Estudi i avaluació de les funcionalitats de PosgreSQL.

Estudiant:
Miracle Cerón Mercadé
Enginyeria en Informàtica

Consultora:
M. Elena Rodríguez González

Data Lliurament:
10 de gener 2006

Dedicatòria i agraïments

A la meva família. A la meva mare, pel seu ajut en les tasques diàries durant aquest darrers anys, i al meu pare que hagués estat orgullós de veure que per fi he complert amb escreix la meva il·lusió.

Als companys d'estudis que he conegut en aquests anys i que espero continuem amics per molts més.

A l'amic que em va animar a començar l'Enginyeria Informàtica i a qui agraeixo la col·laboració en aquest projecte. Amb qui he compartit durant aquests darrers anys moltes hores de dedicació, amb bones i males estones però amb un balanç molt positiu.

A l'Elena, consultora d'aquest projecte, per donar-me l'oportunitat de fer-lo i pel seu valuós assessorament.

A la UOC, per fer possible que gent com jo pugui trobar una manera d'estudiar ja que, en el meu cas, no hagués estat possible sense aquesta Universitat.

Resum

Estudi i avaluació de les funcionalitats de PosgreSQL.

L'objectiu d'aquest projecte ha estat estudiar les característiques del producte PostgreSQL i posar-les a prova.

Tot el treball s'ha fet entorn a la darrera versió disponible al començar, i que és la 8.0.

En el projecte es mostra un resum dels principals procediments oferts per PosgreSQL, per tal de gestionar les bases de dades i els usuaris, concedint i revocant els seus privilegis.

No s'ha entrat a comentar ni el disseny lògic ni el disseny físic d'una base de dades, sols s'ha intentat ressaltar les peculiaritats i la forma d'implementar-ho en aquest Sistema de Gestió de Bases de Dades (SGBD) en concret, ja que cada SGBD ha desenvolupat un llenguatge propi, d'acord amb les característiques de l'entorn. De totes maneres hi ha gran similitud entre gran part dels components, entre PostgreSQL i d'altres gestors, com es podrà anar veient.

Es poden veuen les ordres en llenguatge SQL per crear i esborrar una base dades, i per inserir, esborrar i modificar les diferents taules que la componen i s'han tractat altres aspectes com els dominis, assercions i vistes. També s'han estudiat els procediments emmagatzemats i els disparadors, així com la creació i la destrucció d'índexs.

Per últim els subllenguatges especialitzats: SQL hostatjat, SQL/CLI i JDBC.

Pels exemples s'ha utilitzat la base de dades UOCReserva, que es tracta d'una base de dades d'una reserva d'animals salvatges implementada en les pràctiques de Bases de Dades I (BDI) i Bases de Dades II (BDII) de la UOC, en PostgreSQL i que es troben en l'annex d'aquest projecte.

Índex

1.- Introducció

1.1.- Descripció

Aquest Projecte Fi de Carrera (PFC) consisteix en fer un anàlisi detallat de les funcionalitats del Sistema Gestor de Bases de Dades (SGBD) PostgreSQL. Es començarà per descriure el moment històric i les causes del seu naixement, per passar tot seguit a fer un ampli anàlisi de la implementació de les seves funcionalitats, complimentant-ho amb comparatives entre l'SGBD d'estudi i d'altres existents (tant de comercials, com de l'entorn de la comunitat del programari lliure). L'anàlisi finalitzarà amb l'estudi del programari addicional que cal que tingui tota base de dades, tant per permetre la connectivitat (JDBC/ODBC) com per editar sentencies SQL *ad hoc*.

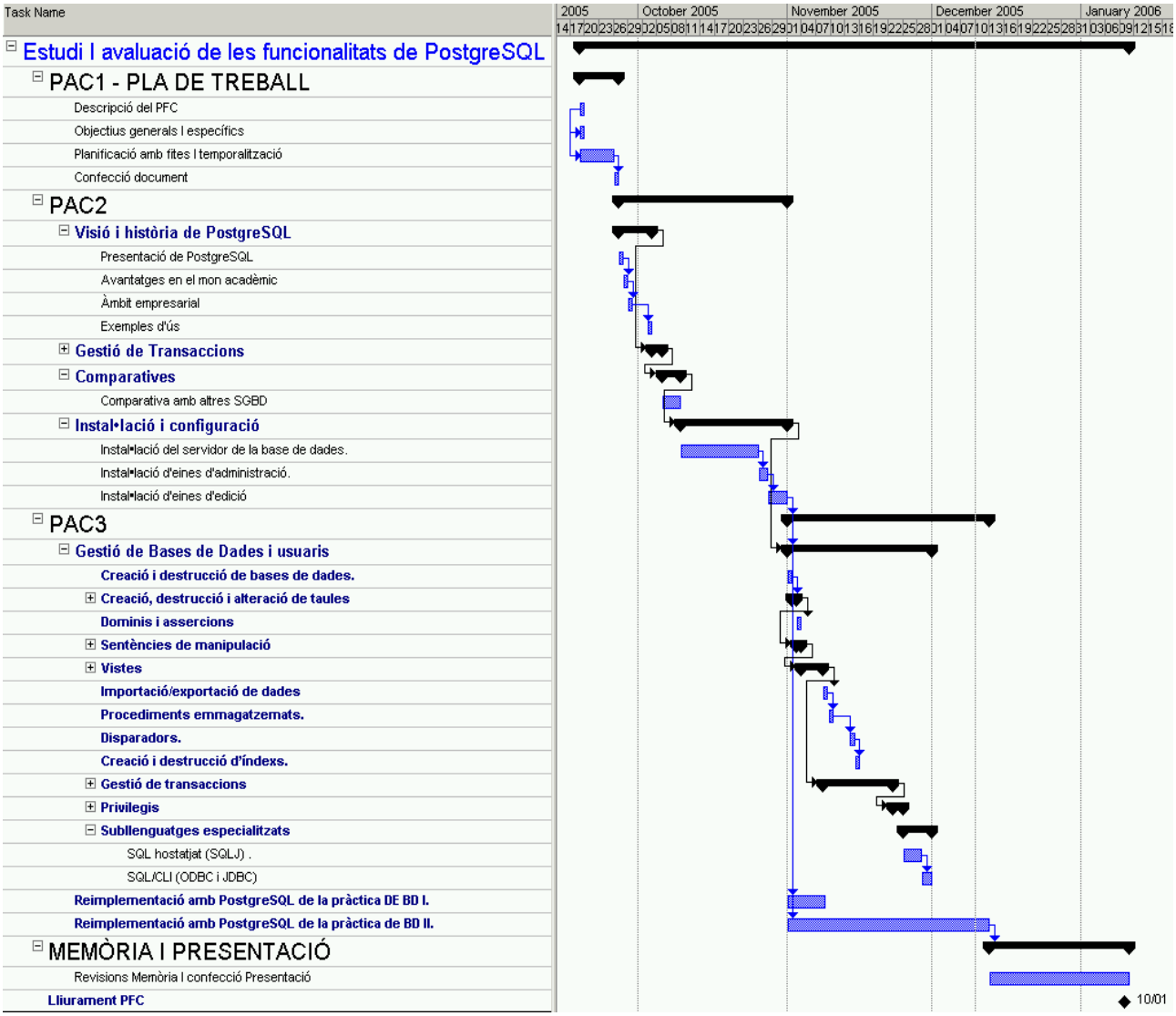
També es reflectirà la importància del moment històric en que va néixer aquest programari. Al néixer PostgreSQL en un entorn universitari (Universitat de Califòrnia, Berkeley - 1977), amb l'objectiu de demostrar els conceptes teòrics que s'esperaven dels SGBD futurs, ha estat pioner en la implementació de funcionalitats que comercialment, i segons estrictes criteris de rendiment econòmics, possiblement encara no estarien implementades en cap SGBD.

1.2.- Objectius

Durant el desenvolupament del PFC s'espera haver provat totes les característiques que s'utilitzarien del programari al llarg d'un cicle educatiu de tipus universitari. Així, un dels objectius d'aquest projecte serà el de mostrar els procediments que ofereix PostgreSQL per tal de gestionar les bases de dades i els usuaris, incloent exemples pràctics.

També serà prioritari la comprovació metòdica del PostgreSQL per assegurar que acompleix estrictament les normes ACID, i la verificació de les opcions de connectivitat JDBC/ODBC existents, que serviran per a garantir que l'SGBD sigui accessible des de diferents llenguatges i editors SQL.

1.3.- Planificació



1.4.- Productes obtinguts

Durant l'execució de la part més pràctica del PFC, s'ha procedit a adaptar les pràctiques de les assignatures Bases de dades I i Bases de Dades II (que estaven fetes per a funcionar sobre l'SGBD Informix), per a que siguin executables en un PostgreSQL (independentment de la plataforma en que funcioni).

S'han inclòs alguns exemples al llarg de la memòria i la solució complerta de les mateixes s'adjunta en l'annex.

1.5.- Descripció de la memòria

Al llarg del projecte s'especifiquen tots els passos per a fer les següents tasques i exemples de les mateixes:

- Creació i destrucció de bases de dades.
- Creació, destrucció i alteració de taules (columnes, tipus de dades, restriccions de columna i de taula, definicions per defecte, efectes d'esborrat i de modificació en claus foranes).
- Dominis i assercions.
- Importació/exportació de dades.
- Sentències de manipulació (consultes, insercions, esborrats, modificacions, unió, intersecció i diferència).
- Vistes (creació i manipulació). Traducció de consultes sobre vistes. Vistes materialitzades.
- Procediments emmagatzemats.
- Disparadors.
- Privilegis (concessió i revocació de privilegis, rols).
- Creació i destrucció d'índexs.
- Gestió de transaccions (inici i finalització de transaccions, nivells d'aïllament, granularitats, sentències per bloquejar taules etc.).
- Subllenguatges especialitzats: SQL hostatjat (SQLJ i ECPG), SQL/CLI (ODBC i JDBC).

2.- Visió i història de PostgreSQL

2.1.- Presentació de PostgreSQL

PostgreSQL és un SGBD Relacional Orientat a Objectes lliure que va ser alliberat sota la llicència BSD (Berkeley Software Distribution - Distribució de Software Berkeley).

És una alternativa a altres sistemes de bases de dades de codi obert (com MySQL, Firebird i MaxDB), així com sistemes propietaris com a Oracle o DB2, i és el gestor de bases de dades de codi obert més avançat avui en dia, oferint control de concurrència multiversió, suportant gairebé tota la sintaxi SQL (incloent subconsultes, transaccions, i tipus i funcions definides per l'usuari). Algunes de les seves principals característiques són: claus foranes, disparadors (*triggers*), vistes, integritat transaccional, accés concurrent multiversió, herència de taules, tipus de dades i operacions geomètriques. Comptant també amb un ampli conjunt d'enllaços amb llenguatges de programació, incloent C, C++, Java, perl, tcl i python.

PostgreSQL és el resultat de l'evolució començada amb el projecte Ingres a la Universitat de Berkeley. El líder del projecte, Michael Stonebraker va abandonar Berkeley per comercialitzar Ingres el 1982, però eventualment va tornar a la universitat. Després del seu retorn a Berkeley el 1985, Stonebraker va començar un projecte post-Ingres per resoldre els problemes amb el model de base de dades relacional que havien estat aclarits a començaments dels anys 1980. El principal d'aquests problemes era la incapacitat del model relacional de comprendre 'tipus', és a dir, combinacions de dades simples que conformen una unitat. Actualment aquests són anomenats objectes.

El projecte resultant va ser anomenat Postgres. La implementació del SGBD Postgres va començar el 1986. Els conceptes inicials per al sistema van ser presentats a The Design of Postgres i la definició del model de dades inicial va aparèixer a The Postgres Data Model. El disseny del sistema de regles va ser descrit en aquell moment en The Design of the Postgres Rules System. La lògica i arquitectura del gestor d'emmagatzemament van ser detallades a The Postgres Storage System. A partir d'aquell any, l'equip va alliberar una sèrie de documents descrivint la base dels sistemes. El primer sistema de proves va ser operacional el 1987 i va ser mostrat en la Conferència ACM-SIGMOD de 1988. La primera versió va ser alliberada a un petit grup d'usuaris el juny de 1989, seguit per la versió 2 amb un sistema de regles reescrit el juny de 1990. Per a la versió 3, alliberada el 1991, el sistema de regles va ser reescrit novament, però també va agregar suport per a múltiples administradors d'emmagatzemament i un sistema de consultes millorat. Cap a 1993, Postgres havia crescut immensament en popularitat i tenia una gran demanda de noves funcionalitats. Després d'alliberar la versió 4, el projecte va ser abandonat. Majoritàriament, les següents versions es van centrar en millorar la portabilitat i la fiabilitat.

Malgrat que el projecte Postgres hagués finalitzat oficialment, la llicència BSD, sota la qual Postgres havia estat alliberat, va permetre a desenvolupadors de codi obert d'obtenir una còpia del codi per continuar el seu desenvolupament.

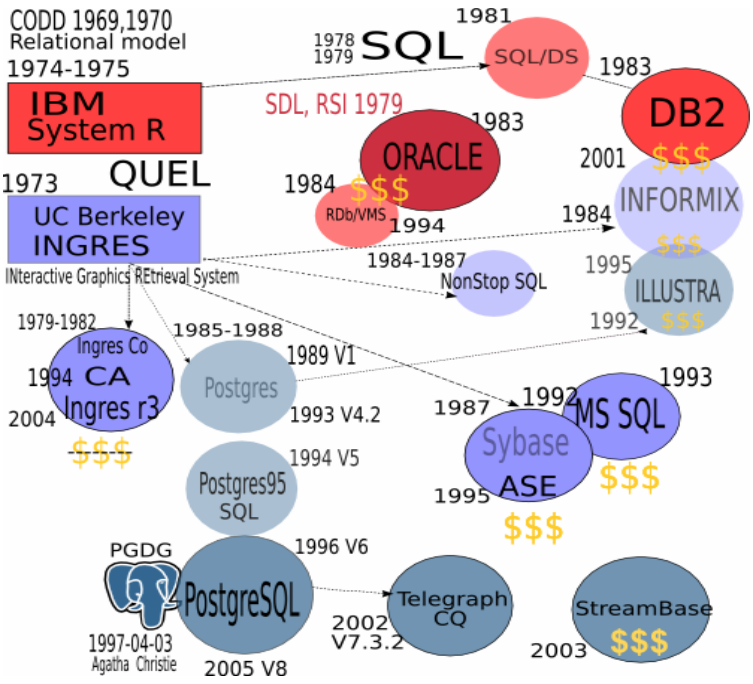
Postgres forma part de la implementació de moltes aplicacions d'investigació i producció. Entre elles: un sistema d'anàlisi de dades financeres, un paquet de monitoratge de rendiment de motors a reacció, una base de dades de seguiment d'asteroides i diversos sistemes d'informació geogràfica. També s'ha utilitzat com una eina educativa a diverses universitats. Finalment, *Illustra Information Technologies* (posteriorment absorbida per Informix) va prendre el codi i el va comercialitzar.

En 1994, Andrew Yu i Jolly Chen van afegir un intèrpret de llenguatge SQL a Postgres. Postgres95 va ser publicat a continuació en la Web. El codi de Postgres95 va ser adaptat a ANSI C i la seva mida reduïda en un 25%. Molts canvis interns van millorar el rendiment i la facilitat de manteniment.

El 1996, es va fer evident que el nom "Postgres95" no resistiria el pas del temps. Es va elegir un nou nom, PostgreSQL, per reflectir la relació entre el Postgres original i les versions més recents amb capacitats SQL. Alhora, van fer que els números de versió partissin de la 6.0, tornant a la seqüència seguida originalment pel projecte Postgres.

Durant el desenvolupament de Postgres95 es va posar èmfasi en identificar i entendre els problemes en el codi del motor de dades. Amb PostgreSQL, l'èmfasi va passar a augmentar característiques i capacitats, encara que el treball continua en totes les àrees.

En el gràfic següent podem veure esquemàticament l'evolució dels SGBD relacionals:



2.2.- Característiques de PostgreSQL.

Característiques principals.

- Conforme als estàndards ANSI SQL92 i SQL99
- Seguretat, confiabilitat i suport
- Escalabilitat

Característiques avançades.

- Suport de transaccions (ACID)
- Integritat referencial
- Disparadors (*triggers*)
- Regles del sistema (*rules*)
- Suport subconsultes
- Control de Concurrència Multi Versió (MVCC)

Suport a múltiples Sistemes Operatius.

- **GNU/Linux** (qualsevol distribució)
- **Unix** (AIX, BSD, HP-UX, SGI IRIX, Mac OS, Solaris, SunOS, Tru64)
- **BeOS**
- **Windows** (de forma nativa des de la v8.0 en la que es basa aquest treball)

2.3.- Utilització en docència.

El món educatiu és el segment d'usuaris ideal per llançar aquestes iniciatives.
És possible que al principi s'aposti pel programari lliure per diners, ja que les llicències de programari propietari per posar en marxa el projecte d'un ordinador per cada dos alumnes en un centre pot tenir un cost important, mentre que amb GNU/Linux el cost és considerablement molt més baix.

El fet de que el programari lliure arribi a les mans de l'usuari sense cost és una conseqüència del model: costa molt produir programari (lliure o propietari) en temps i esforç, en treball indirecte d'universitats i centres d'investigació, en diners aportat per empreses, beques, etc...

El programari lliure s'oposa al programari propietari i a tota aquella pràctica que impedeix la lliure circulació del coneixement. El programari lliure, davant el propietari, a més de ser gairebé gratuït, permet enriquir els programes i divulgar-los. El programari objecte d'aquest estudi, el SGBD PostgreSQL està més dirigit a centres de secundària i universitats més especialitzades. És per això que la seva utilitat encara és més gran perquè la seva utilització pot ajudar a la divulgació i enriquiment, ja que pot ser objecte d'investigació, sobre tot en l'àmbit universitari.

2.4.- Utilització en l'àmbit empresarial.

Fins al moment, la incorporació d'aquest programari a l'empresa ha estat fruit de l'esforç de persones convençudes, compromeses amb la Filosofia del Programari Lliure i pertanyents a la mateixa.

Recentment veiem l'esforç de certs fabricants de maquinari que ofereixen sistemes operatius de programari lliure comprometent-se amb el seu suport (DELL, HP), o alguns fabricants de programari propietari que han portat les seves solucions a aquests sistemes operatius i ofereixen el suport conjunt d'ambdues peces (ORACLE, SAG).

L'objectiu d'aquestes simbiosis és reduir el cost de propietat del client al mateix nivell de suport i prestacions. Aquest retall de costos els permet oferir solucions a segments del mercat amb poder de compra molt baix (ORACLE SMALL BUSINESS SUITE), però també existeixen simbiosis amb el programari lliure per abordar al gran compte (IBM amb Linux en els Z/series) eliminant les granges de servidors.

De la mateixa forma que en el passat les grans consultores se centraven en generar valor per als seus clients usant el programari dels sistemes propietaris, per passar-se més tard al dels sistemes oberts, després a les aplicacions d'Internet, i per últim al món Java, ha arribat el moment de fer un nou salt i ampliar la seva oferta al món del programari lliure.

L'elecció de les corporacions locals dins del sector públic és molt meditada, ja que és un mercat amb pressuposts molt ajustats, i per tant amb una gran avidesa d'incorporar arquitectures multicapa i multicanal a uns costos reduïts. En aquest moment, en que segurament, qualsevol ajuntament està una mica informatitzat, fins i tot pot ser rar aquell que no disposi d'una xarxa local.

La proposta d'incorporació de programari lliure en els ajuntaments comença per aquells que desitgen millorar els seus serveis d'intranets i desitgen tenir presència a Internet sense necessitat de comptar amb personal especialitzat, per, a partir d'allà, estudiar la conveniència i oportunitat d'una estratègia de migració dels sistemes verticals al programari lliure, tant des del punt de vista del "front" com del "back office", i per què no, acompanyar a ajuntaments com el de Barcelona o Madrid en una migració suau i ordenada.

Ja hi ha moltes empreses que utilitzen el SGBD PostgreSQL. Per exemple La Universitat d'Oxford utilitza PostgreSQL des de 1998, i té sistemes en la base de dades Open Source per a correu electrònic, ambient d'aprenentatge virtual, portal web, sistema de registre acadèmic i suport tècnic.

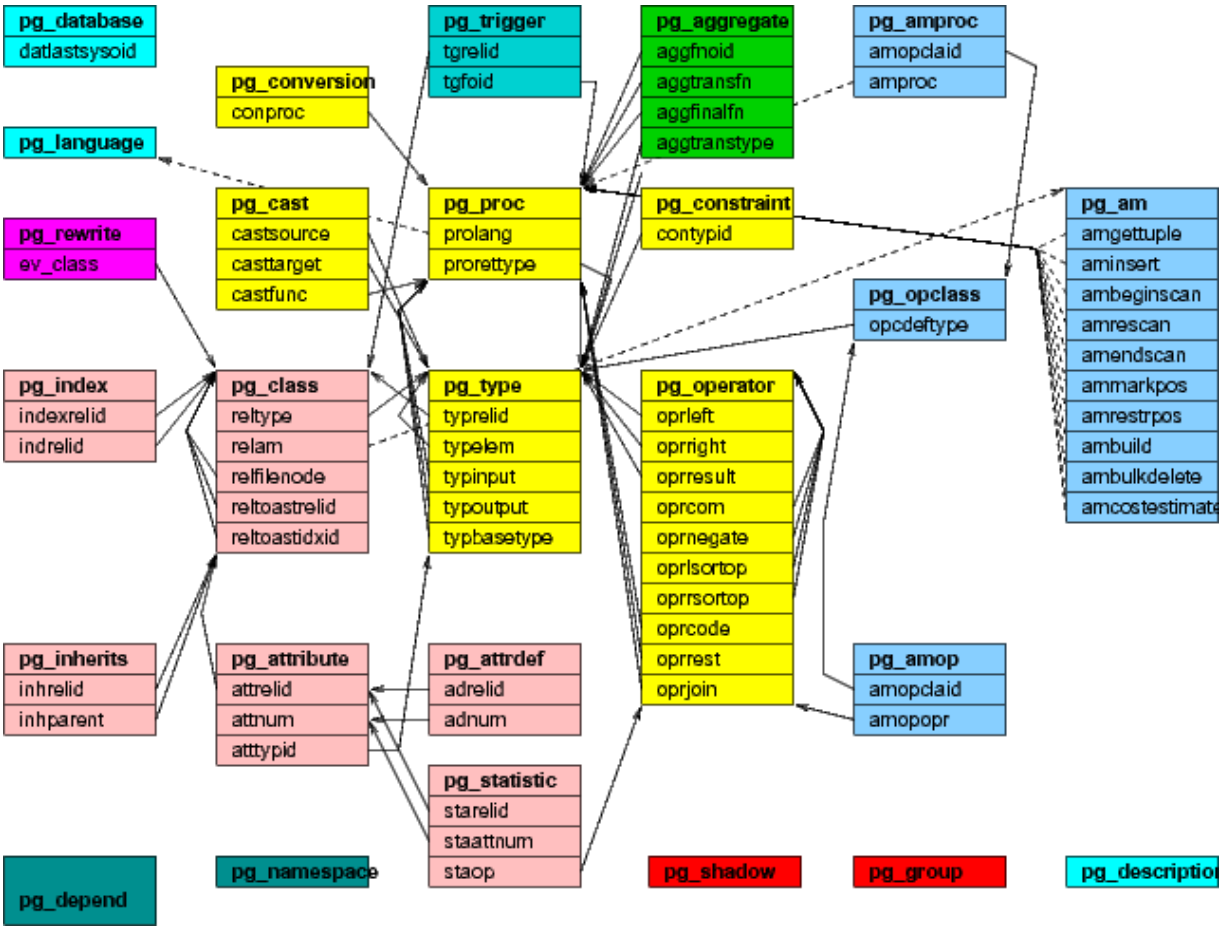
L'elecció de PostgreSQL va ser per la posta en marxa d'un servei d'enviament de correu electrònic per a antics alumnes d'Oxford University. El servei permet als graduats superiors enviar correus electrònics d'un compte d'universitat a qualsevol altra adreça de correu electrònic, permetent-los quedar-se la mateixa adreça de correu electrònic per a sempre.

Un altre exemple és d'una empresa de Madrid. Fins ara no es podia utilitzar un TPV amb Linux. Calia recórrer a solucions cares i privatives. Existeix un programa nou que s'anomena nTPV, que és lliure i per a Linux, i cobreix per fi aquest àrea. Està sent desenvolupat per una empresa espanyola i utilitza PostgreSQL com a sistema gestor de base de dades.

3.- Creació i destrucció de bases de dades.

El motor de PostgreSQL cal que estigui corrent i l'usuari que dóna la comanda cal que sigui l'administrador de PostgreSQL, o haver-hi obtingut privilegis per part d'aquest per crear bases de dades.

En la figura següent es veu el diagrama del catàleg del sistema PostgreSQL, on hi ha tota la informació sobre els objectes del sistema, els operadors i els mètodes d'accés a ells. Durant la inicialització PostgreSQL (comanda `initdb`) es creen dues bases de dades - template0 i template1, que contenen la col·lecció de funcionalitats predeterminades. Qualsevol altra base de dades hereta de template1. Així, els objectes i els mètodes utilitzats més freqüentment es poden agregar en el catàleg del sistema template1.



Un servidor de PostgreSQL pot gestionar moltes bases de dades en un servidor i ser automàticament l'administrador de la base de dades que s'acaba de crear. Els noms de les bases de dades han de començar per una lletra i són limitats a una longitud total de 32 caràcters.

Ho faci un o l'altre el sistema és el mateix i la sentència és `CREATE DATABASE nom_BD`, i la sintaxi és la següent:

```
CREATE DATABASE nom_BD
     [ [ WITH ] [ OWNER [=] propietari ]
            [ TEMPLATE [=] plantilla ]
            [ ENCODING [=] codificació ]
            [ TABLESPACE [=] espai_de_taules] ]
```

Descripció::

`Nom_BD`
Nom de la base de dades a crear.

PARÀMETRES OPCIONALS:

`OWNER`, el nom del propietari. Si s'omet s'entén que el propietari és l'usuari que executa la comanda.

`TEMPLATE`, el nom de la plantilla des de la que crearem la nova base de dades. Si no ho posem utilitzarà el 'template1'.

`TABLESPACE`, nom de l'espai de taules que volem associar amb la nova base de dades.

`ENCODING`, tipus de codificació a utilitzar en la nova base de dades (BD). Per exemple 'SQL_ASCII'. Els tipus de codificacions possibles són els següents:

Nom	Descripció
SQL_ASCII	ASCII
EUC_JP	Japanese EUC
EUC_CN	Chinese EUC
EUC_KR	Korean EUC
JOHAB	Korean EUC (Hangle base)
EUC_TW	Taiwan EUC
UNICODE	Unicode (UTF-8)
MULE_INTERNAL	Mule internal code
LATIN1	ISO 8859-1/ECMA 94 (Latin alphabet no.1)
LATIN2	ISO 8859-2/ECMA 94 (Latin alphabet no.2)
LATIN3	ISO 8859-3/ECMA 94 (Latin alphabet no.3)
LATIN4	ISO 8859-4/ECMA 94 (Latin alphabet no.4)
LATIN5	ISO 8859-9/ECMA 128 (Latin alphabet no.5)
LATIN6	ISO 8859-10/ECMA 144 (Latin alphabet no.6)

Nom	Descripció
LATIN7	ISO 8859-13 (Latin alphabet no.7)
LATIN8	ISO 8859-14 (Latin alphabet no.8)
LATIN9	ISO 8859-15 (Latin alphabet no.9)
LATIN10	ISO 8859-16/ASRO SR 14111 (Latin alphabet no.10)
ISO_8859_5	ISO 8859-5/ECMA 113 (Latin/Cyrillic)
ISO_8859_6	ISO 8859-6/ECMA 114 (Latin/Arabic)
ISO_8859_7	ISO 8859-7/ECMA 118 (Latin/Greek)
ISO_8859_8	ISO 8859-8/ECMA 121 (Latin/Hebrew)
KOI8	KOI8-R(U)
ALT	Windows CP866
WIN874	Windows CP874 (Thai)
WIN1250	Windows CP1250
WIN	Windows CP1251
WIN1256	Windows CP1256 (Arabic)
TCVN	TCVN-5712/Windows CP1258 (Vietnamese)

A continuació es crea la BD de l'exemple, UOCReserva, amb propietari 'postgres', codificació 'SQL_ASCII' i l'espai de taules per defecte:

```
CREATE DATABASE UOCReserva
      WITH OWNER = postgres
      ENCODING = 'SQL_ASCII'
      TABLESPACE = pg_default;
```

Això hauria de produir com a resposta:

```
CREATE DATABASE
```

Si el servidor no estigués connectat a l'intentar crear una base de dades, ens avisaria amb un missatge. El mateix si l'usuari no tingués prou privilegis.

PostgreSQL permet crear una base de dades en un lloc diferent del que va ser destinat durant la instal·lació. Qualsevol consulta a la base de dades es fa a través del motor de la base de dades, de manera que el lloc on sigui creada la base de dades ha de permetre l'accés al motor.

Les ubicacions alternatives de bases de dades es creen i són referides per mitjà de una variable d'estat que dóna el camí absolut del lloc on s'emmagatzemarà la base de dades. Aquesta variable d'estat ha d'haver estat definida abans d'arrencar el motor i

el lloc per a on apunta ha de permetre l'escriptura des del compte de l'administrador de PostgreSQL.

Per raons de seguretat i d'integritat, a qualsevol camí o variable d'estat donada se li agreguen al final alguns camins addicionals. Les ubicacions alternatives deuen ser preparades executant: `initlocation 'àrea d'emmagatzemament alternativa'`.

Aleshores la sentència de creació seria:

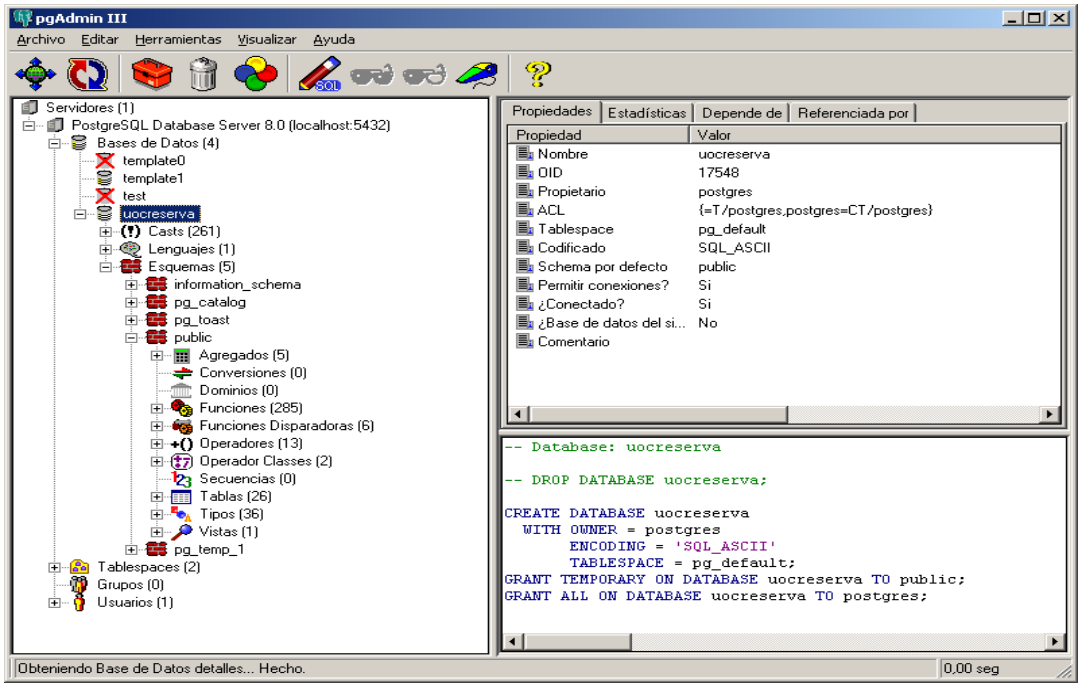
```
CREATE DATABASE UOCReserva
WITH LOCATION = 'àrea d'emmagatzemament alternativa';
```

Per destruir la base de dades cal utilitzar '`DROP DATABASE`'. Per destruir la base de dades que hem creat utilitzarem la sentència següent:

```
DROP DATABASE UOCReserva;
```

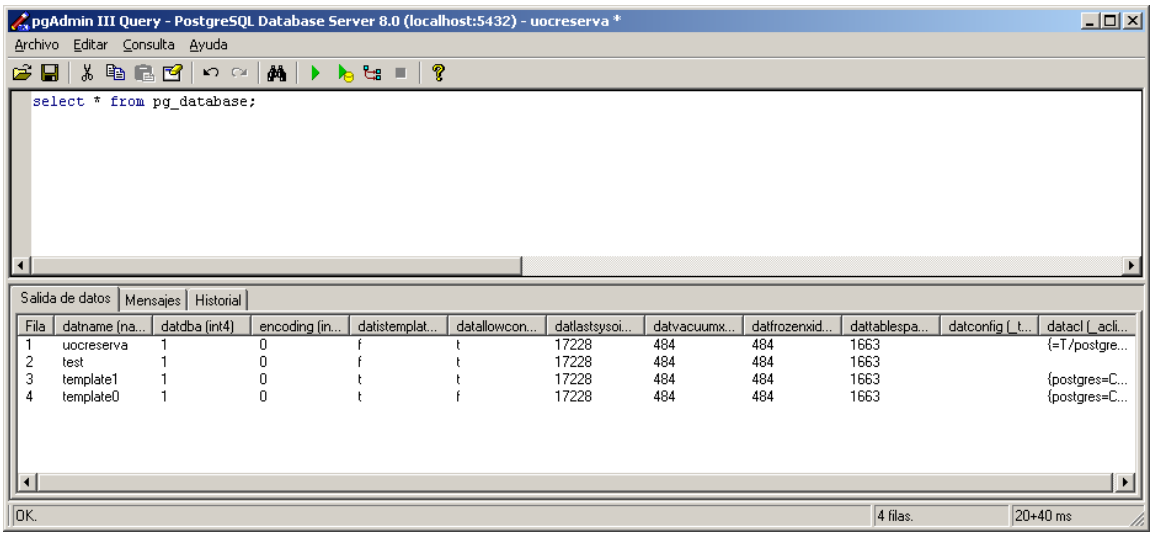
Aquesta acció físicament treu tots els arxius associats a la base de dades i no es pot desfer.

Després de crear la base de dades UOCReserva consultem amb el pgAdmin (eina per treballar amb PostgreSQL i que ja porta incorporat al fer l'instal·lació):

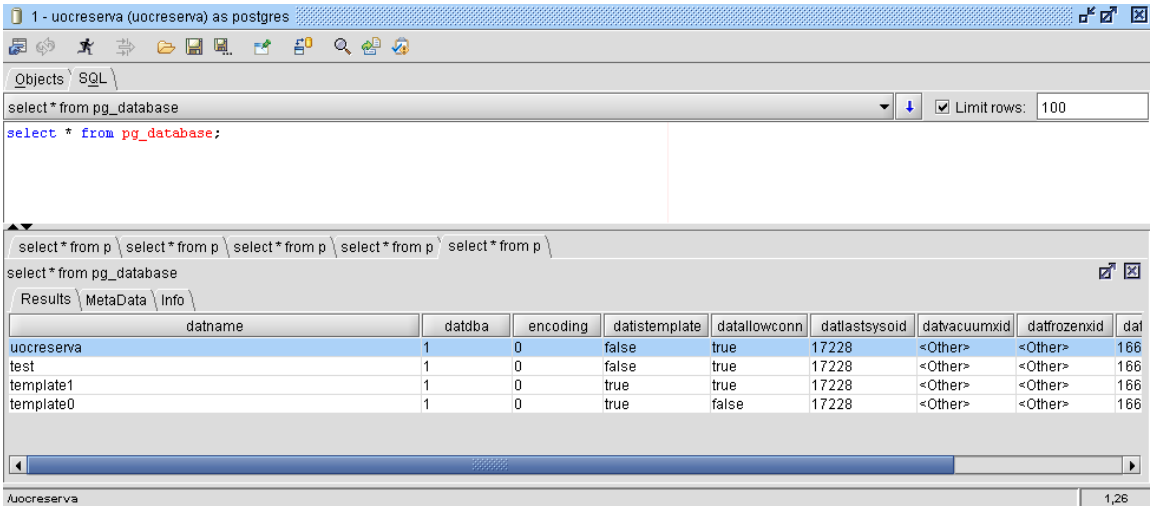


Podem veure les bases de dades existents en aquest moment consultant la vista del sistema pg_database. Aquesta consulta la podem fer des de pgAdmin o des d'un editor d'SQL, com per exemple l'Squirrel.

Amb pgAdmin:



Amb Squirrel:



4.- Creació, destrucció i alteració de taules.

Una vegada està creada la base de dades ja es pot crear l'estructura d'emmagatzemament de les dades: les taules. Per crear les taules s'utilitza la sentència 'CREATE TABLE', que té l'estructura següent:

```
CREATE [ [ LOCAL ] { TEMPORARY | TEMP } ] TABLE nom_taula (
     { nom_columna tipus_dada_columna [ DEFAULT expressió_defecte
]
       [restriccions_columna , ... ] ]
       | restriccions _taula
       | LIKE taula_pare [ { INCLUDING | EXCLUDING } DEFAULTS ] }
[, ... ]
)
[ INHERITS ( taula_pare [, ... ] ) ]
[ WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE espai_de_taules ]
```

Descripció:

LOCAL, aquest paràmetre l'accepta PostgreSQL per compatibilitat amb l'SQL estàndard però no té cap efecte. Igual passa amb 'GLOBAL' de l'SQL Estàndard.

TEMPORARY o TEMP, serveixen per indicar que la taula es crearà com una taula temporal. Aquest tipus de taules s'eliminen automàticament al final d'una sessió, o, opcionalment al final d'una transacció.

DEFAULT, serveix per assignar valors per omissió en una determinada columna.

LIKE, especifica una taula de la que la nova taula copiarà automàticament tots els noms de les columnes, els tipus de les dades i totes les restriccions. Si s'especifica INCLUDING DEFAULTS es copiaran sols les expressions per omissió de les definicions de columna copiades.

INHERITS, especifica una llista de taules de les que la nova taula hereta automàticament totes les seves columnes.

WITH OIDS o WITHOUT OIDS, aquesta clàusula opcional especifica si les files de la taula nova haurien de tenir OIDs (identificadors d'objecte) assignats a ells o no.

ON COMMIT, per controlar el comportament| de taules provisionals al final d'un bloc de transacció. Hi ha tres possibilitats: PRESERVE ROWS, DELETE ROWS i DROP. PRESERVE ROWS, és l'opció per defecte i no es fa cap acció. DELETE ROWS, s'eliminen totes les files

de la taula temporal al final de la transacció. DROP, s'elimina la taula temporal al final de la transacció.

TABLESPACE, nom de l'espai de taules on volem crear la nova taula.

Es pot introduir les sentències amb els salts de línia. PostgreSQL reconeixerà que la comanda no s'acaba fins al punt i coma.

L'espai blanc es pot utilitzar lliurement en comandes de SQL. Això significa que es pot escriure la sentència alineada diferentment, o fins i tot en una línia.

Dos guionets (" -- ") serveixen per introduir comentaris. Qualsevol cosa que els segueixi s'ignora fins al final de la línia.

Podem crear una taula nova especificant el nom de taula, junt amb tots els noms de columna i els seus tipus. Crearem una taula d'exemple que l'anomenarem 'Safaris':

```
CREATE TABLE Safaris
(codi_safari char(10),
 data_inici date not null,
 data_final date not null,
 codi_guia char(5) not null,
 primary key(codi_safari),
 foreign key(codi_guia) references
Guies(codi_guia));
```

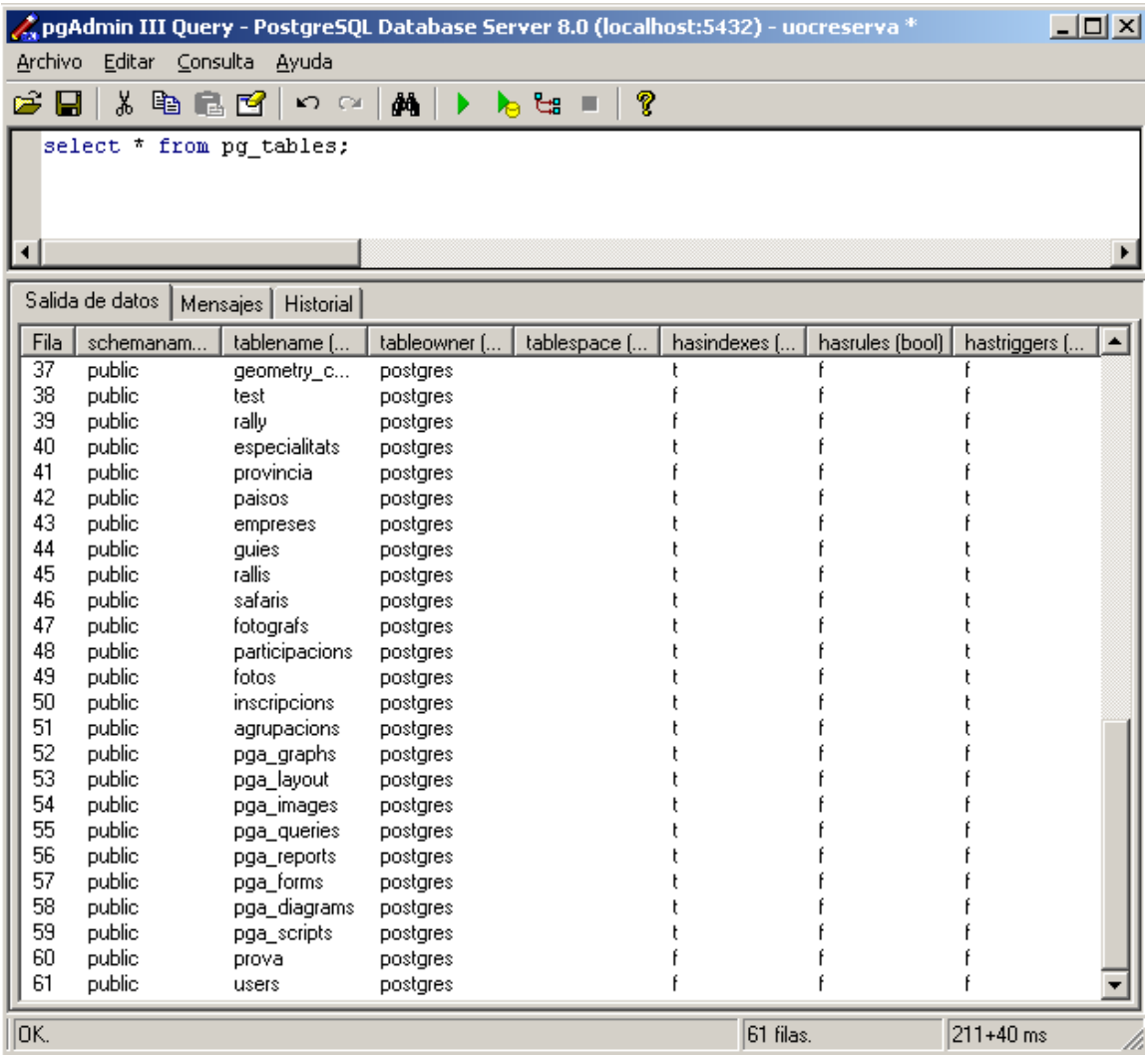
Veure en l'annex la resta d'implementació d'aquesta base de dades.

Si consultem en la vista del sistema pg_tables podrem veure les taules creades, entre elles les de UOCReserva.

Fem la consulta següent:

```
SELECT * FROM pg_tables;
```

Ens surten totes les taules existents i també les nostres amb la informació de l'esquema a que pertanyen, de l'usuari propietari, de l'espai de taules, si tenen taula d'índex, etc. Com es mostra en l'exemple d'executar aquesta consulta:



Com es pot veure apareixen totes les taules de la base de dades que estem utilitzant d'exemple: UOCReserva, a part de les del sistema.

Finalment, s'hauria d'esmentar que si ja no necessitem una taula la podem eliminar utilitzant la sintaxi següent:

```
DROP TABLE nom_taula [, ...] [CASCADE | RESTRICT]
```

Descripció:

Amb `CASCADE`, eliminem els objectes que depenen de la taula.

Amb `RESTRICT` no s'elimina la taula si hi ha objectes que depenguin d'ella. És l'opció per omissió.

Aquesta comanda s'ajusta a l'SQL estàndard, excepte que l'estàndard sols permet eliminar una taula per comanda.

Si volem eliminar-la taula Safaris del nostre exemple ho farem amb la comanda següent:

```
DROP TABLE Safaris;
```

4.1.- Restriccions de columna i de taula.

En el moment de crear una taula es poden definir les restriccions de taula següents:

```
[ CONSTRAINT nom_restriccions]

{ UNIQUE ( nom_columna [, ... ] ) [ USING INDEX TABLESPACE
espai_de_taules ] |
  PRIMARY KEY (nom_columna [, ... ] ) [ USING INDEX TABLESPACE
espai_de_taules ] |
  CHECK ( expressió ) |
  FOREIGN KEY (nom_columna [, ... ] ) REFERENCES ref_taula [ (
ref_columna [, ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ] [ ON DELETE
action ] [ ON UPDATE action ] }

[ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY
IMMEDIATE ]
```

En el moment de crear una taula també es poden definir les restriccions de les columnes amb: `{nom_columna tipus [DEFAULT expressió_defecte ] [restriccions_columna , ... ]...}`

Aquestes restriccions poden tenir les opcions següents:

```
[ CONSTRAINT nom_restriccions]

 { UNIQUE ( nom_columna [, ... ] ) |
   PRIMARY KEY (nom_columna [, ... ] ) |
   CHECK ( expressió ) |
   FOREIGN KEY (nom_columna [, ... ] ) REFERENCES ref_taula [ (
ref_column [, ... ] ) ]
     [ MATCH FULL | MATCH PARTIAL ] [ ON DELETE acció ] [ ON
UPDATE acció ] }
```

```
 [ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY
IMMEDIATE ]
```

Una restricció de taula és una restricció d'integritat definida sobre un o més camps d'una taula base. Les quatre variacions de restriccions de taula són:

```
UNIQUE
CHECK
PRIMARY KEY
FOREIGN KEY
```

Descripció:

UNIQUE. Aquesta restricció especifica que un grup d'una o més columnes d'una taula pot contenir sols valors únics.

CHECK. Les condicions que ha de complir. Una expressió d'un CHECK no pot contenir subconsultes ni referir-se a variables diferents de les de les columnes de la fila actual. Aquesta clàusula especifica una expressió que produeix un resultat booleà que les files noves o actualitzades han de satisfer per executar una operació d'inserció o d'actualització. Actualment, les expressions de la clàusula de CHECK especifica una expressió que produeix un resultat booleà que les files noves o actualitzades han de satisfer per executar una operació d'inserció o actualització no poden contenir subconsultes ni referir-se a variables diferents que les columnes de la fila actual.

PRIMARY KEY. Aquesta restricció especifica que aquest camp és una clau primària. Vol dir que una columna o columnes d'una taula poden contenir sols valors no repetits i no nuls.

FOREIGN KEY. Aquesta clàusula especifica una restricció de clau forana. Un valor introduït en aquesta columna es compara amb els valors de la taula referenciada i columnes referenciades que utilitzen el tipus de comparació (MATCH) donada.

Naturalment, el nombre i tipus de les columnes amb restriccions necessiten lligar amb el nombre i tipus de les columnes referenciades. Una taula pot contenir més d'una restricció de FOREIGN KEY. Això s'utilitza per implementar relacions de molts a molts entre taules.

Si cal assegurar que ningú pugui inserir les files en una taula que no tenen una entrada aparellada en un altre, diem que cal mantenir la integritat de referència entre les dues taules.

Cal utilitzar REFERENCES i la comprovació la fa PostgreSQL.

Hi ha tres tipus de MATCH: MATCH FULL, MATCH PARTIAL i MATCH SIMPLE, que ho és per omissió. MATCH FULL, no permetrà una columna d'ulna clau forana multicolumna sigui nul·la llevat que totes les columnes de la clau forana siguin nul·les. MATCH PARTIAL permet a algunes columnes de la clau forana estar nul·les mentre unes altres parts de la

clau forana no ho estiguin. MATCH SIMPLE encara no ha estat implementat en la darrera versió estudiada.

Se sap que les claus foranes rebutgen la creació de registres que no estiguin referenciats. Però i si cal treure un registre després que de que es creï un altre que el referència? PostgreSQL ens dona dues opcions: ON DELETE RESTRICT i ON DELETE CASCADE.

Aquests dos 'DELETE' són les dues opcions més comunes. RESTRICT evita la supressió d'una fila referenciada. Amb NO ACTION si les files de referència encara existeixen quan la restricció es comprova, deixa que la comprovació en la transacció sigui ajornada fins més tard, mentre que RESTRICT no ho permet.

CASCADE especifica que quan se suprimeix una fila referenciada, la referència s'hauria de suprimir automàticament també. Hi ha dues altres opcions: SET NULL i SET DEFAULT. Aquests provoquen que les columnes de referència siguin posades a nuls o valors per omissió, respectivament, quan se suprimeix la fila referenciada. Aquests no ens lliuren d'observar restriccions.

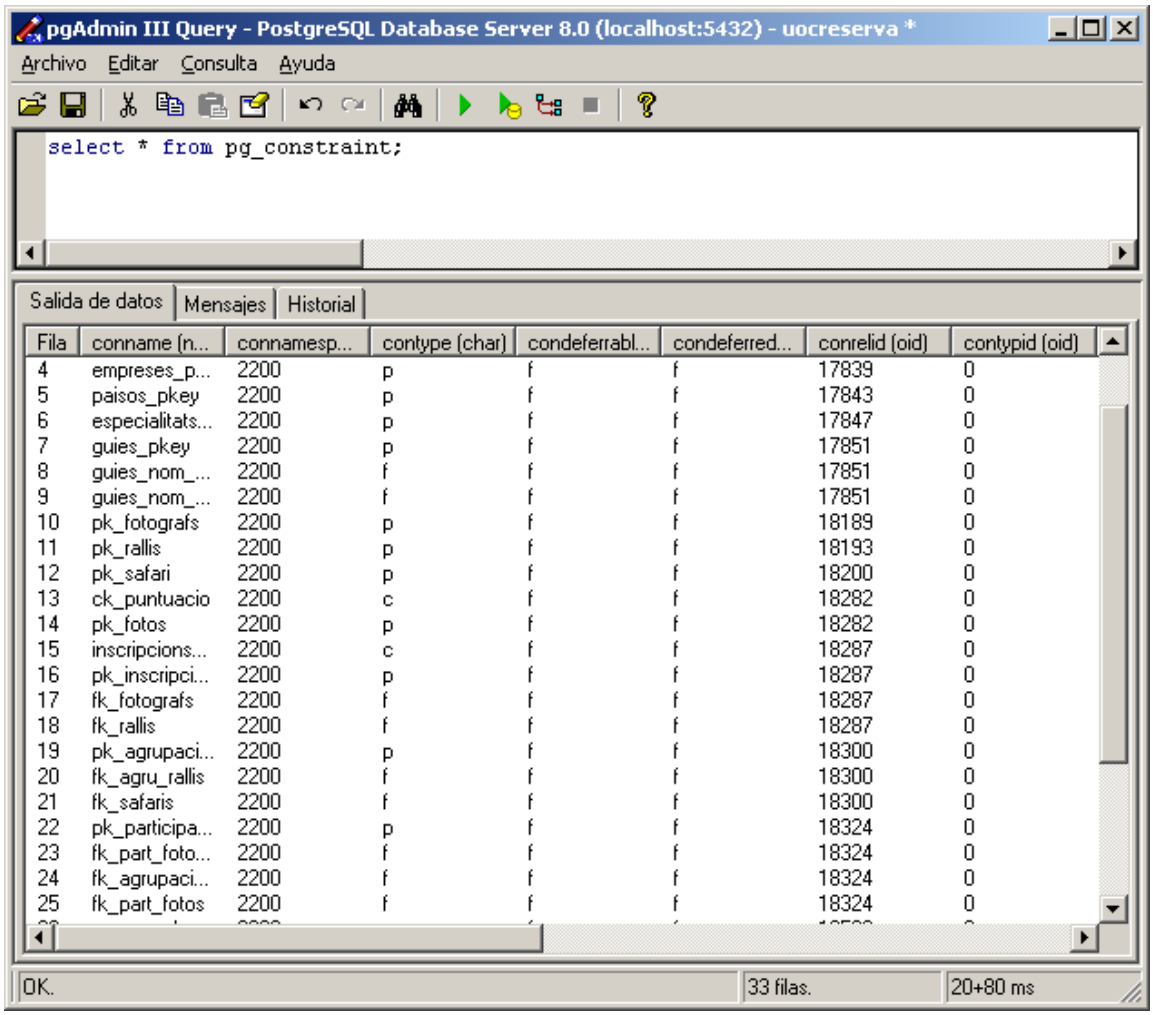
DEFERRABLE i NOT DEFERRABLE. Controla si la restricció es pot ajornar o no.

INITIALLY DEFERRED i INITIALLY IMMEDIATE. Si una restricció és DEFERRABLE, aquesta clàusula especifica el temps d'omissió per comprovar la restricció. Si la coacció és INITIALLY IMMEDIATE, s'atura després de cada declaració. Si la restricció és INITIALLY DEFERRED, s'atura només al final de la transacció.

En la nostra base de dades d'exemple hem creat les taules amb algunes restriccions. En l'annex corresponent a BDII es poden veure uns quants exemples. En el següent hem utilitzat restriccions de clau primària, restriccions de clau forana i restricció amb CHECK:

```
CREATE TABLE Inscripcions(
    passaport  CHAR(10),
    codi_ralli CHAR(10),
    descompte  INTEGER NOT NULL,
    CONSTRAINT pk_inscripcions PRIMARY KEY (passaport, codi_ralli),
    CONSTRAINT fk_fotografs FOREIGN KEY (passaport)REFERENCES
Fotografs(passaport),
    CONSTRAINT fk_rallis FOREIGN KEY (codi_ralli)  REFERENCES
Rallis(codi ralli),
    CHECK (descompte BETWEEN 0 AND 75)
);
```

Podem veure les restriccions creades consultant la vista del sistema pg_constraint:



4.2.- Modificacions i esborrat de taula.

Es poden canviar les definicions de les taules ja creades amb la sentència 'ALTER'. ALTER permet modificar, reanomenar, o afegir columnes a una taula existent. Addicionalment, la mateixa taula es pot reanomenar utilitzant la sintaxi 'ALTER TABLE... RENAME'. Si una taula o columna es reanomena, cap de les dades subjacents no s'afectarà.

Aquesta és la sintaxi per la modificació d'una taula:

```
ALTER TABLE [ ONLY ] nom [ * ]
    accions [, ... ]

ALTER TABLE [ ONLY ] nom [ * ]
    RENAME [ COLUMN ] columna TO nova_columna
```

```
ALTER TABLE nom
    RENAME TO nou_nom
```

I les accions poden ser una de les següents:

```
ADD [ COLUMN ] columna tipus [ restriccions_columna [ ... ] ]

DROP [ COLUMN ] columna [ RESTRICT | CASCADE ]

ALTER [ COLUMN ] columna TYPE tipus [ USING expressió ]

ALTER [ COLUMN ] columna SET DEFAULT expressió

ALTER [ COLUMN ] columna DROP DEFAULT

ALTER [ COLUMN ] columna { SET | DROP } NOT NULL

ALTER [ COLUMN ] columna SET STATISTICS integer

ALTER [ COLUMN ] columna SET STORAGE { PLAIN | EXTERNAL | EXTENDED
| MAIN }

ADD restricció_taula

DROP CONSTRAINT nom_restricció [ RESTRICT | CASCADE ]

CLUSTER ON nom_índex

SET WITHOUT CLUSTER

SET WITHOUT OIDS

OWNER TO nou_propietari

SET TABLESPACE nom_espai_taules
```

Descripció:

`ADD COLUMN`. Afegeix una columna a la taula.

`DROP COLUMN`. Elimina una columna d'una taula. Els índexs i les restriccions de taula que afecten a la columna també s'eliminaran automàticament. Cal afegir la clàusula `CASCADE` si hi ha alguna cosa més de fora de la taula que depengui de la columna, per exemple, referències a claus foranes o vistes, així tot el que referenciï a la columna també s'esborrarà. Amb la clàusula `RESTRICT` la columna si està referenciada no s'esborrarà.

`ALTER COLUMN TYPE`: Canvia el tipus d'una columna de la taula. USING si no està implícit un càsting del anterior al nou tipus.

`SET/DROP DEFAULT`. Per posar o treure el valor per defecte en una columna.

`SET/DROP NOT NULL`. Per admetre o no valors nuls.

`SET STATISTICS`. Aquesta clàusula permet modificar el valor del recull d'estadístiques per columna per les properes operacions d'anàlisi (`ANALYZE`). El valor pot estar entre 0 i 1000. Si el que es vol és utilitzar el valor de les estadístiques per defecte del sistema (*default_statistics_target*), cal posar-ho a -1.

`SET STORAGE`. Per posar la forma d'emmagatzemament d'una columna. Per exemple si aquesta columna en la taula o en una taula suplementària, i si les dades s'haurien de comprimir o no.

`CLUSTER`. selecciona l'índex per omissió per a futurs reagrupament d'operacions. Amb `SET WITHOUT CLUSTER` elimina l'especificació del índex del `CLUSTER` més recentment utilitzat a partir d'aquesta taula. Això afecta a les properes operacions `CLUSTER` que no especifiquen índex.

`SET WITHOUT OIDS`. Elimina el sistema OIDs de la taula.

`OWNER`. Canvia el propietari de la taula.

`SET TABLESPACE`. Per canviar l'espai de taules assignat i moure el fitxer de dades associat a la taula a un nou espai de taules.

`RENAME`. Per canviar el nom de la taula.

Afegir una restricció `CHECK` o `NOT NULL` exigeix mirar la taula per verificar que les files existents trobin la restricció.

Seguint amb l'exemple de la base de dades UOCReserva, si observem que ens cal una informació diferent en una determinada taula de la que tenim, podem fer una modificació. Per exemple en la taula 'Rallis' tenim les columnes següents:

- Codi_rally
- Preu
- Data_inici
- Data_fi

I ens diuen que la data d'inici i de fi no ens fan falta però si que ens cal el nom del 'rally'. Per tant farem les modificacions següents de la taula:

```
ALTER TABLE Rallis
      DROP COLUMN data_inici;

ALTER TABLE Rallis
```

```
      DROP COLUMN data_final;

ALTER TABLE Rallis
      ADD COLUMN nom CHAR (40) ;
```

Així la taula ens quedaria amb les següents columnes:

- Codi_rally
- Nom
- Preu

La sintaxi per esborrar una taula és la següent:

```
DROP TABLE name [, ...] [ CASCADE | RESTRICT ]
```

Amb `CASCADE` elimina els objectes depenent de la taula tals com les vistes.
Amb `RESTRICT` no elimina la taula si hi ha objectes de depenen d'ella. És l'opció per defecte.

4.3.- Tipus de dades.

Encara que es poden escriure pàgines amb aquest tema direm que PostgreSQL dóna suport al tipus '*int*' d'SQL estàndard, *smallint*, *precisió doble*, *real*, *char*(N), *varchar*(N), *data*, *time*, *timestamp*, i *interval*, així com uns altres tipus d'utilitat general i un conjunt ric de tipus geomètrics. El tipus '*point*' (punt geomètric en un plano de 2D) és un exemple d'un tipus de dades específic de PostgreSQL.

A més PostgreSQL es pot personalitzar amb un nombre arbitrari de tipus de dades definits per l'usuari. Conseqüentment, els noms de tipus no són *keywords* sintàctics. Amb l'expressió `CREATE TYPE` es permet a l'usuari registrar un nou tipus de dades d'usuari per ser usat en la base de dades actual. L'usuari que defineix un tipus es converteix en el seu propietari i el nom del nou tipus ha de ser únic dins dels tipus definits per a aquesta base de dades.

Es pot destacar també els OIDs, identificadors d'objecte que són utilitzats internament per PostgreSQL com claus primàries per a diverses taules del sistema. Una columna de sistema d'OID també s'afegeix a taules creades per l'usuari, llevat que s'especifiqui `WITHOUT OIDS` quan es crea la taula, o la variable de configuració `default_with_oids` (que és un *boolean*) es posi a fals. El tipus OID representa un identificador d'objecte.

Tipus de dades en PostgreSQL:

Tipus PostgreSQL	**Tipus SQL estàndard**	**Descripció**
Bool	*boolean*	Boolean lògic (cert/fals)
Box		Capça rectangular en plano de 2D
Char(n)	*Character(n)*	Seqüència de caràcters de longitud fixa
Cidr		IP versió 4 de xarxa o adreça del host
Circle		Cercle en plano de 2D
Date	*date*	Data sense hora
Decimal	*Decimal(p,s)*	Numero exacte per p<=9, s=0
Float4/8	*Float(p)*	Numero de coma flotant amb precisió p
Float8	*Real, double precision*	Numero de coma flotant amb doble precisió
Inet		IP versió 4 de xarxa o adreça del host.
Int2	*Smallint*	Enter de 2 bytes
Int4	*Int, integer*	Enter de 4 bytes
Int8		Enter de 8 bytes
Line		Línia infinita en un plano de 2D
Lseg		Segment de línia en un plano de 2D
Money	*Decimal(9,2)*	Numero fins a 9 enters i 2 decimals
Numeric	*Numeric(p,s)*	Numero exacte amb p=9, s=0
Path		Trajectòria geomètrica en plano de 2D
Point		Punt geomètric en plano de 2D
Polygon		Trajectòria geomètrica tancada en plano de 2D
Serial		identificació única per la indexació de direccions i la remissió
Time	*Time*	Hora del dia
Timespan	*Interval*	durada
timestamp	*Timestamps with time zone*	Dia/hora
Varchar(n)	*Character varying(n)*	Seqüència de caràcters de longitud variable.

5.- Assercions, esquemes i dominis.

El `CREATE TABLE` coincideix amb l'SQL estàndard, amb algunes excepcions.

Una excepció són les assercions ja que PostgreSQL en la darrera versió encara no implementa assercions.

Una asserció és un tipus especial de restricció d'integritat i comparteix el mateix espai de noms que unes altres restriccions. Tanmateix, una asserció no és necessàriament dependent d'una taula particular com ho són les altres restriccions, així SQL estàndard proporciona la declaració `CREATE ASSERTION` com a mètode alternatiu per definir una restricció: `CREATE ASSERTION nom CHECK ( condició ).`

Podem crear un esquema amb la sentència `CREATE SCHEMA`. Un esquema és essencialment un espai lògic, conté objectes designats (taules, tipus de dades, funcions i operadors) dels quals els noms podrien ser els d'altres objectes existents en altres esquemes.

`CREATE SCHEMA` crea un nou esquema en la base de dades en curs. El nom de l'esquema ha de ser diferent del nom dels diferents esquemes existents en la base de dades.

La sintaxi és la següent:

```
CREATE SCHEMA nom_esquema [
AUTHORIZATION nom_usuari ] [
element_esquema [ ... ] ]
CREATE SCHEMA AUTHORIZATION nom_usuari [element_esquema [ ... ] ]
```

`element_esquema` és una instrucció SQL definint un objecte a crear dins de l'esquema. Actualment, únicament `CREATE TABLE`, `CREATE VIEW`, `CREATE SEQUENCE`, `CREATE TRIGGER` i `GRANT` són acceptades com clàusules dins de `CREATE SCHEMA`.

Les restriccions de domini es defineixen amb les sentències `CREATE DOMAIN` o `ALTER DOMAIN.`

`CREATE DOMAIN` permet a l'usuari crear un domini de dades nou amb PostgreSQL per utilitzar amb la base de dades actual. L'usuari que defineix un domini es converteix en el seu propietari.

Si es dóna un nom d'esquema (per exemple, `CREATE DOMAIN el_meu_esquema.el meu_domini ...`) llavors el domini es crea en l'esquema especificat. Altrament es crea en l'esquema actual. El nom del domini ha de ser únic entre els tipus i dominis existents en el seu esquema.

Els dominis són útils per abstreure camps comuns entre taules en una localització única, sobre tot pel seu manteniment. Per exemple, una columna d'adreça d'e-mail es pot utilitzar en unes quantes taules, tots amb les mateixes propietats.

La sintaxi és la següent:

```
CREATE DOMAIN nom [AS] tipus_dada
    [ DEFAULT expressió ]
    [ restricció [ ... ] ]
```

I `restricció` pot ser:

```
[ CONSTRAINT nom_restricció ]
{ NOT NULL | NULL | CHECK (expressió) }
```

Un exemple d'un domini podria ser el telèfon i el podem crear en la BD d'exemple de la UOCReserva. Creem el domini '`d_telefon`' i en el moment de crear una taula l'utilitzem:

```
CREATE DOMAIN d_telefon char(15);

CREATE TABLE fotografs (passaport CHAR(10), nom CHAR (50), país
CHAR (20), telefon d_telefon);
```

Podem canviar la definició d'un domini existent amb `ALTER DOMAIN`. Hi ha unes quantes opcions i la sintaxi és la següent:

```
ALTER DOMAIN nom
    { SET DEFAULT expressió | DROP DEFAULT }

ALTER DOMAIN nom
    { SET | DROP } NOT NULL

ALTER DOMAIN nom
    ADD restricció_del_domini

ALTER DOMAIN nom
    DROP CONSTRAINT nom_restricció [ RESTRICT | CASCADE ]

ALTER DOMAIN nom
    OWNER TO nou_propietari
```

`DROP CONSTRAINT` elimina les restriccions d'un domini però combinat amb `RESTRICT` no procedeix a suprimir la restricció si hi ha diversos objectes dependents. És el comportament per defecte. Combinat amb `CASCADE` suprimeix automàticament els objectes dependents de la restricció.

I els dominis també es poden esborrar amb `DROP DOMAIN`. Només el propietari d'un domini el pot eliminar.

```
DROP DOMAIN nom [, ...]  [ CASCADE | RESTRICT ]
```

`CASCADE`. Esborra automàticament objectes que depenen del domini (com columnes de taula).

`RESTRICT`. No esborra el domini si en depenen objectes. És l'opció per defecte.

6.- Sentències de manipulació.

Un cop creada la base de dades i les seves taules cal que és pugui inserir, esborrar, consultar i modificar els valors de les files de les taules. Per aquestes accions PostgreSQL té les mateixes sentències que l'SQL estàndard amb algunes variants.

6.1.- Insercions.

Quan es crea una taula, no conté dades. La primera cosa per fer abans que una base de dades es pugui utilitzar és introduir dades. Les dades s'introdueixen conceptualment per files. Sempre s'ha de crear una fila completa. Cal tenir privilegis d'inserció en les taules per a afegir-hi files.

Les dades també es poden importar i en parlarem més endavant de com fer-ho.

Per inserir una nova fila s'utilitza la instrucció `INSERT`.

La seva sintaxi és:

```
INSERT INTO taula [ ( columna [, ...] ) ]
            { DEFAULT VALUES | VALUES ( {expressió | DEFAULT }  [,
...] ) | query }
```

Cada columna que no estigui present en la llista d'origen serà inserida usant el valor per defecte, que pot ser tant un valor per defecte declarat `DEFAULT` o bé `NULL`. PostgreSQL rebutjarà la nova columna si s'insereix un NULL en una columna declarada com `NOT NULL`.

Si l'expressió per a cada columna no és del tipus de dades correcte, s'intentarà un *cast* (conversió) de tipus automàticament.

Cal tenir el dret de fer INSERT sobre una taula per poder inserir-hi files.

Per exemple, seguint amb la base de dades i de la taula que hem creat, inserim files en la taula 'Inscripcions':

```
insert into Inscripcions values ('12894578','R2004R01',0);
insert into Inscripcions values ('09884323','R2004R01',25);
insert into Inscripcions values ('23663233','R2004R01',0);
insert into Inscripcions values ('23434388','R2004R01',0);
insert into Inscripcions values ('23434388','R2004C01',10);
insert into Inscripcions values ('59258743','R2004C01',0);
insert into Inscripcions values ('09884323','R2004C01',5);
insert into Inscripcions values ('12894578','R2004C01',0);
insert into Inscripcions values ('09884323','R2004R06',5);
```

```
insert into Inscripcions values ('56898743','R2004R06',0);
insert into Inscripcions values ('59258743','R2004R06',0);
insert into Inscripcions values ('32456654','R2004R06',0);
insert into Inscripcions values ('12894578','R2004R06',0);
insert into Inscripcions values ('56898743','R2004C06',20);
```

Es pot veure en l'annex la resta d'insercions en aquesta taula i en la resta de taules de la base de dades UOCReserva.

6.2.- Esborrats.

Igual que podem inserir també podem esborrar però sols files senceres d'una taula. En la inserció ja s'ha explicat que l'SQL no proporciona una manera de tractar directament files individuals. Per tant, esborrar files sols es pot fer especificant les condicions que han de complir les files que s'esborraran. Si hi ha una clau primària en la taula llavors es pot especificar la fila exacta. Però també es poden esborrar grups de files que compleixin una condició, o també, totes les files d'una taula.

S'utilitza `DELETE` per esborrar files d'una taula amb la sintaxi següent:

```
DELETE FROM [ONLY] taula [ WHERE condició ]
```

Amb aquesta sentència s'esborra les files que satisfan la clàusula WHERE de la taula especificada. Si la condició WHERE està absent, l'efecte és esborrar totes les files de la taula. El resultat és una taula vàlida, però buida.

Per omissió, `DELETE` suprimirà les files de la taula especificada i de totes les seves subtaules. Si desitgem suprimir només de la taula especificada cal utilitzar la clàusula `ONLY`.

Per exemple, i utilitzant sempre la base de dades que hem creat UOCReserva esborrem tots els guies que no són responsables de cap safari i que la seva especialitat és el 'Rastreig':

```
DELETE FROM Guies
WHERE codi_guia NOT IN (SELECT s.codi_guia
                        FROM Safaris s )
                         AND nom_especialitat = 'Rastreig';
```

PostgreSQL també ofereix `TRUNCATE` el qual és un mecanisme més ràpid per a esborrar totes les files d'una taula.

Sintaxi:

```
TRUNCATE [ TABLE ] nom
```

Té el mateix efecte que un `DELETE` però com no explora realment la taula és més ràpid. Aquest és més útil en taules grans.

Sols el propietari d'una taula la pot utilitzar. `TRUNCATE` no pot ser utilitzat si a la taula hi ha referències de la clau forana d'altres taules.

6.3.- Modificacions.

Per fer modificacions es pot utilitzar `UPDATE`. Aquest canvia el valor de les columnes especificades per totes les files que satisfan la condició que li donem. Sols cal indicar les columnes que seran modificades. Les condicions es donen amb `WHERE`.

Sintaxi:

```
UPDATE [ ONLY ] TABLE SET column = { expressió | DEFAULT } [, ...]
    [ FROM llista_expressions ]
    [ WHERE condició ]
```

On '`llista_expressions`' és una extensió no estàndard de PostgreSQL que permet l'aparició de columnes d'altres taules en la condició `WHERE`. Això és similar a la llista de taules que es pot especificar en la clàusula `FROM` d'una instrucció `SELECT`.

Per a referències a llistes s'usa la mateixa sintaxi de `SELECT`. O sigui, pot substituir un únic element d'una llista, un rang d'elements o una llista completa amb una única petició. Cal tenir permís d'escriure en la taula per a poder modificar-la, així com permís de lectura de qualsevol taula els valors de la qual siguin esmentats en la condició `WHERE`.

Per omissió, `UPDATE` modificarà les files de la taula especificada i de totes les seves subtaules. Si desitgem modificar només les de la taula especificada cal utilitzar la clàusula `ONLY`.

Un exemple de la base de dades UOCReserva. Si es vol aplicar una rebaixa d'un 10% al preu dels ral·lis que tenen més de 4 fotògrafs inscrits:

```
UPDATE Rallis
SET preu=preu-0.10*preu
WHERE codi_ralli IN (SELECT i.codi_ralli
```

```
                    FROM Inscripcions i
                GROUP BY i.codi_ralli
                  HAVING count(i.passaport) > 4);
```

6.4.- Consultes.

La sentència bàsica per fer una consulta sobre una taula és `SELECT .... FROM` i té el format següent:

```
SELECT [ ALL | DISTINCT [ ON ( expressió [, ...] ) ] ]
    * | expressió [ AS nom_sortida ] [, ...]
    [ FROM ítem [, ...] ]
    [ WHERE condició ]
    [ GROUP BY expressió [, ...] ]
    [ HAVING condició [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
    [ ORDER BY expressió [ ASC | DESC | USING operador ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start ]
    [ FOR UPDATE [ OF nom_taula [, ...] ] ]
```

`SELECT` retorna registres d'una o més taules. Els candidats a ser seleccionats són aquells registres que compleixen la condició especificada amb `WHERE`; si s'omet `WHERE`, es retornen tots els registres.

`ALL` o `DISTINCT` depenent de si el que es vol és que retorni totes les files candidates incloent duplicats `(ALL)` o que elimini duplicats `(DISTINCT)`. `DISTINCT ON` elimina les files que coincideixen amb totes les expressions especificades.

`DISTINCT ON` no és part d' SQL estàndard. Tampoc ho són `LIMIT` i `OFFSET`. Si s'especifica una d'aquestes dues clàusules la consulta sols retornarà un subconjunt de les files del resultat. Amb *count* s'especifica el nombre màxim de files a retornar, i *start* el nombre de files a saltar abans de començar a retornar files en el resultat.

En l'SQL estàndard, la paraula clau opcional "`AS`" és totalment prescindible i pot ser omesa sense afectar el significat. L'analitzador sintàctic de PostgreSQL requereix la presència d'aquesta paraula quan es reanomenen columnes a causa de les característiques d'extensibilitat de tipus que poden dur a interpretacions ambigües en aquest context.

La clàusula `FROM` pot contenir diversos elements i pot ser un dels següents:

```
[ONLY]nom_taula[*][[AS]àlies[(columna_àlies [,...])]]

    (SELECT)[AS]àlies[(columna_àlies[,...])]
```

```
    Nom_funció([argument[,...]])[AS]àlies[(columna_àlies[,...]|
columna_definició [,...])]

    Nom_funció([argument[,...]])AS(columna_definició[,...])
    ítem[NATURAL]tipus_combinació ítem[ON condició_combinació |
USING( columna_combinació[,...])]
```

En un exemple de `SELECT` de la nostra base de dades de model, cal donar el codi, la data d'inici i la data final dels ral·lis els quals, els safaris que el componen, han tingut assignat un guia amb una experiència menor a 5 anys però que mai han tingut assignat al guia 'John Walker' el qual té una experiència d'un any. Utilitza la clàusula `NOT EXISTS`.

```
SELECT DISTINCT r.codi_ralli, r.data_inici, r.data_final
FROM rallis r,  safaris s, agrupacions a, guies g
          WHERE  r.codi_ralli = a.codi_ralli and a.codi_safari=
s.codi_safari and
s.codi_guia=g.codi_guia and g.anys_antiguitat < 5
           AND NOT EXISTS IN
              (SELECT * FROM guies g2, agrupacions a2, safaris s2
               WHERE a2.codi_ralli = r.codi_ralli AND
                a2.codi_safari= s2.codi_safari AND
                 s2.codi_guia=g2.codi_guia AND
                   g2.nom = 'John Walker');
```

El `tipus_combinació (JOIN)` pot ser un dels següents:

```
-  [ INNER ] JOIN
-  LEFT [ OUTER ] JOIN
-  RIGHT [ OUTER ] JOIN
-  FULL [ OUTER ] JOIN
-  CROSS JOIN
```

Un exemple d'una consulta sobre la UOCReserva per saber el nom del fotògraf que va fer la foto amb el registre X, suposant que a la taula Fotos sols consta el registre de la foto i el DNI del fotògraf:

```
SELECT Fotos.registre, Fotograf.nom, Fotograf.Dni
FROM Fotos INNER JOIN Fotograf
ON Fotos.Fotograf = Fotograf.Dni
WHERE Fotos.registre = 'X';
```

El resultat serà el registre de la foto amb el nom i el DNI del fotògraf.

Amb els operadors `UNION` , `INTERSECT`, i `EXCEPT`, la sortida de més d'una declaració `SELECT` es pot combinar per a formar un sol resultat.

L'operador de `UNION` torna totes les files que estiguin en una o ambdós resultats.

La seva sintaxi és la següent:

```
taula_query UNION [ ALL ] taula_query
[ ORDER BY columna [ ASC | DESC ] [, ...] ]
```

on `taula_query` especifica qualsevol expressió `SELECT` sense la clàusula `ORDER BY`.

Per exemple, suposem que tenim una taula en la nostre UOCReserva amb els treballadors i en tenim un altre amb els guies. Si volem consultar el cognom de tots els guies i treballadors que es diguin 'Joan' de la nostre UOCReserva farem:

```
SELECT treballadors.cognom
     FROM treballadors
     WHERE treballadors.nom LIKE 'Joan'
UNION
SELECT guies.cognom
     FROM guies
     WHERE guies.nom LIKE 'Joan'
```

L'operador `INTERSECT` dóna els registres comuns a dues consultes. Els dos `SELECT` que representen els operands directes de la intersecció han de produir el mateix nombre de columnes, i les columnes corresponents cal que siguin tipus de dades compatibles.

Si s'utilitzen diversos operadors `INTERSECT` en la mateixa declaració `SELECT` se avaluen d'esquerra a dreta. Per modificar-ho cal utilitzar parèntesis.

La sintaxi és la següent:

```
taula_query INTERSECT taula_query
[ ORDER BY columna [ ASC | DESC ] [, ...] ]
```

on `taula_query` especifica qualsevol expressió `SELECT` sense la clàusula `ORDER BY`.

Si el que volem és consultar tots els treballadors que es diguin Joan i que el seu sou sigui més gran de 1000, de la nostre UOCReserva farem:

```
SELECT * FROM treballadors
     WHERE sou > 1000
```

```
INTERSECT
SELECT * FROM treballadors
     WHERE nom LIKE 'Joan'
```

L'operador `EXCEPT` dóna els registres tornats per la primera consulta però no per la segona. Els dos `SELECT` que representen els operands directes de la intersecció han de produir el mateix nombre de columnes, i les columnes corresponents cal que siguin de tipus de dades compatibles.

Si s'utilitzen diversos operadors `EXCEPT` en la mateixa declaració `SELECT` s'avaluen d'esquerra a dreta, a menys que se utilitzin parèntesis para modificar això.

La seva sintaxi és:

```
taula_query EXCEPT taula_query
[ ORDER BY columna [ ASC | DESC ] [, ...] ]
```

on `taula_query`, igual que en el cas del `INTERSECT`, especifica qualsevol expressió `SELECT` sense la clàusula `ORDER BY`.

En l'exemple anterior utilitzant `EXCEPT` ens donarà tots els treballadors amb un sou superior a 1000 però sense relacionar les dones.

```
SELECT * FROM treballadors
     WHERE sou > 1000
EXCEPT
SELECT * FROM treballadors
     WHERE sexe LIKE 'F'
```

7.- Importació/exportació de dades.

Ja s'ha explicat la forma d'inserir files en una taula amb l' `INSERT`, però si el que cal és carregar moltes dades de cop a una taula tenim la comanda `COPY`.

La comanda `COPY` pren dues formes:

```
COPY ... FROM
```

que llegeix dades d'un fitxer extern a una taula. I

```
COPY ... TO
```

que escriu el contingut d'una taula en un fitxer extern.

Els arxius que volem importar a les taules de PostgreSQL cal que estiguin en format ASCII estàndard, els seus camps delimitats per un mateix símbol, o en format binari de taula de PostgreSQL. Els delimitadors més comuns per arxius ASCII són els tabuladors i les comes.

Quan utilitzem un fitxer d'entrada amb format ASCII amb la comanda `COPY`, cada línia del fitxer serà tractada com una fila de dades a inserir i cada camp delimitat serà tractat com un valor d'una columna.

La comanda `COPY FROM` opera de manera molt més ràpida que una `INSERT` normal perquè les dades són llegides com una única transacció. Per altra part, és un format molt estricte, i sols que hi hagi una línia mal formada fallarà tot el procediment complert de `COPY`.

La sintaxi per utilitzar la comanda `COPY...FROM`, quan `nom_taula` és la taula a la que volem inserir els valors i `nom_arxiu` és la direcció complerta per accedir al fitxer que volem llegir:

```
COPY nom_taula [ ( columna [, ...] ) ]
    FROM { 'nom_fitxer' | STDIN }
    [ [ WITH ]
          [ BINARY ]
          [ OIDS ]
          [ DELIMITER [ AS ] 'tipus_delimitador' ]
          [ NULL [ AS ] 'null string' ]
          [ CSV [ QUOTE [ AS ] 'quote' ]
                [ ESCAPE [ AS ] 'escape' ]
                [ FORCE NOT NULL columna [, ...] ]
```

On el `tipus_delimitador` és el caràcter de separació de columnes. Per defecte és un tabulador en el mode de text, o una coma en mode `CSV` (*Selects Comma Separated Value* –Selecció de coma com a valor de separació).

Paràmetres:

`STDIN` i `STDOUT`, per especificar que el fitxer és d'entrada o de sortida.

`BINARY`, format en que les dades s'emmagatzemaran o es llegiran.

`OIDS`, especifica una còpia del OID en cada fila.

`CSV`, mode en que el calor de separació és la coma.

Si la comanda `COPY` s'utilitza sense les opcions `BINARY` o `CSV`, les dades llegides o escrites és un fitxer de text amb una línia per fila de la taula, on les columnes estan separades per un caràcter delimitador. El final de les dades es representa per una única línia contenent sols barres invertides (\.).

`COPY...FROM` pot reconèixer les següents barres invertides seguides d'un caràcter:

CARACTERS	DESCRIPCIÓ
\TAB	Utilitzat com delimitador per defecte
|	Utilitza una pipa com a delimitador
\N	Si s'utilitza NULL en la sortida per defecte
\b	Retrocés (ASCII 8)
\f	Salt de pàgina (ASCII 12)
\n	Nova línia (ASCII 10)
\r	Tecla de retorn (ASCII 13)
\t	Tabulador (ASCII 9)
\v	Tabulador vertical (ASCII 11)
\###	Caràcter representat per nombre octal ###
\\	Barra invertida

Un exemple senzill de d'utilització de la comanda `COPY ... FROM`, si volem poblar una taula 'especies' i tenim un fitxer (/tmp/especies.sql) en que els camps estiguin delimitats per comes (',') faríem:

```
COPY especies FROM 'tmp/especies.sql'
     WITH DELIMITER ',' WITH NULL AS '\null';
```

Es pot utilitzar la comanda `COPY` per inserir les files d'un arxiu binari. Utilitzant el mateix exemple faríem:

```
COPY especies FROM 'tmp/especies.sql'
     WITH BINARY;
```

Una sintaxi similar al `COPY...FROM` s'utilitza per enviar dades d'una taula a un fitxer amb el `COPY...TO`. Cal canviar el `stdin` per el `stdout`.

```
COPY nom_taula [ ( columna [, ...] ) ]
    TO { 'nom_arxiu' | STDOUT }
    [ [ WITH ]
          [ BINARY ]
          [ OIDS ]
          [ DELIMITER [ AS ] 'delimitador' ]
          [ NULL [ AS ] 'null string' ]
          [ CSV [ QUOTE [ AS ] 'quote' ]
                [ ESCAPE [ AS ] 'escape' ]
                [ FORCE QUOTE columna [, ...] ]
```

Si seguint amb la taula d'exemple volem copiar el contingut de la taula 'especies' a un fitxer amb el nom 'especies.txt' farem:

```
COPY especies TO 'especies.txt';
```

Si el `COPY...TO` s'utilitza amb la clàusula `BINARY`, PostgreSQL generarà l'arxiu resultant com un tipus d'arxiu binari. El format d'un arxiu binari és el següent:

PRINCIPI DE FITXER	
uint32	Nombre de instancies en el fitxer
TIPUS DE DADES	**DESCRIPCIÓ**
uint32	Longitud total de l'instancia de dades
uint32	Identificador (si s'especifica)
uint32	Nombre d'atributs nuls
[uint32, ..., uint32]	Nombre d'atributs des de 0.

-	<dades>

Uint32 = quantitats senceres de 4 bits sense signe.

La comanda `COPY` és pròpia de PostgreSQL i no existeix aquesta sentència en SQL estàndard.

`COPY` no invoca regles ni accions per defecte en les columnes però si pot invocar procediments. `COPY` atura les operacions en el primer error. Això no produeix problemes en el cas de `COPY FROM`, però en el cas de `COPY TO` si ja que el destí serà parcialment modificat. Aleshores es pot utilitzar `VACUUM` per netejar després d'una còpia fallida.

`VACUUM` recull les escombraries i opcionalment analitza una base de dades.

La seva sintaxi és la següent:

```
VACUUM [FULL | FREEZE][VERBOSE] [taula]
VACUUM [FULL | FREEZE][VERBOSE] ANALYZE[taula[(columna[,...] )]]
```

Paràmetres de `VACUUM`:

`FULL`, cal seleccionar "`FULL`" per reclamar més espai.

`FREEZE`, és una opció particular que declara les files com a 'congelades' tant aviat com pot. Si es fa en un moment en què no hi ha cap altra transacció oberta sobre la base de dades, llavors es garanteix que totes les línies d'aquesta BD estan 'congelades'. `FREEZE` no és recomanat en el ús normal. Només és destinat a formar part de la preparació de models de base de dades definits per l'usuari o per a altres bases de dades que són exclusivament en lectura i que no cal que tinguin un manteniment regular.

`VERBOSE`, imprimeix un reportatge detallat de l'activitat de `VACUUM` per cada taula.

`ANALYZE`, actualitza les estadístiques utilitzades per l'optimitzador per determinar la manera més eficaç d'executar una consulta. Les estadístiques representen la dispersió de les dades en cada columna. Aquesta informació és valuosa quan hi ha la possibilitat d'execució des de diversos punts.

8.- Components de control.

8.1.- Funcions.

Una funció és una agrupació de sentències que s'executa com una unitat. Són molt útils quan cal fer sovint manipulacions automatitzades de taules. Aquestes s'emmagatzemen a la base de dades.

En PostgreSQL una funció i un procediment emmagatzemat és exactament el mateix. La diferència és més conceptual que concreta.

Les funcions accepten uns valors d'entrada, realitzen alguna consulta o manipulació sobre ell, i tornen un valor de sortida.

PostgreSQl proporciona tres tipus de funcions:

- Funcions de llenguatge de consultes, escrites en SQL.
  Aquestes funcions executen una llista arbitraria de consultes SQL, tornant els resultats de la darrera consulta de la llista. Poden ser:
  - Funcions sobre tipus base: No té arguments i sols retorna un tipus base, com per exemple un int4.
  - Funcions sobre tipus compostos: A l'especificar funcions amb arguments de tipus compostos també cal especificar els atributs d'aquests arguments.

- Funcions de llenguatge procedural, escrites, per exemple en PLSQL.
- Funcions de llenguatge de programació, escrites en un llenguatge de programació compilat, com per exemple en C.

Un exemple d'una funció SQL sobre tipus base pot ser la següent:

```
CREATE FUNCTION suma(int4, int4) RETURNS int4
AS ' SELECT $1 + $2; 'LANGUAGE 'SQL';
```

Si fem la consulta d'aquesta funció passant dos números com a paràmetres, per exemple el 3 i el 7:

```
SELECT suma(3,7) AS resultat;
```

Ens torna:

```
Resultat
----------
```

```
10
```

Després veurem detalladament la sintaxi de la creació de les funcions però '$n' significa l'ordre dels paràmetres, '$1' el primer paràmetre, '$2' el segon, etc.

PostgreSQL disposa de diverses funcions predefinides que es poden consultar amb la comanda `\df` de psql, des de la consola. En la versió 8.0 n'hi ha 1784. La consulta ens informa de l'esquema a que pertany, el nom de la funció, el tipus de dades de sortida i el tipus de dades dels arguments. També és pot utilitzar per veure-ho d'una funció especifica, per veure la informació que ens dona demanem sols una concreta, per exemple de '`UPPER`'. Farem: `\df UPPER` i ens informa d'aquesta funció en concret:

```
                          Listado de funciones
   Schema   | Nombre | Tipo de dato de salida | Tipos de datos de argumentos
------------+--------+------------------------+------------------------------
 pg_catalog | upper  | text                   | text
```

Provem la funció i veiem que accepta una sèrie de caràcters, els converteix en majúscules i torna la nova sèrie:

```
SELECT UPPER('abcdef');
```

Torna: ABCDEF

Les funcions en PostgreSQL poden escriure's en diferents llenguatges, per exemple en C, SQL i PL/pgSQL. Es creen amb `CREATE FUNCTION` i s'eliminen amb `DROP FUNCTION`.

CREACIÓ DE FUNCIONS

`CREATE FUNCTION` està definit en SQL estàndard a partir d'SQL:1999. La versió de PostgreSQL és similar però no plenament compatible.

La sentència `CREATE FUNCTION` requereix, com a mínim:
- Un nom per la funció.
- El nombre d'arguments (paràmetres).
- El tipus de cada argument.
- El tipus de retorn de la funció.
- L'acció (el programa com a tal).
- El llenguatge que utilitza.

Estructura d'una sentència de creació d'una funció:

```
CREATE [ OR REPLACE ] FUNCTION nom ( [ [ arg_nom ] arg_tipus [,
...] ] )
```

```
    RETURNS ret_tipus
  { LANGUAGE lleng_nom
    | IMMUTABLE | STABLE | VOLATILE
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY
DEFINER
    | AS 'definició'
    | AS 'obj_arxiu', 'link_simbol'
  } ...
    [ WITH ( atribut [, ...] ) ]
```

Descripció:

La sentència `CREATE [ OR REPLACE ] FUNCTION` pot crear una funció nova o reemplaçar una funció ja existent. Cal tenir en compte que pot haver altres funcions amb el mateix nom però amb un tipus d'arguments diferents (sobrecàrrega). Per això amb `CREATE [ OR REPLACE ] FUNCTION` no es pot canviar el nom o els tipus d'argument d'una funció, ni el tipus de retorn d'una funció existent. Per fer-ho caldrà suprimir-la i tornar-la a crear.

`arg_nom` i `arg_tipus` són el nom i el tipus d'un argument. El tipus d'una columna s'utilitza escrivint `nom_taula.nom_columna%TYPE`. `%TYPE` proporciona el tipus de dades d'una columna d'una taula. Utilitzar aquesta funcionalitat ens pot ajudar a mantenir una funció depenent de les modificacions de la definició d'una taula.

`ret_tipus` és el tipus de retorn de la funció. El tipus d'una columna també s'utilitza escrivint `nom_taula.nom_columna%TYPE`.

`lleng_nom` és el nom del llenguatge en que la funció està implementada.

Si ens cal afegir un nou llenguatge a la base de dades de PostgreSQL es pot fer amb `CREATELANG` sempre i quan siguin llenguatges proporcionats en la distribució de PostgreSQL:

```
CREATELANG [opcions connexió...] lleng_nom [dbnom]
CREATELANG [opcions connexió...] --llista | -l dbnom
```

També es pot utilitzar `CREATE LANGUAGE:`

```
CREATE [ TRUSTED ] [ PROCEDURAL ] LANGUAGE nom
    HANDLER call_handler [ VALIDATOR valfunció ]
```

Encara que es recomana utilitzar `CREATELANG` perquè realitza un cert nombre de comprovacions i és molt més fàcil d'utilitzar.

Si fem una consulta dels llenguatges que venen en la darrera versió de PostgreSQL, la 8.0, fent un `SELECT` a la taula pg_language:

```
SELECT * FROM pg_language;
```

ens dona el resultat següent:

```
lanname  | lanispl | lanpltrusted | lanplcallfoid | lanvalidator |    lanacl
----------+---------+--------------+---------------+--------------+-------------
 internal | f       | f            |             0 |         2246 |
 c        | f       | f            |             0 |         2247 |
 plpgsql  | t       | t            |         17230 |        17231 |
 sql      | f       | t            |             0 |         2248 | {=U/postgres}
(4 filas)
```

`CREATE LANGUAGE` és una extensió de PostgreSQL i no existeix una sentència en SQL estàndard.

Podem veure que el llenguatge de programació de funcions per defecte de PostgreSQL, és el PL/pgSQL. El PL/pgSQL té estructura de blocs. La definició complerta d'una funció cal que sigui un bloc i aquest es defineix amb:

```
[]=opcional
[<<identificador o etiqueta>>]
[DECLARE declaracions]
BEGIN instruccions END;
```

Sintaxi de PL/pgSQL:

- Cada declaració i cada instrucció s'acaba amb un punt i coma (;).

Un bloc ennierat dintre d'un altre cal que tingui punt i coma després del `END`, encara que l' `END` final no ho requereix.

- Es poden utilitzar majúscules i minúscules sense distinció. Els identificadors es converteixen a minúscules implícitament a menys que s'utilitzin cometes dobles (").
- Els comentaris poden utilitzar (/*) i (*/) o una sola línia amb (--).

Sub-blocs (ennierats):

- Un bloc pot tenir blocs ennierats o sub-blocs.
- Els sub-blocs es poden utilitzar per agrupar instruccions o per declarar variables locals.
- Una variable declarada en un sub-bloc amb un valor per defecte s'inicialitza cada vegada que se entra al bloc, no una vegada per funció.

Declaració de variables:

- Totes les variables cal que siguin declarades. La única excepció és la variable de control d'un `'FOR'` que és `'integer'` automàticament.

- Les variables poden ser de qualsevol tipus d'SQL, tal com `integer`, `varchar`, o `char`.

`IMMUTABLE, STABLE, VOLATILE.` Aquests atributs informen al sistema si cal reemplaçar vàries avaluacions de la funció amb una sola avaluació per una optimització en l'execució. Si no hi ha cap opció, es pren per defecte `VOLATILE`.

- `IMMUTABLE` per indicar que la funció envia sempre el mateix resultat quan rep els mateixos valors en els arguments.

- `STABLE` per indicar que per un sol recorregut de la taula, la funció enviarà el mateix resultat pels mateixos valors de l'argument, però el resultat pot variar per mitjà de les instruccions SQL.

- `VOLATILE` per indicar que el valor de la funció pot canviar amb un sol recorregut de la taula, ja que no es pot realitzar cap optimització.

`CALLED ON NULL INPUT,` és el valor per omissió i indica que la funció serà cridada normalment quan alguns dels seus arguments siguin nuls.

`RETURNS NULL ON NULL INPUT, STRICT.` Indiquen que la funció envia sempre un NULL si un dels seus arguments és un NULL. Si s'especifica aquest paràmetre la funció no s'executa quan hi ha arguments a NULL i s'envia un resultat NULL automàticament.

`[EXTERNAL ] SECURITY INVOKER, [ EXTERNAL ] SECURITY DEFINER. SECURITY INVOKER` és l'opció per defecte i indica que la funció s'ha d'executar amb els drets de l'usuari que la crida. `SECURITY DEFINER` indica que la funció s'ha d'executar amb els drets de l'usuari que la va crear.

A `'definició'` és on es defineix la funció; el significat depèn del llenguatge. Pot ser un nom de funció interna, el camí cap un fitxer objecte, una sentència SQL o un text en un llenguatge de procediments.

Seguint amb el llenguatge PL/pgSQL, tenim:

Declaracions.

Totes les variables, files i columnes que s'utilitzen en un bloc o subbloc cal que siguin declarades. Els paràmetres donats a funció PL/pgSQL es declaren automàticament amb els identificadors $n.

Tipus de dades.

Els tipus d'una variable poden ser qualsevol dels tipus bàsics existents en la base de dades. `type` en la secció de declaració es defineix com:

- `variable%TYPE`
- `class.field%TYPE`

`class` és el nom d'una taula o vista existent, i `field` és el nom d'un atribut.

Expressions.

Totes les expressions en les sentencies PL/pgSQL són processades utilitzant *backends* d'execució. Les expressions que poden contenir constants poden requerir avaluació en temps d'execució ja que és impossible que l'analitzador de PL/pgSQL identifiqui els valors constants diferents de la paraula clau `NULL`. Totes les expressions s'avaluen internament executant una consulta.

```
SELECT expressió
```

Sentencies.

Qualsevol cosa no compresa per l'analitzador PL/pgSQL tal com s'ha especificat serà enviat al gestor de base de dades per la serva execució. La consulta resultant no tornarà cap dada.

Assignacions.

Una assignació d'un valor a una variable o camp de fila o de registre s'escriu:

```
Variable:= expressió;
```

L'expressió s'avalua mitjançant un `SELECT` d'SQL.

Crides a un altre funció.

Totes les funcions definides en una base de dades Postgres tornen un valor. Per tant, la forma normal de cridar a un funció és executar una consulta `SELECT` o realitzar una assignació (que doni lloc a un `SELECT` intern de PL/pgSQL).

Estructures de control

Condicionals.

Aquestes tenen la forma següent:

```
IF expressió THEN
      Sentències
[ELSE
      Sentències]
END IF;
```

`Expressió` cal que retorni un valor que al menys pugui ser adaptat en un tipus booleà.

Bucles.

Hi ha diversos tipus de bucles. Per exemple:

```
[<<label>>]
LOOP
      sentències
END LOOP;
```

Aquest és un bucle no condicional que cal que s'acabi de forma explicita, amb una sentència `EXIT` d'altres bucles ennierats.

```
[<<label>>]
WHILE expressió LOOP
      sentències
END LOOP;
```

És un bucle condicional que s'executa mentre l'avaluació d' `expressió` sigui certa.

```
[<<label>>]
FOR nom IN [reverse]
 Expressió .. expressió LOOP
      sentències
END LOOP;
```

És un bucle que se itera sobre un rang de valors enters. La variable `nom` es crea automàticament amb el tipus enter, i existeix sols dintre el bucle. Les dos expressions donen el límit inferior i superior del rang i son avaluades sols quan s'entra en el bucle. El pas de la iteració és sempre 1.

```
[<<label>>]
FOR record | row IN select_clàusula LOOP
      sentències
END LOOP;
```

El registre o fila s'assigna a totes les files resultants de la clàusula de selecció, i la sentència s'executa per cada una d'elles. Si el bucle s'acaba amb una sentència `EXIT`, la darrera fila assignada és encara accessible després del bucle.

```
EXIT [label ] [WHEN expressió];
```

Cursors

A més d'executar una consulta complerta també és possible configurar un cursor que encapsuli la consulta, i després llegir el resultat d'unes quantes files. Els bucles `FOR` automàticament utilitzen un cursor internament per evitar problemes de memòria.

L'accés als cursors és a través de variables cursor , que són del tipus de dades `refcursor`. Es pot crear una variable de tipus cursor declarant-la com de tipus `refcursor` o utilitzar la sintaxi de declaració de cursor següent:

```
Nom CURSOR [(ARGUMENTS)] for SELECT_QUERY;
```

Abans que un cursor es pugui utilitzar per retornar files cal que s'obri. Hi ha formes d'obrir un cursor utilitzant variables cursor no assignades i utilitzant variables cursor assignades.

```
OPEN unbound-cursor FOR SELECT...;
OPEN unbound-cursor FOR EXECUTE query-string;
OPEN bound-cursor [(argument_values)];
```

Una vegada obert el cursor es pot manipular.

`FETCH` retorna la següent fila des del cursor a una destinació que pot ser una variable fila, registre o una llista de variables simples, separades per comes, igual com en un `SELECT INTO`:

```
FECH cursor INTO target;
```

`CLOSE` tanca el cursor obert. Es pot utilitzar per alliberar recursos abans del final d'una transacció o per alliberar la variable cursor per tornar-la a obrir posteriorment:

```
CLOSE cursor;
```

Errors i missatges

PostgreSQL no disposa d'un model molt acurat de tractament d'excepcions. Quan l'analitzador, l'optimitzador o l'executor decideixen que una sentència no es pot processar,

la transacció complerta s'avorta i el sistema torna per processar la següent consulta de l'aplicació.

L'única cosa que fa PL/pgSQL quan es produeix un avortament d'execució durant l'execució d'una funció o disparador es enviar missatges de depuració al nivell `DEBUG`, indicant en quina funció i on (numero de línia i tipus de sentència) ha succeït l'error.

Es pot utilitzar la sentència `RAISE` per enviar missatges:

```
RAISE level 'format' [,identificador[...]];
```

On `level` pot ser, per exemple, `NOTICE` o `EXCEPTION`. `NOTICE` escriu en la bitàcola de la base da dades i ho envia a l'aplicació del client. `EXCEPTION` escriu en la bitàcola de la bases de dades i avorta la transacció.

MODIFICACIÓ I ELIMINACIÓ DE FUNCIONS

Una vegada creada una funció podem modificar la seva definició de nom i propietat amb les següents sentències:

```
ALTER FUNCTION name ( [ tipus [, ...] ] ) RENAME TO nou_nom
```

i

```
ALTER FUNCTION name ( [tipus [, ...] ] ) OWNER TO nou_propietari
```

També la podem eliminar amb un `DROP FUNCTION`:

```
DROP FUNCTION name ( [tipus [, ...] ] ) [ CASCADE | RESTRICT ]
```

`DROP FUNCTION` elimina la definició d'una funció existent. Per executar aquesta comanda l'usuari ha de ser el propietari de la funció. Cal especificar els tipus d'arguments a la funció ja que poden existir unes quantes funcions diferents amb el mateix nom i llistes d'arguments diferents.

Si posem `CASCADE` elimina també els objectes dependents de la funció.

Amb `RESTRICT` no elimina la funció si hi ha objectes dependents d'ella. És l'opció per defecte.

Per eliminar la funció que acabem de crear faríem:

```
DROP FUNCTION reg_empleat;
```

EXEMPLE DE FUNCIÓ

Seguint amb la nostra base de dades UOCReserva. Exposem dos exemples. El primer molt senzill, és una funció auxiliar per obtenir el nombre de safaris que conté un ralli.

```
CREATE FUNCTION safarisInRally (rally_code CHAR) RETURNS int AS '
    BEGIN
    RETURN (SELECT COUNT(*) FROM Agrupacions WHERE codi_ralli =
rally_code);
    END;
' LANGUAGE 'plpgsql'
```

Ara cal fer la consulta per un determinat codi de ralli:

```
SELECT safarisInRally ('R2005-T1-1');
```

i ens retorna:

```
safarisInRally
--------------
2
```

En l'annex podem veure altres exemples més elaborats de creació de funcions en la pràctica de BDII utilitzant diferents estructures de control i cursors.

8.2.- Disparadors.

Els disparadors (*triggers*) són procediments emmagatzemats a la base de dades que s'executen automàticament quan es du a terme una operació `INSERT`, `DELETE` o `UPDATE` sobre alguna taula en concret i permeten als usuaris executar funcions introduïdes per altres usuaris sense conèixer la seva implementació. No es pot dur a terme en una operació `SELECT`.

Aquesta acció podria servir per fer una comprovació de consistència en un conjunt de valors per ser inserits, al format de les dades abans de ser introduïdes, a una modificació a una taula o a una modificació d'un conjunt de files.

Formen part de PostgreSQL a partir de la seva versió 7.3.

- Els *triggers* són una manera d'estendre la funcionalitat igual que els procediments emmagatzemats.
- No requereixen intervenció de l'usuari, invocant-se automàticament.
- Es poden utilitzar per comprovar la consistència de les dades.
- Es defineixen perquè s'executin abans o després (`BEFORE | AFTER`) de `INSERT`, `UPDATE` o `DELETE`.
- Poden actuar per cada fila (`ROW`) o una vegada per instrucció de SQL (`STATEMENT`).

PostgreSQL permet que els *triggers* estiguin escrits en SQL, C o altre llenguatge procedimental definit.

Un *trigger* cal que tingui un nom diferent de la resta de *triggers* d'una mateixa taula.

CREACIÓ D'UN *trigger*

La sintaxi per la creació d'un *trigger* és:

```
CREATE TRIGGER nom { BEFORE | AFTER } { esdeveniment [ OR ... ] }
    ON taula [ FOR [ EACH ] { ROW | STATEMENT } ]
    EXECUTE PROCEDURE nom_funció ( arguments )
```

`BEFORE | AFTER` per determinar quan s'executa el *trigger* abans o després de l'`esdeveniment`. `BEFORE` s'executa abans de l'instrucció i abans de que s'operi sobre una fila. `AFTER` s'executa després que s'afecti una fila i abans d'acabar l'instrucció.

`esdeveniment` = `INSERT`, `UPDATE` o `DELETE`. Es poden especificar múltiples esdeveniments separats per un `OR`.

Hi ha dos tipus de *triggers*, els *triggers* per fila i els *triggers* per sentència. En els *triggers* per fila la funció s'invoca una vegada per cada fila afectada per la sentència que ha

disparat el *trigger*. Pel contrari en un *trigger* per declaració se invoca sols una vegada quan s'executa la sentència apropiada sense tenir en compte el nombre de files afectades per aquesta declaració. Aquest tipus de *trigger* sempre retornen `NUL`.

`FOR EACH ROW | FOR EACH STATEMENT:` Per cada clàusula `FOR EACH` determina si el *trigger* ha de ser llançat per cada fila afectada o per cada declaració. El valor per defecte és `FOR EACH STATEMENT.`

Per exemple, suposem que volem tenir constància de qui modifica la taula d'empleats ja que en ella hi consten els sous de tots els guies i altres treballadors. En l'exemple següent veiem que cada vegada que es fa una inserció o una modificació en una taula es registra el nom de l'usuari que ho ha fet i l'hora en la fila inserida o modificada. A més comprova que el nom de l'empleat no estigui a nul i que el salari tingui un valor positiu.

La taula 'empleat' la varem crear així:

```
CREATE TABLE empleat (
    nom_empleat text,
    salari integer,
    darrera_modif timestamp,
    darrer_usuari text
);
```

Crearem la funció segons el que hem comentat:

```
CREATE FUNCTION reg_empleat() RETURNS trigger AS $reg_empleat$
    BEGIN
        -- Comprova que s'informi el nom_empleat i el salari.
        IF NEW.nom_empleat IS NULL THEN
            RAISE EXCEPTION 'nom_empleat no pot ser null';
        END IF;
        IF NEW.salari IS NULL THEN
            RAISE EXCEPTION 'el salari no pot ser null,
NEW.nom_empleat;
        END IF;

        -- Comprova que el salari sigui positiu.
        IF NEW.salari < 0 THEN
            RAISE EXCEPTION 'el salari % no pot ser negatiu',
NEW.nom_empleat;
        END IF;

        -- Registre qui ha fet la modificació
        NEW.darrera_modif := 'now';
        NEW.darrer_usuari := current_user;
        RETURN NEW;
    END;
```

```
$reg_empleat$ LANGUAGE plpgsql;
```

Al posar '%' treu el nou salari, en el cas de que el valor sigui negatiu.

Ara caldrà crear el *trigger* 'reg_empleat' i provar-ho.

```
CREATE TRIGGER reg_empleat BEFORE INSERT OR UPDATE ON empleat
    FOR EACH ROW EXECUTE PROCEDURE reg_empleat();
```

Per provar-ho inserim una fila:

```
INSERT INTO empleat VALUES ('mila',1000,null,null);
```

I fem la consulta per veure quin usuari ha accedit al registre d'un determinat empleat:

```
SELECT * FROM empleat WHERE nom_empleat ='mila';
```

el resultat és la fila inserida amb les dades de qui l'ha inserit i en quin moment:

```
nom_empleat | salari |        darrera_modif         | darrer_usuari
------------+--------+------------------------------+-----------
 mila       |   1000 | 2005-10-30 10:57:12.792      | postgres
(1 fila)
```

Si intentem fer aquesta inserció:

```
INSERT INTO empleat VALUES ('miquel',-500,null,null);
```

Ens avisarà amb el missatge que hem previst en la funció que hem creat:

```
ERROR: Miquel salari no pot ser negatiu.
```

El mateix passarà si no introduïm cap 'salari'.

En la funció d'exemple següent, corresponent a la primera part de la pràctica de BDII (veure resta en l'annex). Tenim una taula 'agrupacions' en la que hi ha els camps: 'codi_ral·li' i 'codi_safari' i ens cal limitar el numero de safaris que té cada ral·li a 5.

Crearem la funció que ens generi una excepció si s'intenta afegir un safari a un mateix ral·li quan ja n'hi ha 5:

```
CREATE FUNCTION checkNumAgrup () RETURNS trigger AS '
    BEGIN
    IF (SELECT COUNT(*) FROM Agrupacions WHERE codi_ralli =
NEW.codi_ralli) >= 5 THEN
            RAISE EXCEPTION ''Violació de regla de negoci - No més
de 5 safaris per ralli!'';
        END IF;

        RETURN new;
    END;
' LANGUAGE 'plpgsql';
```

Ara podem crear un *trigger* que s'executi cada vegada que es vol fer un `INSERT` d'un 'safari' i es dispari quan per un 'ral·li' ja n'hi hagi 5 a 'Agrupacions':

```
CREATE TRIGGER InsertAgrupacions
    BEFORE INSERT ON Agrupacions
     FOR EACH ROW
EXECUTE PROCEDURE checkNumAgrup();
```

Si intentem inserir un nou 'safari' d'un mateix 'ral·li' a la taula 'Agrupacions' quan ja n'hi ha 5 ens surt una excepció:

```
Error: org.postgresql.util.PSQLException: ERROR: Violació de regla
de negoci - No més de 5 safaris per ralli!, SQL State: P0001,
Error Code: 0
```

Si consultem els codis d'error de PostgreSQL veurem que `Error Code: 0` vol dir que s'ha acabat satisfactòriament i que `P0001` és un error de PL/pgSQL del tipus `RAISE EXCEPTION`. O sigui, que el procediment ha avortat donada una excepció ocorreguda durant l'execució del mateix.

Tipus de *triggers*.

Triggers múltiples:

Si es defineix més d'un *trigger* s'executen en ordre alfabètic per nom. En el cas de `BEFORE` el resultat d'un es converteix en l'entrada del següent i si un retorna `NULL` s'abandona l'operació. Típicament, els triggers `BEFORE` s'utilitzen per comprovar o modificar les dades que s'introduiran o s'actualitzaran. Per exemple, un *trigger* `BEFORE`

podria ser utilitzat per introduir el temps actual a una columna del tipus *timestamp*, o comprovar que dos elements de la fila són coherents.

Triggers en cascada:

Si una funció executa instruccions de SQL, aquestes podrien originat altres *triggers*. D'aquest fet pot resultar una recursivitat infinita. Un *trigger* per un `INSERT` pot utilitzar una funció que insereixi una fila en una taula, provocant un nou `INSERT` i així successivament. No hi ha cap limitació en el nombre de nivells de la cascada. És la responsabilitat del programador del *trigger* evitar la recursivitat infinita.

MODIFICACIÓ I ELIMINACIÓ D'UN *trigger*:

Igual que es pot crear un *trigger* també hi ha la possibilitat de modificar-lo:

```
ALTER TRIGGER nom ON taula RENAME TO nou_nom
```

O eliminar-lo:

```
DROP TRIGGER nom ON taula [ CASCADE | RESTRICT ]
```

Si eliminem una funció que té un *trigger* definit, el *trigger* fallarà, i tornant a crear la funció amb el mateix nom no corregirà el problema. Cal tornar a crear el *trigger* després de tornar a crear la funció.

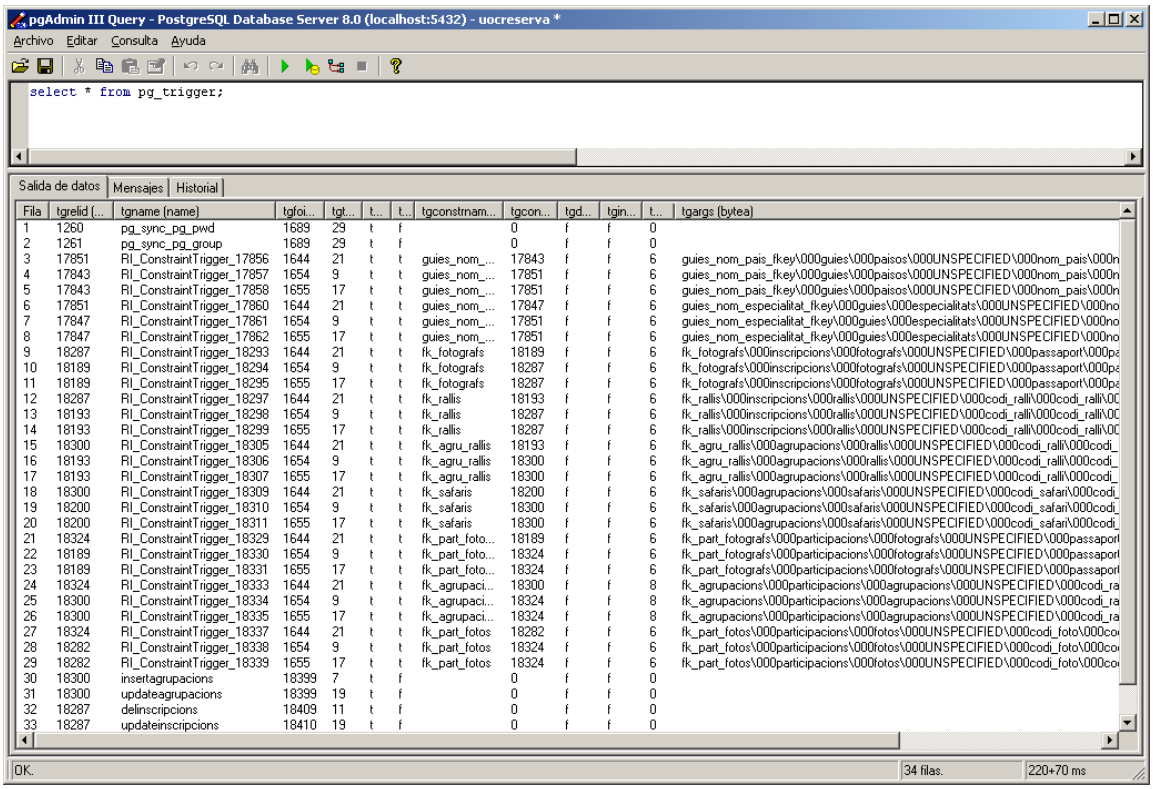
Amb `CASCADE`, suprimeix automàticament els objectes que depenen del *trigger*.

`RESTRICT` és el valor per defecte i es nega a suprimir el *trigger* si hi ha objectes que en depenen.

Eliminem el *trigger* que hem definit en l'exemple:

```
DROP TRIGGER reg_empleat ON empleat;
```

Els *triggers* ja existents en PostgreSQL estan emmagatzemats en la vista del sistema pg_trigger. Si fem la consulta d'aquesta vista veurem que en la versió 8.0 hi ha diversos triggers ja creats i els que hem creat nosaltres amb les restriccions.



La comanda `CREATE TRIGGER` forma part de les eines de l'estàndard SQL:1999, que el permet per disparar actualitzacions a columnes específiques. Però mentre que PostgreSQL només permet l'execució d'una funció definida per l'usuari, l'SQL:1999 permet l'execució d'un cert nombre de sentències de SQL, com `CREATE TABLE`. SQL:1999 especifica que s'hauria de disparar amb *triggers* múltiples en el moment de la creació. PostgreSQL utilitza el nom per ordre alfabètic.

L'habilitat per especificar accions múltiples per a un *trigger* únic que utilitza `OR` és una extensió de l'estàndard de SQL de PostgreSQL.

9.- Vistes

Les vistes en PostgreSQL s'implementen utilitzant el sistema de regles.

El sistema de regles en PostgreSQL consisteix en modificar les consultes d'acord a regles emmagatzemades com a part de la base de dades. Aquestes consultes modificades són passades, en ordre, a l'optimitzador, planificador i finalment a l'executor. Això el diferència d'altres SGBD que s'implementen els sistemes de regles com procediments i *triggers* emmagatzemats.

Aquest sistema és molt poderós i es pot emprar per procediments, vistes i versions.

9.1.- Creació

Per crear una vista s'utilitza `CREATE VIEW` amb la sintaxi següent:

```
CREATE VIEW nom_vista [(nom_columna [,....])] AS query;
```

Però com que les vistes s'implementen utilitzant el sistema de regles (*rules*), no hi ha diferència entre d'anterior seqüència i la següent:

```
CREATE TABLE nom_vista
     (llista d'atributs de nom_taula);

CREATE RULE nom AS ON SELECT TO nom_vista DO INSTEAD
     SELECT * FROM nom_taula;
```

Ja que això és el que fa internament la comanda `CREATE VIEW`. Això té alguns efectes i un d'ells és que la informació sobre una vista en el sistema de catàlegs de PostgreSQL és exactament el mateix que per una taula.

Les opcions opcionals a l'SQL estàndard de `CREATE VIEW` són les següents:

```
CREATE VIEW nom_vista [(columna [, ...])]
     AS query
          [WITH [CASCADE | LOCAL ] CHECK OPTION ]
```

`CHECK OPTION`. Cal utilitzar aquesta opció amb les vistes actualitzables. Totes les comandes `INSERT` i `UPDATE` sobre la vista es controlen per assegurar que les dades són conformes a la condició definida a la vista. Si no, l'actualització o inserció és rebutjada.

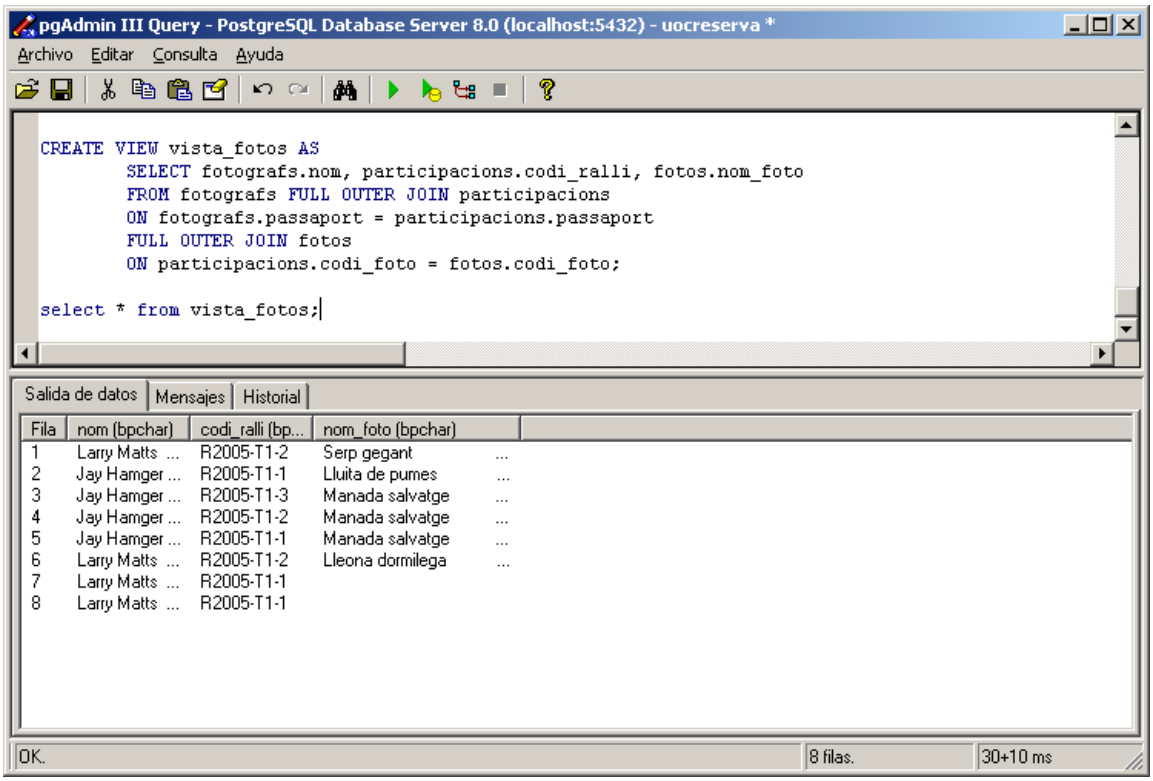
`LOCAL`. Per controlar l'integritat de la vista.

`CASCADE`. Per controlar l'integritat sobre aquesta vista i totes les vistes dependents. Si no s'especifica ni `LOCAL` ni `CASCADE` s'assumeix `CASCADE`, per defecte.

Per exemple i seguint amb la bases de dades d'exemple UOCReserva crearem la següent vista utilitzant el `FULL OUTER JOIN`:

```
CREATE VIEW vista_fotos AS
     SELECT fotografs.nom, participacions.codi_ralli,
fotos.nom_foto
     FROM fotografs FULL OUTER JOIN participacions
     ON fotografs.passaport = participacions.passaport
     FULL OUTER JOIN fotos
     ON participacions.codi_foto = fotos.codi_foto;
```

I des de el pgAdmin la consultem:



9.2.- Manipulació

Normalment les vistes són sols de lectura, a diferència d' SQL estàndard. El sistema no permet fer `INSERT`, `UPDATE` o `DELETE` en una vista. Es pot aconseguir l'efecte d'un `UPDATE` en una vista creant una regla que reescrigui les insercions, modificacions o

esborrats. Per això utilitzàrem una regla `CREATE RULE`. Aquesta és una extensió de PostgreSQL i no existeix en SQL estàndard.

Sintaxi del `CREATE RULE`:

```
CREATE RULE nom AS ON esdeveniment
    TO objecte [ WHERE condició ]
    DO [ ALSO | INSTEAD ] [ acció | NOTHING ]
```

`esdeveniment` = `INSERT`, `UPDATE` o `DELETE`. Es poden especificar múltiples esdeveniments separats per un `OR`.

Un exemple de creació de vista, i de la *rule* necessària per a poder fer actualitzacions és el següent:

Suposem que tenim la taula test que hem creat amb els següents camps:

```
CREATE TABLE test (nom char (20), cognom char(20));
```

Crearem la vista:

```
CREATE VIEW vista AS SELECT nom, cognom FROM test;
```

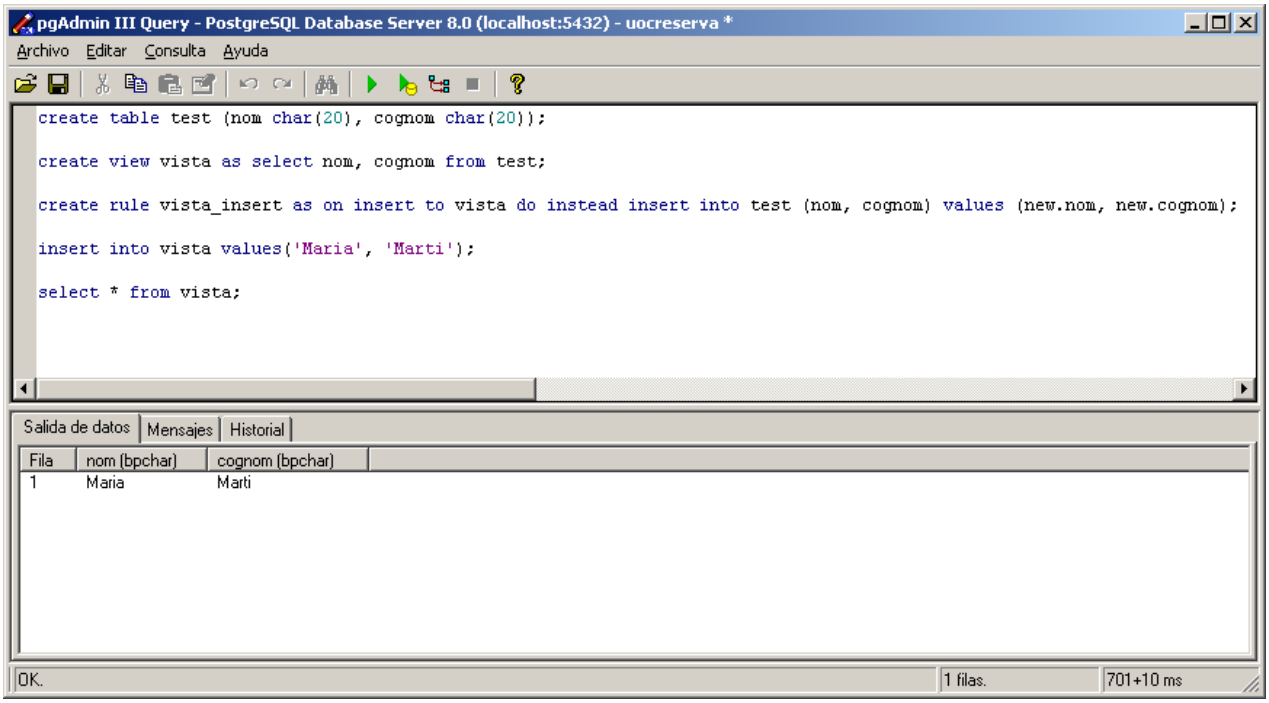
Ara creem la *rule*:

```
CREATE RULE vista_insert AS ON INSERT TO vista DO INSTEAD
INSERT INTO test (nom, cognom) VALUES (new.nom, new.cognom);
```

Ara podem fer d'inserció:

```
INSERT INTO vista VALUES ('Maria', 'Martí');
```

Fem el `SELECT` i veiem com ha funcionat:



El que sí es pot fer és modificar la vista amb la extensió de la sentencia del `CREATE VIEW` utilitzant `CREATE OR REPLACE VIEW`.

`CREATE OR REPLACE VIEW` és similar però si una vista ja existeix amb el mateix nom, la substitueix.

Sols es pot substituir una vista amb una nova consulta que generi idèntica sèrie de columnes (noms i tipus de dades).

La sintaxi de `CREATE VIEW` s'amplia amb el `[OR REPLACE ]:`

```
CREATE [OR REPLACE ] VIEW nom_vista [(columna [, ...])]
     AS query
          [WITH [CASCADE | LOCAL ] CHECK OPTION ]
```

Per eliminar una vista podem utilitzar la mateixa comanda que l'SQL estàndard, `DROP VIEW`. Cal ser el propietari de la vista per eliminar-la.

La sintaxi de `DROP VIEW` és la següent:

```
DROP VIEW nom_vista [,...] [CASCADE | RESTRICT ]
```

Amb l'opció `CASCADE` s'esborren automàticament els objectes dependents de la vista, tals com altres vistes.

Amb `RESTRICT` no esborra la vista si hi ha objectes que en depenen. És l'opció per defecte.

9.3.- Vistes del sistema.

PostgreSQL disposa de algunes vistes ja confeccionades. Algunes vistes del sistema tenen accés a les consultes més utilitzades en els catàlegs del sistema. Altres donen accés al estat del servidor intern.

Algunes de les principals vistes que hi ha disponibles són les següents:

Nom_vista	Objectiu
pg_indexes	índex
pg_locks	bloquejos
pg_rules	regles
pg_settings	paràmetres
pg_stats	estadístiques
pg_tables	taules
pg_user	usuaris
pg_views	vistes

Qualsevol de les vistes anteriors utilitza altres vistes també ja definides.

9.4.- Traducció de consultes sobre vistes.

La informació sobre una vista en el sistema de catàlegs de PostgreSQL és el mateix que per una taula. D'aquesta manera, pels traductors de *queries*, no hi ha diferència entre una taula i una vista, són el mateix: relacions.

El sistema de regles incorpora les definicions de les vistes en l'arbre de traducció original (*querytree*). La implementació del sistema de regles és una tècnica anomenada reescriptura de la consulta. El sistema de reescriptura és un mòdul que existeix entre l'etapa del traductor i el planificador/optimitzador.

El sistema de reescriptura de la consulta processa l'arbre tornat per l'etapa de traducció, que representa una consulta del usuari, i si existeix una regla que calgui aplicar a la consulta, reescriu l'arbre d'una forma alternativa.

El *querytree* és la representació interna de una consulta en la que se separen i agrupen els components en forma d'arbre.

Components d'un *querytree:*

- La **instrucció**: `SELECT`, `UPDATE`, `INSERT` o `DELETE`.
- La '***range table***' (abast de la taula): inclou les relacions que utilitza.
- La '***result relation***' (relació resultant): un índex a la *range table* on estaran els resultats. Generalment `SELECT` no l'inclou.
- La '***target list***' (llista d'etiquetes): és la llista d'elements entre el `SELECT` i el `FROM` en una instrucció de `SELECT`, la llista de files afectades en `INSERT` i `UPDATE`. No s'utilitza en `DELETE`.
- La **qualificació** correspon al `WHERE` i indica si cal actualitzar o no una fila.
- El '***join tree***' (arbre del `JOIN`), combina parts del `FROM` i el `WHERE` per descriure l'estructura del `JOIN`.
- '***others***' (la resta), altres clàusules com `ORDER BY`.

Els beneficis d'implementar les vistes amb el sistema de regles estan en què l'optimització té tota la informació sobre quines taules han de ser revisades, les relacions entre aquestes taules, les qualificacions restrictives a partir de la definició de les vistes i les qualificacions de la *query* original, tot en un únic arbre de traducció. I aquesta és també la situació quan la *query* original és una *JOIN* entre vistes.

L'optimizador cal que decideixi quina és la millor ruta per executar la *query*. Quanta més informació tingui l'optimizador, millor serà la decisió. I la forma en la qual s'implementa el sistema de regles a PostgreSQL assegura que tota la informació sobre la *query* és utilitzable.

Per comprendre com treballa el sistema de regles, és necessari conèixer quan s'invoca i quins són els seus *inputs* i els seus resultats.

El sistema de regles se situa entre el traductor de la *query* i l'optimitzador. Agafa la sortida del traductor, un *querytree*, i les regles de reescriptura del catàleg *pg_rewrite*, que són també *querytree*, amb alguna informació extra, i crea cap o molts *querytree* com a resultat. D'aquesta manera, la seva entrada i la seva sortida són sempre tal com el traductor mateix podria haver-les produït i tot apareix representable com una instrucció SQL.

Aquests *querytree* són visibles quan arrenquem el motor de PostgreSQL amb nivell de *debug* 4 i teclegem *queries* en l'interfície d'usuari interactiu. Les accions de les regles emmagatzemades en el catàleg de sistema *pg_rewrite* estan emmagatzemades també com *querytree*. No estan formades com la sortida del *debug*, però contenen exactament la mateixa informació.

Les representacions de SQL de *querytree* són suficients per entendre el sistema de regles.

A continuació mostrarem un exemple de com implementar vistes utilitzant el sistema de regles.

Suposem que en la UOCReserva d'exemple tenim les taules: 'zona', 'visites' i 'país':

```
CREATE TABLE zona (id char (10), nom char(20));
CREATE TABLE visites (id char (10), reId char(10));
CREATE TABLE pais (reId char (10), nom char(20));
```

Cal que tinguem la relació creada: vista_test:

```
CREATE TABLE vista_test (id char (10), nom char(20));
```

Suposem que tenim la regla següent:

```
CREATE RULE "_RETURN"
AS ON SELECT
TO vista_test
DO INSTEAD
     SELECT z.nom, p.nom
     FROM zona z, visites vi, pais p
     WHERE z.id = vi.id AND
          p.reId = vi.reId;
```

Aquesta regla es dispararà cada vegada que es detecti una `SELECT` contra la relació 'vista_test'.

Suposem que l'usuari fa la següent consulta contra 'vista_test':

```
SELECT nom
FROM vista_test
WHERE nom <> 'Paula';
```

El sistema de reescriptura fa els passos següents:

- Pren la consulta donada per la part d'acció de la regla.
- Adapta la llista-objectiu per recollir el nombre i ordre dels atributs donats en la consulta d'usuari.
- Afegeix la qualificació donada en la clàusula WHERE de la consulta de l'usuari a la qualificació de la consulta donada en la part de l'acció de la regla.

D'acord amb això la consulta de l'usuari serà reescrita de la forma següent:

```
SELECT z.nom
     FROM zona z, visites vi, pais p
     WHERE z.id = vi.id AND
          p.reId = vi.reId AND
               z.nom<> 'Paula';
```

La reescriptura es fa en la representació interna de la consulta de l'usuari tornada per l'etapa de traducció però la nova estructura de dades representarà la consulta anterior.

9.5.- Vistes materialitzades.

Les vistes executen una sentència `SELECT` cada vegada que es fa una consulta sobre ella (llegint les dades directament de les taules que intervenen en la vista).

Les vistes materialitzades no executen cada vegada el `SELECT`, més bé es pot dir que mantenen una taula amb les dades que cal que torni la vista i per tant redueixen dràsticament l'execució de consultes complexes.

Oracle implementa les vistes materialitzades des de la seva versió 8i. Les vistes materialitzades permeten la pre-sumarització de dades agregades (promitjos, comptes, sumes) i pre-unió de taules; de molta utilització en taules poc consultades i *Datawarehousing*.

La darrera versió que estem estudiant, la PostgreSQL 8.0, no implementa vistes materialitzades encara que són possibles utilitzant el seu potent PL/pgSQL, i el sistema funcional de *triggers*.

10.- Índexs.

Els índexs s'utilitzen principalment per incrementar el rendiment d'una BD.

Quan accedim a una taula, PostgreSQL normalment llegeix des del principi de la taula al final, buscant les files pertinents. Amb un índex, pot trobar de pressa valors específics a l'índex, llavors anar-se'n directament a buscar les files. D'aquesta manera, els índexs permeten la recuperació ràpida de files específiques d'una taula.

Internament, PostgreSQL emmagatzema dades en arxius del sistema operatiu. Cada taula té el seu propi arxiu, i les files de dades s'emmagatzemen una després d'una altra a l'arxiu. Un índex és un arxiu separat ordenat per una o més columnes. Conté indicadors a l'arxiu de taula, permetent accés ràpid a valors específics d'aquesta.

Es poden crear tants índexs com es vulgui. Naturalment, un índex en una columna que rarament és utilitzada és un malbaratament d'espai de disc. També, el rendiment pot davallar si existeixen molts índexs ja que els canvis de fila exigeixen una actualització a cada índex.

És possible crear un índex que avarca columnes múltiples. Els índexs multicolumna estan ordenats per la primera columna indexada. Quan la primera columna conté uns quants valors iguals, l'ordenació continua utilitzant la segona columna indexada. Els índexs multicolumna són útils només en columnes amb molts valors duplicats.

10.1.- Creació, modificació i destrucció d'índexs.

Amb la comanda `CREATE INDEX` construirem un índex sobre una determinada taula:

```
CREATE [ UNIQUE ] INDEX nom ON
taula [ USING model ]
    ( { columna | ( expressió ) } [ classe_op ] [, ...] )
    [ TABLESPACE espai_lògic ]
    [ WHERE predicat ]
```

On `UNIQUE` fa que el sistema verifiqui els valors duplicats en la taula en la creació de l'índex i cada vegada que s'insereix una dada.

`model` és el nom del mètode d'accés que s'utilitzarà per l'índex. Les opcions són: `btree`, `hash`, `rtree` i `gist`. El model per defecte és el `btree`.

- Sols els models `btree` i `gist` poden suportar els índexs multicolumna.
- Sols el `btree` es poden declarar `UNIQUE`.

Tots admeten expressions (`expressió`).

`expressió` vol dir una expressió basada sobre una o varies columnes de la taula. L'expressió cal que sigui escrita entre parèntesis.

En tots es poden posar operadors de classe (`classe_op`).

`Classe_op` representa el nom d'una classe de l'operador. Es pot especificar una classe d'operador per a cada columna d'un índex. La classe d'operador identifica els operadors a utilitzar per l'índex per a aquesta columna.

`espai_lògic` és l'espai lògic on es crea l'índex. Si no s'especifica, s'utilitza el espai de taules per defecte.

Tots admeten índexs parcials: Són aquells que no afecten a tota la taula i per tant estan definits amb una expressió condicional (`predicat`).

`predicat` és l'expressió de restricció per a un índex parcial.

Els diferents tipus d'índex proposats per PostgreSQL utilitzen diferents algoritmes i tenen diferents utilitats.

El `btree` és adequat per treballar amb consultes d'igualtat o classificació que utilitzi els següents operadors: < , <=, =, >=, >.

El `rtree` serveix per treballar amb consultes sobre dades espacials. Per les comparacions s'utilitzen els següents operadors: <<, &<, &>, >>, @, ~=, &&.

L'índex `hash` sols es pot utilitzar amb comparacions d'igualtat simples. Sols utilitza l'operador =.

Els índexs `gist` no són un sol gènere d'índex sinó més aviat una infraestructura dins de la qual es poden implementar diverses estratègies d'indexatge.

Per crear un índex `b-tree` en la taula 'visites' en la columna 'reId' faríem:

```
CREATE UNIQUE INDEX reId_idx ON visites (reId);
```

També es pot modificar la definició d'un índex i el seu nom. Per això s'utilitza la comanda `ALTER INDEX`:

```
ALTER INDEX nom
    acció [, ... ]
```

```
ALTER INDEX nom
    RENAME TO nou_nom
```

On acció pot ser:

```
OWNER TO nou_propietari
SET TABLESPACE nom_espai_lògic
```

`OWNER`: Aquesta forma modifica el propietari d'un índex per l'usuari especificat. Això només pot ser fet per un superusuari.

`SET TABLESPACE`: Aquesta forma canvia l'espai lògic de l'índex per l'espai lògic especificat i desplaça el fitxer de dades associat amb l'índex per al nou espai lògic.

`RENAME`. La forma `RENAME` modifica el nom de l'índex. Allò no té cap efecte sobre les dades emmagatzemades.

Totes les accions excepte `RENAME` poden ser combinades en una llista de diverses modificacions a aplicar en paral·lel.

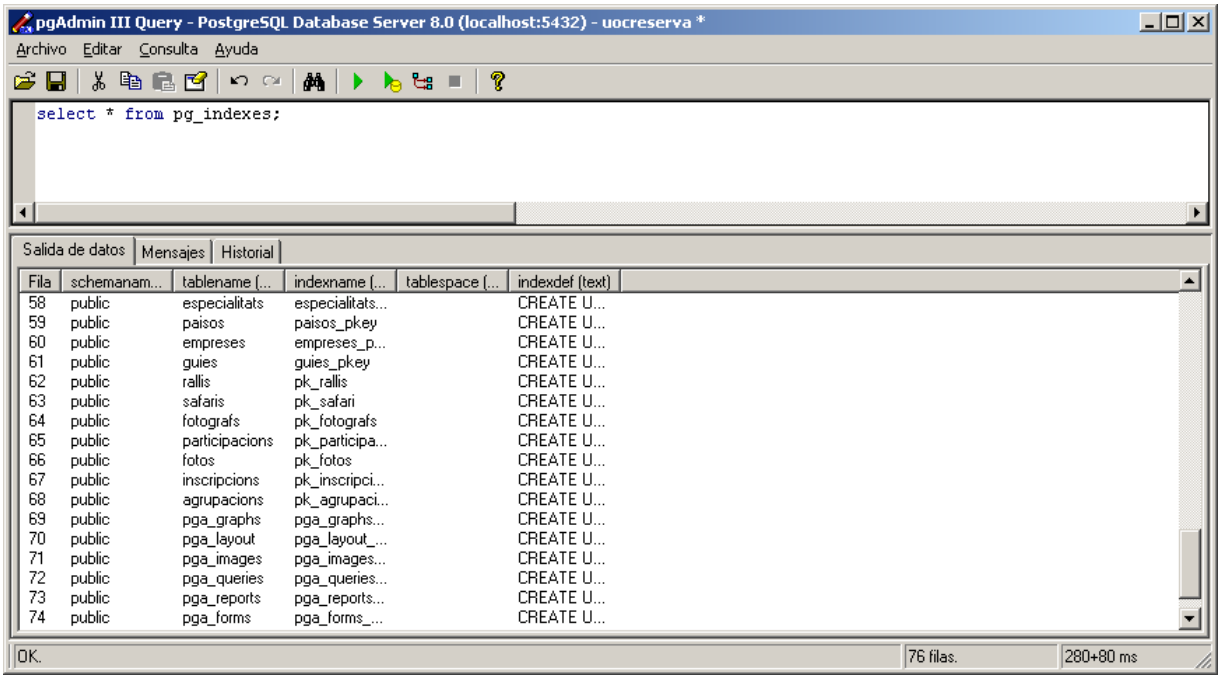
Per esborrar un índex s'utilitza `DROP INDEX`:

```
DROP INDEX nom [, ...] [ CASCADE |
RESTRICT ]
```

Amb `CASCADE` suprimeix automàticament els objectes depenent de l'índex.

Amb `RESTRICT`, que és l'opció per defecte, no suprimeix l'índex si hi ha objectes que en depenen.

Podem veure els índexs creats consultant la vista del sistema pg_indexes:


pgAdmin III Query - PostgreSQL Database Server 8.0 (localhost:5432) - uocreserva *
Archivo Editar Consulta Ayuda
select * from pg_indexes;
Salida de datos Mensajes Historial
Fila schemanam... tablename (... indexname (... tablespace (... indexdef (text)
58 public especialitats especialitats... CREATE U...
59 public paisos paisos_pkey CREATE U...
60 public empreses empreses_p... CREATE U...
61 public guies guies_pkey CREATE U...
62 public rallis pk_rallis CREATE U...
63 public safaris pk_safari CREATE U...
64 public fotografs pk_fotografs CREATE U...
65 public participacions pk_participa... CREATE U...
66 public fotos pk_fotos CREATE U...
67 public inscripcions pk_inscripci... CREATE U...
68 public agrupacions pk_agrupaci... CREATE U...
69 public pga_graphs pga_graphs... CREATE U...
70 public pga_layout pga_layout_... CREATE U...
71 public pga_images pga_images... CREATE U...
72 public pga_queries pga_queries... CREATE U...
73 public pga_reports pga_reports... CREATE U...
74 public pga_forms pga_forms_... CREATE U...
OK.
76 filas.
280+80 ms

11.- Transaccions

PostgreSQL manté la consistència de les dades en un model multiversió: Control de la Concurrència Multi Versió (MVCC). Aquesta és una tècnica avançada per millorar les prestacions d'una base de dades en un entorn multiusuari que implementa PostgreSQL des de la versió 6.5 del juny de 1999.

MVCC és la tecnologia utilitzada per evitar bloqueigs innecessaris. Si alguna vegada hem utilitzat algun SGBD amb capacitats SQL, tal com MySQL o Informix, probablement hem pogut notar que hi ha ocasions en les que una lectura ha d'esperar per accedir a la informació de la base de dades. L'espera està provocada per usuaris que estan escrivint en aquesta base de dades. Resumint, el lector està bloquejat pels escriptors que estan actualitzant els registres.

Per exemple considerarem una sèrie de transaccions sobre la taula clients.

La taula clients ha estat creada amb els següents camps:

```
CREATE TABLE clients (id_client char(10), nom char (20), cognoms
char (35), compte char(30), import integer);
```

Si considerem les transaccions següents:

Usuari A	**Temps**	**Usuari B**
`BEGIN TRANSACTION;`	`T1`	
	`T2`	`BEGIN TRANSACTION;`

`UPDATE clients`
`SET import = import – 3`
`WHERE id_client = '2' and compte = '111-222';`
	`T3`	

	`T4`	`SELECT SUM (import)` `FROM clients;`
	`T5`	`COMMIT TRANSACTION;`

`ROLLBACK TRANSACTION;`	T6	

En el moment T1, l'usuari A comença una nova transacció. En el moment T2 l"usuari B comença una nova transacció. En el moment T3 l'usuari A descompte 3 en un compte del client 2. En el moment T4, l'usuari B consulta la suma del compte de tots els clients i completa la transacció en el moment T5. En el moment T6, l'usuari A fa un `ROLLBACK` descarregant tots els canvis de la transacció. Si aquestes transaccions no estan aïllades per cada un, l'usuari B pot tenir una resposta incorrecta.

La solució a aquest problema s'anomena `READ COMMITTED`. `READ COMMITTED` és un dels dos nivells d'aïllament de transaccions suportats per PostgreSQL. Una transacció corrent amb el nivell de aïllament `READ COMMITTED` no permet llegir dades abans de haver fet un `COMMIT`.

Veiem un altre exemple:

Usuari A	**Temps**	**Usuari B**
	T1	`BEGIN TRANSACTION;`
`BEGIN TRANSACTION;`	T2	
	T3	`SELECT import FROM clients WHERE id_client = '2' AND compte = '111-222';`
`UPDATE clients SET import = 20 WHERE id_client = 2 AND compte = '111-222';`	T4	
`COMMIT TRANSACTION;`	T5	
	T6	`SELECT import FROM clients WHERE id_client = 2 AND compte = '111-222';`

	T7	COMMIT TRANSACTION;

Un altre cop els dos usuaris comencen la transacció en el moment T1 i T2 respectivament. En el moment T3 l'usuari B veu que el client 2 té una compte amb import X. L'usuari A en el moment T4 ha modificat l'import del compte d'aquest client a 20 i fa el COMMIT en el moment T5. En el moment T6, l'usuari B executa la mateixa consulta que abans però ara troba una compte amb 20. Aquest problema es coneix com a lectura no repetible.

I en un tercer exemple veiem:

Usuari A	**Temps**	**Usuari B**
	T1	BEGIN TRANSACTION;
BEGIN TRANSACTION;	T2	
	T3	SELECT * FROM clients;
INSERT INTO clients VALUES (6,'Andreu','Font','222-333',0,00);	T4	
COMMIT TRANSACTION;	T5	
	T6	SELECT * FROM clients;
	T7	COMMIT TRANSACTION;

En aquest exemple l'usuari B executa la mateixa consulta dues vegades entre les dues transaccions. Entre les dues consultes l'usuari A ha inserit una nova fila i ha fet el COMMIT. En el moment T6, en la nova consulta l'usuari B troba una nova fila. Aquest problema es conegut com a lectura fantasma. La solució als dos problemes és la transacció amb nivell d'aïllament serialitzable, que sols permet veure dades en que s'ha fet el COMMIT abans de que les transaccions de l'altre usuari comencin.

Mitjançant l'ús de MVCC, PostgreSQL evita aquest problema per complet. MVCC està considerat millor que el bloqueig en l'àmbit de fila perquè un lector mai no és bloquejat

per un escriptor. En comptes d'això, PostgreSQL manté una ruta de totes les transaccions realitzades pels usuaris de la base de dades. PostgreSQL és capaç de manipular els registres sense necessitat de que els usuaris hagin d'esperar que els registres estiguin disponibles.

11.1.- Gestió de transaccions

Mentre es consulta una base de dades, cada transacció veu una imatge de les dades, és a dir, una versió de la base de dades, sense tenir en compte l'estat actual de les dades que hi ha per sota. Així s'evita que la transacció vegi dades inconsistents produïdes per l'actualització d'una altra transacció concurrent, proporcionant aïllament transaccional per cada sessió de la base de dades.

PostgreSQL utilitza el que s'anomena aïllament transaccional i regles de resolució de conflictes per resoldre operacions concurrents i té dos nivells d'aïllament: serialitzable i lectura cursada (`SERIALIZABLE` i `READ COMMITTED`).

En el nivell serialitzable (`SERIALIZABLE`), es pren una instantània al començament de la transacció. Es fixa una vista de la lectura de la base de dades durant la transacció.

En el nivell de lectura cursada (`READ COMMITTED`), es pren una nova instantània al començament de cada consulta. Així, la vista de la base de dades és estable durant l'iteració d'una consulta, però pot canviar durant les altres consultes que es facin dins d'una transacció.

Cal remarcar que en PostgreSQL la sentència `START TRANSACTION` té la mateixa funció que `BEGIN` i com hem vist serveixen per iniciar una transacció.

La seva sintaxi és així:

```
BEGIN [ WORK | TRANSACTION ] [ mode_transacció [, ...] ]
```

O així:

```
START TRANSACTION [ mode_transacció [, ...] ]
```

En ambdós casos el `mode_transacció` pot ser un dels següents:

```
ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED
| READ UNCOMMITTED }
READ WRITE | READ ONLY
```

Si cal es pot canviar el nivell d'aïllament utilitzant la següent sentència:

```
ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ | READ COMMITTED
| READ UNCOMMITTED }
READ WRITE | READ ONLY
```

La sentència `SET TRANSACTION` afecta sols a la transacció actual i inicialitza les seves característiques.

Si es vol canviar el nivell d'aïllament per la sessió es pot utilitzar la sentència `SET SESSION`.

```
SET SESSION CHARACTERISTICS AS
     TRANSACTION ISOLATION LEVEL { SERIALIZABLE | REPEATABLE READ
| READ COMMITTED | READ UNCOMMITTED }
READ WRITE | READ ONLY
```

En PostgreSQL, les transaccions per defecte funcionen amb el nivell d'aïllament `READ COMMITTED`.

Per donar per finalitzat una transacció s'utilitza indistintament el `COMMIT` i l'`END`. La comanda `END` és un sinònim de `COMMIT`. La seva sintaxi és la següent:

```
END [WORK | TRANSACTION]
```

La sintaxi de `COMMIT` és:

```
COMMIT [WORK | TRANSACTION]
```

Per avortar la transacció s'utilitza `ROLLBACK`:

```
ROLLBACK [WORK | TRANSACTION]
```

En tots els casos `WORK | TRANSACTION` són opcionals i poden ser ignorats o utilitzar per fer el nostre SQL més llegible.

11.2.- Nivells d'aïllament.

El nivell d'aïllament d'una transacció determina quines dades podem veure de la transacció quan altres transaccions estan funcionant en el mateix moment.

`READ COMMITTED`, quan una transacció sols pot veure els canvis confirmats abans de que ella comenci. És el valor per defecte.

`SERIALIZABLE`, quan totes les instruccions de la transacció en curs poden veure sols les canvis confirmats abans de que la primera consulta o la primera instrucció de modificació de dades s'hagi executat en aquesta transacció.

L'SQL estàndard defineix dos nivells addicionals, `READ UNCOMMITTED` i `REPEATABLE READ`. En PostgreSQL, sols dos i `READ UNCOMMITTED` és tractat com `READ COMMITTED` i `REPEATABLE READ` és tractat com `SERIALIZABLE`.

Nivells d'aïllament amb les seves accions:

Accions / Nivells aïllament	**Lectura 'bruta'**	**Lectura no repetible**	**Lectura 'fantasma'**
`READ UNCOMMITTED`	Possible	Possible	Possible
`READ COMMITTED`	No Possible	Possible	Possible
`REPEATABLE READ`	No Possible	No Possible	Possible
`SERIALIZABLE`	No Possible	No Possible	No Possible

El nivell d'aïllament de la transacció no es pot modificar després que la primera consulta o la primera instrucció de modificació de dades (`SELECT`, `INSERT`, `DELETE`, `UPDATE`, `FETCH` o `COPY`) d'una transacció hagi estat executada.

11.3.- Bloquejos a nivell de taula.

PostgreSQL ofereix diferents tipus de bloqueig per controlar l'accés concurrent a les dades d'una taula. Totes les formes de bloqueig (excepte el tipus de bloqueig `ACCESS SHARE`), adquirits en una transacció es mantenen fins el final de la mateixa.

Dues transaccions no podent conservar bloquejos de tipus en conflicte sobre una mateixa taula en el mateix moment. No obstant això, una transacció no entre mai en conflicte amb ella mateixa.

TIPUS DE BLOQUEJOS A NIVELL DE TAULA		
ACCES SHARE	És un tipus de bloqueig adquirit automàticament sobre taules que estan sent consultades. PostgreSQL allibera aquests bloquejos després de que s'hagi executat una instrucció.	En conflicte amb el tipus de bloqueig ACCESS EXCLUSIVE
ROW SHARE	Adquirit per SELECT FOR UDPATE i LOCK TABLE per declaracions IN ROW SHARE MODE.	Entra en conflictes amb els tipus EXCLUSIVE i ACCESS EXCLUSIVE
ROW EXCLUSIVE	L'adquireixen UPDATE, DELETE, INSERT i LOCK TABLE per declaracions IN ROW EXCLUSIVE MODE.	Entra en conflicte amb els tipus SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE i ACCESS EXCLUSIVE.
SHARE UPDATE EXCLUSIVE	L'adquireix VACUUM (sense FULL). Aquest tipus protegeix una taula contra les modificacions concurrents d'esquema i l'execució de VACUUM.	En conflicte amb els tipus SHARE UPDATE EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE i ACCESS EXCLUSIVE.
SHARE	L'adquireixen CREATE INDEX i LOCK TABLE per declaracions IN SHARE MODE.	En conflicte amb els tipus ROW EXCLUSIVE, SHARE ROW EXCLUSIVE, EXCLUSIVE i ACCES EXCLUSIVE.
SHARE ROW EXCLUSIVE	L'adquireix LOCK TABLE per declaracions IN SHARE ROW EXCLUSIVE MODE.	Entre en conflicte amb els tipus de bloquejos ROW EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE, ACCESS EXCLUSIVE.

EXCLUSIVE	L'adquireix LOCK TABLE per declaracions IN ESCLUSIVE MODE.	Entre en conflicte amb els tipus ROW SHARE, ROW EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE i ACCESS EXCLUSIVE.
ACCESS EXCLUSIVE	L'adquireixen ALTER TABLE, DROP TABLE, VACUUM i LOCK TABLE.	Entra en conflicte amb ROW SHARE, ROW EXCLUSIVE, SHARE, SHARE ROW EXCLUSIVE, EXCLUSIVE i ACCESS EXCLUSIVE.

11.4.- Bloquejos a nivell de fila.

Aquest tipus de bloquejos es produeixen quan s'actualitzen camps interns d'una fila (o s'esborren o es marquen per ser actualitzats). PostgreSQL no reté en memòria cap informació sobre les files modificades.

Cal tenir en compte que SELECT FOR UPDATE modificarà les files seleccionades marcant-les de tal manera que s'escriguin en el disc.

Els bloquejos a nivell de fila no afecta les dades consultades. Aquest s'utilitzen sols per bloquejar escriptures a la mateixa fila.

11.5.- Comparativa envers altres productes comercials.

La concurrència és un dels punts forts de PostgreSQL amb relació amb altres SGBD. La majoria de SGBD utilitzen bloquejos per el control de la concurrència.

La principal diferència entre multiversió i el model de bloqueig és que els bloquejos MVCC derivats d'una consulta no entren en conflicte amb els bloquejos derivats de l'escriptura de dades i d'aquesta manera la consulta no bloqueja l'escriptura i aquesta mai bloqueja la consulta.

La majoria de SGBD utilitzen el sistema de 'semàfors' pel control de concurrència. Quan es fa una lectura es bloqueja la informació que es va a llegir i ningú pot escriure fins que s'allibera. Quan s'escriu es fa el mateix i ningú pot llegir o escriure fins que s'allibera.

Per exemple MySQL realitza els bloquejos d'escriptura a nivell de taula, perjudicant la concurrència d'insercions en la base de dades.

Entre els SGBD de lliure distribució com són MySQL, Interbase, SAPDB i PostgreSQL, sols aquest darrer implementa la concurrència multiversió i el SAPDB també però li cal un mòdul apart.

Oracle també implementa el control de concurrència multi-versió, en la seva versió Oracle 9i i en l'Oracle 10g. En Oracle els mecanismes subjacents són els arxius de control, nombre de canvis de sistema (SCN), i la forma d'utilitzar el `ROLLBACK` i `l'UNDO`.

12.- Usuaris i Privilegis.

Cada grup de bases de dades conté un conjunt d'usuaris. Aquests usuaris són diferents dels usuaris que administren el sistema operatiu en el qual corre el servidor. El usuaris són propietaris dels objectes de la bases de dades (per exemple, de les taules) i poden assignar privilegis dels seus objectes a altres usuaris per controlar qui accedeix als seus objectes.

12.1.- Usuaris de la base de dades.

Conceptualment, els usuaris de la base de dades estan totalment separats dels usuaris del sistema d'explotació.

Per crear un usuari s'utilitza la comanda `CREATE USER:`

```
CREATE USER nom_usuari [ [ WITH ] opcions [ ... ] ];
```

També s'utilitza `CREATEUSER.`

Les opcions poden ser:

```
SYSID ID_usuari
    | CREATEDB | NOCREATEDB
    | CREATEUSER | NOCREATEUSER
    | IN GROUP nom_grup [, ...]
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'abstime'
```

També tenim la possibilitat de crear grups amb la comanda `CREATE GROUP`, modificar-los amb `ALTER GROUP` i esborrar-los amb `DROP GROUP`. PostgreSQL difereix de l'SQL estàndard ja que aquest utilitza `CREATE ROLE`, per crear grups d'usuaris.

`abstime` Vol dir 'vàlid fins'. La clàusula posa un temps absolut després del qual la contrasenya de l'usuari ja no és vàlida. Si aquesta clàusula s'omet la contrasenya serà vàlida per sempre.

I per donar de baixa un usuari s'utilitza `DROP USER`:

```
DROP USER nom_usuari;
```

També s'utilitza `DROPUSER`.

Un usuari d'una base de dades pot tenir una sèrie d'atributs que defineix els seus privilegis i interactuació amb el sistema d'autentificació del client. Són atributs `CREATEUSER` o `CREATEDB`, que els confereix el permís de crear nous usuaris o crear bases de dades, respectivament.

Els atributs dels usuaris es poden modificar amb la comanda `ALTER USER`:

```
ALTER USER nom_usuari [ [ WITH ] opcions [ ... ] ]
```

I les opcions poden ser:

```
CREATEDB | NOCREATEDB
    | CREATEUSER | NOCREATEUSER
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'abstime'

ALTER USER nom RENAME TO nou_nom

ALTER USER nom SET paràmetre { TO | = } { valor | DEFAULT }
ALTER USER nom RESET paràmetre
```

Amb `SET` canvia la configuració de sessió per defecte d'un usuari per una configuració determinada.

Amb `RESET` es restaura la configuració per defecte.

12.2.- Privilegis dels usuaris.

Quan es crea un objecte aquest és assignat a un propietari. El propietari normalment és el mateix usuari que ha executat la comanda de creació.

Per a la majoria dels objectes, l'estat inicial és aquell en que el propietari (o un superusuari) pot fer alguna cosa amb aquest objecte. Per tal de deixar a altres usuaris utilitzar l'objecte, cal atorgar-li privilegis. Existeixen diferents privilegis: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `RULE`, `REFERENCES`, `TRIGGER`, `CREATE`, `TEMPORARY`, `EXECUTE` i `USAGE`.

Per exemple si el privilegi és `RULE` o `TRIGGER` vol dir que l'usuari pot crear regles o *triggers* en la taula especificada.

Per tal d'assignar privilegis s'utilitza la comanda `GRANT`. El `GRANT` té les possibilitats següents:

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | RULE | REFERENCES |
TRIGGER }
    [,...] | ALL [ PRIVILEGES ] }
    ON [ TABLE ] nom_taula [, ...]
    TO { nom_usuari | GROUP
Nom_grup | PUBLIC } [, ...] [ WITH
GRANT OPTION ]

GRANT { { CREATE | TEMPORARY | TEMP } [,...] | ALL [ PRIVILEGES ]
}
    ON DATABASE nom_bd [, ...]
    TO { nom_usuari | GROUP
Nom_grup | PUBLIC } [, ...] [ WITH
GRANT OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON FUNCTION nom_funció
([type, ...]) [, ...]
    TO { nom_usuari | GROUP
Nom_grup | PUBLIC } [, ...] [ WITH
GRANT OPTION ]

GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE nom_llenguatge [, ...]
    TO { nom_usuari | GROUP
Nom_grup | PUBLIC } [, ...] [ WITH
GRANT OPTION ]

GRANT { { CREATE | USAGE } [,...] | ALL [ PRIVILEGES ] }
    ON SCHEMA nom_esquema [, ...]
    TO { nom_usuari | GROUP
Nom_grup | PUBLIC } [, ...] [ WITH
GRANT OPTION ]

GRANT { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE nom_espai_lògic [, ...]
    TO { nom_usuari | GROUP
Nom_grup | PUBLIC } [, ...] [ WITH
GRANT OPTION ]
```

On `PUBLIC` significa que els drets són donats a tots els usuaris. Inclòs aquells que es puguin crear posteriorment.

`WITH GRANT OPTION` significa que el que té aquest privilegi el pot transferir a altres usuaris.

`ALL PRIVILEGES` significa que li dóna tots els drets disponibles de l'objecte de cop.

Un objecte pot ser assignat a un nou usuari amb la comanda `ALTER`. Sols el súper usuari pot canviar els privilegis dels altres usuaris.

Per eliminar els drets d'un usuari o grups d'usuaris s'utilitza `REVOKE`:

`REVOKE` té la següent sintaxi:

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | RULE | REFERENCES |
TRIGGER }
    [,...] | ALL [ PRIVILEGES ] }
    ON [ TABLE ] nom_taula [, ...]
    FROM { nom_usuari | GROUP
    nom_grup | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { CREATE | TEMPORARY | TEMP } [,...] | ALL [ PRIVILEGES ] }
    ON DATABASE nom_bd [, ...]
    FROM { nom_usuari | GROUP
    nom_grup | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { EXECUTE | ALL [ PRIVILEGES ] }
    ON FUNCTION nom_funció
([type, ...]) [, ...]
    FROM { nom_usuari | GROUP
    nom_grup | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE nom_llenguatge [, ...]
    FROM { nom_usuari | GROUP
    nom_grup | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { CREATE | USAGE } [,...] | ALL [ PRIVILEGES ] }
    ON SCHEMA nom_esquema [, ...]
    FROM { nom_usuari | GROUP
    nom_grup | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE nom_espai_lògic [, ...]
    FROM { nom_usuari | GROUP
```

```
nom_grup | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

En el cas de `REVOKE GRANT OPTION FOR` sols es revoca el dret a transferir el privilegi, no el privilegi mateix. Altrament, tant el privilegi com el dret a transferir-lo es revoquen.

Si s'està revocant el privilegi per l'usuari i existeixen privilegis dependents, si especifiquem `CASCADE` aquells privilegis dependents també es revocaran.

PostgreSQL assumeix `RESTRICT` per defecte i en aquest cas no revoca el privilegi per l'usuari si n'hi ha que en depenen.

13.- Subllenguatges especialitzats

PostgreSQL permet accedir a la base de dades des d'una aplicació feta amb algun llenguatge de programació del tipus C, C++ o Java. És el anomenat '*embedded SQL'* (SQL hostatjat) utilitzant un precompilador o SQL/CLI (*SQL/Call-Level Interface*), sense l'ús d'un precompilador.

13.1.- SQL hostatjat.

Un programa de SQL hostatjat consta de codi escrit en un llenguatge de programació corrent, per exemple en C, i ordres de SQL en seccions marcades de manera especial. Per construir el programa, el codi font es passa primer a través del preprocessador de SQL incrustat, que el converteix en un programa corrent, i després pot ser processat per un compilador C.

En aquest cas els programes escrits per a la interfície de SQL hostatjar són programes C normals amb codi especial introduït per realitzar accions relacionades de base de dades. Aquest codi especial sempre comença per la forma:

```
EXEC SQL ...;
```

Podem emprar sentències SQL dintre de les nostres aplicacions amb llenguatge Java amb l'SQLJ (*Structured Query Language embedded in Java*) o amb llenguatge C, amb l'ECPG (*Embedded SQL in C*), per exemple.

Per connectar-se a la base de dades s'utilitza:

```
EXEC SQL CONNECT TO target [AS nom_connexió] [USER usuari];
```

target es pot especificar en els següents casos:

- nom_BD [@hostname][:port]
- tcp:postgresql://hostname[:port][/nom_BD][?opcions]
- unix:postgresql://hostname[:port][/ nom_BD][?opcions]
- Un literal SQL contenint un dels anteriors formats.
- Una referència a una variable contenint un dels anterior formats.
- DEFAULT

Per tancar la connexió s'utilitza:

```
EXEC SQL DISCONNECT [connexió];
```

connection es pot especificar en els següents casos:

- *Nom_connexió*
- DEFAULT
- CURRENT
- ALL

D'aquesta manera es poden utilitzar totes les sentències del tipus CREATE TABLE, CREATE INDEX, INSERT INTO, DELETE FROM, etc.

13.2.- SQL/CLI.

És una tècnica de SQL programat que segueix l'enfocament interpretat.

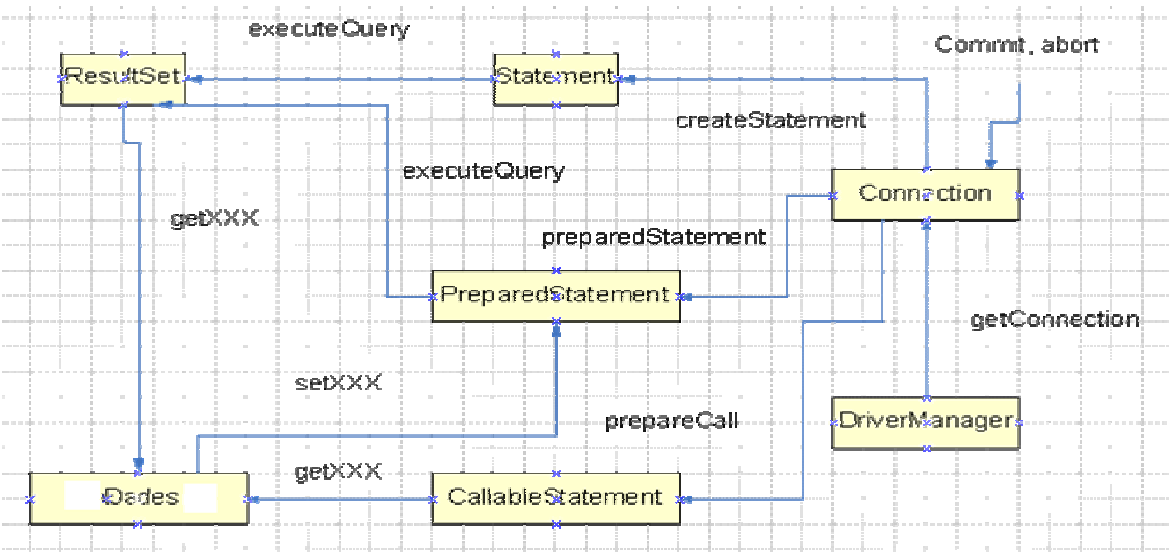
Les SQL/CLI permeten incloure sentències SQL mitjançant crides a llibreries, a aplicacions desenvolupades en un cert llenguatge de programació. Amb les SQL/CLI no cal precompilar el programa ja que les sentències SQL s'interpreten en temps d'execució.

L'interfície ODBC (*Open Database Connectivity*) defineix una llibreria de funcions per permetre l'accés de les aplicacions a la base de dades utilitzant SQL. ODBC està implementat en C. Java ha adaptat aquest sistema i el resultat ha estat l'API (*Aplication Program Interface*) estàndard de Java per treballar amb SQL, el JDBC (*Java Database Connectivity*).

El JDBC és una especificació d'un conjunt de classes i interfícies i cal disposar d'un *driver* (programa de control) per treballar-hi.

Un programa JDBC cal que localitzi i carregui el *driver*, connecti a l'SGBD, faci les operacions necessàries i es desconnecti de l'SGBD.

Esquema de funcionament:



Seguint amb el nostre exemple; partint de la taula 'província' que té codi de província i nom de la província, i fem un programa Java simple que sols es connecti a la base de dades UOCReserva i demani un codi de província a l'usuari. Retorna el nom corresponent a aquella província.

```
/*
*  programa exemple connexió JDBC
*/

import java.io.*;
import java.sql.*;

public class sample
{
  Connection  conn;             // guardar connexió a la BD
  Statement   stmt;             // guardar sentència SQL
  String      codi_prov;        // guardar codi provincia entrat pel usuari

  public sample() throws ClassNotFoundException, FileNotFoundException,
IOException, SQLException
  {
   Class.forName("org.postgresql.Driver");  // carrega la interface de la BD
                                            // conecta a la BD
   conn = DriverManager.getConnection("jdbc:postgresql:uocreserva", "postgres",
"postgres");
   stmt = conn.createStatement();

   System.out.print("Entra un codi de provincia:  ");      // demana el codi
provincia a l'usuari

   System.out.flush();
   BufferedReader r = new BufferedReader(new InputStreamReader(System.in));
   codi_prov = r.readLine();

   ResultSet res = stmt.executeQuery(              // envia la query
            "SELECT nom " +
            "FROM provincia " +
            "WHERE codi = '" + codi_prov + "'");

             if (res != null)
                 while(res.next())
                 {
                    String nom_prov = res.getString(1);
                     System.out.println(nom_prov);
                  }

            res.close();
            stmt.close();
            conn.close();
   }

   public static void main(String args[])
   {
      try {
         sample uocreserva = new sample();
      }  catch(Exception exc)
      {
         System.err.println("Exception caught.\n" + exc);
         exc.printStackTrace();
```

```
      }
    }
 }
```

En l'anterior exemple veiem la càrrega del *driver*, la connexió a la BD:

```
Class.forName("org.postgresql.Driver");  // carrega la interface de la BD
                                         // conecta a la BD
   conn = DriverManager.getConnection("jdbc:postgresql:uocreserva", "postgres",
"postgres");
   stmt = conn.createStatement();
```

l'enviament de la consulta:

```
ResultSet res = stmt.executeQuery(              // envia la query
            "SELECT nom " +
            "FROM provincia " +
            "WHERE codi = '" + codi_prov + "'");
```

La realització de les operacions de recorregut i impressió del resultat:

```
             if (res != null)
                 while(res.next())
                 {
                    String nom_prov = res.getString(1);
                     System.out.println(nom_prov);
                  }
```

I posteriorment fa tots els tancaments:

```
           res.close();
           stmt.close();
           conn.close();
```

Es poden veure més exemples en l'annex, en la pràctica de BDII de la BD UOCReserva.

14.- Comparació amb altres SGBD relacionals.

Les taules següents comparen informació general i tècnica de diferents SGBD.

14.1.- Informació general.

SGBD	Creador	data de la primera versió pública	Última versió estable	Llicencia de software
ANTs Data Server	ANTs Software	1999	3.0	Propietat
DB2	IBM	1982	8.2	Propietat
Firebird	Firebird Foundation	2000	1.5.2	Llicencia Pública InterBase
Informix	Informix Software	1985	10.0	Propietat
HSQLDB	Hsqldb.Org	2001	1.8.0	Llicencia BSD
Ingres	Berkeley University, Computer Associates	1980	r3 3.0.1	CA-TOSL
InterBase	Borland	1985	7.5.1	Propietat
Microsoft SQL Server	Microsoft	1989	8.00.2039 (2000 SP4)	Propietat
MySQL	MySQL AB	1996	4.1	GPL o Propietat
Oracle	Oracle Corporation	1977	10g Release 2	Propietat
PostgreSQL	PostgreSQL Global Development Group	1989	8.0.3	Llicencia BSD
SQLite	D. Richard Hipp	2000	3.1.3	Domini públic

14.2.- Sistema operatiu suportat.

SGBD	Windows	Mac OS X	Linux	BSD	Unix
ANTs Data Server	Sí	Sí	Sí	Sí	Sí
DB2	Sí	No	Sí	No	Sí
Firebird	Sí	Sí	Sí	Sí	Sí
HSQLDB	Sí	Sí	Sí	Sí	Sí
Informix	Sí	Sí	Sí	Sí	Sí
Ingres	Sí	?	Sí	?	Sí

InterBase	Sí	No	Sí	No	Sí (Solaris)
Microsoft SQL Server	Sí	No	No	No	No
MySQL	Sí	Sí	Sí	Sí	Sí
Oracle	Sí	Sí	Sí	No	Sí
PostgreSQL	Sí	Sí	Sí	Sí	Sí
SQLite	Sí	Sí	Sí	Sí	Sí

14.3.- Característiques fonamentals.

SGBD	ACID	Integritat referencial	Transaccions	Unicode
ANTs Data Server	Sí	Sí	Sí	Sí
DB2	Sí	Sí	Sí	Sí
Firebird	Sí	Sí	Sí	Sí
HSQLDB	Sí	Sí	Sí	?
Informix	Sí	Sí	Sí	Sí
Ingres	Sí	Sí	Sí	Sí
InterBase	Sí	Sí	Sí	Sí
Microsoft SQL Server	Sí	Sí	Sí	Sí
MySQL	Depèn [1]	Depèn [1]	Depèn [1]	Sí
Oracle	Sí	Sí	Sí	Sí
PostgreSQL	Sí	Sí	Sí	Sí
SQLite	Sí	No [2]	Bàsic [2]	Sí

Nota (1): Per les transaccions i la integritat referencial, cal utilitzar el tipus de taula InnoDB; el tipus de taula per defecte, MyISAM, no suporta aquestes característiques. Encara que, el tipus de taula InnoDB també permet l'emmagatzemament de valors que excedeixin el rang de dades; algunes vistes violen la limitació d' ACID.

Nota (2): Les limitacions de CHECK i FOREIGN KEY són analitzades però no forçades.

14.4.- Taules i vistes.

SGBD	Taula temporal	Vista materialitzada
ANTs Data Server	Sí	Sí
DB2	Sí	Sí
Firebird	No	No
HSQLDB	Sí	No

Informix	Sí	Sí
Ingres	Sí	No
InterBase	Sí	No
Microsoft SQL Server	Sí	Similar [4]
MySQL	Sí	No
Oracle	Sí	Sí
PostgreSQL	Sí	No [3]
SQLite	Sí	No

Nota (3): La vista materialitzada pot ser emulada amb PL/PgSQL.

Nota (4): El servidor MS SQL proveeix vistes indexades.

14.5.- Tipus d'índex.

SGBD	Arbre R-/R+	Hash	Expressió	Parcial	Reversal	Mapa de bits
ANTs Data Server	Sí	Sí	Sí	Sí	Sí	Sí
DB2	No	?	No	No	Sí	Sí
Firebird	No	No	No	No	No	No
Informix	Sí	Sí	Sí	No	No	No
Ingres	Sí	Sí	No	No	No	No
InterBase	?	?	No	No	No	No
Microsoft SQL Server	?	?	No	No	No	No
MySQL	Sols taules MyISAM	Sols taules HEAP	No	No	No	No
Oracle	Sols Edició EE	?	Sí	No	Sí	Sí
PostgreSQL	Sí	Sí	Sí	Sí	No	No
SQLite	No	No	No	No	No	No

14.6.- Altres Objectes.

SGBD	Domini	Cursor	Trigger	Funció [6]	Procediment [6]	Rutina externa [6]
ANTs Data Server	Sí	Sí	Sí	Sí	Sí	Sí
DB2	No	Sí	Sí	Sí	Sí	Sí

Firebird	Sí	Sí	Sí	Sí	Sí	Sí
HSQLDB	?	No	Sí	Sí	Sí	Sí
Informix	No	Sí	Sí	Sí	Sí	Sí
Ingres	Sí	Sí	Sí	Sí	Sí	?
InterBase	Sí	Sí	Sí	Sí	Sí	Sí
Microsoft SQL Server	No	Sí	Sí	Sí	Sí	Sí
MySQL	No	No [5]	No [5]	No [5]	No [5]	Sí
Oracle	Sí	Sí	Sí	Sí	Sí	Sí
PostgreSQL	Sí	Sí	Sí	Sí	Sí	Sí
SQLite	No	No	Sí	No	No	Sí

Nota (5): Aquests objectes de base de dades estan disponibles sols en MySQL 5.0, que és una versió experimental.

Nota (6): **Funció i procediment** es refereixen a les rutines internes escrites en SQL i/o llenguatges procedimentals como PL/SQL. **Rutina externa** es refereix a la escriptura en els llenguatges amfitrions como C, Java, Cobol, etc. "Procediment emmagatzemat" s'utilitza normalment per aquest tipus de rutines però la seva definició vària entre diferents venedors de bases de dades.

14.7.- Particionament.

SGBD	Rang	Hash	Compost (Rang+Hash)	Llista
ANTs Data Server	Sí	Sí	Sí	Sí
DB2	Sí	Sí	Sí	Sí
Firebird	No	No	No	No
Ingres	Sí	Sí	Sí	Sí
InterBase	No	No	No	No
Microsoft SQL Server	Sí	No	No	No
MySQL	No	No	No	No
Oracle	Sí	Sí	Sí	Sí
PostgreSQL	No	No	No	No
SQLite	No	No	No	No

15.- Glossari

ACID: Propietats que ha de complir tota transacció: Atomicitat, Consistència, Isolament, Definitivitat.

ANSI: *American National Standards Institute.*

API: *Aplication Program Interface* (Interfície d'aplicació).

Benchmarks: Comparatives de rendiments d'un sistema, de software o de hardware.

BD: Base de Dades.

BSD: Tipus de llicencia d'ús de programari molt similar a la llicencia d'us GNU.

CSV: *Selects Comma Separated Value* (Selecció de coma com a valor de separació).

Driver: Programa de control. Implementació particular del JDBC.

ECPG: *Embedded SQL in C.* (SQL hostatjat en C).

GNU: Tipus de llicencia d'ús de programari, per la que es permet copiar, modificar, distribuir els programes.

ISO: *International Standards Organization.*

JDBC: *Java Database Connectivity* (Connexió a la BD Java).

MVCC: *Multi-Version Concurrency Control* (Control de la Concurrència Multi Versió).

ODBC: *Open Database Connectivity* (Apertura de la connexió a la BD).

OIDs: *Object Identifiers* (Identificadors d'objecte).

pgAdmin: Eina per treballar amb PostgreSQL, incorporat en el paquet d'instal·lació amb editor d'SQL.

pgAcces: Eina per treballar amb PostgreSQL.

PL/pgSQL: Llenguatge procedural de programació de funcions per PostgreSQL.

psql: És la consola interactiva de PostgreSQL.

SGBD: Sistema de Gestió de Bases de Dades.

SQL: *Structured Query Language*. Llenguatge de consulta estructurat, és l'estàndard ANSI/ISO de definició, manipulació i control de bases de dades relacionals.

SQL/CLI: *SQL/Call-Level Interface* (Interficie a Nivell de Crida SQL).

SQLJ: *Structured Query Language embedded in Java*. (Llenguatge de consulta estructurat hostatjar en Java).

Squirrel: Programa client per a executar consultes SQL sobre SGBD.

Trigger: Disparador**.** És un programa que s'emmagatzema en el servidor i que s'executa en ocórrer un esdeveniment que prèviament s'ha definit.

16.- Bibliografia

Llibres

- B. Stinson. PostgreSQL Essential Reference. Ed. New Riders. 2001.
- J. C. Worsley and J. D. Drake. Practical PostgreSQL.Ed. O'Relilly. 2002.
- J. Melton and A. Eisenberg. Understanding SQL and Java Together: A guide to SQJL, JDBC and other related Technologies. Ed. Morgan Kaufmann Publishers. 2000.
- K. Douglas and S. Douglas. PostgreSQL. A comprehensive guide to building, programming, and administering PostgreSQL databases. Ed. Sams. 2003.
- PostgreSQL Development Team, PostgreSQL Programmer's Guide. Ed. Thomas Lochart. 2000.

Enllaços

- http://www.postgresql.org
- http://technodocs.postgresql.org
- http://jdbc.postgresql.org/
- http://es.tldp.org/Articulos-periodisticos/jantonio/odbc/odbc3.html
- http://pginstaller.projects.postgresql.org/faq/FAQ_windows.html
- http://www.pgaccess.org/
- http://squirrelsql.org/
- http://es.tldp.org/Postgresql-es/web/navegable/user/mvcc.html
- http://candle.pha.pa.us/main/writings/pgsql/sgml/index.html
- http://www.cygwin.com/

Annexos

- Implementació en PostgreSQL de la pràctica de BDI.
- Implementació en PostgreSQL de la pràctica de BDII.

Implementació en PostgreSQL de la pràctica de BDI.

Pràctica de Disseny de BD

Presentació i objectius

Objectius

El principal objectiu d'aquesta pràctica és fixar els coneixements adquirits després de l'estudi del mòdul 6 sobre el disseny de bases de dades. Així, l'estudiant ha de saber dissenyar una base de dades i ha d'estar preparat per crear i manipular una base de dades a través del llenguatge SQL, un cop finalitzada la realització d'aquesta pràctica.

Presentació de la pràctica

Aquesta pràctica constitueix la primera part de la pràctica de l'assignatura. Es demana realitzar a partir de l'enunciat el disseny conceptual de la base de dades utilitzant el model Entitat-Interrelació i la transformació al corresponent model lògic de la base de dades relacional.

Enunciat

Un país exòtic, del qual no tenim constància del nom, ha decidit d'una vegada per totes intentar salvar la innumerable fauna salvatge que hi habita al seu territori. Per això s'ha creat una reserva animal en la que podran viure de forma natural un gran nombre d'animals, la UOCRESERVA.

Els responsables d'aquest engrescador projecte han decidit disposar d'una Base de Dades per emmagatzemar i gestionar la informació de la reserva. Per això, han decidit encomanar aquesta tasca als estudiants de l'assignatura de Bases de Dades I de la UOC.

La Base de Dades ha de mantenir tota la informació respecte al que es descriu a continuació:

A la reserva hi ha una gran quantitat d'animals, els quals han d'estar classificats. De cadascun dels animals s'ha de guardar la següent informació: nom comú, número d'anys d'esperança de vida i pes mitjà. Es desitja identificar els animals amb un codi numèric únic.

Els animals de la reserva es pot dividir , almenys, en animals carnívors i animals herbívors. Dels carnívors es vol controlar la quantitat diària de carn que necessiten i quins animals formen part de la seva dieta alimentària. A més, per cada animal que pot formar part de la dieta d'un altre, es necessita conèixer, mitjançant un valor entre 1 i 10, el grau de predilecció que té el carnívor per ell. En canvi, dels herbívors, només es necessita registrar la quantitat d'aigua que necessiten diàriament.

Per poder assumir les grans despeses que implica el manteniment de la reserva, els responsables volen treure profit de l'explotació turística d'aquesta. De forma sostenible, això si. A la reserva hi ha uns determinats llocs que estan destinats a la acampada i hospedatge de turistes que venen a veure els animals. En concret hi ha dos tipus de llocs diferents: zones de càmping i lodges (petits hotels), amb habitacions totalment condicionades i amb tota mena de luxes. Dels càmpings es desitja saber el nombre de places de les que disposa cadascun, i dels lodges, el nombre d'habitacions dobles i individuals. A més, de tots els llocs, es desitja conèixer el preu base per nit, el nom, el telèfon i un codi que els identifica.

Els llocs es troben a una zona concreta. Les zones s'identifiquen per un codi i també disposen d'un nom.

En cada lloc d'hospedatge poden haver-hi un o varis pous artificials d'aigua. Això fa possible que a les temporades seques els turistes puguin veure animals des del mateix lloc on s'hospeden, tot aprofitant que els animals van a beure aigua. Els responsables volen controlar quins pous hi ha en cada lloc d'hospedatge. Els pous tenen un codi que únicament permet identificar pous diferents d'un mateix lloc d'hospedatge, i estan

classificats de millors a pitjors depenent de l'afluència d'animals que tenen. Aquesta classificació es fa mitjançant un valor numèric entre 1 i 3. A més, és normal que a cada pou hi acudeixin uns animals més o menys periòdicament, per tant, s'ha de tenir constància dels animals que es poden veure a cadascun dels pous a cada temporada (exemple de temporada podria ser primavera o estiu) i amb quina probabilitat. Tot això els hi ajuda a gestionar els preus per nit de cada lloc.

Vigilant la reserva hi ha també uns guardes que s'encarreguen de que es compleixin les normes. Aquests guardes detecten les possibles incidències, que estan identificades per un codi i tenen associada una descripció. Exemples de descripció serien: localització de furtius, animals morts, descobriment de trampes, detecció de turistes infringint les normes, etc. Per cadascuna de les incidències detectades, els guardes realitzen un informe en el que, a més de la incidència i el guarda que la detecta, s'emmagatzema la data i zona en que es produeix la incidència. S'ha de tenir emmagatzemada la informació històrica de totes les incidències detectades a la reserva.

Dels guardes s'ha d'emmagatzemar el seu nom, un codi identificador, el telèfon, el domicili i els anys d'antiguitat a la reserva. A més a mes, els guardes s'organitzen jeràrquicament, de tal manera que cada guarda té un superior (o cap) directe. Es vol conèixer, per cada guarda, qui és el seu superior.

A la UOCReserva s'organitzen una sèrie d'excursions (safaris). Aquestes excursions tenen un codi que les identifica. A més, cada safari es pot organitzar només durant un cert període de temps, per això es vol enregistrar la data d'inici i la data final de realització de cada safari.

A cadascun dels safaris hi ha un guia assignat. Dels guies s'ha d'emmagatzemar el seu nom, un codi identificador, el telèfon, el domicili i els anys d'antiguitat a la reserva. A més s'ha de guardar: país de procedència i especialitat (caçador, expert en rastreig, conductor, etc). Els guies no poden exercir de guardes (ni a la inversa).

Per patrocinar la reserva, s'organitzen ral·lis fotogràfics. Hi participen fotògrafs professionals de tot el món. Un ral·li (que queda identificat per un codi) agrupa una sèrie de safaris. Cada ral·li té assignat un preu, una data d'inici i una data de finalització. Res impedeix que un mateix safari formi part de diversos ral·lis. Els fotògrafs s'inscriuen en un o diversos ral·lis, i se'ls pot aplicar un descompte en la inscripció que cal enregistrar.

Els fotògrafs s'identifiquen pel seu passaport, i es vol guardar el nom, adreça, telèfon, país d'origen i, opcionalment el nom de l'empresa que representen.

Cada fotògraf inscrit en un ral·li pot optar al concurs de fotografia del ral·li. Per poder optar al concurs de cada ral·li, els fotògrafs necessiten participar com a mínim a la meitat dels safaris que composen cada ral·li i lliurar com a mínim una fotografia. Les fotografies es realitzen durant les participacions dels fotògrafs als safaris dels ral·lis i no és permès lliurar més d'una foto per participació. De les fotos guardem un codi identificador, un nom donat pel fotògraf, els punts que la UOCReserva atorga a la fotografia i a quina participació del fotògraf a un safari del ral·li s'ha pres la fotografia. No necessàriament els fotògrafs han de lliurar una fotografia a la finalització de cada participació a un safari del ral·li (recordeu, però, que si lliuren foto, només en poden lliurar una, la que el fotògraf considera que és la seva millor foto). Al final del ral·li, el fotògraf amb més punts, i que hagi participat a la meitat dels safaris del ral·li, és el guanyador d'un premi.

Es demana:

A) Realitzeu l'esquema E/R recollint tota la semàntica de l'enunciat. No us oblideu que cal fer una descripció dels atributs de cada entitat. Indiqueu clarament quins requisits no han quedat reflectits en l'esquema E/R que proposeu. Si heu realitzat alguna suposició semàntica addicional indiqueu-la també.

B) Obtenir el disseny lògic de la BD anterior, és a dir, transformar el model anterior al model relacional. Per a cada relació, caldrà indicar els diferents atributs que incorpora i quines són les claus primàries, alternatives (si existeixen) i foranes (si existeixen). Indiqueu quins atributs poden prendre valor nul.

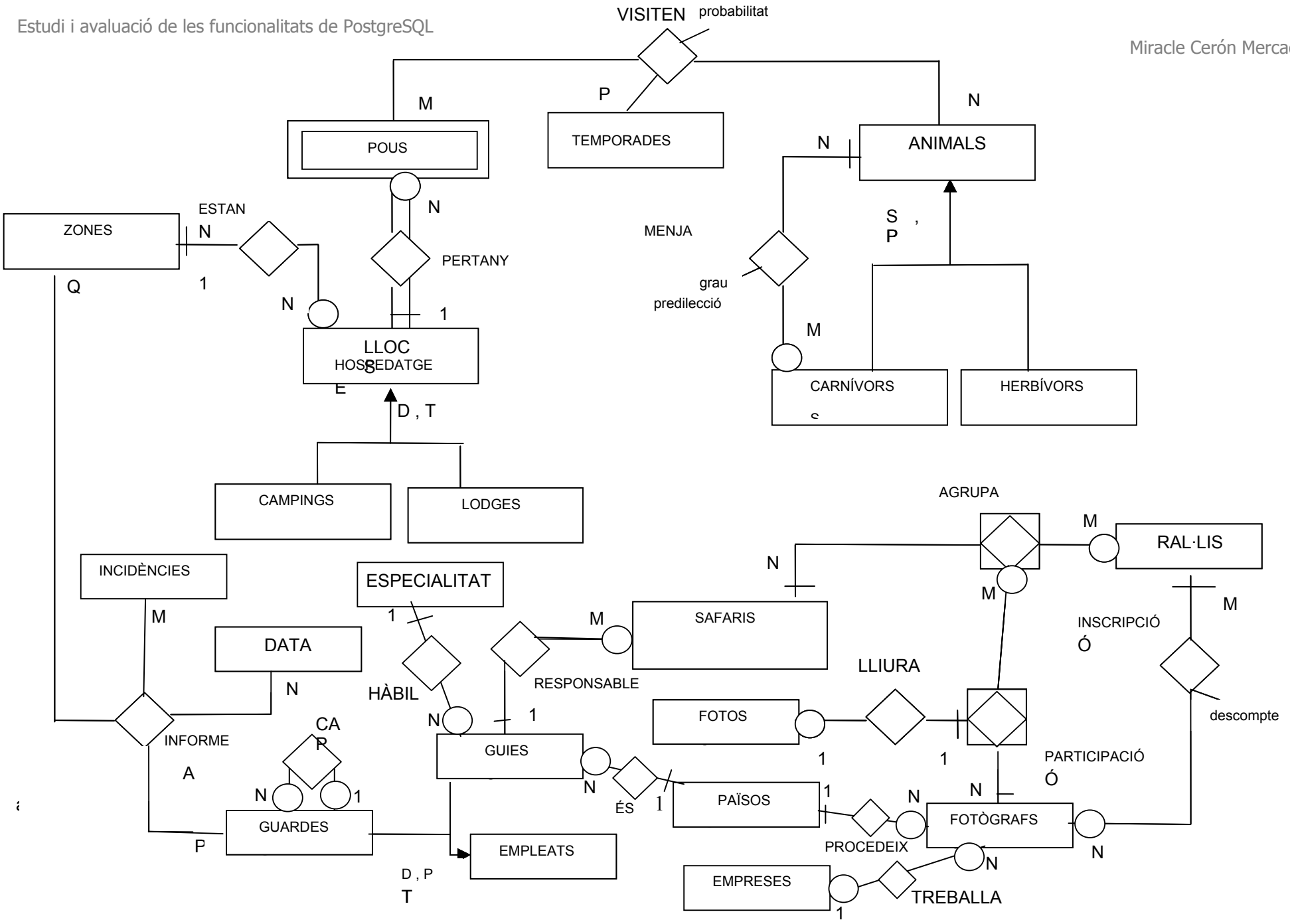
Important:

- Podeu consultar pràctiques de semestres anteriors per fer-vos una idea exacta del que cal lliurar a cada apartat.
- L'esquema E/R s'ha de realitzar utilitzant la notació gràfica especificada en el mòdul 6 de la documentació de l'assignatura.

Materials

Per a la realització d'aquesta PRA1 és suficient amb utilitzar la documentació en paper del mòdul 6 i la bibliografia que es proposa en el mateix.

SOLUCIÓ:


VISITEN
probabilitat
M
P
N
POUS
TEMPORADES
ANIMALS
N
ESTAN
N
1
ZONES
Q
N
PERTANY
N
1
MENJA
grau
predilecció
S
P
M
LLOC
HOSPEDATGE
E
D , T
CARNÍVORS
HERBÍVORS
CAMPINGS
LODGES
AGRUPA
M
RAL·LIS
N
M
M
INCIDÈNCIES
M
ESPECIALITAT
1
M
SAFARIS
INSCRIPCIÓ
Ó
M
DATA
N
HÀBIL
N
1
RESPONSABLE
LLIURA
FOTOS
descompte
INFORME
A
CA
P
GUIES
1
1
PARTICIPACIÓ
Ó
N
1
N
N
ÉS
1
PAÏSOS
1
N
N
GUARDES
1
FOTÒGRAFS
P
PROCEDEIX
N
N
EMPLEATS
D , P
T
EMPRESES
N
TREBALLA
1

Entitats i atributs de l'esquema E/R

Animals
codi_animal, nom_comú, esperança_vida, pes_mitjà

AnimalsCarnívors (entitat subclasse d'Animals)
codi_carnívor, quantitat_carn

AnimalsHerbívors (entitat subclasse d'Animals)
codi_herbívor, quantitat_aigua

LlocsHospedatge
codi_lloc, nom_lloc, telèfon, preu_nit

Campings (entitat subclasse de LlocsHospedatge)
codi_camping, num_places

Lodges (entitat subclasse de LlocsHospedatge)
codi_lodge, num_hab_dobles, num_hab_individuals

Zones
codi_zona, nom_zona

Pous (Entitat dèbil de LlocsHospedatge)
codi_pou, classificació

Temporades
nom_temporada

Empleats
codi_empleat, nom, domicili, telèfon, anys_antiguetat

Guardes (entitat subclasse d'Empleats)
codi_guarda

Guies (entitat subclasse d'Empleats)
codi_guia

Fotògrafs
passaport, nom, domicili, telèfon

Empreses
nom_empresa

Països
nom_país

Especialitats
nom_especialitat

Data
data

Ral·lis
codi_ral·li, preu, data_inici, data_final

Safaris
codi_safari, data_inici, data_final

Fotos
codi_foto, nom, puntuació

Incidències
codi_incidència, descripció

Justificació a la solució proposada

- Comentaris en relació al tipus de les especialitzacions:
    - Animals: de l'enunciat s'infereix que ha de ser parcial ("Els animals de la reserva es poden dividir, **almenys**, en animals carnívors i herbívors"). Hem decidit que poden existir animals omnívors, per tant, l'especialització és encavalcada.
    - Empleats: de l'enunciat s'infereix que és disjunta ("Els guies **no poden** exercir de guardes (**ni a la inversa**)"). Hem decidit que sigui parcial, és a dir, assumim que podem tenir enregistrades dades d'altres tipus d'empleats.

- Totes les especialitzacions són necessàries:
    - La d'animals degut a l'existència de la interrelació MENJA que només s'aplica als carnívors).
    - La de llocs d'hospedatge i empleats perquè tenen atributs i/o interrelacions diferenciades.
- Els fotògrafs no poden compartir una jerarquia d'herència amb els empleats (aquesta jerarquia seria una jerarquia de persones), donat que els darrers s'identifiquen per un codi donat per nosaltres i els fotògrafs pel seu número de passaport.
- No hem modelat les restriccions de domini (p.e. "per cada animal que pot formar part de la dieta d'un altre, es necessita conèixer, mitjançant **un valor entre 1 i 10**, el grau de predilecció que té el carnívor per ell").

- La interrelació INFORME, depenent del gràNul de temps que s'hagués decidit, podria haver tingut una cardinalitat diferent. Assumim que només guardem la data (dia-mes-any), per això la interrelació és Q:M:N:P. Si haguéssim guardat una data que incorporés l'hora i els minuts, llavors, la cardinalitat hauria estat 1:M:N:P (un guarda en un instant concret prou precís pot detectar una incidència en una **única zona**).

- Les entitats PAÏSOS, EMPRESES i ESPECIALITATS i les seves corresponents interrelacions, a manca de dades addicionals que no figuren a l'enunciat, podien haver estat substituïdes per atributs (definint en el moment de creació de la BD el domini de valors vàlids per a cada atribut) a les entitats GUIES i FOTÒGRAFS.
- En relació als ral·lis:
    - No hem pogut modelar la restricció "els fotògrafs necessiten participar com a mínim a la meitat dels safaris que composen cada ral·li".
    - Tampoc hem modelat quin és el fotògraf guanyador d'un determinat ral·li. Per saber-ho, cal efectuar una consulta a les dades.
    - No hem pogut modelar que les participacions dels fotògrafs a safaris de ral·lis siguin a ral·lis on el fotògraf s'hagi inscrit.

B) Transformació al model relacional del model conceptual obtingut a l'apartat anterior.

(**atenció**: claus primàries subratllades i atributs que poden pendre valors NULL com a conseqüència de l'enunciat de la pràctica en lletra *cursiva*)

ENTITATS

Animals (codi_animal, nom_comú, esperança_vida, pes_mitjà)

AnimalsCarnívors (codi_carnívor, quantitat_carn)
{codi_carnívor} clau forana cap a Animals(codi_animal)

AnimalsHerbívors (codi_herbívor, quantitat_aigua)
{codi_herbívor} clau forana cap a Animals(codi_animal)

LlocsHospedatge (codi_lloc, nom_lloc, telèfon, preu_nit, codi_zona)
{codi_zona} clau forana cap a Zones(codi_zona)

Campings (codi_camping, num_places)
{codi_camping} clau forana cap a LlocsHospedatge(codi_lloc)

Lodges (codi_lodge, num_hab_dobles, num_hab_individuals)
{codi_lodge} clau forana cap a LlocsHospedatge(codi_lloc)

Zones (codi_zona, nom_zona)

Pous (codi_lloc, codi_pou, classificació)
{codi_lloc} clau forana cap a LlocsHospedatge(codi_lloc)

Temporades (nom_temporada)

Empleats (codi_empleat, nom, domicili, telèfon, anys_antiguetat)

Guardes (codi_guarda, *codi_cap*)
{codi_guarda} clau forana cap a Empleats(codi_empleat)
{codi_cap} clau forana cap a Guardes(codi_guarda)

Guies (codi_guia, nom_país, nom_especialitat)
{codi_guia} clau forana cap a Empleats(codi_empleat)
{nom_país} clau forana cap a Països(nom_país)
{nom_especialitat} clau forana cap a Especialitats(nom_especialitat)

Fotògrafs (passaport, nom, domicili, telèfon, nom_país~~,~~ *nom_empresa*)
{nom_país} clau forana cap a Països(nom_país)
{nom_empresa} clau forana cap a Empreses(nom_empresa)

Empreses (nom_empresa)

Països (nom_país)

Especialitats (nom_especialitat)

Data (data)

Ral·lis (codi_ral·li, preu, data_inici, data_final)

Safaris (codi_safari, data_inici, data_final, codi_guia)
{codi_guia} clau forana cap a Guies(codi_guia)

Fotos (codi_foto, nom, puntuació)

Incidències (codi_incidència, descripció)

INTERRELACIONS

Visiten (codi_lloc, codi_pou, codi_animal, nom_temporada, probabilitat)
{codi_lloc,codi_pou} clau forana cap a Pous(codi_lloc,codi_pou)
{codi_animal} clau forana cap a Animals(codi_animal)
{nom_temporada} clau forana cap a Temporades(nom_temporada)

Menja (codi_animal, codi_carnívor, grau_predilecció)
{codi_animal} clau forana cap a Animals(codi_animal)
{codi_carnívor} clau forana cap a AnimalsCarnívors(codi_carnívor)

Informe(codi_zona, codi_incidència, codi_guarda, data)
{codi_zona} clau forana cap a Zones(codi_zona)
{codi_incidència} clau forana cap a Incidències(codi_incidència)
{codi_guarda} clau forana cap a Guardes(codi_guarda)
{data} clau forana cap a Data(data)

Inscripció (passaport, codi_ral·li, *descompte*)
{passaport} clau forana cap a Fotògrafs(passaport)
{codi_ral·li} clau forana cap a Ral·lis(codi_ral·li)

Agrupa (codi_ral·li, codi_safari)
{codi_ral·li} clau forana cap a Ral·lis(codi_ral·li)
{codi_safari} clau forana cap a Safaris(codi_safari)

Participació (passaport, codi_ral·li, codi_safari, *codi_foto*)
{passaport} clau forana cap a Fotògrafs(passaport)
{codi_ral·li,codi_safari} clau forana cap a Agrupa(codi_ral·li,codi_safari)
{codi_foto} clau forana cap a Fotos(codi_foto)

VARIANTS EN LA TRANSFORMACIÓ DEL MODEL CONCEPTUAL

Atenent al material (veure pàgina 36) Fotos i Participació admeten la següent alternativa de transformació:

Fotos (codi_foto, nom, puntuació, codi_ral·li, codi_safari)
{codi_ral·li, codi_safari} clau forana cap a Participació(codi_ral·li, codi_safari)

Participació (codi_fotògraf, codi_ral·li, codi_safari)
{codi_fotògraf} clau forana cap a Fotògrafs(passaport)
{codi_ral·li,codi_safari} clau forana cap a Agrupa(codi_ral·li,codi_safari)

També existeix una tercera alternativa vàlida que no està ressenyada al material:

Fotos (codi_foto, nom, puntuació)

Participació (codi_fotògraf, codi_ral·li, codi_safari)
{codi_fotògraf} clau forana cap a Fotògrafs(passaport)
{codi_ral·li,codi_safari} clau forana cap a Agrupa(codi_ral·li,codi_safari)

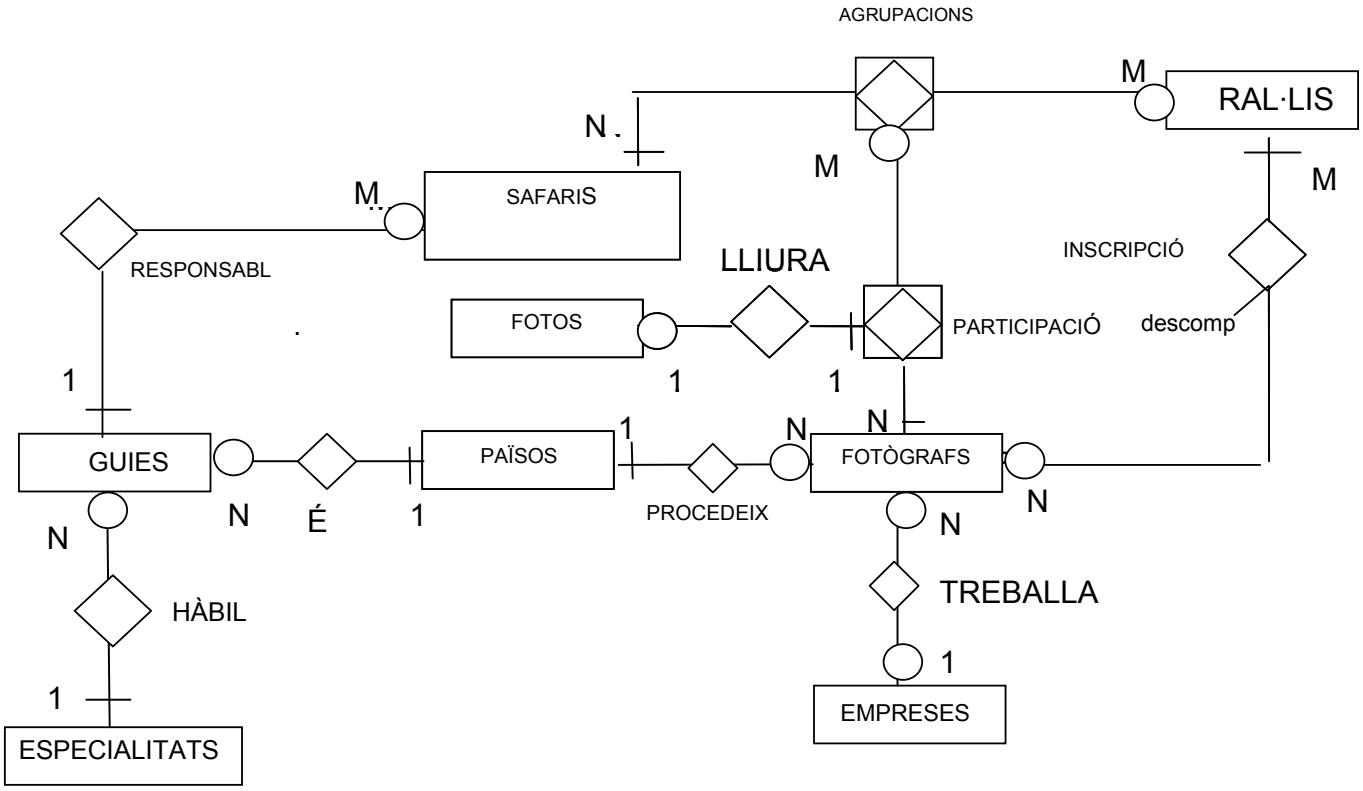
Lliura (codi_fotògraf, codi_ral·li, codi_safari, codi_foto)
{codi_fotògraf, codi_ral·li,codi_safari } clau forana cap a Participació (codi_fotògraf, codi_ral·li,codi_safari)
{codi_foto} clau forana cap a Fotos(codi_foto)
{codi_foto} és clau alternativa

Pràctica de Disseny i Manipulació de BD

Aquesta pràctica constitueix la segona part de la pràctica de l'assignatura. Es demana realitzar la creació d'una BD d'una part de l'esquema conceptual de la primera pràctica, incloent-hi les restriccions necessàries per a recollir la semàntica de l'enunciat. També es realitzaran insercions, consultes, creació de vistes, modificacions, i esborrats de les dades.

Enunciat

Donat **l'esquema conceptual** (parcial i lleugerament reformat respecte la primera part de la pràctica):



L'esquema lògic associat a l'esquema conceptual previ és el següent:

Guies (<u>codi_guia</u>, nom, domicili, telèfon, anys_antiguetat, nom_país, nom_especialitat)
{nom_país} clau forana cap a Països(nom_país)
{nom_especialitat} clau forana cap a Especialitats(nom_especialitat)

Fotògrafs (<u>passaport</u>, nom, domicili, telèfon, nom_país, *nom_empresa*)
{nom_país} clau forana cap a Països(nom_país)
{nom_empresa} clau forana cap a Empreses(nom_empresa)

Empreses (<u>nom_empresa</u>)

Països (nom_país)

Especialitats (nom_especialitat)

Ral·lis (<u>codi_ral·li</u>, preu, data_inici, data_final)

Safaris (<u>codi_safari</u>, data_inici, data_final, codi_guia)
{codi_guia} clau forana cap a Guies(codi_guia)

Fotos (codi_foto, nom, puntuació)

Inscripcions (<u>passaport, codi_ral·li,</u> descompte)
{passaport} clau forana cap a Fotògrafs(passaport)
{codi_ral·li} clau forana cap a Ral·lis(codi_ral·li)

Agrupacions (<u>codi_ral·li, codi_safari</u>)
{codi_ral·li} clau forana cap a Ral·lis(codi_ral·li)
{codi_safari} clau forana cap a Safaris(codi_safari)

Participacions (<u>passaport, codi_ral·li, codi_safari</u>, *codi_foto*)
{passaport} clau forana cap a Fotògrafs(passaport)
{codi_ral·li,codi_safari} clau forana cap a Agrupa(codi_ral·li,codi_safari)
{codi_foto} clau forana cap a Fotos(codi_foto)

NOTA: Al disseny lògic presentat s'ha decidit que la columna *'descompte'* de la taula 'Inscripcions' no pot prendre valor NULL. Per tant en cas de no aplicar un descompte a la inscripció d'un fotògraf a un ral·li, emmagatzemarem un 0.

A la BD es vol **inserir** el conjunt de dades següent:

NOTA: no s'han posat accents per evitar possibles problemes amb el joc de caràcters de les bases de dades que instal·leu.

Països

Nom País
Anglaterra
Alemanya
Argentina
Canada
Congo
Espanya
Estats Units
França
Japo
Namibia
Kenia
Portugal
Russia
Sudafrica

Safaris

Codi Safari	Data Inici	Data Final	Codi Guia
S01LL01	05-01-2004	10-01-2004	G01
S01LL02	07-01-2004	15-01-2004	G06
S01LL03	22-01-2004	29-01-2004	G04
S06LL01	03-06-2004	11-06-2004	G09
S06LL03	07-06-2004	14-06-2004	G04
S06LL05	19-06-2004	25-06-2004	G05
S06LL07	22-06-2004	29-06-2004	G01
S10LL02	09-10-2004	14-10-2004	G05
S10LL03	16-10-2004	21-10-2004	G01
S10LL04	17-10-2004	23-10-2004	G09
S10LL05	04-10-2004	11-10-2004	G08
S11LL01	02-11-2004	09-11-2004	G06
S11LL03	07-11-2004	13-11-2004	G07
S11LL06	15-11-2004	23-11-2004	G04
S11LL07	19-11-2004	28-11-2004	G03

Ral·lis

Codi Ral·li	Preu	Data Inici	Data Final
R2004R01	750	05-01-2004	15-01-2004
R2004C01	1100	05-01-2004	29-01-2004
R2004R06	925	01-06-2004	14-06-2004
R2004C06	1400	03-06-2004	30-06-2004
R2004R10	820	07-10-2004	22-10-2004
R2004C10	1200	04-10-2004	25-10-2004

Fotos

La puntuació de les fotos és de 1 a 10 punts

Codi Foto	Nom	Puntuació
F01	Savanna Sunset	3
F02	Vie, Vie, Vie !!!	5
F03	The Lion King	7
F04	Happy Family	2
F05	Green Power	6
F06	I want to stay here	4
F07	El Meu Planeta	9
F08	Agua de Vida	5
F09	Alive Paradise	7
F10	Explosion de Color	9
F11	Samba, Samba !!	7
F12	Eo	6
F13	Terra Madre	2
F14	Africa's Rules	4
F15	Wild Wind	5

Guies

Codi Guia	Nom	Domicili	Telèfon	Antig.	País	Especialitat
G01	Francis Cocoon	Kenya University	111111	2	Kenia	Zoologia
G02	Nwala Nimira	Namibia University	222222	2	Namibia	Rastreig
G03	John Walker	66, Queen Elisabeth	333333	1	Anglaterra	Espeleologia

G04	Enguele Lelu	43, Descartes St.	444444	6	Kenia	Orientacio
G05	René de la Court	Rue Savanna, 2	555555	10	Congo	Orientacio
G06	Pauline Prie	Rue. Victor Hugo, 12	666666	7	França	Zoologia
G07	Sanga Bole	54, Highway Ave.	777777	6	Kenia	Vigilant
G08	Andrew Philips	45, Johnsson St	888888	3	Sudafrica	Conductor
G09	Maria Llopis	Diagonal, 423	999999	8	Espanya	Biologia
G10	Tommey Eto	33, Livingstone Ave.	101010	4	Kenia	Rastreig

Especialitats

Nom Especialitat
Vigilant
Conductor
Biologia
Espeleologia
Rastreig
Domador
Orientacio
Zoologia

Fotògrafs

Passaport	Nom	Domicili	Telèfon	País	Empresa
12894578	Joao Sousa	Rua Algarve, 12	111111	Portugal	
59258743	Kenny Rogers	64, Smith St.	222222	Estats Units	Wild Life
09884323	Alain Poulet	Rue Guerlain, 34	333333	Canada	
23663233	Henrich Schoffer	Bonn Strasse 23	444444	Alemanya	Fotolux AG
56898743	Jaume Busquets	Consell de Cent, 2	555555	Espanya	Natura Viva
23434388	Brigitte Colette	Rue Latin 55	666666	França	Soleil Photo
09992344	Nimira Tao	Nikita 5	777777	Japo	
32456654	Andy Lucas	32 Waterloo Str.	888888	Anglaterra	BBC
09999823	Diego Pesini	Mayo, 23	999999	Argentina	Natura Viva

Empreses

Nom Empresa
Soleil Photo
Fotolux AG
Photo Industry
BBC
Wild Life
Natura Viva

Agrupacions (interrelació *Agrupa*)

Safaris que composen un Ral·li

Ral·li	Safari
R2004R01	S01LL01
R2004R01	S01LL02
R2004C01	S01LL01
R2004C01	S01LL02
R2004C01	S01LL03
R2004R06	S06LL01
R2004R06	S06LL03
R2004C06	S06LL01
R2004C06	S06LL03
R2004C06	S06LL05
R2004C06	S06LL07
R2004R10	S10LL02
R2004R10	S10LL03
R2004R10	S10LL05
R2004C10	S10LL02
R2004C10	S10LL03
R2004C10	S10LL04
R2004C10	S10LL05

Inscripcions

Fotògrafs que s'han inscrit als ral·lis

Passaport	Ral·li	Descompte
12894578	R2004R01	0
09884323	R2004R01	25
23663233	R2004R01	0
23434388	R2004R01	0
12894578	R2004C01	0
23434388	R2004C01	10
59258743	R2004C01	0
09884323	R2004C01	5
09884323	R2004R06	5
12894578	R2004R06	0
56898743	R2004R06	0
59258743	R2004R06	0
32456654	R2004R06	0
56898743	R2004C06	20
23663233	R2004C06	0
12894578	R2004C06	10
23434388	R2004R10	0
12894578	R2004R10	0
32456654	R2004R10	8
09999823	R2004R10	0
59258743	R2004C10	7
09884323	R2004C10	0
12894578	R2004C10	0
32456654	R2004C10	10
23663233	R2004C10	0
09999823	R2004C10	15
09992344	R2004C10	0

Participacions

Fotògrafs que participen en els safaris dels ral·lis i que opcionalment presenten fotos per al concurs de fotografia.

Passaport	Ral·li	Safari	Codi Foto
12894578	R2004R01	S01LL01	
12894578	R2004R01	S01LL02	
09884323	R2004R01	S01LL01	
09884323	R2004R01	S01LL02	
23663233	R2004R01	S01LL01	
23434388	R2004R01	S01LL01	
23434388	R2004R01	S01LL02	
23434388	R2004C01	S01LL01	F02
23434388	R2004C01	S01LL02	
23434388	R2004C01	S01LL03	
59258743	R2004C01	S01LL01	F04
59258743	R2004C01	S01LL02	
09884323	R2004C01	S01LL01	F10
09884323	R2004C01	S01LL02	
09884323	R2004C01	S01LL03	
09884323	R2004R06	S06LL01	F01
09884323	R2004R06	S06LL03	
56898743	R2004R06	S06LL01	F07
56898743	R2004R06	S06LL03	
59258743	R2004R06	S06LL01	F03
59258743	R2004R06	S06LL03	
32456654	R2004R06	S06LL01	F06
32456654	R2004R06	S06LL03	
56898743	R2004C06	S06LL01	
56898743	R2004C06	S06LL03	
56898743	R2004C06	S06LL05	
56898743	R2004C06	S06LL07	
23663233	R2004C06	S06LL01	
23663233	R2004C06	S06LL03	
23663233	R2004C06	S06LL05	

23663233	R2004C06	S06LL07	
12894578	R2004C06	S06LL05	
12894578	R2004C06	S06LL07	
23434388	R2004R10	S10LL02	
23434388	R2004R10	S10LL03	
32456654	R2004R10	S10LL05	
32456654	R2004R10	S10LL02	
32456654	R2004R10	S10LL03	
09999823	R2004R10	S10LL02	
09999823	R2004R10	S10LL03	
09999823	R2004R10	S10LL05	
59258743	R2004C10	S10LL02	
59258743	R2004C10	S10LL03	
59258743	R2004C10	S10LL04	
59258743	R2004C10	S10LL05	F05
09884323	R2004C10	S10LL02	F12
09884323	R2004C10	S10LL03	
09884323	R2004C10	S10LL04	
09884323	R2004C10	S10LL05	F08
32456654	R2004C10	S10LL02	F13
32456654	R2004C10	S10LL03	
32456654	R2004C10	S10LL04	
32456654	R2004C10	S10LL05	F09
23663233	R2004C10	S10LL02	F14
23663233	R2004C10	S10LL03	
23663233	R2004C10	S10LL04	
09999823	R2004C10	S10LL05	F11
09999823	R2004C10	S10LL02	
09999823	R2004C10	S10LL03	
09992344	R2004C10	S10LL04	
09992344	R2004C10	S10LL05	F15

Es demana:

1. Proposar el conjunt de sentències SQL necessàries per tal de definir i crear una BD sobre PostgreSQL que s'ajusti al disseny previ. És important destacar que **el nom de les taules i**

de les seves columnes ha de coincidir exactament amb els noms de les relacions i dels atributs que figuren a l'esquema lògic que es proporciona com a part d'aquest enunciat. Altrament, la vostra nota final de pràctiques es veurà penalitzada.

En cas que decidiu no implementar una relació determinada de l'esquema lògic, caldrà argumentar **breument** la vostra decisió.

A més a més, a l'hora de definir i crear la BD cal considerar les **restriccions següents**:

a) Totes aquelles restriccions que siguin inherents al model relacional (com, per exemple, claus primàries i claus foranes) i totes aquelles que es puguin deduir a partir de l'esquema conceptual previ (restriccions d'UNIQUE, admissió de valors null o no per un atribut o conjunt d'atributs etc.).
b) Les restriccions següents (ja sigui mitjançant *checks* o amb assercions):

b.1) Cal garantir que les participacions de fotògrafs a safaris de ral·lis siguin de ral·lis a on el fotògraf s'hagi inscrit.

b.2) En el cas d'aplicar un descompte a les inscripcions dels fotògrafs als ral·lis, aquest no pot superar mai el 75%.

b.3) La puntuació de les fotografies ha de ser entre 1 i 10 punts.

2. A partir del conjunt de dades que es proporcionen, realitzar les INSERCIONS de files a les taules que s'estimin més oportunes per tal de tenir dades emmagatzemades.

3. Realitzar les consultes següents sobre les taules de la BD creada a l'apartat 1 i amb les dades inserides a l'apartat 2:

a) Donar el passaport i el nom dels fotògrafs que s'han inscrit a tots i cadascun dels ral·lies que la UOCReserva organitza.

b) Donar el total de guanys de la UOCReserva obtinguts per la organització dels ral·lis, independentment de si els fotògrafs inscrits participen o no als safaris dels ral·lis.
Cada registre de la consulta ha de donar la següent informació: el codi del ralli, el preu del ralli, el número total de fotògrafs inscrits al ralli i el total de guanys per inscripcions al ral·li.

c) Donar el codi, la data d'inici i la data final dels ral·lis els quals, els safaris que el componen, han tingut assignat un guia amb una experiència menor a 5 anys però que mai han tingut assignat al guia 'John Walker' el qual té una experiència d'un any.

4. Crear les vistes que es detallen a continuació. Per a cada vista cal argumentar si és o no actualitzable, és a dir, si admet o no operacions d'inserció, d'esborrat i de modificació.

a) Crear un vista amb la classificació de la puntuació rebuda pels fotògrafs per les fotos presentades als safaris dels ral·lis.

En concret, es vol guardar el codi del ral·li, el nom del fotògraf i el total de punts acumulats pel fotògraf dins del ral·li.
A continuació feu una consulta sobre la vista que mostri la informació ordenada per ral·li i puntuació, de forma descendent.
En aquesta vista no es tindrà en compte que els fotògrafs hagin participat com a mínim a la meitat dels safaris del ral·li.

b) Pels safaris més perillosos es necessita crear una vista amb els guies que són especialistes en orientació o rastreig, i que a més, tenen més de 5 anys d'experiència.
Concretament es vol guardar el codi del guia, nom, domicili, telèfon, país, anys d'experiència i l'especialitat.

5. Procés de modificació: aplicar una rebaixa d'un 10% al preu dels ral·lies que tenen més de 4 fotògrafs inscrits.

6. Procés d'esborrat: Esborrar tots els guies que no són responsables de cap safari i que la seva especialitat és 'Rastreig'.

Documentació a lliurar:

1) Sentències SQL necessàries per definir i crear la BD. És necessari també incorporar les sentències d'inserció.
2) Sentències SQL que resolen les peticions anteriors, és a dir, cos de les sentències SQL de consulta (apartat 3), de modificació (apartat 5) i d'esborrat (apartat 6). En el cas de les consultes també és necessari adjuntar el seu resultat. En el cas dels processos d'esborrat i de modificació, és necessari adjuntar el contingut de la taula/taules implicades després d'haver executat les sentències d'esborrat i de modificació.
3) Sentències de creació de les vistes demanades a l'apartat 4 i resposta a les preguntes formulades.

SOLUCIÓ

1. Creació de la base de dades

A partir de la informació del enunciat, és important destacar:

a) S'ha optat per crear les taules Països, Especialitats i Empreses atès que, encara que només incorporen un atribut, poden aparèixer nous valors, ja que aquests no són estables.

b) És per això no s'ha implementat els valors d'aquests taules com a Checks de taula que controlin el rang de valors permesos.

c) La especialització Empleats – Guies de la primera part de la pràctica , ha estat resumida en una única taula Guies. Es podria haver tingut una taula d'empleats i una altra pels guies.

d) Fixeu-vos també que les taules incorporen les claus primàries i claus foranes que es poden deduir a partir de la solució de la primera part de la pràctica.

e) Cal recordar que les claus primàries, per definició, porten implícit un NOT NULL. Això també passa en les claus compostes. Així si <x,y> és una clau primària, vol dir que ni x ni y poden ser nuls.

f) Tots aquells atributs numèrics susceptibles d'aparèixer en un càlcul, els hem definit amb NOT NULL. El motiu és evitar problemes (per exemple, què passarà si comparem dos valors enters i un d'ells és nul?). El mateix raonament podem fer amb tots els atributs que continguin algun CHECK.

Creació BD

```
create database UOCReserva
with owner = postgres
encoding = 'SQL_ASCII'
tablespace = pg_default;
grant all on database UOCReserva to postgres;
```

Creació taules

```
create table Empreses(nom_empresa char(20),
                  primary key(nom empresa));

create table Paisos(nom_pais char(20),
                  primary key(nom_pais));

create table Especialitats(nom especialitat char(20),
                  primary key(nom_especialitat));

create table Guies(codi_guia char(5), nom char(50) not null, domicili
char(50) not null, telefon char(15) not null,
                  anys antiguetat integer not null, nom pais char(20)
not null, nom_especialitat char(20) not null,
                  primary key(codi_guia),
                  foreign key(nom_pais) references Paisos(nom_pais),
                  foreign key(nom_especialitat) references
Especialitats(nom_especialitat));
```

```
create table Fotografs(passaport char(10), nom char(50) not null,
domicili char(50) not null,
                      telefon char(15) not null, nom_pais char(20) not
null, nom_empresa char(20),
                  primary key(passaport),
                  foreign key(nom_pais) references Paisos(nom_pais),
                  foreign key(nom empresa) references
Empreses(nom_empresa));

create table Rallis(codi ralli char(10), preu integer not null,
data_inici date not null, data_final date not null,
                  primary key(codi_ralli));

create table Safaris(codi_safari char(10), data_inici date not null,
data final date not null,
                  codi_guia char(5) not null,
                  primary key(codi_safari),
                  foreign key(codi_guia) references Guies(codi_guia));

create table Fotos(codi foto char(5), nom char(25) not null, puntuacio
integer not null,
                      primary key(codi_foto),
                      check(puntuacio between 1 and 10));

create table Inscripcions(passaport char(10), codi_ralli char(10),
descompte integer default 0 not null,
                        primary key(passaport, codi_ralli),
                        foreign key(passaport) references
Fotografs(passaport),
                        foreign key(codi_ralli) references
Rallis(codi_ralli),
                        check(descompte >= 0 and descompte <= 75));

create table Agrupacions(codi_ralli char(10), codi_safari char(10),
                        primary key(codi_ralli, codi_safari),
                        foreign key(codi ralli) references
Rallis(codi_ralli),
                        foreign key(codi_safari) references
Safaris(codi_safari));

create table Participacions(passaport char(10), codi ralli char(10),
codi_safari char(10), codi_foto char(5),
                        primary key(passaport, codi_ralli, codi_safari),
                        foreign key(passaport) references
Fotografs(passaport),
                        foreign key(codi ralli, codi safari) references
Agrupacions(codi_ralli, codi_safari));
```

Restriccions d'integritat (checks i assercions)

AVÍS. En primer lloc, cal recordar que les assercions és un mecanisme de definició de propietats que han de tenir les dades, però que els SGBD no suporten. És a dir, no podeu provar-les, tal qual, al PostreSQL.

Per fer una asserció el que fem és fer una consulta dels casos erronis, i tot seguit dir que aquesta consulta ha de ser buida (NOT EXISTS).

Això ens dóna una pauta per provar les assercions: executem la consulta i comprovem que no genera resultats; inserim una dada errònia, tornem a executar la consulta, i comprovem que ara sí que dóna resultats (la dada errònia introduïda).

Les restriccions d'integritat que ens demanen són:

- Cal garantir que les participacions de fotògrafs a safaris de ral·lis siguin de ral·lis a on el fotògraf s'hagi inscrit.

```
create assertion RIParticipacionsFotografs
check ((not exists (select p.passaport
                    from Participacions p
                    where p.codi_ralli not in (
                          select i.codi_ralli
                          from Inscripcions i
                          where i.passaport = p.passaport))));
```

Aquesta restricció d'integritat no va poder ser representada a l'esquema conceptual de la primera part de la pràctica.

- En el cas d'aplicar un descompte a les inscripcions dels fotògrafs als ral·lis, aquest no pot superar mai el 75%.

És pot implementar via un *check* a l'hora de definir/crear la taula **Inscripcions**. Consultar la sentència de creació de la taula.

- La puntuació de les fotografies ha de ser entre 1 i 10 punts.

És pot implementar via un *check* a l'hora de definir/crear la taula **Fotos**. Consultar la sentència de creació de la taula.

Inserció de dades

```
begin work;
set transaction read write;
insert into Paisos values ('Anglaterra');
insert into Paisos values ('Alemanya');
insert into Paisos values ('Argentina');
insert into Paisos values ('Canada');
insert into Paisos values ('Congo');
insert into Paisos values ('Espanya');
insert into Paisos values ('Estats Units');
insert into Paisos values ('França');
insert into Paisos values ('Japo');
insert into Paisos values ('Namibia');
insert into Paisos values ('Kenia');
insert into Paisos values ('Portugal');
```

```
insert into Paisos values ('Russia');
insert into Paisos values ('Sudafrica');
commit;
select * from Paisos;
```

```
begin work;
set transaction read write;
insert into Especialitats values ('Vigilant');
insert into Especialitats values ('Conductor');
insert into Especialitats values ('Biologia');
insert into Especialitats values ('Espeleologia');
insert into Especialitats values ('Rastreig');
insert into Especialitats values ('Domador');
insert into Especialitats values ('Orientacio');
insert into Especialitats values ('Zoologia');
commit;
select * from Especialitats;
```

```
begin work;
set transaction read write;
insert into Empreses values ('Soleil Photo');
insert into Empreses values ('Fotolux AG');
insert into Empreses values ('Photo Industry');
insert into Empreses values ('BBC');
insert into Empreses values ('Wild Life');
insert into Empreses values ('Natura Viva');
commit;
select * from Empreses;
```

```
begin work;
set transaction read write;
insert into Guies values ('G01','Francis Cocoon','Kenya
University','111111',2,'Kenia','Zoologia');
insert into Guies values ('G02','Nwala Nimira','Namibia
University','222222',2,'Namibia','Rastreig');
insert into Guies values ('G03','John Walker','66, Queen
Elisabeth','333333',1,'Anglaterra','Espeleologia');
insert into Guies values ('G04','Enguele Lelu','43, Descartes
St.','444444',6,'Kenia','Orientacio');
insert into Guies values ('G05','Rene de la Court','Rue Savanna,
2','555555',10,'Congo','Orientacio');
insert into Guies values ('G06','Pauline Prie','Rue. Victor Hugo,
12','666666',7,'França','Zoologia');
insert into Guies values ('G07','Sanga Bole','54, Highway Ave.
','777777',6,'Kenia','Vigilant');
insert into Guies values ('G08','Andrew Philips','45, Johnsson
St','888888',3,'Sudafrica','Conductor');
insert into Guies values ('G09','Maria Llopis','Diagonal, 423
','999999',8,'Espanya','Biologia');
insert into Guies values ('G10','Tommey Eto','33, Livingstone
Ave.','101010',4,'Kenia','Rastreig');
commit;
select * from Guies;
```

```
begin work;
set transaction read write;
insert into Safaris values ('S01LL01','01-05-2004','01-10-2004' ,'G01');
insert into Safaris values ('S01LL02','01-07-2004','01-15-2004' ,'G06');
insert into Safaris values ('S01LL03','01-22-2004','01-29-2004', 'G04');
insert into Safaris values ('S06LL01','06-03-2004','06-11-2004', 'G09');
insert into Safaris values ('S06LL03','06-07-2004','06-14-2004', 'G04');
insert into Safaris values ('S06LL05','06-19-2004','06-25-2004', 'G05');
insert into Safaris values ('S06LL07','06-22-2004','06-29-2004', 'G01');
insert into Safaris values ('S10LL02','10-09-2004','10-14-2004', 'G05');
insert into Safaris values ('S10LL03','10-16-2004','10-21-2004', 'G01');
insert into Safaris values ('S10LL04','10-17-2004','10-23-2004', 'G09');
insert into Safaris values ('S10LL05','10-04-2004','10-11-2004', 'G08');
insert into Safaris values ('S11LL01','11-02-2004','11-09-2004', 'G06');
insert into Safaris values ('S11LL03','11-07-2004','11-13-2004', 'G07');
insert into Safaris values ('S11LL06','11-15-2004','11-23-2004', 'G04');
insert into Safaris values ('S11LL07','11-19-2004','11-28-2004', 'G03');
commit;
select * from Safaris;
```

```
begin work;
set transaction read write;
insert into Rallis values ('R2004R01',750,'01-05-2004','01-15-2004');
insert into Rallis values ('R2004C01',1100,'01-05-2004','01-29-2004');
insert into Rallis values ('R2004R06',925,'06-01-2004','06-14-2004');
insert into Rallis values ('R2004C06',1400,'06-03-2004','06-30-2004');
insert into Rallis values ('R2004R10',820,'10-07-2004','10-22-2004');
insert into Rallis values ('R2004C10',1200,'10-04-2004','10-25-2004');
commit;
select * from Rallis;
```

```
begin work;
set transaction read write;
insert into Fotos values ('F01','Savanna Sunset',3);
insert into Fotos values ('F02','Vie, Vie, Vie !!!',5);
insert into Fotos values ('F03','The Lion King',7);
insert into Fotos values ('F04','Happy Family',2);
insert into Fotos values ('F05','Green Power',6);
insert into Fotos values ('F06','I want to stay here',4);
insert into Fotos values ('F07','El Meu Planeta',9);
insert into Fotos values ('F08','Agua de Vida',5);
insert into Fotos values ('F09','Alive Paradise',7);
insert into Fotos values ('F10','Explosion de Color',9);
insert into Fotos values ('F11','Samba, Samba !!',7);
insert into Fotos values ('F12','Eo',6);
insert into Fotos values ('F13','Terra Madre',2);
insert into Fotos values ('F14','Africa Rules',4);
insert into Fotos values ('F15','Wild Wind',5);
commit;
select * from Fotos;
```

```
begin work;
set transaction read write;
```

```
insert into Fotografs values ('12894578','Joao Sousa','Rua Algarve,
12','111111','Portugal',NULL);
insert into Fotografs values ('59258743','Kenny Rogers','64, Smith
St.','222222','Estats Units','Wild Life');
insert into Fotografs values ('09884323','Alain Poulet','Rue Guerlain,
34','333333','Canada',NULL);
insert into Fotografs values ('23663233','Henrich Schoffer','Bonn Strasse
23','444444','Alemanya','Fotolux AG');
insert into Fotografs values ('56898743','Jaume Busquets','Consell de
Cent, 2','555555','Espanya','Natura Viva');
insert into Fotografs values ('23434388','Brigitte Colette','Rue Latin
55','666666','França','Soleil Photo');
insert into Fotografs values ('09992344','Nimira Tao','Nikita
5','777777','Japo',NULL);
insert into Fotografs values ('32456654','Andy Lucas','32 Waterloo
Str.','888888','Anglaterra','BBC');
insert into Fotografs values ('09999823','Diego Pesini','Mayo,
23','999999','Argentina','Natura Viva');
commit;
select * from Fotografs;
```

```
begin work;
set transaction read write;
insert into Agrupacions values ('R2004R01','S01LL01');
insert into Agrupacions values ('R2004R01','S01LL02');
insert into Agrupacions values ('R2004C01','S01LL01');
insert into Agrupacions values ('R2004C01','S01LL02');
insert into Agrupacions values ('R2004C01','S01LL03');
insert into Agrupacions values ('R2004R06','S06LL01');
insert into Agrupacions values ('R2004R06','S06LL03');
insert into Agrupacions values ('R2004C06','S06LL01');
insert into Agrupacions values ('R2004C06','S06LL03');
insert into Agrupacions values ('R2004C06','S06LL05');
insert into Agrupacions values ('R2004C06','S06LL07');
insert into Agrupacions values ('R2004R10','S10LL02');
insert into Agrupacions values ('R2004R10','S10LL03');
insert into Agrupacions values ('R2004R10','S10LL05');
insert into Agrupacions values ('R2004C10','S10LL02');
insert into Agrupacions values ('R2004C10','S10LL03');
insert into Agrupacions values ('R2004C10','S10LL04');
insert into Agrupacions values ('R2004C10','S10LL05');
commit;
select * from Agrupacions;
```

```
begin work;
set transaction read write;
insert into Inscripcions values ('12894578','R2004R01',0);
insert into Inscripcions values ('09884323','R2004R01',25);
insert into Inscripcions values ('23663233','R2004R01',0);
insert into Inscripcions values ('23434388','R2004R01',0);
insert into Inscripcions values ('23434388','R2004C01',10);
insert into Inscripcions values ('59258743','R2004C01',0);
insert into Inscripcions values ('09884323','R2004C01',5);
insert into Inscripcions values ('12894578','R2004C01',0);
insert into Inscripcions values ('09884323','R2004R06',5);
```

```
insert into Inscripcions values ('56898743','R2004R06',0);
insert into Inscripcions values ('59258743','R2004R06',0);
insert into Inscripcions values ('32456654','R2004R06',0);
insert into Inscripcions values ('12894578','R2004R06',0);
insert into Inscripcions values ('56898743','R2004C06',20);
insert into Inscripcions values ('23663233','R2004C06',0);
insert into Inscripcions values ('12894578','R2004C06',10);
insert into Inscripcions values ('23434388','R2004R10',0);
insert into Inscripcions values ('32456654','R2004R10',8);
insert into Inscripcions values ('09999823','R2004R10',0);
insert into Inscripcions values ('12894578','R2004R10',0);
insert into Inscripcions values ('59258743','R2004C10',7);
insert into Inscripcions values ('09884323','R2004C10',0);
insert into Inscripcions values ('32456654','R2004C10',10);
insert into Inscripcions values ('23663233','R2004C10',0);
insert into Inscripcions values ('09999823','R2004C10',15);
insert into Inscripcions values ('09992344','R2004C10',0);
insert into Inscripcions values ('12894578','R2004C10',0);
commit;
select * from Inscripcions;
```

```
begin work;
set transaction read write;
insert into Participacions values ('12894578','R2004R01','S01LL01',NULL);
insert into Participacions values ('12894578','R2004R01','S01LL02',NULL);
insert into Participacions values ('09884323','R2004R01','S01LL01',NULL);
insert into Participacions values ('09884323','R2004R01','S01LL02',NULL);
insert into Participacions values ('23663233','R2004R01','S01LL01',NULL);
insert into Participacions values ('23434388','R2004R01','S01LL01',NULL);
insert into Participacions values ('23434388','R2004R01','S01LL02',NULL);
insert into Participacions values
('23434388','R2004C01','S01LL01','F02');
insert into Participacions values ('23434388','R2004C01','S01LL02',NULL);
insert into Participacions values ('23434388','R2004C01','S01LL03',NULL);
insert into Participacions values
('59258743','R2004C01','S01LL01','F04');
insert into Participacions values ('59258743','R2004C01','S01LL02',NULL);
insert into Participacions values
('09884323','R2004C01','S01LL01','F10');
insert into Participacions values ('09884323','R2004C01','S01LL02',NULL);
insert into Participacions values ('09884323','R2004C01','S01LL03',NULL);
insert into Participacions values
('09884323','R2004R06','S06LL01','F01');
insert into Participacions values ('09884323','R2004R06','S06LL03',NULL);
insert into Participacions values
('56898743','R2004R06','S06LL01','F07');
insert into Participacions values ('56898743','R2004R06','S06LL03',NULL);
insert into Participacions values
('59258743','R2004R06','S06LL01','F03');
insert into Participacions values ('59258743','R2004R06','S06LL03',NULL);
insert into Participacions values
('32456654','R2004R06','S06LL01','F06');
insert into Participacions values ('32456654','R2004R06','S06LL03',NULL);
insert into Participacions values ('56898743','R2004C06','S06LL01',NULL);
insert into Participacions values ('56898743','R2004C06','S06LL03',NULL);
insert into Participacions values ('56898743','R2004C06','S06LL05',NULL);
```

```
insert into Participacions values ('56898743','R2004C06','S06LL07',NULL);
insert into Participacions values ('23663233','R2004C06','S06LL01',NULL);
insert into Participacions values ('23663233','R2004C06','S06LL03',NULL);
insert into Participacions values ('23663233','R2004C06','S06LL05',NULL);
insert into Participacions values ('23663233','R2004C06','S06LL07',NULL);
insert into Participacions values ('12894578','R2004C06','S06LL05',NULL);
insert into Participacions values ('12894578','R2004C06','S06LL07',NULL);
insert into Participacions values ('23434388','R2004R10','S10LL02',NULL);
insert into Participacions values ('23434388','R2004R10','S10LL03',NULL);
insert into Participacions values ('32456654','R2004R10','S10LL05',NULL);
insert into Participacions values ('32456654','R2004R10','S10LL02',NULL);
insert into Participacions values ('32456654','R2004R10','S10LL03',NULL);
insert into Participacions values ('09999823','R2004R10','S10LL02',NULL);
insert into Participacions values ('09999823','R2004R10','S10LL03',NULL);
insert into Participacions values ('09999823','R2004R10','S10LL05',NULL);
insert into Participacions values ('59258743','R2004C10','S10LL02',NULL);
insert into Participacions values ('59258743','R2004C10','S10LL03',NULL);
insert into Participacions values ('59258743','R2004C10','S10LL04',NULL);
insert into Participacions values
('59258743','R2004C10','S10LL05','F05');
insert into Participacions values
('09884323','R2004C10','S10LL02','F12');
insert into Participacions values ('09884323','R2004C10','S10LL03',NULL);
insert into Participacions values ('09884323','R2004C10','S10LL04',NULL);
insert into Participacions values
('09884323','R2004C10','S10LL05','F08');
insert into Participacions values
('32456654','R2004C10','S10LL02','F13');
insert into Participacions values ('32456654','R2004C10','S10LL03',NULL);
insert into Participacions values ('32456654','R2004C10','S10LL04',NULL);
insert into Participacions values
('32456654','R2004C10','S10LL05','F09');
insert into Participacions values
('23663233','R2004C10','S10LL02','F14');
insert into Participacions values ('23663233','R2004C10','S10LL03',NULL);
insert into Participacions values ('23663233','R2004C10','S10LL04',NULL);
insert into Participacions values
('09999823','R2004C10','S10LL05','F11');
insert into Participacions values ('09999823','R2004C10','S10LL02',NULL);
insert into Participacions values ('09999823','R2004C10','S10LL03',NULL);
insert into Participacions values ('09992344','R2004C10','S10LL04',NULL);
insert into Participacions values
('09992344','R2004C10','S10LL05','F15');
commit;
select * from Participacions;
```

2. Consultes

Les consultes demanades són les següents:

a) Donar el passaport i el nom dels fotògrafs que s'han inscrit a tots i cadascun dels ral·lies que la UOCReserva organitza.

2.a

```
select a.passaport, a.nom
from fotografs a
where not exists
       (select *
         from rallis p
         where not exists(select *
                          from inscripcions da
                          where da.codi_ralli=p.codi_ralli and
                                da.passaport = a.passaport));
```

Resultat:

Passaport	Nom
12894578	Joao Sousa

b) Donar el total de guanys de la UOCReserva obtinguts per la organització dels ral·lis, independentment de si els fotògrafs inscrits participen o no als safaris dels ral·lis. Cada registre de la consulta ha de donar la següent informació: el codi del ralli, el preu del ralli, el número total de fotògrafs inscrits al ralli i el total de guanys per inscripcions al ral·li.

2.b

```
select i.codi_ralli, r.preu, count(i.passaport) as Inscrits,
sum(r.preu) - sum(r.preu*(i.descompte)/100) as Total_Guanys
      from inscripcions i, rallis r
            where i.codi_ralli = r.codi_ralli
                  group by i.codi_ralli, r.preu;
```

Resultat:

Codi_ralli	Preu	Inscrits	Total_Guanys
R2004C01	1100	4	4235
R2004C06	1400	3	3780
R2004C10	1200	7	8016
R2004R01	750	4	2812,5
R2004R06	925	5	4578,75
R2004R10	820	4	3214,4

c) Donar el codi, la data d'inici i la data final dels ral·lis els quals, els safaris que el componen han tingut assignat un guia amb una experiència menor a 5 anys però que mai han tingut assignat al guia 'John Walker' el qual té una experiència d'un any.

2.c

```
select distinct r.codi_ralli, r.data_inici, r.data_final
    from rallis r,  safaris s, agrupacions a, guies g
    where   r.codi_ralli = a.codi_ralli and a.codi_safari= s.codi_safari
and s.codi_guia=g.codi_guia and g.anys_antiguetat < 5
```

```
    and not exists
      (select * from guies g2, agrupacions a2, safaris s2
         where a2.codi_ralli = r.codi_ralli and
               a2.codi_safari= s2.codi_safari and
               s2.codi_guia=g2.codi_guia and
               g2.nom = 'John Walker');
```

Resultat:

codi_ralli	data_inici	data_final
R2004C01	01-05-2004	01-29-2004
R2004C06	06-03-2004	06-30-2004
R2004C10	10-04-2004	10-25-2004
R2004R01	01-05-2004	01-15-2004
R2004R10	10-07-2004	10-22-2004

Una altra forma de fer aquesta consulta és utilitzant la combinació JOIN. Com el que cal és fer combinacions de més de dos taules s'han d'anar fent combinacions per parelles de taules i la taula resultant es converteix en la primera parella de la combinació següent. La consulta quedaria així:

```
select distinct r.codi_ralli, r.data_inici, r.data_final
    from ((rallis r JOIN agrupacions a ON
      r.codi_ralli = a.codi_ralli)
JOIN safaris s ON a.codi_safari= s.codi_safari )
    JOIN guies g  ON s.codi guia=g.codi guia and g.anys antiguetat < 5
    and not exists
      (select * from guies g2, agrupacions a2, safaris s2
         where a2.codi_ralli = r.codi_ralli and
               a2.codi_safari= s2.codi_safari and
               s2.codi guia=g2.codi guia and
               g2.nom = 'John Walker');
```

3.Creació de vistes

a) Crear un vista amb la classificació de la puntuació rebuda pels fotògrafs per les fotos presentades als safaris de cada ral·li.
En concret, es vol guardar el codi del ral·li, el nom del fotògraf i el total de punts acumulats pel fotògraf dins del ral·li.
A continuació feu una consulta sobre la vista que mostri la informació ordenada per ral·li i puntuació, de forma descendent.
En aquesta vista no es tindrà en compte que els fotògrafs hagin participat com a mínim a la meitat dels safaris del ral·li.

3.a

```
create view VClassifFotos
(codi_ralli, nom_fotograf, total_punts) as
select p.codi_ralli, t.nom, sum(f.puntuacio) as punts
```

```
from Participacions p, Fotos f, Fotografs t
where p.codi_foto = f.codi_foto
   and t.passaport = p.passaport
         and p.codi_foto is not null
group by p.codi_ralli,t.nom;

select * from VClassifFotos order by codi_ralli, total_punts desc;
```

Aquesta vista no és actualitzable, perquè no hi ha manera de propagar els canvis des de la vista a les taules de base sobre les quals s'ha definit la vista. Les vistes com aquesta que incorporen operacions de combinació o funcions d'agregació mai no són actualitzables.

Resultat:

```
select * from VClassifFotos order by codi_ralli, total_punts desc;
```

codi_ralli	nom_fotograf	Total_punts
R2004C01	Alain Poulet	9
R2004C01	Brigitte Colette	5
R2004C01	Kenny Rogers	2
R2004C10	Alain Poulet	11
R2004C10	Andy Lucas	9
R2004C10	Diego Pesini	7
R2004C10	Kenny Rogers	6
R2004C10	Nimira Tao	5
R2004C10	Henrich Schoffer	4
R2004R06	Jaume Busquets	9
R2004R06	Kenny Rogers	7
R2004R06	Andy Lucas	4
R2004R06	Alain Poulet	3

b) Pels safaris més perillosos es necessita crear una vista amb els guies que són especialistes en orientació o rastreig, i que a més, tenen més de 5 anys d'experiència.
Concretament es vol guardar el codi del guia, nom, domicili, telèfon, país, anys d'experiència i l'especialitat.

3.b

```
create view VGuiesExperts
(codi_guia, nom, domicili, telefon, anys_antiguetat, nom_pais,
nom especialitat ) as
select g.codi_guia, g.nom, g.domicili, g.telefon, g.anys_antiguetat,
```

```
        g.nom_pais, g.nom_especialitat
from Guies g
where g.anys antiguetat > 5
and (g.nom_especialitat = 'Rastreig' or g.nom_especialitat =
'Orientacio');

select * from VGuiesExperts;
```

Codi Guia	Nom	Domicili	Telèfon	Antig.	País	Especialitat
G04	Enguele Lelu	43, Descartes St.	444444	6	Kenia	Orientacio
G05	Rene de la Court	Rue Savanna, 2	555555	10	Congo	Orientacio

Aquesta vista no és actualitzable atès que en PostgreSQL les vistes no són actualitzables. Perquè ho sigui caldria crear una *rule* que al mateix moment que es realitzi una acció a la taula base es realitzi sobre la vista.

4. Modificació

Aplicar una rebaixa d'un 10% al preu dels ral·lies que tenen més de 4 fotògrafs inscrits.

```
update Rallis
set preu=preu-0.10*preu
where codi_ralli in (select i.codi_ralli
                     from Inscripcions i
                     group by i.codi ralli
                     having count(i.passaport) > 4);
```

Resultat:

S'actualitzen 2 registres de la taula "Rallis", els ral·lis R2004R06 i R2004C10 .

El contingut dels registres modificats és:

Codi Ral·li	Preu	Data Inici	Data Final
R2004R01	750	01-05-2004	01-15-2004
R2004C01	1100	01-05-2004	01-29-2004
R2004R06	832	06-01-2004	06-14-2004
R2004C06	1400	06-03-2004	06-30-2004
R2004R10	820	10-07-2004	10-22-2004
R2004C10	1080	10-04-2004	10-25-2004

5. Esborrat

Esborrar tots els guies que no són responsables de cap safari i que la seva especialitat és 'Rastreig'.

```
delete from Guies
where codi_guia not in (select s.codi_guia
                        from Safaris s )
and nom_especialitat = 'Rastreig';
```

Resultat:

S'esborren de la taula de "Guies" 2 registres: **"G02"** i **"G10"**.

6. Neteja de tot plegat

Finalment presentem les instruccions necessàries per destruir la base de dades, i tornar-la a crear.

```
drop database "UOCReserva";
```

Implementació en PostgreSQL de la pràctica de BDII.

Enunciat de la Pràctica (Part I i Part II)

La UOCReserva

Presentació i objectius

Objectius

L'objectiu d'aquesta pràctica és comprovar el grau de comprensió de diferents conceptes de bases de dades que es tracten en aquesta assignatura, com són principalment els components lògics (dades i control), els components d'emmagatzematge, les transaccions i la programació amb SQL i Java.

Presentació

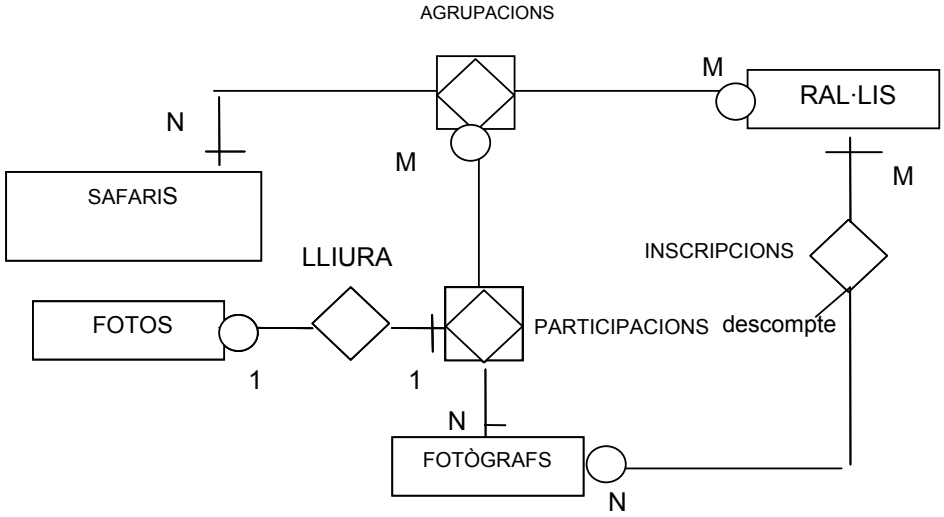
Aquesta pràctica consta de dues parts, que cal lliurar en terminis diferents i que s'indiquen més endavant. A la primera part de la pràctica es treballen les transaccions, l'accés a Base de Dades fent servir Java, els procediments emmagatzemats i els disparadors. A la segona part de la pràctica es fan servir totes les eines desenvolupades a la primera part per a resoldre un problema una mica més complex.

Enunciat de la pràctica

La UOCReserva

La UOCReserva ha decidit seguir confiant en nosaltres per tal de desenvolupar els seus sistemes d'informació, de la mateixa manera que ho va fer a **Bases de Dades I**. En l'anterior ocasió se'ns va demanar la construcció d'una complexa Base de Dades amb ral·lis, safaris, fotògrafs i altres, i el desenvolupament d'una sèrie de consultes sobre aquesta.

En aquesta ocasió, la Base de Dades s'ha simplificat considerablement.



En el directori `sql` de la pràctica podeu trobar el fitxer de creació d'aquestes taules (**`uocreserva_bd.sql`**) i un fitxer amb sentències d'inserció de dades de prova (**`uocreserva_bd_ins.sql`**).

Esquema lògic de partida

(claus primàries subratllades)
(en cursiva atributs que poden prendre valor nul)

Fotògrafs (passaport, nom, telèfon, nom_pais)

Dades personals dels fotògrafs com, per exemple, el seu pais de procedència.

Ral·lis (codi_ral·li, nom_ral·li, preu, total_recaptació)

Dades dels ral·lis. Entre d'altres, el preu d'inscripció i el total recaptat en concepte d'inscripcions de fotògrafs.

Safaris (codi_safari, nom_safari, nom_guia)

Dades dels safaris, com el seu nom i el nom del guia del safari.

Fotos (codi_foto, nom, puntuació)

Dades de les fotos com per exemple el nom que el fotògraf dóna a la foto i la puntuació atorgada pel jurat.

Inscripcions (passaport, codi_ral·li, descompte)
{passaport} clau forana cap a Fotògrafs(passaport)
{codi_ral·li} clau forana cap a Ral·lis(codi_ral·li)

Inscripcions de fotògrafs al ral·li. Hi ha fotògrafs que tenen descompte en la seva inscripció sobre el preu del ral·li (el preu d'un ral·li està a la taula ral·lis).

Agrupacions (codi_ral·li, codi_safari)
{codi_ral·li} clau forana cap a Ral·lis(codi_ral·li)
{codi_safari} clau forana cap a Safaris(codi_safari)

Safaris que composen cada ral·li.

Participacions (passaport, codi_ral·li, codi_safari, *codi_foto*)
{passaport} clau forana cap a Fotògrafs(passaport)
{codi_ral·li,codi_safari} clau forana cap a Agrupacions(codi_ral·li,codi_safari)
{codi_foto} clau forana cap a Fotos(codi_foto)

Participacions de fotògrafs a safaris dels ral·lis. Opcionalment, el fotògraf pot lliurar una foto efectuada durant la celebració del safari per tal d'optar a concursar (i potser guanyar) el premi de cada ral·li.

PRIMERA PART

A la primera part de la pràctica volem que treballeu la definició de taules, el control de les transaccions, l'accés a les dades des de Java i els disparadors i procediments emmagatzemats.

Per a la part de Java hem desenvolupat un seguit de classes que us permetran concentrar-vos en les parts que són estrictament de Base de Dades, descarregant-vos de tot el que és entrada de dades i formateig.

De fet us donem totes les classes que heu de desenvolupar, però algunes d'elles no estan completes. En aquests casos us ho indiquem en el propi codi. Les parts pendents d'implementar ("per fer", en anglès "to do") tenen un comentari com el següent:

```
// TODO Obtenir una connexió de la Base de Dades
```

Si feu servir una eina com Eclipse o JEdit podreu obtenir un llistat complet de les tasques pendents. Al final de l'enunciat podeu trobar aquesta llista.

Totes les classes estan i hauran d'estar en un *package* Java anomenat `uocreserva`. Les classes genèriques que se us faciliten són:

Classe	Descripció	Estat
`uocreserva.Utilities`	Mètodes diversos per a l'entrada de dades i la seva transformació.	No cal que modifiqueu res
`uocreserva.BDAccessor`	Mètodes per a facilitar la gestió de la connexió a la Base de Dades. Fa servir un fitxer `bd.properties` on es defineixen els paràmetres de la connexió a la BD. Potser haureu de modificar aquest arxiu per ajustar-ho a les característiques del vostre equip.	No estan implementats tots els mètodes.
`uocreserva.PrintList`	Mètodes per mostrar llistats per pantalla.	No cal que modifiqueu res

Per a tots els exercicis també se us proporciona un conjunt de classes que facilitaran l'accés a les taules. Així, per exemple, tenim una classe `uocreserva.Rally` on tenim modelat un atribut (variable) per a cada columna de la taula Rallis, i un conjunt de mètodes `set` i `get`, que permeten accedir a les mateixes. Així tenim a la classe Rally els atributs `code`, `name` i `price`, i els mètodes `getCode`, `setCode`, `getName`, `setName`, `getPrice` i `setPrice`. Aquestes classes d'ajuda no caldrà que les modifiqueu.

Podeu consultar el *javadoc* al directori `doc` de la pràctica per una descripció de tots els mètodes de les classes. Obriu el fitxer `index.html`.

Com a resultat de la pràctica haureu de lliurar el codi font de totes les classes Java (**`*.java`**), els procediments i disparadors que creeu (**`*.sql`**) i un fitxer Word (**`practica1.doc`**) on heu d'afegir totes les explicacions que considereu necessàries.

Si heu de crear mètodes nous en qualsevol de les classes els haureu d'indicar en el fitxer Word.

Exercici 1. Creació de disparadors

En aquest exercici definirem un seguit de disparadors per tal de que la informació de la BD sigui coherent segons les següents regles de negoci:

a) Un ral·li no pot tenir més de **5 safaris**.
b) Una inscripció no podrà ser eliminada un cop el fotògraf ja hagi participat en algun safari del ral·li inscrit.
c) També ens hem d'assegurar que un fotògraf no pot participar en un safari d'un ral·li si no hi està inscrit. Comproveu primer que el ral·li i el fotògraf existeixen i doneu l'excepció corresponent si calgués.
d) Finalment, ens cal un disparador que ens mantingui l'atribut *total_recaptació* d'un ral·li actualitzat en funció de les inscripcions fetes a aquest ral·li. Segons les converses amb els responsables d'UOCReserva, el preu d'un ral·li i el descompte d'una inscripció no es modificaran mai després d'haver estat creats, per tant no cal que el disparador tingui en compte les modificacions sobre aquests atributs. També cal tenir en compte que en cap cas no es pot fer una actualització sobre l'atribut *total_recaptació* directament, ja que sempre es calcularà amb el nostre disparador, però tampoc cal desenvolupar codi addicional per a garantir que no es faran modificacions directes de l'atribut *total_recaptació*.

   Nota: Tant en aquest últim disparador, com en els anteriors, penseu més enllà del cas evident.

   Es demana: Un fitxer **`triggers.sql`** que contingui la definició de tots els *triggers* i procediments definits. Si voleu fer una explicació de la vostra solució, feu-la al fitxer Word.

Si heu de fer alguna observació respecte al disseny o a la implementació, podeu fer-ho en el fitxer Word.

Exercici 2. Procediment que retorna informació de ral·lis

La UOCReserva sovint comprova les inscripcions que hi ha al diversos ral·lis que ofereix. Hem de realitzar un procediment emmagatzemat que retorni un llistat de les dades dels ral·lis que tenen 5 o més inscripcions. Tot i això, hauria de ser possible executar el procediment amb un nombre d'inscripcions diferent.

El resultat s'ha d'obtenir ordenat pel nom del ral·li.

Les dades a obtenir són:

Nom del ral·li, Preu del ral·li, Total participants inscrits, Nombre total de safaris del ral·li

La signatura d'aquest procediment serà la següent:

```
CREATE PROCEDURE listRallies (minInscriptions INT DEFAULT 5)
  RETURNING CHAR(50), INTEGER, INTEGER, INTEGER;
```

> **Es demana**: Un fitxer **`procedures.sql`** on es defineixi aquest procediment. Els comentaris els podeu incloure en el fitxer `Word`.

Si heu de fer alguna observació respecte al disseny o a la implementació, podeu fer-ho en el fitxer Word.

Exercici 3. Inserció de ral·lis a la BD

També necessitarem un programa Java que permeti donar d'alta un nou ral·li. L'alta del nou ral·li implica, tenint en compte que els safaris ja s'han d'haver creat a la BD amb anterioritat a l'execució d'aquest programa, omplir les dades de la taula `agrupacions` adequadament.
Davant de qualsevol tipus d'error cal descartar tot canvi que s'hagi produït a la BD durant l'execució del programa d'inserció de ral·lis.
Aquest programa no demanarà les dades per teclat sinó que les llegirà d'un fitxer de dades. La raó per no demanar les dades per teclat és que això suposa mantenir connexions i recursos de la BD ocupats durant molt de temps, així que podem suposar que la creació d'aquest fitxer s'ha realitzat mitjançant una aplicació auxiliar que demana les dades a l'usuari i les posa en el fitxer en el format adequat per fer la càrrega posterior amb el programa que vosaltres creareu. Podem assumir que les dades del fitxer són correctes.

El format del fitxer és el següent:

```
<codi_ral·li>;<nom_ral·li>;<preu_ral·li>;<codi_safari_1>;...;<codi_safari_N>
```

on en una única línia i separats per ";" indiquen, i en aquest ordre, el codi del ral·li a donar d'alta, el nom d'aquest ral·li, el preu del mateix, i a continuació els codis de tots els safaris que agrupa aquest ral·li.

El fitxer que llegirà el programa Java, per donar d'alta el nou ral·li, es diu "exercici3.txt" i el podeu trobar al directori arrel. A continuació us mostrem les dades que conté el fitxer a tall d'exemple:

```
R2005-T1-3;Felins perillosos;2500;S2005-F1-2;S2005-M1-2
```

Aquest programa Java haurà d'estar en una classe anomenada `RallyCreation`, de la qual també us facilitem l'esquelet.

Classe	Descripció	Estat
`uocreserva.RallyCreation`	Mètodes per a la inserció d'un ral·li i els seus safaris.	Cal implementar les insercions i la connexió a la BD.

`uocreserva.Rally`	Per facilitar l'accés a les dades de la taula `Rallis`.	No cal que modifiqueu res.
`uocreserva.RallyWithSafaris`	Per facilitar l'accés a les dades de la taula `Agrupacions`.	No cal que modifiqueu res.

Es demana: A més del codi Java, haureu d'explicar en el fitxer Word com feu el control de les transaccions.

Si heu de fer alguna observació respecte al disseny o a la implementació, podeu fer-ho en el fitxer Word.

Exercici 4. Inserció de participacions a la BD

Per tal de donar d'alta la participació d'un determinat fotògraf a un safari concret d'un determinat ral·li realitzarem un altre programa en Java.

En el cas que el fotògraf lliuri una foto, caldrà introduir les dades corresponents a la foto lliurada a la taula corresponent. Cal comprovar, entre altres coses, els possibles errors següents:

- Si el fotògraf o el ral·li no existeixen, caldrà mostrar els missatges, "Participació incorrecta. No existeix el fotògraf" o "Participació incorrecta. No existeix el ral·li".
- Si el fotògraf no s'ha inscrit al ral·li caldrà mostrar el missatge "Participació incorrecta. Fotògraf no inscrit". Això ho podreu detectar amb l'excepció de la regla de negoci que haurem creat en l'exercici 1.
- Si el safari no pertany al ral·li caldrà mostrar el missatge "Participació incorrecta. El safari no pertany al ral·li".
- Si existeix la participació o la foto que volem crear, caldrà mostrar els missatges, "Participació incorrecta. Ja existeix la participació" o "Participació incorrecta. Ja existeix la foto".

Davant de qualsevol altra situació d'error caldrà mostrar el missatge "Error intern en l'execució del programa".

Com en l'exercici anterior, la lectura de les dades es realitzarà a partir d'un fitxer. El raonament és el mateix que s'ha exposat abans i l'estructura del fitxer és la següent:

```
<passaport>;<codi_ral·li>;<codi_safari>;<codi_foto>;<nom_foto>;<puntuació
>
```

on en una única línia i separats per ";" indiquen, i en aquest ordre, el passaport del fotògraf, el codi del ral·li, i del safari al que ha participat, i opcionalment si ha entregat una foto també hi haurà el codi de la foto, el nom d'aquesta i la puntuació que ha rebut la foto.

El fitxer que llegirà el programa Java, per donar d'alta una participació, es diu "exercici4.txt" i el podeu trobar al directori arrel, igual que el fitxer de l'exercici anterior. A continuació us mostrem les dades que conté el fitxer a tall d'exemple:

```
7911234578;R2005-T1-2;S2005-F1-2;F0004;Lleona dormilega;5
```

Aquest programa Java haurà d'estar en una classe anomenada `ParticipationCreation`, de la qual també us facilitem l'esquelet.

Classe	Descripció	Estat
`uocreserva.ParticipationCreation`	Mètodes per a la inserció d'una participació d'un fotògraf en un safari d'un ral·li.	Cal implementar les insercions, la connexió a la BD i la gestió d'errors
`uocreserva.Participation`	Per facilitar l'accés a les dades de la taula `Participacions`.	No cal que modifiqueu res.
`uocreserva.Photo`	Per facilitar l'accés a les dades de la taula `Fotos`.	No cal que modifiqueu res.

Es demana: A més del codi Java, haureu d'explicar en el fitxer Word com feu el control de les transaccions i la gestió dels possibles errors.

Si heu de fer alguna observació respecte al disseny o a la implementació, podeu fer-ho en el fitxer Word.

Solució de la Pràctica. Primera Part

UOCReserva

Juntament amb aquesta document us adjuntem el codi de la solució proposada. Aquest codi està estructurat en directoris de la mateixa manera que ho estava l'enunciat i és una extensió del mateix.

Primera part.

Creació de les taules:

```
CREATE TABLE Fotografs(
    passaport CHAR(10),
    nom          CHAR(50) NOT NULL,
    telefon      CHAR(15) NOT NULL,
    nom_pais  CHAR(20) NOT NULL,
    PRIMARY KEY (passaport),
    CONSTRAINT pk_fotografs UNIQUE (passaport)
);

CREATE TABLE Rallis(
    codi_ralli       CHAR(10),
    nom ralli       CHAR(50) NOT NULL,
    preu              INTEGER NOT NULL,
    total_recaptacio INTEGER DEFAULT 0 NOT NULL,
    PRIMARY KEY (codi_ralli),
    CONSTRAINT pk_rallis UNIQUE (codi_ralli)
);

CREATE TABLE Safaris(
    codi_safari CHAR(10),
    nom_safari  CHAR(50) NOT NULL,
    nom_guia    CHAR(50) NOT NULL,
    PRIMARY KEY (codi_safari),
    CONSTRAINT pk_safari UNIQUE (codi_safari)
);

CREATE TABLE Fotos(
    codi_foto CHAR(5),
    nom_foto  CHAR(50) NOT NULL,
    puntuacio INTEGER NOT NULL,
    PRIMARY KEY (codi_foto),
    CONSTRAINT pk_fotos UNIQUE (codi_foto),
    CONSTRAINT ck_puntuacio CHECK (puntuacio BETWEEN 1 AND 10)
);
```

```
CREATE TABLE Inscripcions(
    passaport  CHAR(10),
    codi_ralli CHAR(10),
    descompte  INTEGER NOT NULL,
    CONSTRAINT pk_inscripcions PRIMARY KEY (passaport, codi_ralli),
    CONSTRAINT fk_fotografs FOREIGN KEY (passaport)REFERENCES
Fotografs(passaport),
    CONSTRAINT fk_rallis FOREIGN KEY (codi_ralli)  REFERENCES
Rallis(codi_ralli),
    CHECK (descompte BETWEEN 0 AND 75)
);


CREATE TABLE Agrupacions(
    codi ralli  CHAR(10),
    codi_safari CHAR(10),
    CONSTRAINT pk_agrupacions PRIMARY KEY (codi_ralli, codi_safari),
    CONSTRAINT fk_agru_rallis FOREIGN KEY (codi_ralli) REFERENCES
Rallis(codi_ralli),
    CONSTRAINT fk safaris FOREIGN KEY (codi safari) REFERENCES
Safaris(codi_safari)
);

CREATE TABLE Participacions(
    passaport  CHAR(10),
    codi_ralli  CHAR(10),
    codi_safari CHAR(10),
    codi_foto CHAR(5),
    CONSTRAINT pk_participacions PRIMARY KEY (passaport, codi_ralli,
codi_safari),
    CONSTRAINT fk_part_fotografs FOREIGN KEY (passaport)  REFERENCES
Fotografs(passaport),
    CONSTRAINT fk_agrupacions FOREIGN KEY (codi_ralli, codi_safari)
REFERENCES Agrupacions(codi_ralli, codi_safari),
    CONSTRAINT fk_part_fotos FOREIGN KEY (codi_foto) REFERENCES
Fotos(codi foto)
);
```

Inserció de les dades:

```
INSERT INTO Fotografs VALUES ('7890123456', 'Jay Hamger', '555868684',
'Belgica');
INSERT INTO Fotografs VALUES ('7911234578', 'Larry Matts', '255868422',
'Canada');

INSERT INTO Rallis(codi_ralli, nom_ralli, preu) VALUES('R2005-T1-1',
'Mamifers Carnivors', 3800);
INSERT INTO Rallis(codi ralli, nom ralli, preu) VALUES('R2005-T1-2',
'Fauna Exotica', 2000);

INSERT INTO Safaris VALUES('S2005-F1-2', 'Llops nocturns', 'Amalia
Pouler' );
```

```
INSERT INTO Safaris VALUES('S2005-F2-1', 'Pumes negres', 'Amalia Pouler'
);
INSERT INTO Safaris VALUES('S2005-M1-2', 'Serps verinoses', 'Joao Sousa'
);

INSERT INTO Agrupacions VALUES('R2005-T1-1', 'S2005-F1-2');
INSERT INTO Agrupacions VALUES('R2005-T1-1', 'S2005-F2-1');
INSERT INTO Agrupacions VALUES('R2005-T1-2', 'S2005-F1-2');
INSERT INTO Agrupacions VALUES('R2005-T1-2', 'S2005-M1-2');

INSERT INTO Inscripcions VALUES ('7890123456', 'R2005-T1-1', 10);
INSERT INTO Inscripcions VALUES ('7911234578', 'R2005-T1-1', 20);
INSERT INTO Inscripcions VALUES ('7911234578', 'R2005-T1-2', 10);

INSERT INTO Fotos VALUES ('F0001', 'Serp gegant', 5);
INSERT INTO Fotos VALUES ('F0002', 'Lluita de pumes', 6);
INSERT INTO Fotos VALUES ('F0003', 'Manada salvatge', 6);

INSERT INTO Participacions VALUES('7890123456', 'R2005-T1-1', 'S2005-F1-
2', 'F0003');
INSERT INTO Participacions VALUES('7890123456', 'R2005-T1-1', 'S2005-F2-
1', 'F0002');
INSERT INTO Participacions VALUES('7911234578', 'R2005-T1-1', 'S2005-F1-
2', NULL);
INSERT INTO Participacions VALUES('7911234578', 'R2005-T1-1', 'S2005-F2-
1', NULL);
INSERT INTO Participacions VALUES('7911234578', 'R2005-T1-2', 'S2005-M1-
2', 'F0001');
```

Una vegada creades les taules i inserides les files de prova cal construir la classe que farà BDAccessor amb els mètodes per a facilitar la gestió de la connexió a la Base de Dades UOCReserva amb PostgreSQL.

Classe BDAccessor. Connexió a la Base de Dades:

```
package uocreserva;

import java.io.FileInputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Properties;

/**
 * Classe auxiliar de manegament de la Base de Dades.
 */
public class BDAccessor {

      private String dbname;
      private String host;
      private String params;
      private String port;
      private String server;
```

```
	/**
	 * Constructor per defecte
	 */
	public BDAccessor() {
		super();
		dbname = null;
		host = null;
		params = null;
		port = null;
		server = null;
	}

	/**
	 * Constructor amb tots els paràmetres.
	 * @param host La màquina on està la BD.
	 * @param port El port des d'on serveix.
	 * @param dbname El nom de la base de dades.
	 * @param server El nom del servidor.
	 * @param params Paràmetres adicionals.
	 */
	public BDAccessor(
				String host,
				String port,
				String dbname,
				String server,
				String params) {
		this.host = host;
		this.port = port;
		this.dbname = dbname;
		this.server = server;
		this.params = params;
	}

	/**
	 * Llegeix un fitxer de propietats i el retorna en un objecte Properties
	 * @param file Nom del fitxer de propietats
	 * @return Properties Objecte propietats carregat amb els continguts del
	 *                fitxer
	 * @throws IOException Si no troba el fitxer
	 */
	private static Properties loadParams( String file )
		throws IOException {
		Properties prop = new Properties();
		prop.load(new FileInputStream (file));
		return prop;
	}

	/**
	 * Obté una connexió de la BD.
	 * Primer de tot llegeix el fitxer de configuració (bd.properties).
	 * @return La connexió a la BD.
	 * @throws IOException Si no troba el fitxer de paràmetres de la connexió
	 * @throws SQLException Si no pot obtenir la connexió a BD.
	 * @throws ClassNotFoundException Si no troba la classe del driver de BD
	 */
	public Connection getConnection()
		throws SQLException, IOException, ClassNotFoundException {
	    Connection con = null;
		// Llegeix el fitxer de configuració de la BD.
		try {
			this.loadDBParams();
		} catch (IOException e1) {
```

```
                        System.err.println( "ERROR: No s'ha pogut llegir el fitxer de
paràmetres" );
                        throw e1;
                }

                // Càrrega del driver
                try {
                        Class.forName( "org.postgresql.Driver" );
                } catch (ClassNotFoundException e) {
                        System.err.println( "ERROR: No s'ha pogut carregar el driver
JDBC" );
                        throw e;
                }

                // Connexió
                try {
                   String url = "jdbc:postgresql://localhost:5432/"+ dbname;
                   con = DriverManager.getConnection( url, "postgres", "postgres" );
                }
                catch ( SQLException e ) {
                        System.err.println( "ERROR: No s'ha pogut connectar amb el
SGBD" );
                        throw e;
                }

                con.setAutoCommit( false );

                return con;
        }

        /**
         * Tanca la connexió a la BD.
         * @param con Connexió a la BD.
         */
        public static void closeConnection( Connection con ) {
                if ( con != null ) {
                        try {
                                con.close();
                        } catch ( SQLException e ) {
                                System.err.println( "Error tancant la connexió a la BD"
);
                                System.err.println( e.getMessage() );
                                System.exit( -1 );
                        }
                }
        }

        /**
         * Tanca el PreparedStatement si encara està obert.
         * @param pstmt PreparedStatement que es preten tancar.
         */
        public static void closePreparedStatement( PreparedStatement pstmt ) {
                if ( pstmt != null ) {
                        try {
                                pstmt.close();
                        } catch (SQLException e) {
                                System.err.println( "Error en tancar el
PreparedStatement");
                                System.err.println( e.getMessage() );
                        }
                }
        }
```

```
	/**
	 * Carrega el fitxer de propietats de la BD i les assigna als atributs
	 * de la classe.
	 * @throws IOException Si no llegeix el fitxer de configuració.
	 */
	private void loadDBParams()
		throws IOException {
		// Obtenim la configuració de la base de dades
		Properties prop = null;
		try {
			prop = loadParams( "bd.properties" );
			this.host   = prop.getProperty( "host" );
			this.port   = prop.getProperty( "port" );
			this.dbname = prop.getProperty( "dbname" );
			this.server = prop.getProperty( "server" );
			this.params = prop.getProperty( "params" );
		}
		catch ( IOException e ) {
			System.err.println( "No s'ha trobat el fitxer bd.properties");
			throw e;
		}
	}
}
```

Exercici 1. Creació de disparadors

En aquest exercici definirem un seguit de disparadors per tal de que la informació de la BD sigui coherent segons les següents regles de negoci:

Solució:

a) Un ral·li no pot tenir més de **5 safaris**.

Per tal de satisfer aquesta regla de negoci cal controlar que la taula *Agrupacions* no conté més de 5 safaris per ral·li. El cas evident a controlar és quan es realitzen les insercions d'agrupacions. Però, com ja indicava l'enunciat cal pensar en altres maneres que podríem acabar amb més de 5 safaris per ral·li. En aquest cas, hem de vigilar també els possibles updates de l'atribut *codi_ralli*.

Hem definit un procediment emmagatzemat, *checkNumAgrup(rally LIKE Agrupacions.codi_ralli)*, que llença una excepció si es supera el nombre de safaris per ral·li.

Finalment definim un disparador FOR EACH ROW inserida i un altre disparador FOR EACH ROW que actualitzi l'atribut *codi_ralli*. Aquests disparadors criden al procediment comentat anteriorment.

```
-------------------------------------------------------------------------

-- Descripcio: Genera una excepcio si hi ha mes de 5 safaris per ralli

-------------------------------------------------------------------------

CREATE FUNCTION checkNumAgrup () RETURNS trigger AS '
    BEGIN
```

```
    IF (SELECT COUNT(*) FROM Agrupacions WHERE codi_ralli =
NEW.codi_ralli) >= 5 THEN
            RAISE EXCEPTION ''Violació de regla de negoci - No més de 5
safaris per ralli!'';
        END IF;

        RETURN NEW;
    END;
' LANGUAGE 'plpgsql';
```

```
-------------------------------------------------------------------------
-- Descripcio: comprova que no es viola la regla de negoci al fer un
insert
-------------------------------------------------------------------------
CREATE TRIGGER InsertAgrupacions
    BEFORE INSERT ON Agrupacions
      FOR EACH ROW
EXECUTE PROCEDURE checkNumAgrup();
```

```
-------------------------------------------------------------------------
-- Descripcio: comprova que no es viola la regla de negoci al fer un
update
-------------------------------------------------------------------------
CREATE TRIGGER UpdateAgrupacions
    BEFORE UPDATE ON Agrupacions
      FOR EACH ROW
EXECUTE PROCEDURE checkNumAgrup();
```

b) Una inscripció no podrà ser eliminada un cop el fotògraf ja hagi participat en algun safari del ral·li inscrit.

En aquesta regla hem de tenir cura quan s'operi sobre la taula *Inscripcions*. Com en el cas anterior, hi ha un disparador obvi, que és quan s'intenta eliminar una inscripció, i un altre cas a tenir en compte, que és quan es realitza una actualització sobre els atributs *passaport* o *codi_ralli*. Si algun d'aquests atributs canviés sense controlar-los podríem arribar a violar la regla de negoci.

Els disparadors que definim, FOR EACH ROW en tots dos casos, acaben cridant al mateix procediment emmagatzemat *checkParticipa(passport LIKE Participacions.passaport, rally LIKE Participacions.codi_ralli)*, per tal de no haver de replicar el mateix codi en ambdós casos.

Fixeu-vos en el detall que no es crida directament a aquest procediment, sinó a un altre que acaba cridant aquest. El motiu el trobareu a l'explicació de l'apartat d.

```
-------------------------------------------------------------------------
-- Descripcio: Genera una excepcio si s'intenta eliminar una inscripcio -
-- d'algu que ja ha participat
-------------------------------------------------------------------------
```

```
CREATE OR REPLACE FUNCTION checkParticipa (participacions.passaport%type,
participacions.codi_ralli%type) RETURNS void AS '
    BEGIN
        IF (SELECT COUNT(*) FROM Participacions WHERE passaport = $1 and
codi_ralli = $2) > 0 THEN
            RAISE EXCEPTION ''Violació de regla de negoci - Fotògraf ha
anat a safari!'';
        END IF;
     END;
' LANGUAGE 'plpgsql';
```

```
-------------------------------------------------------------------------
--Descripcio: comprova la regla de negoci i actualitza el
total_recaptacio --del ralli (Exercici 1d)
-------------------------------------------------------------------------

CREATE OR REPLACE FUNCTION checkAndDecPrice () RETURNS TRIGGER AS '
        BEGIN
       PERFORM checkParticipa(OLD.passaport, OLD.codi_ralli);
        PERFORM p decPrice(OLD.codi ralli, OLD.descompte);
        RETURN NULL;
    END;
' LANGUAGE 'plpgsql';
```

```
-------------------------------------------------------------------------
--Descripcio: comprova la regla de negoci i actualitza el
--total_recaptacio del ralli (Exercici 1d)
-------------------------------------------------------------------------

CREATE OR REPLACE FUNCTION checknUpdatePrice () RETURNS TRIGGER AS '
     BEGIN
        PERFORM checkParticipa(OLD.passaport, OLD.codi ralli);
        IF ($5 <> $3) THEN
           PERFORM decPrice(OLD.codi_ralli, OLD.descompte);
           PERFORM p_incPrice(NEW.codi_ralli, NEW.descompte);
        END IF;
       RETURN NULL;
    END;
' LANGUAGE 'plpgsql';
```

```
-------------------------------------------------------------------------
-- Descripcio: comprova la regla de negoci al eliminar inscripcions i
-- actualitza el total_recaptacio d'un ralli si cal (exercici 1d)
-------------------------------------------------------------------------
CREATE TRIGGER DelInscripcions
    BEFORE DELETE ON Inscripcions
      FOR EACH ROW
EXECUTE PROCEDURE checkAndDecPrice();
```

```
-------------------------------------------------------------------------
-- Descripcio: comprova la regla de negoci al fer updates d'inscripcions
- i actualitza el total_recaptacio d'un ralli si cal (exercici 1d)
-------------------------------------------------------------------------
CREATE TRIGGER UpdateInscripcions
    BEFORE UPDATE ON Inscripcions
      FOR EACH ROW
EXECUTE PROCEDURE checknUpdatePrice();
```

c) També ens hem d'assegurar que un fotògraf no pot participar en un safari d'un ral·li si no hi està inscrit. Comproveu primer que el ral·li i el fotògraf existeixen i doneu l'excepció corresponent si calgués.

En aquest punt, el que se'ns demana és que tinguem cura de la taula *Participacions* en els casos que poguéssim incomplir la regla. Aquests casos, són evidentment el moment en el que fem una inserció a *Participacions*, però també hem de vigilar quan fem una actualització dels atributs *passaport* o *codi_ralli*.

Definirem dos disparadors per als casos comentats anteriorment. Els dos disparadors cridaran directament al procediment emmagatzemat *checkSubscrip(passport LIKE Inscripcions.passaport, rally LIKE Inscripcions.codi_ralli)*. Procediment que tal com demana l'enunciat llençarà una excepció quan no existeixi el fotògraf, no existeixi el ral·li o el fotògraf no estigui subscrit al ral·li. El motiu per realitzar totes aquestes comprovacions és que ens facilitarà la implementació de l'exercici 4 de la pràctica.

```
-------------------------------------------------------------------------
-- Descripcio: Genera una excepcio si s'intenta afegir una participacio -
-- d'un no inscrit
-------------------------------------------------------------------------
CREATE OR REPLACE FUNCTION checkSuscrip() RETURNS TRIGGER AS '
    DECLARE
        subscrit int;
        existRally int;
        existPhotographer int;
    BEGIN
        existRally := (SELECT count(*) FROM Rallis WHERE codi_ralli =
NEW.codi_ralli);
        IF (existRally = 0) THEN
            RAISE EXCEPTION ''Violació de la regla de negoci - Ralli no
existeix!'';
        END IF;
        existPhotographer := (SELECT count(*) FROM Fotografs WHERE
passaport = NEW.passaport);
        IF (existPhotographer = 0) THEN
            RAISE EXCEPTION ''Violació de la regla de negoci - Fotògraf
no existeix!'';
        END IF;
        subscrit := (SELECT count(*) FROM Inscripcions WHERE passaport =
NEW.passaport AND codi ralli = NEW.codi ralli);
        IF (subscrit = 0) THEN
```

```
            RAISE EXCEPTION ''Violació de la regla de negoci - Fotògraf
no inscrit en aquest ralli!'';
        END IF;
        RETURN NEW;
    END;
' LANGUAGE 'plpgsql';
```

```
-------------------------------------------------------------------------
-- Descripcio: Valida la regla de negoci per cada insert de
-- participacions
-------------------------------------------------------------------------
CREATE TRIGGER InsertParticipa
    BEFORE INSERT ON Participacions
      FOR EACH ROW
EXECUTE PROCEDURE checkSuscrip();
```

```
-------------------------------------------------------------------------
-- Descripcio: Valida la regla de negoci per cada update de
-- participacions
-------------------------------------------------------------------------
CREATE TRIGGER UpdateParticipa
    BEFORE INSERT ON Participacions
      FOR EACH ROW
EXECUTE PROCEDURE checkSuscrip();
```

d) Finalment, ens cal un disparador que ens mantingui l'atribut *total_recaptació* d'un ral·li actualitzat en funció de les inscripcions fetes a aquest ral·li. Segons les converses amb els responsables d'UOCReserva, el preu d'un ral·li i el descompte d'una inscripció no es modificaran mai després d'haver estat creats, per tant no cal que el disparador tingui en compte les modificacions sobre aquests atributs. També cal tenir en compte que en cap cas no es pot fer una actualització sobre l'atribut *total_recaptació* directament, ja que sempre es calcularà amb el nostre disparador.

Com de ben segur ja vàreu deduir, la raó per la que l'enunciat ens assegura que el preu d'un ral·li o el descompte d'una inscripció mai es modificaran un cop creats és per fer la implementació de la regla una mica més senzilla. Si volguéssim cobrir aquests casos hauríem de definir disparadors sobre les actualitzacions d'aquests atributs.

Un altra situació que ens podria interessar vigilar és l'actualització de l'atribut *total_recaptació* directament. Caldria controlar que aquesta actualització només pot fer-se des de dins del procediment emmagatzemat que manté la recaptació dels ral·lis.

Un cop comentats aquests dos detalls, passem a la implementació realitzada. Per tal de mantenir l'atribut degudament actualitzat ens cal poder modificar *total_recaptacio* sempre que s'afegeixi una nova inscripció, quan s'elimini una inscripció (que compleixi

la regal de negoci b, es clar) i quan es canviï el ral·li al que està inscrit un fotògraf (també si ho permet la regla de negoci b).

En l'apartat b hem definit uns disparadors per eliminació d'inscripcions i actualització de l'atribut *codi_ralli*, així que el que haurem de fer ara és modificar els procediments que criden aquests disparadors per afegir la funcionalitat que mantindrà *total_recaptacio*. Per insercions de noves inscripcions definirem un nou disparador.

Cridat per la funció checkParticipa:

```
-------------------------------------------------------------------------
-- Descripcio: Incrementa el total recaptacio quan hi ha una nova
-- inscripcio
-------------------------------------------------------------------------
CREATE OR REPLACE FUNCTION p_incPrice (Rallis.codi_ralli%type,
Inscripcions.descompte%type) RETURNS int AS '
    DECLARE
        price Rallis.preu%type;
    BEGIN
        price := (SELECT preu FROM Rallis WHERE codi_ralli = $1) * (1 -
$2/100);
        UPDATE Rallis SET total_recaptacio = total_recaptacio + price
WHERE codi_ralli = $1;
        RETURN 0;
    END;
' LANGUAGE 'plpgsql';
```

Cridat pel *trigger* AddInscripcio:

```
-------------------------------------------------------------------------
-- Descripcio: Incrementa el total recaptacio quan hi ha una nova
--inscripcio
-------------------------------------------------------------------------
CREATE OR REPLACE FUNCTION t_incPrice () RETURNS TRIGGER AS '
    DECLARE
        newRec Rallis.preu%type;
    BEGIN
        newRec := (SELECT preu FROM Rallis WHERE codi_ralli =
NEW.codi_ralli) * (1 - NEW.descompte/100);
        UPDATE Rallis SET total recaptacio = total recaptacio + newRec
WHERE codi_ralli = NEW.codi_ralli;
        RETURN NEW;
    END;
' LANGUAGE 'plpgsql';
```

```
-------------------------------------------------------------------------
-- Descripcio: Decrementa el total recaptacio quan s'elimina una
-- inscripcio
-------------------------------------------------------------------------
```

```
CREATE OR REPLACE FUNCTION decPrice (Rallis.codi_ralli%type,
Inscripcions.descompte%type) RETURNS int AS '
    DECLARE
        price Rallis.preu%type;
    BEGIN
        price := (SELECT preu FROM Rallis WHERE codi_ralli = $1) * (1 -
$2/100);
        UPDATE Rallis SET total recaptacio = total recaptacio - price
WHERE codi_ralli = $1;
        RETURN 0;
    END;
' LANGUAGE 'plpgsql';
```

```
-----------------------------------------------------------------------
-- Descripcio: Actualitza total_recaptacio al afegir una inscripcio
-- Aquest atribut tambe cal actualitzar-lo en modificacions i
-- updates, pero el trigger es declara a l'Exercici 1b
-----------------------------------------------------------------------
CREATE TRIGGER AddInscripcio
    BEFORE INSERT ON Inscripcions
      FOR EACH ROW
EXECUTE PROCEDURE t incPrice();
```

Exercici 2. Procediment que retorna informació de ral·lis

Cal anar amb cura que cobrim tots els casos a l'hora d'implementar aquest procediment, especialment quan hi ha 0 inscripcions en un ral·li. Si féssim una inner join entre Rallis i Inscripcions perdríem els ral·lis que tenen 0 inscripcions, així que la manera de fer-ho es obtenir primer la informació dels ral·lis, ordenats per nom i comprovar per cada un si compleixen la condició del nombre d'inscripcions o no.

Finalment, en els cas dels ral·lis que compleixen la condició, obtenim els nombre de safaris que té i retornem totes les dades. Si obtenim el nombre de safaris un cop ja sabem si el nombre d'inscripcions del ral·li compleix la condició s'optimitza una mica més el nombre de consultes a fer a la BD.

```
-----------------------------------------------------------------------
-- Descripcio: Funcio auxiliar per obtenir el nombre de safaris que conte
-- un ralli
-----------------------------------------------------------------------
CREATE OR REPLACE FUNCTION safarisInRally (Agrupacions.codi_ralli%type)
RETURNS INT AS '
BEGIN
RETURN (SELECT COUNT(*) FROM Agrupacions WHERE codi_ralli = $1);
END;
' language 'plpgsql';
```

```
-------------------------------------------------------------------------
-- Descripcio: creem un tipus per utilitzar en la seguen funcio,
-- listRallies
-------------------------------------------------------------------------
CREATE TYPE r_listRallies as (rally char(50), price int, inscriptions
int, safarisInRally int);
```

```
-------------------------------------------------------------------------
-- Descripcio: Retorna el llistat de rallis
-------------------------------------------------------------------------
CREATE OR REPLACE FUNCTION listRallies(int) RETURNS SETOF r_listRallies
AS '
DECLARE
  retorn r_listRallies%ROWTYPE;
  actual RECORD;
  inscriptions INT;
BEGIN
  FOR actual IN
    SELECT nom_ralli, preu, codi_ralli
    FROM rallis
    ORDER BY nom_ralli
  LOOP
    inscriptions := (SELECT COUNT(*) FROM Inscripcions WHERE codi_ralli =
actual.codi_ralli);
    IF (inscriptions >= $1) THEN
      retorn.rally := actual.nom_ralli;
      retorn.price := actual.preu;
      retorn.inscriptions := inscriptions;
      retorn.safarisInRally := (select
safarisInRally(actual.codi ralli));
      RETURN NEXT retorn;
    END IF;
  end loop;
  return;
end;
' language 'plpgsql';
```

Exercici 3. Inserció de ral·lis a la BD

En aquest exercici, els aspectes més importants a considerar són:

- La utilització de sentències preparades del JDBC (*PreparedStatements*), per que es podran reaprofitar cada vegada que s'introdueixi una nova agrupació dins de la mateixa execució del programa que dóna d'alta un ral·li i totes les seves agrupacions amb safaris. Si no fem ús d'aquest tipus de sentències, a cada nova inserció d'una agrupació, es recalcularà l'estratègia (el conjunt d'instruccions de baix nivell necessàries) per dur a terme la inserció. Amb la utilització de sentències preparades, el càlcul de l'estratègia només es realitza un cop, i s'aplica tantes vegades com agrupacions amb safaris vulguem inserir. Per tant, en la inserció de agrupacions, és molt més eficient la utilització de sentències preparades.

- En canvi, per la inserció d'un ral·li, no està justificat utilitzar sentències preparades, ja que per cada execució del programa és donarà d'alta un únic ral·li, i per tant, el cost del càlcul previ de l'estratègia no es veu compensat per l'execució continuada d'insercions de ral·lis.
- La utilització de transaccions. El fet de donar d'alta un nou ral·li, comporta la inserció de les seves agrupacions amb safaris. Per tant, des d'un punt de vista conceptual, el que acabem de descriure constitueix una unitat atòmica d'execució. La manera de recollir aquestes execucions atòmiques en una BD és mitjançant el concepte de transacció. Davant de qualsevol error (per exemple, si intentem inserir una agrupació amb un safari que no existeix), tota la feina realitzada contra la BD s'haurà de desfer. En aquest cas, si fem ús de transaccions, simplement caldrà indicar que la transacció finalitza amb la cancel·lació (*rollback*) dels seus resultats. I llavors serà l'SGBD qui s'encarrega automàticament de desfer els canvis produïts com a conseqüència de l'execució del nostre programa (cancel·lació de les agrupacions introduïdes i del propi ral·li). Si no es produeix cap error, llavors la transacció finalitza la seva execució amb una petició de *commit*. Si no fem ús de transaccions, llavors serem nosaltres, manualment, per programa, els que haurem d'anular els resultats produïts (esborrat de les agrupacions introduïdes i esborrat del propi ral·li). Això és ineficient i complica la lògica de l'aplicació. Es important delegar tot allò que sigui possible en l'SGBD.
- Hem delegat en l'SGBD totes les comprovacions que aquest pot fer com, per exemple, que l'agrupació que volem donar d'alta sigui amb un safari existent (recordeu que a la taula `Agrupacions`, l'atribut `codi_safari` és clau forana a la taula Safaris), entre d'altres.

Amb la classe RallyCreation s'insereixen ral·lis a la Base de dades.

Classe RallyCreation. Inserció de dades:

```
package uocreserva;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Iterator;
import java.util.List;

/**
 * Pràctica 1 - Exercici 3.
 * Classe que permet crear un nou ral·li amb les seves agrupacions a safaris.
 */
public class RallyCreation {

    private PreparedStatement pstmtInsAgrupation;

    /**
     * Constructor per defecte.
     */
    public RallyCreation() {
        pstmtInsAgrupation = null;
    }
```

```
/**
   * Fa commit de la transacció en curs. Si es produeix un error, l'ignora.
   * @param con Connexió a la BD
   */
   private void commit(Connection con) {
          try {
              con.commit();
          } catch (SQLException e) {
              e.printStackTrace();
          }
   }

   /**
   * Fa rollback de la transacció en curs. Si es produeix un error, l'ignora.
   * @param con Connexió a la BD
   */
   private void rollback(Connection con) {
          try {
              con.rollback();
          } catch (SQLException e) {
              e.printStackTrace();
          }
   }

   /**
    * Obté les dades del ral·li i els safaris que el composen
    * @return Ral·li amb els seus safaris
    * @throws FileNotFoundException No ha trobat el fitxer de dades del ral·li
    * @throws IOException Error en la lectura del fitxer de dades del ral·li
    */
   public RallyWithSafaris getRallyWithSafaris ()
          throws FileNotFoundException, IOException {

          String file = "exercici3.txt";
          List data = Utilities.readFile(file, ";");

       Rally rally = new Rally ();
       rally.setCode((String) data.get(0));
       rally.setName((String) data.get(1));
       rally.setPrice((new Integer((String) data.get(2))).intValue());

       List safariList = data.subList(3,data.size());

       RallyWithSafaris rallyWithSafaris =
           new RallyWithSafaris (rally, safariList);

       return rallyWithSafaris;
   }

   /**
    * Inserta a la BD el <code>rally</code>
    * @param con Connexió a la BD
    * @param rally Ral·li a insertar
    * @throws SQLException Error en la inserció
    */
   public void insertRally (Connection con, Rally rally)
          throws SQLException {
       Statement stmt = con.createStatement();
       StringBuffer sqlInsert = new StringBuffer ();

       sqlInsert.append("INSERT INTO rallis (codi_ralli, nom_ralli, preu) ");
       sqlInsert.append("VALUES ( '");
```

```
            sqlInsert.append(rally.getCode());
            sqlInsert.append("', '");
            sqlInsert.append(rally.getName());
            sqlInsert.append("', ");
            sqlInsert.append(rally.getPrice());
            sqlInsert.append(" )");

               stmt.executeUpdate( sqlInsert.toString() );
               stmt.close();
        }

        /**
         * Inserta a la BD les diferents agrupacions del <code>rally</code> amb
         * <code>safariList</code>
         * @param con Connexió a la BD
         * @param rally Ral·li a insertar
         * @param safariList Llista dels safaris que tenen agrupacions
         *        amb <code>rally</code>
         * @throws SQLException Error en la inserció
         */
        public void insertAgrupation (Connection con, Rally rally, List safariList)
               throws SQLException {
            if (safariList != null) {
                if ( pstmtInsAgrupation == null ) {
                      String sqlInsert =
                          "INSERT INTO Agrupacions (codi_ralli, codi_safari) " +
                          "VALUES ( ?, ? )";
                      pstmtInsAgrupation = con.prepareStatement( sqlInsert );
                   }
                   Iterator list = safariList.iterator();
                   while (list.hasNext()) {
                       pstmtInsAgrupation.clearParameters();
                          pstmtInsAgrupation.setString(1, rally.getCode());
                          pstmtInsAgrupation.setString(2, (String) list.next());
                          pstmtInsAgrupation.executeUpdate();
                   }
            }
        }

    /**
     * Mètode principal.
     * @param args Paràmetres de la crida
     */
      public static void main(String[] args) {
          RallyCreation rc = new RallyCreation();
          Connection con = null;
          boolean error = false;

          // Obté una connexió de la Base de Dades
          BDAccessor db = new BDAccessor();
          try {
            con = db.getConnection();
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(-1);
        }

         try {
            List rallyWithSafarisList = null;
            RallyWithSafaris rallyWithSafaris = null;

            rallyWithSafaris = rc.getRallyWithSafaris();
```

```
            rc.insertRally(con, rallyWithSafaris.getRally());
            System.out.println("Ral·li creat");

            rc.insertAgrupation(con, rallyWithSafaris.getRally(),
                                    rallyWithSafaris.getSafariList());
            System.out.println("Agrupacions creades");
        } catch (Exception e1) {
            System.out.println("ERROR    ("    +    e1.getClass()    +    "):    "    +
e1.getMessage());
            error = true;
        }

        if (error) {
            rc.rollback(con);
            System.out.println("Transacció avortada");
        } else {
            rc.commit(con);
            System.out.println("Transacció guardada");
        }

        BDAccessor.closePreparedStatement(rc.pstmtInsAgrupation);
        BDAccessor.closeConnection(con);
        System.exit(0);
      }
}
```

La sortida és:

```
Ral·li creat
Agrupacions creades
Transacció guardada
```

Exercici 4. Inserció de participacions a la BD

Alguns dels aspectes exposats en la solució de l'exercici anterior, són aplicables en aquest exercici. A continuació els expliquem, per aquest cas concret:

- La utilització de transaccions. De manera equivalent a un ral·li i les seves agrupacions amb safaris, en aquest exercici tenim la participació d'un fotògraf en un safari d'un ral·li i, opcionalment, la foto que pot presentar a concurs. En cas d'existir aquesta foto es donarà d'alta, i per tant, això també constitueix una unitat atòmica. Així doncs, si hi ha hagut algun error en la inserció de la foto (cas que existeixi) o bé de la participació, la transacció finalitzarà amb la cancel·lació (*rollback*) dels resultats, i caldrà desfer les insercions que s'hagin pogut fer. Per contra, si tot acaba correctament es finalitzarà amb la petició de *commit*.
- Pel que fa a les insercions, tant de la foto (en el cas que existeixi) com de la participació, no està justificada la utilització de sentències preparades (*PreparedStatments*) ja que hi haurà una única inserció com a màxim en cada una de les taules corresponents, i seria un cas equivalent al que succeïa en la inserció d'un ral·li en l'exercici anterior.

L'altre tema que calia argumentar era la gestió dels possibles errors en la creació de la participació. Els temes més rellevants en aquest àmbit són:

- Per controlar els possibles errors utilitzarem, igual que per l'exercici anterior, les restriccions definides en l'SGBD i les comprovacions que aquest fa per garantir la integritat de les seves dades. Així doncs, només caldrà definir els mecanismes pertinents per, en funció dels diferents tipus d'error que generi l'SGBD, indicar el missatge que desitgem en cada ocasió. Això evita haver de duplicar, en el programa, les comprovacions que ja s'estan fent, i per tant, empitjorar el rendiment d'aquest.
- Per fer això, tenim diferents opcions, i la que explicarem aquí és la que s'ha considerat més elegant, ja que es basa en obtenir el tipus de restricció violada i en funció d'això, seleccionar la restricció concreta. Un cop detectat el tipus de restricció violada, podem prosseguir a la identificació de la restricció concreta. En el cas de l'error indicat pel disparador creat en l'exercici 1c, ens pot indicar que el ral·li indicat no existeix, que el fotògraf indicat no existeix o bé que el fotògraf no està inscrit en el ral·li. Per diferenciar-los només podem comparar el missatge d'error obtingut amb cadascun dels missatges que retorna el disparador que hem creat amb anterioritat. Pel cas de la violació de clau primària, els errors es produeixen si ja existeixen, prèviament, la foto o bé la participació que intentem crear. Per controlar quina restricció s'ha violat us oferíem el mètode `constraintName`, que donat el missatge d'una excepció retornava el nom de la restricció que es violava, de manera que comparant el resultat obtingut amb el nom de les dues restriccions possibles teníem identificat l'error que calia indicar. Per l'últim cas, violació de clau forana, només hi ha una única possibilitat de violació i és quan el safari no pertany al ral·li, però per garantir que l'error que volem indicar és el que realment s'ha produït, també es fa la comprovació amb el nom de la restricció (igual que pel cas de restriccions de clau primària). Això es justifica tenint en compte el supòsit que el disparador estigués desactivat i el fotògraf no existís, ja que al intentar crear la participació hi hauria una excepció per violació de clau forana. Si no comprovem la restricció que s'ha violat, indicaríem que el safari no pertanyi al ral·li, i en canvi l'error produït seria per que el fotògraf no existeix a la taula `Fotografs`.

Cal fer insercions de participació a la Base de Dades. Es donarà d'alta la participació d'un determinat fotògraf a un safari concret d'un determinat ral·li. El programa és ParticipationCreation.

Classe ParticipationCreation. Inserir una participació d'un fotògraf en un safari d'un ral·li:

```
package uocreserva;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.SQLException;
import java.util.List;

/**
 * Pràctica 1 - Exercici 4.
```

```
 * Classe que permet gestionar la participació d'un fotògraf a un safari d'un
ral·li.
 */
public class ParticipationCreation {

    private static String INITIAL_CONSTRAINT_SEPARATOR = "(";
    private static String FINAL_CONSTRAINT_SEPARATOR = ")";
    private static String USER_CONSTRAINT_SEPARATOR = ".";

    private static int PK_ERROR = -268;
    private static int FK_ERROR = -691;
    private static int TRIGGER_ERROR = -746;

    private static String TRIGGER_NOT_RALLY = "Ralli no existeix";
    private static String TRIGGER_NOT_PHOTOGRAPHER = "Fotograf no existeix";
    private static String TRIGGER_NOT_REGISTERED = "Fotograf no inscrit en aquest
ralli";

    private static String CONSTRAINT_PK_PARTICIPATION = "pk_participacions";
    private static String CONSTRAINT_PK_PHOTO = "pk_fotos";
    private static String CONSTRAINT_FK_AGRUPATION = "fk_agrupacions";

    /**
     * Constructor per defecte.
     */
    public ParticipationCreation() {
    }

       /**
       * Fa commit de la transacció en curs. Si es produeix un error, l'ignora.
       * @param con Connexió a la BD
       */
       private void commit(Connection con) {
              try {
                  con.commit();
              } catch (SQLException e) {
                  e.printStackTrace();
              }
       }

       /**
       * Fa rollback de la transacció en curs. Si es produeix un error, l'ignora.
       * @param con Connexió a la BD
       */
       private void rollback(Connection con) {
              try {
                  con.rollback();
              } catch (SQLException e) {
                  e.printStackTrace();
              }
       }

       /**
        * Obté les dades de la participació i la foto associada si s'escau.
        * @return Participació amb la seva foto, si existeix
        * @throws FileNotFoundException No ha trobat el fitxer de dades de la
participació
        * @throws IOException Error en la lectura del fitxer de dades de la
participació
        */
       public Participation getParticipation ()
              throws FileNotFoundException, IOException {
```

```
            String file = "exercici4.txt";
            List data = Utilities.readFile(file, ";");

          Participation participation = new Participation ();
          participation.setPhotographer((String) data.get(0));
          participation.setRally((String) data.get(1));
          participation.setSafari((String) data.get(2));
          if (data.size() > 3) {
              Photo photo = new Photo ();
              photo.setCode((String) data.get(3));
              photo.setName((String) data.get(4));
              photo.setScore((new Integer((String) data.get(5))).intValue());

              participation.setPhoto(photo);
          }
          return participation;
      }

      /**
       * Inserta a la BD la <code>photo</code>
       * @param con Connexió a la BD
       * @param photo Foto a insertar
       * @throws SQLException Error en la inserció
       */
      public void insertPhoto (Connection con, Photo photo)
            throws SQLException {
          Statement stmt = con.createStatement();
          StringBuffer sqlInsert = new StringBuffer ();

          sqlInsert.append("INSERT INTO fotos (codi_foto, nom_foto, puntuacio)
");
          sqlInsert.append("VALUES ( '");
          sqlInsert.append(photo.getCode());
          sqlInsert.append("', '");
          sqlInsert.append(photo.getName());
          sqlInsert.append("', ");
          sqlInsert.append(photo.getScore());
          sqlInsert.append(" )");

          stmt.executeUpdate( sqlInsert.toString() );
            stmt.close();
      }

      /**
       * Inserta a la BD la <code>participation</code>
       * @param con Connexió a la BD
       * @param participation Participació a insertar
       * @throws SQLException Error en la inserció
       */
      public void insertParticipation (Connection con, Participation
participation)
            throws SQLException {
          Statement stmt = con.createStatement();
            StringBuffer sqlInsert = new StringBuffer ();

            sqlInsert.append("INSERT INTO participacions (passaport, codi_ralli,
codi_safari");

            // Si la participació té foto la inclourem l'atribut
            // en la inserció, en cas contrari no es tindrà en compte
            if (participation.getPhoto() != null)
                    sqlInsert.append(", codi_foto");
```

```
            sqlInsert.append(") VALUES ( '");
            sqlInsert.append(participation.getPhotographer());
            sqlInsert.append("', '");
            sqlInsert.append(participation.getRally());
            sqlInsert.append("', '");
            sqlInsert.append(participation.getSafari());

            // Només en cas que existeixi foto s'inclourà el seu identificador
            if (participation.getPhoto() != null) {
                sqlInsert.append("', '");
                  sqlInsert.append(participation.getPhoto().getCode());
            }
            sqlInsert.append("')");

            stmt.executeUpdate( sqlInsert.toString() );
            stmt.close();
        }

        /**
         * Donat un missatge d'error de la BD busca el nom de la restricció
         * @param message Missatge d'error de la BD
         * @return Nom de la restricció que ha fallat
         */
        private String constraintName (String message) {
            String userConstraint,constraint = null;

            int initial = message.indexOf(INITIAL_CONSTRAINT_SEPARATOR);
          int end =  message.indexOf(FINAL_CONSTRAINT_SEPARATOR);
            if (initial != -1 && end != -1) {
                userConstraint = message.substring(initial+1, end);
                initial = userConstraint.indexOf(USER_CONSTRAINT_SEPARATOR);
                if (initial != -1)
                    constraint = userConstraint.substring(initial+1);
                else
                    constraint = userConstraint;
            }
            return constraint;
        }

        /**
         * Processa l'excepció generada per la BD i retorna el missatge
         * d'error adequat.
         * @param se Excepció de la BD
         * @return Missatge d'error de l'excepció
         */
        private String processExceptionMessage (SQLException se) {
            String message = "Participació incorrecta. ";
            // Excepció llançada pel nostre trigger
          if (se.getErrorCode() == TRIGGER_ERROR) {
            if (se.getMessage().endsWith(TRIGGER_NOT_RALLY))
                return (message + "No existeix el ral·li.");
            else if (se.getMessage().endsWith(TRIGGER_NOT_PHOTOGRAPHER))
                return (message + "No existeix el fotògraf.");
            else if (se.getMessage().endsWith(TRIGGER_NOT_REGISTERED))
                return (message + "Fotògraf no inscrit.");
          }
          // Excepció de violació de clau primaria
          else if (se.getErrorCode() == PK_ERROR) {
              String restriccio = constraintName(se.getMessage());
              if (restriccio != null) {
                  if (restriccio.equalsIgnoreCase(CONSTRAINT_PK_PARTICIPATION))
                      return (message + "Ja existeix la participació.");
                  else if (restriccio.equalsIgnoreCase(CONSTRAINT_PK_PHOTO))
```

```
                    return (message + "Ja existeix la foto.");
                }
            }
            // Excepció de violació de clau forana
            else if (se.getErrorCode() == FK_ERROR) {
                String restriccio = constraintName(se.getMessage());
                if (restriccio != null &&
                    restriccio.equalsIgnoreCase(CONSTRAINT_FK_AGRUPATION))
                        return (message + "El safari no pertany al ral·li.");
            }
            return ("Error intern en l'execució del programa");
          }

          /**
         * Mètode principal.
         * @param args Paràmetres de la crida
         */
        public static void main(String[] args) {
            ParticipationCreation pc = new ParticipationCreation();
              Connection con = null;
              boolean error = false;

              // Obté una connexió de la Base de Dades
              BDAccessor db = new BDAccessor();
              try {
                con = db.getConnection();
            } catch (Exception e) {
                System.err.println(e.getMessage());
                System.exit(-1);
            }

            try {
              Participation participation = null;

              participation = pc.getParticipation();

                if (participation.getPhoto() != null) {
                    pc.insertPhoto(con, participation.getPhoto());
                    System.out.println("Foto creada");
                }

                pc.insertParticipation(con, participation);
                System.out.println("Participació creada");
            } catch (SQLException se) {
                System.out.println (pc.processExceptionMessage(se));
                error = true;
            } catch (Exception e1) {
                System.out.println("Error intern en l'execució del programa");
                error = true;
            }

            if (error) {
                pc.rollback(con);
                System.out.println("Transacció avortada");
            } else {
                pc.commit(con);
                System.out.println("Transacció guardada");
            }

            BDAccessor.closeConnection(con);
            System.exit(0);
        }
}
```

La resposta a l'execució d'aquest programa és:

```
Foto Creada
Participació Creada
Transacció guardada
```

Recordeu que per a realitzar la segona part de la pràctica és necessari usar la solució oficial de la primera part de la pràctica.

SEGONA PART

Com a resultat de la segona part de la pràctica haureu de lliurar el codi font de totes les classes Java (**`*.java`**), els procediments que creeu (***.sql**) i un fitxer Word (**`practica2.doc`**) on heu d'afegir totes les explicacions que considereu necessàries.

Si heu de crear mètodes nous en qualsevol de les classes els haureu d'indicar en el fitxer Word.

Exercici 1. Programa Java per consultar les dades del concurs de fotografia

Necessitem un programa Java que ens pugui mostrar les dades del concurs per a poder atorgar els premis corresponents. Llavors, ens demanen que realitzem un programa que mostri la classificació final del concurs de fotografia d'un determinat ral·li, del qual coneixem el seu codi. El codi del ral·li, a diferència dels exercicis anteriors, el sol·licitarà l'aplicació en el moment de la seva execució, havent-lo d'introduir per teclat. No us preocupeu de verificar si el ral·li existeix o no, podem assumir que sempre existirà.

Us recordem que per poder optar al concurs de fotografia (i per tant, per poder formar part del resultat del programa que us demanem) els fotògrafs han de participar, com a mínim, a la meitat dels safaris que formen part del ral·li i han de lliurar com a mínim una fotografia. Per tant, els fotògrafs no han de lliurar obligatòriament una foto a cada participació de cada safari d'un ral·li.

El fotògraf guanyador del concurs és el fotògraf que verifiqui les condicions prèvies i que més puntuació acumuli per les fotos lliurades a les seves participacions. En el llistat només han d'aparèixer els fotògrafs que entren a concurs segons les condicions prèvies. En el cas de que hi hagués més d'un fotògraf amb la mateixa acumulació de punts, el guanyador el decidiria el jurat segons el seu propi criteri, així que no cal que us preocupeu de fer res en aquest cas.

Si necessiteu informació addicional, a la solució de la primera part de la pràctica del semestre passat, estan explicades acuradament les bases del concurs de fotografia (arxiu `05002_SOLPRA1_20041.doc`).

El resultat es vol ordenat ascendentment per ordre de puntuació total. Més concretament cal mostrar:

```
NomRalli NomFotògraf PuntuacióTotal TotalFotosLliurades NomFotoPuntuacióMax PuntuacióMaxFoto
============================================================
```

El contingut de cada fila ha de ser:

NomRalli contindrà el nom del ralli del que volem obtenir el llistat.

NomFotògraf serà el nom d'un fotògraf que ha participat al ral·li especificat. Cal assegurar-se que compleix la condició d'haver lliurat com a mínim una fotografia i d'haver participat, almenys, en la meitat dels safaris que componen el ral·li.

PuntuacióTotal és la suma de les puntuacions de cada fotografia que el fotògraf ha lliurat per aquest ral·li. El llistat ha d'estar ordenat ascendentment per aquest camp, i el guanyador és el que tingui més punts.

TotalFotosLliurades conté el número de fotografies que el fotògraf ha lliurat per aquest ral·li concret.

NomFotoPuntuacióMax és el nom de la fotografia amb puntuació màxima que ha aconseguit el fotògraf en aquest ral·li. En el cas que hi hagi més d'una foto amb la mateixa puntuació màxima, es retornarà qualsevol d'elles.

PuntuacióMaxFoto és la puntuació de la fotografia que s'ha indicat a NomFotoPuntuacióMax.

Aquest programa Java haurà d'estar en una classe anomenada `ContestList`, de la qual us facilitem l'esquelet.

Classe	Descripció	Estat
`uocreserva.ContestList`	Mètodes per obtenir les dades del concurs de fotografia.	Cal implementar les consultes i la connexió a la BD.
`uocreserva.ContestData`	Per facilitar l'accés a les dades que formen el llistat del concurs.	No cal que modifiqueu res.

Es demana: El programa en Java.

Si heu de fer alguna observació respecte al disseny o a la implementació, podeu fer-ho en el fitxer Word.

Exercici 2. Modificació del programa anterior

Resulta que després d'un temps de funcionament de la UOCReserva, l'organització ens ha demanat si en lloc de realitzar la consulta des d'una aplicació en Java, podíem implementar un procediment emmagatzemat que executés la consulta, i que el programa cridés a aquest procediment.

La capçalera del procediment que hem d'implementar és:

```
CREATE PROCEDURE listContest (rally LIKE Rallis.codi_ralli)
  RETURNING CHAR(50), CHAR(50), INTEGER, INTEGER, CHAR(50), INTEGER;
```

Donat que només variarà la manera d'obtenir les dades, el format d'aquest programa Java serà molt similar al anterior, i haurà d'estar en una classe anomenada `ContestListSPL`, de la qual també us facilitem l'esquelet. Caldrà que utilitzeu també la classe `ContestData`, definida en el programa anterior, per la gestió de la informació.

Classe	Descripció	Estat
`uocreserva.ContestListSPL`	Mètodes per obtenir les dades del concurs de fotografia.	Cal implementar les consultes i la connexió a la BD.

Es demana:

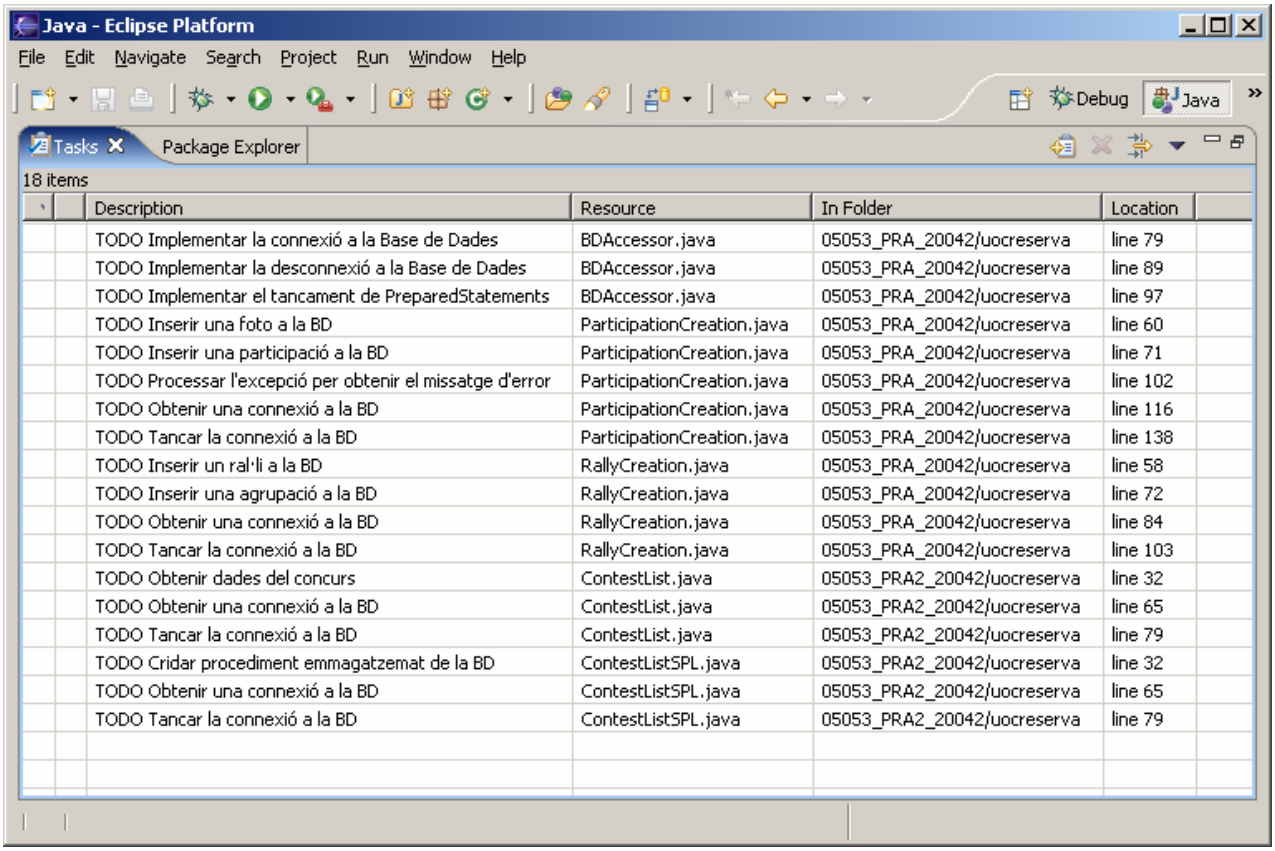
- El procediment emmagatzemat i programa en Java que utilitza aquest procediment.
- Resposta a la pregunta: - Per quina raó poden voler usar un procediment emmagatzemat en lloc de la solució que els vam oferir originalment en l'Exercici 1?

Si heu de fer alguna observació respecte al disseny o a la implementació, podeu fer-ho en el fitxer Word.

Llistat de tasques a fer

En el codi Java s'han afegit comentaris per indicar les parts que queden per implementar. Algunes eines són capaces de fer servir aquestes comentaris per construir una llista de tasques pendents.

Per a que us feu una idea, amb el codi que us hem lliurat, Eclipse treu un llistat com el que es mostra a continuació:



Description	Resource	In Folder	Location
TODO Implementar la connexió a la Base de Dades	BDAccessor.java	05053_PRA_20042/uocreserva	line 79
TODO Implementar la desconnexió a la Base de Dades	BDAccessor.java	05053_PRA_20042/uocreserva	line 89
TODO Implementar el tancament de PreparedStatements	BDAccessor.java	05053_PRA_20042/uocreserva	line 97
TODO Inserir una foto a la BD	ParticipationCreation.java	05053_PRA_20042/uocreserva	line 60
TODO Inserir una participació a la BD	ParticipationCreation.java	05053_PRA_20042/uocreserva	line 71
TODO Processar l'excepció per obtenir el missatge d'error	ParticipationCreation.java	05053_PRA_20042/uocreserva	line 102
TODO Obtenir una connexió a la BD	ParticipationCreation.java	05053_PRA_20042/uocreserva	line 116
TODO Tancar la connexió a la BD	ParticipationCreation.java	05053_PRA_20042/uocreserva	line 138
TODO Inserir un ral·li a la BD	RallyCreation.java	05053_PRA_20042/uocreserva	line 58
TODO Inserir una agrupació a la BD	RallyCreation.java	05053_PRA_20042/uocreserva	line 72
TODO Obtenir una connexió a la BD	RallyCreation.java	05053_PRA_20042/uocreserva	line 84
TODO Tancar la connexió a la BD	RallyCreation.java	05053_PRA_20042/uocreserva	line 103
TODO Obtenir dades del concurs	ContestList.java	05053_PRA2_20042/uocreserva	line 32
TODO Obtenir una connexió a la BD	ContestList.java	05053_PRA2_20042/uocreserva	line 65
TODO Tancar la connexió a la BD	ContestList.java	05053_PRA2_20042/uocreserva	line 79
TODO Cridar procediment emmagatzemat de la BD	ContestListSPL.java	05053_PRA2_20042/uocreserva	line 32
TODO Obtenir una connexió a la BD	ContestListSPL.java	05053_PRA2_20042/uocreserva	line 65
TODO Tancar la connexió a la BD	ContestListSPL.java	05053_PRA2_20042/uocreserva	line 79

Materials

Els materials que es necessiten per fer la pràctica són bàsicament el PostgreSQL i el JDK, tots dos proporcionats com a part del material docent. A més, disposeu dels continguts teòrics en suport paper dels diferents mòduls que composen l'assignatura, destacant el mòdul 8, on s'expliquen els fonaments bàsics de la programació SQL i Java .

Solució de la Pràctica. Segona Part

UOCReserva

Juntament amb aquest document us adjuntem el codi de la solució proposada. Aquest codi està estructurat en directoris de la mateixa manera que ho estava l'enunciat i és una extensió del mateix.

El directori `sql` conté el codi SQL i SPL, amb la definició del procediment demanat.

El directori `uocreserva` conté el codi Java complet.

Exercici 1. Programa Java per consultar les dades del concurs de fotografia

La solució plantejada per aquest exercici, i que explicarem a continuació, passa per fer varies consultes i enllaçar els resultats d'aquestes sense haver d'utilitzar *outer joins*.

Donat el codi del ral·li, del qual volem obtenir el llistat resultant del concurs, el primer que hem de saber és el número d'agrupacions que té amb safaris, ja que es requerirà per saber si un fotògraf ha participat en, com a mínim, la meitat dels safaris del ral·li. Fixeu-vos que aquesta informació l'obtenim inicialment i l'emmagatzemem en una variable, per evitar haver-la de consultar a la base de dades cada vegada que vulguem comprovar si un fotògraf compleix el requisit de participació.

A partir d'aquí, obtenim la llista de fotògrafs que han participat en algun safari del ral·li i que tenen alguna foto. Exactament obtenim el nom del ral·li, el nom del fotògraf, la suma de la puntuació de totes les fotos del fotògraf, el número de fotos, la foto de puntuació màxima i el passaport del fotògraf (necessari per identificar el fotògraf i que utilitzarem més endavant). Aquesta llista és el resultat de fer una consulta de les taules `Rallis`, `Fotografs`, `Participacions` i `Fotos` vinculades entre si amb *joins*, agrupant les files pel codi del ral·li i passaport del fotògraf i ordenades pel resultat de la suma de la puntuació de les fotos, tal i com demanen els requisits del concurs.

D'aquesta llista caldrà eliminar aquells fotògrafs que no compleixin el requisit de la participació. Així doncs, per cada fotògraf del resultat anterior, consultem les participacions de cadascun d'ells en el ral·li en qüestió. Per fer aquesta consulta ens cal el passaport que identifica cada fotògraf, i per això també l'hem obtingut en la consulta anterior. Un cop sabem el número de participacions del fotògraf i coneixent, també, el número d'agrupacions amb safaris del ral·li, que hem obtingut prèviament, comprovem si el fotògraf pot participar al concurs. En aquesta comprovació, en lloc de dividir les agrupacions per la meitat, s'ha optat per multiplicar per dos el valor de la participació del fotògraf. El motiu que ens ha dut a fer-ho, és que la multiplicació per potències de dos és molt més eficient que qualsevol altra operació, ja que treballant amb representacions binaries significa fer únicament un desplaçament a l'esquerra.

En el cas, que el fotògraf que estem tractant compleixi el requisit de participació, caldrà obtenir el nom de la foto amb puntuació màxima. Per aquesta consulta utilitzarem la informació de la que ja disposem, com és el passaport del fotògraf, el codi del ral·li i la puntuació de la foto que busquem. Així la consulta només ens dóna el nom de la foto desitjada o una llista de fotos que compleixen la condició i de les quals

en podrem escollir una qualsevol. Al fer-ho en aquest moment, només estarem buscant aquesta informació per aquells fotògrafs que ja sabem que han d'aparèixer en el llistat, i farem únicament el número de consultes necessàries.

Un cop tenim tota la informació d'un participant del concurs que compleix amb tots els requisits, creem un element `ContestData` amb aquesta informació i l'afegim al llistat per tal que es pugui mostrar per pantalla.

És important alliberar els recursos un cop ja no els necessitem, en el nostre cas, cal fer un *close* dels elements *ResultSet* i *Statement* que anem utilitzant, un cop han finalitzat la seva funció.

Classe ContestList. Consulta les dades del concurs de fotografia.

```
package uocreserva;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

/**
 * Pràctica 2 - Exercici 1.
 * Llista les dades del concurs de fotografia d'un ral·li.
 */
public class ContestList extends PrintList {

    /**
     * Constructor per defecte.
     */
    public ContestList() {
        super();
    }

    /**
     * Obté les dades del concurs de fotografia d'un ral·li
     * @param con Connexió a la base da dades
     * @param rallyCode Codi del ral·li que fa el concurs
     * @return Dades del concurs de fotografia
     * @throws SQLException Excepció de la BD
     */
    public List getContestData(Connection con, String rallyCode)
      throws SQLException {
        List contestData = new ArrayList();
        String rallyName;
        String photographerName;
        int totalPoints;
        int totalPhotos;
        String photoMaxPointName;
        int maxPoints;
        String photographerPassport;
        int safaris = 0;
```

```
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // Obtenim el número de safaris del ral·li
              StringBuffer selectSafaris = new StringBuffer ();
              selectSafaris.append("SELECT COUNT(*) FROM Agrupacions
WHERE codi_ralli='");
              selectSafaris.append(rallyCode);
              selectSafaris.append("'");

              stmt = con.createStatement();
              rs = stmt.executeQuery(selectSafaris.toString());

            if (rs.next())
                safaris = rs.getInt( 1 );
            rs.close();
            stmt.close();

            // Busquem tots els fotògrafs amb alguna foto del ral·li
            // i obtenim també les dades que necessitem pel llistat
            StringBuffer selectContest = new StringBuffer ();
            selectContest.append("SELECT r.nom_ralli, ft.nom,
SUM(fo.puntuacio), ");
            selectContest.append("COUNT(*), MAX(fo.puntuacio),
ft.passaport ");
            selectContest.append("FROM rallis r, fotografs ft, fotos fo,
participacions p ");
            selectContest.append("WHERE ft.passaport=p.passaport AND
p.codi_ralli = r.codi_ralli ");
            selectContest.append("AND p.codi_foto=fo.codi_foto AND
r.codi_ralli='");
            selectContest.append(rallyCode);
            selectContest.append("' ");
            selectContest.append("GROUP BY r.codi_ralli, r.nom_ralli,
ft.nom, ft.passaport ");
            selectContest.append("ORDER BY 3 DESC");

            stmt = con.createStatement();
              rs = stmt.executeQuery(selectContest.toString());

              while (rs.next()) {
                  rallyName = rs.getString( 1 );
                  photographerName = rs.getString( 2 );
                  totalPoints = rs.getInt( 3 );
                  totalPhotos = rs.getInt( 4 );
                  maxPoints = rs.getInt( 5 );
                  photographerPassport = rs.getString( 6 );

                  // Per cada fotògraf comprovem que la seva participació
sigui correcta
                  StringBuffer selectParticipation = new StringBuffer ();
                  selectParticipation.append("SELECT COUNT(*) FROM
Participacions WHERE passaport='");
                  selectParticipation.append(photographerPassport);
                  selectParticipation.append("' AND codi_ralli='");
                  selectParticipation.append(rallyCode);
```

```
                    selectParticipation.append("'");

                    Statement stmt2 = con.createStatement();
                    ResultSet rs2 =
stmt2.executeQuery(selectParticipation.toString());

                    int participation = 0;
                    if (rs2.next())
                        participation = rs2.getInt( 1 );
                    rs2.close();
                    stmt2.close();

                    // La participació ha de ser, com a mínim,
                    // la meitat dels safaris del ral·li
                    if (participation*2 >= safaris) {
                        StringBuffer selectPhoto = new StringBuffer ();
                        selectPhoto.append("SELECT nom_foto ");
                        selectPhoto.append("FROM Fotos f, Participacions p
");
                        selectPhoto.append("WHERE f.codi_foto=p.codi_foto
AND p.passaport='");
                        selectPhoto.append(photographerPassport);
                        selectPhoto.append("' AND codi_ralli='");
                        selectPhoto.append(rallyCode);
                        selectPhoto.append("' AND puntuacio=");
                        selectPhoto.append(maxPoints);

                        stmt2 = con.createStatement();
                          rs2 = stmt2.executeQuery(selectPhoto.toString());

                          photoMaxPointName = null;
                          if (rs2.next())
                              photoMaxPointName = rs2.getString( 1 );
                          rs2.close();
                          stmt2.close();

                          ContestData cd = new ContestData( rallyName,
photographerName,
                                  totalPoints, totalPhotos,
photoMaxPointName, maxPoints );
                          contestData.add( cd );
                    }
                }
                rs.close();
                stmt.close();
            } catch (SQLException se) {
                try {
                    if (rs != null) {
                        rs.close();
                    }
                    if (stmt != null) {
                        stmt.close();
                    }
                } catch (SQLException se2) {
                    throw se2;
                }
                throw se;
```

```
        }

        return contestData;
    }

    /**
     * Mostra per pantalla les dades del concurs de fotografia
     * @param contestData Dades del concurs de fotografia
     * @return Indica si el llistat pintat està buit
     */
    public boolean printContestData (List contestData) {
        int[] size = { 50, 50, 10, 10, 50, 10 };
        String[] labels = { "Nom Ral·li", "Nom Fotògraf", "Puntuació
Total",
                "Total Fotos Lliurades", "Nom Foto Puntuació Màxima",
"Puntuació Màxima" };
        String[] attributes = { "rallyName", "photographerName",
                    "totalPoints", "totalPhotos", "PhotoMaxPointName",
"maxPoints" };

        this.setAttributes( attributes );
        this.setLabels( labels );
        this.setSize( size );

        return (this.printList( contestData ));
    }

    /**
     * Mètode principal.
     * @param args Paràmetres de la crida
     */
    public static void main(String[] args) {
        ContestList cl = new ContestList ();
          Connection con = null;
          BDAccessor db = new BDAccessor();

          // Obté una connexió de la Base de Dades
          try {
            con = db.getConnection();
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(-1);
        }

        try {
            List contestData = null;
            String rallyCode = null;
            rallyCode = Utilities.getString("Introduiu el codi del
ral·li");
            contestData = cl.getContestData(con, rallyCode);
            boolean empty = cl.printContestData(contestData);
            if (empty)
                  System.out.println("Llistat sense dades.");
        } catch (Exception e1) {
            e1.printStackTrace();
        }
```

```
            BDAccessor.closeConnection(con);
            System.exit(0);
        }
}
```

A d'executar el programa demana:

```
Introduiu el codi del ral·li:
```

Introduïm un codi, per exemple: '*R2005-T1-1*'

I ens treu el llistat demanat:

Nom Ral·li	Nom Fotògraf	Puntuació Total	Total Fotos Lliurades	Nom Foto Puntuació Màxima	Puntuació Màxima
Mamifers Carnivors	Jay Hamger	12	2	Manada salvatge	6

Exercici 2. Modificació del programa anterior

En aquest exercici se'ns demanava implementar el mateix comportament que l'exercici anterior, utilitzant procediments emmagatzemats, i justificar la conveniència d'un procediment emmagatzemat.

Per la implementació del procediment emmagatzemat, s'ha optat per crear un procediment auxiliar que, donat un fotògraf, un ral·li i una puntuació ens dóna el nom d'una foto que el fotògraf ha fet en el ral·li i té la puntuació indicada. Aquest procediment s'utilitzarà per obtenir els nom de les fotos de puntuació màxima que mostrarem en el llistat del concurs. Com ja hem vist, en cas que hi hagi més d'una foto amb la mateixa puntuació, donarem el primer nom que retorni la consulta.

De manera equivalent a la solució anterior, obtenim prèviament el número de safaris que té el ral·li per poder comprovar el requisit de participació de cada fotògraf. A continuació, obtenim el llistat de possibles fotògrafs, que tenen alguna foto del ral·li, amb la informació necessària pel llistat, tal i com explicàvem prèviament. Així doncs, per cada fotògraf comprovem la seva participació i en cas de complir les condicions necessàries, obtenim el nom de la foto amb puntuació màxima. Aquí també tenim en compte l'optimització de la multiplicació per dos enlloc de la divisió.

Pel que fa a la justificació en la utilització del procediment emmagatzemat, cal tenir en compte varies consideracions, de les quals n'hi ha dues de fonamentals.

D'entrada, un dels principals motius, pels quals ens podem plantejar aquest canvi, és la millora del rendiment de la base de dades, ja que els procediments emmagatzemats es guarden precompilats en aquesta, de manera que l'estratègia d'execució es calcula en el moment de la seva creació i no cada vegada que s'utilitzen. Per tant, en els casos en que la seva utilització és elevada el increment de rendiment és considerable.

L'altre motiu, que ens justificaria la utilització de procediments emmagatzemats, és l'estalvi en el trànsit de dades a través de la xarxa (considerant que el servidor de base de dades no es troba en la mateixa màquina que la utilitzada per executar el programa client, i que seria l'arquitectura habitual en un entorn real). Com hem pogut observar en l'exercici 1, per obtenir el resultat final del concurs, hem hagut d'executar

varies consultes, algunes d'aquestes s'han hagut de repetir, en alguns casos, tantes vegades com possibles participants del concurs hi havia. Això genera un consum elevat de recursos de xarxa, ja que per obtenir el resultat final hem hagut de transmetre, a través d'aquesta, tots els resultats que s'anaven obtenint de les diferents consultes, i on possiblement, un cert nombre d'aquests valors s'han acabat descartant per no complir amb els requisits demanats.

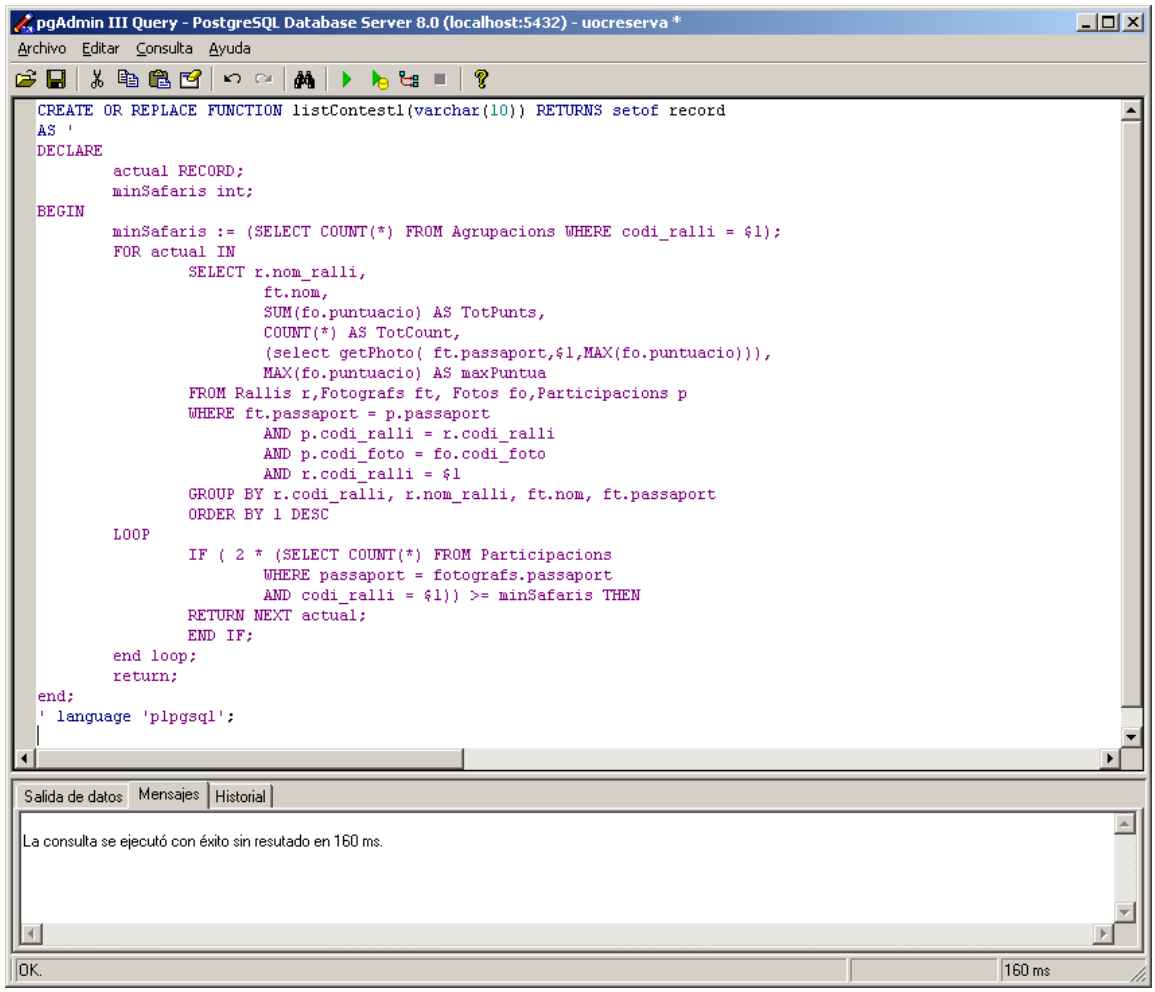
Algun dels altres factors, que podrien haver influït en la creació d'un procediment emmagatzemat, serien l'encapsulament de les operacions per obtenir el resultat del concurs, ja que això permetria modificar fàcilment algunes regles, com per exemple definir un ordre específic en cas d'empat en la puntuació de les fotos, o modificar els requisits de participació. També es podria considerar força interessant, en el supòsit que vulguem tenir diferents canals de distribució d'aquesta informació, permeten consultar el resultat del concurs des de múltiples entorns, com ara una pàgina web o utilitzant tecnologia WAP per telefonia mòbil. En aquests casos, no caldria desenvolupar, per cada un dels diferents canals, la lògica de negoci necessària per obtenir el llistat del concurs, i es reutilitzaria el codi del procediment emmagatzemat.

```
-----------------------------------------------------------------
-- Descripcio: Funcio auxiliar per obtenir la foto amb la
-- puntuacio indicada
-----------------------------------------------------------------
CREATE OR REPLACE FUNCTION getPhoto (Participacions.passaport%type,
Rallis.codi ralli%type, Fotos.puntuacio%type) RETURNS Fotos.nom foto%type
AS '
    BEGIN
        RETURN (SELECT nom foto
                    FROM Fotos f, Participacions p
                    WHERE f.codi_foto = p.codi_foto and p.passaport = $1
and p.codi_ralli = $2 and puntuacio = $3 limit 1);
    END;
' LANGUAGE 'plpgsql';
```

```
-----------------------------------------------------------------
-- Descripcio: Retorna el llistat sobre el concurs
-----------------------------------------------------------------
CREATE OR REPLACE FUNCTION listContest1(varchar(10)) RETURNS setof record
AS '
DECLARE
  actual RECORD;
  minSafaris int;
  BEGIN
  minSafaris := (SELECT COUNT(*) FROM Agrupacions WHERE codi_ralli = $1);
  FOR actual IN
    SELECT  r.nom_ralli,
            ft.nom,
            SUM(fo.puntuacio) AS TotPunts,
            COUNT(*) AS TotCount,
            (select getPhoto( ft.passaport,$1,MAX(fo.puntuacio))),
             MAX(fo.puntuacio) AS maxPuntua
    FROM    Rallis r,Fotografs ft, Fotos fo,Participacions p
    WHERE   ft.passaport = p.passaport
```

```
        AND p.codi_ralli = r.codi_ralli
        AND p.codi_foto = fo.codi_foto
        AND r.codi_ralli = $1
    GROUP BY r.codi_ralli, r.nom_ralli, ft.nom, ft.passaport
    ORDER BY 1 DESC
  LOOP
    IF ( 2 * (SELECT COUNT(*) FROM Participacions
    WHERE passaport = fotografs.passaport
       AND codi_ralli = $1)) >= minSafaris THEN
     RETURN NEXT actual;
    END IF;
  end loop;
  return;
end;
' language 'plpgsql';
```

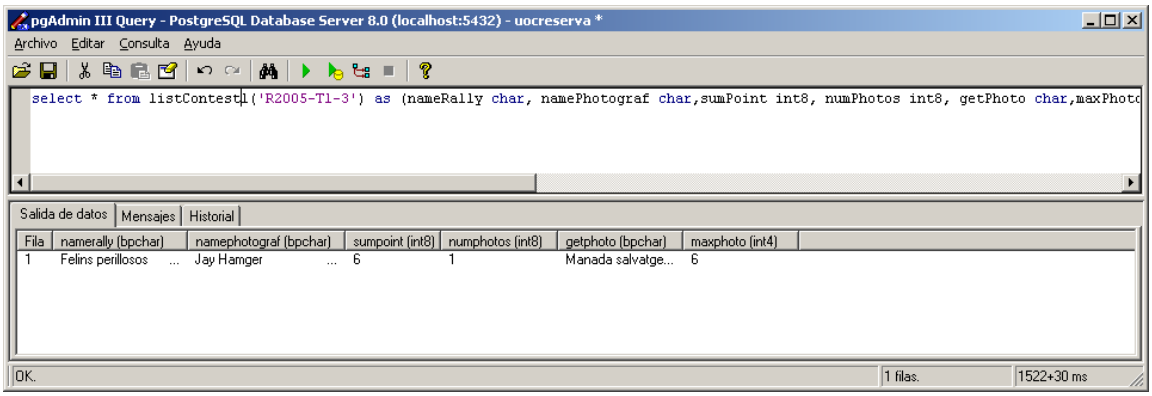
Veiem l'execució en l'editor de pgAdmin:



Cal executar per veure el resultat:

```
select * from listContest1('R2005-T1-3')  as (nameRally char,
namePhotograph char, sumPoint int8, numPhotos int8, getPhoto char,
maxPhoto int4);
```

Perquè ens relacioni el nom del rally, el nom del fotograf, la puntuació total, les fotos llurades, el nom de la foto amb puntuació màxima i la puntuació màxima:



I amb la següent funció declarem un cursor pels resultats anteriors:

```
create or replace function listContest(char(10)) returns refcursor as '
declare ref refcursor;
begin
open ref for select * from listContest1($1) as (nameRally char,
namePhotograph char, sumPoint int8, numPhotos int8, getPhoto char,
maxPhoto int4);
return ref;
end;
'LANGUAGE 'plpgsql';
```

Classe ContestListSPL. Consulta les dades del concurs de fotografia.

```
package uocreserva;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

/**
 * Pràctica 2 - Exercici 2.
 * Permet cridar un procediment emmagatzemat que llista les dades del
 * concurs de fotografia d'un ral•li.
```

```
 */
public class ContestListSPL extends PrintList {

    /**
     * Constructor per defecte.
     */
    public ContestListSPL() {
    }

    /**
     * Obté les dades del concurs a partir d'un procediment emmagatzemat
     * @param con Connexió a la base da dades
     * @param rallyCode Codi del ral•li que fa el concurs
     * @return Dades del concurs de fotografia
     * @throws SQLException Excepció en la gestió del procediment
     * emmagatzemat
     */
    public List getContestData(Connection con, String rallyCode)
      throws SQLException {
        List contestData = new ArrayList();
        String rallyName;
        String photographerName;
        int totalPoints;
        int totalPhotos;
        String photoMaxPointName;
        int maxPoints;

        CallableStatement cstmt = null;
        ResultSet rs = null;

        try
        {
              cstmt = con.prepareCall( "{ ? = call listContest( '" +
rallyCode + "' ) }" );
              cstmt.registerOutParameter(1, Types.OTHER);
              cstmt.execute();
              rs = (ResultSet) cstmt.getObject(1);

              while ( rs.next() ) {
                  rallyName = rs.getString( 1 );
                  photographerName = rs.getString( 2 );
                  totalPoints = rs.getInt( 3 );
                  totalPhotos = rs.getInt( 4 );
                  photoMaxPointName = rs.getString( 5 );
                  maxPoints = rs.getInt( 6 );

                  ContestData cd = new ContestData( rallyName,
photographerName,totalPoints,totalPhotos,photoMaxPointName,maxPoints );
                  contestData.add( cd );
              }
              rs.close();
              cstmt.close();
      } catch (SQLException se) {
            try {
                if (rs != null) {
                    rs.close();
                }
```

```
                if (cstmt != null) {
                    cstmt.close();
                }
            } catch (SQLException se2) {
                throw se2;
            }
            throw se;
        }
        return contestData;
    }

    /**
     * Mostra per pantalla les dades del concurs de fotografia
     * @param contestData Dades del concurs de fotografia
     * @return Indica si el llistat pintat està buit
     */
    public boolean printContestData (List contestData) {
        int[] size = { 50, 50, 10, 10, 50, 10 };
        String[] labels = { "Nom Ral•li", "Nom Fotògraf", "Puntuació
Total","Total Fotos Lliurades", "Nom Foto Puntuació Màxima", "Puntuació
Màxima" };
        String[] attributes = { "rallyName", "photographerName",
        "totalPoints", "totalPhotos", "PhotoMaxPointName", "maxPoints" };

        this.setAttributes( attributes );
        this.setLabels( labels );
        this.setSize( size );

        return (this.printList( contestData ));
    }

    /**
     * Mètode principal.
     * @param args Paràmetres de la crida
     */
    public static void main(String[] args) {
        ContestListSPL cl = new ContestListSPL ();
          Connection con = null;
          BDAccessor db = new BDAccessor();

          // Obté una connexió de la Base de Dades
          try {
            con = db.getConnection();
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(-1);
        }

        try {
            List contestData = null;
            String rallyCode = null;
            rallyCode = Utilities.getString("Introduiu el codi del
ral•li");
            contestData = cl.getContestData(con, rallyCode);
            boolean empty = cl.printContestData(contestData);
            if (empty)
                  System.out.println("Llistat sense dades.");
```

```
        } catch (Exception e1) {
            e1.printStackTrace();
        }

        BDAccessor.closeConnection(con);
        System.exit(0);
    }
}
```

A d'executar el programa demana:

```
Introduiu el codi del ral·li:
```

Introduïm un codi, per exemple: '*R2005-T1-3*'

Resultat:

```
Nom Ral·li          Nom Fotògraf   Puntuació Total  Total Fotos Lliurades  Nom Foto Puntuació Màxima  Puntuació Màxima
==================  =============  ===============  =====================  =========================  ================
Felins perillosos   Jay Hamger     6                1                      Manada salvatge            6
```