.sql`.

Per introduir aquestes dades, encara que no sigui imprescindible, convé remuntar la base de dades completa important l'escript `club.sql`. En particular, en la Caixa 6.24 hi diu que la Carme de Cadaqués (`27673812M`) coneix en Carles i l'Anna Sanàbria de Badalona, i al Klauss de Berlín.

coneix	**es_coneguda**
27673812M Carme	X3478937A Carles
X3478937A Carles	27673812M Carme
27673812M Carme	47548338K Anna
27673812M Carme	294394950 Klauss
294394950 Klauss	27673812M Carme
294394950 Klauss	C00001549 Mick

Taula 6.5: *Relacions de coneixença, no necessàriament simètriques, entre les persones.*

En la Taula 6.5 es mostra el mateix contingut de la Caixa 6.24 però amb els noms de les persones al costat del número de passaport. Un cop les dades introduïdes, hi ha la capacitat de respondre a qüestions tan complicades com la de l'exemple 6.9.

exemple 6.9. *A qui coneix la* Carme*?*

solució

```
SELECT q.nom
FROM persona p, coneix, persona q
WHERE p.nom = 'Carme'
AND p.passaport = coneix
AND q.passaport = es_coneguda;
```

Analitzem la solució de l'exemple 6.9.

Per pensar aquest tipus de consulta és imprescindible adonar-se'n que el que es demana fa dues referències a la taula `persona`. Un cop comprès això, comencem posant un prefix, en aquest cas `q`, a l'atribut del `SELECT`. Llavors, en el `WHERE` cal actuar en coherència segons el prefix que s'ha donat en el `SELECT`, ja que indica la tupla o el registre del qual en última instància estem demanant el resultat. Descrivim els lligams que cal respectar, tenint en compte quin és el nom que li hem donat a la relació en el `SELECT`, o sigui la `q`, i quin nom té la taula que serveix per establir els criteris que ens han donat, o sigui `p`. És senzill, veure que de les dues referències a la taula `persona` una és per establir criteris i l'altra per demanar resultats.

La resposta del PostgreSQL a l'exemple 6.9 es pot observar a la Pantalla 6.16.

```
cmd - psql -h localhost -U postgres -d club
---------- inserts taula persona
INSERT 0 6
---------- inserts taula coneix
INSERT 0 6
club=# SELECT q.nom FROM persona p, coneix, persona q
club-# WHERE p.nom = 'Carme'
club-# AND p.passaport = coneix
club-# AND q.passaport = es_coneguda;
  nom
--------
 Carles
 Anna
 Klauss
(3 rows)

club=# _
```

Pantalla 6.16. *Persones a qui la* Carme *coneix.*

6.4.2 Funcions d'Agregació

Les funcions d'agregació en l'SQL són un reflex fidel de les definicions de l'àlgebra relacional, de la Secció 5.6.3. En essència, són funcions que retornen un sol registre, que normalment és d'un sol atribut, calculat a partir d'una col·lecció de registres. Com a conseqüència, si en una columna passem de moltes files a una de sola, caldrà agrupar valors iguals quan es demani per més d'una atribut, i aquesta agrupació regirà les agregacions calculades. Cal concentrar-se per entendre-ho. L'agrupació regirà les agregacions calculades. Això es veurà en la propera secció.

Pels títols de les columnes resultants, PostgreSQL utilitza el nom de la mateixa funció d'agregació. Sempre es pot renomenar amb la clàusula `AS`, o inclús, com en el cas del `FROM`, sense cap paraula. Si en la clàusula `SELECT` apareix una expressió i una paraula separades només per un espai en blanc, l'SQL entén que la segona és un nou nom per la primera, com s'ha vist en l'exemple 6.14.

En endavant faran falta més quantitat de dades que les que s'ha fet servir fins ara. Per això, qui vulgui fer el seguiment executant cada exemple pot descarregar de l'adreça que es diu al final del preàmbul totes les necessàries. Els resultats que es mostren en les properes seccions han estat calculats amb la totalitat d'aquestes dades. O també hi ha la possibilitat de fer-ho amb paper i llapis, tenint en compte que les dades inserides a la base són les que es mostren a l'Apèndix C.

Les funcions d'agregació estàndar en SQL es llisten tot seguit.

De tots els tipus d'atributs

Les funcions d'agregació per atributs alfanumèrics que podem garantitzar que ens retornaran una sola tupla, és a dir, un sol registre, són el recompte, el mínim, i el màxim.

Funció `COUNT()`

Funció de recompte del nombre de registres que hi hauria a la relació resultant de la consulta. És clar que per comptar el nombre de registres podem comptar el nombre de valors de clau primària diferents que hi ha a la taula.

exemple 6.10. *Quantes persones hi ha a la base de dades?*

solució
```
SELECT COUNT(passaport) FROM persona;
```

Una manera més genèrica de fer-ho és posant-hi un asterisc.

solució `SELECT COUNT(*) FROM persona;`

Alerta. Es compten tots els registres. És a dir, pel cas de seleccionar un sol atribut que es repeteix en el resultat, es compta tants cops com aparegui. Si volem comptar només el nombre de valors diferents, llavors podem utilitzar el `DISTINT` vist a la Secció 6.4.1. L'exemple 6.11 calcula la llista de les comarques que tenen alguna ciutat que hi visqui alguna persona. I després compta els valors diferents d'aquesta llista.

exemple 6.11. *De quantes comarques hi ha persones a la base de dades?*

solució
```
SELECT COUNT(DISTINCT c.comarca)
FROM persona p, ciutat c
WHERE  p.ciutat = c.ciutat;
```

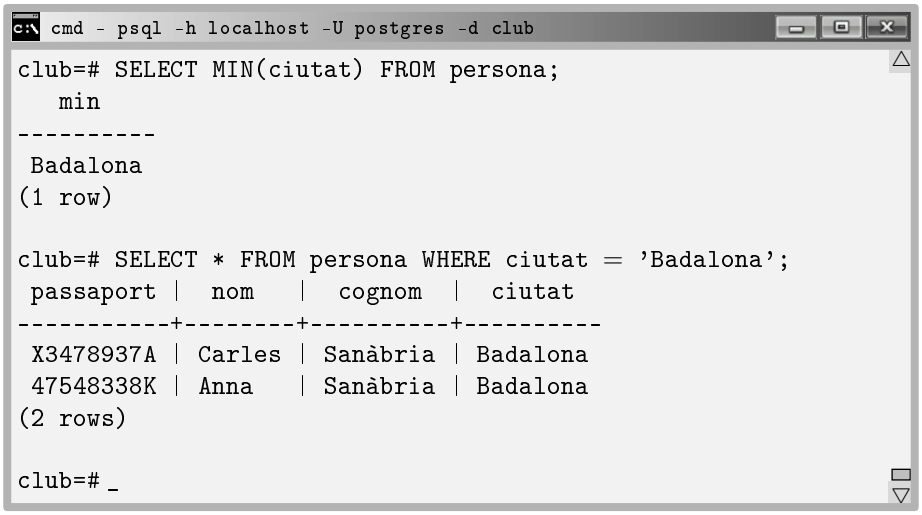
Funcions `MIN()` i `MAX()`

Funcions que retornen el valor mínim, o el màxim, de l'atribut demanat.

exemple 6.12. *Quina és la primera ciutat de les que hi viu alguna persona de la base, per ordre alfabètic?*

solució `SELECT MIN(ciutat) FROM persona;`

La captura corresponent a l'exemple 6.12 es mostra en la Pantalla 6.17. Observeu que si el mínim valor es produeix en varis registres, igualment retorna una sola tupla. Per tant, no té sentit usar la clàusula `DISTINCT`.



Pantalla 6.17. *El valor mínim és únic, encara que es produeixi en més d'un registre.*

D'atributs numèrics

Les funcions d'agregació per atributs numèrics retornen sempre un sol número. De fet, els atributs que no són numèrics en les consultes que utilitzen funcions d'agregació fan el paper de criteri d'agregació. Pels tipus numèrics es disposa addicionalment de les funcions descrites a continuació. Només se'n presenten dues. Això no obstant, les funcions d'agregació disponibles depenen de l'SGBD. Tots saben calcular desviacions tipus, i moltíssimes altres funcions, però no coincideixen amb els noms que utilitzen. Per això, cal consultar l'ajuda.

Funció `SUM()`

Retorna la suma dels valors donats. Si hi ha nuls els ignora, és a dir, els tracta com si fossin zeros.

exemple 6.13. *Quants habitants hi ha entre totes les ciutats de la base de dades?*

solució
```
SELECT SUM(habitants) FROM ciutat;
```

En la Pantalla 6.19 s'imprimeix l'execució de l'exemple 6.13.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT SUM(habitants) FROM ciutat;
    sum
----------
 37469270
(1 row)

club=# _
```

Pantalla 6.18. *Consultes d'agregació numèriques.*

Funció `AVG()`

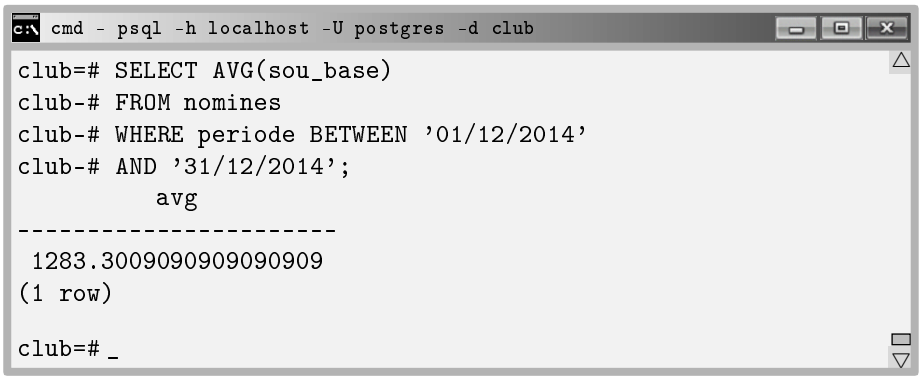
Retorna la mitja dels valors donats. Si hi ha nuls els ignora, és a dir, no comptabilitzen a l'hora de fer la divisió per calcular la mitja.

exemple 6.14. *Calcular el sou mig dels treballadors del club esportiu el mes de desembre del 2014.*

solució
```
SELECT AVG(sou_base)
FROM nomines
WHERE periode BETWEEN '01/12/2014' AND '31/12/2014';
```

En l'exemple 6.14 hem usat un operador booleà que es diu BETWEEN AND i és equivalent a posar dues proposicions una de menor o igual i l'altra de més gran o igual. Són coses de l'SQL. Ja s'ha dit que quan va néixer pretenia impressionar entre altres coses per el seu grau d'humanitat en el llenguatge.

En la Pantalla 6.19 es pot veure el resultat de l'exemple 6.14.



Pantalla 6.19. *Consultes d'agregació numèriques.*

Clàusula `GROUP BY`

En la Caixa 6.25 es mostra el format per aquest tipus de consultes, sent $r = r(A_1, A_2, \ldots, A_n)$ amb $n \geq k + \ell$.

```
SELECT A1, A2, ..., Ak, f1(Ak+1), f2(Ak+2), ..., fℓ(Ak+ℓ)
FROM r
GROUP BY A1, A2, ..., Ak;
```

Caixa 6.25. *Format d'una consulta amb criteris d'agregació.*

Els atributs $A_1, A_2, \ldots, A_k$ formen el criteri d'agregació. El fet que s'hagin de repetir forçosament a la clàusula GROUP BY significa que si volem saber alguna informació addicional, a part del resultat de la funció d'agregació, la informació que volguem saber s'utilitzarà per segmentar el resultat global.

Per tant, la clàusula GROUP BY ha d'aparèixer obligatòriament al final de les consultes on a part de la funció d'agregació apareguin més atributs en el SELECT. És a dir que si no, es produeix un error sintàctic.

Vist com en la Caixa 6.25 aquest tipus de consultes semblen més complicades del que són. Normalment, $k = 1$, i $\ell = 2$. O sigui, que demanem per la suma o la mitja d'un atribut segmentat segons un altre.

Com es pot suposar, l'impacte del criteri d'agregació en la consulta és total. Segons els atributs que es projectin acompanyant un funció d'agregació varia radicalment el nombre de registres que s'obtenen. Veiem-ho pel cas de la suma per exemple, en dues consultes.

exemple 6.15. *Quin guany mensual es té en el club esportiu?*

solució
```
SELECT SUM(quota)
FROM fa;
```

L'exemple 6.15 calcula una suma agregada total. Al no posar cap més atribut que aquell sobre el qual es calcula la funció d'agregació, la consulta es converteix en una agregació d'aquell atribut per tota la relació de la consulta, `fa`. És el que s'ha vist en la secció anterior.

Ara bé, si volem saber qualsevol altra informació relativa a la suma de l'exemple 6.15, aquesta mateixa informació segmentarà la suma total.

exemple 6.16. *Quin guany mensual es té en el club esportiu per cada esport?*

solució
```
SELECT esport, SUM(quota)
FROM fa
GROUP BY esport;
```

Això es així de clar. Manen els atributs alfabètics del criteri d'agregació.

Per entendre'ns suposem que el criteri d'agregació és un sol atribut, com en el cas de l'exemple 6.16. Llavors se suposa que els valors d'aquest atribut apareixen repetidament en la relació de la consulta. També se suposa que la relació de la consulta té algun atribut numèric. Bé, doncs la funció d'agregació es calcula a partir dels valors de la columna numèrica per totes les files que coincideixin en el valor del criteri d'agregació. Si els valors de l'atribut que forma el criteri d'agregació no es repeteixen, llavors l'agregació no tindrà massa sentit. Tot plegat també val entenent que el criteri d'agregació for estar format de varis atributs, en el qual cas entenem que diferents valors pot voler dir diferents combinacions de valors.

Ergo, si el resultat d'una consulta amb funcions d'agregació té més d'una fila, la consulta ha de tenir una clàusula `GROUP BY`.

Sovint els usuaris finals es confonen a l'hora de demanar atributs no agregats juntament amb agregats.

La llei bàsica és que qualsevol relació resultant ha de tenir el mateix número de files per totes les columnes. Per tant, si només agreguem algunes columnes, les altres han d'agrupar-se tan com poden, és a dir, per cada valor diferent fan una línia. I com a conseqüència, l'agregació se segmentarà per aquests valors.

A la Figura 6.7 es pot observar la captura de la pantalla pels exemples 6.15 i 6.16. Sobre aquesta captura s'ha dibuixat el que representa la implicació sintàctica, i de fet semàntica, de l'atribut `esport` fent de criteri d'agregació.

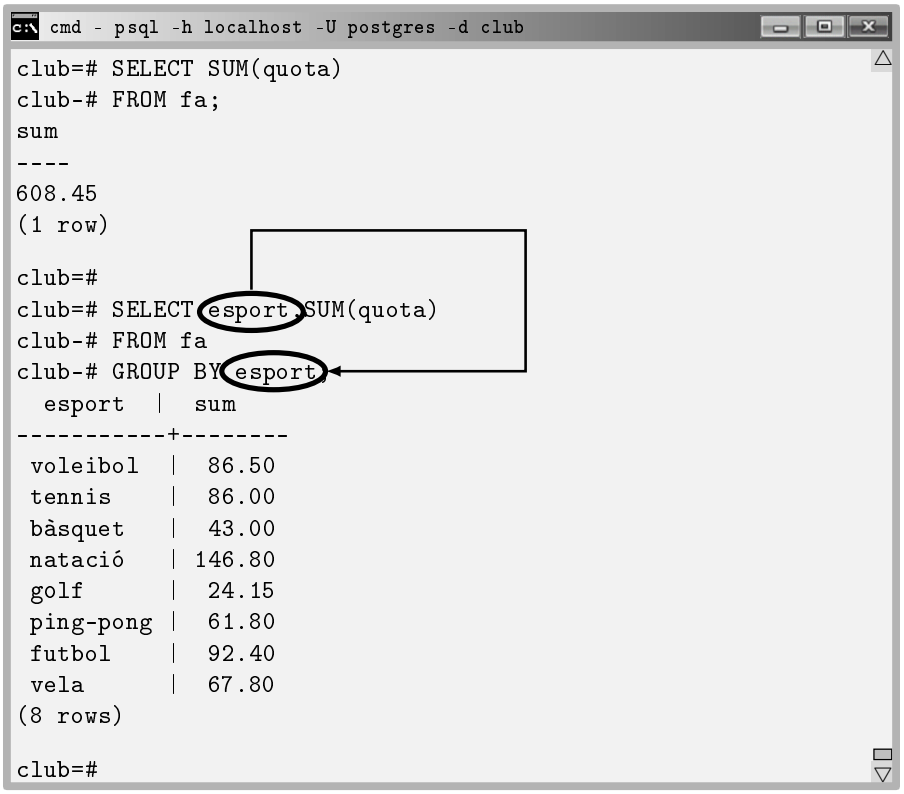


Figura 6.7: *Implicació sintàctica de la clàusula* `GROUP BY`.

Clàusula `HAVING`

La clàusula `HAVING` serveix per establir predicats sobre el resultat del valor de la funció agregada. O sigui, que incorpora un expressió booleana que involucra el resultat d'una funció d'agregació. Per exemple, una mitja més alta que un valor.

És doncs, semblant al `WHERE`, però enlloc d'operar a partir de valors d'atributs ho fa a partir del resultat de les funcions.

Raonem. Si es tracta de filtrar el resultat d'una consulta agregada, és a dir, donar com a resultat només les tuples que satisfacin el predicat de la clàusula HAVING, llavors és que estem parlant d'una selecció que en principi obtindrà varies tuples. I si el resultat de la consulta d'agregació abans de seleccionar les que cumpleixin el predicat té varies columnes, llavors és que la consulta agregada té un GROUP BY, ja que l'única manera possible que una consulta amb una funció d'agregació tingui més d'un registre resultant és que hi hagi un GROUP BY. En conseqüència, en la major part dels casos que s'utilitza HAVING, en la mateixa consulta s'ha utilitzat el GROUP BY.

exemple 6.17. *Quines comarques tenen més de cent mil habitants?*

solució
```
SELECT comarca, SUM(habitants)
FROM ciutat
GROUP BY comarca
HAVING SUM (habitants) > 100000;
```

Aquesta conseqüència es pot portar encara més lluny. Es pot assegurar que si una consulta conté la clàusula HAVING i no la GROUP BY, llavors és una consulta d'existència. És a dir, que el resultat només serveix per saber si el valor de la funció d'agregació per tota la relació de la consulta satisfà el predicat. És el cas de l'exemple 6.18, que pot tornar un registre, o cap.

exemple 6.18. *Entre totes les ciutats de la base, hi ha més d'un milió d'habitants?*

solució
```
SELECT SUM(habitants)
FROM ciutat
HAVING SUM (habitants) > 1000000;
```

6.4.3 Clàusula d'Ordenació

L'última de les clàusules que pot portar una consulta de lectura és la que permet ordenar, ascendenment o descendent, la relació resultant segons un atribut, o més d'un si es vol establir ordres secundaris pels desempats en l'ordre principal.

Clàusula ORDER BY

Quan es vol demanar la relació ordenada per algun dels atributs de la relació de la consulta es pot afegir la clàusula ORDER BY seguit del nom de l'atribut en qüestió. Per defecte, l'opció per ordenar és ASC que vol dir ascendent. Podem invertir-lo posant DESC després de l'atribut.

Tot seguit es detalla què vol dir ascendent i descendent segons el tipus dels atributs.

- Pels atributs numèrics els sentits d'ordres ascendent i descendent són ben clars. Ascendent és creixent.
- Pels alfanumèrics, l'ordre alfabètic és l'ordre ascendent. Bé, més que l'alfabètic, l'ordre entre tots els caràcters de la taula ASCII que es mostra a l'Apèndix B. O sigui, que no és estrictament alfabètic, perquè la a minúscula va darrera de la zeta majúscula.
- Pels temporals, el passat és menor que el futur.
- I pels booleans, fals és menor que cert.

Quan no es demana cap ordre en una consulta que només consta d'una taula en el `FROM`, PostgreSQL la dóna ordenada segons l'ordre cronològic de les insercions. És a dir, en la relació resultant apareixen primer els registres que fa més temps que van ser inserits. Això és fàcilment comprovable consultant una taula, esborrant el primer element, i tornant-lo a inserir.

El criteri d'ordenació ha de ser un atribut present en la relació de la consulta, i no necessàriament de la clàusula `SELECT`. Per exemple, en la Pantalla 6.20 es llista el nom de les ciutats ordenades pel nombre d'habitants, que no apareix.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT ciutat FROM ciutat ORDER BY habitants;
-----ciutat-------
 Tredòs
 Cadaqués
 Tona
 Tiana
 Montgat
 Roses
 Amposta
 Sitges
```

Pantalla 6.20. *Ciutats ordenades per nombre d'habitants.*

Com a criteri d'ordenació es pot donar més d'un atribut pel cas que en el primer hi hagi valors repetits en la relació resultant. Per exemple si es volqués les persones de la base de dades ordenades alfabèticament, i per aquelles que es diguin igual, ordenades segons l'ordre alfabètic de la seva ciutat, la consulta seria

```
SELECT nom,ciutat FROM persona ORDER BY nom,ciutat;
```

Addicionalment, amb la clàusula `ORDER BY` sovint es dóna la clàusula `LIMIT` per limitar el nombre de registres de la relació resultant. Això és especialment útil per aplicacions client que treballen amb pàgines de mida fixa a l'hora de fer les consultes.

Consulta de lectura amb totes les clàusules

Tanquem el tema de les consultes de lectura amb un exemple que utilitza les sis clàusules. Per pensar-la, cal tenir en compte que volem un filtre abans i un després d'una funció d'agregació. En l'exemple 6.19 s'exposa l'estructura completa que pot tenir una consulta de lectura.

exemple 6.19. *Donar el nom dels esports i la quantitat de socis que els practiquen, pels esports que practiquin més de tres socis i que es juguin en solitari, ordenats per nombre de socis.*

solució

```
SELECT f.esport, COUNT(f.passaport) as socis
FROM fa f, esport e
WHERE f.esport = e.esport
AND e.jugadors = 1
GROUP BY f.esport
HAVING COUNT(f.passaport) > 3
ORDER BY socis;
```

I en la Pantalla 6.21 es pot veure la relació resultant.

```
cmd - psql -h localhost -U postgres -d club
club=#
club=# SELECT f.esport, COUNT(f.passaport) as socis
club-# FROM fa f, esport e
club-# WHERE f.esport = e.esport
club-# AND e.jugadors = 1
club-# GROUP BY f.esport
club-# HAVING COUNT(f.passaport) > 3
club-# ORDER BY socis;
  esport   | socis
-----------+-------
 tennis    |     4
 ping-pong |     6
 natació   |     8
(3 rows)

club=#
```

Pantalla 6.21. *Consulta de lectura amb totes les clàusules.*

En SQL es pot utilitzar el nom d'una columna renomenada en el `SELECT` en la clàusula `ORDER BY`, però no en el `HAVING`. O sigui, si en la consulta de la Pantalla 6.21 haguéssim posat `ORDER BY COUNT(f.passaport)` cap problema, en canvi si haguéssim posat `HAVING socis > 3`, llavors s'hauria produït un error.

6.4.4 Vistes

Una vista és el suport físic per una consulta. És clar que com que les dades de la base van variant amb el temps pot interessar la mateixa consulta en diferents moments. I per tant, convé guardar-les d'alguna manera per no haver-les de tornar a teclejar.

Els objectes de tipus vista, com les taules, es guarden a les metadades de la base. Però a diferència d'una taula que guarda físicament les dades, de les vistes tan sols es guarda el codi SQL de la consulta que implementen. Això vol dir que si una vista mostra una columna d'una taula i s'insereix, es modifica o s'esborra algun registre de la taula, la relació resultant de la vista es veurà afectada per aquests canvis.

El concepte de vista és dels primers SQLs que van existir, a principis dels setanta. Després, l'SQL del 92 va introduir les funcions que poden fer tot el que fa una vista, i més coses. De tota manera, les vistes són un concepte més estàndar, ja que les funcions utilitzen llenguatges de programació que depenen de l'SGBD. I en canvi, les vistes tan sols utilitzen SQL pur.

La sintaxi de creació de vistes es presenta a la Caixa 6.26.

```
CREATE VIEW nom_vista AS <consulta de lectura>
```

Caixa 6.26. *Sentència de creació d'una vista.*

Cal entendre que una vista és com una taula pel que fa a les operacions. Per exemple, per consultar la relació resultant d'una vista es fa amb un `SELECT` exactament igual que si fos una taula. Les vistes també es poden posar en el `FROM` de qualsevol consulta, i combinar-les amb altres taules o inclús amb altres vistes provocant una dependència en cascada.

Per esborrar una vista, hi ha la comanda `DROP VIEW nom_vista`. I també igual que en el cas de les taules, si afegim l'opció `CASCADE` després del nom, abans del punt i coma, llavors s'esborraran totes les vistes que facin servir aquesta en la seva definició.

De la mateixa manera, si fem un `DROP TABLE nom_taula CASCADE`; d'una taula utilitzada en la definició d'alguna vista també s'eliminarà la vista en qüestió.

En la Caixa 6.27 s'implementa una vista amb la relació del que paga cada soci cada mes.

```
\echo ----- vista pagaments_soci

CREATE VIEW pagaments_soci AS
SELECT p.passaport,p.nom,p.cognom, SUM(f.quota) AS pagament
FROM persona p, fa f
WHERE p.passaport = f.passaport
GROUP BY p.passaport,p.nom,p.cognom
ORDER BY p.cognom;
```

Caixa 6.27. *Arxiu* `club/fa/pagament_soci.sql`.

Exactament igual que en l'etapa de construcció, creem l'arxiu `pagaments_soci.sql` en la carpeta rel del projecte. Llavors des de la línia de comandes del `psql` l'importem fent `\i pagaments_soci.sql`. I verifiquem el seu funcionament, tal com es mostra en la Pantalla 6.22.

```
cmd - psql -h localhost -U postgres -d club
club=#
club=# \i pagaments_soci.sql
----- vista pagaments_soci
CREATE VIEW
club=# SELECT * FROM pagaments_soci;
 passaport |   nom   |  cognom   | pagaments
-----------+---------+-----------+-----------
 39238229E | Sònia   | Aragall   |     33.75
 C00001549 | Michael | Bros      |     31.20
 59119283Z | Pere    | Camprubí  |     32.85
 187448338 | Rita    | Derbeken  |     24.00
 27961020N | Pere    | Garcia    |     32.85
 45493393Z | Jesús   | Hortesa   |     44.10
 38433548L | Anna    | Margalef  |     28.65
 19891898A | Maria   | Martín    |     31.80
 42065765F | Camila  | Noriega   |     43.00
 27673812M | Carme   | Peralta   |     44.05
 46372382N | Roser   | Puente    |     33.75
 C01X01TN  | Roberto | Rietto    |     28.65
 X3478937A | Carles  | Sanàbria  |     44.05
```

Pantalla 6.22. *Verificació de l'escript de la Caixa 6.27.*

Dins l'espai del projecte guardarem els escripts que implementin les vistes en la carpeta corresponent a l'última taula que utilitzi en la seva definició, segons l'ordre del model relacional. Aquest nou escript, `pagaments_soci.sql`,

el guardarem a la carpeta `club/fa` del projecte. I la importació anirà a l'escript principal, just després de la importació de l'escript de creació de la taula `fa`.

6.4.5 Consultes d'Actualització

Anomenem consultes d'actualització les que actualitzen l'estat de la base de dades. És extrany dir consulta a una inserció o eliminació, però així és. Sí que s'entén en canvi que una inserció en una taula és una actualització de la base de dades. Alerta doncs, que això pot portar confusió. Hi ha tres comandes per fer les consultes d'actualització, `INSERT INTO`, `UPDATE SET`, i `DELETE FROM`. Això contempla altes, modificacions i baixes. Les consultes d'actualització es caracteritzen pel tipus de les dades que retornen. Un número enter. El resultat d'una consulta d'actualització és la quantitat de registres que s'hagin vist afectats en la base de dades.

Clàusula `INSERT INTO`

En la Secció 6.3.4 ja s'ha anticipat l'ús de la comanda per inserir registres de valors constants. En la seva versió més senzilla es mostra en la Caixa 6.28.

```
INSERT INTO r VALUES (a_1, a_2, ..., a_n);
```

Caixa 6.28. *Forma bàsica de la comanda d'inserció.*

sent $r = r(A_1, A_2, \ldots, A_n)$, amb $A_i \subseteq D_i$ i $a_i \in D_i$, per $i = 1, \ldots, n$.

En tots els escripts d'inserció que hi ha en la carpeta del projecte s'utilitza profusament la forma de la Caixa 6.28, on a més, ja s'ha vist que en una sola inserció es poden introduir molts registres constants, separats per comes.

Si D_i és un domini de tipus alfanumèric o temporal, llavors el valor a_i del registre inserit ha d'anar entre cometes simples, o apòstrofs, de manera que per introduir un apòstrof cal posar-ne dos de seguits. Si D_i és de tipus numèrics es posen sense cometes. Pels booleans es dóna el seu valor amb els mots clau *true* i *false*. I per un valor nul, s'escriu la paraula *null*.

Observeu doncs que en la seva forma més simple, la comanda `INSERT INTO` té molt present l'esquema de la relació r, ja que els valors donats han de ser compatibles amb aquest esquema.

Ara bé, si es desconneix l'ordre dels atributs en l'esquema d'r llavors també es pot fer una inserció. Si no se sap o no es pot utilitzar l'ordre dels atributs en l'esquema, cal forçosament saber els seus noms, així com els dominis per poder donar els valors. A més, també si són requerits o no, ja que si són requerits caldrà donar-los en qualsevol cas.

Amb la forma que es mostra en la Caixa 6.29 la comanda fa explícit l'esquema d'r, simesnó la part que afecta a la inserció,

```
INSERT INTO r(A_1, A_2, ..., A_k) VALUES (a_1, a_2, ..., a_k);
```

Caixa 6.29. *Forma alternativa de la comanda d'inserció.*

sent $k \leq n$.

D'aquesta manera, l'associació entre valors i atributs és posicional. O sigui, el primer valor donat es correspon amb el primer atribut de l'esquema donat, i el segon amb el segon. Se suposa que els atributs d'r no presents en la comanda, $A_{k+1}, \ldots, A_n$, no són requerits, i per tant queden nuls després de la inserció, a no ser que en la taula on s'està inserint hi hagi valors definits per defecte amb l'opció `DEFAULT`. Si algun dels atributs no donats en la comanda fos requerit i no tingués declarat un valor per defecte, es produiria un error.

Així doncs, aquesta forma alternativa d'inserir valors constants en una taula consisteix en donar el nom dels atributs que s'estan sumministrant del nou registre. A més, amb la forma 6.29 tenim la possibilitat de donar els atributs en qualsevol ordre. O sigui, que aquesta forma per la instrucció d'inserció té una doble vessant. Encara que els donem tots, o sigui $k = n$, també té sentit utilitzar-la si és perquè interessa introduir els valors en algun ordre específic.

exemple 6.20. *Inserir un nou esport,* atletisme, *amb un jugador per equip.*

solució
```
INSERT INTO esport VALUES('atletisme',null,1);
```

o alternativament

solució
```
INSERT INTO esport(jugadors,nom) VALUES(1,'atletisme');
```

En les dues soluccions de l'exemple 6.20 fem ús del valor per defecte de l'atribut `preu`, que és `10.0`.

La forma més complexa de la clàusula és donant una expressió relacional, és a dir, sense el mot clau `VALUES`.

```
INSERT INTO r (E);
```

Caixa 6.30. *Forma més complexa de la comanda d'inserció.*

En la Caixa 6.30 es presenta la forma d'inserir el resultat d'una consulta a una taula. Noteu que cal tenir molt clar l'esquema resultant de l'expressió E a fi que sigui compatible amb el de la relació r.

exemple 6.21. *Regalar una inscripció per fer* atletisme *als deu socis que més paguen.*

solució
```
INSERT INTO fa (
    SELECT passaport,'atletisme',0.0
    FROM pagaments_soci
    ORDER BY pagaments DESC
    LIMIT 10
    );
```

En la Pantalla 6.23 es captura l'execució d'aquestes últimes comandes.

Pantalla 6.23. *Inserció complexa en base a una consulta.*

Dos comentaris respecte el contingut de la Pantalla 6.23. Per un costat mencionar l'ús de la projecció d'un valor constant, *atletisme*, o el 0.0 de la quota en la clàusula SELECT de la inserció. I també notar l'ús de l'ordre descendent, DESC i de la clàusula LIMIT. S'havia parlat d'aquesta estructura en la Secció 6.4.3.

Errors habituals provocats per aquesta instrucció són el d'existència prèvia de clau primària, o el de necessitat de valors requerits que no es donen. Un altre error molt comú és, quan s'insereix en una taula que té una clau forana,

que no existeixi cap registre en la taula apuntada amb el valor de clau primària que s'està donant en la clau forana en qüestió.

Clàusula `UPDATE SET`

Per la mateixa naturalesa que al fer una inserció cal fer explícits tots els valors requerits dels registres que s'insereixen, per fer una modificació d'un conjunt de registres existents, tan sols cal identificar-los i conèixer el nom i els tipus dels atributs que es pretenen modificar, a part dels nous valors, clar.

En la Caixa 6.31 es presenta la clàusula que serveix per fer modificacions en el contingut de la base de dades. L'ordre en que es donin els atributs $A_i, \ldots, A_{i+k-1}$ és irrellevant. Si no es posa el `WHERE` la modificació es fa en tots els registres de la taula. Els $a_1, a_2, \ldots, a_k$ són els nous valors, o expressions que donen un resultat, del domini de l'atribut al que s'assignen. L'expressió de la Caixa 6.31 modifica tots els registres d'r pels quals el predicat doni cert. Això no obstant, en la seva versió més simple només es modifica el valor d'un atribut. I a més, el predicat p consta d'una sola proposició que involucra la clau primària d'r.

`UPDATE` r `SET` $A_i = a_1, A_{i+1} = a_2, \ldots, A_{i+k-1} = a_k$ `WHERE` p;

Caixa 6.31. *Sintaxi per la comanda d'actualització.*

sent $r = r(A_1, \ldots, A_i, \ldots, A_{i+k-1}, \ldots, A_n)$ i p un predicat que involucra atributs d'r i constants.

En l'exemple 6.22 es fa una modificació de tants registres com tenistes hi hagi a la base de dades. L'exemple 6.23 és més complexe.

exemple 6.22. *Augmentar un 10% la quota del* tennis *als socis.*

solució

```
UPDATE fa
SET quota = quota * 1.10
WHERE esport = 'tennis';
```

exemple 6.23. *Nomenar nova cap d'entrenadors l'*Helena Pérez.

solució

```
UPDATE treballador
SET obeeix = (SELECT passaport
              FROM persona
              WHERE nom = 'Helena'
              AND cognom = 'Pérez')
WHERE departament = 'entrenador';
```

La consulta de l'exemple 6.23 suposa que tan sols hi ha una persona que es digui *Helena Pérez*. I com es pot veure en la Pantalla 6.24 tres registres de la taula treballador queden afectats per aquesta modificació, és a dir, hi ha tres entrenadors en la base de dades. En la mateixa pantalla s'il·lustren els exemples 6.22 i 6.23.

```
cmd - psql -h localhost -U postgres -d club
club=#
club=# UPDATE fa
club-# SET quota = quota * 1.10
club-# WHERE esport = 'tennis';
UPDATE 4
club=# UPDATE treballador
club-# SET obeeix = (SELECT passaport
club(# FROM persona
club(# WHERE nom = 'Helena'
club(# AND cognom = 'Pérez')
club-# WHERE departament = 'entrenador';
UPDATE 3
club=#
```

Pantalla 6.24. *Actualitzacions amb la clàusula* `UPDATE SET`.

Errors freqüents d'aquesta instrucció passen per la definició de les restriccions en el moment de creació de la base de dades. Si no s'ha declarat la manera d'actualitzar claus foranes quan es modifica el valor de la clau apuntada, per defecte és `ON UPDATE RESTRICT`, que vol dir que es prohibeixi l'actualització i s'emeti un missatge error en cas d'intent de modificació del valor d'una clau principal.

Clàusula `DELETE FROM`

És natural que per esborrar registres de la base de dades tan sols calgui identificar-los. Això es fa exactament igual que amb un `SELECT` d'una sola taula. La manera d'eliminar registres de la base de dades és la que es mostra en la Caixa 6.32,

```
DELETE FROM r WHERE p;
```

Caixa 6.32. *Sintaxi per la comanda d'eliminació.*

sent r una relació i p un predicat definit sobre els seus atributs.

Tots els registres d'r que satisfacin el predicat p se suprimiran. Efectivament, igual costa trobar un registre per consultar-lo, o sigui amb un SELECT, que per esborrar-lo amb un DELETE FROM. De fet, que l'esforç computacional de la consulta i de l'eliminació siguin equivalents succeeix amb totes les estructures de dades. Per això té la seva lògica que la sintaxi per l'eliminació sigui exactament igual que la de la selecció amb asterisc, però sense l'asterisc.

exemple 6.24. *Esborreu les comarques que comencin amb la lletra* O.

solució
```
DELETE FROM comarca WHERE comarca LIKE 'O%';
```

En l'exemple 6.24 s'utilitza l'operador LIKE que s'ha introduït en la Secció 6.2.4, i ha estat utilitzat en la creació del domini pels mails en la Secció 6.3.4. Amb l'execució d'aquesta consulta s'elimina la comarca *Osona*, ja que és l'única amb aquesta inicial. Observem en un segon exemple la directiva ON DELETE SET NULL que s'estableix en la creació de la taula `persona` pel camp `ciutat`.

exemple 6.25. *Donar de baixa la ciutat d'en* Klauss Stallman.

solució
```
DELETE FROM ciutat
WHERE ciutat = (SELECT ciutat
                FROM persona
                WHERE nom = 'Klauss'
                AND cognom = 'Stallman');
```

En la Pantalla 6.25 es descriu l'execució dels dos exemples anteriors. Per el 6.25 es verifica prèviament que la ciutat existeix, i enlloc d'aniuar la consulta es fa en dues passes.

```
cmd - psql -h localhost -U postgres -d club
club=# DELETE FROM comarca WHERE nom LIKE 'O%';
club=# DELETE 1
club=# SELECT * FROM persona WHERE nom = 'Klauss' AND cognom = 'Stallman'
 passaport |  nom   |  cognom  | ciutat
-----------+--------+----------+----------
 294394950 | Klauss | Stallman | Berlín
(1 row)
club=# DELETE FROM ciutat WHERE nom = 'Berlín';
club=# DELETE 1
club=# SELECT * FROM persona WHERE nom = 'Klauss' AND cognom = 'Stallman'
 passaport |  nom   |  cognom  | ciutat
-----------+--------+----------+----------
 294394950 | Klauss | Stallman |
(1 row)
club=# _
```

Pantalla 6.25. *Eliminacions amb la clàusula* DELETE FROM.

6.4.6 Operacions amb Conjunts

Encara que només fos per respecte als seus ancestres, l'àlgebra relacional i encara més enllà la teoria de conjunts, l'SQL implementa de la manera que pot les tres operacions bàsiques de la teoria de conjunts. De la manera que pot vol dir que tenint en compte que les relacions en aquest capítol són multiconjunts, cal formalitzar el comportament de les operacions en aquest nou entorn.

Aquestes tres operacions són la unió, la diferència i la intersecció de conjunts. Les tres operacions són binàries, o sigui que operen a partir de dues relacions d'entrada. I també totes elles requereixen la compatibilitat d'aquestes relacions. La compatibilitat entre relacions s'ha definit en la Secció 5.4.4, a la pàgina 129, en base a la compatibilitat entre conjunts, introduïda ja en la Caixa 1.2. En definitiva, cada un dels atributs demanats ens els dos `SELECT`s de les relacions d'entrada han de pertànyer a dominis compatibles dos a dos. O sigui, que o són iguals, o un és subconjunt de l'altre.

Clàusula `UNION`

L'operació d'unió de relacions implementada en SQL té la sintaxi que es mostra en la Caixa 6.33,

```
SELECT A_1, A_2, ..., A_k
FROM r
WHERE p
UNION
SELECT B_1, B_2, ..., B_k
FROM s
WHERE q;
```

Caixa 6.33. *Sintaxi de l'operació d'unió de relacions.*

sent els esquemes $A_1, A_2, \ldots, A_k$ i $B_1, B_2, \ldots, B_k$ compatibles.

El resultat és una relació formada pels registres resultants de la primera consulta i després els de la segona, a no ser que es canvi l'ordre explícitament en la mateixa consulta.

És clar que moltes relacions que podrien calcular-se amb aquesta operació resolen el problema afegint una disjuntiva en el predicat del `WHERE`. Tot i així, en altres casos no és possible.

L'exemple 6.26 obté un llistat de les persones del club juntament amb el pagament si són socis, o el sou base de la nòmina si són treballadors.

A més del número de passaport, el nom, el cognom i el pagament, que és negatiu quan es tracta de nòmines, també mostra en una columna addicional que anomena `vincle` si el registre correspon a un soci o a un treballador, de manera que redunda amb el signe de la columna `pagament`. Així doncs, els registres corresponents als treballadors coincideixen amb els registres de pagaments negatius.

exemple 6.26. *Quins ingressos o despeses ha suposat pel club cada persona el novembre del 2014?*

solució

```
SELECT *,'soci' AS vincle
FROM pagaments_soci
UNION
SELECT p.passaport, p.nom, p.cognom,
            n.sou_base * (-1), 'treballador'
FROM persona p, nomines n
WHERE n.passaport = p.passaport
AND n.periode BETWEEN '01/11/2014' AND '01/12/2014'
ORDER BY cognom;
```

El fet de projectar atributs constants, com 'soci' o 'treballador', en l'última columna ja s'havia vist en la Secció 6.4.5. Concretament en la comanda d'inserció.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT *,'soci' AS vincle
club-# FROM pagaments_soci
club-# UNION
club-# SELECT p.passaport, p.nom, p.cognom,
club-# n.sou_base * (-1), 'treballador'
club-# FROM persona p, nomines n
club-# WHERE n.passaport = p.passaport
club-# AND n.periode BETWEEN '01/11/2014' AND '01/12/2014'
club-# ORDER BY cognom;
 passaport |    nom    |  cognom   | pagaments |   vincle
-----------+-----------+-----------+-----------+-------------
 39238229E | Sònia     | Aragall   |     33.75 | soci
 C00001549 | Michael   | Bros      |     31.20 | soci
 C00021549 | Mick      | Brown     |   -915.35 | treballador
 59119283Z | Pere      | Camprubí  |     32.85 | soci
 X4534332C | Gabriel   | Cobos     |  -1050.30 | treballador
```

Pantalla 6.26. *Unió de relacions.*

A més, en la Pantalla 6.26 és pot veure la forma com la unió de dues relacions assigna títols de les columnes. Tan sols que una de les dues relacions tingui títol,

el pren. Per això no cal donar un títol a la columna amb el sou base negatiu. En la relació resultant es dirà `pagament`, ja que la primera de les relacions d'entrada sí que té títol per aquesta columna. Exactament el mateix passa amb la columna `vincle`.

Multiconjunts

La unió de relacions agrega per defecte els registres equivalents. Això es pot evitar amb l'opció `ALL`. O sigui, exactament al contrari que el `SELECT`, que per defecte repeteix repeticions, i cal l'opció `DISTINCT` perquè les agregui.

A fi d'interioritzar la diferència entre treballar amb conjunts i treballar amb multiconjunts, analitzem meticulosament les dues opcions amb un experiment de cinc passes. Cal entendre que es tracta d'un experiment tècnic amb aquest objectiu.

És clar que els resultats de les consultes que tot seguit es detallen es podrien obtenir més senzillament tan sols amb l'estructura fonamental de les consultes de lectura.

1. Seleccionem nom i cognom de les persones que viuen a *Amposta*.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT nom,cognom FROM persona WHERE ciutat = 'Amposta';
  nom   | cognom
--------+--------
 Carme  | Ferrer
(1 row)

club=# _
```

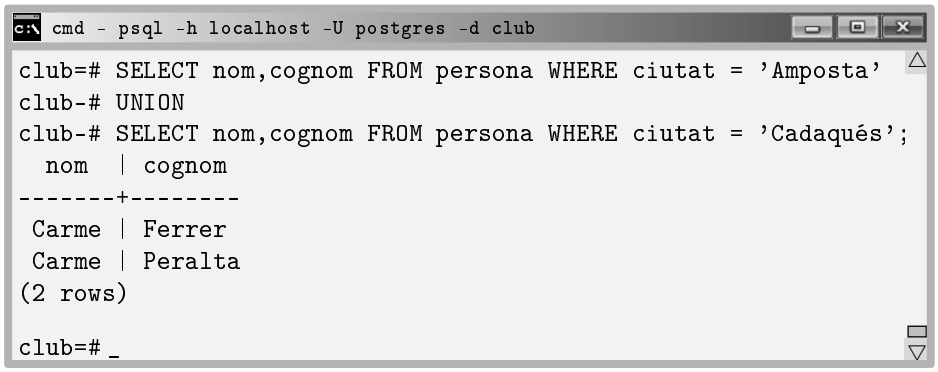
Pantalla 6.27. *Pas 1. Persones d'*Amposta*, 1 resultat.*

2. Seleccionem nom i cognom de les persones que viuen a *Cadaqués*.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT nom,cognom FROM persona WHERE ciutat = 'Cadaqués';
  nom   | cognom
--------+--------
 Carme  | Peralta
(1 row)

club=# _
```

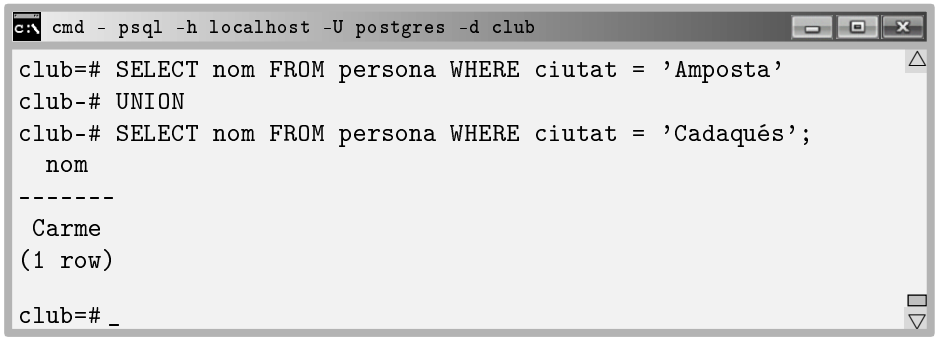
Pantalla 6.28. *Pas 2. Persones de* Cadaqués*, 1 resultat.*

3. Seleccionem nom i cognom de les persones que viuen a *Amposta* i de les persones que viuen a *Cadaqués*.



Pantalla 6.29. *Pas 3. Persones d'*Amposta *i de* Cadaqués*, 2 resultats.*

4. Seleccionem només el nom de les persones que viuen a *Amposta* i de les persones que viuen a *Cadaqués*.



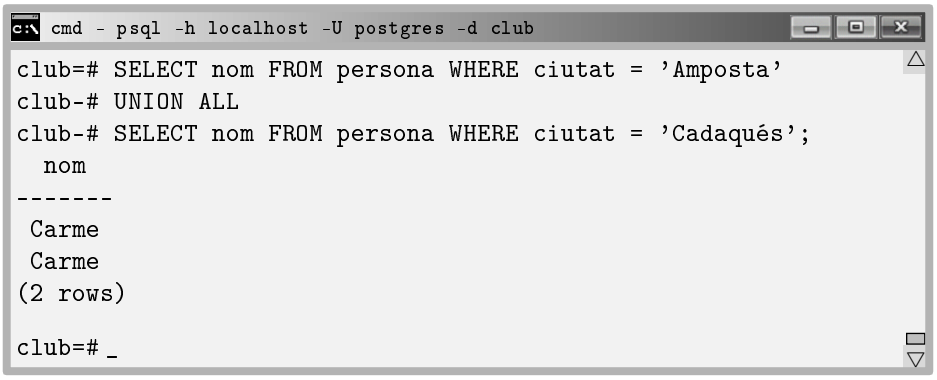
Pantalla 6.30. *Pas 4. Només el nom de les persones d'*Amposta *i de* Cadaqués*, 1 resultat.*

Fins aquí, ja es veu que l'operació es comporta de la manera més fidel a la teoria de conjunts. Cal entendre que en la Pantalla 6.29 apareixen dos resultats gràcies a que el cognom distingeix els dos registres. En el moment que deixem de projectar el cognom, passen a ser iguals, i l'operació d'unió els agrega per defecte.

Si es vol treballar amb la versió multiconjunt de l'operació cal afegir l'opció `ALL` després del mot `UNION`. L'operació `UNION ALL` és la versió multiconjunt de l'operació `UNION`.

Com en el cas del `SELECT` i el `SELECT DISTINCT`, les versions multiconjunt serveixen per quan amb les dades es vol comptar aparicions.

5. Seleccionem només el nom de totes les persones d'*Amposta* i *Cadaqués*.



Pantalla 6.31. *Pas 5. Només el nom de totes les persones d'Amposta i de Cadaqués, 2 resultats.*

En definitiva, per defecte les operacions `SELECT` i `UNION` actuen de maneres oposades, cosa que podria fer reflexionar.

Clàusula `EXCEPT`

L'operació de diferència de relacions s'implementa amb l'`EXCEPT`. La seva sintaxi és com la de l'operació `UNION` de la secció anterior. Es presenta a la Caixa 6.34,

```
SELECT A_1, A_2, ..., A_k
FROM r
WHERE p
EXCEPT
SELECT B_1, B_2, ..., B_k
FROM s
WHERE q;
```

Caixa 6.34. *Sintaxi de l'operació de diferència de relacions.*

amb esquemes $A_1, A_2, \ldots, A_k$ i $B_1, B_2, \ldots, B_k$ compatibles.

El resultat és la relació formada pels registres de la primera consulta que no estan a la segona. Noteu doncs que els que hi hagi a la segona que no siguin a la primera no tenen cap incidència en l'operació.

L'operació de diferència de relacions és un reflex de la definida en la Secció 5.4.5 per l'àlgebra relacional. Observeu que el mot *except* és un terme nou per aquest concepte.

Filosòficament, la mateixa sinapsi que hi ha entre la disjunció *o* i l'operació d'unió, o la conjunció *i* i la operació d'intersecció, és la que hi ha entre la diferència i la interjecció *i no*, altrament dir *però*. L'operació `EXCEPT` retorna els elements del primer operand però no del segon.

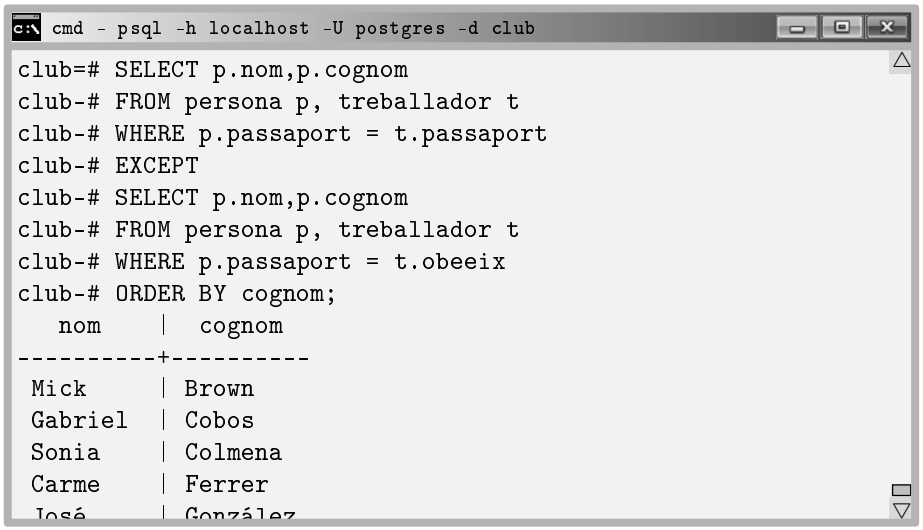
exemple 6.27. *Obtenir els noms i cognoms dels treballadors que no són caps de ningú.*

solució

```
SELECT p.nom,p.cognom
FROM persona p, treballador t
WHERE p.passaport = t.passaport
EXCEPT
SELECT p.nom,p.cognom
FROM persona p, treballador t
WHERE  p.passaport = t.obeeix
ORDER BY cognom;
```

La resolució de l'exemple 6.27 s'ha implementat a partir del raonament que els números de passaport dels treballadors que figuren a la columna `obeeix` de la taula `treballador` corresponen a caps d'algú. Per tant el que es demana és el nom de tots els treballadors excepte aquests.

Per altra banda, aquesta solució es pren la llibertat d'ordenar els registres alfabèticament. Aquests extrems són de sentit comú, i no haurien d'introduir cap tipus de confusió. L'execució d'aquesta consulta es pot veure en la Pantalla 6.32.



Pantalla 6.32. *Diferència de relacions.*

Clàusula INTERSECT

Per la implementació de l'operació d'intersecció de relacions s'utilitza la clàusula INTERSECT, i s'utilitza igual que les dues operacions anteriors. També és l'operació d'intersecció que s'ha vist en la Secció 5.5.1 del capítol de l'àlgebra relacional. El seu ús s'introdueix en la Caixa 6.35,

```
SELECT A_1, A_2, ..., A_k
FROM r
WHERE p
INTERSECT
SELECT B_1, B_2, ..., B_k
FROM s
WHERE q;
```

Caixa 6.35. *Sintaxi de l'operació d'intersecció de relacions.*

sent els esquemes $A_1, A_2, \ldots, A_k$ i $B_1, B_2, \ldots, B_k$ compatibles, com sempre.

El resultat és una relació formada pels registres de la primera consulta que també estan en la segona. I com en el cas de la unió, en moltes ocasions es pot prescindir d'aquesta operació amb el predicat adient en l'estructura fonamental. És a dir, amb proposicions conjuntives en el predicat del WHERE.

Igualment com abans, hi ha casos en que això resulta complicat, com en l'exemple 6.28.

exemple 6.28. *Quins socis practiquen* futbol *i* bàsquet*?*

solució

```
SELECT p.passaport,p.nom,p.cognom
FROM persona p, fa f
WHERE p.passaport = f.passaport
AND f.esport = 'futbol'
INTERSECT
SELECT p.passaport,p.nom,p.cognom
FROM persona p, fa f
WHERE p.passaport = f.passaport
AND f.esport = 'bàsquet'
ORDER BY cognom;
```

En particular, un conjunt de casos en els que resulta complicat prescindir de la intersecció són aquells en els que el predicat del WHERE hauria de tenir proposicions incidents en diferents registres d'una mateixa taula en el predicat. Noteu però, que la complexitat d'aquest tipus de consultes rau en la doble referència d'una mateixa taula en la consulta, i per tant es poden resoldre com es

mostra en la Caixa 6.36. O sigui que aquesta expressió es basa en un tractament de totes les parelles possibles de tuples d'una relació. Es tracta d'una solució alternativa.

$$r_1 \leftarrow \sigma_{x.esport=\texttt{'bàsquet'} \wedge y.esport=\texttt{'futbol'}}(\rho_x(\texttt{fa}) \times \rho_y(\texttt{fa}))$$
$$r_2 \leftarrow \Pi_{passaport}(r_1)$$
$$\Pi_{passaport,nom,cognom}(r_2 \bowtie \texttt{persona})$$

Caixa 6.36. *Expressió alternativa a la consulta d'intersecció de l'exemple 6.36.*

I com es pot veure en la Pantalla 6.33, per casualitat, són els dos de Berlín.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT p.passaport,p.nom,p.cognom
club-# FROM persona p, fa f
club-# WHERE p.passaport = f.passaport
club-# AND f.esport = 'futbol'
club-# INTERSECT
club-# SELECT p.passaport,p.nom,p.cognom
club-# FROM persona p, fa f
club-# WHERE p.passaport = f.passaport
club-# AND f.esport = 'bàsquet'
club-# ORDER BY cognom;
 passaport |  nom   |  cognom
-----------+--------+----------
 187448338 | Rita   | Derbeken
 294394950 | Klauss | Stallman

(2 rows)
club=#
```

Pantalla 6.33. *Intersecció de relacions.*

6.4.7 Operacions de Reunió de la Clàusula FROM

Les operacions de reunió de l'àlgebra relacional tenen la seva versió en SQL. Al llarg d'aquestes seccions suposarem que r és una relació amb esquema d'n atributs dels quals n'hi ha k que coincideixen, o sigui es diuen igual que k altres atributs d'una altra relació s, que en té m. Sense pèrdua de generalitat podem suposar que els k atributs comuns a les dues relacions són els k primers. És a dir, $r = r(C_1, \ldots, C_k, A_{k+1}, \ldots, A_n)$, i $s = s(C_1, \ldots, C_k, B_{k+1}, \ldots, B_m)$.

Totes les operacions de reunió es defineixen segons una condició. Se li diu *condició de reunió*, i pot ser o bé NATURAL, o bé USING, o bé ON. Forçosament n'ha d'aparèixer alguna de les tres i només una. Cada condició relaxa més i més el predicat intern de la reunió. És a dir, la reunió interna amb la condició NATURAL és la més restrictiva. La condició USING és una relaxació de la reunió natural, i la condició ON és una relaxació de la condició USING.

Si la reunió és NATURAL, llavors no cal dir res més. L'esquema resultant tindrà $m + n - k$ columnes, per qualsevol dels quatre tipus de reunió. Si es fa servir la condició USING s'ha de donar una llista ordenada d'atributs que sigui un subconjunt dels comuns, que normalment serà un subconjunt propi. Llavors les relacions resultants tindran, d'entre els atributs comuns, un sol cop els que s'hagi donat a la llista, i repetits els altres. Amb la condició ON cal donar algun predicat. En aquest cas els esquemes resultants tindran tots els $m + n$ atributs dels quals k seran repetits i per tant els noms de les columnes de la relació resultant, iguals.

En concret, les reunió interna natural selecciona files amb valors iguals en totes les columnes homònimes. Totes. Amb la condició USING selecciona files amb valors iguals en algunes, només algunes, de les columnes homònimes, i amb la condició ON, es pot establir qualsevol predicat com si fos una clàusula WHERE.

Tot seguit es veu la sintaxis de cada una de les operacions de reunió, i se'n mostren alguns exemples. En les properes seccions, el tipus de reunió es pot ometre. És a dir, els mots clau INNER i OUTER no calen. Pels exemples, s'utilitzen la relació `persona(passaport,nom,cognom,mail,ciutat)` que té 31 registres, i la relació `ciutat(ciutat,habitants,comarca)` que en té 21.

Clàusula `NATURAL INNER JOIN`

La reunió natural interna en versió SQL té l'aspecte de la Caixa 6.37,

```
SELECT X_1, X_2, ..., X_ℓ
FROM r NATURAL JOIN s
WHERE p;
```

Caixa 6.37. *Sintaxi de la reunió natural.*

sent $X_1, \ldots, X_\ell$ qualssevol dels atributs d'r o d's. Els esquemes de les relacions r i s, com s'ha dit més amunt. I p, un predicat que involucra atributs d'r, atributs d's, i constants.

El resultat d'aquesta consulta tindrà per esquema els atributs d'r i a continuació els que faltin d's. PostgreSQL posarà primer les columnes comunes, és a dir, $(C_1, \ldots, C_k, A_{k+1}, \ldots, A_n, B_{k+1}, \ldots, B_m)$.

Les tuples de la relació resultant són aquelles en les que els valors de les columnes homònimes coincideixin. A més a més, clar, si hi ha la clàusula `WHERE` han de satisfer el predicat.

Així doncs, recollim per fi el fruit d'haver anomenat igual, sempre que s'ha pogut, les claus primàries que les foranes que les apunten.

exemple 6.29. *Donar totes les dades de les persones amb les ciutats on viuen.*

solució
```
SELECT * FROM persona NATURAL JOIN ciutat;
```

Tenint en compte que `persona` té quatre columnes, `ciutat` tres, i hi ha una única columna homònima, l'esquema de la reunió natural tindrà sis columnes. En la Pantalla 6.34 es mostra la part inicial de la relació resultant.

```
cmd - psql -h localhost -U postgres -d club
club=#
club=# SELECT * FROM persona NATURAL JOIN ciutat;
    ciutat     | passaport |    nom    |  cognom  | habitants |    comarca
---------------+-----------+-----------+----------+-----------+------------
 Cadaqués      | 27673812M | Carme     | Peralta  |      2938 | Alt Empordà
 Badalona      | X3478937A | Carles    | Sanàbria |    219708 | Barcelonès
 Badalona      | 47548338K | Anna      | Sanàbria |    219708 | Barcelonès
 Castelldefels | 45493393Z | Jesús     | Hortesa  |     63077 | Baix Llobrega
 San Francisco | C00001549 | Michael   | Bros     |    805235 |
 Berlín        | 294394950 | Klauss    | Stallman |   3499879 |
 Berlín        | 187448338 | Rita      | Derbeken |   3499879 |
 Tiana         | 46372382N | Roser     | Puente   |      8221 | Maresme
 Roma          | C01X01TN  | Roberto   | Rietto   |   2796102 |
 Tiana         | 38433548L | Anna      | Margalef |      8221 | Maresme
 Castelldefels | 32234958K | José      | Sanlúcar |     63077 | Baix Llobrega
```

Pantalla 6.34. *Reunió natural de* `ciutat` *i* `persona`.

Observeu els títols de les columnes i el seu ordre. Primer les comunes, després les restants de la primera relació, i després les que quedin de la segona. En l'exemple 6.29 tenim com a columnes homònimes `ciutat`. Això fa que l'SGBD seleccioni aquelles files del producte cartesià en les quals el valor de la columna `ciutat` en les dues taules sigui el mateix. Si enlloc de ser una sola columna hi haguessin varies columnes comunes, totes elles apareixerien un sol cop, i el nombre de files seria inferior o igual a l'actual. Per altra banda, en la Pantalla 6.34 no apareixen les persones de les que no es conegui la seva ciutat, ni les ciutats de les que no hi ha cap persona, com es veurà tot seguit.

Clàusula `NATURAL LEFT OUTER JOIN`

La reunió natural externa per l'esquerra, a més dels registres resultants de la reunió natural interna, afegirà qualsevol fila de la relació de l'esquerra a la relació resultant, amb nuls a les columnes corresponents a la relació de la dreta.

exemple 6.30. *Donar les dades de totes les persones i si se sap, de les ciutats on viuen.*

solució `SELECT * FROM persona NATURAL LEFT JOIN ciutat;`

La part inicial de l'execució de l'exemple 6.30 es mostra en la Pantalla 6.35.

```
cmd - psql -h localhost -U postgres -d club
club=#
club=# SELECT * FROM persona NATURAL LEFT JOIN ciutat;
    ciutat     | passaport |    nom    |  cognom  | habitants |    comarca
---------------+-----------+-----------+----------+-----------+------------
 Cadaqués      | 27673812M | Carme     | Peralta  |      2938 | Alt Empordà
 Badalona      | X3478937A | Carles    | Sanàbria |    219708 | Barcelonès
 Badalona      | 47548338K | Anna      | Sanàbria |    219708 | Barcelonès
 Castelldefels | 45493393Z | Jesús     | Hortesa  |     63077 | Baix Llobrega
 San Francisco | C00001549 | Michael   | Bros     |    805235 |
 Berlín        | 294394950 | Klauss    | Stallman |   3499879 |
               | 39238229E | Sònia     | Aragall  |           |
 Berlín        | 187448338 | Rita      | Derbeken |   3499879 |
 Tiana         | 46372382N | Roser     | Puente   |      8221 | Maresme
 Roma          | C01X01TN  | Roberto   | Rietto   |   2796102 |
 Tiana         | 38433548L | Anna      | Margalef |      8221 | Maresme
 Castelldefels | 32234958K | José      | Sanlúcar |     63077 | Baix Llobrega
 Tredòs        | 38474483Z | Miquel    | Vila     |       154 | Val d'Aran
 Castelldefels | 42065765F | Camila    | Noriega  |     63077 | Baix Llobrega
               | 59119283Z | Pere      | Camprubí |           |
 Barcelona     | 27961020N | Pere      | Garcia   |   1611822 | Barcelonès
```

Pantalla 6.35. *Reunió natural externa per l'esquerra de* `ciutat` *i* `persona`.

Observeu l'aparició dels dos registres corresponents a persones de ciutat desconeguda, la Sònia Aragall i en Pere Camprubí, dels quals no es té informació sobre la seva ciutat en la base de dades. De fet, entre l'experiment de l'exemple 6.25 i l'execució d'aquesta consulta s'ha remuntat la base de dades, ja que com es pot veure en la Pantalla 6.35, la ciutat *Berlín* torna a existir. Els espais blancs es corresponen amb valors nuls. Aquesta relació té 31 registres, que tenint en compte que la relació és 1:N, corresponen a les 31 persones de la base.

Clàusula `NATURAL RIGHT OUTER JOIN`

La reunió natural externa per la dreta, a més dels registres resultants de la reunió natural interna, afegirà qualsevol fila de la relació de la dreta a la relació resultant, posant nuls a les columnes corresponents a la relació de l'esquerra.

En aquest cas les files s'afegiran al final, a no ser que es modifiqui l'ordre explícitament amb una clàusula `ORDER BY`.

exemple 6.31. *Donar les dades de totes les ciutats i les persones que hi viuen si se sap.*

solució
```
SELECT * FROM persona NATURAL RIGHT JOIN ciutat;
```

En la Pantalla 6.36 s'ha suprimit la part corresponent a quinze files centrals. Es mostra just per veure que la Sònia Aragall i en Pere Camprubí de l'exemple anterior han desparegut, i per altra banda, que al final de la relació han aparegut tres ciutats de la base de dades, en les que no hi viu ningú.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT * FROM persona NATURAL RIGHT JOIN ciutat;
     ciutat     | passaport |    nom     |  cognom  | habitants |    comarca
----------------+-----------+------------+----------+-----------+------------
 Cadaqués       | 27673812M | Carme      | Peralta  |      2938 | Alt Empordà
 Badalona       | X3478937A | Carles     | Sanàbria |    219708 | Barcelonès
 Badalona       | 47548338K | Anna       | Sanàbria |    219708 | Barcelonès
 Castelldefels  | 45493393Z | Jesús      | Hortesa  |     63077 | Baix Llobreg
 San Francisco  | C00001549 | Michael    | Bros     |    805235 |
 Berlín         | 294394950 | Klauss     | Stallman |   3499879 |
 Berlín         | 187448338 | Rita       | Derbeken |   3499879 |
 Tiana          | 46372382N | Roser      | Puente   |      8221 | Maresme
 Roma           | C01X01TN  | Roberto    | Rietto   |   2796102 |
 Tiana          | 38433548L | Anna       | Margalef |      8221 | Maresme
 Castelldefels  | 32234958K | José       | Sanlúcar |     63077 | Baix Llobreg
 Tredòs         | 38474483Z | Miquel     | Vila     |       154 | Val d'Aran
 Castelldefels  | 42065765F | Camila     | Noriega  |     63077 | Baix Llobreg
 Barcelona      | 27961020N | Pere       | Garcia   |   1611822 | Barcelonès
 ...
 Viladecans     | Y3439185D | Boris      | Santos   |     65444 | Baix Llobreg
 Rio de Janeiro |           |            |          |   6320446 |
 Esplugues      |           |            |          |     46667 | Baix Llobreg
 Atenes         |           |            |          |    664046 |
(32 rows)

club=# _
```

Pantalla 6.36. *Reunió natural externa per la dreta de* `persona` *i* `ciutat`.

Clàusula `NATURAL FULL OUTER JOIN`

Tanquem els tipus de reunió naturals amb la reunió natural externa completa. Ve a ser la unió de les dels dos costats. És a dir, en la relació resultant hi apareixen tots els valors de tots els registres de les dues taules.

exemple 6.32. *Obtenir totes les dades de les persones i les ciutats en una sola relació*

solució `SELECT * FROM persona NATURAL FULL JOIN ciutat;`

L'execució de l'exemple 6.32 es mostra en la Pantalla 6.37.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT * FROM persona NATURAL FULL JOIN ciutat;
     ciutat     | passaport |    nom    |  cognom  | habitants |    comarca
----------------+-----------+-----------+----------+-----------+------------
 Cadaqués       | 27673812M | Carme     | Peralta  |      2938 | Alt Empordà
 Badalona       | X3478937A | Carles    | Sanàbria |    219708 | Barcelonès
 Badalona       | 47548338K | Anna      | Sanàbria |    219708 | Barcelonès
 Castelldefels  | 45493393Z | Jesús     | Hortesa  |     63077 | Baix Llobrega
 San Francisco  | C00001549 | Michael   | Bros     |    805235 |
 Berlín         | 294394950 | Klauss    | Stallman |   3499879 |
                | 39238229E | Sònia     | Aragall  |           |
 Berlín         | 187448338 | Rita      | Derbeken |   3499879 |
 Tiana          | 46372382N | Roser     | Puente   |      8221 | Maresme
 Roma           | C01X01TN  | Roberto   | Rietto   |   2796102 |
 Tiana          | 38433548L | Anna      | Margalef |      8221 | Maresme
 Castelldefels  | 32234958K | José      | Sanlúcar |     63077 | Baix Llobrega
 Tredòs         | 38474483Z | Miquel    | Vila     |       154 | Val d'Aran
 Castelldefels  | 42065765F | Camila    | Noriega  |     63077 | Baix Llobrega
                | 59119283Z | Pere      | Camprubí |           |
 Barcelona      | 27961020N | Pere      | Garcia   |   1611822 | Barcelonès
 ...
 Roses          | X4534332C | Gabriel   | Cobos    |     19891 | Alt Empordà
 Viladecans     | Y3439185D | Boris     | Santos   |     65444 | Baix Llobrega
 Rio de Janeiro |           |           |          |   6320446 |
 Esplugues      |           |           |          |     46667 | Baix Llobrega
 Atenes         |           |           |          |    664046 |
(34 rows)

club=# _
```

Pantalla 6.37. *Reunió natural externa completa de* `ciutat` *i* `persona`.

Clàusula `INNER JOIN USING`

La condició `USING` serveix per quan $k > 1$. Això és, quan hi hagi varies columnes amb els mateixos noms. Si volem evitar el que ens resultaria de la reunió natural, i només lligar alguns dels atributs homònims però no tots, llavors utilitzem aquesta condició.

```
SELECT X_1, X_2, ..., X_ℓ
FROM r JOIN s USING (C_1, ..., C_j);
```

Caixa 6.38. *Sintaxi de la reunió interna amb la condició* `USING`.

sent $j < k$. És a dir, darrera la condició `USING` s'hi afegeix una llista entre parèntesis d'atributs comuns a les dues relacions. Es tracta d'un subconjunt propi de la interesecció d'atributs ordenat de manera que reflecteix l'ordre en la relació resultant.

O sigui, primer es va inventar la reunió natural, fins que va arribar algú i va dir que li molestava que lligués totes les columnes comunes. Llavors es va inventar la reunió interna amb la condició `USING` per relaxar la reunió natural. I després va arribar un altre i va dir que posats a fer, podrien relaxar encara més el predicat i deixar-lo obert a qualsevol altra expressió sense necessitat de fer referència a les columnes homònimes, i va aparèixer l'`ON`.

L'exemple 6.29 amb la condició `USING` és a la Pantalla 6.38.

```
cmd - psql -h localhost -U postgres -d club
club=#
club=# SELECT * FROM persona JOIN ciutat USING (ciutat);
    ciutat     | passaport |    nom    |  cognom  | habitants |    comarca
---------------+-----------+-----------+----------+-----------+------------
 Cadaqués      | 27673812M | Carme     | Peralta  |      2938 | Alt Empordà
 Badalona      | X3478937A | Carles    | Sanàbria |    219708 | Barcelonès
 Badalona      | 47548338K | Anna      | Sanàbria |    219708 | Barcelonès
 Castelldefels | 45493393Z | Jesús     | Hortesa  |     63077 | Baix Llobregat
 San Francisco | C00001549 | Michael   | Bros     |    805235 |
 Berlín        | 294394950 | Klauss    | Stallman |   3499879 |
 Berlín        | 187448338 | Rita      | Barbaken |   3499879 |
```

Pantalla 6.38. *Reunió interna amb la condició* `USING`.

Com que entre les dues relacions només hi ha una columna homònima, el resultat de la Pantalla 6.38 coincideix totalment amb el de la reunió natural de la

Pantalla 6.34. Però bé, serveix per il·lustrar l'execució. Observeu en l'ordre de les columnes que aquelles que sent comunes s'han agregat en el resultat es posen primer, `ciutat`. Si hi haguessin columnes comunes no presents en la clàusula `USING` s'haguéssin mostrat en el lloc corresponent de la taula corresponent.

Les clàusules `LEFT OUTER JOIN USING`, `RIGHT OUTER JOIN USING`, i `FULL OUTER JOIN USING` funcionen de manera anàloga.

Clàusula `INNER JOIN ON`

La condició `ON` serveix per introduir qualsevol tipus de predicat. Aquesta operació pren la forma indicada en la Caixa 6.39. La relació resultant de l'operació ve condicionada pel predicat obligatori p que pot implicar qualsevol atribut de les dues relacions d'entrada, i constants.

```
SELECT X_1, X_2, ..., X_ℓ
FROM r JOIN s ON p;
```

Caixa 6.39. *Sintaxi de la reunió interna amb la condició* `ON`.

Normalment, aquesta condició de reunió és utilitzada quan hi ha claus foranes que no es diuen igual que la primària a la que apunten. Observeu que si el predicat és una conjunció de proposicions que demanin la coincidència de columnes homònimes, el resultat és com la reunió natural però amb les columnes que es diguin igual repetides, i equivalents. A la Pantalla 6.39 es mostra l'exemple 6.29 resolt amb l'`INNER JOIN` amb la condició `ON`.

```
cmd - psql -h localhost -U postgres -d club
club=#
club=# SELECT * FROM persona JOIN ciutat ON persona.ciutat = ciutat.ciutat;
 passaport |    nom    |  cognom  |    ciutat     |    ciutat     | habitants |    coma
-----------+-----------+----------+---------------+---------------+-----------+--------
 27673812M | Carme     | Peralta  | Cadaqués      | Cadaqués      |      2938 | Alt Emp
 X3478937A | Carles    | Sanàbria | Badalona      | Badalona      |    219708 | Barcelo
 47548338K | Anna      | Sanàbria | Badalona      | Badalona      |    219708 | Barcelo
 45493393Z | Jesús     | Hortesa  | Castelldefels | Castelldefels |     63077 | Baix Ll
 C00001549 | Michael   | Bros     | San Francisco | San Francisco |    805235 |
 294394950 | Klauss    | Stallman | Berlín        | Berlín        |   3499879 |
```

Pantalla 6.39. *Reunió interna amb la condició* `ON`.

Observeu com sempre que es fa servir la condició `ON` l'esquema resultant és, rigorosament, la concatenació dels dos esquemes de les relacions entrants.

Fer servir predicats que no afectin a les claus primàries o foranes en la clàusula `ON` funciona, però no és habitual.

Així doncs, les raons que justifiquen l'ús de la reunió interna són primer, que les claus foranes no es diguin com les claus primàries. Aquesta és ben clara. Però encara hi ha un altre motiu pel qual aquesta operació està definida. Es tracta de quan no volem tots els atributs d'una reunió natural. Observeu que si de les relacions resultants de les Pantalles 6.34 i 6.39 es demanés per tan sols algun atribut concret donarien el mateix resultat.

En síntesi, quan es projecten només alguns atributs de la reunió natural el resultat és el mateix que quan es projecten els mateixos atributs en la reunió interna amb altres condicions de reunió, sempre en el benentès que el predicat lliga columnes amb el mateix nom.

Les clàusules `LEFT OUTER JOIN ON`, `RIGHT OUTER JOIN ON`, i `FULL OUTER JOIN ON` també funcionen de manera anàloga.

6.4.8 Valors Nuls

Essencialment hi ha dues clàusules directament relacionades amb el valor nul.

Predicat `IS NULL`

A l'hora d'establir els predicats per les seleccions en les clàusules `WHERE` o `JOIN ON` podem requerir que el valor d'un atribut de la relació de la consulta no existeixi, és a dir, que es tracti d'un valor nul. Alerta, la proposició $A =$ `NULL` és falsa, independenment de si A té valor a no. En qualsevol cas, `NULL` no és un valor. Per preguntar si A no té valor la sintaxis és A `IS NULL`.

Més en general, E `IS NULL` és un predicat on E pot ser qualsevol expressió formada d'atributs i constants. Això es presta a confusió, i per això ho experimentem en dues passes.

1. Demanar incorrectament les persones que no se sap on viuen fent

   ```
   SELECT * FROM persona WHERE ciutat = NULL;
   ```

   Observar que no hi ha resultats. Les comparacions amb nuls sempre retornen fals.

2. Demanar correctament les persones que no se sap on viuen, fent

   SELECT * FROM persona WHERE ciutat IS NULL;

   Observar com el resultat té les dues persones que ja coneixíem de la Secció 6.4.7

L'experiment es mostra en la Pantalla 6.40.

```
cmd - psql -h localhost -U postgres -d club
club=#
club=# SELECT * FROM persona WHERE ciutat = NULL;
passaport | nom | cognom | ciutat
----------+-----+--------+--------
(0 rows)

club=# SELECT * FROM persona WHERE ciutat IS NULL;
 passaport |  nom  |  cognom  | ciutat
-----------+-------+----------+--------
 39238229E | Sònia | Aragall  |
 59119283Z | Pere  | Camprubí |
(2 rows)

club=# _
```

Pantalla 6.40. *Experiment amb el predicat* IS NULL.

No cal dir que pel cas que interessi la pregunta contrària, o sigui requerir l'existència de valor en algun atribut en un predicat, llavors posarem

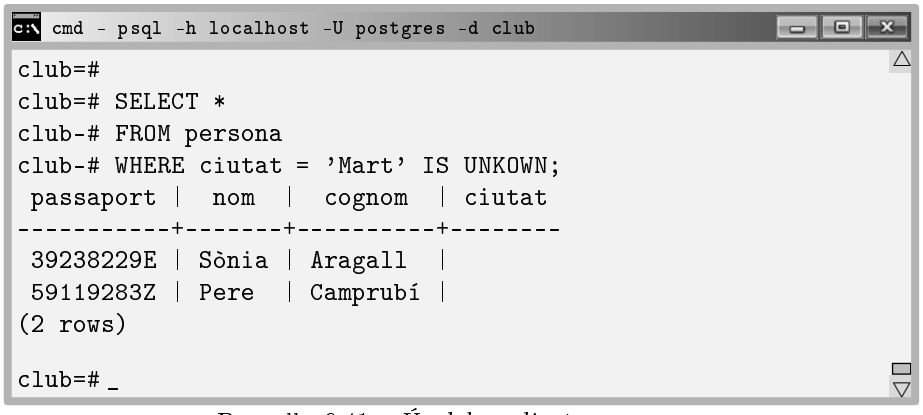
WHERE A IS NOT NULL.

Predicat IS UNKNOWN

Una cosa és preguntar per un valor i una altra preguntar pel resultat de l'avaluació d'un predicat. En la secció anterior s'ha vist com preguntar si existia el valor d'un atribut, o d'una expressió en general. Ara es mostra l'eina per averiguar si un predicat es resol en NULL. Així doncs, la diferència entre el predicat E IS UNKNOWN i el predicat E IS NULL rau en el tipus de l'expressió E. Si és una expressió booleana, convé utilitzar el predicat IS UNKNOWN és a dir, el resultat de l'avaluació lògica és deconegut. I en altres casos, s'utilitza el predicat IS NULL.

En la Pantalla 6.41 es pot veure una manera alternativa de resoldre l'experiment anterior, persones que no se sap on viuen. Ara es resol preguntant les persones que no se sap si viuen a *Mart*, és a dir, qualsevol valor concret.

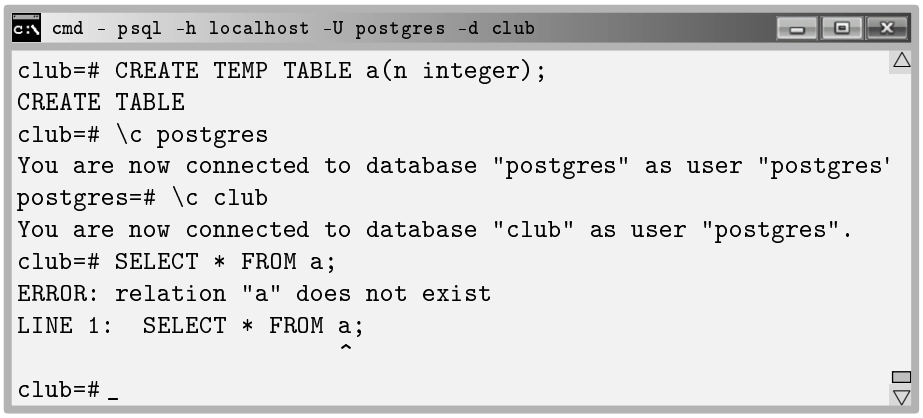
Com que l'avaluació resulta coneguda per totes aquelles persones de les que se sàpiga la ciutat, s'obté altre cop els mateixos registres que en la segona part de l'experiment de la secció anterior. Hi ha molta lògica en aquest comportament.



Pantalla 6.41. *Ús del predicat* IS UNKNOWN.

6.4.9 Consultes Aniuades

Arribats aquest punt, ja podem resoldre consultes més complexes. Consultes dins de consultes, cosa que ja s'ha anat anticipant en les darreres seccions. Si la cosa es complica massa, sempre queda l'opció de crear vistes, o taules temporals amb la CREATE TEMP TABLE, que crea taules que s'esborren automàticament al finalitzar una sessió. En la Pantalla 6.42 es mostra el fet.



Pantalla 6.42. *Les taules temporals desapareixen al sortir de la sessió.*

En l'estructura fonamental SELECT FROM WHERE qualsevol de les tres clàusules accepta consultes aniuades enlloc del que s'ha vist fins ara. Tot i així, posar un SELECT dins un SELECT o en un FROM és lleig, i de difícil legibilitat, com s'explica tot seguit.

En l'argument de la clàusula SELECT

Malgrat no sigui una bona pràctica, i tan sols convé saber-ho per fer proves intermitges, funciona el fet de posar una consulta aniuada en un SELECT. en Això permet obtenir valors d'atributs descriptius a partir de claus primàries utilitzant renomenaments. Han de ser consultes que només retornin una columna, per cada lloc on correspon a un atribut de la clàusula SELECT . És a dir, no es pot posar un SELECT que retorni dos atributs aniuat en un altre SELECT.

exemple 6.33. *Obtenir el cognom dels socis, amb el que paguen per cada esport.*

solució

```
SELECT (SELECT cognom
        FROM persona
        WHERE passaport = f.passaport),f.quota,f.esport
FROM fa f;
```

El PostgreSQL ens retornaria un error si en l'exemple 6.33 s'hagués demanat pel nom, a més del cognom.

En la Pantalla 6.43 es visualitza la part inicial de l'execució d'aquesta consulta, que llista 38 registres.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT (SELECT cognom
club(# FROM persona
club(# WHERE passaport = f.passaport),f.quota,f.esport
club-# FROM fa f;
  cognom   | quota |   esport
-----------+-------+-----------
 Peralta   | 18.35 | natació
 Sanàbria  | 18.35 | natació
 Sanàbria  | 21.50 | tennis
 Vila      | 21.50 | tennis
 Hortesa   | 21.50 | voleibol
```

Pantalla 6.43. SELECT *dins de* SELECT.

L'interès de l'exemple 6.33 rau en la sintaxi de la consulta aniuada. És important el fet que això no aporta potència expressiva addicional, ja que la mateixa consulta podria haver estat resolt sense aniuaments.

solució
```
SELECT cognom,quota,esport
FROM persona NATURAL JOIN fa;
```

En l'argument de la clàusula FROM

Igualment, l'SQL també suporta posar consultes enlloc de taules o vistes en la clàusula FROM. I també resulta mala pràctica per qüestions de legibilitat i convé evitar-ho. Aquests nyaps tan sols convenen a l'hora de desenvolupar, per fer proves.

Sempre que es posi una subconsulta a la clàusula FROM s'ha de renomenar.

exemple 6.34. *Socis que practiquen* futbol.

solució
```
SELECT passaport,nom,cognom
FROM (SELECT passaport
      FROM fa
      WHERE esport='futbol') AS futbol NATURAL JOIN persona;
```

La solució d'aquesta consulta es pot veure en la Pantalla 6.44.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT passaport,nom,cognom
club-# FROM (SELECT passaport
club(# FROM fa
club(# WHERE esport='futbol') AS futbol NATURAL JOIN persona;
 passaport |  nom   |  cognom
-----------+--------+----------
 27673812M | Carme  | Peralta
 X3478937A | Carles | Sanàbria
 294394950 | Klauss | Stallman
 39238229E | Sònia  | Aragall
 187448338 | Rita   | Derbeken
 46372382N | Roser  | Puente
(6 rows)

club=#
```

Pantalla 6.44. SELECT *dins de* FROM.

Altre cop, una solució alternativa i més clàssica es la següent.

solució
```
SELECT passaport,nom,cognom
FROM persona NATURAL JOIN fa
WHERE esport='futbol';
```

6.4.10 Consultes Aniuades en la Clàusula WHERE

Les consultes aniuades en la clàusula WHERE és on han de ser.

Sempre es pot comparar el valor d'un atribut amb el resultat d'una consulta enlloc de comparar-lo amb una constant o un altre atribut. Es fa simplement sustituint la constant o el segon atribut per la consulta entre parèntesis. En la Caixa 6.40 es presenta la sintaxis d'aquestes comparacions.

```
SELECT A_1, A_2, ..., A_k
FROM r
WHERE A_j = (
     SELECT B
     FROM s
);
```

Caixa 6.40. *Consulta aniuada.*

sent $r = r(A_1, A_2, \ldots, A_n)$, $A_i \subseteq D_i$ per $i = 1, \ldots, n$, $k \leq n$, $j \leq n$, i s una relació amb l'atribut $B \subseteq D_B$ com a mínim. L'operador d'igualtat pot ser qualsevol operador lògic de comparació. És important que els dominis d'A_j i B siguin compatibles, $D_j \sim D_B$. A més, cal poder demostrar prèviament que la consulta aniuada només tornarà un únic valor, o cap.

exemple 6.35. *Nom dels socis amb alguna quota més alta que algun sou d'algun treballador?*

solució
```
SELECT nom
FROM persona NATURAL JOIN fa
WHERE quota > (
      SELECT sou_base
      FROM nomines
);
```

Aquesta solució a l'exemple 6.35 és incorrecte i obtindrà un error per resposta informant que la subconsulta usada com a expressió, o sigui com a valor, retorna

més d'un registre. No cal esforçar-se massa per entendre que si es demana que la quota sigui més gran algun sou, es pot solucionar fàcilment demanant pel mínim sou enlloc de per tots els sous. Això farà que la subconsulta retorni un sol valor, i per tant la consulta sigui correcta, així.

solució
```
SELECT nom
FROM persona NATURAL JOIN fa
WHERE quota > (
      SELECT MIN(sou_base)
      FROM nomines
);
```

En la Pantalla 6.45 s'imprimeix la introducció de l'exemple 6.35, la primera aproximació que provoca un error, i la correcta que ens informa que no hi ha cap soci que compleixi aquesta condició.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT nom
club-# FROM persona NATURAL JOIN fa
club-# WHERE quota > (
club(# SELECT sou_base
club(# FROM nomines
club(# );
ERROR: more than one row returned by a subquery used as an expr
club=#
club=# SELECT nom
club-# FROM persona NATURAL JOIN fa
club-# WHERE quota > (
club(# SELECT MIN(sou_base)
club(# FROM nomines
club(# );
nom
---
(0 rows)

club=# _
```

Pantalla 6.45. *Ús de subconsultes com expressions.*

Més enllà d'utilitzar subconsultes com expressions, el fet de poder-les aniuar en més nivells fa d'aquest mecanisme el que té més capacitat expressiva de l'SQL. Aquesta potència ve reforçada per uns nous operadors lògics, que retornen cert o fals, als quals se'ls hi dóna un valor i una relació. Són operadors que ens responen qüestions de pertinença d'elements en conjunts, o d'existència de valors en les relacions. Es presenten a continuació.

Clàusula IN

La clàusula IN actua com un operador lògic en notació infixa, és a dir, entre els seus dos operands. Se li passa un valor i una relació amb l'esquema compatible amb aquest valor. Retorna cert si el valor pertany a la relació.

```
SELECT A_1, A_2, ..., A_k
FROM r
WHERE A_i IN (
     SELECT B_j
     FROM s
);
```

Caixa 6.41. *Clàusula lògica* IN.

Una intersecció de consultes és fàcilment implementable amb aquesta clàusula, com es veu en l'exemple 6.36.

exemple 6.36. *Socis que practiquin* futbol *i* bàsquet.

solució

```
SELECT passaport,nom,cognom
FROM fa NATURAL JOIN persona
WHERE esport = 'futbol'
AND passaport IN (SELECT passaport
            FROM fa
            WHERE esport = 'bàsquet');
```

Aniuant unes consultes dins les altres es poden resoldre qüestions que afectin a multitud de registres de taules diferents. Aquesta manera de treballar es basa en l'operació de canvi de nom.

El procediment mental per formular aquestes consultes més complexes passa per considerar que cada renomenament que fem en alguna taula és com crear una variable de tipus tupla amb els camps corresponents als atributs de la taula. Entès així, podem disposar de tuples per qualsevol taula de la base de dades que ens serveixen per comparar i restringir el que finalment serà la tupla resultant, que és la que apareix en cl clàusula SELECT principal.

Com es pot comprovar en l'exemple 6.37, amb aquest mètode es poden resoldre consultes que entren per una extrem del model ER per acabar seleccionant registres de l'altra punta. És clar que es tracta de les expressions més complicades.

exemple 6.37. *Persones conegudes pels treballadors de l'Alt Empordà amb sous superiors a 1000.00 .*

solució

```
SELECT p.passaport,p.nom,p.cognom
FROM coneix c JOIN persona p
ON c.es_coneguda = p.passaport
AND c.coneix IN (SELECT n.passaport
      FROM nomines n
      WHERE n.sou_base > 1000.00
      AND n.passaport IN (SELECT r.passaport
            FROM persona r NATURAL JOIN ciutat cc
            WHERE cc.comarca = 'Alt Empordà'
            )
);
```

La solució 6.37 conté dues tuples.

Alerta amb el concepte de contrari quan es treballa amb conjunts. La clàusula IN vol dir que algun element de la relació sigui el que es pregunta, o sigui que vol dir "pertany". I per tant amb la negació, NOT IN vol dir no pertany.

Clàusules SOME i ALL

El terme *some* es pot traduir per *algun*.

Un operador de comparació seguit de la clàusula SOME actua com un operador lògic entre els seus dos operands, que com amb la clàusula IN són un valor i una relació amb l'esquema compatible amb aquest valor. Segons el comparador, el valor donat, i els de la relació, retorna cert o fals.

En llenguatge formal es pot definir l'operació SOME de forma senzilla, com es presenta en la Caixa 6.42, que utilitza el símbol $\star$ per indicar qualsevol operador de comparació.

$$F \star \texttt{SOME}(r) \Leftrightarrow \exists\ t \in r \mid F \star t$$

Caixa 6.42. *Definició de la clàusula* SOME *amb el comparador genèric.*

sent F una expressió, r una relació, i el símbol estrella $\star$ qualsevol comparador del conjunt $\star \in \{=, \neq, <, >, \leqslant, \geqslant\}$.

En la Caixa 6.43 s'indica la manera d'utilitzar aquesta clàusula pel cas de la igualtat, en SQL.

```
SELECT A_1, A_2, ..., A_k
FROM r
WHERE A_i = SOME (
      SELECT B_j
      FROM s
);
```

Caixa 6.43. *Clàusula lògica* SOME.

El predicat de la clàusula WHERE de la Caixa 6.43 retornarà cert si algun valor de la columna B_j de la relació s és igual al valor de l'atribut A_i d'r. És important la presència de l'operador de comparació. Igual podria ser menor, major, diferent, menor o igual, i major o igual.

En la Figura 6.8 s'exposen quatre exemples d'avaluacions amb el corresponent resultat.

$$\left(5 < \texttt{SOME} \begin{array}{|c|}\hline 6 \\ \hline 5 \\ \hline 0 \\ \hline \end{array}\right) \text{ cert} \qquad \left(6 > \texttt{SOME} \begin{array}{|c|}\hline 6 \\ \hline 5 \\ \hline 0 \\ \hline \end{array}\right) \text{ fals}$$

$$\left(5 = \texttt{SOME} \begin{array}{|c|}\hline 6 \\ \hline 5 \\ \hline 0 \\ \hline \end{array}\right) \text{ cert} \qquad \left(5 \neq \texttt{SOME} \begin{array}{|c|}\hline 6 \\ \hline 5 \\ \hline 0 \\ \hline \end{array}\right) \text{ cert}$$

Figura 6.8: *Exemples de la clàusula* SOME.

Així doncs, $=$ SOME és equivalent a IN. Però en canvi, $\neq$ SOME no és equivalent a NOT IN. Estem en un terreny relliscós. Cal diferenciar entre el contrari de la pertinença i la pertinença del contrari.

Filosòficament, la clàusula SOME és en certa manera mitja generalització de la clàusula IN. Mitja generalització en el sentit que la clàusula IN sempre pregunta si el valor és igual algun de la relació. En canvi amb la clàusula SOME, es relaxa l'operador, i enlloc de l'igual podem preguntar diferent, o major, o menor, etcètera. En aquest sentit, és una generalització. Això no obstant, la versió negada de la clàusula IN, és a dir, la clàusula NOT IN es impossible de construir a partir de la clàusula SOME. Per això mitja generalització, ja que en positiu és més genèric, però no negatiu, no és reproduïble.

Anàlogament a la clàusula SOME, existeix la clàusula ALL. Com és ben sabut, el terme *all* es pot traduir per *tots*. En la Caixa 6.44 es mostra la definició formal d'aquesta clàusula.

$$F \star \texttt{ALL}(r) \;\Leftrightarrow\; \forall\ t \in r \mid F \star t$$

Caixa 6.44. *Definició de la clàusula* ALL *amb el comparador genèric.*

La diferència entre les dues operacions, doncs, rau en el quantificador existencial $\exists$ de la clàusula SOME versus el quantificador universal $\forall$ de la clàusula ALL.

S'utilitza com en la Caixa 6.45.

```
SELECT A_1, A_2, ..., A_k
FROM r
WHERE A_i <> ALL (
      SELECT B_j
      FROM s
);
```

Caixa 6.45. *Clàusula lògica* ALL.

En SQL el comparador de desigualtat $\neq$ s'escriu com menor major, <>. El predicat de la clàusula WHERE de la Caixa 6.45 retornarà cert si tots els valors de la columna B_j de la relació s són diferents al valor de l'atribut A_i d'r.

L'operador de desigualtat també pot ser qualsevol de comparació.

Altre cop es mostren quatre avaluacions en la Figura 6.9

$$\left(5 < \texttt{ALL} \begin{array}{|c|}\hline 6\\\hline 5\\\hline 0\\\hline\end{array}\right) \text{ fals} \qquad \left(6 > \texttt{ALL} \begin{array}{|c|}\hline 6\\\hline 5\\\hline 0\\\hline\end{array}\right) \text{ fals}$$

$$\left(4 = \texttt{ALL} \begin{array}{|c|}\hline 6\\\hline 5\\\hline 0\\\hline\end{array}\right) \text{ fals} \qquad \left(5 \neq \texttt{ALL} \begin{array}{|c|}\hline 6\\\hline 4\\\hline 0\\\hline\end{array}\right) \text{ cert}$$

Figura 6.9: *Exemples de la clàusula* ALL.

Es veu que $\neq$ ALL és equivalent a NOT IN. Però $=$ ALL no és equivalent a IN.

Així com en la Caixa 6.43 s'ha usat la igualtat, aquí en la Caixa 6.45 s'utilitza la desigualtat per ser els operadors més habituals en les operacions correspo-

nents. Els signes que s'utilitzen pels operadors de l'estrella de la definició formal, en SQL són $\{=, <>, <, >, <=, >=\}$.

Observeu que pel cas de la clàusula ALL es pot produir una situació absorvent, és a dir, que el resultat no depengui del valor pel qual s'està preguntant. Pel cas que s'utilitzi l'operador d'igualtat amb una relació on hi hagi valors diferents l'avaluació resultarà negativa per qualsevol valor que se li passi.

I també des d'un prisma més espiritual, en el fons, la clàusula ALL és l'altra meitat de la generalització de la clàusula IN.

Clàusula EXISTS

El terme *exists* es pot traduir per *existeix*, en tercera persona del singular del present simple.

És el mateix utilitzar la clàusula EXISTS amb una relació que preguntar si la relació no és buida. La definició d'aquesta clàusula en llenguatge formal és a la Caixa 6.46.

$$\texttt{EXISTS}(r) \Leftrightarrow r \neq \varnothing$$

Caixa 6.46. *Definició de la clàusula* EXISTS.

És a dir, que aquesta és la clàusula que ens permet preguntar si el resultat d'una expressió relacional és buida.

Per fer-ne ús en un predicat de la clàusula WHERE cal fer-ho com en la Caixa 6.47.

```
SELECT A_1, A_2, ..., A_k
FROM r
WHERE EXISTS (
     SELECT B_j
     FROM s
);
```

Caixa 6.47. *Clàusula lògica* EXISTS.

És una clàusula especialment útil per verificar l'existència de referències a una clau primària.

exemple 6.38. *Donar el nom de les ciutats de les que no hi hagi cap persona.*

solució
```
SELECT c.ciutat
FROM ciutat c
WHERE NOT EXISTS (SELECT *
            FROM persona p
            WHERE p.ciutat = c.ciutat
);
```

La resposta de l'exemple 6.38 és imprès a la Pantalla 6.46.

```
cmd - psql -h localhost -U postgres -d club
club=# SELECT c.ciutat
club-# FROM ciutat c
club-# WHERE NOT EXISTS (SELECT *
club(# FROM persona p
club(# WHERE p.ciutat = c.ciutat
club(# );
     ciutat
----------------
 Rio de Janeiro
 Esplugues
 Atenes

(3 rows)
club=# _
```

Pantalla 6.46. *Exemple d'ús de la clàusula* EXISTS.

I encara una consulta més complicada pot demanar que no existeixin elements en una intersecció, o en una diferència de relacions. Així es poden respondre qüestions com la de l'exemple 6.39.

exemple 6.39. *Donar el nom dels esports que practiquin socis de totes les ciutats de la comarca del Barcelonès.*

solució
```
SELECT DISTINCT f.esport
FROM fa f
WHERE NOT EXISTS (
            SELECT ciutat
            FROM ciutat
            WHERE comarca = 'Barcelonès'
      EXCEPT
            SELECT p.ciutat
            FROM fa ff, persona p
            WHERE ff.passaport = p.passaport
            AND ff.esport = f.esport
);
```

Analitzem aquest darrer exemple. Amb les dues primeres línies demanem els esports que practica algun soci. I a més les anomenem amb el prefix `f`. Cal retenir-ho al cap. El prefix `f` fa referència a la tupla que formarà part de la relació resultant si satisfà el predicat.

Llavors, amb el `WHERE` establim que a més a més per aparèixer en la solució de la consulta cada un d'aquests esports ha de complir una condició: que les ciutats del Barcelonès sigui un subconjunt de les ciutats dels socis que el practiquen.

Cal tenir en compte que

$$X - Y = \varnothing \Rightarrow X \subseteq Y.$$

I per tant, es pot comprendre fàcilment la solució 6.39.

Pam a pam. Els dos `SELECT`s interns computen els conjunts

- X : Conjunt de ciutats de la comarca del Barcelonès.
- Y : Conjunt de ciutats de socis que practiquen l'esport `f.esport`.

El `NOT EXISTS` és el $= \varnothing$.

En paraules, quan eliminant tots els elements d'un conjunt que estiguin en un altre ens quedem sense cap element, és que el primer conjunt és un subconjunt del segon. El fet que si d'un conjunt de gats treiem tots els animals ens quedem sense res, vol dir que els gats són un subconjunt dels animals. Per la mateixa raó, si treient a la llista de ciutats del Barcelonès les dels socis que practiquin l'esport tal s'obté la llista buida, llavors és que tal esport es practica per socis de totes les ciutats del Barcelonès.

En aquest capítol s'ha vist primerament com construir una base de dades a partir d'un model relacional utilitzant la creació de taules amb les corresponents restriccions d'integritat referencial. Després s'ha fet un recorregut a les estructures bàsiques de les consultes relacionals, moltes de les quals reflecteixen operacions de l'àlgebra relacional vistes en el Capítol 5, i altres més sofisticades pròpies d'estar operant amb multiconjunts i no amb conjunts com en aquell capítol. Tot plegat, proveeix el material necessari per formular respostes a les consultes que es puguin plantejar.

Capítol 7

El Sistema Gestor de Bases de Dades PostgreSQL

El Sistema Gestor de Bases de Dades PosgreSQL [7], és el programari de codi obert que més s'ajusta als estàndars SQL. A diferència del capítol anterior on tot el contingut és estàndar del llenguatge estructurat de consultes, en aquest capítol fem una immersió en aquest SGBD en particular. Això no vol dir que tot el que es mostra aquí sigui exclusiu d'aquest sistema. Significa que no es pot assumir que valgui per qualsevol. I malgrat les funcionalitats que s'exposen es mostren amb diferents facetes en els altres sistemes, les maneres de fer-ho no deixen de ser singularitats de cada un d'ells.

Aquest capítol comença introduint-se en la sintaxi de les sinopsis del manual. Aprendre a interpretar la gramàtica que s'hi utilitza. Gran part d'aquest capítol no és més que una selecció de les comandes més representatives amb les opcions més habituals que hi ha en la documentació de PostgreSQL, a [8]. És molt recomanable familiaritzar-se amb aquest tipus de sinopsis.

Llavors es fa una breu reestructuració de l'espai pel projecte del club esportiu, introduït en el capítol anterior. S'enllesteix tot plegat de cara a deixar l'aplicació acabada.

Tot seguit s'aborda l'administració d'usuaris i grups d'usuaris descrivint meticulosament la gestió de permisos i privilegis. Això introduirà un problema tan interessant com és el de la concurrència en l'accés i modificació de les dades. I condueix a l'aparició de les transaccions com a unitat indivisible d'acció modificadora de la base de dades.

Després es presenta una classificació dels components lògics de les bases de dades per aprofundir llavors en els components de control, o funcionals.

Més enllà de l'emmagatzemament de dades un SGBD disposa d'eines per gestionar-les que en el millor dels casos contenen fins i tot un llenguatge de programació natiu, com en el cas del PostgreSQL. Amb aquest llenguatge s'hauria de ser capaç de deixar l'aplicació pel club esportiu acabada, pel que fa al costat del servidor. Per això es fa una breu incursió en el llenguatge procedimental de l'SGBD PostgreSQL, que es diu PL/pgSQL. Aquest nom extrany prové de *procedural language PostgreSQL*. S'utilitzarà aquest llenguatge de programació senzill per implementar funcions. I després, les funcions serviran per afegir disparadors.

Arribats aquest punt, es dóna per closa l'adquisició de coneixement relatiu al disseny i la implementació de bases de dades i s'aborden altres temes implicats. Això inclou algunes eines bàsiques per al desenvolupament de programes d'interfície que facin de clients, així com els detalls de la instal·lació, iniciació i parada del sistema, còpies de seguretat, i alguns paràmetres bàsics de la configuració del servidor.

7.1 Documentació de PostgreSQL

La Figura 7.1 es tracta d'una captura de la part inicial de la plana web corresponent a la comanda `SELECT` de PostgreSQL.

Name

SELECT -- retrieve rows from a table or view

Synopsis

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    * | expression [ AS output_name ] [, ...]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY expression [, ...] ]
    [ HAVING condition [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start ]
    [ FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT ] [...] ]
```

Figura 7.1: *Plana web del manual.*

Cal concentrar-se. En la Figura 7.1, tot allò que està fora de claudàtors, és obligatori que aparegui en un `SELECT` . És a dir, el que està entre claudàtors és optatiu. La barra vertical significa disjunció exclusiva. O sigui, per cada barra vertical es pot posar el que tingui a l'esquerra o el que tingui a la dreta, però no les dues coses. Quan hi ha més de dues opcions, llavors utilitza claus per definir el conjunt de possibilitats, i entre cada una d'elles hi ha una barra vertical. La seqüència `[, ...]` després d'una paraula en minúscules, i per tant en negreta i en cursiva, significa que aquella mateixa última cosa es pot repetir. Tots els

símbols estan equilibrats en cada línia. Tants claudàtors o claus s'obren com es tanquen. Per agilitzar la comprensió d'aquest llenguatge, podem interpretar que on diu ***expression*** vol dir atribut. Encara que en rigor, també vol dir, per exemple, `2 * sou_base`.

Comencem. Per fer un SELECT forçosament cal escriure la mateixa paraula SELECT, i un asterisc o una expressió. Res més. Fixeu-vos-hi bé. Perquè no es fa explícit de cap manera, o sigui no es diu enlloc, que en el cas d'optar per l'asterisc, llavors és obligatori alguna cosa més. I en canvi, tot el que segueix és optatiu. És a dir, som nosaltres que hem de saber-ho. I per tant, les sinopsis no ho diuen tot, però sí que són molt útils.

Interpretant el que es diu en la Figura 7.1 doncs, es veu que després de la paraula SELECT es pot posar ALL, o bé DISTINCT. En aquest segon cas, o sigui si posem DISTINCT, llavors podem afegir-hi el mot ON seguit d'un atribut respecte el qual es donaran els diferents valors en la solució. Per exemple, amb la base de dades del club esportiu, la comanda

```
SELECT DISTINCT ON (ciutat) nom,ciutat FROM persona;
```

retorna una persona de cada ciutat. Quina? La primera que troba segons l'ordre que es doni en la clàusula ORDER BY, que en aquest cas, com que no es dóna, seria l'ordre per defecte.

Però no ens anem del tema. El propòsit d'aquesta secció és comprendre aquest tipus de sinopsis. Resulta materialment impossible explicar tot el que es pot fer amb un SELECT en una sola secció. Per tant, anant per feina, després se segueix amb o bé un asterisc, o bé un atribut, i si optem per aquest segon cas, llavors podem renomenar la columna amb l'operador AS, donant un ***output_name*** que podem intuir que vol dir un nom qualsevol. A més, també en aquest segon cas, podem repetir l'estructura afegint-hi més atributs, cada un del quals podem renomenar, o no.

Després podem posar FROM. Enlloc diu tampoc que si no posem FROM està prohibit posar WHERE. El significat de ***from_item*** es descriu més avall a la mateixa plana web. Seguidament podem posar la clàusula WHERE seguida d'una condició que també podem intuir que representa una expressió booleana.

A continuació hi ha la possibilitat d'agrupar segons algun atribut. No s'indica res respecte el tema de les funcions d'agregació. Tot i així, la persona que consulta el manual sap que si s'agrupa per algun atribut, és per agregar-ne d'altres. En la Figura 7.1 també s'explica que després, a l'agrupació s'hi pot establir condicions amb la clàusula HAVING, que acostuma a ser utilitzada si s'ha utilitzat el GROUP BY, encara que això tampoc queda plasmat en la sinopsis.

En canvi, el que sí que es mostra és la possibilitat de realitzar operacions lògiques entre relacions amb les clàusules corresponents. Després del HAVING podem posar una de tres, UNION, INTERESECT, o bé EXCEPT, i continuar amb un altre

SELECT, cosa que converteix la sinopsis en recursiva. A més, en qualsevol opció del trilema se li pot afegir l'operador ALL abans del SELECT aniuat.

La següent clàusula que s'il·lustra és la d'ordenació, amb la disjuntiva de si és ascendent, descendent, o bé d'una altra forma que no tractarem.

Finalment, l'opció de limitar el nombre de files de la relació resultant, o no, obtenint totes les files que resultin del còmput. En aquesta clàusula, l'opció ALL, és la que es dóna per defecte. Això serveix per programes client que utilitzin paginació de mida fixa per les transferències. Igualment, la clàusula següent, OFFSET indica el primer registre que es desitja obtenir. I de l'última opció, tampoc no en fem cap comentari.

En fi, més o menys, seria interessant comprendre bé aquestes sinopsis ja que són de gran ajuda a l'hora de consultar els formats de les instruccions.

7.2 Estructura del Projecte

El projecte pel club esportiu ha estat introduït en ls Secció 2.6. En el Capítol 5 s'ha mostrat la manera de transformar el disseny en un model relacional. I quan s'ha presentat el DDL s'ha implementat el model, i alguns arxius per disposar d'exemples d'insercions.

Ha arribat l'hora d'engrandir la jerarquia d'escripts que estructuren el projecte. Fins ara l'escript principal era el mateix que el de la creació de les taules. Ara però, lògicament l'arxiu de gestió d'usuaris ha d'estar a la mateixa alçada en l'estructura que l'arxiu de creació de taules, ja que són conceptes independents.

Per tant, l'objectiu d'aquesta transformació és que aquests dos arxius ocupin un mateix nivell a la jerarquia. Com que això provoca una certa proliferació d'arxius, en endavant els numerarem per ordre seqüencial que indiqui com s'han d'importar des del `psql` per crear l'aplicació.

Comencem fragmentant el contingut de l'actual `club.sql` en quatre arxius. El principal, un d'inicialització, el de creació de taules, i el d'inserts.

En la Caixa 7.1 es descriu el fitxer `1-init.sql` que tan sols conté la creació.

```
\echo -------------------- 1-init.sql

\c postgres
DROP DATABASE club;
CREATE DATABASE club;
\c club
```

Caixa 7.1. *Arxiu* `1-init.sql`.

L'arxiu `2-taules.sql` queda exclusivament amb la creació de les taules, sense els inserts. El seu contingut es detalla en la Caixa 7.2.

```
\echo -------------------- 2-taules.sql
\i 'comarca\\comarca.sql'
\i 'ciutat\\ciutat.sql'
\i 'persona\\persona.sql'
\i 'telefons\\telefons.sql'
\i 'coneix\\coneix.sql'
\i 'soci\\soci.sql'
\i 'treballador\\treballador.sql'
\i 'esport\\esport.sql'
\i 'fa\\fa.sql'
\i 'nomines\\nomines.sql'
```

Caixa 7.2. *Arxiu* `2-taules.sql`.

Llavors, renomenem l'arxiu `inserts.sql` a `3-inserts.sql`, creem un arxiu de nom `4-usuaris.sql`, i deixem l'escript principal com a la Caixa 7.3.

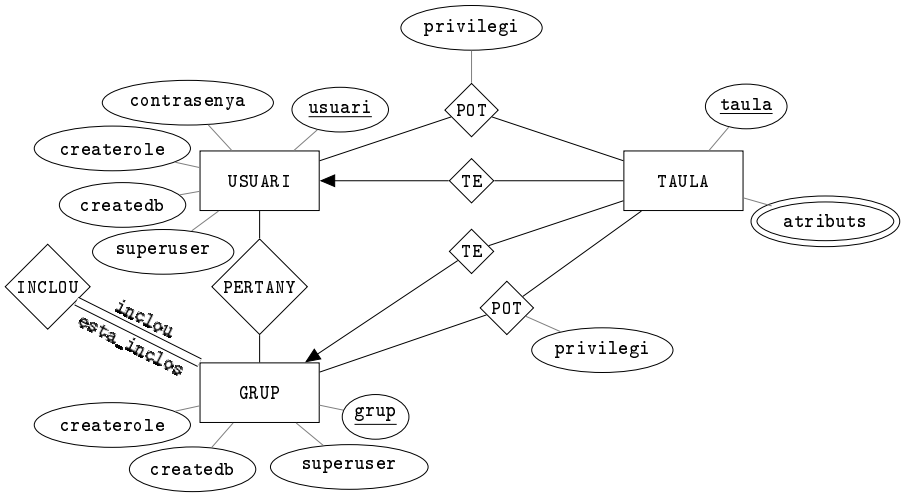
```
\set ON_ERROR_STOP on
\echo escript principal per la base de dades club
\echo ------------------------------------------

\i 1-init.sql
\i 2-taules.sql
\i 3-inserts.sql
\i 4-usuaris.sql
```

Caixa 7.3. *Arxiu principal,* `club.sql`.

7.3 Usuaris i Seguretat en la Base de Dades

Els usuaris no són exclusius d'una base de dades en concret, sinó de totes les bases de dades que se gestionen en un servidor.

Per una sola base, el Model 7.1 explica com es referencien els diferents objectes implicats en la seguretat de la base de dades.



Model 7.1. *Permisos i privilegis.*

El Model 7.1 és un model barroer. No és un model ben dissenyat, ja es veu. Els tres permisos que tenen els usuaris i els grups es poden entendre així com a booleans, tot i que en un bon disseny s'hauria de declarar un domini. Tal com es mostra serveix per fer-se una idea de com va el tema de permisos i privilegis, no es tracta de cap implementació. És exclusivament il·lustratiu, utilitzem els models ER per expressar idees.

La relació `TE` del Model 7.1 realment en PostgreSQL es diu `owner`, que significa amo. Qualsevol taula té un amo que pot ser un usuari o un grup. El terme `TAULA` s'utilitza per simplificar. Realment vol dir objecte, que podem entendre com taula, vista, funció,... i més coses. La relació `POT` realment no té un nom conegut. Els valors de l'atribut `privilegi` són cadenes de caràcters, tal com es veurà més endavant.

Noteu que els permisos són atributs de l'usuari, i els privilegis atributs de la relació M:N entre usuaris i taules.

7.3.1 Permisos

En la Secció 7.9.3 es veurà que la declaració de quins usuaris es poden connectar a quines bases de dades es dóna en un dels arxius de configuració.

Qualsevol usuari té permisos per crear taules, i altres objectes, en les bases de dades on es pugui connectar. Això el converteix en l'amo dels objectes que creï, i li permet donar privilegis a altres usuaris per poder-los llegir, modificar, i esborrar.

Cada permís s'identifica amb una paraula. N'hi ha tres. El permís de crear bases de dades, `CREATEDB`, el de crear usuaris i grups, `CREATEROLE`, i el que pot fer-ho tot, `SUPERUSER`. Estan pseudo ordenats. Cadascun, en certa manera, està incluït en el següent.

- `CREATEDB`. Aquest permís permet als usuaris la creació de noves bases de dades en el clúster.
- `CREATEROLE`. Amb aquest permís es poden crear usuaris i grups, però no superusuaris. Tampoc no es poden crear bases de dades directament. O sigui qui té permís de `CREATEROLE` pot crear usuaris que tinguin permís de `CREATEDB`, i connectar-se posteriorment amb aquest altre usuari per crear bases de dades, però no poden crear bases de dades amb la instrucció corresponent.
- `SUPERUSER`, o `CREATEUSER`. Aquests dos són el mateix, són sinònims, i qui el té és superusuari. La forma `CREATEUSER` està obsoleta, encara que segueix sent acceptada. Qui té pérmís de `SUPERUSER` també té el permís `CREATEROLE`, i el `CREATEDB`. Normalment tan sols hi ha un superusuari per cada base de dades. Els superusuaris són els únics que tenen tots els privilegis per llegir, modificar i borrar, tots els objectes de tothom.

Tot seguit fem una ullada a les comandes que serveixen per crear bases de dades, usuaris i grups. És a dir, les comandes que tan sols es poden fer segons quins permisos es tinguin.

Qui té el permís `CREATEDB` pot crear bases de dades fent servir l'ordre de la Caixa 7.4, coneguda ja des de la Secció 6.3.2.

```
CREATE DATABASE club;
```

Caixa 7.4. *Creació de bases de dades.*

Si es té el permís CREATEROLE, una forma habitual de crear un usuari és amb la instrucció que es mostra en la Caixa 7.5.

```
CREATE USER pepito;
```

Caixa 7.5. *Creació d'usuaris.*

Amb la instrucció de la Caixa 7.5 es crea un usuari amb codi *pepito*. Els noms dels usuaris en PostgreSQL comencen amb una lletra, i no hi ha diferència entre majúscules i minúscules. Sempre es guarden en minúscules.

Per donar permisos als usuaris que es creen es pot fer de dues maneres.

- En el moment de la creació, afegint el nom del permís després del nom de l'usuari. Per exemple la comanda CREATE USER pepito CREATEDB crea un usuari de nom pepito amb permís de creació de bases de dades.
- Posteriorment a la creació amb la comanda ALTER USER pepito CREATEDB.

D'aquestes dues mateixes maneres també es pot assignar la contrasenya. Afegint PASSWORD '123' en la creació, per exemple, o també posteriorment amb la comanda ALTER USER pepito PASSWORD '123'. Qualsevol usuari pot canviar-se el seu propi password d'aquesta segona manera, posant el seu propi codi d'usuari. O sigui, l'usuari pepito pot fer ALTER USER pepito PASSWORD '456'. En principi, un usuari sense contrasenya no pot entrar, i per tant no té massa sentit crear-lo. En la Secció 7.9.3 veurem com podem saltar-nos aquesta restricció, i entrar sense donar contrasenyes.

Els permisos es poden treure igual que es posen, prefixant el nom del permís amb NO. Per exemple, ALTER USER pepito NOCREATEBD treu el permís per crear bases de dades. I també amb els altres dos, NOCREATEROLE, i NOSUPERUSER.

Un usuari doncs, no és més que un conjunt de permisos i de privilegis. És previsible que diferents usuaris tindran els mateixos permisos. Per això apareix el concepte de grup d'usuaris. Amb el permís CREATEROLE, també es poden crear grups, tal com s'indica en l'exemple de la Caixa 7.6, on es crea un grup d'usuaris anomenat socis.

```
CREATE GROUP socis;
```

Caixa 7.6. *Creació de grups d'usuaris.*

Tots els usuaris que pertanyin a un grup tenen els permisos que hagin estat especificats per al grup, com a mínim, ja que per això serveixen els grups. Si un usuari pertany a diferents grups té els màxims permisos que es puguin tenir de totes les pertinences. I després, és possible que ells, en particular, en tinguin més.

A la Pantalla 7.1 és mostra la creació de l'usuari i el grup d'aquestes dues darreres caixes.

```
cmd - psql -h localhost -U postgres -d club
club=# CREATE USER pepito PASSOWORD '123' CREATEDB;
CREATE ROLE
club=# CREATE GROUP socis;
CREATE ROLE
club=# _
```

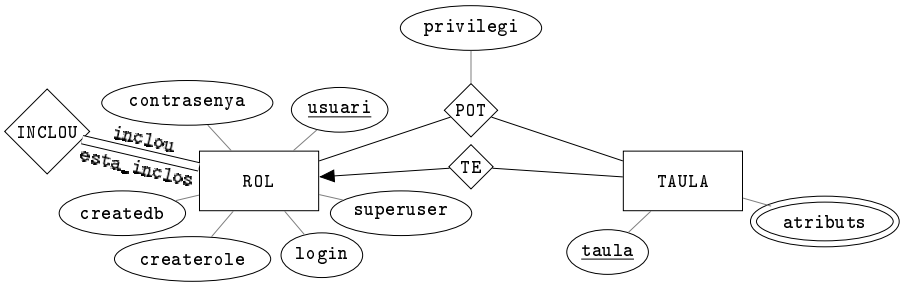
Pantalla 7.1. *Creació d'usuaris i de grups d'usuaris.*

Lògicament, als grups més que permisos se'ls hi acostuma assignar privilegis sobre objectes.

Sorprèn que PostgreSQL respongui `CREATE ROLE` tant pels usuaris com pels grups quan la comanda ha estat exitosa. Veiem per què.

O sigui, reflexionem. Un usuari es correspon amb uns permisos, i tots els usuaris d'un grup tenen els permisos que tingui assignats aquell grup. Ergo, un usuari i un grup és gairebé el mateix. Amb tot, irromp el concepte de rol. Canviem de filosofia.

Tant els usuaris com els grups són rols. I aleshores, ens inventem un permís que sigui `LOGIN`. De manera que el que fins ara consideràvem usuaris, a partir d'ara seran rols amb permís de `LOGIN`, i en canvi els que dèiem grups, seran rols sense aquest permís. Tot plegat impacta fortament en el Model 7.1, que queda transformat en el Model 7.2.



Model 7.2. *Permisos i Privilegis.*

I de fet, les comandes per crear usuaris i grups de les Caixes 7.5 i 7.6 són obsoletes tot i encara ser acceptades. Per això, són preferibles les versions de la Caixa 7.7.

```
CREATE ROLE pepito LOGIN;
CREATE ROLE socis;
```

Caixa 7.7. *Versions més actuals per la creació d'usuaris i grups.*

Com s'ha dit més amunt els rols poden ser amos d'objectes. Si el rol és un usuari, en el moment en que crea una taula, llavors es converteix en l'amo de la taula. També pot passar que sigui algú amb permís de `CREATEROLE` qui assigni un amo a una taula, fent `ALTER TABLE taula OWNER TO pepito`. Aquesta segona manera és l'única possible si el rol és un grup d'usuaris.

Quan algún usuari és l'amo d'una taula, o de qualsevol altre objecte, pot donar privilegis de lectura, modificació o eliminació a altres usuaris. Quan un grup és amo d'algun objecte, llavors tots els membres del grup es poden comportar com amos de l'objecte, és a dir, donar privilegis a altres usuaris.

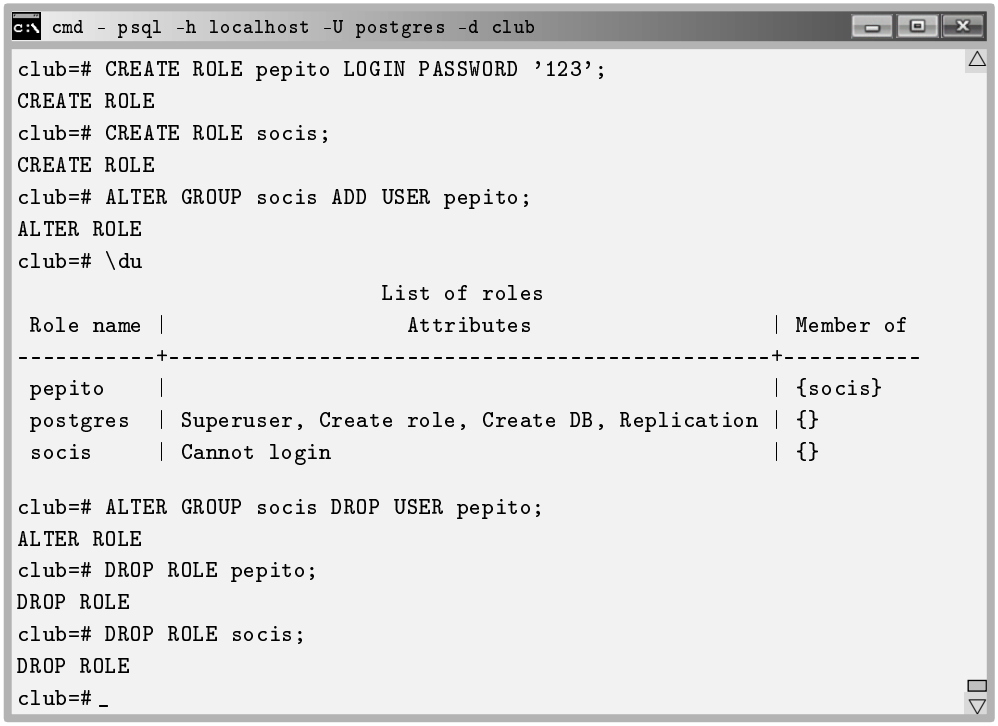
Bé, els conceptes d'usuari i grup es confonen en el de rol. Tot i així, per afegir i treure usuaris de grups hi ha la comanda `ALTER GROUP`, que se li dóna el nom del grup, `ADD` o `DROP`, i el nom de l'usuari, o sigui

```
ALTER GROUP socis {ADD | DROP} USER pepito;
```

Això no es pot fer amb rols. No es pot afegir un rol a un altre, ni un usuari a un grup fent `ALTER ROLE socis ADD USER pepito`. No funciona. I això és així perquè els rols com a usuaris són indiferents dels rols com a grups pel que fa als permisos, no a la jerarquia. Un usuari pertany a un grup, i un grup pot pertànyer a un altre grup, però no pot ser que un grup pertanyi a un usuari.

Per esborrar un usuari o un grup s'utilitza la comanda `DROP ROLE` seguit del seu nom.

A la Pantalla 7.2 es mostra un diàleg d'exemple. Primer es crea un usuari anomenat pepito, i un grup anomenat socis. Després s'afegeix l'usuari pepito al grup socis i es consulta la informació dels usuaris amb una comanda `psql`. En acabat, es treu l'usuari del grup, i s'eliminen usuari i grup.



Pantalla 7.2. *Inserció, modificació, i consulta dels usuaris i grups del clúster.*

Algunes consideracions respecte aquesta captura.

- La comanda `\du` de `psql` serveix per obtenir la informació dels rols existents.
- PostgreSQL anomena atributs d'usuari als permisos. Observeu que el rol `socis` no té permís de LOGIN, cosa que fa que com a rol sigui un grup.
- La tercera columna llista la col·lecció de grups als que pertany cada usuari.
- Hi ha dependències d'existència entre els elements. No es pot esborrar un grup si conté algun usuari.
- El permís de REPLICATION té que veure amb còpies de seguretat. Se'n parlarà amb més detall a la Secció 7.9.2.

Aportacions al projecte del club esportiu

Per a la base de dades del club esportiu, creem en aquest moment un superusuari que anomenem `club`. Això no vol dir que no ens haguem de tornar a connectar amb l'usuari `postgres`. En tot moment, l'usuari que executarà l'escript serà postgres. Això servirà a la llarga, quan es doni per vàlida l'aplicació. En la

Secció 7.9.3 limitarem l'accés del superusuari `club` a la base de dades `club` del servidor exclusivament.

Encetem així l'arxiu `4-usuaris.sql` amb el que es mostra a la Caixa 7.8. A més de la creació del superusuari, hi afegim la creació dels grups que finalment han de ser presents a l'aplicació.

```
DROP ROLE IF EXISTS club;
CREATE ROLE club LOGIN CREATEROLE PASSWORD '1';

DROP ROLE IF EXISTS socis;
CREATE ROLE socis;
DROP ROLE IF EXISTS comercials;
CREATE ROLE comercials;
DROP ROLE IF EXISTS entrenadors;
CREATE ROLE entrenadors;
DROP ROLE IF EXISTS administratius;
CREATE ROLE administratius;
```

Caixa 7.8. *Arxiu* `usuaris.sql`.

7.3.2 Privilegis

Els usuaris d'una base de dades tenen privilegis per actuar en les taules. Poden tenir privilegi de lectura, d'inserció, de modificació i d'esborrat. Realment n'hi ha més. Qualsevol usuari pot donar privilegis a un altre usuari, o a un grup d'usuaris, per poder llegir una taula si n'és propietari. I per això cal fer servir la instrucció de la Caixa 7.9.

```
GRANT SELECT ON taula TO pepito;
```

Caixa 7.9. *Garantir privilegis.*

Per poder utilitzar la comanda de la Caixa 7.9 cal o bé ser l'amo de la taula, o bé ser superusuari. Si el permís ha de ser d'inserció, doncs amb `INSERT`. Es poden posar varis privilegis separats per comes, com per exemple `GRANT SELECT,DELETE ON taula TO pepito`. També es pot dir tots, fent `GRANT ALL PRIVILEGES ON...` I també es pot assignar privilegis a columnes concretes de taules. `GRANT SELECT (passaport) ON TABLE persona TO pepito`. Si es vol donar el privilegi a tothom, `GRANT SELECT ON taula TO PUBLIC`.

Si es volen retirar els privilegis s'ha de fer tal com s'indica en la Caixa 7.10.

```
REVOKE SELECT ON taula FROM pepito;
```

Caixa 7.10. *Revocar privilegis.*

I com abans, amb les possibilitats `INSERT`, `UPDATE`, `DELETE`... i més que no veurem.

Per poder consultar els privilegis que hi ha associats a una taula en `psql` hi ha la comanda `\dp`. Com sempre, si li donem una taula ens dirà els privilegis que tingui associats. I si no, ens ho dirà de totes les taules. Però desgranem el cas d'una taula amb un experiment de quatre passes.

1. L'usuari `postgres` connectat a la base de dades `postgres` crea l'usuari `pepito`, amb contrasenya '1' i permís `CREATEROLE`. I es deconnecta.

```
cmd - psql -U postgres -d postgres
postgres# CREATE ROLE pepito LOGIN CREATEROLE PASSWORD '1';
CREATE ROLE
postgres# \q
```

Pantalla 7.3. *Creació d'un usuari amb dret a crear usuaris*

2. L'usuari `pepito` es connecta a la base de dades `postgres`, crea una taula buida que li diu `prova`, i consulta els privilegis amb `\dp prova`.

```
psql -U pepito -d postgres
C:\club> psql -U pepito -d postgres
Password for user pepito:
psql (9.3.4)
Type "help" for help.
postgres=> CREATE TABLE prova();
CREATE TABLE
postgres=> \dp prova
                            Access privileges
 Schema | Name  | Type  | Access privileges | Column access privileges
--------+-------+-------+-------------------+-------------------------
 public | prova | table |                   |
(1 row)

postgres=> _
```

Pantalla 7.4. *Consulta dels privilegis d'una taula tot just creada.*

3. Crea l'usuari `pepeta`, i li dóna permís per llegir la taula `prova`. Llavors torna a consultar els privilegis de la taula.

```
psql -U pepito -d postgres
postgres=> CREATE ROLE pepeta LOGIN PASSOWRD '1';
CREATE ROLE
postgres=> GRANT SELECT ON prova TO pepeta;
GRANT
postgres=> \dp prova
                              Access privileges
 Schema | Name  | Type  |   Access privileges   | Column access privil
--------+-------+-------+-----------------------+---------------------
 public | prova | table | pepito=arwdDxt/pepito+|
        |       |       | pepeta=r/pepito       |
(1 row)

postgres=> _
```

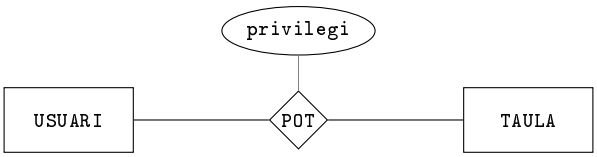
Pantalla 7.5. *Assignació i consulta de privilegis.*

4. Finalment, revoca el permís donat tornant a l'estat 2, i per tercer cop observa els privilegis.

```
psql -U pepito -d postgres
postgres=> REVOKE SELECT ON prova FROM pepeta;
REVOKE
postgres=> \dp prova
                              Access privileges
 Schema | Name  | Type  |   Access privileges   | Column access privil
--------+-------+-------+-----------------------+---------------------
 public | prova | table | pepito=arwdDxt/pepito |
(1 row)

postgres=> _
```

Pantalla 7.6. *Gestió de privilegis en una taula.*

Per l'anàlisi d'aquest l'experiment, primer convé conèixer els valors que pot prendre l'atribut privilegis que apareixia en el Model 7.2. En el Model 7.3 reprenem el fragment implicat en l'experiment.



Model 7.3. *Fragment superior del Model 7.2.*

Aquest atribut és de tipus TEXT, i conté una seqüència de lletres cada una de les quals té sentit per ella mateixa. Per l'amo d'una taula, el valor per defecte de l'atribut privilegi és

.

```
privilegi = 'arwdDxt'
```

La traducció del que això vol dir no és gens mnnemotècnica degut a que enlloc de basar-se en les paraules de l'SQL per prendre'n les inicials, es basa en els permisos que s'utilitzen en els sistemes d'arxius dels sistemes operatius línux. Pels profans en la matèria, tot seguit s'indica la semàntica d'aquestes cadenes de caràcters.

- `a`: Significa append, que és INSERT.
- `r`: Significa read, que és SELECT.
- `w`: Significa write, que és UPDATE.
- `d`: Significa delete, que és DELETE.
- `D`: Significa trucate, que és TRUNCATE. Aquest permís permet esborrar tots els registres d'una taula en una sola transacció.
- `x`: Significa que l'usuari pot referenciar la taula. O sigui, executar consultes que fagin joins amb aquesta taula.
- `t`: Permet executar els disparadors associats a la taula.

Aquest valor és el que s'assigna quan es fa un GRANT ALL PRIVILEGES...

El que pot resultar desconcertant de l'experiment és sens dubte que els resultats de les passes segona i quarta siguin diferents. És a dir quan s'acaba de crear, la columna `Access privileges` està buida. I després de donar el permís i treure-li a un tercer usuari, ja no. O sigui, que no es pot recuperar l'estat inicial.

Això és fruit d'un comportament extrany. La convenció és la següent. Degut a que estadísticament la major part de les taules de les bases de dades mantenen els privilegis per defecte de la creació, convenim en que un valor nul en la columna de privilegis d'accés, de la vista de privilegis, significa que l'amo té els privilegis `'arwdDxt'` concedits per ell mateix. Així, mentre no hi hagi cap modificació dels privilegis, ens haurem estalviat la feina de l'assignació. Ara bé, un cop s'ha fet alguna concessió de privilegis sobre la taula, llavors sí que s'estableix el valor real per aquesta columna.

És deduïble que el que ens mostra la comanda `\dp` ha de ser una vista, ja que si fos una taula no respectaria ni tan sols la primera forma normal que diu que el número de columnes d'una taula ha de ser un número fix. I la columna

`Access privileges` de les pantalles anteriors no només mostra un valor, sinó un conjunt de valors separats amb el signe `'+'`.

Analitzem meticulosament el contingut de la columna `Access privileges` de la Pantalla 7.5, que es mostra novament en la Figura 7.2.

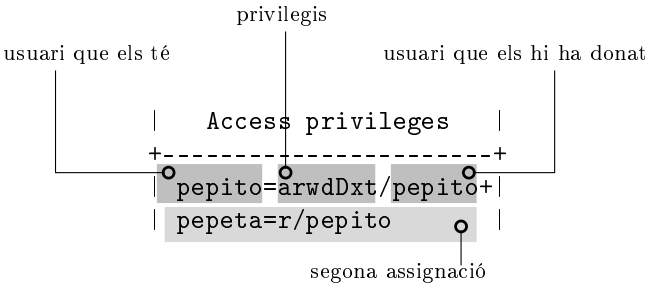


Figura 7.2: *Columna de privilegis d'accés a una taula*

La comanda `\dp prova` de la Pantalla 7.5 ha retornat una sola fila corresponent lògicament a la taula `prova`. Tot i així, aquesta fila té una columna, la de privilegis d'accés, que té més d'un valor, i per tant, es representa en una cel·la múltilínia. Doncs bé, cada una d'aquestes línies interiors a la cel·la representa una assignació de privilegis sobre la taula `prova` a algun usuari o rol. Aquestes assignacions tenen el format

```
usuari_que_els_te = privilegis_donats / usuari_que_els_ha_atorgat
```

i per això en la primera línia de la columna de la Figura 7.2 el donador del privilegi acostuma a correspondre amb el receptor, que és l'amo. També podria ser que després de la barra hi aparegués algun superusuari. Guardar qui ha donat els permisos a qui serveix per poder revocar-los en cascada.

En la mateixa Figura 7.2 podem observar que en la segona línia s'indica que la usuària `pepeta` té el permís de lectura, o sigui `SELECT` sobre aquesta taula, `prova`. I a més, que qui li va donar aquest permís en el seu moment va ser l'usuari `pepito`. El signe `'+'` serveix per separar assignacions.

Tota l'explicació anterior ha estat simplificada per tal de fer comprendre la forma d'assignar permisos i privilegis. De totes, la simplificació més notòria és la d'haver utilitzat el concepte de taula com a objecte de la base de dades. Cal entendre doncs, que enlloc de sobre una taula, els privilegis es poden establir sobre vistes, funcions, bases de dades, i més coses. Com a conseqüència, els valors de l'atribut privilegi mostrats en aquesta simplificació són exclusius dels privilegis definits sobre les taules. És a dir, si s'estigués parlant d'assignació de privilegis sobre una funció, per exemple, llavors la `'x'` dels privilegis de la Figura 7.2 voldria dir privilegi d'execució, i no de referència com és el cas.

7.3.3 Concurrència i Transaccions

Anomenem *transacció* a una seqüència de comandes que formen una acció indivisible en la base de dades.

Aquest concepte entra en escena en aquest moment perque previsiblement, amb l'augment del número d'usuaris actuant en la base de dades sorgiran problemes de concurrència. De totes maneres, el concepte de transacció afecta a aspectes que no tenen rea a veure amb treballar en un entorn multiusuari. Com s'explica tot seguit.

Quan fem un INSERT, estem executant una transacció. La diferència entre fer dos inserts d'una sola tupla constant, o fer un sol insert amb dues tuples, és que en el primer cas, si falla el segon insert (per exemple per clau primària repetida), llavors el primer quedarà fet igualment. En canvi si es tracta d'un sol insert amb dues tuples, és a dir una sola transacció, llavors si fallés el segon, el primer tampoc es produiria. De la mateixa manera que amb les insercions, amb les modificacions i les eliminacions.

En aquesta secció observarem problemes de concurrència i veurem com crear transaccions que facin el que puguin per resoldre'ls.

Com exemple de transacció, una transferència de diners entre dos comptes corrents. Si no es pot fer completament, que no es faci res. O sigui, que es tiri marxa enrera el que s'ha començat a fer. Això és una transacció. Una de dues. O es realitza del tot, o la base queda exactament com estava abans de començar-la.

Suposem que tenim una relació `compte = compte(titular,saldo)` amb dues tuples. En pepito té 1000€ i la pepeta 2000€. Es tracta de fer una transferència de 50€ que en pepito li dóna a la pepeta. En PostgreSQL, i més en general en SQL, es pot desgranar la transacció amb la seqüència d'instruccions de la Caixa 7.11.

```
UPDATE compte SET saldo = saldo - 50 WHERE titular = 'pepito';
UPDATE compte SET saldo = saldo + 50 WHERE titular = 'pepeta';
```

Caixa 7.11. *Dues transaccions en dues comandes.*

Però ep!, alerta. Si hi hagués un terratrèmol, o qualsevol altra cosa que pugui passar, a la meitat de l'execució d'aquestes dues instruccions els cinquanta euros haurien desaparegut, ja que el pepito els perdria, però la pepeta, degut al terratrèmol, no els hauria cobrat. I això no pot ser. Per resoldre aquest tipus

de problemes, podem transformar les dues transaccions en una de sola, com en la Caixa 7.12. Llavors, seguiríem sense haver pogut fer la transferència, però almenys en pepito recuperaria els 50€.

```
BEGIN;
UPDATE compte SET saldo = saldo - 50 WHERE titular = 'pepito';
UPDATE compte SET saldo = saldo + 50 WHERE titular = 'pepeta';
COMMIT WORK;
```

Caixa 7.12. *Transacció de dues comandes.*

La instrucció BEGIN serveix per començar una transacció. Internament, convé imaginar-ho com que crea una còpia de la base de dades que és la que veurà l'usuari que hagi executat el BEGIN. Llavors entra en un estat de prova, com si estigués jugant. Mentre duri aquest estat, des de la seva perspectiva la base ha de mantenir la coherència que tindria si no hagués fet el BEGIN. Pot fer insercions, modificacions, i eliminacions en les taules que vulgui, que tot plegat no deixarà de ser un miratge mentre no acabi la transacció.

La transacció és pot acabar bé, amb la comanda COMMIT WORK, que també es pot dir COMMIT a seques. Si acaba bé, els canvis fets per l'usuari seran visibles des de qualsevol altra connexió a la base de dades.

O pot acabar malament. Penedint-se. Això es fa amb la comanda ROLLBACK WORK, i llavors, tant si ha fet altes com baixes o el que sigui, el contingut de la base de dades es restaurarà al moment immediatament anterior haver començat la transacció. O sigui, abans del BEGIN.

En els dos casos es diu que la transacció ha acabat. Si és amb COMMIT, es diu que ha acabat exitosament. Si és amb ROLLBACK es diu que la transacció ha estat avortada.

Fem un experiment amb vuit passes.

1. Com a usuari `postgres`, o sigui com a superusuari, creem la base de dades `compte`, en la qual hi creem una taula `compte(titular,saldo)`. Per aquest experiment no calen claus primàries. En aquesta taula hi afegim les dues tuples. Això es fa amb les instruccions que es mostren a la Pantalla 7.7.

```
psql -h localhost -U postgres -d postgres
postgres# CREATE DATABASE compte;
CREATE DATABASE
postgres# \c compte
You are now connected to database "compte" as user "postgres".
compte# CREATE table compte(titular TEXT saldo DOUBLE PRECISIOI
CREATE TABLE
compte# INSERT INTO compte VALUES('pepito',1000),('pepeta',200(
INSERT 0 2
compte# _
```

Pantalla 7.7. *Creació de la taula per l'experiment de concurrència*

2. Creem dos usuaris que es diguin `anna` i `carles` amb permisos per llegir i modificar la taula `compte`. I ens desconnectem.

```
psql -h localhost -U postgres -d postgres
compte# CREATE ROLE anna LOGIN PASSWORD '1';
CREATE ROLE
compte# GRANT SELECT,UPDATE ON compte TO anna;
GRANT
compte# CREATE ROLE carles LOGIN PASSWORD '1';
CREATE ROLE
compte# GRANT SELECT,UPDATE ON compte TO carles;
GRANT
compte# \q

C:\club> _
```

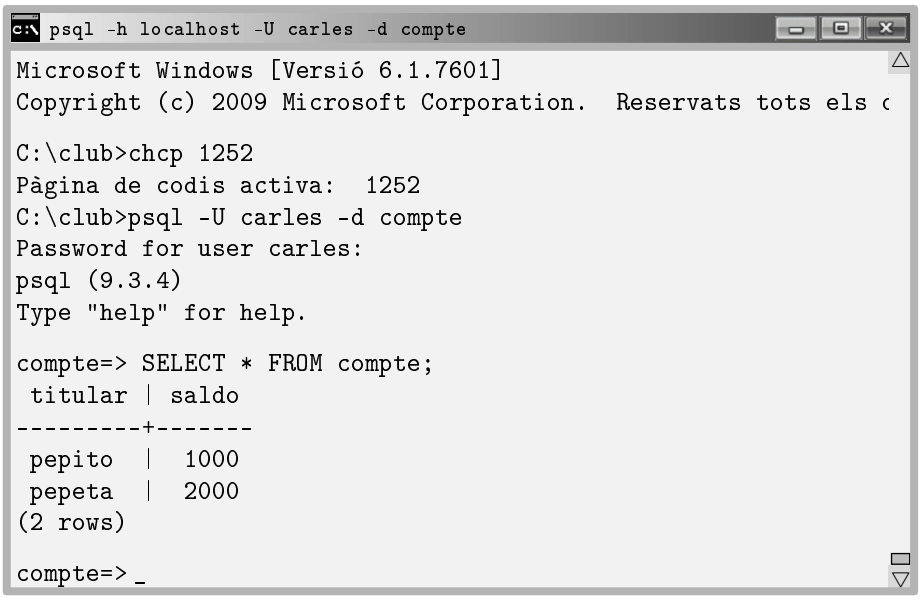
Pantalla 7.8. *Creació de dos usuaris, i assignació de privilegis*

3. Ens connectem a la base de dades `compte` amb l'usuari `anna`, i consultem el contingut de la taula `compte`.

```
psql -h localhost -U anna -d compte
C:\club> psql -h localhost -U anna -d compte
Password for user anna:
psql (9.3.4)
Type "help" for help.

compte=> SELECT * FROM compte;
 titular | saldo
---------+-------
 pepito  |  1000
 pepeta  |  2000
(2 rows)

compte=> _
```

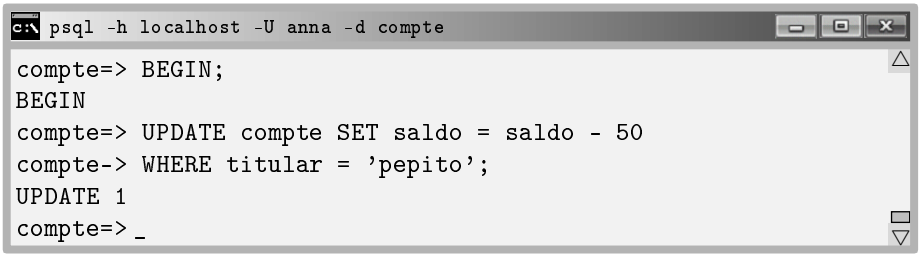
Pantalla 7.9. *L'usuària* `anna` *consulta la taula* `compte`.

4. Obrim una altra terminal, ens connectem amb l'usuari `carles`, i també consultem la taula `compte`. En aquest moment el contingut és igual en les dues terminals.



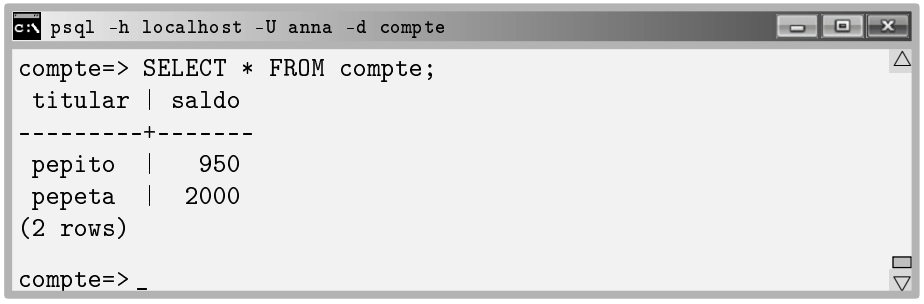
Pantalla 7.10. *L'usuari* `carles` *consulta la taula* `compte`.

5. L'anna inicia la transferència treient 50€ del compte d'en pepito.



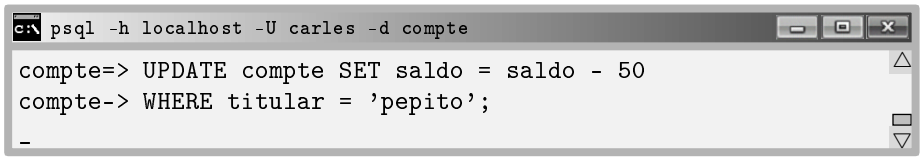
Pantalla 7.11. *L'usuària* `anna` *inicia la transferència de* 50€.

6. En aquest moment la visió de la base de dades segons en `carles` és diferent de la visió que té l'`anna`. De fet, si en `carles` consultés la taula compte obtindria el mateix resultat que en la Pantalla 7.10. És a dir, segons en `carles`, en pepito té 1000€. Observem com segons l'`anna`, en té 950.



Pantalla 7.12. *L'usuària* anna *consulta la taula* compte.

7. La incoherència va més enllà fins arribar a l'absurd. O sigui, que el PostgreSQL es queda penjat. Això passaria si en aquest moment en carles prtengués modificar el saldo d'en pepito.



Pantalla 7.13. *L'usuari* carles *intenta modificar el saldo d'en pepito.*

8. Observeu que en la Pantalla 7.13 el psql no mostra el prompt. La terminal està penjada. De fet, està esperant que l'anna es decideixi, i seguirà així fins que faci una de dues.

   - O bé acaba la transacció exitosament amb la comanda COMMIT, deixant un saldo de 900€.
   - O bé avorta la transacció, amb la comanda ROLLBACK, deixant un saldo de 950.

   De manera que l'actualització feta en la Pantalla 7.13 es fa en qualsevol cas, ja en carles no ha iniciat transacció i per tant, la seva comanda és executada sempre que sigui possible. Imagineu per exemple que l'anna després del BEGIN hagués eliminat l'usuari pepito. Llavors al fer COMMIT en la terminal de l'anna, a la terminal d'en carles hagués aparegut la resposta UPDATE 0.

La conclusió d'aquest experiment és que la cosa no és senzilla. Aquí només hi ha hagut un usuari iniciant una transacció, si l'altre també n'hagués iniciat una altra, i fessin coses incompatibles, i acabessin avortant o comprometent..., tot plegat es podria complicar bastant. I de fet, aquest problema apareix en la vida real quan es fa un overbooking d'entrades per una obra de teatre, o pels bitllets d'un vol.

Que un simple INSERT, UPDATE, o DELETE sigui per ell mateix una transacció és degut a la variable global AUTOCOMMIT de PostgreSQL. És booleana, i per defecte val cert.

7.4 Components Lògics d'una Base de Dades

Amb rigor, entenem per components lògics tots aquells objectes registrats amb el seu identificador d'objecte, OID, en el catàleg de cada una de les base de dades que allotja un SGBD. Els components lògics d'una base de dades és tot allò amb el què pot interaccionar qui les implementa. S'agrupen en dues vessants. Com sempre, l'espai i el temps. Distingim entre components de dades i components de control.

7.4.1 Components de Dades

Els components de dades són els dominis, les taules, vistes, i els índexos. Les comandes per tractar amb aquest tipus de components són les més estàndar en tots els dialectes de l'SQL, i per això ja han estat descrites en el Capítol 6. De fet, s'han vist els components de dades involucrats directament amb la creació de la base, que són tots excepte els objectes de tipus índex, que no se n'ha parlat, i per això aquí se'n fa una referència.

Atributs indexats

Un índex és un concepte associat a un atribut o conjunt d'atributs d'una taula de la base de dades. És un dels objectes que requereix informació interna, i per això té una repercusió en les metadades. Són objectes amb OID.

Buscar un valor en una seqüència ordenada d'n valors és un procediment logarítmic, com s'explica amb detall en [3]. Això vol dir proporcional al nombre de xifres que tingui n. En canvi, en una seqüència desordenada és lineal, o sigui proporcional al nombre de dades, n, que és molt més. Posem per cas, per buscar un valor entre mil, si estan ordenats es triga deu unitats de temps, i si no, mil. Això en el cas pitjor, o sigui tenint molta mala sort, que és del que parlem.

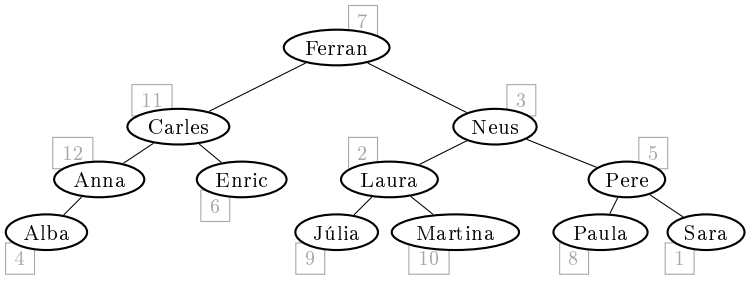
Crear un índex per un atribut és com dir que volem la taula ordenada segons aquell atribut. Podem ordenar la mateixa relació segons diferents atributs declarant-los índex cada un d'ells. Això provocarà guardar-se una estructura de dades arborescent per cada índex.

En la Figura 7.3*(a)* hi ha part del contingut d'una taula. La clau primària és un número enter. Per això convé imaginar que la taula està físicament ordenada segons aquest camp, tal com es mostra.

Declarar un índex per l'atribut `nom` d'aquesta taula provoca la creació i manteniment d'una estructura de dades que podem imaginar com la de la Figura 7.3*(b)*.

clau primària	nom	...
1	Sara	...
2	Laura	...
3	Neus	...
4	Alba	...
5	Pere	...
6	Enric	...
7	Ferran	...
8	Paula	...
9	Júlia	...
10	Martina	...
11	Carles	...
12	Anna	...

(a)



(b)

Figura 7.3: *(a) Part d'una relació; (b) Estructura de dades per un atribut índex.*

Observeu que a partir de la rel de l'arbre no trigarem en cap cas més de tres passes en trobar un element dins un conjunt de dotze. Aquesta estructura s'anomena arbre binari de cerca, o *binary search tree* BST, i es caracteritza pel fet que sent un arbre binari, qualsevol fill esquerre és ordenadament anterior al seu pare, i el pare anterior al fill dret, com es pot comprovar concentrant-se en la Figura 7.3*(b)*. Si es vol recórrer la taula completa segons l'ordre marcat per l'índex, la informació rellevant no és més que la permutació que representa l'inordre de l'arbre de la Figura 7.3, o sigui 4,12,11,6,7,9,2,10,3,8,5,1. Això significa la posició de les tuples en la taula ordenades segons l'atribut `nom`. Si tan sols interessa accedir algun element concret, llavors amb l'arbre de la Figura 7.3*(b)* trigarem tres unitats de temps per saber on és. L'arbre de la Figura 7.3*(b)* és un dels mètodes com pot treballar un índex. N'hi ha d'altres.

Tot i així, cal recordar que l'ordre físic dels registres és el cronològic d'inserció, no el de la clau primària. Clar, si no fos així caldria moure registres en cada inserció. I això no pot ser perquè triga un temps proporcional al nombre de registres que hi hagi a la taula, que poden ser molts centenars de milers. Físicament, inserir és afegir al final.

Es pot declarar índexos a funcions dels atributs. És a dir, enlloc d'ordenar segons els valors de l'atribut, ordenar segons el resultat de sotmetre aquests valors a alguna funció de transformació.

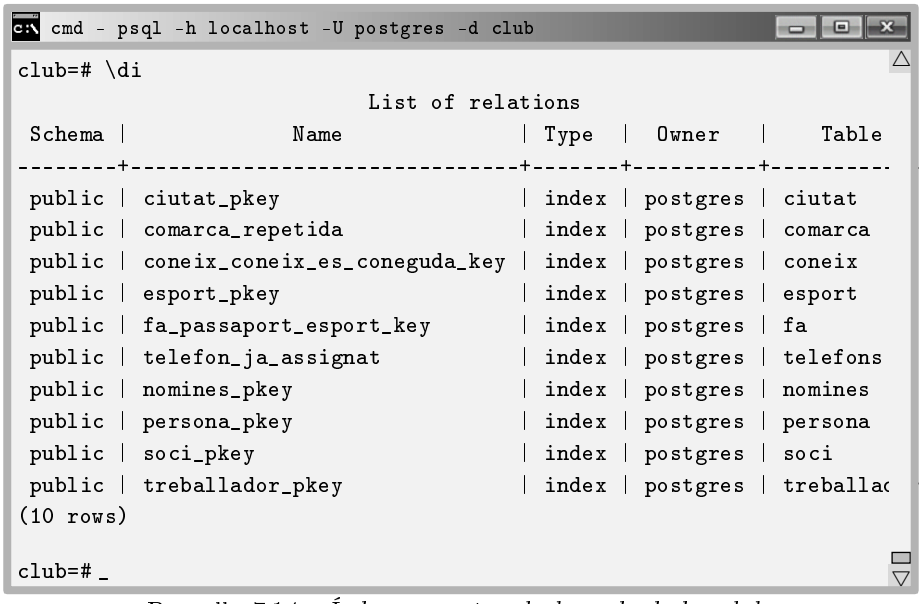
Automàticament es crea un índex per cada restricció d'unicitat que definim en la base. És indispensable que cada clau primària tingui el seu índex, ja que l'accés per clau primària ha de ser eficient necessàriament.

Només per curiositat, per tenir constància del moment en que es creen els índexos, es pot posar el paràmetre de configuració `client_min_messages` a nivell de `'DEBUG'`, amb la comanda

```
SET client_min_messages TO 'DEBUG';
```

just abans de la creació de la primera taula, en la primera línia de l'arxiu `2-taules.sql`, i tornar a importar el `club.sql`. Tingueu en compte que la comanda `\c` esborra els valors donats als paràmetres de configuració en la sessió. És ben clar, l'última línia de l'arxiu `1-init.sql`, que diu `\c club`, obre una sessió nova, i per tant anul·la les comandes `SET` prèvies. O sigui que si poséssim la comanda `SET` abans, no serviria per res.

En la Pantalla 7.14 es llisten els índexos creats en la base de dades club.



Pantalla 7.14. *Índexos creats a la base de dades club.*

La gestió dels índexos no té un impacte directe en la feina de desenvolupament d'una base de dades. La intenció d'aquesta secció tan sols ha estat donar a conèixer l'existència d'aquests objectes i la seva funció, que és agilitzar els tractaments amb les taules quan els accessos es produeixin per mitjà d'atributs indexats.

És a dir, els índexos serveixen pels atributs que no sent clau primària es pretén sovint una de dues, o bé accedir a la totalitat de la taula ordenadament segons l'atribut, o bé accedir a tuples que continguin valors concrets de l'atribut. Això és, ordenacions o cerques. Noteu doncs que si es tracta de funcions d'agregació, que recorren la taula completa per tractar els valors d'un atribut, llavors l'ordre és indiferent. I per tant no justifica la creació d'indexos.

Una base de dades pot crear els seus índexos addicionals amb la comanda `CREATE INDEX` i donant-li els atributs, encara que normalment ho fa automàticament establint les restriccions d'unicitat en la creació de les taules. Crear-los no es freqüent.

7.5 Components de Control

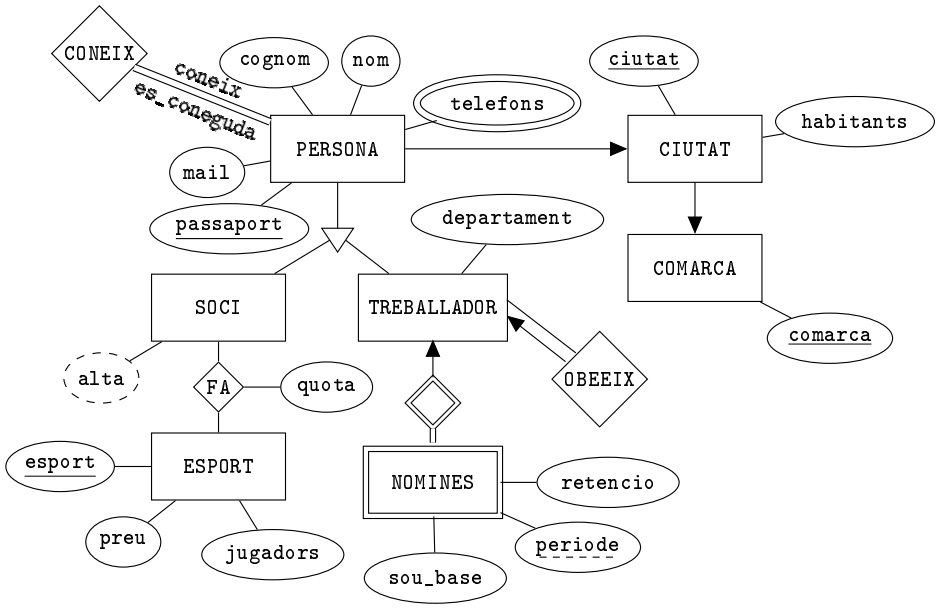
Entenem per components de control aquells que serveixen per processar les dades en instants concrets. Són programes curts, interpretats, que es poden executar per sol·licitud explícita de l'usuari client, o bé associats a accions que ocorren en la base dades. De totes formes són funcions.

Si es pretén poder-les executar de manera asíncrona, és a dir amb sol·licituds explícites de l'usuari, han de retornar algun tipus de dades conegut. En canvi, quan s'associen a accions concretes com insercions o modificacions de valors en taules, llavors es diuen funcions de tipus disparador, i només poden ser utilitzades per aquest efecte.

7.5.1 Implementació de l'Aplicació per al Club Esportiu

La part procedural de l'aplicació pel club esportiu que s'ha anat veient en els darrers capítols s'implementaria per rols. El primer objectiu és satisfer l'usuari administrador, després els socis, i finalment els treballadors.

En els exemples de les properes seccions s'implementarà primerament altes i baixes necessàries dels registres de les entitats del model ER del projecte, que es repeteix en el Model 7.4.



Model 7.4. *Model ER per l'aplicació del club esportiu.*

De totes les funcions que cal implementar n'hi ha una part de genèriques que sempre calen per totes les bases de dades. Són les que insereixen, modifiquen, o esborren els objectes del model. I després, hi haurà una col·lecció de funcions específiques per aquesta aplicació, que aniran lligades a la definició de requeriments per cada rol.

Implementació de funcions genèriques

Aquesta feina es troba automatitzada en moltes tecnologies de programació. S'acostuma a utilitzar el terme *mapeig* per reflectir en alguna interfície gràfica cada un dels objectes de la base de dades.

L'*hibernate*, [5], és un entorn d'objectes persistents en el que el programador pot treballar com si la base de dades estigués a memòria central podent fer insercions de la mateixa manera que un programa fa assignacions.

Utilitzar *binding* des de java per desenvolupar una aplicació d'interfície per una base de dades significa associar els objectes de la base a estructures de dades del programa client. Per exemple, per mapejar el resultat d'una consulta concreta es proporciona una estructura de tipus taula a més de la sentència SQL amb la consulta en qüestió. I per donar d'alta un objecte se li associa un formulari. Les tècniques de lligam entre la base de dades i les variables d'una aplicació van ser introduïdes per l'Access de Microsoft amb els formularis i després han estat reproduïdes profusament en multitud d'entorns, amb més o menys requeriments tècnics per part del programador.

Aquí es faran les funcions genèriques des del no-res per conèixer el que en moltes utilitats va empaquetat en interfícies gràfiques.

És clar que si és una feina àmpliament automatitzada és que ha de ser una feina rutinària. En molts casos aquestes funcions serien simples *wrappers*. Un wrapper, embolcall, és una funció que rep uns paràmetres i l'única cosa que fa és cridar-ne una altra passant-li els mateixos paràmetres que ha rebut. Serveixen per ajustar formats entre entorns i poca cosa més. En aquest text, no mostrarem implementacions de cap wrapper, perquè són prou senzilles com per poder-nos-les estalviar. En concret, per donar d'alta comarques, ciutats o esports, no es farà cap funció específica. L'aplicació client podrà fer directament les comandes SQL, `INSERT INTO`, `UPDATE SET`, o `DELETE FROM`.

En cap cas solaparem feina que fagi l'SGBD. Això vol dir que si es produeix un error d'integritat, serà PostgreSQL qui informi l'usuari llançant una excepció.

Implementació de funcions específiques

Aquesta tasca ha de reproduir les decissions de disseny i la definició de requeriments de l'aplicació concreta. Un manera vàlida de fer-ho és en base al menú d'opcions corresponent a cada rol.

Per cada tipus d'usuari es pot començar fent una funció que simplement retorni el nom de les funcions disponibles, de manera que les aplicacions clients puguin automatitzar les opcions en els seus menús.

7.6 Funcions

Les funcions es creen amb la comanda `CREATE FUNCTION` i s'esborren amb `DROP FUNCTION`. A l'hora de definir-les considerarem que l'estructura de les funcions es composa de tres parts. La capçalera, el cos, i una última línia constant en la que hi figura el nom del llenguatge de programació que s'està utilitzant. A l'abast del que es mostra en aquest llibre, sempre utilitzarem un mateix llenguatge com s'ha dit més amunt.

Els drets que té un usuari a executar codi escrit en un llenguatge es poden restringir. Precisament, la diferència que hi ha entre les plantilles *template0* i *template1* en una instal·lació de PostgreSQL consisteix en afegir la capacitat d'utilitzar el llenguatge PL/pgSQL a l'usuari postgres, en la plantilla 1. La plantilla zero és una base de dades totalment buida.

En la Figura 7.4 es presenta una funció "hola món", amb la terminologia usada per les seves parts. Fent un arxiu com aquest en la carpeta del projecte, que és com el banc de proves, podem reproduir el diàleg de la Pantalla 7.15, on a més, es mostra les dues maneres d'invocar la funció, encara que quan només retorna una columna com és el cas, resulten equivalents.

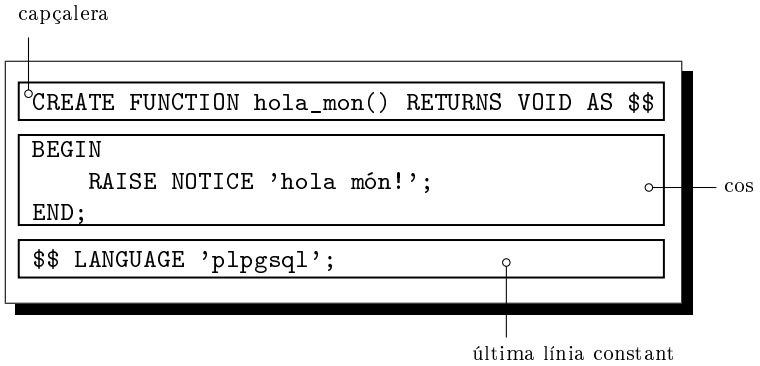
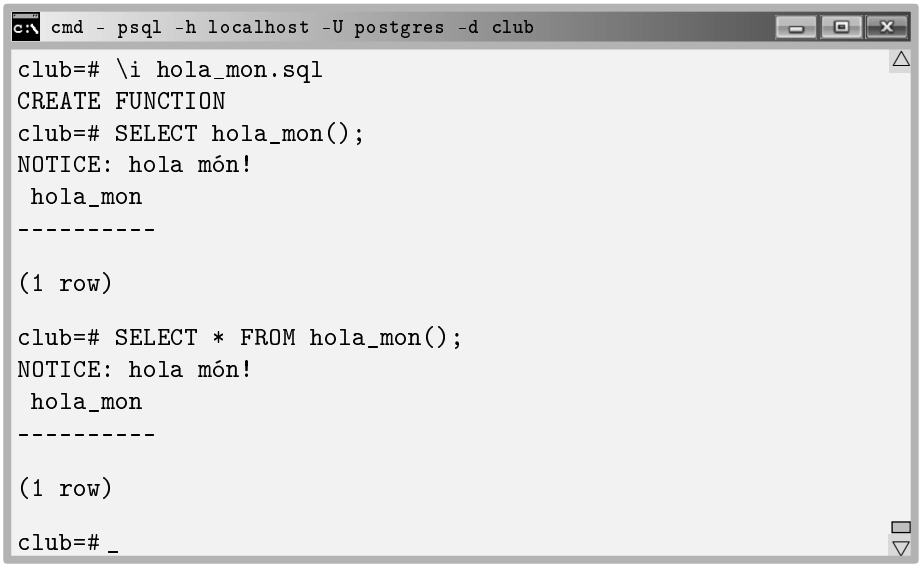


Figura 7.4: *Arxiu* `hola_mon.sql`

També és notable de la Pantalla 7.15, que el missatge apareix després de l'etiqueta NOTICE. En PL/pgSQL tots els missatges tenen un grau de severitat. Per això podem posar el paràmetre `client_min_messages` al nivell que `'DEBUG'`, el més laxe, com hem fet a la Secció 7.4.1 per veure els índexos. Cada nivell mostra tots els missatges dels nivells anteriors, més els seus. Si poséssim el valor d'aquest paràmetre a ERROR, llavors no apareixerien els missatges de severitat NOTICE. Respecte aquest extrem, podeu consultar el manual, [8].



Pantalla 7.15. *Importació i invocacions d'una funció.*

Que `SELECT nomfuncio()` sigui equivalent a `SELECT * FROM nomfuncio()` és degut a que la funció només retorna una columna. Aquestes dues comandes es diferencien en els resultats quan les funcions retornen varies columnes.

Immediatament després del missatge que la funció escriu amb la instrucció `RAISE NOTICE` per la terminal, acaba amb la mateixa sintaxi que si fos una relació. En cas que la funció només retorni una dada, és igual que quan retorna una relació amb una columna d'una fila. Com es pot contemplar en la Pantalla 7.15, la columna es diu `hola_mon`, i el seu contingut consta d'una única fila amb un valor nul, la línia en blanc. Aquest valor és el que ha retornat la funció.

7.6.1 Capçalera

Mirant l'escript de la Figura.7.4 ja es veu que la sentència per crear una funció es diu `CREATE FUNCTION`. També s'admet posar `CREATE OR REPLACE FUNCTION`. Quan desenvolupem funcions complicades que difícilment funcionaran a la primera, és altament recomanable aquesta segona opció, ja que si no, llavors haurem

d'esborrar la funció abans de tornar-la a importar. I això no és tan fàcil com podria semblar, ja que en PL/pgSQL s'ornamenten els noms de les funcions. Ho fa l'intèrpret del PL/pgSQL. Això vol dir que automàticament s'afegeix la llista de tipus dels paràmetres en el nom de la funció, i PostgreSQL les identifica amb tot plegat. Per això si es vol esborrar una funció, cosa que cal fer amb la instrucció `DROP FUNCTION`, s'ha de donar el nom de la funció seguit de la llista dels tipus entre parèntesis.

Seguint amb la creació de funcions, després del nom de la sentència va el nom de la funció, `hola_mon`, i la llista de paràmetres d'entrada en format `nom tipus`, que en la Figura 7.4 és buida. Llavors la paraula clau `RETURNS` en tercera persona de singular del present simple, ja que és ella, la funció, qui retorna alguna cosa. Segueix el tipus de l'expressió que es retorna. Si no es vol retornar res es pot posar `VOID`, que fa que es retorni un valor nul. I el mot clau `AS`. En SQL estàndar el cos de la funció va entre apòstrofs. En PL/pgSQL podem posar doble dòlar. Així evitem col·lisions amb les cadenes de caràcters constants que pugui haver-hi en el cos de la funció, com és el cas de la Figura 7.4. En SQL estàndar els apòstrofs que limiten les cadenes de caràcters constants es posen dos cops, fent el codi més enrevessat.

Una funció pot retornar una taula, `RETURNS TABLE(...)`. Per això, ha de fer tota la declaració de la taula, amb noms de columnes i tipus, en la capçalera. I llavors, són habituals les dues opcions següents.

- Que l'última instrucció del cos de la funció consisteixi en la instrucció `RETURN QUERY(SELECT...);`
- O si no, en el cas més complicat, una funció pot retornar un conjunt de registres retornant-ne un en cada iteració d'un bucle. Això és un número indeterminat d'elements formats per varis atributs. Cada element es retorna amb la instrucció `RETURN NEXT` $A_1, A_2, \ldots, A_n$, dins el bucle que després d'acabar fa un `RETURN` pelat per avisar que s'ha acabat.

7.6.2 Cos

El cos de la funció comença amb una secció que conté la declaració de variables locals, malgrat en la Figura 7.4 no n'hi ha. Aquesta àrea va precedida del mot clau `DECLARE`, i es poden declarar variables de tots els tipus que s'ha vist a la Secció 6.2.4, i algun més que es veurà més endavant en aquesta mateixa secció. La declaració de variables locals acaba amb el mot clau `BEGIN`. I entre `BEGIN` i `END` hi va el codi imperatiu que descriu l'acció que es realitza. En aquesta part s'hi pot posar qualsevol sentència d'SQL igual que si fos una instrucció més del llenguatge procedural. En aquest sentit, el PL/pgSQL és una generalització de l'SQL. A més, s'hi pot posar

- Assignacions a variables locals, en forma de composició seqüencial.

- Sentències alternatives entre els mots clau `IF ... THEN ... END IF;`
- Bucles indexats, `FOR ... LOOP ... END LOOP;`
- Bucles de condició final desconeguda `WHILE ... LOOP ... END LOOP;`
- Crides a altres funcions.

Hi ha una gran quantitat de funcions definides pels diversos tipus de dades. Totes elles disponibles des del cos de qualsevol funció. Per exemple, la funció `length(str)` a la que se li passa una cadena de caràcters i retorna un enter amb la seva longitud. O bé `substr(str,p0,len)` que retorna la subcadena de caràcters de la cadena `str` des de la posició `p0` que és un nombre enter, amb una longitud `len` que també és un enter. Per exemple, `substr('hola',2,1)` és la cadena formada per la lletra `'o'`.

En la Caixa 7.13 es mostra una funció senzilla. Es diu `set_bits`. Serveix per treure els accents o dièresis que pugui tenir el caràcter que se li passa com a paràmetre, i el retorni a set bits. Si el caràcter que se li dóna ja és de set bits, el retorna igual que el rep.

```
\echo ---------- funcio set_bits()

CREATE OR REPLACE FUNCTION set_bits(c CHAR) RETURNS CHAR AS $$
DECLARE
    i INTEGER;
    n INTEGER;
    origen TEXT DEFAULT 'ÀÁÈÉÍÏÒÓÚÜÑÇàáèéíïòóúüçñ';
    desti TEXT DEFAULT 'AAEEIIOOUUNCaaeeiioouucn';
BEGIN
    n = length(origen);
    FOR i IN 1..n LOOP
        IF c = substr(origen,i,1) THEN
            c = substr(desti,i,1);
            RETURN c;
        END IF;
    END LOOP;
    RETURN c;
END;
$$ LANGUAGE 'plpgsql';
```

Caixa 7.13. *Arxiu* `set_bits.sql`. *Exemple de bucle senzill.*

La forma de transformar un caràcter a set bits és simple. Es declara dues cadenes de caràcters, `origen` i `desti`. S'inicialitza `origen` amb els caràcters que es transformaran, i `desti` amb els caràcters corresponents transformats.

Aquestes inicialitzacions es fan en la mateixa declaració utilitzant l'operador DEFAULT. Així doncs, hi ha una correspondència posicional entre els caràcters de les dues cadenes.

La funció comença guardant en la variable entera `n` la longitud de la cadena `origen`, igual que podria ser la de la cadena `desti`. De fet, podria ser una constant igual a 24, però tal com està resulta més senzill poder ampliar els caràcters que es puguin voler transformar en el futur. I llavors entra en un bucle en el que compara la lletra que ha rebut com a paràmetre amb cada una de les que està en la cadena `origen`. Si la troba, retorna la lletra corresponent a la mateixa posició de la cadena `desti`.

Aquesta funció s'utilitzarà més endavant. I com que no depèn de cap taula convé guardar-la en una nova carpeta `club/_llibreria`, i afegir la importació al final de l'escript que inicialitza la base de dades `1-init.sql`, dient

```
\i '_llibreria\\_llibreria.sql'
```

prefixant els noms de l'escript i de la carpeta amb un guió baix perquè quedin els primers de la carpeta alfabèticament. I clar, dins la carpeta `_llibreria` caldrà crear-hi un escript que es digui `_llibreria.sql`, i que de moment tan sols hauria de tenir la línia

```
\i '_llibreria\\set_bits.sql'
```

però bé, ja queda preparat per poder afegir funcions addicionals sempre que siguin independents dels objectes de la base. Podem anar implementant funcions a mida que desenvolupem l'aplicació. Aquestes funcions que no depenen del contingut de la base de dades es diuen *immutables*.

7.6.3 Entrada i sortida de dades amb PL/pgSQL

Atenció a l'entrada i sortida de dades en les funcions fetes en PL/pgSQL. No hi ha cap manera d'introduir una dada des del teclat. El PL/pgSQL no inclou cap comanda per poder llegir dades del teclat perquè no calen.

Per poder imprimir dades a pantalla hi ha tan sols una instrucció, RAISE, que s'utilitza com s'ha vist a la Figura 7.4 per escriure una cadena de caràcters constant. El nom fa reflexionar. No es diu ni *print*, ni *write*, ni res que soni a acció freqüent. RAISE vol dir llançar. I aquest verb vol significar que no es té clar el destí on va a parar. És més, utilitzar aquest verb fa que sigui més normal llançar excepcions que missatges, com efectivament serà.

La comanda RAISE serveix sovint per posar traces. En qualsevol llenguatge de programació, posar una traça en el codi font significa escriure pel canal de sortida el valor d'alguna variable. Això es fa exclusivament en la fase de

desenvolupament, i és una feina que els entorns de desenvolupament integrats, com l'*eclipse* o el *netbeans* han anat estalviant als programadors. Anys enrera no hi havia forma alternativa de depurar un codi que no fos amb l'ús de traces. I així se segueix treballant quan es desenvolupa una base de dades des de línia de comandes com ho estem fent aquí.

Per averiguar quin valor té una variable en un punt concret de l'execució utilitzem la instrucció

```
RAISE NOTICE 'la variable pepito = %',pepito;
```

per una variable que es digués `pepito`. O sigui, el símbol del tant per cent serà subsituït pel valor real en el moment de l'execució. Igualment poden aparèixer varis tant per cents en la cadena de caràcters, i varies variables separades per comes enlloc de `pepito`. Llavors, l'associació entre els tant per cents i les variables és posicional. És d'agrair que la manera de fer-ho, amb el caràcter de tant per cent, no depengui del tipus de la variable, com habitualment succeeix en la majoria de llenguatges de programació d'alt nivell. Clar, per altra banda, tampoc podem, ni cal, especificar formats.

Una manera alternativa que podem fer ús de la comanda RAISE és amb el qualificador EXCEPTION enlloc de NOTICE. Si fem això avortarem l'execució de la funció en aquest punt, i retornarem el missatge donat com a text de l'excepció. Els programes client podran així atendre excepcions de tipus `SQLException` de la manera que tinguin definida, com és el mecanisme `try-catch`.

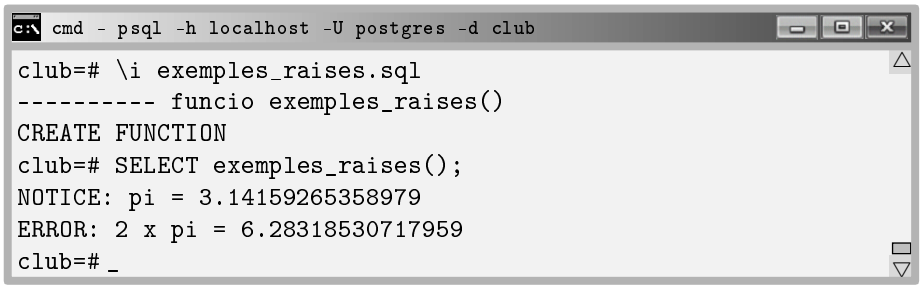
Per clarificar l'ús de la comanda RAISE del PL/pgSQL observem la funció de la Caixa 7.14. Amb esperit divulgatiu emetem un missatge calculant el número π com a quatre cops l'arctangent d'1.0.

```
\echo ---------- funcio exemples_raises()

CREATE FUNCTION exemples_raises() RETURNS INTEGER AS $$
DECLARE
    pi DOUBLE PRECISION;
BEGIN
    pi = 4 * atan(1.0);
    RAISE NOTICE 'pi = %',pi;
    RAISE EXCEPTION '2 x pi = %',2*pi;
    RAISE NOTICE 'Això ja no s''escriurà';
END;
$$ LANGUAGE 'plpgsql';
```

Caixa 7.14. *Arxiu* `exemples_raises.sql`. *Exemples d'ús de la comanda* RAISE.

Es demostra que estem tractant amb un intèrpret més que amb un compilador perquè malgrat dir en la capçalera que es retorna un enter, la carència d'una instrucció RETURN en el cos no impedeix que pugui ser executada.

En la Pantalla 7.16 es detalla el diàleg resultant de la importació i execució de la funció `exemples_raises()`.



Pantalla 7.16. *Execució de l'escript de la Caixa 7.14.*

És clar que la instrucció RAISE EXCEPTION avorta l'execució. Això fa que no surti ni el missatge següent ni l'error que diria que ha arribat a final de la funció sense trobar una instrucció RETURN. Tampoc surt la impressió tipus relació amb el títol de la columna que apareixia en la Figura 7.4. Observeu que el nivell de severitat de l'últim missatge, ERROR, va lligat al final de l'execució.

7.6.4 Consultes d'actualització en PL/pgSQL

Les consultes d'actualització es fan exactament igual que quan es treballa interactivament amb el `psql`, podent usar variables on fins ara s'usaven valors constants. Dit això ja podem implementar les dues funcions necessàries per donar d'alta. Les de socis i treballadors.

A la Caixa 7.15 es descriu la funció `alta_soci()`. Rep les dades dels socis i les distribueix entre les taules `persona` i `soci`. A més, noteu que establim el valor de l'atribut derivat `alta` de la taula `soci`. La variable `alta` de tipus TIMESTAMP WITH TIME ZONE té un caràcter merament il·lustratiu, i és perfectament prescindible.

El valor de retorn consisteix en el nombre de socis que s'ha inserit, que pot ser un o cap. Si s'arriba a executar el RETURN és que es tracta d'un cas exitós, que pot no ser sempre, ja que les dues transaccions prèvies podrien resultar en una excepció avortant l'execució.

En la Caixa 7.15 donem d'alta la persona abans que el soci per no produir un error d'integritat referencial. Respecte aquesta possibilitat, o sigui que el

passaport que pretenem inserir com a soci no existeixi en la taula persona, queda resolta fent-ho en aquest ordre.

```
\echo ---------- funcio alta_soci()

CREATE FUNCTION alta_soci(p TEXT, n TEXT, c TEXT,
         m domini_mail, ci TEXT) RETURNS INTEGER AS $$
DECLARE
    alta TIMESTAMP WITH TIME ZONE;
BEGIN
    INSERT INTO persona VALUES(p,n,c,m,ci);
    alta = now();
    INSERT INTO soci VALUES(p,alta);
    RETURN 1;
END;
$$ LANGUAGE 'plpgsql';
```

Caixa 7.15. *Arxiu* `alta_soci.sql`. *Consultes d'actualització.*

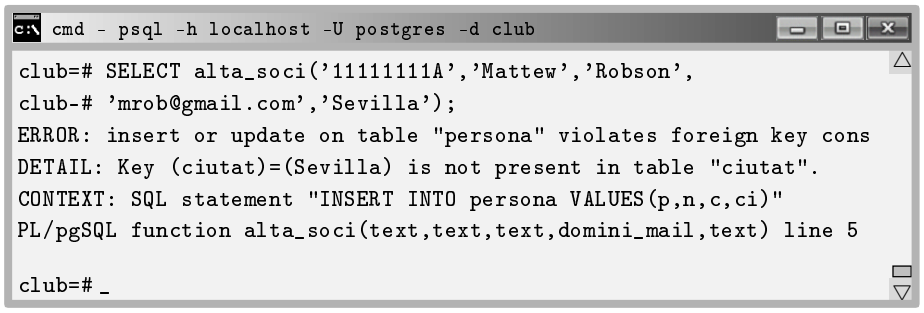
Malgrat tot, hi ha encara dos altres errors d'integritat referencial que efectivament es poden produir. Un per duplicitat de clau primària, i l'altre per inexistència de valor apuntat per la clau forana `ciutat`. Convé conèixer els missatges d'error associats a cada un d'ells. Per això es descriuen tot seguit, amb exemples.

- Que el número de passaport ja existeixi a la base de dades. El passaport `27673812M` és el de la Carme Peralta de Cadaqués. Observeu el missatge de la Pantalla 7.17, que té tres parts indicant l'error, el detall i el context, on també es pot veure l'ornamentació que practica PostgreSQL amb els noms de les funcions.

```
cmd - psql -h localhost -U postgres -d club
club=# \i alta_soci.sql
---------- funcio alta_soci()
club=# CREATE FUNCTION
club=# SELECT alta_soci('27673812M','Mattew','Robson',
club-# 'mrob@gmail.com','Nova Tork');
ERROR: duplicate key value violates unique constraint "persona_pkey"
DETAIL: Key (passaport)=(27673812M) already exists.
CONTEXT: SQL statement "INSERT INTO persona VALUES(p,n,c,ci)"
PL/pgSQL function alta_soci(text,text,text,domini_mail,text) line 5

club=# _
```

Pantalla 7.17. *Error d'inserció clau primària duplicada.*

- Que la ciutat, sent un valor vàlid, no existeixi en la taula de ciutats. Llavors es produirà l'error de la Pantalla 7.18.



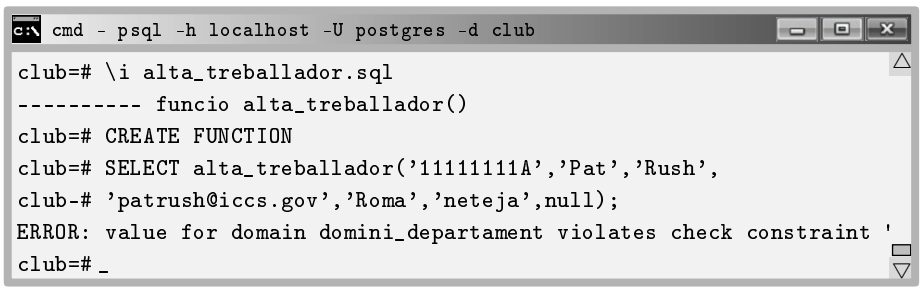
Pantalla 7.18. *Error d'inserció clau forana inexistent.*

A continuació s'implementa la funció per donar d'alta treballadors.

```
\echo ---------- funcio alta_treballador()

CREATE FUNCTION alta_treballador(p TEXT, n TEXT, c TEXT,
         m domini_mail, ci TEXT,
         d domini_departament, o TEXT) RETURNS INTEGER AS $$
BEGIN
    INSERT INTO persona VALUES(p,n,c,m,ci);
    INSERT INTO treballador VALUES(p,d,o);
    RETURN 1;
END;
$$ LANGUAGE 'plpgsql';
```

Caixa 7.16. *Arxiu* `alta_treballador.sql`. *Consultes d'actualització.*

Un nou error es pot ocasionar amb el codi de la Caixa 7.16. En la Pantalla 7.19 es mostra el cas. Es dóna un nom pel departament, *neteja*, que no pertany al `domini_departament`.



Pantalla 7.19. *Error valor incorrecte del domini.*

Aquest parell de funcions, `alta_soci()` i `alta_treballador()` només haurien de ser executades per treballadors administratius. Per això cal afegir un parell de línies al final de l'arxiu `4-usuaris.sql` que estableixin aquests privilegis, com en la Caixa 7.17.

```
...
CREATE ROLE entrenadors;
DROP ROLE IF EXISTS administratius;
CREATE ROLE administratius;

GRANT EXECUTE ON FUNCTION alta_soci(text,text,text,
       domini_mail,text) TO administratius;
GRANT EXECUTE ON alta_treballador(text,text,text,
       domini_mail,text,text,text) TO administratius;
```

Caixa 7.17. *Addició de privilegis sobre les funcions en l'arxiu* `usuaris.sql`.

7.6.5 Consultes de lectura en PL/pgSQL

Les consultes d'actualització es caracteritzen pel fet que la informació transita de l'usuari cap a la base de dades. En canvi, en les consultes de lectura passa el contrari. Això no té més transcendència quan es treballa interactivament, ja que hi ha la pantalla que suporta aquesta transmissió. de manera que si fem la selecció de tot el contingut d'una taula, simplement l'obtenim a través del monitor. Fer una consulta de lectura des d'un programa ha de ser diferent, ja que no tindria cap sentit que el programa volqués la informació pel monitor quan ningú ho estigués mirant. Per tant, calen variables per allotjar les dades que obtenim via consulta de lectura. I això vol dir discernir entre si la consulta retorna un sol valor, o una col·lecció. Per això s'ha dividt aquesta secció en aquestes dues situacions.

En qualsevol dels dos casos, en PL/pgSQL hi ha una variable d'entorn que es diu `FOUND`. És una variable booleana que s'actualitza després de qualsevol `SELECT` i val cert si l'últim `SELECT` que s'ha fet ha donat alguna tupla de resultat.

Les consultes de lectura introdueixen una novetat interessant. Nous tipus per les variables. El sufix `%ROWTYPE` serveix per expressar que una variable és del tipus que sigui una fila d'una taula. Per exemple, `persona%ROWTYPE` significa un tipus de variable que es correspon amb una fila de la taula `persona`. I això vol dir una estructura amb cinc camps, passaport, nom, cognom, mail, i ciutat. Bàsicament, amb variables d'aquests tipus podem fer dues coses. O bé un `SELECT * INTO` la variable (que s'explica tot seguit), o bé un bucle `FOR` la variable `IN (SELECT * FROM persona) LOOP`, que s'explica més endavant.

És important comprendre el que signifiquen els tipus %ROWTYPE perquè s'utilitzaran intensament en els disparadors.

Consultes que retornen un sol valor

En aquest cas podem guardar el resultat del SELECT en una variable local d'un tipus compatible amb el resultat de la consulta posant la paraula clau INTO seguit del nom de la variable després de la paraula SELECT.

En la Caixa 7.18 es mostra una funció que donat un número de passaport retorna el nom i el cognom de la persona corresponent, separats per un blanc.

```
\echo ---------- funcio nom_i_cognom()

CREATE FUNCTION nom_i_cognom(p TEXT) RETURNS TEXT AS $$
DECLARE
    rp persona%ROWTYPE; -- registre amb la persona corresponent.
BEGIN
    SELECT INTO rp * FROM persona WHERE passaport = p;
    RETURN rp.nom || ' ' || rp.cognom;
END;
$$ LANGUAGE 'plpgsql';
```

Caixa 7.18. *Primera versió arxiu* nom_i_cognom.sql. *Selecció d'un sol valor.*

Després d'haver fet una importació des del psql podem comprovar el seu funcionament com es fa en la Pantalla 7.20.

```
cmd - psql -h localhost -U postgres -d club
club=# \i nom_i_cognom.sql
---------- funcio nom_i_cognom()
CREATE FUNCTION
club=# SELECT nom_i_cognom('47548338K');
 nom_i_cognom
---------------
 Anna Sanàbria
(1 row)

club=# _
```

Pantalla 7.20. *Importació i ús de la funció* nom_i_cognom().

Si se li passa un número de passaport no present a la taula `persona`, la funció de la Caixa 7.18 peta intent d'ús de punter nul en la última línia, `NullPointerException`, retornant una excepció. I probablement aquest no hauria de ser el comportament esperat. Una versió alternativa, doncs, es mostra en la Caixa 7.19.

```
\echo ---------- funcio nom_i_cognom()

CREATE FUNCTION nom_i_cognom(p TEXT) RETURNS TEXT AS $$
DECLARE
    rp persona%ROWTYPE; -- registre amb la persona corresponent.
BEGIN
    SELECT INTO rp * FROM persona WHERE passaport = p;
    IF FOUND THEN
          RETURN rp.nom || ' ' || rp.cognom;
    END IF;
    RETURN 'el número de passaport % és inexistent.',p;
END;
$$ LANGUAGE 'plpgsql';
```

Caixa 7.19. *Arxiu* `nom_i_cognom.sql`. *Selecció i verificació d'un sol valor.*

Un cop verificat el seu correcte funcionament, provant-ho per números de passaport existents i inventats, guardem l'escript a la carpeta `club/persona`, afegim en l'arxiu `persona.sql` la línia `\i 'persona\\nom_i_cognom.sql'`, després de la creació de la taula, i tornem a remuntar la base des del no-res important l'arxiu principal, `club.sql`. Observeu que mentre es desenvolupa l'aplicació com ho estem fent potser ens interessa més la versió de la Caixa 7nicdp2, però en canvi a partir del moment en que aquesta base de dades hagi de ser utilitzada per aplicacions GUI pot resultar més convenient llançar una excepció, utilitzant doncs la funció en la versió de la Caixa 7.18.

L'única font real d'errors als que l'aplicació s'exposa és a partir de les dades introduïdes per l'usuari. Així doncs, és normal que no esmercem esforços en controlar tant com es pugui aquesta entrada de dades, i possible font d'inconsistències. Utilitzarem varies funcions addicionals per verificar que no es donin valors amb errors d'accentuació a l'hora de donar ciutats, comarques, i esports. És a dir, en aquelles taules en les que la clau primària sigui textual i l'usuari pugui introduir-les errant en els accents.

En aquest sentit, ens aprovisionem primer d'una funció immutable addicional que ens retorni la forma canònica d'una cadena de caràcters. Direm forma canònica a la mateixa cadena però sense accents ni blancs, i en majúscules. És una definició que establim en aquest moment. Hi ha qui li diria forma normalitzada, o forma estàndar. En qualsevol cas, representa la forma de referència de cada terme.

En la Caixa 7.20 es mostra la funció per canonitzar qualsevol mot. I tot seguit es fa una anàlisi minuciosa.

```
\echo ---------- funcio canonitza()

CREATE FUNCTION canonitza(mot TEXT) RETURNS TEXT AS $$
DECLARE
    i INTEGER;
    n INTEGER;
    c CHAR;
    str TEXT DEFAULT '';
BEGIN
    n = length(mot);
    mot = upper(mot);
    FOR i IN 1..n LOOP -- per cada lletra del mot
        c = substr(mot,i,1);
        IF c <> ' ' THEN
            str = str || set_bits(c);
        END IF;
    END LOOP;
    RETURN str;
END;
$$ LANGUAGE 'plpgsql';
```

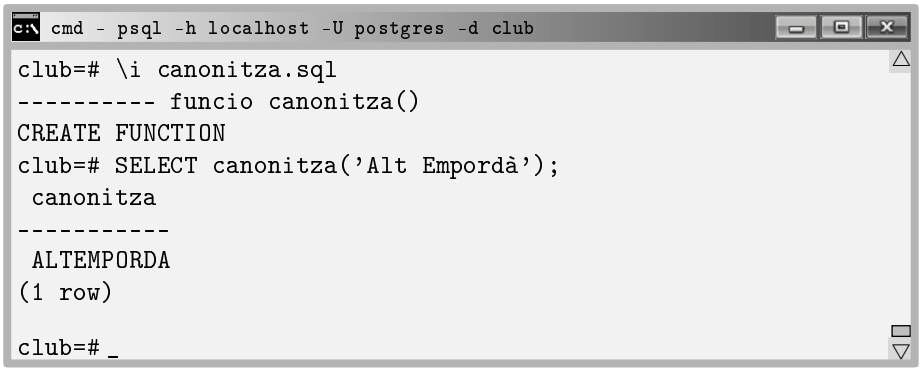
Caixa 7.20. *Arxiu* `canonitza.sql`. *Funció auxiliar.*

Aquesta funció comença guardant-se la longitud de la cadena de caràcters que rep com a paràmetre d'entrada, i seguidament la passa a majúscules amb la funció `upper()`, predefinida en el PL/pgSQL. Després en fa un recorregut eliminant blancs i transformant cada lletra amb la funció `set_bits()` de la Caixa 7.13. La variable `str` és la que finalment es retorna.

Pel correcte funcionament de la funció `canonitza()` és indispensable que la variable `str` s'inicialitzi amb una cadena de caràcters buida. A la Caixa 7.20 s'ha fet amb la clàusula `DEFAULT`. També s'hagués pogut fer en qualsevol lloc previ al bucle amb una assignació, `str = '';`. Aquest valor inicial resulta necessari per la manera de funcionar del procediment.

A cada iteració del bucle concatena una lletra més a la variable `str`. Si no se li hagués donat un valor inicial hagués començat l'execució amb un valor nul, i llavors, concatenar qualsevol cosa a `str` no tindria cap efecte, ja que en tot moment el valor seria nul. En certa manera doncs, aquesta variable fa el paper d'acumulador, i com a tal, cal que s'inicialitzi a blanc, com ho faria a zero si fos numèric.

En la Pantalla 7.21 es mostra el resultat d'aquesta funció per una cadena de caràcters d'exemple, *Alt Empordà*.



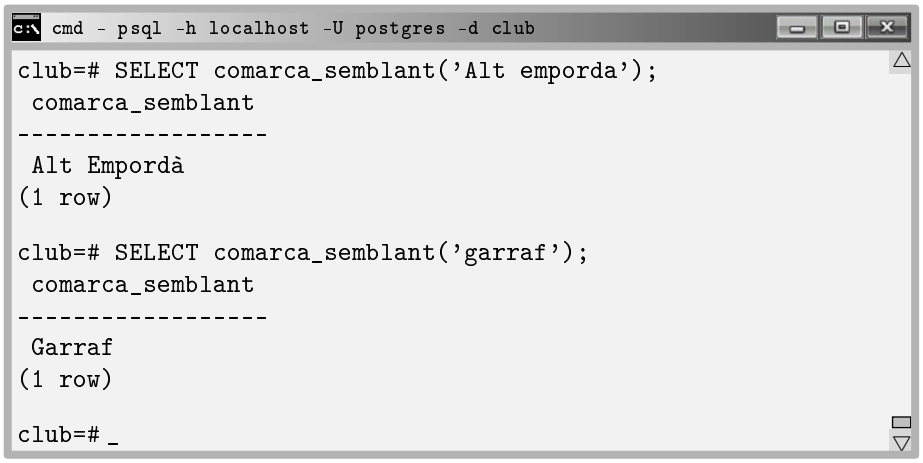
Pantalla 7.21. *Importació i ús de la funció* `canonitza()`.

Anant una mica més enllà, ens aprovisionem a més d'una funció que anomenem `comarca_semblant()`, que ja no és immutable. Les funcions *volàtils* són les que depenen del contignut de la base de dades. Que sigui immutable o volàtil ens afecta per guardar-la. Les immutables a la llibreria, en canvi les volàtils en la carpeta correponent a l'última taula que referencïin. La funció rep un mot, suposadament un nom de comarca, i en cas que a la base de dades hi hagi alguna altra comarca que sigui canònicament equivalent, retorna l'existent. Si no és així, retorna el mateix paràmetre d'entrada.

```
\echo ---------- funcio comarca_semblant()

CREATE FUNCTION comarca_semblant(c TEXT) RETURNS TEXT AS $$
DECLARE
    vc; -- nom de la comarca vella, preexistent.
    cc; -- nom canonic de la comarca nova.
BEGIN
    cc = canonitza(c);
    SELECT comarca INTO vc
    FROM comarca
    WHERE cc = canonitza(comarca);
    IF FOUND THEN
        RETURN vc;
    END IF;
    return c;
END;
$$ LANGUAGE 'plpgsql';
```

Caixa 7.21. *Arxiu* `comarca_semblant.sql`. *Funció auxiliar.*

La funció de la Caixa 7.21 és útil per corregir alguns errors bàsics de lletreig que pot cometre l'usuari en la introducció del nom de les comarques. En la Pantalla 7.22 veiem un parell d'execucions d'aquesta funció.



Pantalla 7.22. *Exemples d'ús de la funció* `comarca_semblant()`.

Aquesta funció es farà servir des del disparador d'inserció a la taula `comarca`. Les funcions corresponents per les ciutats, en l'escript `ciutat_semblant.sql`, i pels esports en l'`esport_semblant.sql` es poden crear anàlogament, i malgrat no es mostren aquí, sí que es troben penjades al repositori que s'indica al final del preàmbul.

Clàusula `PERFORM`

L'ús de la variable `FOUND` és molt útil. Tant, que a voltes més que estar interessats en el que retorna un `SELECT` estem interessats únicament en si retorna alguna cosa. En aquests casos, tant la variable que caldria declarar com la clàusula `INTO` del `SELECT` poden evitar-se utilitzant la instrucció `PERFORM`.

La clàusula `PERFORM` funciona exactament igual que un `SELECT`, amb l'avantatge de no haver de donar cap variable per guardar el resultat de la consulta. Ara bé, la variable `FOUND` sí que quedarà actualitzada amb el valor correcte igual que si el `SELECT` corresponent al `PERFORM` obtingués algun resultat.

Es podria utilitzar la clàusula `PERFORM`, per exemple, pel que fa a les insercions de nòmines. Considerarem que tant el sou base com la retenció dels treballadors, quan no es passi per paràmetre, serà el mateix que el de la nòmina anterior. Pel cas del `sou_base` que és un atribut requerit, si no es dóna i és la primera nòmina del treballador es produirà un error. Per saber si quan es dóna d'alta una nòmina és la primera d'aquell treballador caldria usar la clàusula `PERFORM`.

Aquesta clàusula també es pot utilitzar per invocar una funció que sigui un procediment, és a dir, que faci coses però no retorni cap informació. Per exemple, en el cos d'una funció podria haver-hi la instrucció `PERFORM hola_nom()`.

Tot i així, àixò no és cap novetat, ja que per qualsevol que fos el tipus d'una variable anomenada `var`, per exemple, és ben sabut que podria aparèixer la instrucció `SELECT INTO var hola_nom()`, que tenint en compte que retorna un valor nul, la instrucció seria vàlida.

Consultes que retornen registres

Les consultes que retornen varis registres poden ser fonamentalment de dues formes. Les que contenen la instrucció `RETURN QUERY` i les que es basen en la instrucció `RETURN NEXT`.

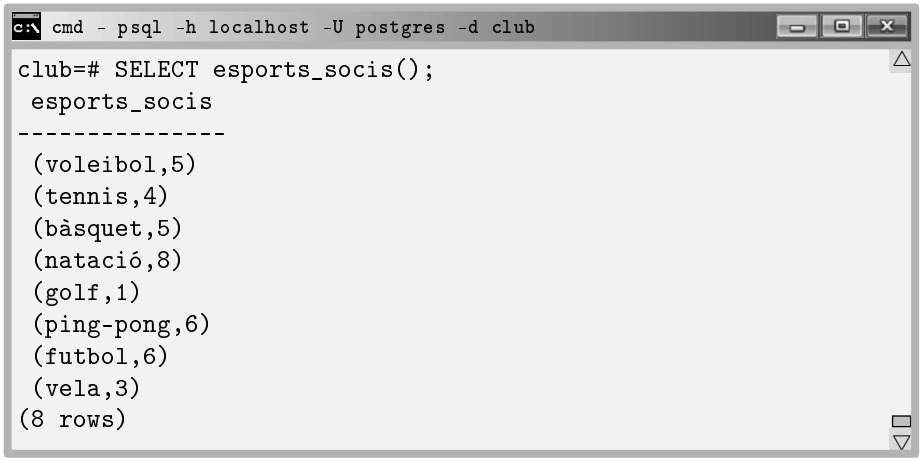
Les funcions amb `RETURN QUERY` són semblants a una vista. En la Caixa 7.22 s'implementa una funció que retorna el nombre de socis inscrits per cada esport. Aquesta manera de resoldre-ho es va definir en l'SQL 92. És clar que el mateix es podria fer amb una vista, que forma part de l'SQL estàndar des de la primera versió.

```
\echo ---------- funcio esports_socis()

CREATE FUNCTION esports_socis()
    RETURNS TABLE(esports TEXT, socis INTEGER) AS $$
BEGIN
    RETURN QUERY(
        SELECT esport,count(*)
        FROM fa
        GROUP BY esport
    );
END;
$$ LANGUAGE 'plpgsql';
```

Caixa 7.22. *Arxiu* `esports_socis.sql`. *Exemple d'ús de la instrucció* `RETURN QUERY`.

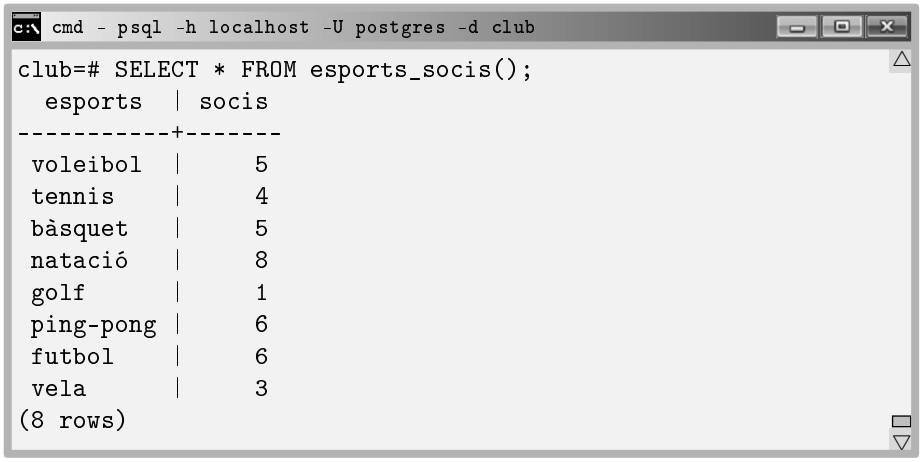
En aquest cas, les dues formes d'invocar la funció resulten diferents.

A la Pantalla 7.23 es mostra la forma d'una única columna. Noteu que cada fila es tanca entre parèntesis, i tota ella forma una sola cadena de caràcters que és el tipus de la columna resultant. El títol d'aquesta columna és el mateix nom de la funció, com abans.



Pantalla 7.23. *Forma poc aconsellable de cridar la funció de la Caixa 7.22.*

La diferència entre la crida il·lustrada en la Pantalla 7.23 i la segona manera de fer-ho, la de la Pantalla 7.24 pot portar greus problemes pels programes clients.



Pantalla 7.24. *Forma correcta de cridar la funció de la Caixa 7.22.*

És notori que en aqesta segona forma d'invocar la funció els títols de les columnes són els que s'ha expressat en la capçalera de la funció. Guardarem l'arxiu esports_socis.sql en la carpeta `club/fa`, lògicament.

L'error que suposa confondre la crida d'una sola columna a la de tantes com la consulta interna doni és freqüent mentre es desenvolupa l'aplicació. I difícil de detectar, perquè quan el programa client pretén accedir a la segona columna és produeix una excepció de punter nul, ja que la consulta tan sols ha retornat una sola columna.

Respecte les consultes de lectura que retornen varis registres però per cada un d'ells cal un tractament específic s'usen intensivament els tipus fila. Com s'ha mostrar més amunt en la funció `nom_i_cognom()` de les Caixes 7.18 i 7.19 va associat a les tuples de les taules de la base de dades. El nom d'aquests tipus es forma amb el nom de la taula en qüestió seguit de la cadena `%ROWTYPE`. Per exemple, el tipus `ciutat%ROWTYPE` significa una estructura de dades de quatre camps amb els noms dels atributs de la taula `ciutat`.

A més, les variables declarades de tipus fila poden utilitzar-se com a índex de bucles que recorrin una consulta. Ens trobem doncs davant una de les capacitats expressives més potents del llenguatge PL/pgSQL.

En la funció de la Caixa 7.23 es mostra la funció `ciutats_esports_socis()`. Aquesta funció obtindrà la quantitat de socis de cada ciutat que practiquen cada esport.

```
\echo ---------- funcio ciutats_esports_socis()

CREATE FUNCTION ciutats_esports_socis()
    RETURNS TABLE(ciutat TEXT, esport TEXT, socis BIGINT) AS $$
DECLARE
    rc ciutat%ROWTYPE; -- registre de la taula ciutat
    re esport%ROWTYPE;
    s BIGINT;
BEGIN
    FOR rc IN (SELECT * FROM ciutat c) LOOP
        FOR re IN (SELECT * FROM esport e) LOOP
            SELECT INTO s COUNT(*)
            FROM fa f
            WHERE f.esport = re.esport
            AND f.passaport IN (SELECT passaport
                 FROM persona
                 WHERE ciutat = rc.ciutat);
            IF s > 0 THEN
                 RETURN NEXT c.ciutat,e.esport,s;
            END IF;
        END LOOP;
    END LOOP;
    RETURN NULL;
END;
$$ LANGUAGE 'plpgsql';
```

Caixa 7.23. *Arxiu* `ciutats_esports_socis.sql`. *Recorreguts de consultes.*

Aquest arxiu s'hauria de guardar en la carpeta de la taula `esport`. Si d'una ciutat no hi ha ningú que practiqui un esport concret, llavors no apareixerà la

fila corresponent en la relació resultant. Això és un tractament especial per cada element del producte cartesià entre esports i ciutats. De fet, observeu que els dos bucles de la funció de la Caixa 7.23 no fan més que recórrer aquest producte cartesià, i tractar el cas de cada parella possible una per una amb un `IF`.

Observeu el tipus `BIGINT` que té la columna `socis` de la taula resultant. Això és per allotjar el resultat de la funció d'agregació `COUNT(*)`. En anglès, *big* vol dir gran però no de magnitud, sinó de cosa. Cosa gran. Pel números, per dir que un número és gran s'usa l'adjectiu *large*. Aquest matís en aquest parell d'adjectius és un punt a favor de la llengua anglesa, ja que sens dubte és més correcta que les llatines pel que fa aquest extrem. Un número és més aviat llarg que gran, ja que resideix en una sola dimensió, la que quantifica. Però bé, tampoc no va més enllà. En l'ús del tipus `BIGINT`, contracció de *big integer*, en la funció de la Caixa 7.23 no vol dir doncs que els números de socis hagin de ser números grans, sinó que l'espai de memòria que facilitem per allotjar-los sí que efectivament serà més gran que per un número enter qualsevol. Tot plegat no té més transcendència.

7.7 Disparadors

Els disparadors, *triggers* en anglès, són funcions que no s'executen quan l'usuari o programa client ho demana, sinó que van associats a altes, baixes, o modificacions de les taules de la base de dades. Els paràmetres que cal establir per crear un disparador són cinc.

1. La funció. Cal donar, és clar, la funció que s'executarà. Ha de ser una funció de tipus `TRIGGER`. Una funció és del tipus del valor que retorna. Si una funció retorna un enter, és una funció entera.
2. Tot i així, el primer paràmetre que caracteritza un disparador és la taula en la que està declarat. Hi ha una dependència d'existència del disparador vers la taula. Això vol dir que com a conseqüència d'una comanda `DROP TABLE taula CASCADE` també s'eliminen els disparadors que hi hagi declarats, que no vol dir les funcions. La funció d'un trigger i el trigger mateix són objectes diferents en les metadades, amb noms diferents si es vol.
3. L'event. Cal establir si el disparador s'executarà quan hi hagi insercions, modificacions, o eliminacions. O qualsevol combinació d'aquestes comandes.
4. Abans o després que es produeixi l'event.
5. El tipus de disparador, que pot ser `STATEMENT`, o bé `ROW`. Això és, el disparador s'executarà un cop per cada transacció, o un cop per cada event. Mirant els escripts d'inserció del capítol anterior es veu que amb una sola comanda d'inserció es pot afegir múltiples registres. I per tant cal decidir si es vol executar el disparador un cop per cada comanda, o un cop per cada registre inserit.

7.7.1 Funcions de Tipus TRIGGER

Les funcions de tipus TRIGGER s'executen asíncronament. Per tant, ningú sap quan s'executaran. Ningú les crida. I per tant, no poden tenir paràmetres d'entrada. Per la mateixa raó, tampoc poden retornar cap valor, tot i que aquest extrem es matitzarà en breu.

Tenim dos conjunts. Un format pels events {INSERT,UPDATE,DELETE} i l'altre format pels modificadors abans i després, {BEFORE,AFTER}. Doncs bé, per cada possible parella d'elements formada per un de cada conjunt hi ha la possibilitat d'associar-hi un disparador.

Amb l'experiència, qui desenvolupa bases de dades aprèn que els disparadors associats a abans dels events acostumen a practicar feines de verificació dels valors que es pretenen inserir o modificar. Els associats a després dels events, fan tasques de publicació o notificació del fet.

Capçalera de les funcions TRIGGER

Les capçalera de les funcions d'aquest tipus tenen una estructura molt rígida. Sempre igual. A la Caixa 7.24 es mostra l'única estructura possible per totes les funcions de disparadors.

```
CREATE FUNCTION nom_funcio() RETURNS TRIGGER AS $$
```

Caixa 7.24. *Capçalera d'una funció de tipus* TRIGGER.

L'única paraula que pot variar en la capçalera de la Caixa 7.24 és el nom de la funció, que aquí s'ha anomenat `nom_funcio`. Aquestes funcions es poden codificar després de la creació de la taula a la que van associades, en el mateix arxiu. I després de la seva implementació, llavors es podrà declarar el disparador.

Cos de les funcions TRIGGER

Les funcions del PL/pgSQL, tan sols pel fet de declarar que retornen un tipus TRIGGER, tenen disponibles dues variables. Bé, depèn de l'event al que el disparador s'associï finalment. Comprendre el sentit d'aquestes variables és fonamental, ja que en última instància són les úniques candidates com a possibles valors de retorn per aquest tipus de funcions.

- Disparadors associats a insercions.
  Sempre que una funció es declara com a disparador activat per insercions en una taula `T` disposa dins el seu àmbit de visibilitat una variable amb el nom clau `NEW`. Aquesta variable és de tipus fila de la taula, `T%ROWTYPE`. El seu contingut són els valors del nou registre que s'està pretenent inserir. La funció pot modificar aquests valors abans d'inserir el now registre.

- Disparadors associats a modificacions.
  Pel cas dels disparadors associats a l'event de modificació es disposa de dues variables del mateix tipus fila de la taula on van lligats. Ambdós, amb paraules clau `NEW` i `OLD`. La variable `NEW` funciona igual que pel cas de la inserció. La variable `OLD` conté els valors que estan a punt de ser sobreescrits en la modificació.

- Disparadors associats a eliminacions.
  Sempre que associem un disparador a l'eliminació en una taula disposem en el codi de la variable `OLD`. Conté els valors del registre que està a punt de ser eliminat.

Aquestes variables només estan disponibles pels row triggers, és a dir, pels que s'executen un cop per cada registre afectat en l'operació.

Valors de retorn dels disparadors

Pels disparadors associats abans de l'event hi ha dues opcions. Si retornen `NULL` anul·len l'event. Per exemple, pel cas d'una inserció. Un disparador associat abans d'una inserció en una taula pot evitar que les insercions es produeixin retornant un valor nul, o pot indicar que la inserció procedeixi retornant la variable de tipus fila amb el nom clau `NEW`. Això mateix succeeix pel cas de les modificacions. En canvi, si el disparador s'associa abans d'una eliminació, per evitar que s'esborri el registre també pot retorna un valor nul, però per procedir amb la supressió hauria de retornar la variable de tipus fila `OLD`.

Tots els disparadors associats a després dels events han de retornar `NULL` forçosament.

7.7.2 Sintaxis de Declaració d'un Disparador

En aquesta secció es proposa el desenvolupament d'un disparador que s'executarà abans de fer una inserció en la taula `comarca`. La seva funció és assegurar-se que no s'hagi produït errors de lletreig en la introducció del nom. La forma de procedir és senzilla. Abans de fer la inserció es mira si el nom que es pretén inserir és equivalent canònicament algun que ja estigui dins la taula. Això es fa prenent el valor resultant de la funció `comarca_semblant()` de la Caixa 7.22. Si ho és,

avorta la inserció retornant un valor nul. I si no, s'indica que es prosegueixi retornant el valor NEW.

```
\echo ---------- funcio verifica_comarca()

CREATE FUNCTION verifica_comarca() RETURNS TRIGGER AS $$
DECLARE
    nc TEXT; -- nom correcte de la comarca
BEGIN
    nc = comarca_semblant(NEW.comarca);
    PERFORM * FROM comarca WHERE comarca = nc;
    IF FOUND THEN
        RETURN NULL;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE 'plpgsql';
```

Caixa 7.25. *Disparador abans de la inserció, per cada fila, en la taula* `comarca`.

En la Caixa 7.26 es mostra la declaració del disparador que servirà per corregir errors de lletreig en la introducció de noves comarques.

```
CREATE TRIGGER BEFORE INSERT ON comarca
FOR EACH ROW EXECUTE PROCEDURE verifica_comarca();
```

Caixa 7.26. *Declaració d'un disparador per la taula* `comarca`.

El contingut de les Caixes 7.25 i 7.26 es poden posar a continuació de la mateixa creació de la taula `comarca` ja que són conceptes totalment lligats.

Observeu que malgrat tot, en cap moment hem establert un format canònic d'interfície, que anomenem formats normalitzats. És a dir, no estaria de més implementar una funció anomenada `comarca_normalitzada()` en la que s'establissin les regles que es desitgessin a l'hora de mostrar els noms de les comarques a les aplicacions client. Per exemple, es podria regular que els noms de les comarques comencen sempre amb la primera lletra majúscula, i també en majúscula qualsevol paraula de més de tres lletres que hi hagi en el nom de la comarca, és a dir qualsevol lletra que segueixi un blanc i vingui succeïda de més de dues lletres. Això per filtrar els relatius, "de" o "d'" i els articles "la" o "el". Aquest seria un format normal.

7.8 Aplicacions Client

Es important adonar-se'n que estrictament, el contingut d'aquesta secció no és propi d'un llibre de bases de dades. Si es vol estudiar un robot cal fer-ho comprenent com realitza les accions que se li demanen. Això acaba resultant en una col·lecció de comandes que activen el robot per fer alguna cosa. Si després, a més a més, volem manipular el robot des d'un programa informàtic realitzat en un llenguatge de programació informàtica, cal que qui ha construït el robot ens proporcioni un interfície de comandes per poder-les executar des del programa en java, per exemple. Doncs això és el mateix per les bases de dades. El tema de les aplicacions client per les bases de dades és exactament igual que el tema de les aplicacions client per qualsevol altre tipus de sistema que es pugui manipular per mitjà d'un programa en llenguatge estàndar. És a dir, tot plegat consisteix en la llibreria. Quan els sistemes que s'ofereixen per ser manipulats informàticament contenen mecanismes físics i dispositius materials, llavora aquestes llibreries reben el nom de *drivers*.

La llibreria per poder treballar des d'un programa amb una base de dades PostgreSQL la proporciona el mateix PostgreSQL, clar. Com sempre. La llibreria per poder manipular un robot també la proporciona el fabricant del robot. De manera que la única cosa que ha de fer el programa és incloure aquesta llibreria en la compilació. Llavors té disposibles les funcions a les que podrà cridar des del codi per executar les accions que vulgui.

El nom de la llibreria per java té la forma `postgresql-9.2-1002-jdbc4.jar`. Aquest arxiu s'ha d'importar al projecte java per poder utilitzar les classes que es veuen tot seguit.

Qualsevol connexió a una base de dades requereix un nom d'usuari i contrasenya. Fins i tot les aplicacions client `GUI`. Per a què un programa client pugui connectar-se a una base de dades ha de donar el nom de l'usuari i la contrasenya com a paràmetres d'alguna funció que consti en la llibreria.

7.8.1 Llenguatge Amfitrió i SQL Incrustat

S'entén per llenguatge amfitrió aquell llenguatge de programació d'alt nivell que permet la incorporació de comandes SQL. Els dos més utilitzats són el java i el php. Per poder oferir aquesta prestació normalment caldrà fer ús de llibreries que proporcionen les classes necessàries per fer, essencialment, tres tipus d'interacció amb les bases de dades.

Connexió

Per poder-se connectar des d'un programa client la llibreria del programa ha de contenir alguna funció a la que se li passin els paràmetres de connexió. Normalment l'adreça internet de l'ordinador servidor, un codi d'usuari, i la contrasenya. Així mateix, també s'acostuma a oferir una funció de desconnexió. En java, la classe que suporta aquests mètodes s'anomena `Connection`.

Respecte aquest extrem existeix un dilema freqüent. Molts programadors es pregunten si convé obrir una connexió cada cop que es fa una consulta, o bé fer-ho un sol cop quan arrenca el programa client i mantenir-la oberta durant tota l'execució. Inqüestionablement la resposta és la primera opció. Cal tenir en compte que qualsevol procés informàtic amb intervenció humana de manera regular molt difícilment podrà flaquejar en qüestions d'eficiència. En altres paraules, els usuaris finals difícilment notaran el temps addicional que suposa obrir i tancar connexions en cada consulta. I a més, tenint en compte que com administradors ens interessa poder controlar la base de dades, si utilitzéssim la segona opció mai podríen fer el manteniment en exclusió mútua respecte els usuaris finals.

Consultes de lectura

En les llibreries dels SGBDs per als llenguatges de programació que implementen aplicacions clients s'ofereixen classes les instàncies de les quals són objectes especialment dedicats a les consultes de lectura. Se'n diuen objectes de tipus `RecordSet`, i formen part de les llibreries ODBC, d'*Object Data Base Connection*, que en java se li diu JDBC. Normalment s'acostumen a utilitzar dos mètodes d'aquesta classe d'objectes. Un que retorna un booleà, `rs.next()`, i un altre per obtenir el valor d'una columna del registre actual, `rs.getColumn(int c)`.

Consultes d'actualització

Per les consultes d'actualització hi ha el mètode `executeQuery(String str)` de la classe `Statement`. Aquest mètode rep una cadena de caràcters que conté se sentència SQL que es pretén executar. I retorna un nombre enter que es correspon amb el mateix que retorna PostgreSQL quan la connexió és via `psql`. És a dir, el nombre de files afectades per la consulta d'actualització.

7.9 Administració de la Base de Dades

En aquesta secció es pretén conèixer les comandes bàsiques que cal utilitzar per poder treballar amb PostgreSQL tant des del sistema operatiu MS-DOS com dels sistemes línux. Això inclou la consulta de si el servei està actiu, la parada, i la iniciació del sistema. Després es veuen les comandes de backup, i de manteniment del clúster, i finalment es parla d'alguns paràmetres de configuració, amb la qual cosa es pretén familiaritzar-se amb aquestes qüestions sense aprofundir-hi massa.

7.9.1 Parada i Iniciació del Sistema

Veiem els procediments en els diferents sistemes operatius per portar terme les operacions més importants que es poden fer amb un SGBD, parar-lo i iniciar-lo. Com que estem parlant d'un servei, lo normal és que estigui actiu, ja que s'engega automàticament quan s'arrenca l'ordinador. Per això, l'ordre en que es presenten les operacions parteix d'aquesta base.

MS-DOS

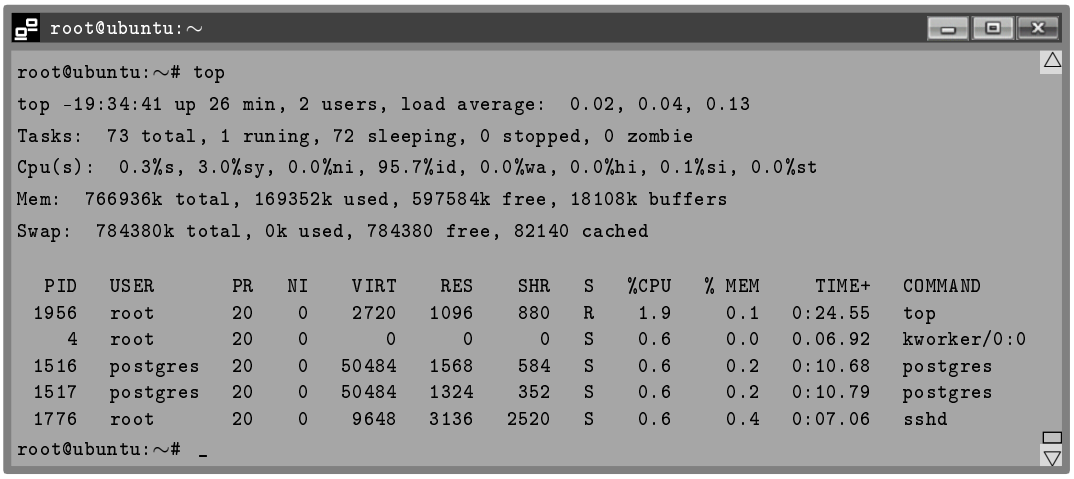
Primer de tot obrim una consola amb permisos d'administració. Una manera de fer això és des del menú *inici* escrivint `cmd` en la caixa *executar:* i prement les tres tecles, control i majúscules, quan es fa intro. Si es tenen permisos d'administració, el sistema demana confirmació ombrejant la pantalla. Acceptem, i som dins.

D'entrada, podem consultar si el PostgreSQL està iniciat amb la comanda `net start`. Això provocarà una hemorràgia de serveis pel monitor. Amb la barra lliscant podem buscar `postgresql-9.3 - PostgreSQL Server 9.3`. Parem PostgreSQL amb la comanda `net stop postgresql-9.3`, i el tornem arrencar amb `net start postgresql-9.3`. Tot plegat es descriu en la Pantalla 7.25. Per això, s'ha abreviat la llista de serveis.

Pantalla 7.25. *Verificació de servidor iniciat, parada i inici.*

Línux

Des d'una terminal línux hi ha una manera ràpida, encara que poc acurada, de saber si PostgreSQL està iniciat en el servidor. La comanda `top`. En la Pantalla 7.26 es mostra una execució.



Pantalla 7.26. *Verificació de servidor iniciat amb la utilitat* `top`.

Aquesta comanda ens proporciona el mateix tipus d'informació que l'administrador de tasques en windows. Això és, per cada procés podem veure l'ús del processador en tant per cent del temps, quantitat de memòria que utilitza el procés, quantitat de memòria de paginació, i altres indicadors que queden fora de l'abast d'explicacions que es donen en aquest llibre.

Si es vol tenir una idea més clara dels quatre processos que formen l'SGBD es pot mirar amb la comanda *process status*, `ps auxww`. I per evitar un allau d'informació, es pot usar la comanda `grep` que busca les cadenes que se li passin utilitzant expressions regulars. En concret, per expressar la condició de començament de paraula, les expressions regulars utilitzen l'accent circumflex.

Entre el `ps` i el `grep` cal fer un *pipe*. Pipejar la sortida d'un procés com entrada d'un altre significa que el segon ha de treballar amb la sortida del primer com arxiu d'entrada. En línux això es comanda amb una barra vertical entre les crides als processos.

Per tant, amb la comanda `ps auxww | grep ^ postgres` connectem la sortida del `ps auxww` com entrada al programa `grep` al qual li demanem que tan sols ens mostri aquelles línies on hi aparegui la paraula postgres com a principi de paraula, és a dir, després d'un separador.

Per parar i arrencar el servidor PostgreSQL des de línux tenim les comandes `service postgresql stop`, i `service postgresql start`. En la Pantalla 7.27 es mostra tot plegat.

```
root@ubuntu:~
root@ubuntu:~# ps auxww | grep ^ postgres
postgres    2667    1.2    1.0    50504    8048 ?      S    00:36  0:03  /usr/lib/postgres
ql/9.1/bin/postgres -D /var/lib/postgresql/9.1/main/postgresql.conf
postgres    2670    0.8    0.2    50488    1568 ?      Ss   00:36  0:02  postgres:  writer
process
postgres    2672    0.1    0.3    50912    2444 ?      Ss   00:36  0.00  postgres:  autovac
uum launcher process
postgres    2673    0.1    0.1    20736    1388 ?      Ss   00:36  0.00  postgres:  stats c
ollector launcher process
root@ubuntu:~# service postgresql stop
    Stopping PostgreSQL 9.1 database server                                    [OK]
root@ubuntu:~# service postgresql start
    Starting PostgreSQL 9.1 database server                                    [OK]
root@ubuntu:~# _
```

Pantalla 7.27. *Verifcació de servidor iniciat amb* `ps`*, parada i inici.*

Amb aquesta segona opció veiem el procés principal, *writer process*, el procés dedicat a la replicació que escriu els arxius *write ahead log*, el disparador que arrenca el regenerador, *autovacuum*, i el recollector d'estadístiques.

El programa `pg_ctl`

Internament, les formes utilitzades en les Pantalles 7.25 i 7.27 per iniciar o parar el servidor executen el programa `pg_ctl` amb els paràmetres predefinits per les crides automàtiques. Si es vol variar alguna d'aquestes opcions, llavors cal invocar al programa directament. Per exemple, a l'hora de parar-lo, hi ha l'opció de fer-ho amb l'opció *smart*, que vol dir que s'esperi a que els programes clients acabin, fer-ho amb l'opció *fast*, que espera que s'acabin les transaccions que puguin estar realitzant-se, i l'opció *inmediate*, que para abruptament el servidor, i requereix fer una restauració quan es torni a engegar.

El programa `pg_ctl`, a més d'*start* i *stop*, també té les opcions de *restart*, i *reload*. Aquesta última tan sols torna a carregar l'arxiu de configuració.

Aquest programa requereix d'un paràmetre el contingut del qual ha de ser el nom de la carpeta on es troben els arxius de configuració, `-D`. O sigui que des del sistema operatiu s'invoca amb la comanda

```
pg_ctl -D "C:\Program Files (x86)\PostgreSQL\9.3\data"
```

pel cas d'MS-DOS. Si no es passa aquest paràmetre en la crida al programa, PostgreSQL també pot obtenir aquesta informació a partir de l'una variable de sistema, com es veurà més endavant.

7.9.2 Manteniment i Còpies de Seguretat

La manera més comprensible de fer una còpia de seguretat és amb el programa client `pg_dump`. Se li passen els paràmetres de connexió de la mateixa manera que al `psql`, i s'hi afegeix un redireccionament de la sortida a un fitxer. En acabar l'execució tenim un arxiu de text amb la seqüència de comandes SQL que restauren la base de dades al mateix punt en el que estava a l'executar la comanda `pg_dump`. Fem-ho amb un experiment de sis passes.

1. Des d'una consola amb permisos d'administració, situats en la carpeta d'on penja la del projecte club, fem un backup de la base de dades amb el programa `pg_dump`, redireccionant la sortida al fitxer `club.bck`. La comanda que donem al sistema operatiu és

   ```
   pg_dump -h localhost -U postgres -d club > club.bck
   ```

2. Comprovem que existeix el nou arxiu `club.bck` en el directori actual amb la comanda `dir` d'MS-DOS. També resulta interessant obrir-lo amb qualsevol editor de text pla i fer una ullada al seu contingut, que en gran part consisteix en el mateix escript de creació que havíem fet en capítols anteriors.

3. Ens connectem a PostgreSQL com sempre, amb `psql -h localhost -U postgres -d postgres`.

4. Eliminem la base de dades club amb la comanda de sempre, `DROP DATABASE club` sense por, confiant en el backup. Clar, si tenim els escripts de creació i d'inserció això no té tanta gràcia.

5. Creem la base de dades club, i ens hi connectem de la manera que ho fa l'escript principal, `CREATE DATABASE club`, i `\c club`.

6. Importem l'arxiu `club.bck` també igual que amb qualsevol altre arxiu.

Finalment, estem en el mateix estat que al començament. L'incidència de cada un dels passos es pot comprovar afegint comandes a l'experiment. Però vaja, ja es pot intuir que a partir d'aquí. podem implementar qualsevol política de còpies de seguretat.

En la Pantalla 7.28 es reprodueix el diàleg d'aquestes sis passes.

```
Administrador:  C:\Windows\system32\cmd.exe
C:\>
C:\>pg_dump -h localhost -U postgres -d club > club.bck
Password:

C:\> dir club.bck
 El volum de la unitat C és sistema
 El número de sèrie del volum es:  C83F-2014

 Directori de C:\

27/09/2014  00:24            21.886 club.bck
               1 arxius         21.886 bytes
               0 dirs  31.577.726.976 bytes lliures

C:\> psql -h localhost -U postgres -d postgres
Password for user postgres:
psql (9.3.4)
Type "help" for help.

postgres=# DROP DATABASE club;
DROP DATABASE
postgres=# CREATE DATABASE club;
CREATE DATABASE
postgres=# \c club
You are now connected to database "club" as user "postgres".
club=# \i club.bck
SET
SET
```

Pantalla 7.28. *Còpia de seguretat a l'arxiu* `C:\club.bck`.

L'exemple de la Pantalla 7.28 és la mínima expressió per poder fer un backup de la base de dades. Amb les polítiques usades per les còpies de seguretat hi ha un ventall de possibilitats amb diferents nivells de protecció. Es distingeix entre backups totals, i incrementals. És a dir els que fan una còpia de la base des de zero, i els que el fan a partir de l'últim backup realitzat. Pels tipus incrementals de backup, la restauració pot ser complicada, ja que s'ha de reseguir la seqüència feta des de l'últim backup total.

Però per davant de tot, la metodologia emprada comença a ser robusta a partir del moment que es parla de backups cíclics. Normalment a cada instal·lació es determina la màxima antiguitat amb la que es volen guardar les dades, per exemple cinc anys, i llavors s'estableix un protocol pels noms dels arxius bck que conté la data en que es fa el backup, com per exemple `club-01-01-2015.bck`. El fet de fer-ho cíclicament és per assegurar-se que la quantitat de memòria secundària que es vol dedicar aquesta finalitat es mantingui estable, ja que pel cas de l'exemple, en el moment d'escriure l'arxiu `club-01-01-2020.bck` s'esborraria automàticament el `club-01-01-2015.bck`.

Hi ha altres maneres més sofisticades per gestionar còpies de seguretat de la base de dades. Probablement la més innovadora consisteixi en la replicació. La filosofia és que un servidor faci el paper de gestor del clúster, i altres romanen copiant l'arxiu de log que el servidor principal va emetent.

L'atribut `REPLICATION` que té l'usuari postgres en la Pantalla 7.2, a principis d'aquest capítol, a la pàgina 259, està relacionat amb la replicació de bases de dades. Es tracta d'un sistema de còpies múltiples que a partir d'un servidor principal es copien en un nombre qualsevol de servidors *estants*. Això és fa amb fitxers d'extensió `WAL`, *de write ahead log*. I un usuari amb permís de replicació pot accedir aquests arxius, que normalment es guarden en dispositius de memòria secundària dels servidors estants, o sigui, els secundaris. La diferència entre els dos tipus de servidor és que l'únic amb capacitat per modificar el contingut de les dades és el servidor principal. Els altres poden fer consultes de lectura en les còpies que han aconseguit per mitjà d'aquests fitxers. No cal dir que això requereix mecanismes de sincronització.

7.9.3 Configuració d'un servidor PostgreSQL

L'arxiu `postgres.conf`, així com els altres dos arxius de configuració d'un servidor PostgreSQL, són arxius de text pla. Així, poden editar-se amb els editors més senzills de cada sistema operatiu, notepad per windows o le, o vi, pels sistemes linux. Són tres arxius. El principal, el d'autentificació basada en el host, i el de mapeig d'usuaris. En tots ells s'utilitza el coixinet per introduir comentaris de línia.

Arxiu de paràmetres de configuració

El nom per defecte de l'arxiu principal de la configuració de l'SGBD PostgreSQL és `postgresql.conf`. I el directori per defecte, clar, depèn del sistema operatiu. En qualsevol cas, estant connectats amb el `psql` amb permisos de superusuari, sempre es pot consultar quin és l'arxiu de configuració que s'està utilitzant amb la comanda `show config_file`. En una instal·lació per defecte, la resposta que s'obté es mostra a les Pantalles 7.29 i 7.30.

Pantalla 7.29. *Identificació de l'arxiu de configuració des d'MS-DOS.*

Pantalla 7.30. *Identificació de l'arxiu de configuració des de sistemes línux.*

Observeu pel prompt que cal ser superusuari.

Aquest arxiu està format de paràmetres de configuració. Cada línia té una estructura `nom = valor`, sent el nom el paràmetre en qüestió que pot ser de tipus booleà, enter, real, cadenes o enumeracions. Inicialment, els paràmetres estan comentats i a més, contenen els valors per defecte. Això obliga a pensar-s'ho dues vegades abans de canviar qualsevol valor. Quan es descomenta, i quan es dóna el valor. En tots els paràmetres que necessiten reiniciar el servidor per tenir efecte, en l'arxiu `postresql.conf` ho indica com a comentari.

Cada cop que ens connectem a una base de dades, ja sigui des del sistema operatiu amb el programa `psql`, o des de dins mateix fent `\c`, es llegeixen tots els paràmetres del fitxer, i un cop llegits queden a la taula `pg_settings` de l'SGBD. En aquesta taula, les dues primeres columnes són el nom del paràmetre, `name`, i el valor actual, `setting`.

També podem utilitzar la funció `pg_reload_conf()` a fi de llegir el fitxer de configuració de forma asíncrona, fent `SELECT pg_reload_conf()`.

L'arxiu `postgresql.conf` està segmentat en seccions. La primera es diu lloc pels fitxers, i el primer de tots els paràmetres és precisament el nom del directori on es troba l'arxiu de configuració, `data_directory`. Sembla contradictori que dins l'arxiu de configuració hi hagi un paràmetre, el primer de tots, que diu una informació que cal saber prèviament per poder accedir a ella. I de contradictori no en té res. Aquesta informació pot ser utilitzada per les funcions que hi hagi en l'SGBD, o també per les aplicacions client que es puguin connectar aquest servidor. La funció d'aquest paràmetre és doncs, de mirall. I per tant, per iniciar el PostgreSQL cal donar el valor d'aquest paràmetre d'alguna altra manera.

Una possibilitat és a la línia de comandes amb l'opció `-D` quan s'inicia amb el programa `pg_ctl`, tal com s'ha vist en la Secció 7.9.1. I l'altra possible manera de fer-ho és declarant una variable en el sistema operatiu amb el nom clau `PGDATA`.

Si es canvia el valor d'aquest paràmetre en l'arxiu de configuració cal reiniciar el servidor perquè tingui efecte. No n'hi ha prou amb una relectura de l'arxiu.

A més, dins aquesta mateixa secció de l'arxiu `postgresql.conf`, de llocs pels fitxers, també hi ha la ruta dels altres dos arxius de configuració.

En la següent secció de l'arxiu de configuració, connexions i autenticació, hi ha el paràmetre `listen_addresses` que serveix per restringir les adreces d'internet dels ordinadors client pels quals s'accepten connexions. És habitual assignar-li el valor '*' indicant que s'accepten connexions des de tot arreu. En aquesta secció també hi ha els paràmetres de connexió que vam acceptar per defecte en el moment de la instal·lació. O sigui, el port, i el màxim nombre de connexions.

No és el propòsit d'aquesta secció explicar el significat de cada paràmetre. A més, molts d'ells resultarien difícils de comprendre per qui no tingui un coneixement més o menys profund d'administració de xarxes. Això no obstant, sí que resulta interessant comprendre quines prioritats hi ha quan es modifiquen els valors dels paràmetres. Per això, prenem d'exemple el paràmetre `client_min_messages`. En la Caixa 7.27 es mostra la consulta que ens permet saber qui ha establert el valor d'un paràmetre.

```
SELECT name,setting,source
FROM pg_settings
WHERE name = 'client_min_messages';
```

Caixa 7.27. *Consulta del valor i l'origen d'un paràmetre.*

Amb aquesta consulta podem averiguar el valor que té un paràmetre i qui ha posat aquest valor. El resultat obtingut es descriu en la Pantalla 7.31.

```
psql -h localhost -U postgres -d postgres
postgres=# SELECT name,setting,source
postgres-# FROM pg_settings
postgres-# WHERE name = 'client_min_messages';

        name          | setting |       source
----------------------+---------+--------------------
 client_min_messages | notice  | configuration file
(1 row)
```

Pantalla 7.31. *Valor i origen del paràmetre* `client_min_messages`.

El valor que preval pels paràmetres de configuració ve determinat per l'origen des d'on s'hagi assignat, cosa que es pot veure en la columna `source` de la taula `pg_settings`.

Fem un experiment de set passes amb el mateix paràmetre `client_min_messages`.

1. Modifiquem amb algun editor de text l'arxiu de configuració. Establim el valor del paràmetre igual a `error`. Per això, caldrà descomentar la línia corresponent, i modificar el valor que hi ha darrera el signe d'igual. La situació just després d'haver fet el canvi es mostra en la Pantalla 7.32.

```
postgresql.conf:  Bloc de notes
# This is only relevant when logging to eventlog (win32):
#event_source = 'PostgreSQL'

# - When to Log -

client_min_messages = error # values in order of decreasing detail:
                            #   debug5
                            #   debug4
                            #   debug3
                            #   debug2
                            #   debug1
                            #   log
                            #   notice
                            #   warning
                            #   error
#log_min_messages = warning # values in order of decreasing detail:
```

Pantalla 7.32. *Modificació d'un valor en l'arxiu* `postgresql.conf`.

2. Rellegim l'arxiu de configuració amb la funció `pg_reload_conf()`.

3. Tornem a mirar el valor que té el paràmetre amb la comanda de la Caixa 7.27.

4. Establim un nou valor pel paràmetre des de la mateixa línia de comandes del `psql`, teclejant `SET client_min_messages = debug;`

5. Tornem a mirar el valor que té el paràmetre prement dos cops la fletxa amunt. Observem el canvi en la columna `source`.

6. Reconnectem amb la base a la que estiguem connectats, fent `\c`.

7. Tornem a mirar el valor que té el paràmetre prement altre cop dos cops la fletxa amunt. Observem el canvi en la columna `source`.

En la Pantalla 7.33 es mostren les darreres passes de l'experiment.

```
psql -h localhost -U postgres -d postgres
postgres=# SELECT pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)

postgres=# SELECT name,setting,source FROM pg_settings WHERE name = 'client_min_messages';
        name         | setting |       source
---------------------+---------+--------------------
 client_min_messages | error   | configuration file
(1 row)

postgres=# SET client_min_messages = debug;
SET
postgres=# SELECT name,setting,source FROM pg_settings WHERE name = 'client_min_messages';
        name         | setting | source
---------------------+---------+---------
 client_min_messages | debug   | session
(1 row)

postgres=# \c
You are now connected to database "postgres" as user "postgres".
postgres=# SELECT name,setting,source FROM pg_settings WHERE name = 'client_min_messages';
        name         | setting |       source
---------------------+---------+--------------------
 client_min_messages | error   | configuration file
(1 row)
postgres=# _
```

Pantalla 7.33. *Origens i valors del paràmetre* `client_min_messages`.

Un darrer paràmetre que pot portar problemes de sincronització entre el servidor i els clients, i per això cal tenir-lo en compte és `log_timezone` que aquí acostuma a valdre *'Europe/Brussels'*. S'utilitza a l'hora de traduir entre els tipus temporals de l'SQL.

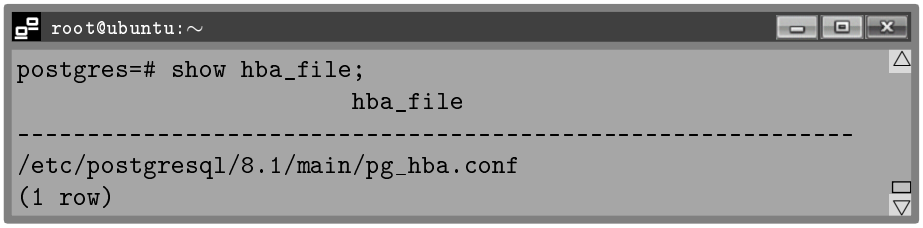
Fent una ullada una mica més acurada a l'arxiu de configuració és notable que la major part dels paràmetres estan comentats. Els altres, és a dir els actius, es van establir en el moment de la instal·lació del PostgreSQL.

Arxiu d'autenticació basada en el host

L'arxiu `pg_hba.conf` és el de la configuració de l'autenticació basada en el host, *host based autentication*. Estableix una associació entre el format físic de l'adreça internet que es connecta i el tipus de contrasenya que se li sol·licitarà. Aquest tipus d'arxiu té un molt baix índex de volatilitat. És habitual establir els valors que calgui en el moment de la instal·lació de l'aplicació en la xarxa, i ja no tornar-lo a modificar, o fer-ho molt excepcionalment.

Estant connectats al servidor amb el `psql`, podem consultar la seva ubicació en el host demanant pel valor de la variable de sistema que el guarda, `hba_file`, tal com es mostra en les Pantalles 7.34 i 7.35, segons el sistema operatiu que s'utilitzi.

Pantalla 7.34. *Identificació de l'arxiu d'autenticació basada en el host des de MS-DOS.*



Pantalla 7.35. *Identificació de l'arxiu d'autenticació basada en el host des de línux.*

Els registres d'aquest arxiu ocupen un línia, de cinc columnes. Per comprendre en profunditat la funcionalitat de cada una d'elles cal dominar conceptes propis de sistemes operatius més que de bases de dades. Elements definits en capes molt nucleiques dels sistemes operatius com la sincronitzacions entre processos, l'accés compartit a recursos, el tractament d'exclusions mútues i altres conceptes de l'enginyeria.

Per tant, aquí s'exposarà un mínim de contingut referent allò que té incidència quan pretenem posar l'aplicació en producció. En la Pantalla 7.36 es volca el contingut de la part final d'aquest arxiu. Tota la part inicial són comentaris.

pg_hba.conf: Bloc de notes

```
# TYPE  DATABASE        USER            ADDRESS                 METH
# IPv4 local connections:
 host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
#host    all             all             ::1/128                 tru
# Allow replication connections from localhost, by a user with the
# replication privilege.
#host    replication     postgres        127.0.0.1/32            md5
#host    replication     postgres        ::1/128                 md5
```

Pantalla 7.36. *Arxiu de configuració de l'autenticació basada en el host.*

Si li feu una ullada, veureu que l'arxiu `pg_hba` té poc menys de cent línies, de les quals les que no estan comentades es poden comptar amb els dits d'una mà. És a dir, aquest arxiu està comentat amb cura, explicant la seva estructura. El que es relata tot seguit no és més que la traducció d'algunes de les parts inicials d'aquest arxiu.

Des d'MS-DOS, tots els registres tenen cinc columnes. La primera, `TYPE`, indica el tipus de connexió al que fa referència aquesta regla d'autenticació. Els valors que pot contenir és connexió via internet, `IPv4` i `IPv6`, o connexió des d'ordinadors connectats directament al del servidor, `host`. En línux, aquesta columna accepta un valor addicional, `local`, que representen les connexions fetes des del mateix ordinador on resideix el servidor, i que per tant es poden fer en capes inferiors de la OSI, ja sigui utilitzant *sockets* o altres mecanismes de comunicació entre processos. Quan s'utilitza aquest tipus de connexió exclusiva dels sistemes línux, llavors no hi ha la columna `ADDRESS` en la línia corresponent.

La segona columna diu a quines bases de dades es poden connectar els usuaris que utilitzin aquest tipus de connexió, com es veu en la Pantalla 7.36 podem utilitzar el mot clau `all` per dir a totes les bases de dades del clúster. Observeu en les dues últimes línies de l'arxiu, encara que estiguin comentades, que com a base de dades també es pot posar el mot clau `replication`. Això està relacionat amb el que s'ha dit, en la Secció 7.9.2, encara que s'ha vist ben poc.

La tercera columna restringeix els usuaris PostgreSQL que poden connectar-se de la manera indicada en aquesta línia. També es pot posar `all`, i si efectivament es vol restringir l'accés a un conjunt concret d'usuaris, es pot posar el nom d'un arxiu de text pla. Llavors, en coherència, en el mateix directori que indica el paràmetre de configuració `data_directory` hi ha de figurar l'arxiu amb un nom d'usuari en cada línia, i res més.

La columna ADDRESS restringeix les adreces d'internet que poden connectar-se. Això explica la dependència funcional amb la primera columna. En l'exemple de la Pantalla 7.36 es veu que l'única línia descomentada permet l'accés des de qualsevol ordinador que es trobi en la mateixa xarxa local que el del servidor. Això vol dir, en el cas més senzill, tots els ordinadors que comparteixin el router. Aquesta és la configuració per treballar amb el localhost.

I l'última columna descriu l'algorisme de verificació o encriptació de contrasenyes que s'utilitzarà per aquest tipus de connexions. L'`md5` és un dels més utilitzats. N'hi ha molts altres. I si tenim plena confiança, podem posar el mot clau `trust`, i llavors per aquest tipus de connexió no es demanarà contrasenya. Això pot resultar molt interessant quan no en recordem alguna. Respecte aquest extrem, se n'ha fet referència en la Secció 7.3.

Arxiu de mapeig d'usuaris

Com s'ha dit en la Secció 6.1, si a l'establir una connexió a PostgreSQL no es dóna el nom d'usuari amb el paràmetre `-U`, llavors s'interpretarà que el nom de l'usuari PostgreSQL és el mateix que el del sistema operatiu.

L'arxiu `pg_ident.conf` serveix per establir associacions entre usuaris del sistema operatiu i de l'SGBD.

Dels tres arxius de configuració, sens dubte aquest és el menys important, ja que en la major part de les instal·lacions no s'utilitza. El podeu consultar de la mateixa manera que els dos arxius de les seccions anteriors, i veureu que efectivament està completament comentat. Per fer-ne ús cal donar noms clau que identifiquen els mapes de transformació, és a dir les associacions. Llavors, en l'arxiu de configuració `pg_hba.conf` se li ha de dir que pel tipus de connexió que es desitgi s'utilitzi aquell mapa de transformació dels noms dels usuaris del sistema operatiu, als noms dels usuaris PostgreSQL.

Allò desitjable després d'haver seguit aquest darrer capítol és que el lector es vegi amb cor d'investigar per ell mateix sinó tot, una part prou significativa del que permet un sistema gestor de bases de dades. A partir dels instruments sumministrats no hi hauria d'haver problema per arrodonir l'aplicació d'exemple que s'ha anat observant al llarg de tot el llibre. És important saber consultar l'ajuda, i saber organitzar-se l'espai de treball reservant la carpeta principal com a banc de proves. Tot el coneixement adquirit pel que fa al llenguatge procedural pot servir tanmateix per descobrir-ne de nou. No és un llenguatge massa extens i sense massa esforç es poden conèixer totes les seves prestacions. Un cop adquirida una certa fluïdesa, la implementació de disparadors no té més secret. Per altra banda s'ha donat a entendre que el desenvolupament d'aplicacions client que facin les tasques d'interfície no ha de suposar cap esforç per una persona que desenvolupi codi en llenguatges d'alt nivell. I s'ha tancat el capítol presentant la manera de funcionar dels paràmetres de configuració del PostgreSQL.

Apèndix A

Estil

Tal com s'explica en el preàmbul del llibre, el disseny i la implementació de bases de dades és una teoria oberta. Això vol dir que a més dels procediments que cal efectuar per completar un projecte ens queda encara la possibilitat de fer-ho de diferents maneres. En aquest apèndix es detallen les normes que se segueixen al llarg del llibre respecte aspectes menors que no tenen impacte funcional, però de cara la legibilitat del codi resulta convenient establir-les.

A.1 Model Entitat Relació

Els identificadors que es plasmen per les relacions en un model ER són exclusivament els que transcendeixen en la implementació. Això fa que les relacions 1:N es representin com a simples fletxes.

- Tots els identificadors que apareixen en els diagrames ER ja sigui per les entitats, relacions o atributs estan en ASCII de set bits. Això vol dir que no hi ha accents, ni dièresis, ni enyes, ni ces trencades.
- Els identificadors no tenen espais en blanc, o sigui, en la major part dels casos consten d'un sol mot. En cas que calgui, es posa un guió baix per separar paraules.
- Les entitats s'anomenen amb sustantius en singular, excepte pel cas de les entitats febles. Es representen en majúcules en els diagrames.
- Quan cal anomenar relacions s'utilitzen verbs en tercera persona del singular del present d'indicatiu, també en majúscules.
- Els noms dels atributs es posen en minúscula.

- No es posa en cap cas el tipus d'un atribut en el nom. Un atribut que guardi una data inicial es diu `inici`.
- Els identificadors d'atributs que per nom tenen un sustantiu en singular s'acostumen a implementar en strings.
- Els identificadors d'atributs que per nom tenen un sustantiu en plural representen quantitats que s'acostumen a implementar en enters.
- També s'utilitzen sustantius en plural pels noms de les entitats febles i d'atributs multivalorats.

A.2 Model Relacional

Tant els identificadors de relacions com els d'atributs s'escriuen en minúscules, i en font de `màquina d'escriure`. Els valors constants, per exemple les dades en les relacions, s'escriuen en *cursiva*.

A.3 Llenguatge Procedural

Els noms de les funcions tenen el format `funcio_de_variable(v)`. Els paràmetres de les funcions, així com les variables locals tenen noms curts, normalment formats per les inicials del que descriuen.

A.4 Versions dels Sistemes Utilitzats

Pel desenvolupament de la major part del llibre s'ha utilitzat el sistema operatiu Windows 7. Sovint, es fan referències a aspectes exclusius d'aquest sistema sense fer explícit el fet. Igualment, tot el codi implementat en els últims capítols treballa amb la versió 9.3 de PostgreSQL quan s'utilitza en MS-DOS, i la versió 8.1 en línux. Tot i així, per no fer ferragosa la lectura amb aquests detalls, se li suposa al lector la capacitat de comprendre el que s'explica encara que realitzi els seus experiments en altres versions, així com discernir aquest fet i atribuir a la diferència de versions els funcionaments en els que hi aparegui alguna diferència en el diàleg o en el comportament.

Apèndix B

Codis ASCII

En la Taula B.1 es mostra el conjunt dels 95 caràcters imprimibles, amb els codis corresponents, establert en l'American Standard Code for Interchange Information. El primer de tots ells és l'espai blanc amb codi 32.

		32	!33	"34	#35	$36	%37	&38	'39
(40	)41	*42	+43	,44	-45	·46	/47	048	149
250	351	452	553	654	755	856	957	:58	;59
<60	=61	>62	?63	@64	A65	B66	C67	D68	E69
F70	G71	H72	I73	J74	K75	L76	M77	N78	O79
P80	Q81	R82	S83	T84	U85	V86	W87	X88	Y89
Z90	[91	\92	]93	^94	_95	‘96	a97	b98	c99
d100	e101	f102	g103	h104	i105	j106	k107	l108	m109
n110	o111	p112	q113	r114	s115	t116	u117	v118	w119
x120	y121	z122	{123	|124	}125	~ 126			

Taula B.1: *Codis Imprimibles en ASCII de set bits.*

Els primers trenta dos caràcters, del zero al trenta u, són caràcters de control, o en general no imprimibles. Per exemple el caràcter 7 és el bip sonor. El 9, el tabulador, el 10 linia avall, i el 13 el retorn de corró.

Tenint en compte que els caràcters es guarden en un byte, que és un número en binari de 8 xifres i per tant pot representar els codis des del zero al 255, convé tenir en compte que la meitat superior, és a dir, els codis que comencen en 1 que en decimal vol dir a partir del 128, s'anomenen caràcters de vuit bits, i el signe que codifiquen depèn de quina part del món ens trobem. Aquesta part de la taula es diu ASCII estès. En aquest marge hi tenim les vocals accentuades així com la ce trencada o la enya. És perillós utilitzar aquest tipus de caràcters en textos que hagin de ser compilats o interpretats per algun sistema informàtic,

ja que requereix que l'intèrpret en qüestió comparteixi la pàgina de codis amb les dades. Tot i així, si es vol disposar d'un major nombre de codis de caràcters, llavors es pot utilitzar la codificació UTF-8, d'*Unicode Transformation Format*, que tot i treballar amb bytes com unitat de codi per caràcter, permet codis que ocupin varis bytes.

Apèndix C

Dades per l'Exemple del Club Esportiu

En aquest apèndix es proporcionen les dades per poder seguir els exemples dels capítols d'SQL. Les taules es donen sense ordre per representar més acuradament un cas real. Les dades de nombre d'habitants de les ciutats s'han tret de la wikipèdia.

C.1 Taula comarca

comarca
Barcelonès
Alt Empordà
Baix Llobregat
Val d'Aran
Osona
Garraf
Segrià
Montsià
Maresme
Tarragonès

C.2 Taula ciutat

ciutat	habitants	comarca
Cadaqués	2938	Alt Empordà
Badalona	219708	Barcelonès
San Francisco	805235	
Berlín	3499879	
Rio de Janeiro	6320446	
Castelldefels	63077	Baix Llobregat
París	2249975	
Lleida	139809	Segrià
New York	18897109	
Sitges	29140	Garraf
Tona	8085	Osona
Montgat	11.055	Maresme
Tiana	8221	Maresme
Esplugues	46667	Baix Llobregat
Atenes	664046	
Tredòs	154	Val d'Aran
Barcelona	1611822	Barcelonès
Viladecans	65444	Baix Llobregat
Roma	2796102	
Roses	19891	Alt Empordà
Amposta	21511	Montsià

C.3 Taula persona

passaport	nom	cognom	mail	ciutat
27673812M	Carme	Peralta	carmep@ionos.cat	Cadaqués
X3478937A	Carles	Sanàbria	carsan1994@gmail.com	Badalona
47548338K	Anna	Sanàbria	annasanabria@gmail.com	Badalona
45493393Z	Jesús	Hortesa	rexstat143@rediris.es	Castelldefels
C00001549	Michael	Bros	mbros1989@aol.com	San Francisco
294394950	Klauss	Stallman	kstallman@dvw.tum.de	Berlín
39238229E	Sònia	Aragall	sonia1995@mismail.com	
187448338	Rita	Derbeken	matrit@iic.esd.edu	Berlín
46372382N	Roser	Puente	rpmasip@terra.es	Tiana
C01X01TN	Roberto	Rietto	rrrietto@gmail.com	Roma
38433548L	Anna	Margalef	amargalefgarcia@gamil.com	Tiana
32234958K	José	Sanlúcar	josesanmitraet@gmail.com	Castelldefels
38474483Z	Miquel	Vila	mvvalverde@gmail.com	Tredòs
42065765F	Camila	Noriega	cnoriega@ics.cat	Castelldefels
59119283Z	Pere	Camprubí	perec@dpto.erf.org	
27961020N	Pere	Garcia	prx132@yahoo.com	Barcelona
19891898A	Maria	Martín	mariamartin@dia.udr.edu	Barcelona
Y4394950D	Helena	Pérez	hpmiravet@yahoo.com	Tona
37228901C	Sebastià	Fonollà	sfmiranda@miliwat.cat	Sitges
29874567M	Leonardo	Soler	rexstan12@gmail.com	Lleida
05CK02337	Jean Marie	Godard	jmgodard@ille.fr	París
C00021549	Mick	Brown	mibr@esportespot.cat	New York
36940559Y	Magdalena	Pinós	mapi@esportespot.cat	Sitges
38223890Y	Jordi	Parmalat	jparmalat@uab.edu	Badalona
51234329N	Mireia	Matas	mima@esportespot.cat	Barcelona
45847558W	José	González	jogo@esportespot.cat	Montgat
48377283A	Daniel	González	dago@esportespot.cat	Montgat
37866969E	Carme	Ferrer	cafe@esportespot.cat	Amposta
27827228B	Sonia	Colmena	soco@esportespot.cat	Roses
X4534332C	Gabriel	Cobos	gaco@esportespot.cat	Roses
Y3439185D	Boris	Santos	bosa@esportespot.cat	Viladecans

C.4 Taula telefons

passaport	telefon
27673812M	343684473882
05CK02337	0011833493388
05CK02337	655944839
05CK02337	972102293
C01X01TN	968339229
19891898A	677384487
38223890Y	632384493
38223890Y	001434944989
38223890Y	977268394
Y3439185D	934883210
Y4394950D	00343992303920
37228901C	913483494
29874567M	977243758
05CK02337	00012932304493
C00021549	945855948
36940559Y	972395921
51234329N	977602330
45847558W	3431232205
48377283A	972343679
37866969E	934405550
27827228B	639032291
X4534332C	936189283
Y3439185D	605449338

C.5 Taula coneix

coneix	es_coneguda
27673812M	X3478937A
X3478937A	27673812M
27673812M	47548338K
27673812M	294394950
294394950	27673812M
294394950	C00001549
45493393Z	32234958K
32234958K	42065765F
X4534332C	27827228B
27827228B	42065765F

C.6 Taula soci

passaport	alta
27673812M	21/03/2010
X3478937A	12/09/2011
47548338K	01/04/2013
38474483Z	02/03/2014
45493393Z	31/05/2011
C00001549	30/06/2009
294394950	14/06/2012
39238229E	14/11/2011
187448338	17/07/2013
46372382N	15/12/2009
C01X01TN	09/01/2009
38433548L	18/12/2012
32234958K	16/07/2010
42065765F	27/04/2013
59119283Z	26/10/2012
27961020N	09/10/2014
19891898A	09/12/2014

C.7 Taula treballador

passaport	departament	obeeix
Y4394950D	administració	
37228901C	administració	Y4394950D
29874567M	administració	Y4394950D
05CK02337	administració	Y4394950D
C00021549	administració	05CK02337
36940559Y	comercial	
38223890Y	comercial	36940559Y
51234329N	comercial	36940559Y
45847558W	administració	05CK02337
48377283A	administració	37228901C
37866969E	comercial	36940559Y
27827228B	entrenador	
X4534332C	entrenador	
Y3439185D	entrenador	

C.8 Taula esport

nom	preu	jugadors
natació	18.35	1
tennis	21.50	1
tennis dobles	21.50	2
ping-pong	10.30	1
bàsquet	8.60	5
futbol	15.40	11
handbol	9.50	7
voleibol	14.50	6
golf	24.15	1
vela	22.60	2

C.9 Taula fa

soci	esport	quota
27673812M	natació	18.35
X3478937A	natació	18.35
47548338K	tennis	21.50
38474483Z	tennis	21.50
45493393Z	tennis dobles	21.50
C00001549	bàsquet	8.60
294394950	bàsquet	8.60
39238229E	natació	18.35
187448338	bàsquet	8.60
46372382N	natació	18.35
C01X01TN	ping-pong	10.30
38433548L	ping-pong	10.30
32234958K	ping-pong	10.30
42065765F	tennis dobles	21.50
59119283Z	natació	18.35
27961020N	voleibol	14.50
19891898A	tennis	21.50
27673812M	futbol	15.40
X3478937A	futbol	15.40
47548338K	bàsquet	8.60
38474483Z	bàsquet	8.60
27673812M	ping-pong	10.30
X3478937A	ping-pong	10.30
47548338K	golf	24.15
38474483Z	vela	22.60
45493393Z	vela	22.60
C00001549	vela	22.60
294394950	futbol	15.40
39238229E	futbol	15.40
187448338	futbol	15.40
46372382N	futbol	15.40
C01X01TN	natació	18.35
38433548L	natació	18.35
32234958K	voleibol	14.50
42065765F	tennis	21.50
59119283Z	voleibol	14.50
27961020N	natació	18.35
19891898A	ping-pong	10.30

C.10 Taula nomines

passaport	periode	sou_base	retencio
37228901C	28/01/2015	1525.67	8.15
37228901C	29/12/2014	1530.80	8.15
37228901C	29/11/2014	1523.42	8.15
37228901C	29/10/2014	1523.42	8.15
29874567M	28/01/2015	1213.93	10.25
29874567M	29/12/2014	1213.93	10.25
29874567M	29/11/2014	1213.93	10.25
29874567M	29/10/2014	1201.46	10.25
29874567M	28/09/2014	1201.46	6.80
29874567M	30/08/2014	1201.46	6.80
29874567M	29/07/2014	1201.46	6.80
29874567M	29/06/2014	1213.93	6.80
29874567M	31/05/2014	1213.93	6.80
05CK02337	31/01/2014	1450.69	2.00
C00021549	28/01/2015	812.30	3.15
C00021549	29/12/2014	812.30	3.15
C00021549	29/11/2014	915.35	3.15
C00021549	29/10/2014	915.35	3.15
C00021549	28/09/2014	915.35	3.15
C00021549	30/08/2014	915.35	3.15
C00021549	29/07/2014	812.30	3.15
36940559Y	28/01/2015	1090.23	5.00
36940559Y	29/12/2014	1090.23	5.00
36940559Y	29/11/2014	1090.23	5.00
38223890Y	28/01/2015	1102.43	9.10
38223890Y	29/12/2014	1102.43	9.10
38223890Y	29/11/2014	1102.43	7.35
38223890Y	29/10/2014	1060.90	7.35
38223890Y	28/09/2014	1060.90	2.00
51234329N	28/01/2015	1232.02	2.00
45847558W	28/01/2015	729.45	3.59
45847558W	29/12/2014	711.34	3.59
45847558W	29/11/2014	708.10	3.59
45847558W	29/10/2014	703.56	3.59
48377283A	28/01/2015	698.32	11.50
48377283A	29/12/2014	698.32	11.50
37866969E	28/01/2015	1115.45	9.80
37866969E	29/12/2014	1102.76	9.80
37866969E	29/11/2014	1094.07	9.80
27827228B	28/01/2015	1050.30	9.80
27827228B	29/12/2014	1050.30	9.80
X4534332C	28/01/2015	1050.30	9.10
X4534332C	29/12/2014	1050.30	9.10
X4534332C	29/11/2014	1050.30	7.35
X4534332C	29/10/2014	1050.30	7.35
Y3439185D	28/01/2015	3750.12	1.00
Y3439185D	29/12/2014	3753.60	1.00
Y3439185D	29/11/2014	3752.00	1.00

Bibliografia

[1] C. J. Date and Sergio Luis María Ruiz Faudón. *Introducción a los Sistemas de Bases de Datos.* 7 edició. Prentice Hall, 2001.

[2] diccionari.cat. *Diccionari de l'Enciclopèdia Catalana.*

[3] C. Franquesa. *Algorísmia Comentada.* Publicacions UB, 2012.

[4] T. Hastie, R. Tibshirani, and J Friedman. *The Elements of Statistical Learning.* 4a. Edició. Springer-Verlag. DOI 10.1007/978-1-4614-7138-7, 2012.

[5] http://hibernate.org/. *Hibernate. Everything data.*

[6] http://www14.gencat.cat/llc/AppJava/index.html. *Optimot Consultes Lingüístiques.*

[7] http://www.postgresql.org/. PostgreSQL. *The most advanced open source database.* 2014.

[8] http://www.postgresql.org/docs/. Documentació i Manuals de PostgreSQL. En francès i anglès. 2014.

[9] G. James, D. Witten, T. Hastie, and R. Tibshirani. An introduction to statistical learning with applications in r. *4a. Edició*, 2014.

[10] A. Shilberschatz, H. F. Korth, and S. Surdarshan. *Fundamentos de Bases de Datos.* 6a. Edició. Mc Graw Hill, 2014.

Caixes

1.1 *Definició de subconjunt propi.* 12
1.2 *Definició de compatibilitat entre conjunts.* 12

1.3 *Definició de la unió de conjunts.* . . . 17
1.4 *Definició de la intersecció de conjunts.* . . . 19
1.5 *Definició de la diferència de conjunts.* . . . 20
1.6 *Definició del producte cartesià de conjunts.* . . . 22
1.7 *Lleis de De Morgan.* . . . 26
1.8 *Relació entre la lògica de predicats i la teoria de conjunts.* . . . 27
4.1 *Pseudocodi per establir les cardinalitats d'una base de dades BD.* . . . 82
5.1 *Esquema d'una relació en l'àlgebra relacional.* . . . 99
5.2 *Exemple d'esquema de relació en l'àlgebra relacional.* . . . 100
5.3 *Definició formal d'una relació.* . . . 101
5.4 *Definició de compatibilitat entre les relacions r i s.* . . . 102
5.5 *Definició de la relació per la visió cartesiana.* . . . 104
5.6 *Expressió relacional d'una selecció.* . . . 121
5.7 *Expressió relacional d'una projecció.* . . . 123
5.8 *Expressió relacional del producte cartesià entre dues relacions.* . . . 126
5.9 *Expressió relacional de la unió de dues relacions.* . . . 129
5.10 *Expressió relacional de la diferència de relacions.* . . . 130
5.11 *Expressió relacional per al renomenament de qualsevol expressió relacional.* . . . 131
5.12 *Expressió relacional de la intersecció de relacions.* . . . 135
5.13 *La intersecció no és una operació bàsica.* . . . 136
5.14 *Expressió relacional de la reunió natural entre dues relacions.* . . . 137
5.15 *Expressió relacional de la reunió interna entre dues relacions.* . . . 139
5.16 *Expressió relacional de les reunions externes. (a) per l'esquerra. (b) completa. (c) per la dreta.* . . . 140
5.17 *Expressió relacional de la divisió entre dues relacions.* . . . 142
5.18 *La divisió no és una operació bàsica.* . . . 143
5.19 *Expressió per l'assignació de relacions.* . . . 143
5.20 *Assignació d'una relació amb una sola tupla a la relació* `persona`. . . . 144
5.21 *Projecció generalitzada dels atributs d'una relació.* . . . 145
5.22 *Nova versió de l'assignació de la Caixa 5.20.* . . . 146
5.23 *Expressió relacional mínima per les funcions d'agregació.* . . . 148
5.24 *Expressió relacional per les funcions d'agregació.* . . . 150
6.1 *Contingut inicial de l'arxiu* `club.sql`*, escript principal.* . . . 166
6.2 *Domini per l'atribut* `departament` *de la taula* `treballador`. . . . 168
6.3 *Creació d'un domini pels números positius.* . . . 168
6.4 *Domini per l'atribut* `mail` *de la taula* `persona`. . . . 169
6.5 *Arxiu* `comarca.sql`. . . . 170
6.6 *Primera proposta per l'arxiu* `ciutat.sql`. . . . 172
6.7 *Restricció d'eliminació per la clau apuntada.* . . . 172
6.8 *Arxiu* `ciutat.sql`. . . . 174
6.9 *Arxiu* `persona.sql`. . . . 176
6.10 *Contingut actual de l'escript principal, arxiu* `club.sql`. . . . 177
6.11 *Arxiu* `telefons.sql`. . . . 178
6.12 *Arxiu* `coneix.sql`. . . . 179
6.13 *Arxiu* `soci.sql`. . . . 180
6.14 *Arxiu* `treballador.sql`. . . . 181
6.15 *Arxiu* `esport.sql`. . . . 182
6.16 *Arxiu* `fa.sql`. . . . 182
6.17 *Arxiu* `nomines.sql`. . . . 183

6.18 *Contingut final de l'escript principal, arxiu* `club.sql`. 184
6.19 *Estructura fonamental d'una consulta de lectura en SQL.* 187
6.20 *Estructura habitual d'una consulta de lectura en SQL.* 188
6.21 *Arxiu* `club/comarca/inserts.sql`. 189
6.22 *Arxiu* `club/ciutat/inserts.sql`. 189
6.23 *Consulta de tot el contingut de la taula* `persona`. 191
6.24 *Arxiu* `club/coneix/inserts.sql`. 199
6.25 *Format d'una consulta amb criteris d'agregació.* 204
6.26 *Sentència de creació d'una vista.* 210
6.27 *Arxiu* `club/fa/pagament_soci.sql`. 211
6.28 *Forma bàsica de la comanda d'inserció.* 212
6.29 *Forma alternativa de la comanda d'inserció.* 213
6.30 *Forma més complexa de la comanda d'inserció.* 214
6.31 *Sintaxi per la comanda d'actualització.* 215
6.32 *Sintaxi per la comanda d'eliminació.* 216
6.33 *Sintaxi de l'operació d'unió de relacions.* 218
6.34 *Sintaxi de l'operació de diferència de relacions.* 222
6.35 *Sintaxi de l'operació d'intersecció de relacions.* 224
6.36 *Expressió alternativa a la consulta d'intersecció de l'exemple 6.36.* . 225
6.37 *Sintaxi de la reunió natural.* 226
6.38 *Sintaxi de la reunió interna amb la condició* `USING`. 231
6.39 *Sintaxi de la reunió interna amb la condició* `ON`. 232
6.40 *Consulta aniuada.* 238
6.41 *Clàusula lògica* `IN`. 240
6.42 *Definició de la clàusula* `SOME` *amb el comparador genèric.* 241
6.43 *Clàusula lògica* `SOME`. 242
6.44 *Definició de la clàusula* `ALL` *amb el comparador genèric.* 243
6.45 *Clàusula lògica* `ALL`. 243
6.46 *Definició de la clàusula* `EXISTS`. 244
6.47 *Clàusula lògica* `EXISTS`. 244
7.1 *Arxiu* `1-init.sql`. 253
7.2 *Arxiu* `2-taules.sql`. 253
7.3 *Arxiu principal,* `club.sql`. 253
7.4 *Creació de bases de dades.* 255
7.5 *Creació d'usuaris.* 256
7.6 *Creació de grups d'usuaris.* 257
7.7 *Versions més actuals per la creació d'usuaris i grups.* 258
7.8 *Arxiu* `usuaris.sql`. 260
7.9 *Garantir privilegis.* 260
7.10 *Revocar privilegis.* 261
7.11 *Dues transaccions en dues comandes.* 265
7.12 *Transacció de dues comandes.* 266
7.13 *Arxiu* `set_bits.sql`. *Exemple de bucle senzill.* 278
7.14 *Arxiu* `exemples_raises.sql`. *Exemples d'ús de la comanda* `RAISE`. 281
7.15 *Arxiu* `alta_soci.sql`. *Consultes d'actualització.* 282
7.16 *Arxiu* `alta_treballador.sql`. *Consultes d'actualització.* 283
7.17 *Addició de privilegis sobre les funcions en l'arxiu* `usuaris.sql`. . . 284
7.18 *Primera versió arxiu* `nom_i_cognom.sql`. *Selecció d'un sol valor.* . 285
7.19 *Arxiu* `nom_i_cognom.sql`. *Selecció i verificació d'un sol valor.* . . . 286
7.20 *Arxiu* `canonitza.sql`. *Funció auxiliar.* 287

7.21 *Arxiu* `comarca_semblant.sql`. *Funció auxiliar.* 289
7.22 *Arxiu* `esports_socis.sql`. *Exemple d'ús de la instrucció* `RETURN QUERY`. 290
7.23 *Arxiu* `ciutats_esports_socis.sql`. *Recorreguts de consultes.* . . . 292
7.24 *Capçalera d'una funció de tipus* `TRIGGER`. 294
7.25 *Disparador abans de la inserció, per cada fila, en la taula* `comarca`. 296
7.26 *Declaració d'un disparador per la taula* `comarca`. 296
7.27 *Consulta del valor i l'origen d'un paràmetre.* 306

Diagrames

5.1 *Diagrama bivariant com exemple d'ús de les coordenades cartesianes.* 103
5.2 *Exemple de diagrama d'esquema de relació.* 118
5.3 *Diagrama d'esquemes de relació del Model ER de l'exemple.* 119

Figures

1.1 *Conjunts C i D utilitzats en els exemples.* 16
1.2 *Mnemograma per la unió de conjunts.* 17
1.3 *Conjunt Unió $C \cup D$ dels conjunts de la Figura 1.1.* 18
1.4 *Mnemograma per la intersecció de conjunts.* 18
1.5 *Conjunt Intersecció $C \cap D$ dels conjunts de la Figura 1.1.* 19
1.6 *Mnemograma per la diferència de conjunts.* 20
1.7 *Conjunt Diferència $C \setminus D$ dels conjunts de la Figura 1.1.* 20
1.8 *Mnemograma pel producte cartesià de conjunts.* 21
1.9 *Conjunt Producte Cartesià $C \times D$ dels conjunts de la Figura 1.1.* 22

2.1 *Arquitectura Client-Servidor.* 31
2.2 *Model en forma de ceba d'un sistema informàtic.* 32
2.3 *Arquitectura a tres nivells.* 33

3.1 *Graf de 5 nodes i 7 arcs* 42
3.2 *(a) Fortament connexe; (b) Unilateralment connexe; (c) Dèbilment connexe.* 43
3.3 *Graf acíclic connexe, o sigui arbre.* 43
3.4 *Exemple de model de domini pel club d'esports.* 45

4.1 *Representació d'una entitat en un model ER.* 50

4.2 *Representació d'un atribut en una entitat d'un model ER.* 52
4.3 *Representació d'un atribut clau en una entitat d'un model ER.* . . 54
4.4 *Representació d'un atribut multivalorat en una entitat d'un model ER.* 55
4.5 *Representació d'un atribut compost en una entitat d'un model ER.* 57
4.6 *Representació d'un atribut calculat en una entitat d'un model ER.* 59
4.7 *Participació parcial d'E_1 i total d'E_2 en la relació R.* 61
4.8 *Relació 1:1 en un model ER.* 63
4.9 *Relació 1:N en un model ER.* 65
4.10 *Entitat feble E_2 identificada via E_1.* 69
4.11 *Relació jeràrquica entre relacions 1:N.* 72
4.12 *Relació M:N en un model ER.* 73
4.13 *Potència expressiva de les autorrelacions 1:1.* 77
4.14 *Autorelació 1:1 en un model ER.* 78
4.15 *Vectorització d'un arbre. (a) Arbre; (b) Redisposició seqüencial; (c) Vector de predecessors.* 79
4.16 *Autorelació 1:N en un model ER.* 79
4.17 *Autorelació M:N en un model ER.* 81
4.18 *Relació ternària en un model ER.* 84
4.19 *Relació ternària amb cardinalitat 1:M:N.* 86
4.20 *Alternativa incorrecta a una relació ternària.* 86
4.21 *Especialització o generalització en un model ER.* 88
4.22 *Agregació d'una relació. (a) Relació M:N. (b) Entitat agregada.* . 91

5.1 *Una relació és un conjunt de tuples.* 96
5.2 *Representació de la paràbola que descriu la funció $f(x) = x^2$.* . . 100
5.3 *Instància de la relació de la Caixa 5.5: (a) Descripció tabular. (b) Descripció cartesiana.* 104
5.4 *Implementació amb claus foranes en el model relacional d'una relació 1:N del model ER* 110
5.5 *Relació* `persona` *dels exemples. (a) Descripció tabular. (b) Descripció cartesiana.* 120
5.6 *Selecció de persones de* parís. *(a) Descripció tabular. (b) Descripció cartesiana.* 121
5.7 *Selecció de persones de* 35 *anys, de* parís. *(a) Descripció tabular. (b) Descripció cartesiana.* 122
5.8 *Selecció de persones de* 35 *anys, o de* parís. *(a) Descripció tabular. (b) Descripció cartesiana.* 123
5.9 `edat` *i* `ciutat` *de tothom. (a) Descripció tabular. (b) Descripció cartesiana.* 124
5.10 `edat` *de totes les persones. (a) Descripció tabular. (b) Descripció cartesiana.* 125
5.11 `edat` *i* `ciutat` *de les persones que es diuen* anna. *(a) Descripció tabular. (b) Descripció cartesiana.* 126
5.12 *Producte cartesià. (a) Relacions d'entrada. (b) Descripció tabular. (c) Descripció cartesiana.* 127
5.13 *La historieta de l'àlgebra relacional.* 128
5.14 *Inserció de la tupla* ('joan',25,'atenes'). *(a) Descripció tabular. (b) Descripció cartesiana.* 130

5.15 *Eliminació de les persones de* 35 *anys. (a) Descripció tabular. (b) Descripció cartesiana.* 131
5.16 *Renomenament dels atributs de la relació* `persona`*. (a) Descripció tabular. (b) Descripció cartesiana.* 132
5.17 *Última etapa de la resolució de l'exemple 5.10.* 134
5.18 *La intersecció és un menys la diferència.* 136

6.1 *Arquitectura de l'entorn de treball* 154
6.2 *Classificació de les relacions d'una base de dades.* 159
6.3 *Estructura actual de l'espai de treball.* 171
6.4 *Estructura de l'espai amb la taula* `ciutat`. 175
6.5 *Estructura de directoris del projecte.* 185
6.6 *Incorporació d'escripts d'inserció.* 186
6.7 *Implicació sintàctica de la clàusula* `GROUP BY`. 206
6.8 *Exemples de la clàusula* `SOME`. 242
6.9 *Exemples de la clàusula* `ALL`. 243

7.1 *Plana web del manual.* 250
7.2 *Columna de privilegis d'accés a una taula* 264
7.3 *(a) Part d'una relació; (b) Estructura de dades per un atribut índex.* 271
7.4 *Arxiu* `hola_mon.sql` 275

Models

4.1 *Exemple d'una entitat en un model ER.* 51
4.2 *Exemple de l'ús d'atributs en l'entitat del Model 4.1.* 52
4.3 *Exemple més acurat de l'ús d'atributs en l'entitat del Model 4.1.* . . . 53
4.4 *Representació d'una entitat completa en un model ER.* 55
4.5 *Exemple d'atribut multivalorat en el Model 4.4.* 56
4.6 *Exemple d'atribut compost en el Model 4.4.* 58
4.7 *Exemple d'atribut calculat en el Model 4.4.* 60
4.8 *Exemple de relació 1:1 en un model ER.* 64
4.9 *Simplificació del Model 4.8.* 64
4.10 *Representació d'una entitat amb atributs incorrectes en un model ER.* 65
4.11 *Model correcte per l'exemple del Model 4.10.* 66
4.12 *Model correcte simplificat per l'exemple del Model 4.10.* 66
4.13 *Exemple del Model 4.11 amb una relació nova.* 67
4.14 *Entitat de suport* `COMARCA`. 68
4.15 *Exemple d'ús d'una entitat feble.* 70
4.16 *Model inicial abans d'introduir una relació M:N.* 74
4.17 *Exemple de relació M:N en un model ER.* 75
4.18 *Exemple d'atributs en una relació M:N d'un model ER.* 76

4.19 *Exemple d'autorelació 1:1 en el Model 4.16.* 78
4.20 *Exemple d'autorelació 1:N en el Model 4.16.* 80
4.21 *Exemple d'autorelació M:N en el Model 4.16.* 81
4.22 *Exemple de relació ternària afegit al Model 4.18.* 85
4.23 *Exemple d'especialització o generalització en un model ER.* 89
4.24 *Bunyol inadmissible en un model ER.* 89
5.1 *Exemple d'esquema de relació.* 107
5.2 *Exemple d'esquema de relació amb clau primària binomial.* 107
5.3 *Model ER per l'aplicació del club esportiu.* 111
5.4 *Primeres relacions de la versió relacional de l'exemple 5.3* 112
5.5 *Incorporació de la relació* `persona`. 112
5.6 *Transformació d'un atribut multivalorat.* 113
5.7 *Transformació d'una autorelació M:N al model relacional.* 113
5.8 *Model ER pels exemples, repetició del 5.3.* 114
5.9 *Transformació d'entitats especialitzades i d'autorelació 1:N.* 115
5.10 *Transformació de relacions M:N prèvia descripció de les entitats relacionades.* 115
5.11 *Model Relacional del disseny del Model Entitat Relació 5.3.* 117
5.12 *Fragment Model 5.11 relacional.* 140
5.13 *Fragment del Model 5.11 per als exemples de les funcions d'agregació.* 149
6.1 *Model ER de la base de dades.* 164
6.2 *Model Relacional del disseny del Model Entitat Relació 6.1.* 165
7.1 *Permisos i privilegis.* 254
7.2 *Permisos i Privilegis.* 258
7.3 *Fragment superior del Model 7.2.* 263
7.4 *Model ER per l'aplicació del club esportiu.* 273