Estudi i avaluació de les funcionalitats de PostgreSQL

Ivo Plana Vallvé
Enginyeria en Informàtica

M. Elena Rodríguez González

10/01/2006

Agraïments

En aquest mon canviant, de ràpids oblits i de feixuga saturació d'informació, m'agradaria seguir oferint els meus agraïments...

als meus pares, per les hores que m'han condonat. Per totes aquelles tasques que m'han estalviat amb la il·lusió de veure com cada estona era dedicada a l'esforç d'assolir nous horitzons.

A la companya d'estudis que m'ha fet replantejar cada cosa que ja donava per assolida. Per totes aquelles coses de les que hem tingut que prescindir, i per totes aquelles altres de les que hem gaudit.

A tots els amics que han comprés les meves absències, els meus mals humors, i que han estat capaços d'aguantar *el tema* d'aquests últims sis anys.

A la persona que em va animar a fer una enginyeria. A la persona a la que li dec el present que ja gaudeixo.

m'agradaria afegir...

els meus agraïments al rector Ferraté i a tot l'equip docent i administratiu de la UOC, per haver fet possible els estudis virtuals i per aconseguir que pugui dir amb orgull:
He estudiat l'Enginyeria Informàtica a la UOC!

Resum

L'objectiu del projecte és fer un resum de les funcionalitats i la idoneïtat del sistema de gestió de base de dades PostgreSQL, per a facilitar la possible implantació en l'àmbit docent universitari.

Tot recollint els coneixements adquirits durant la realització del Treball de Fi de Carrera de l'Enginyeria Tècnica Informàtica de Sistemes, realitzat a la Universitat oberta de Catalunya a la tardor de 2004, es va plantejar la possibilitat de fer un estudi del PostgreSQL per a verificar que compleix tots aquells punts que requereix un Sistema de Gestió de Base de Dades per a poder considerar-se com a tal, i poder servir així d'eina per a desenvolupar en la pràctica, els casos teòrics de les assignatures relacionades amb l'àrea de les Bases de Dades d'una Enginyeria Tècnica Informàtica, o una Enginyeria Informàtica.

Al llarg de la present memòria s'estudiarà la sintaxi bàsica de les instruccions elementals, els camins d'execució i d'optimització de les sentències SQL, les propietats transaccionals i d'integritat concretes del PostgreSQL, fins als detalls de com utilitzar les eines del mateix programari per a fer còpies de seguretat, o per a restaurar el sistema desprès d'una fallada. Tot això, s'ha complementat amb la verificació de la compatibilitat del programari en diversos sistemes operatius, amb l'elecció i prova en profunditat d'un client SQL, i amb la confecció dels manuals d'ús i instal·lació de tot aquests materials, en els diferents sistemes operatius contemplats.

Al llarg del projecte es comprovarà que PostgreSQL pot substituir almenys part del programari utilitzat a l'actualitat a l'àmbit docent, aportant fins i tot valor afegit, tant per característiques, com pel futur que es preveu de cada sistema gestor de bases de dades en el mon empresarial.

Índex

Annexes

1.- Història

1.1.- Introducció

PostgreSQL agafa el seu nom i les seves característiques bàsiques en ser recollit el projecte Postgres95 per la *Comunitat del Programari Lliure*. L'evolució d'aquest, amb els afegits del llenguatge de consulta SQL, fa que assoleixi el seu nom definitiu.

El concepte de Base de Dades (BD) relacional és relativament nou. El desenvolupament de tota la gamma de programari d'aquesta branca de la informàtica ha evolucionat bàsicament des de dos inicis clarament diferenciats; per un costat va començar IBM amb el seu System R, i per l'altra el Ingres de la Universitat de Berkeley a Califòrnia. Més tard va aparèixer Oracle, el System R va evolucionar fins a l'actual DB2, i paral·lelament va sorgir tot una nova sèrie de fabricants de Sistemes de Gestió de Bases de Dades (SGBD), descendents directes de la primigènia Ingres.

1.2.- Ingres i altres

A l'any 1973 Michael Stonebraker i Eugene Wong van començar a pensar en la necessitat d'un SGBD de característiques avançades. Gran part de la idea inicial va provenir dels *papers* que IBM va publicar sobre el projecte que estava desenvolupant, el System R. Amb un sistema de rotació de programadors, i amb alguns ajuts d'institucions publiques (DARPA, ARO, NSF, NESL), es va portar a terme tot el desenvolupament del Ingres.

El maquinari sobre el que funcionava el Ingres era força modest, ja que s'executava sobre ordinadors PDP11. El diferent enfocament en quant a segment de mercat amb IBM (el System R només funcionava en grans mainframes), va fer que Ingres assolis gran popularitat, i conseqüentment quota de mercat.

El fet de que tingués un tipus de llicència de distribució BSD, va fer addicionalment que molts altres desenvolupadors agafessin aquest sistema com a base per a fer nous sistemes comercials. Així, en pocs anys va aparèixer Informix, Sybase, Non Stop SQL i altres, essent tots evolucions d'Ingres.

Informix va ser el segon SGBD en volum de mercat (per darrera de l'evolucionada bases de dades de IBM – ara ja DB2), cosa que va fer que en passar aquesta companyia per uns incidents econòmics derivats de problemes en la seva gestió (1997), IBM aprofites per a adquirir-la.

El propi èxit del projecte Ingres, va ofegar als desenvolupadors d'aquest programari en un cicle continu de manteniment. Com que aquesta no era la finalitat acadèmica del projecte, des de Berkeley es va decidir donar aquest per acabat.

Al 1994 Computer Associates (CA) va comprar a una de les companyia que comercialitzaven el Ingres (ASK Corporation), una de les evolucions més avançades del moment. Donat que la seva posterior comercialització no va tenir l'èxit esperat per CA, a l'any 2004 es van tornar a alliberar el codi del projecte amb llicència CATOSL (una llicència similar a la BSD), aconseguint així que la comunitat del programari lliure es fes càrrec de les evolucions següents.

1.3.- PostgreSQL

Al donar Stonebraker per acabada l'etapa de comercialització del Ingres (des de la companyia Relational Technologies, Inc), torna al 1985 a Berkeley per a desenvolupar un nou concepte de SGBDs.

La problemàtica de transformar el disseny conceptual original d'un nou desenvolupament informàtic, al seu equivalent SQL relacional, resultava cada vegada més evident en aquelles èpoques. S'esperava poder dissenyar un SGBD amb tipus nadius, amb unes característiques mínimes d'herència, i fer fins i tot que ell mateix fes el control de les restriccions imposades amb les relacions.

Des d'un principi s'evita utilitzar com a base del nou desenvolupament el codi font de Ingres, ja que si bé la *forma* del nou projecte era similar a l'anterior, el *fons* tenia un enfocament completament diferent.

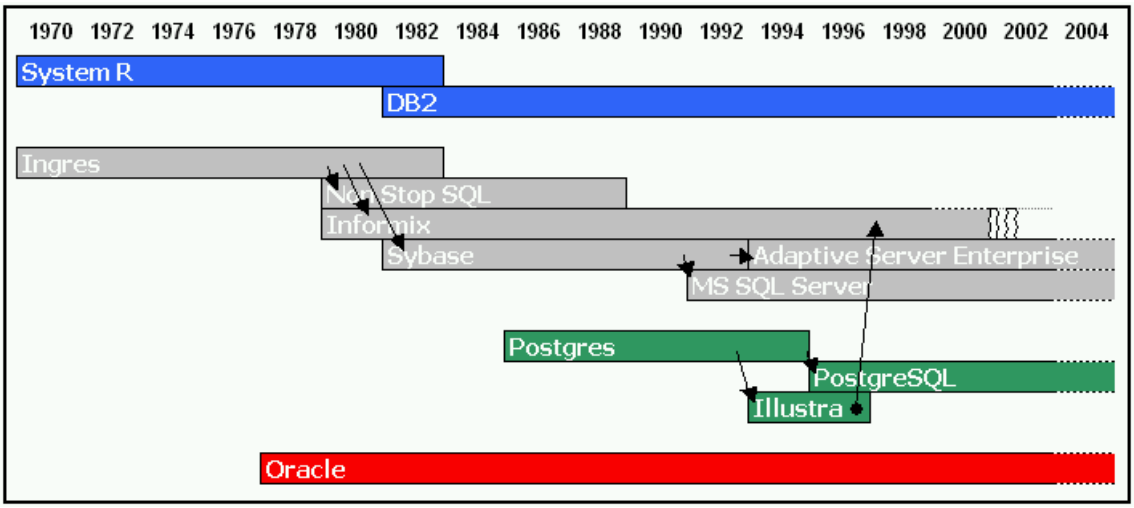
Els conceptes de funcionament van començar a veure la llum l'any 1986. Al 1988 el primer prototipus ja era funcional. La primera versió era publicada al 1989.
Versió darrera versió es van anar millorant i afegint funcionalitats, fins arribar a la versió 4 (Postgres95).

A l'igual que va passar amb l'Ingres, l'èxit del projecte va fer que les peticions de noves funcionalitats i de manteniment sobrepassés les possibilitats de l'equip de desenvolupament, pel que es va lliurar aquesta última versió, i es va decidir donar el projecte per finalitzat.

Malgrat que Stonebraker va crear l'empresa Illustra Information per a comercialitzar Postgres, tot el projecte va ser alliberat sota llicència BSD, per a que pogués servir de base per a futurs desenvolupaments.

En aquest punt, dos estudiants graduats de la mateixa Berkeley (Andrew Yu i Jolly Chen), van reescriure part del Postgres, per a que en lloc d'utilitzar el llenguatge de consulta original QUEL, acceptes el ja gairebé estàndard SQL.

Donades les importàncies dels canvis fets, i que aquest ja era mantingut per la comunitat del programari lliure, es va decidir també fer evolucionar el nom fins a l'actual conegut: PostgreSQL.



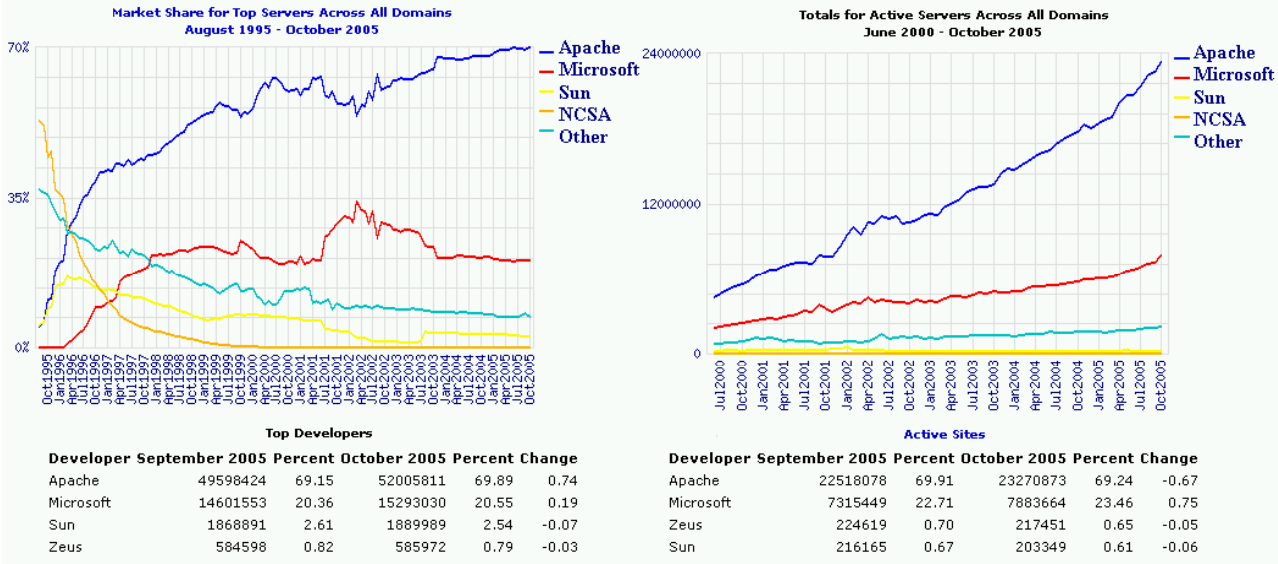
2.- PostgreSQL en el món acadèmic

Tal com s'ha justificat a la planificació del present Projecte de Fi de Carrera, PostgreSQL a part de ser Programari Lliure (amb llicència d'ús BSD), és 'estrictament' ACID, és multi-Sistema Operatiu, és multi-maquinari, té diverses comunitats que l'evolucionen i li donen suport, i hi han diverses empreses que ofereixen manteniment i serveis. És un dels desenvolupaments de Programari Lliure dignes d'admiració, tant per la seva història inicial, com per l'evolució que va tenir desprès de que es finalitzés el seu desenvolupament a Berkeley.

A l'haver estat PostgreSQL iniciat en un entorn universitari precisament amb l'objectiu de demostrar els conceptes teòrics que s'esperaven dels SGBD futurs, aquest implementa totes les funcionalitats habituals, i fins i tot, ha estat pioner en la utilització de determinats components, ja que al no haver-se desenvolupat segons criteris comercials, es va poder prioritzar el model teòric vers un ràpid retorn de la inversió.

El seu rendiment estable i continuat (demostrat a l'apartat de Benchmarks), l'alta concurrència de processos que permet (gràcies al MVCC), la seva versatilitat, l'existència dels procediments emmagatzemats i dels disparadors, l'existència del connectors JDBC/ODBC, fa que la seva utilització hagi sobrepassat l'àmbit teòric i que hagin empreses que la utilitzen com a component principal dels seus sistemes (NASA en el projecte SEQUOIA, BASF en el back-office del seu sistema de comandes de fertilitzants, etc).

Donat el constant creixement de la popularitat d'aquest programari i al fet de que companyies de la categoria d'IBM, Novell, o Fujitsu estan apostant pel seu ús, es creu que en un futur seguirà les tendències d'èxit que ja s'estan veient en algun altre programari concret.



El servidor web Apache és un bon exemple de l'èxit del programari lliure.
Actualment, més del 70% dels servidors d'internet actius estan funcionant amb aquest programa!

En un món en que les fusions de grans companyies ja és habitual, els avantatges de disposar del codi font del programari que s'utilitza és clara, ja que s'evita dependre de cap fabricant per a tenir la continuïtat garantida d'un producte estratègic per la subsistència de l'empresa d'avui en dia.

En el cas d'utilitzar programari lliure, malgrat desaparegués l'equip principal de desenvolupament del programa, la inversió feta fins al moment estaria assegurada, ja que l'SGBD (per exemple) seguiria tenint un manteniment i evolució garantit. També s'ha de considerar que no s'està subjecte al fabricant per a disposar dels mòduls per a un idioma en concret, o de certes característiques que siguin molt especifiques, ja que al ser el codi font públic, es pot desenvolupar o fer-ho desenvolupar lliurement.

El fet de d'introduir a l'àmbit empresarial programari lliure, fa que el teixit empresarial de la zona/país millori, ja que a banda d'evitar-se l'exportació de divises, es creen proveïdors locals de tecnologia i manteniment.

Si les expectatives d'evolució exposades són encertades, el mercat laboral dels propers anys necessitarà gran quantitat de tècnics especialistes en Programari Lliure. D'entre les diferents especialitzacions que caldrien en l'àmbit dels SGBD destacaríem la d'administrador de l'SGBD (o DBA per les seves sigles en anglès), la del Dissenyador Analista amb coneixements del programari concret, o la del programador en les seves diferents especialitzacions i variants pròpies.

El fet de que PostgreSQL incorpori moltes característiques avançades és molt positiu per l'àmbit educatiu, ja que utilitzant aquest programari com a eina d'aprenentatge s'assoliran uns coneixements que d'utilitzar altres SGBD no es podrien consolidar. Així, si una persona que hagi utilitzat aquest programari hagués d'utilitzar un altre SGBD (Programari Lliure o comercial), no tindria mancances de coneixements, ja que únicament s'hauria d'adaptar a les limitacions d'aquest altre sistema, o a les petites diferències de sintaxi existents.

3.- Sentències de manipulació

En aquest apartat no es pretén repetir el hi ha en els manuals de PostgreSQL, únicament es pretén fer un resum ràpid de les sentències més habituals, amb les opcions que s'utilitzen de forma més usual.

Per ordre d'ús en començar a utilitzar PostgreSQL.

3.1.- Create Database

S'utilitza per a crear la Base de Dades (BD).
En fer la connexió amb l'SGBD s'haurà de seleccionar la BD que s'utilitza, i a partir d'aquí, quan es crea una taula, o es faci una consulta, es farà sobre les dades que hi ha en aquesta.
Una taula pot existir amb el mateix nom a diferents BD, ja que estan físicament separades.

Sintaxi bàsica:

```
CREATE DATABASE nomBaseDades [ WITH LOCATION = 'cami_d'acces' ]
```

Exemple:

```
CREATE DATABASE ProvesUOC;
```

3.2.- Create Table

A l'interior de la base de dades crearem taules per a emmagatzemar dades segons les seves característiques. Els atributs de cada columna de cada taula definiran el tipus de dades que podran emmagatzemar. Es podran afegir clàusules addicionals a la sintaxi de la creació de la taula, per a especificar restriccions que s'hauran de complir en inserir, esborrar i modificar les dades.

Sintaxi bàsica:

```
CREATE TABLE NomTaula (nomColumna1 tipusColumna1,
                       nomColumna2 tipusColumna2,.... );
```

On els tipus bàsics que es poden utilitzar a tipusColumna poden ser:

char(longitud)	Cadena de caràcters de longitud fixa.
varchar(longitud)	Cadena de caràcters de longitud variable.
Bool	Pot contenir els valors booleans true/false.
Date	Per emmagatzemar la data (sense l'hora).
Timestamp	Per emmagatzemar la data, hora i segons, amb zona horària inclosa.
Float4	Per emmagatzemar números en coma flotant.
Float8	Per emmagatzemar números en coma flotant (doble precisió)

Int2	Per emmagatzemar números enters (32.768 a –32.767).
Int4	Per emmagatzemar números enters (2.147.483.648 a –2147483647)

Els tipus addicionals que afegeix el propi PostgreSQL (entre d'altres) poden ser:

Money	Per emmagatzemar quantitats monetàries (2 decimals).
Box(x1,y1, x2,y2)	Per a emmagatzemar rectangles.
Circle (centre, radi)	Per a emmagatzemar cercles.
Line (x1,y1, x2,y2)	Per a emmagatzemar línies.
Point (x,y)	Per a definir un punt a l'espai.
Poligon	Per a emmagatzemar les coordenades de polígons.
Inet	Per a emmagatzemar adreces IP.

S'ha de destacar que al tenir PostgreSQL com a característica 'base' la possibilitat de definir nous tipus, la llista es pot ampliar tant com convingui.

En el moment de la creació de la taula també es poden assignar característiques (restriccions) que les dades han de complir.

Si podem traspassar la feina de fer aquestes comprovacions bàsiques a la mateix base de dades, serà impossible que per una errada de programació o per una errada de manipulació de dades ad-hoc, transgredim aquestes directrius, tot evitant inconsistències de dades o violacions de les regles de negoci.

Exemple:

```
CREATE TABLE Clients (
      CIF CHAR(10) PRIMARY KEY,
      nomComercial VARCHAR(40),
      deute INT8,
      codiPostal CHAR(4)
            CONSTRAINT credit CHECK (Deute < 1000));
```

Creem una taula amb les columnes necessàries per a contenir les dades bàsiques dels clients, i indiquem que el CIF serà la clau principal que s'utilitzarà per a localitzar a aquests (al indicar que és clau principal es crea automàticament un índex per accelerar l'accés, i la restricció de que ha de ser *unique*), i que el possible deute d'un Client ha de ser sempre inferior a 1000.

Amb aquesta última restricció, assegurarem que en intentar fer una operació que excedeixi aquesta xifra en sumar una quantitat, falli, i retorni així l'error conforme 'està a màxim de crèdit'.

3.3.- Insert Into

Amb aquesta clàusula es poden inserir dades a les taules. És important comprovar el resultat de l'operació, ja que la base de dades pot retornar un codi d'error, si s'incompleix alguna de les restriccions indicades durant la creació de la taula.

Sintaxi:

```
INSERT INTO nomTaula (nomCamp1, nomCamp2, ...)
     VALUES (valorsPelCamp1, valorsPelCamp2, ....)
```

S'ha d'anotar que els noms dels camps són opcionals, si s'insereixen tants valors com camps té la taula (i en el mateix ordre). En cas el de no inserir dades a tots els camps, caldrà especificar aquests, per identificar on s'han de col·locar les dades.

Exemple:

```
INSERT INTO Clients
     VALUES ('G43014969', 'Associació BlauCel', 25, '4301');
```

Es possible fer una inserció de dades en una taula, amb les dades recuperades directament d'una consulta.

Exemple:

```
INSERT INTO taulaProva SELECT * FROM unaAltraTaula;
```

(si no coincideixen el nombre de columnes, caldrà especificar les columnes on han d'anar a parar les dades en la taula de destinació).

3.4.- Select From

Aquesta és la instrucció bàsica de consulta de dades. S'utilitza tant per a recuperar informació d'una única taula, com per a retornar dades relacionades de vàries taules, tant generant aquesta sortida amb l'objectiu de ser visualitzat per pantalla, com per a formar part de l'entrada de dades d'un altre procés o consulta.

Sintaxi:

```
SELECT nomCamp1, nomCamp2, ....
     FROM nomTaula
     WHERE nomCampX = 'literal';
```

Exemple:

```
SELECT *
     FROM Clients
     WHERE CIF = 'G43014969';
```

A l'exemple bàsic s'observa com es fa una consulta que retornarà tots els camps, de tots els registres que compleixin la condició d'equivalència del DNI. En el cas de substituir l'asterisc per uns noms de camps concrets de la taula, només es retornarien aquests.

Exemples de consultes complexes

Una vegada vist l'exemple bàsic, s'ha de destacar que normalment es combinen diverses taules, per a tornar un resultat compost amb les dades d'aquestes.

Els exemples que es mostren a continuació, es fan considerant l'estructura de taules següents:

```
CREATE TABLE persona
      (nom varchar(20) PRIMARY KEY,
      cognoms VARCHAR(40),
      edat INT,
      dni CHAR(10));

CREATE TABLE vehicle
      (marca VARCHAR(10) PRIMARY KEY,
      model VARCHAR(15),
      anyfabricacio INT,
      propietari CHAR(10));
```

Les dades que contindran aquestes taules per a fer les proves seran:

```
INSERT INTO persona
      VALUES ('Josep', 'Hernandez Sole', 25, '39123123T');
INSERT INTO persona
      VALUES ('Joana', 'Sicó Solà', 37, '39321321F');
INSERT INTO persona
      VALUES ('David', 'Sanahuja Sevilla', 35, '37123321A');
INSERT INTO persona
      VALUES ('Jordi', 'Lluc Català', 27, '39696969Z');
INSERT INTO vehicle
      VALUES ('Seat', 'Leon 1.9TDI', 2002, '39123123T');
INSERT INTO vehicle
      VALUES ('Suzuki', 'GSX 750', 1997, '39123123T');
INSERT INTO vehicle
      VALUES ('Opel', 'Vectra DTI S', 2001, '39321321F');
INSERT INTO vehicle
      VALUES ('Audi', 'S3 Sport', 2004, '37123321A');
INSERT INTO vehicle
      VALUES ('Fiat', '500', 1972, '7123123');
```

EXEMPLE 1 (consultes amb subconsultes):

```
SELECT persona.nom,
       persona.cognoms,
       (SELECT distinct vehicle.propietari
          FROM vehicle WHERE persona.dni = vehicle.propietari)
```

```
        AS Identificador
     FROM persona;
```

S'observa que es fa una subconsulta que utilitza el resultat d'una fila d'una columna, per fer una nova cerca per a cada resultat. La subconsulta es realitza un cop recuperat el resultat de la taula principal, tot just abans de retornar la informació. En aquest cas, és obligatori que la subconsulta només retorni un resultat per a cada registre de la consulta principal.

També s'observa la utilització de la clàusula *as*, tot just desprès de la subconsulta. Amb aquesta clàusula donarem nom a una columna que és generada a partir d'una consulta, o d'un càlcul.

Pel conjunt de dades de prova, el resultat retornat és:

```
Josep      Hernandez Sole        39123123T
Joana      Sicó Solà             39321321F
David      Sanahuja Sevilla      37123321A
Jordi      Lluc Català           <null>
```

S'ha de fer notar que en no tenir en Jordi Lluc cap cotxe, falla la subconsulta, obtenint un null en la posició on hauria d'anar el DNI del propietari del vehicle.

EXEMPLE 2 (consultes amb subconsultes):

```
SELECT persona.nom, persona.cognoms
     FROM persona
     WHERE persona.dni IN
       (SELECT distinct vehicle.propietari
            FROM vehicle);
```

En aquest exemple s'observa que la SELECT principal està restringida pels resultats que es troben a la SELECT que hi ha desprès del WHERE, o sigui, de totes les possibles persones, només es retornen els DNI d'aquells que tenen algun vehicle.

Resultat de la consulta:

```
Josep      Hernandez Sole        39123123T
Joana      Sicó Solà             39321321F
David      Sanahuja Sevilla      37123321A
```

Com que en Jordi Lluc no té cap vehicle (no apareix a l'última subconsulta), no se'ns retornarà.

EXEMPLE 3 (consulta amb combinació de taules):

```
SELECT persona.nom, persona.cognoms, vehicle.marca
     FROM persona, vehicle
```

```
     WHERE persona.dni = vehicle.propietari;
```

En aquest cas, combinem les taules persona i vehicle, per a relacionar cada un d'aquests amb el seu propietari, per tant, la cardinalitat serà d'un o varis de la segona taula, per cadascun de la primera, si existeix a la de vehicles.

Resultat de la consulta:

```
David      Sanahuja Sevilla      Audi
Josep      Hernandez Sole        Suzuki
Josep      Hernandez Sole        Seat
Joana      Sicó Solà             Opel
```

S'observa que no apareix la persona que no té vehicle, i que la que en té dos, apareix per a cadascun d'aquests.

EXEMPLE 4 (consulta amb combinació de taules):

La mateixa consulta de l'exemple 3, però amb la clàusula JOIN, segons sintaxi estàndard SQL92.

```
SELECT persona.nom, persona.cognoms, vehicle.marca
     FROM persona JOIN vehicle
        ON persona.dni = vehicle.propietari;
```

(retorna el mateix resultat que l'exemple 3).

EXEMPLE 5 (consulta amb combinació de taules):

```
SELECT persona.nom, persona.cognoms, vehicle.marca
     FROM persona RIGHT JOIN vehicle
        ON persona.dni = vehicle.propietari;
```

Aquest cas es diferència dels anterior per la clàusula RIGHT del JOIN.
Com s'observa en el resultat, la combinació de taules es fa per vehicle, apareixent tots els vehicles, però no totes les persones. Juntament amb cada vehicle, apareix el seu propietari (o null si no en té).

Resultat de la consulta:

```
David      Sanahuja Sevilla      Audi
Josep      Hernandez Sole        Suzuki
Josep      Hernandez Sole        Seat
Joana      Sicó Solà             Opel
<null>     <null>                Fiat
```

EXEMPLE 6 (consulta amb combinació de taules):

```
SELECT persona.nom, persona.cognoms, vehicle.marca
     FROM persona LEFT JOIN vehicle
        ON persona.dni = vehicle.propietari;
```

És el cas contrari a l'exemple 5. Aquí s'utilitza la clàusula LEFT en el JOIN. Això fa que apareguin tots els propietaris, i si en tenen, la marca del vehicle corresponent.

Resultat de la consulta:

```
David Sanahuja Sevilla      Audi
Josep Hernandez Sole        Suzuki
Josep Hernandez Sole        Seat
Joana Sicó Solà             Opel
Jordi Lluc Català           <null>
```

EXEMPLE 7 (consulta amb combinació de taules):

```
SELECT persona.nom, persona.cognoms, vehicle.marca
     FROM persona FULL JOIN vehicle
        ON persona.dni = vehicle.propietari;
```

Utilitzant la clàusula FULL en el JOIN se'ns retorna tant els registres de la taula persones, com els de la taula vehicles. Si en fer-se la combinació, no hi ha corresponença, ens apareixeran valors null allí on faltin les dades.

Resultat de la consulta:

```
David      Sanahuja Sevilla      Audi
Josep      Hernandez Sole        Suzuki
Josep      Hernandez Sole        Seat
Joana      Sicó Solà             Opel
Jordi      Lluc Català           <null>
<null>     <null>                Fiat
```

EXEMPLE 8 (consulta amb unió de dades):

De vegades pot ser convenient que els resultats de dues consultes s'uneixin per ser retornades com si fossin un únic resultat.

Podríem voler aconseguir (amb les dades de prova) un llistat de noms i acrònims en una única columna.

```
SELECT persona.nom FROM persona
     UNION
SELECT vehicle.marca FROM vehicle;
```

Resultat de la consulta:

```
Audi
David
Fiat
Joana
Jordi
Josep
Opel
Seat
Suzuki
```

3.5.- Update

S'utilitza la sentència UPDATE per a actualitzar les dades de les columnes d'un o varis registres.

Sintaxi:

```
UPDATE nomTaula SET nomColumna = 'valor' WHERE condició;
```

Exemple:

```
UPDATE vehicle SET propietari = '37123321A'
     WHERE propietari = '7123123' AND model = '500';
```

A l'exemple estem substituint el DNI del propietari del vehicle, per un dels existents a la taula persona (tal com si fos erroni el valor del camp a vehicle).

És important observar que es poden introduir subconsultes en la sentencia.
En l'exemple següent és mostra com recuperar el valor que s'actualitzarà, amb una nova consulta.

```
UPDATE vehicle SET propietari =
     (SELECT DISTINCT dni
          FROM persona
          WHERE cognoms ILIKE '%lluc catal%'
                AND edat BETWEEN 20 AND 30)
     WHERE marca = 'Fiat';
```

En aquest cas, es vol canviar el DNI del propietari de l'únic vehicle que hi ha de la marca Fiat, pel d'una persona que sabem que es diu Lluc Catal (s'ha utilitzat el ILIKE per ometre la distinció entre majúscules i minúscules), i que té entre 20 i 30 anys.

3.6.- Delete

S'utilitza la sentencia DELETE per a esborrar un o varis registres d'una taula.

Sintaxi:

```
DELETE FROM nomTaula WHERE condicio;
```

Exemple:

```
DELETE FROM vehicle WHERE propietari  = '37123321A';
```

S'ha de fer notar que d'utilitzar aquesta sentencia sense la condició, s'esborraria tot el contingut de la taula !

D'haver-se d'esborrar completament una taula amb molts registres, es pot utilitzar la instrucció TRUNCATE (que és especifica del PostgreSQL), ja que està optimitzada especialment per aquests casos.

Exemple:

```
TRUNCATE vehicle;
```

3.7.- Sintaxi addicional de manipulació dels resultats

Desprès de la condició limitadora WHERE de les SELECT, es pot afegir les clàusules:

ORDER BY
S'utilitzarà el ORDER BY nomCamp1, nomCamp2, etc per a que se'ns mostri els resultats de la SELECT ordenats segons el camp, o camps especificats.

GROUP BY
D'utilitzar la sentencia d'agrupació GROUP BY nomCamp1, nomCamp2, etc, s'agruparan les files que tinguin el nomCamp1 (i següents) igual, per a ser mostrades només com una de sola. És imprescindible que només apareguin com a camps a mostrar, els que hi han en aquesta clàusula, ja que no és pot mostrar camps agrupats amb atributs diferents.

LIKE
En el cas de voler fer una cerca pel contingut d'un camp, però no estar segur de que el literal de cerca sigui el contingut total d'aquest, es pot utilitzar el operador LIKE per a cercar els registres que en el camp especificat continguin la cadena literal. El signe 'tant per cent' és el 'comodí'. En aquest cas, s'especificaria que es cerqués una paraula concreta, malgrat tingués altres caràcters al davant i al darrera.

Així si féssim (segons les dades d'exemple utilitzades a l'apartat de les SELECT):

```
SELECT * FROM persona WHERE dni LIKE '%123%';
```

Se'ns retornaria com a resultat de la consulta:

```
Josep      Hernandez Sole        25    39123123T
David      Sanahuja Sevilla      35    37123321A
```

En el cas de voler que la cerca no tingui en compte les majúscules / minúscules, es pot utilitzar el ILIKE (instrucció pròpia del PostgreSQL):

```
SELECT * FROM vehicle WHERE propietari ILIKE '%123t%';
```

En el resultat s'observa que s'han retornar dos registres, malgrat la lletra del seu NIF estigui en majúscules.

Resultat de la consulta:

```
Seat  Leon 1.9   TDI   2002  39123123T
Suzuki     GSX   750   1997  39123123T
```

Expressions Regulars

En el PostgreSQL es poden utilitzar expressions regulars en les condicions d'acotament (no és una característica estàndard SQL).

Aquests tipus d'expressions són prou potents (i complexes) com per a requerir tot un ampli manual d'ús. L'exemple senzill i il·lustratiu consistirà en fer una cerca dels usuaris que tinguin com a segon caràcter del seu nom la lletra 'o':

```
SELECT * FROM persona WHERE nom ~'^.o'
```

Resultat de la consulta:

```
Josep      Hernandez Sole  25   39123123T
Joana      Sicó Solà       37   39321321F
Jordi      Lluc Català     27   39696969Z
```

EXPRESSIONS REGULARS

Es construeixen seguint el següent patró:

Inici:

~ indicació d'expressió regular
~* indicació d'expressió regular no sensible a les majúscules/minúscules
!~ indicació de negació d'expressió regular
!~* indicació de negació d'expressió regular no sensible a majúscules/minúscules

Condicions:

^ inici
$ final
. qualsevol caràcter
[ccc] conjunt de caràcters
[^ccc] diferent al conjunt de caràcters
[a-z] rang de caràcters
[^a-z] diferent al rang de caràcters
? zero o un caràcter
* zero o varis caràcters
+ un o varis caràcters
| operador OR

SIMILAR TO

L'operador SIMILAR TO avalua una expressió a cert o fals, depenent de si la cadena avaluada compleix amb el patró indicat.

El funcionament de l'operador és similar al LIKE, però en aquest cas el patró de cerca és interpretat com una expressió regular.

Exemple 1:

```
SELECT * FROM persona WHERE dni SIMILAR TO '%(3_1+)%'
```

Resultat de la consulta:

```
Josep      Hernandez Sole       25   39123123T
Joana      Sicó Solà            37   39321321F
David      Sanahuja Sevilla     35   37123321A
```

D'interessar únicament els registres que comencin per 3 i que tinguin en la tercera posició el 1, s'hauria d'eliminar el primer símbol % a l'expressió de cerca.

4.- El llenguatge pl/pgSQL

En el PostgreSQL es poden utilitzar diversos llenguatges per a escriure les funcions que s'executaran com a procediments emmagatzemats. Entre ells hi ha el java, el C i el pl/pgSQL.
En el cas d'utilitzar el C, podrien sorgir problemes de portabilitat de canviar el maquinari que executa l'SGBD (si es canvia de 32 a 64 bits, s'hauria de revisar tot el tractament dels numèrics). En el cas d'elegir el java com a llenguatge procedimental per a executar les rutines a la base de dades, tindríem un ampli suport, però probablement el servidor patiria de problemes de rendiment. L'opció recomanada és la d'utilitzar el llenguatge nadiu de l'SGBD, el pl/pgSQL.

El pl/pgSQL no és més que l'ampliació del conjunt d'instruccions de l'SQL habitual, per a poder controlar el flux dels programes, poder utilitzar estructures de dades, variables, i a la fi, aconseguir totes aquelles altres funcionalitats pròpies dels llenguatges tradicionals.

Aquest apartat es presenta com la continuació de la lectura de la guia de pg/plSQL inclosa en el manual del PostgreSQL, donat que s'ha considerat que els exemples són molt bàsics, i els addicionals trobats a internet no aportaven més informació que la que apareixia a la web oficial de PostgreSQL, o al manual 'oficial' (habitualment amb lleugers 'retocs'). En alguns casos es plantegen les funcionalitats de forma comparativa amb les del'Informix-4GL.

4.1.- Els Procediments.

Els procediments emmagatzemats són anomenats realment funcions en el PostgreSQL. Al crear-los, així s'haurà d'especificar a la sintaxis. Aquests poden tenir tant paràmetres d'entrada com de sortida. Els paràmetres poden ser de qualsevol tipus permès, i poden ser un o varis.

El cos del codi de l'interior del procediment pot anar entre comentes senzilles (entre les clàusules *AS* i *LANGUAGE*), o entre els símbols $$. En el primer cas, cada vegada que aparegui una cometa a l'interior d'aquest codi, s'haurà de doblar.

Exemple:

```
CREATE OR REPLACE FUNCTION afegeixAmic (nomEntrada VARCHAR(20))
   RETURNS VARCHAR(32) AS '
BEGIN
  RETURN ''Hola amic ''||$1;
  END;
' LANGUAGE 'plpgsql';
```

S'observa que allí on s'han inserit cometes (línia del RETURN), s'han doblat.
També s'observa que per a fer la concatenació de dues cadenes de text, s'han d'utilitzar les barres verticals.

Per a executar un procediment s'utilitza el *select*. En el cas de l'exemple següent passem un valor al procediment, i esperem que es retorni un resultat:

```
SELECT * FROM afegeixAmic('Pere');
```

D'haver de recollir el valor en una variable (des del interior d'un procediment):

```
temps := (SELECT FROM TempsTrajecte(horaentrada, horasortida));
```

En lloc de fer l'assignació es podria utilitzar la clàusula *into* per a recollir el resultat.

En el cas en que no es vulgui recollir el resultat que retorna la *select* s'haurà d'utilitzar el PERFORM.

Exemple:

```
PERFORM ActualitzaTemperatures('Tarragona', 34);
```

(en el cas de que per la lògica de negoci, l'actualització de la temperatura sigui un procés complex que s'ha d'executar des d'un procediment).

De forma similar a altes llenguatges, el signe 'igual' es reserva per a les comparacions, mentre que per a les assignacions s'ha d'utilitzar els 'dos punt igual'.

Exemple:

```
CREATE FUNCTION ....
DECLARE
  var1 INT;
BEGIN
  var1 := 99;
  IF (var1 = varEntrada) THEN ...
...
```

A diferencia d'altres SGBD a PostgreSQL no s'inclou codi en els disparadors. Aquests han de cridar immediatament a un procediment emmagatzemat.
Aquest fet es deu a que al poder utilitzar diferents llenguatges per a escriure la codificació dels procediments, es fa una certa normalització en determinar que 'a partir' del disparador, el codi que s'executa ja pot ser en el llenguatge que s'indica en el mateix procediment (Informix si permet incloure les rutines SQL en els mateixos disparadors).

En cridar-se un procediment emmagatzemat des d'un disparador, els valors OLD i NEW passen per defecte en indicar-se a la capçalera del procediment el tipus d'entrada de dades (no cal, com en l'Informix, passar els valors que necessitem al procediment, explícitament des del trigger).

Exemple (considerant que el procediment és cridat des d'un disparador):

```
CREATE OR REPLACE FUNCTION comprovarSou () RETURNS TRIGGER AS '
  BEGIN
  IF (OLD.Sou > NEW.Sou) THEN
     RAISE EXCEPTION ''ERROR - El sou únicament es
       pot incrementar!'';
  END IF;
  RETURN NEW;
END;
' LANGUAGE 'plpgsql';
```

Pel mateix funcionament del pas de paràmetres entre els disparadors i els procediments (s'ha d'indicar explícitament que el tipus d'entrada d'un procediment és 'trigger'), no és possible que un mateix procediment pugui ser cridat des d'un disparador i des d'un altra procediment. De ser necessari això, s'haurà d'escriure dos procediments diferents. En el que és cridat des d'un altra procediment, haurà d'indicar els valors d'entrada que espera.

Per a millorar la fiabilitat del codi, i suportar millor els possibles canvis de tipus, o de llargada dels camps de les taules, es pot especificar amb la clàusula *%TYPE* que una variable determinada sigui del tipus d'un camp determinat.

També s'ha d'observar que els valors que es reben d'entrada als procediments, tindran com a nom el nombre d'ordre en que s'han rebut (segons la capçalera), amb un signe dòlar anteposat.

Exemple:

```
CREATE FUNCTION p_trajecte (trajecte.data_e%TYPE,
    trajecte.data_s%TYPE) RETURNS INT AS '
BEGIN
  IF ($1 = $2) THEN .....
     .......
     .......
  END;
' LANGUAGE 'plpgsql';
```

S'observa que es reben dos paràmetres (desprès s'utilitzen amb el nom de $1 i de $2). Aquests seran del mateix tipus que el que té els camps data_e i data_s de la taula trajecte (res hagués impedit posar directament a la capçalera (date, date) en lloc d'utilitzar el %type).

El tipus de dades més complex que pot retornar una funció, és el de varies files de tipus registre (varies columnes, no coincidents en tipus amb els d'una taula). En aquest cas s'haurà de declarar que retornarem un *SETOF RECORD*.

En cridar-se una funció que retorna un *SETOF RECORD* s'haurà d'especificar la seva estructura, o sigui, els camps i els tipus d'aquests amb la clàusula AS. El noms dels camps que s'especifiquin, seran els dels camps que se'ns retornaran.

Exemple complert:

```
CREATE TABLE ciutats (ciutat VARCHAR(20), prov VARCHAR(20),
   temperatura INT);

INSERT INTO ciutats VALUES ('Tarragona', 'Tarragona', 15);
INSERT INTO ciutats VALUES ('Valls', 'Tarragona', 17);
INSERT INTO ciutats VALUES ('Reus', 'Tarragona', 20);
INSERT INTO ciutats VALUES ('Barcelona', 'Barcelona', 30);
INSERT INTO ciutats VALUES ('Mollet', 'Barcelona', 31);

CREATE OR REPLACE FUNCTION tempCiutatsProv(ciutats.prov%TYPE)
   RETURNS SETOF RECORD AS '
DECLARE
   actual RECORD;
BEGIN
   FOR actual IN
       SELECT ciutat, temperatura
           FROM ciutats
           WHERE prov = $1
   LOOP
       RETURN NEXT actual;
   END LOOP;
RETURN;
END;
' LANGUAGE 'plpgsql';
```

S'observa que a l'interior del procediment s'executa el codi que hi ha entre el LOOP i el END LOOP (aquest exemple és bàsic) per a cada registre que compleix el criteri de la *select*.

En fer la consulta s'haurà d'especificar quina és l'estructura del RECORD que es retornarà (amb la clàusula AS):

```
SELECT * FROM tempCiutatsProv('Tarragona')
     AS (ciutat VARCHAR(20), graus INT);
```

El resultat retornat serà:

ciutat	graus
Reus	20
Tarragona	15
Valls	17

En el cas de que els valors a retornar coincideixin amb els dels camps d'una taula, es pot indicar que els valors de sortida del procediment són: *SETOF* nomTaula.

Modificant l'exemple anterior:

```
CREATE OR REPLACE FUNCTION tempCiutatsProv2(ciutats.prov%TYPE)
   RETURNS SETOF ciutats AS '
DECLARE
  actual ciutats%ROWTYPE;
BEGIN
  FOR actual IN
    SELECT ciutat, prov, temperatura
      FROM ciutats
      WHERE prov = $1
  LOOP
      RETURN NEXT actual;
  END LOOP;
RETURN;
END;
' LANGUAGE 'plpgsql';
```

En fer la consulta:

```
SELECT * FROM tempCiutatsProv2('Barcelona');
```

Es retornarien els següents resultats:

ciutat	prov	temperatura
Barcelona	Barcelona	30
Mollet	Barcelona	31

4.2.- Els Cursors.

Es pot aprofitar el fet que un procediment que retorna varis registres pot ser considerat com una taula, per a simplificar el codi dels programes. Es poden utilitzar els valors resultants com una taula al crear cursors i fer altres operacions.

En l'exemple següent crearem un cursor en un procediment. D'haver-se d'executar aquest des d'un altra rutina pl/pgSQL es podria implementar l'obertura d'aquest des de la mateixa, però tal com es fa, aquest pot ser cridat fins i tot des d'un programa extern (en java mitjançant JDBC, per exemple) o des d'una altra procediment fet en un altra llenguatge:

```
CREATE OR REPLACE FUNCTION ex_cursor (refcursor,
    ciutats.provincia%TYPE)
  RETURNS refcursor AS '
BEGIN
OPEN $1 FOR SELECT * FROM tempCiutatsProv2($2)
    AS (ciutat VARCHAR(20), graus INT);
RETURN $1;
END;
' LANGUAGE plpgsql;
```

Tal com se'ns indiqui que s'ha creat correctament, podem obrir el cursor:

```
BEGIN;
SELECT ex_cursor('cursortest', 'Tarragona');
```

La primera vegada se'ns retornarà un valor conforme s'ha obert correctament.

En demanar els valors d'un en un amb les instruccions (l'haurem d'executar diverses vegades);

```
FETCH NEXT IN cursortest;
```

se'ns retornarà una ciutat i una temperatura cada vegada.

Acabarem tancant el cursor amb la instrucció:

```
COMMIT;
```

5.- Índexs.

Els índexs permeten optimitzar l'accés a les dades d'una taula, per una o vàries de les columnes que té. Aquests consisteixen bàsicament en una estructura de dades separada de la taula que indexa, fàcilment recorrible (amb forma d'arbre habitualment), on cercar un valor sol costar log(n), i que relaciona el valor concret que se cerca, amb la seva posició dintre de l'estructura on s'emmagatzemen les dades, per a poder accedir a aquesta directament.

Amb la instal·lació per defecte del PostgreSQL, s'incorporen tres tipus diferents d'índex; els B-Tree, els R-Tree i els Hash.

Habitualment, i si no s'indica el contrari, PostgreSQL utilitzarà el index B-Tree. No obstant s'ha de considerar que cada índex té unes peculiaritats que el fan més òptim en uns casos que en d'altres, pel que s'ha de revisar els criteris d'utilització, per a assegurar l'ús adequat en cada cas.

També és important considerar que no tots els tipus d'índex tenen ni el mateix rendiment, ni el mateix cost d'execució, ni les mateixes possibilitats de concurrència. Aquest últim punt és especialment crític en entorns en que hi ha un alt nombre d'insercions i modificacions, ja que tal com es veurà, la inserció de noves branques o fulles per a mantenir l'arbre on s'emmagatzema l'índex ben equilibrat, pot fer que de no tenir-se en compte, altres processos en curs puguin fallar per trobar-se registres blocats (en els accessos a les estructures de dades que contenen els índexs encara no s'utilitza el tipus d'accés Multi Version Concurrent Control).

5.1.- B-Tree

És un índex d'alta concurrència, basat en els de Lehman Yao. Aquest tipus d'índex és totalment dinàmic, ja que no requereix ni de manteniments ni d'optimitzacions periòdiques.

PostgreSQL utilitza aquest tipus d'índex per defecte, ja que en casos genèrics és el que pot donar uns millors temps d'accés i a que és l'únic que té suport per creació de índexs per més d'una columna (en un PostgreSQL instal·lat de forma estàndard, es poden utilitzar índexs com a màxim de 16 columnes).

Es pot utilitzar en fer cerques amb condicions de menor, major o d'equivalència a uns valors determinats.

5.2.- R-Tree

Aquests índex són especialment eficients quan s'utilitzen per a cercar formes geomètriques o punts a l'espai. Estan basats en els treball de divisions quadràtiques d'en Atonin Buttman.

Al igual que el B-Tree, aquests tipus d'índex no requereixen ni de manteniment, ni d'optimitzacions periòdiques.

5.3.- Hash

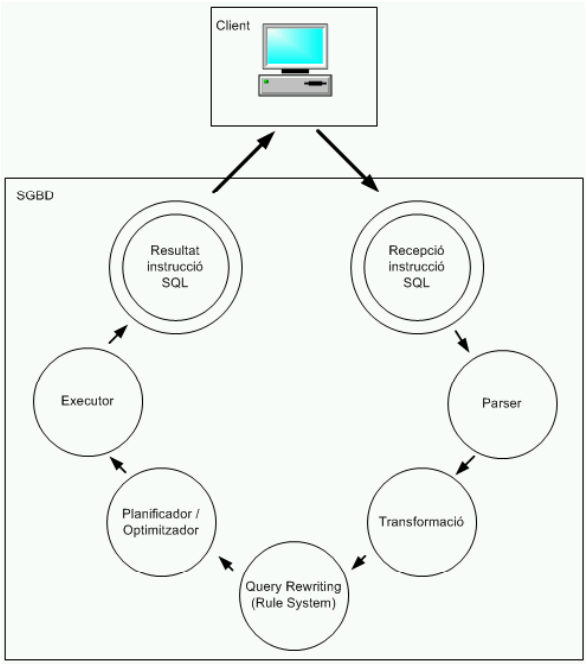
Aquests índexs són una implementació de l'algorisme lineals Hash d'en Litwin, i no necessiten ni d'optimitzacions ni de manteniments periòdics (veure últim punt d'aquest apartat).

Aquest índex només es pot utilitzar quan es cerca un valor concret, ja que s'aplica una funció sobre el valor del camp a buscar, per a trobar la seva posició en l'estructura de dades on s'emmagatzema. És important que la funció hash faci que la distribució de valors sigui uniforme i única, ja que si a un resultat corresponen varis valors, s'hauran de fer recorreguts de cost lineal.

La implementació que es fa dels índexs linear-hash al PostgreSQL (al menys fins a la versió 8.0) fa que aquests tinguin un rendiment baix (pitjor en accés a les dades que els b-tree), que siguin lents de creació i regeneració, i que pel sistema utilitzat en l'enregistrament dels canvis, en cas de fallada del sistema puguin quedar inconsistents (havent-se de regenerar). Per tots els motius anteriors, la comunitat de desenvolupadors recomana només utilitzar aquest tipus d'índex per a fins acadèmics (textual), i argumenta que no s'han posat en estat 'deprecated', donat que en qualsevol moment, qualsevol desenvolupador pot recollir el testimoni per a millorar el seu funcionament.

6.- Pla d'execució de consultes

Una consulta pot tardar més o menys temps en executar-se, depenent del disseny de les taules, les seves interrelacions, la quantitat de dades, de la planificació en l'ordre d'execució de les operacions que cal fer per a extreure les dades concretes de les taules on s'emmagatzemen, i d'altres paràmetres addicionals.



El client genera una consulta SQL i:

- L'SGBD la recull.
- Comprova si sintàcticament és correcte.
- La transforma en un arbre d'execució.
- Aplica la reescriptura si és necessari.
- Planifica les possibles formes d'execució, i tria la de menor cost.
- Executa cadascun dels nodes de l'arbre en que s'ha descompost la petició SQL.
- Es transmet el resultat al client.

Per a verificar que una consulta s'executa de forma òptima és necessari comprovar els passos que indica el planificador que es faran per a resoldre la tasca.

Fent un EXPLAIN ANALYZE *consulta*, obtindrem una explicació esquemàtica, amb costos, de la consulta concreta. La interpretació d'aquesta execució s'ha de fer de les fulles (part inferior) a l'arrel, ja que les operacions dels nivells superior, utilitzen els resultats obtinguts pels processaments de les parts inferiors.

El pla d'execució retornat, no té perquè haver estat l'únic que s'ha considerat, ja que una consulta pot tenir més d'un forma vàlida d'executar-se. En aquests casos, el planificador tria la de menor cost (segons les estimacions).

En determinats casos, especialment si la consulta a analitzar inclou més de N taules relacionades, l'estimació de quina és la millor possibilitat d'execució pot tenir un cost d'execució tant alt, que és preferible utilitzar el mètode Genetic Query Optimitzer (GEQO), per a obtenir el pla d'execució millor 'estimat'. Amb aquest sistema (basat en algorismes genètics), no s'assegura que s'utilitzarà el millor mètode de tots, però si que el que s'utilitzarà serà una bona aproximació a aquest, ja que durant l'estimació de les possibilitats, es fa evolucionar les alternatives a considerar, obtenint millors resultats que amb un anàlisi aleatori.

D'haver-se d'executar múltiples vegades una mateixa consulta, es pot obviar el temps que utilitza el planificador per a obtenir la millor forma d'execució, d'utilitzar-se una consulta preparada (segons es detalla a l'apartat d'optimitzacions - capítol 6).

És important adonar-se'n que les estimacions de costs de les consultes que fa l'SGBD parteixen de les estadístiques que té sobre el contingut de les taules (nombre de registres, quantitat de pàgines ocupades, etc), i que aquestes estadístiques únicament s'actualitzen en executar-se la instrucció *analyze*. (instrucció pròpia del PostgreSQL, equivalent a *update statistics* de l'SQL'92). Cal remarcar que la base de dades no actualitza aquestes estadístiques automàticament, pel que és imprescindible, per tal de mantenir un bon rendiment, l'execució de les mateixes de forma periòdica. En cas contrari, l'increment del nombre de registres d'una taula (per exemple), podria fer que el planificador de l'SGBD decidís fer un recorregut (de cost lineal) sobre aquesta, en lloc de fer-ho utilitzant un índex disponible.

Els 19 operadors que poden aparèixer durant el processament d'una consulta, són:

6.1.- Seq Scan:

És el operador més bàsic. Consisteix en fer una lectura seqüencial dels registres d'una taula. Així per a cada fila llegida s'avaluarà la condició de cerca, tot afegint aquesta al conjunt de sortida, si la condició es compleix.

Si l'expressió de cerca que s'utilitza, és d'equivalència sobre un registre que no pot tenir repeticions (UNIQUE), es pot fer acabar el recorregut desprès de trobar la primera correspondència.

L'operació és de cost lineal, i serà la utilitzada de no existir cap índex a la taula que pugui ser utilitzat per a accedir segons la consulta feta, o en el cas de que s'estimi que té menys cost que les altres opcions.

En taules amb pocs registres pot ser més òptim aquest accés, que un d'alternatiu per índex, ja que tota la informació estarà ubicada en un sol bloc, i es realitzarà en una única E/S de disc.

S'ha de considerar que en utilitzar maquinari modern, el cost de llargues lectures seqüencials pot ser més baix que el de múltiples lectures distribuïdes per tot el disc (pel temps emprat pels desplaçaments dels capçals d'aquest), pel que s'hauria de considerar forçar aquest mètode en taules de no molts registres (fins a un centenar), sobretot si tenen mida petita.

En el cas que la consulta no impliqui altres operacions que facin manipular altres taules, o facin una ordenació dels resultats posteriorment (per ex.), la primera fila serà tornada d'immediat (mentre la resta de la consulta s'acaba d'executar), pel que en molts casos (en utilitzar-se en un cursor), d'haver-se de tractar tota una taula, serà l'opció més ràpida.

6.2.- Index Scan:

Aquest tipus d'accés s'utilitza en el cas d'haver d'accedir a un conjunt de registres, i poder utilitzar un índex per a accedir al primer d'ells.

Si la consulta especifica un rang de valors, el primer accés té cost log(n) (si índex tipus B-Tree per ex.), i la resta lineal entre la selecció.

A diferència de Seq Scan, que retorna els valors segons l'ordre en que es troben a la taula, Index Scan en utilitzar un índex, retorna aquests ordenats pel mateix.

Aquesta tècnica només s'utilitzarà si el tipus d'índex utilitzat és B-Tree, R-Tree o GiST (Generalized Search Tree), però no HASH, ja que amb aquest últim només és poden localitzar registres individuals, però no es possible fer un recorregut, ja que donat el propi concepte d'aquest índex, els registres no tenen vincles entre ells.

6.3.- Sort

El operant *sort* s'utilitza (tal com el seu nom indica), per a ordenar un conjunt de registres, per la columna o columnes indicades.

Hi han diferents mètodes d'ordenació, i tant es pot fer l'operació en la memòria principal com sobre disc (la decisió de fer-ho utilitzant un o l'altre suport es pren automàticament segons el volum de dades).

És important tenir en compte que el conjunt resultant de l'operació *sort* només es retorna quan s'ha executat tot l'algorisme d'ordenació, pel que s'hauria d'intentar ajornar l'esmentada operació (en el procés de planificació de la consulta) fins a tenir un conjunt de dades mínim.

La operació d'ordenació no només s'utilitza quan s'especifica que es vol el resultat ordenat, ja que per a realitzar altres operacions internes (tal com interseccions i unions), també cal que s'executi.

6.4.- Unique

Amb aquest operador s'eliminen valors duplicats d'un conjunt de dades. Es pot utilitzar, per exemple, en el cas de voler retornar un únic registre de cada categoria, malgrat apareguin diverses dades de cadascuna d'aquesta. Així, sempre s'executarà en especificar la clàusula DISTINCT en una consulta, o en eliminar duplicats per a fer una UNION.

El cost serà lineal, pel conjunt de registres implicats en l'operació.

6.5.- Limit

Aquest és l'operador que s'utilitza per a obtenir un conjunt de resultats finit, segons el valor especificat. Així, s'utilitzarà aquesta funció quan vulguem que només se'ns retorni els primers *n* registres d'una consulta.

La poda que fa aquest operador s'executa en una de les últimes etapes de la consulta, ja que de demanar els primers *n* registres d'una operació ordenada, en el cas de fer-ho en una primera etapa, no obtindríem els primers resultats del producte ordenat, tal com s'esperava. En el cas en el que l'ordre no sigui important, es retornaran els primers resultats, sense que sigui necessari l'execució de tota la consulta.

El cost d'execució que afegeix aquesta clàusula a una consulta és negligible, ja que aquesta s'ha d'executar normalment, per passar a eliminar desprès els resultats sobrers en fer la sortida.

6.6.- Aggregate

L'operador *aggregate* s'utilitza en el cas de que la consulta tingui operadors que retornin resultats calculats, a partir dels valors de les columnes.

Així, en el cas de que s'hagi de calcular la mitjana aritmètica d'un conjunt de valors, desprès de fer l'agrupació d'aquests segons els criteris especificats, s'executarà el *aggregate* per a dur l'operació d'obtenció de la mitjana.

6.7.- Append

Aquest operador fa una agregació dels resultats de dues o més consultes, per a que el resultat d'aquestes apareguin com un d'únic.

El cost total de l'operació serà el de la suma de les suboperacions que s'executen i normalment s'utilitzarà en realitzar una UNION entre els resultats de vàries consultes.

És imprescindible que el resultat que retornen les consultes siguin del mateix tipus, i tinguin el mateix nombre de columnes.

6.8.- Result

Aquest tipus d'operació s'anotarà com a execució, quan la consulta no tingui que retornar dades de cap taula, per estar-se executant una acció que retorna una dada de l'entorn o generada (la data, per exemple), o en el cas en que a la clàusula WHERE s'utilitzi com a condició boolena, per a retornar, o no, un resultat.

6.9.- Nested Loop

És l'operació que s'utilitza en combinar dues taules, partint d'una d'elles.

A partir de la recuperació de cada registre de la taula principal, se cerca (mitjançant un índex, si existeix), el registre corresponent (segons la condició establerta), a l'altra taula.

La cardinalitat dependrà del nombre de registres que existeixin a la segona taula, per a cadascun dels que apareixen a la primera, però en cap cas, apareixeran en el resultat els registres que apareguin a la segona, sense correspondència amb la primera. Així, aquesta operació s'utilitzarà en el cas dels LEFT JOIN, però no en el cas dels RIGHT JOIN o dels FULL JOIN.

6.10.- Merge Join

S'utilitza quan s'ha de fer una JOIN de dues o més taules. El funcionament consisteix en recuperar tots els registres implicats en la condició, segons les claus especificades, per passar a fer l'operació d'extracció combinada dels registres d'aquesta relació, malgrat algun d'ells només existeixi en una de les taules.

La operació de Merge Join normalment es realitzarà desprès de que s'hagi fet un Index Scan o un Seq Scan de les dues taules i un Sort dels resultats previs a la consulta que retornarà una cardinalitat en la que existiran els registres coincidents, i els no coincidents, tant de la primera com de la segona taula.

6.11.- Hash Join

Aquest operador s'utilitza en fer una INNER JOIN, un LEFT OUTER JOIN o un UNION, i no requereix que les taules estiguin ordenades, ni que tinguin cap índex creat amb anterioritat. En utilitzar-se el Hash Join, en primer lloc es crea una taula temporal,

creant un índex hash per la columna que s'utilitza per fer la combinació amb l'altra taula. Un cop fet això, es pot fer una lectura dels registres de la segona taula, per accedir directament pel índex hash creat a la primera, per trobar els registres corresponents.

6.12.- Group

El operador d'agrupació s'utilitza exclusivament quan s'especifica la clàusula GROUP BY a la condició de la consulta. Per a poder portar-se a terme, cal que els registres que intervenen en aquesta operació estiguin ordenats (pot fer augmentar considerablement el temps de procés).

6.13.- Subquery Scan i Subplan

Aquestes dues operacions es poden obviar al veure-les en l'arbre d'execució d'una consulta, ja que no afegeixen cost a la mateixa, i són 'de funcionament intern'.

La de Subquery Scan s'utilitza per a obtenir els resultats de dues consultes que retornen el resultat conjuntament, donat un UNION, mentre que el Subplan s'utilitza en el cas de haver d'executar una subconsulta.

6.14.- Tid Scan

És una operació que no sol ser utilitzada, ja que s'ha de forçar una consulta específica per a que aparegui.

El Tid (tuple idenficator), és un nombre que l'SGBD emmagatzema en cada registre. Consta del numero de bloc on està emmagatzemat el registre, i la posició del registre concret en aquest.

En aquesta operació d'accés, de cost 1, s'utilitza l'identificador únic ctid de cada registre, per a recuperar de forma ràpida i unívoca un valor.

Es poden aconseguir optimitzacions en l'accés a les dades de guardar el valor ctid, en el cas d'haver d'accedir a un valor d'un registre per a operar amb aquest, i posteriorment haver-lo d'actualitzar en el registre original. Serà molt més ràpid accedir pel ctid, que pel índex!

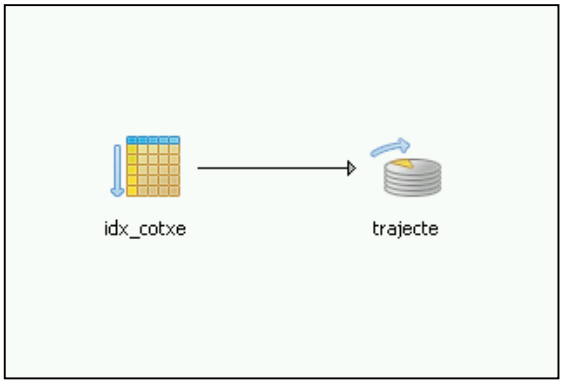
Exemple de cas d'ús del Tid Scan:

```
select * from trajecte
     where ctid = (select ctid from trajecte
                     where cotxe = 'B-0000-TF'
                     and horaentrada = '21/08/2003 03:34:59');
```

Al fer l'explain plan se'ns indicarà:

```
Tid Scan on trajecte (cost=3.89..7.91 rows=1 width=77)
  Filter: (ctid = $0)
  InitPlan
   -> Index Scan using idx_cotxe on trajecte (cost=0.00..3.89 rows=1 width=6)
      Index Cond: ((cotxe)::text = 'B-0000-TF'::text)
      Filter: ((horaentrada)::text = '21/08/2003 03:34:59'::text)
```

La representació gràfica de l'accés mitjançant el ctid és l'adjunta:



6.15.- Materialize

Tal com el seu nom indica, aquesta operació 'materialitza' un conjunt de dades en una taula temporal. Això es duu a terme quan l'optimitzador calcula que aquest procés serà més ràpid que no pas haver de fer repetides consultes d'alt cost al llarg d'una operació (en molts casos en fer Merge Joins).

6.16.- Setop Intersect, Setop Intersect All, Setop Except, Setop Except All

Són operacions que només s'executaran en els casos d'utilitzar la instrucció Intersect o Except en les seves diverses variants. L'operació s'executaria en últim terme, ja que primer s'executarien els Seq Scan o accés per índex necessaris per a resoldre les consultes independents, per passar desprès a fer-se l'*append*, tot acabant (si no hi ha ordenacions o altres operacions de format de sortida) amb el Setop corresponent.

7.- Optimització física

7.1.- Consultes preparades

Per a executar una consulta repetitiva de la forma més eficient possible, hi ha l'opció d'evitar la fase d'anàlisi i d'optimització de la consulta per part del planificador. Això només serà d'utilitat si la mida de les taules, les relacions entre aquestes i els índex (entre d'altres) es mantenen iguals entre les diferents execucions, ja que de canviar aquests paràmetres de forma substancial, la planificació inicial de la consulta podria ja no ser òptima.

Per fer una consulta preparada, caldrà emmagatzemar-la amb un nom. Desprès només caldrà fer la crida corresponent per aconseguir la seva execució immediata.

Exemple:
Tenim una taula clients, i volem tenir una consulta preparada per que ens retorni els registres que tinguin un nom determinat.

Crearem la consulta preparada:

```
PREPARE CPrep_Clients (varchar(20)) AS
    SELECT * FROM client WHERE nom = $1;
```

I desprès la podrem executar:

```
EXECUTE CPrep_Clients('Francio');
```

Al fer l'EXPLAIN ANALYZE de l'execució d'aquesta consulta preparada, o de la sentencia equivalent:

```
SELECT * FROM client WHERE nom = 'Francio';
```

comprovarem que indica que mentre la primera triga 1482.000 ms, la segona triga 1533.000 ms. Restant un temps de l'altra, obtenim que s'estalvien els 51ms que triga la fase d'anàlisis en decidir com s'ha d'executar!

```
PREPARE CPrep_Clients (varchar(20)) AS
    select * from client where nom = $1;

explain analyze EXECUTE CPrep_Clients('Francio');
```

select * from c | select * from c | explain analyze | EXECUTE CPrep_C | explain analyze | explain analyze

explain analyze EXECUTE CPrep_Clients('Francio')

Results | MetaData | Info

QUERY PLAN
Seq Scan on client (cost=0.00..984.77 rows=6 width=115) (actual time=0.000..1472.000 rows=6 loops=1)
Filter: ((nom)::text = ($1)::text)
Total runtime: 1482.000 ms

1 - PostgreSQL - UOC (template1) as postgres (2)

explain analyze select * from client where nom = 'Francio'

```
explain analyze select * from client where nom = 'Francio';
```

explain analyze

explain analyze select * from client where nom = 'Francio'

Results | MetaData | Info

QUERY PLAN
Seq Scan on client (cost=0.00..984.77 rows=6 width=115) (actual time=0.000..1552.000 rows=6 loops=1)
Filter: ((nom)::text = 'Francio'::text)
Total runtime: 1552.000 ms

7.2.- Creació de cluster

En utilitzar un índex per fer l'accés a una taula, ens trobarem que aquest està ordenat segons el camp pel que s'ha creat, però que en fer l'accés als registres de la taula corresponent, aquests no estaran ubicats consecutivament en les pàgines de la BD, pel que s'hauran de fer múltiples accessos al disc.

Cotxe
B-0000-TF
B-0001-OS
B-0001-OS
B-0001-RL
B-0001-SY
B-0001-VK
B-0002-OB
B-0002-WM
B-0003-TU
B-0003-WB
B-0004-PT
B-0004-RW
B-0004-SD
B-0004-UH
B-0004-WJ

Cotxe	P_Entrada	HoraEntrada	P_Sortida	HoraSortida
B-0001-OS	AP7-7	14-4-03 3:34 AM	AP7-12	14-4-03 3:50 AM
B-0000-TF	AP2-4	11-7-02 10:11 AM	AP2-8	11-7-02 11:11 AM
B-0004-RW	AP7-12	5-11-02 12:22 PM	AP7-25	5-11-02 12:52 PM
B-0004-UH	AP2-11	25-11-03 7:06 AM	AP7-28	25-11-03 7:26 AM
B-0002-OB	AP7-12	6-12-04 6:37 AM	AP2-6	6-12-04 7:27 AM
B-0003-WB	AP2-7	2-12-03 7:31 AM	AP7-10	2-12-03 8:11 AM
B-0001-SY	AP7-32	28-4-01 3:36 AM	AP7-27	28-4-01 4:01 AM
B-0004-SD	AP7-2	21-3-01 3:32 AM	AP7-9	21-3-01 3:52 AM
B-0004-PT	AP7-26	18-3-03 6:45 AM	AP2-11	18-3-03 7:15 AM
B-0001-VK	AP7-31	17-3-01 10:51 AM	AP2-9	17-3-01 11:01 AM
B-0004-WJ	AP2-9	13-10-03 6:43 AM	AP7-26	13-10-03 7:23 AM
B-0001-OS	AP7-26	14-2-04 6:46 AM	AP7-2	14-2-04 7:06 AM
B-0002-WM	AP2-6	2-7-04 3:08 AM	AP7-4	2-7-04 3:28 AM
B-0001-RL	AP7-29	14-5-04 3:28 AM	AP2-2	14-5-04 3:48 AM
B-0003-TU	AP7-31	17-5-04 1:27 AM	AP2-3	17-5-04 1:57 AM

S'observa que les dades estan emmagatzemades en 5 pàgines físiques de la BD, i que si només es pot tenir una pagina a la memòria cau simultàniament, per a recuperar-ho tot, caldrà fer l'accés a un total d'onze pagines!

En l'exemple del gràfic superior s'observa que per a recuperar seqüencialment les dades de la taula (pel ordre del camp índex cotxe), s'ha d'accedir a diferents pàgines aleatòriament. Funcionant així l'SGBD ha de renovar constantment les dades de la cache (hi ha pocs encerts), incrementant-se per tant, el nombre d'accessos físics al sistema d'emmagatzemament.

De tenir les dades de la taula ordenades pel mateix camp pel que s'ha creat l'índex, en fer una lectura seqüencial, només s'hauria de carregar una vegada cada pàgina de dades des de la BD, optimitzant les lectures, tot aconseguint uns temps d'execució menors.

PostgreSQL implementa la creació de CLUSTERs, per a aconseguir precisament que l'ordre de les dades emmagatzemades, coincideixin amb les d'un índex determinat.

D'utilitzar taula *trajecte* (utilitzades a l'apartat de benchmarks), primer caldrà crear un índex per la columna per la que es voldrà crear el cluster:

```
CREATE INDEX idx_cotxe_trajecte ON trajecte(cotxe);
```

Després crearem el cluster:

```
CLUSTER idx_cotxe_trajecte ON trajecte;
```

En tornar a executar la consulta del primer exemple, ens trobaríem amb un accés similar a:

Cotxe	Cotxe	P_Entrada	HoraEntrada	P_Sortida	HoraSortida
B-0000-TF	B-0000-TF	AP2-4	11-7-02 10:11 AM	AP2-8	11-7-02 11:11 AM
B-0001-OS	B-0001-OS	AP7-7	14-4-03 3:34 AM	AP7-12	14-4-03 3:50 AM
B-0001-OS	B-0001-OS	AP7-26	14-2-04 6:46 AM	AP7-2	14-2-04 7:06 AM
B-0001-RL	B-0001-RL	AP7-29	14-5-04 3:28 AM	AP2-2	14-5-04 3:48 AM
B-0001-SY	B-0001-SY	AP7-32	28-4-01 3:36 AM	AP7-27	28-4-01 4:01 AM
B-0001-VK	B-0001-VK	AP7-31	17-3-01 10:51 AM	AP2-9	17-3-01 11:01 AM
B-0002-OB	B-0002-OB	AP7-12	6-12-04 6:37 AM	AP2-6	6-12-04 7:27 AM
B-0002-WM	B-0002-WM	AP2-6	2-7-04 3:08 AM	AP7-4	2-7-04 3:28 AM
B-0003-TU	B-0003-TU	AP7-31	17-5-04 1:27 AM	AP2-3	17-5-04 1:57 AM
B-0003-WB	B-0003-WB	AP2-7	2-12-03 7:31 AM	AP7-10	2-12-03 8:11 AM
B-0004-PT	B-0004-PT	AP7-26	18-3-03 6:45 AM	AP2-11	18-3-03 7:15 AM
B-0004-RW	B-0004-RW	AP7-12	5-11-02 12:22 PM	AP7-25	5-11-02 12:52 PM
B-0004-SD	B-0004-SD	AP7-2	21-3-01 3:32 AM	AP7-9	21-3-01 3:52 AM
B-0004-UH	B-0004-UH	AP2-11	25-11-03 7:06 AM	AP7-28	25-11-03 7:26 AM
B-0004-WJ	B-0004-WJ	AP2-9	13-10-03 6:43 AM	AP7-26	13-10-03 7:23 AM

En aquest exemple seguim tenint un total de 5 pàgines físiques amb dades, però per a recuperar-les totes només caldrà fer 5 accessos al sistema d'emmagatzemament!

Es important anotar que l'ordenació de les dades només es fa durant el procés de creació del Cluster, pel que en afegir o modificar files, s'anirà perdent eficàcia en els accessos, tornant en el pitjor dels casos a la situació inicial.

Per a regenerar l'ordenació que es fa durant el procés de creació del Cluster, es pot fer:

```
CLUSTER nom_taula;
```

De voler-se incloure aquesta funció de regeneració dels índexs, en un procés automàtic (en hores de baixa activitat), també es pot indicar que es regenerin tots els Cluster existents, executant senzillament:

```
CLUSTER;
```

L'efecte d'utilitzar la clusterització, és similar a fer la seqüència següent:

```
SELECT * INTO TABLE nova_taula FROM taula_vella ORDER BY cotxe;
```

Per passar desprès a esborrar taula_vella, i renombrar nova_taula amb el nom antic.

En el cas d'utilitzar la instrucció específica, asseguraríem si en un futur es té en compte l'existència del cluster en executar-se el pla de consulta, l'analitzador triï l'alternativa més òptima.

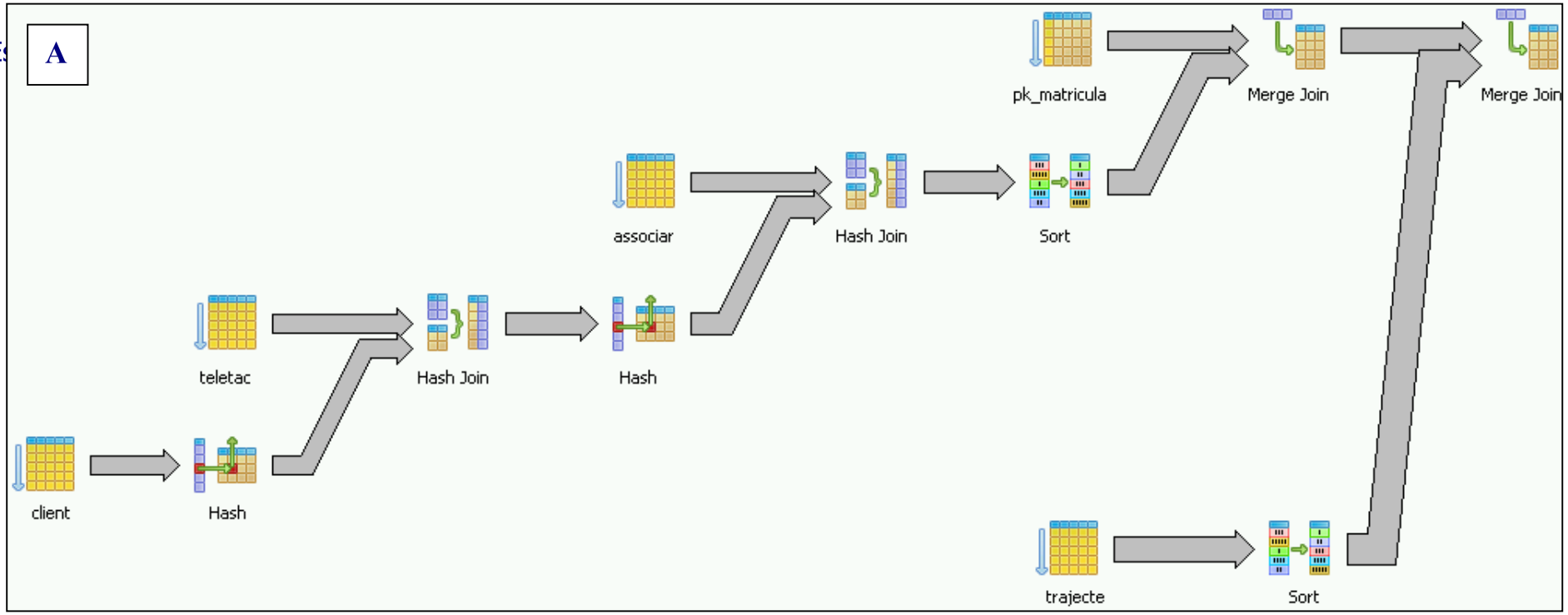
7.3.- Analyze i els plans d'execució

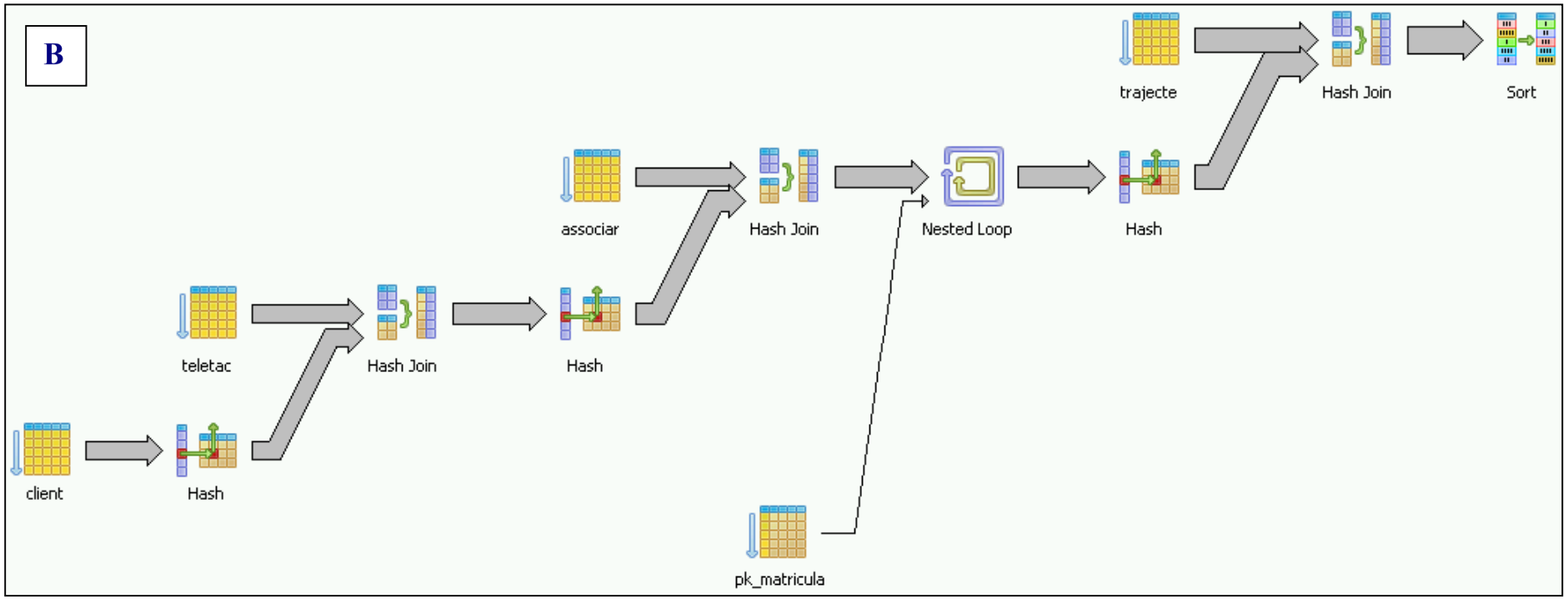
S'utilitza Analyze (o Analyse) per a fer que l'SGBD recopili informació de les taules, ocupacions, fragmentació, existència i tipus d'índex (entre d'altres), per a realitzar les estadístiques que permetran desprès al planificador triar el millor mètode d'execució de les consultes.

En l'apartat de benchmarks es fa referència a les diferències en el temps d'execució d'una mateixa consulta (veure també Annex III), de fer-se abans o desprès d'executar el Analyze. Els diferents camins d'execució es poden observar en el gràfic A i B (pàgina següent).

Els dos plans d'execució corresponen a la planificació de la consulta:

```
SELECT client.DNI, teletac.Contracte, associar.Teletac,
          cotxe.Matricula, trajecte.PeatgeEntrada,
          trajecte.HoraEntrada
     FROM client, teletac, associar, trajecte, cotxe
     WHERE client.DNI = teletac.Client
          AND teletac.Contracte = associar.Teletac
          AND associar.Cotxe = cotxe.Matricula
          AND cotxe.Matricula = trajecte.Cotxe
          AND client.Nom = 'Maria'
     ORDER BY matricula;
```


A
pk_matricula
Merge Join
Merge Join
associar
Hash Join
Sort
teletac
Hash Join
Hash
client
Hash
trajecte
Sort


B
trajecte
Hash Join
Sort
associar
Hash Join
Nested Loop
Hash
teletac
Hash Join
Hash
client
Hash
pk_matricula

El primer pla d'execució és el que l'SGBD ha fet no tenint actualitzades les estadístiques de la base de dades. En utilitzar l'*analyze* i tornar a repetir la consulta, el planificador ha triat una altre camí en el procés de planificació de l'execució de la mateixa.

El temps d'execució, segons l'entorn descrit en l'apartat de Benchmarks, passa dels 23,7 segons inicials als 2,4 segons, degut bàsicament a que en el cas B no s'executa el pas d'ordenació dels registres de la taula trajecte (de 459.267 registres!).

Amb aquest exemple es constata que s'hauria de planificar l'execució de la instrucció Analyze periòdicament. El fer-ho en moments de poca activitat de l'SGBD pot ser important, ja que depenent de la mida de les taules i d'altres factors, el procés pot ser costós en temps i en utilització de maquinari.

El detall textual dels plans d'execució de la consulta de prova es poden consultar a l'annex II.

A l'annex III es troba el pla d'execució òptim, un cop fetes les possibles millores a la BD.

7.4.- Índex calculats

En els casos en que s'hagi d'accedir freqüentment a una taula utilitzant una expressió, pot ser òptim utilitzar un índex calculat. De fer-ho així, en fer-se insercions o modificacions a un dels camps implicats en el índex calculat, el valor d'aquest s'actualitzarà, i podrà ser utilitzat posteriorment a les consultes per accedir ràpidament als registres d'interès.

Exemple:

```
CREATE INDEX idx_nom ON persona (UPPER(nom));
```

A partir d'aquest moment, en fer una cerca en majúscules pel camp nom de persona, s'utilitzarà aquest índex (en el cas que el nombre de registres de la taula sigui prou gran, com per a ser òptim).

De consultar el contingut d'una còpia de seguretat feta amb el pg_dump (punt 12 de la memòria), s'observarà que les dades generades durant la creació de l'índex no apareixen. A la còpia només constarà la instrucció de creació de l'índex, per a que tal com es recuperi aquesta instrucció, les dades associades a aquest es torni a generar (la copia de seguretat podria ocupar menys espai que l'ocupat pels fitxers de dades i índexs de la BD!).

A l'annex I s'ha preparat un exemple complert d'índex calculat, basat en un possible cas real.

7.5.- Índex parcials

Els índex parcials són aquells que es creen per a ser utilitzats només per accedir a un determinat conjunt de dades d'una taula. Per a ser utilitzats pel planificador d'accés, el predicat que s'utilitza en la consulta ha de coincidir amb l'utilitzat per a crear l'esmentat índex.

Exemple:
En el cas d'haver d'accedir a un volum important d'informació, segons un camp on consta l'any, es podria crear un índex parcial, amb la següent instrucció:

```
CREATE INDEX idx_trajecte_any2003
     ON trajecte (horaentrada)
     WHERE SUBSTRING(horaentrada FROM 7 FOR 4) = '2003';
```

En fer l'*explain* de la següent consulta:

```
SELECT *
     FROM trajecte
     WHERE SUBSTRING(horaentrada FROM 7 FOR 4) = '2003';
```

comprovarem que se'ns indicar que s'utilitza l´índex per a fer l'accés a les dades:

```
Index Scan using idx_trajecte_any2003 on trajecte
  (cost=0.00..8234.22 rows=2297 width=76)
  Filter: ("substring"((horaentrada)::text,7,4)= '2003'::text)
```

8.- Model de concurrència

Les sigles MVCC són una constant en buscar qualsevol tipus d'informació relacionada amb el bloquejos de taules i registres, i la simultaneïtat d'accessos en el PostgreSQL.

El Multi-Version Concurrency Control és el mecanisme que implementa aquest SGBD per augmentar el rendiment de les consultes i escriptures, tot evitant haver d'utilitzar els mecanismes de bloqueig de registre i/o de taules. L'objectiu és que les consultes i les escriptures a la BD no interfereixin entre si.

Tal com es diu en el propi manual d'ús, la finalitat és que 'els escriptors mai hagin d'esperar als lectors, i que els lectors mai hagin d'esperar als escriptors'.

Les bases de dades que no utilitzen aquest mecanisme de concurrència, asseguren que les dades tinguin una integritat ACID mitjançant els mecanismes 'tradicionals' de bloquejos de registres;

- En fer una lectura, es comprova que el registre no s'estigui modificant, o no estigui blocat per una altra transacció. Si és així, s'aturaria fins que quedés alliberat.
- En fer una escriptura, una modificació, o un esborrat, els registre es marquen com a blocats per a que cap altra procés el pugui utilitzar mentre es porta a terme una de les tasques enumerades.

Procedint d'aquesta manera la BD assegura la consistència esmentada, ja que es compleixen les condicions ACID en les transaccions:

- Atomicitat: Cal que totes les operacions d'una transacció s'executin completament, o en cas contrari, que no s'executi cap en absolut.
- Consistència: Cal que les transaccions no violin cap de les regles d'integritat relacional establertes a la base de dades.
- Isolació: Cal que mentre s'executa una transacció, altres processos no puguin obtenir una visió parcial de les diferents modificacions que es poden fer dels registres.
- Durabilitat: Les dades han de perdurar, fins i tot en cas de fallada de maquinari o de programari.

Les bases de dades que no implementen el MVCC tenen fortes penalitzacions en el rendiment en executar moltes transaccions concurrentment, ja que moltes de les que hi haurà a la cua de pendents d'execució, hauran d'esperar per a processar-se a que s'alliberi alguna fila blocada per un altra procés (o pot haver-se de repetir, en fer l'execució i no poder recuperar una fila determinada per aquest mateix blocatge).

Aquest mètode tradicional, també s'anomena d'execució serialitzada, ja que cada transacció ha d'esperar (en part) a que acabi l'anterior. És evident que amb aquest mètode, l'ampliabilitat futura de la base de dades es pot veure compromesa, ja que malgrat s'augmentin els recursos del maquinari (processadors, memòria, etc) el rendiment no augmentarà en proporció a aquesta major potència.

Els SGBD que s'ha pogut verificar que utilitzen el mecanisme MVCC per a millorar la concurrència són: Oracle, Interbase i PostgreSQL. Totes les altres (incloses DB2 i MySQL), utilitzen la serialització i bloquejos.

8.1.- Funcionament MVCC

Conceptualment sembla complicat que la BD pugui tenir constància per a cada transacció, de quin és l'estat de tots els registres que puguin intervenir en la mateixa.

La solució que s'utilitza, a més de ser elegant, és ràpida, segura i senzilla:
El sistema MVCC utilitza una marca de temps en cada transacció de modificació de files, per que quedi en el registre implicat la 'marca' identificativa de la versió del registre. En fer una de les esmentades operacions de modificació, la versió del registre que s'ha modificat no s'esborra. A partir del moment de l'esmentada modificació, existiran dos (o més) registres a la base de dades. Cadascun tindrà una marca de temps que correspondrà a la transacció que l'ha modificat. Funcionant d'aquesta manera, a la BD hi hauran vàries versions d'un mateix registre, cadascun amb la marca de temps del moment del seu canvi. És important notar que en fer la modificació d'un registre, l'espai que ocupa aquest a la BD queda doblat. Només en fer una neteja especifica (amb la instrucció VACUUM), s'alliberarà l'espai ocupat per les versions velles dels registres.

Com que PostgreSQL crea una nova còpia del registres que s'està modificat, els altres usuaris poden estar utilitzant els valors originals, mentre no es consolidin (commit) les noves dades. Les consultes simultànies de tipus READ COMMITTED veuran d'immediat les noves dades, mentre que les que es facin en mode SERIALIZABLE, seguiran executant-se correctament, tot utilitzant les dades de la versió vella del registre.
En el cas de desfer la transacció (rollback), les consultes en curs no es veuran afectades.

L'avantatge d'aquest funcionament consisteix en el seguiment que es fa de les versions de les files, evitant els conseqüents blocats a nivell de taula o de fila (malgrat això, aquest tipus de blocats concrets existeixen, i es poden utilitzar de necessitar-se).

El mateix sistema que identifica les versions dels registres s'utilitza en el mòdul de PostgreSQL time travel. Aquest permet fer consultes tot indicant el moment determinat en el temps en que es volen obtenir els resultats. Com que en aquest cas es mantenen totes les versions dels registres modificats, l'SGBD tindrà en compte el time-stamp d'aquests, per resoldre l'operació segons les dades existents en moment demanat donat.

Una altra avantatja del MVCC és que es poden fer còpies de seguretat en calent, ja que en començar la còpia, tindrem un ID de transacció, pel que només caldrà emmagatzemar els registres amb ID inferior, per a tenir una versió de la BD consistent salvaguardada.

8.2.- Exemples MVCC vs blocat de registres

Per a comprovar la consistència ACID i els avantatges del MVCC, s'executaran sobre la mateixa BD, un conjunt de tres proves, tot intentant reproduir la simultaneïtat d'accessos a l'SGBD per part de varis clients. Aquestes proves actualitzaran i consultaran un conjunt reduït de registres, tot buscant la coincidència en els accessos, i aquells casos que es consideren crítics.

Per a comparar les diferències del MVCC vers un model de concurrència per serialització, es faran les mateixes proves amb PostgreSQL 8.03 i Informix 7.33.

Prova A

Amb aquesta prova s'intentarà verificar el correcte aïllament entre dues transaccions. S'inserirà un registres des d'una d'elles, i aquest no ha d'aparèixer en l'altra, almenys fins que no es tanqui la transacció des de la que s'ha afegit el registre.

PostgreSQL 8.03

Consola 1

	Comanda	Resultat
1	start transaction isolation level serializable;	
2		start transaction
3	create table test (camp1 varchar(50));	
4		create table
5	commit;	
6		commit
7		
8		
9	start transaction isolation level serializable;	
10		start transaction
11	insert into test values ('Reg num 1');	
12		inserted 1
13		
14		
15	commit;	
16		commit
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		

Consola 2

	Comanda	Resultat
1		
2		
3		
4		
5		
6		
7	start transaction isolation level serializable;	
8		start transaction
9		
10		
11		
12		
13	select * from test;	
14		0 rows returned
15		
16		
17	select * from test;	
18		0 rows returned
19	commit;	
20		commit
21	start transaction isolation level serializable;	
22		start transaction
23	select * from test;	
24		retorna: 'Reg num 1'
25	commit;	
26		commit

S'observa que:

- Primer es crea la taula que s'utilitzarà per a fer les proves (línies 1 a 5).
- S'engega la primera transacció des de la cònsola 2.
- La inserció es fa des de la cònsola 1 (línia 11).
- Les consultes que es duen a terme des de la cònsola 2 no retornen la dada inserida des de la cònsola 1.
- Malgrat es fa el *commit* a la cònsola 1 (línia 15), el nou registres no apareix a la cònsola 2 (línia 17).
- Només en tancar la transacció de la cònsola 2, i obrir una de nova, es retorna el nou registre.

Amb aquesta prova s'observa que les operacions lectores no han d'esperar les escriptores, i que al llarg de totes les transaccions, aquestes treballen amb les dades que havien en el moment de començar la mateixa.

Informix 7.33

Consola 1

	Comanda	Resultat
1	begin;	
2		transaction started
3	set transaction isolation level serializable;	
4		isolation level set
5	set lock mode to wait;	
6		lock mode set
7	create table test (camp1 varchar(50)) lock mode row;	
8		table created
9	commit;	
10		transaction committed
11		
12		
13		
14		
15		
16		
17	begin;	
18		transaction started
19	set transaction isolation level serializable;	
20		isolation level set
21	set lock mode to wait;	
22		lock mode set
23	insert into test values ('Reg num 1');	
24		1 row inserted
25		
26		
27	commit;	
28		transaction committed
29		
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		
40		
41		
42		
43		
44		

Consola 2

	Comanda	Resultat
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11	begin;	
12		transaction started
13	set transaction isolation level serializable;	
14		isolation level set
15	set lock mode to wait;	
16		lock mode set
17		
18		
19		
20		
21		
22		
23		
24		
25	select * from test;	
26		**Entra en estat d'espera**
27		
28		
29		**Surt d'estat espera i retorna fila:**
30		'Reg num 1'
31	select * from test;	
32		'Reg num 1'
33	commit;	
34		transaction committed
35	begin;	
36		transaction started
37	set transaction isolation level serializable;	
38		isolation level set
39	set lock mode to wait;	
40		lock mode set
41	select * from test;	
42		'Reg num 1'
43	commit;	
44		transaction committed

S'observa que:

- En primer lloc es crea la taula que s'utilitzarà en les tres proves, amb el mode de blocat per registre (la BD sobre la que es crea la taula és de tipus ANSI).
- S'obra en primer lloc una transacció des de la cònsola 2.
- Des de la cònsola 1 s'insereix un registre (línia 23).
- Des de la cònsola 2 es fa una consulta a la taula, i es queda en mode 'd'espera' fins que no es fa el *commit* a la cònsola 1.
- Des del mateix moment en que s'ha 'comitat' a la cònsola 1, es pot tornar a fer una consulta des de la transacció de la cònsola 2, i malgrat aquesta hagi començat en primer lloc, ja apareix el registre inserit des de la cònsola 1.

Diferències: L'aïllament entre els dos SGBD és completament diferent, ja que mentre amb PostgreSQL la transacció utilitzarà les dades que hi havien en el moment en que aquesta ha començat, en Informix s'utilitzaran totes aquelles dades que estiguin 'comitades'. En cap cas l'SGBD PostgreSQL ha tingut que introduir estats d'espera per a poder executar les transaccions correctament.

Prova B

La prova B és similar a la A, però canviat l'ordre en que s'inicien les transaccions de la cònsola 1 i 2.
S'utilitza la taula creada a la prova A, inicialment buida.

PostgreSQL 8.03

Consola 1

	Comanda	Resultat
1	start transaction isolation level serializable;	
2		start transaction
3		
4		
5	insert into test values ('Reg num 1');	
6		inserted 1
7		
8		
9	commit;	
10		commit
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		

Consola 2

	Comanda	Resultat
1		
2		
3	start transaction isolation level serializable;	
4		start transaction
5		
6		
7	select * from test;	
8		0 rows returned
9		
10		
11	select * from test;	
12		0 rows returned
13	commit;	
14		commit
15	start transaction isolation level serializable;	
16		start transaction
17	select * from test;	
18		retorna: 'Reg num 1'
19	commit;	
20		commit

S'observa que:

- La primera transacció s'inicia des de la cònsola 1.
- La inserció de la línia 5 es fa una vegada iniciada la transacció de la cònsola 2.
- Malgrat s'ha 'comitat' a la cònsola 1, a les *select* de les línies 7 i 11 no es retorna cap registre.
- Només un cop s'ha tancat la transacció en curs, i s'ha iniciat una de nova, apareix el nou registre.

Informix 7.33

Consola 1

	Comanda	Resultat
1	begin;	
2		transaction started
3	set transaction isolation level serializable;	
4		isolation level set
5	set lock mode to wait;	
6		lock mode set
7		
8		
9		
10		
11		
12		
13	insert into test values ('Reg num 1');	
14		1 row inserted
15		
16		
17	commit;	
18		transaction committed
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		
31		
32		
33		
34		

Consola 2

	Comanda	Resultat
1		
2		
3		
4		
5		
6		
7	begin;	
8		transaction started
9	set transaction isolation level serializable;	
10		isolation level set
11	set lock mode to wait;	
12		lock mode set
13		
14		
15	select * from test;	
16		**Entra en estat d'espera**
17		
18		
19		**Surt d'estat espera i retorna fila:**
20		'Reg num 1'
21	select * from test;	
22		retorna: 'Reg num 1'
23	commit;	
24		transaction committed
25	begin;	
26		transaction started
27	set transaction isolation level serializable;	
28		isolation level set
29	set lock mode to wait;	
30		lock mode set
31	select * from test;	
32		retorna: 'Reg num 1'
33	commit;	
34		transaction committed

S'observa que:

- La primera transacció s'inicia a la cònsola 1, en mode serialitzable, i amb espera en cas de blocat de registre.
- La inserció es fa des de la cònsola 1, i un cop fet el *commit* en aquesta, ja es pot utilitzar el nou registre des de la cònsola 2 (tal com surt de l'estat d'espera ja el retorna – línia 20).

Diferències:

- Al igual que a la primera prova, s'observa que mentre PostgreSQL treballa amb les dades que existeixen quan s'inicia la transacció, Informix utilitza les que hi han en el moment 'comitades'.
- En el PostgreSQL, els processos lectors no han d'esperar als escriptors (no s'entra en cap estat d'espera), tal com s'indica en les especificacions del MVCC, pel que la concurrència serà més elevada, havent menys interferències entre processos.

Prova C

A la darrera prova s'intentarà comprovar com es comporta el MVCC en els casos de coincidència en l'actualització de registres.
S'ha de tenir en compta que s'utilitza la taula creada a la prova A, i que inicialment existeix el registre que s'observa al final de la prova B.

PostgreSQL 8.03

Consola 1

	Comanda	Resultat
1	start transaction isolation level serializable;	
2		start transaction
3		
4		
5	insert into test values 'Reg num 2';	
6		inserted 1
7		
8		
9	select * from test	
10		retorna: 'Reg num 1'
11		'Reg num 2'
12		
13		
14	update test set camp1 = 'Nou Reg 1.2' where camp1 = 'Reg num 1';	
15		**Entra en estat d'espera**
16		
17		
18		**ERROR:** could not serialize acces
19		due to concurrent update
20	commit;	
21		rollback
22	start transaction isolation level serializable;	
23		start transaction
24	select * from test;	
25		retorna: 'Nou Reg 1.1'
26	commit;	
27		commit

Consola 2

	Comanda	Resultat
1		
2		
3	start transaction isolation level serializable;	
4		start transaction
5		
6		
7	update test set camp1 = 'Nou Reg 1.1' where camp1 = 'Reg num 1';	
8		update 1
9		
10		
11		
12	select * from test	
13		retorna: 'Nou Reg 1.1'
14		
15		
16	commit;	
17		commit
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		

S'observa que:

- La primera transacció s'inicia a la cònsola 1.
- Es fa una inserció des de la cònsola 1, desprès de que s'inicií la transacció de la cònsola 2.
- Des de la cònsola 2 s'actualitza el contingut del registre que existeix inicialment a la taula.
- Des de la cònsola 1 es recupera el contingut de la taula, des del seu punt de vista, i es comprova que existeix el registre inicial, i el que s'acaba d'inserir.
- Des de la cònsola 2 també es fa una consulta del contingut de la taula, i s'observa que el registre s'ha actualitzat correctament.
- S'intenta actualitzar el registre que existia inicialment des de la cònsola 1, i s'entra en estat d'espera, donat que es detecta que ha estat actualitzat desprès d'iniciar la transacció.
- En fer el commit de la cònsola 2, la primera surt de l'estat d'espera, i en fer el commit, en lloc de 'comitar', fa un rollback per avortar tot el procés (eliminant-se la dada que s'havia inserit).
- El procés que ha aconseguit actualitzar el registre, ha estat el que primer ho ha intentat, malgrat la transacció no haguí començat en primer lloc.

Informix 7.33

Consola 1

	Comanda	Resultat
1	begin;	
2		transaction started
3	set transaction isolation level serializable;	
4		isolation level set
5	set lock mode to wait;	
6		lock mode set
7		
8		
9		
10		
11		
12		
13	insert into test values ('Reg num 2');	
14		1 row inserted
15		
16		
17	select * from test;	
18		retorna: 'Reg num 1'
19		'Reg num 2'
20	update test set camp1 = 'Nou Reg 1.2' where camp1 = 'Reg num 1';	
21		1 row update
22	commit;	
23		transaction committed
24		
25		
26		
27		
28	begin;	
29		transaction started
30	set transaction isolation level serializable;	
31		isolation level set
32	set lock mode to wait;	
33		lock mode set
34	select * from test;	
35		retorna: 'Nou Reg 1.2'
36		'Reg num 2'
37	commit;	
38		transaction committed

Consola 2

	Comanda	Resultat
1		
2		
3		
4		
5		
6		
7	begin;	
8		transaction started
9	set transaction isolation level serializable;	
10		isolation level set
11	set lock mode to wait;	
12		lock mode set
13		
14		
15	update test set camp1 = 'Nou Reg 1.1' where camp1 = 'Reg num 1';	
16		**Entra en estat d'espera**
17		
18		
19		
20		
21		
22		
23		
24		**Surt d'estat espera:**
25		0 row updated
26	commit;	
27		transaction committed
28		
29		
30		
31		
32		
33		
34		
35		
36		
37		
38		

S'observa que:

- La primera transacció s'inicia des de la cònsola 1.
- Malgrat que en crear la taula s'ha especificat 'lock mode row', en fer la inserció de la línia 13, l'actualització de la línia 15 entra en estat d'espera.
- Només en fer el commit a la cònsola 1, l'operació que ha quedat latent continua.
- Al fer la consulta final, s'observa que s'ha porta a terme la inserció i l'actualització (línia 34 a 36).

Diferències:

Donades les diferències en el blocat de registres, el resultat final no és el mateix. Mentre que al PostgreSQL avorta (correctament) tota una transacció, per haver-se trobat un registre que s'utilitza modificat (des del moment en que s'ha començat la transacció), Informix no ho fa, ja que bloca la taula en el moment de fer la inserció a la línia 13, i no permet que es faci cap operació amb aquesta fins que es fa el *commit* (línia 22). En aquest cas, l'ordre de les operacions si és important, i s'hauria de fer blocats explícits, de voler un comportament determinat.

8.3.- Blocats de taules i registres explícits

Per molt que el mecanisme MVCC permeti un major concurrència d'accessos, fent els blocats a nivell de taula o de registre de forma automàtica, en determinats casos pot ser necessari fer els esmentats blocats de forma explicita.

Els modes de blocat a nivell de taula que té PostgreSQL són:
ACCESS SHARE
ROW SHARE
ROW EXCLUSIVE
SHARE UPDATE EXCLUSIVE
SHARE
SHARE ROW EXCLUSIVE
EXCLUSIVE
ACCESS EXCLUSIVE

A nivell de fila el blocat es produeix automàticament en fer-se una modificació del registre, o un esborrat d'aquest. Pel mateix mecanisme del MVCC les lectures no es veuen afectades pels blocats del registres (però si que es veurien aturats els processos de modificació o esborrat de registres concrets).

Tal com està implementat el mecanisme de reserva de files, no s'emmagatzema cap informació en memòria, pel que no hi ha límit pel nombre de registres que es poden tenir blocats en un moment determinat. Com que cada reserva implica una escriptura al disc, s'ha de comptar amb la sobrecarrega del subsistema d'emmagatzemament que això pot produir.

Exemple:
Volem esborrar el registre que conté les dades de la compra de més import d'un mes concret. Com que es vol esborrar el mestre-detall, assegurant que cap altra procés pugui interferir, es bloquen les actualitzacions de la taula despeses. Un cop obtingut el 'id' màxim, es passa a esborrar el registre del 'detall' i posteriorment el del mateix 'mestre'.

```
CREATE OR REPLACE FUNCTION esborraDespesa(despeses.mes%TYPE)
  RETURNS int AS '
DECLARE
  idCompra INT;
BEGIN
  LOCK TABLE despeses IN SHARE ROW EXCLUSIVE MODE;
  SELECT ordreCompra into idCompra
      FROM despeses
      WHERE mes = $1
      ORDER BY TotalCompra DESC LIMIT 1;
  DELETE FROM comandes WHERE Compra = idCompra;
  DELETE FROM despeses WHERE ordreCompra = idCompra;
return 0;
end;
' language 'plpgsql';
```

La possibilitat de fer blocats de forma explicita és especialment important d'utilitzar-se connexions concurrents amb l'SGBD, ja que cadascuna d'aquestes connexions tindrà una visió diferent del que hi ha a les taules, donat que el MVCC crearà un 'vista-congelada' de l'estat de la base de dades a l'inici de les transaccions. En procedir així, dues transaccions que siguin 'llançades' gairebé en paral·lel pel mateix programa (cadascuna amb la seva pròpia sessió amb la BD), treballaran sobre conjunts de dades diferents. D'aquí l'especial importància d'utilitzar els blocats explícits en els casos especials.

Aquestes situacions també es poden produir en interactuar diversos programes (OO) o sistemes, cadascun amb la seva pròpia sessió amb la BD.

9.- Comparatives entre SGBDs

Consultant la informació que faciliten els diferents fabricants, les característiques dels productes, i utilitzant alguns pressupostos que s'han obtingut, s'han confeccionat els quadres-resum d'aquest apartat.

La disponibilitat dels SGBD per a diferents entorns, cada vegada és més habitual. S'ha de notar que gairebé totes tenen versió per a Linux

	Disponibilitat Sistemes Operatius				
	Windows	Mac OS X	Linux	BSD	Unix
PostgreSQL	√	√	√	√	√
MySQL MyISAM	√	√	√	√	√
Informix	√	√	√	√	√
DB2	√	•	√	•	√
Oracle	√	√	√	•	√
Microsoft SQL Server	√	•	•	•	•
Ingres	√	√	√	√	√

√ L'SGBD compleix les característiques que figuren a la capçalera de la columna.
• L'SGBD no té la característica concreta.

Que un SGBD no tingui integritat referencial, ni sigui ACID, ni pugui executar grups d'instruccions transaccionalment, pot evidenciar que pot no ser utilitzable en molts contexts.

També s'ha de comparar el tipus de control de la concurrència, ja que d'utilitzar-se el un sistema similar al MVCC, es millorarà la contenció en sistemes d'alt ús.

El tipus de llicència del programari també pot ser un factor determinant. S'ha de sospesar l'estalvi vers el fet que hagi una empresa fabricant que respongui del producte.

	Característiques generals					
	ACID	Integritat referencial	Transaccions	Unicode	Model de Concurrència	Llicència d'ús
PostgreSQL	√	√	√	√	MVCC	BSD
MySQL MyISAM	•	•	•	√	Bloqueig	Dual Propietari/GPL
Informix	√	√	√	√	Bloqueig	Propietari
DB2	√	√	√	√	Bloqueig	Propietari
Oracle	√	√	√	√	MVCC	Propietari
Microsoft SQL Server	√	√	√	√	Bloqueig	Propietari
Ingres	√	√	√	√	Bloqueig	CATOSL (similar GPL)

Depenent de l'entorn en el que tingui que funcionar l'SGBD i del tipus d'ús que es faci, s'hauran de valorar més uns aspectes que d'altres, però en tots els casos, sembla imprescindible poder disposar de disparadors i de procediments emmagatzemants,

	Característiques avançades (a)			
	Taules temporals	Vistes Materialitzades	Dominis	Cursors
PostgreSQL	√	Similar	√	√
MySQL	√	●	●	●
Informix	√	√	●	√
DB2	√	√	●	√
Oracle	√	√	√	√
Microsoft SQL Server	√	Similar	●	√
Ingres	√	●	√	√

	Característiques avançades (b)			
	Disparadors	Funcions	Procediments emmagatzemats	Procediments en altres llenguatges
PostgreSQL	√	√	√	√
MySQL	●	●	●	√
Informix	√	√	√	√
DB2	√	√	√	√
Oracle	√	√	√	√
Microsoft SQL Server	√	√	√	√
Ingres	√	√	√	√

En els costos de funcionament d'un SGBD, el preu inicial pot tenir més o menys incidència. Caldrà assegurar que aquest queda justificat per les característiques que incorpora el programari.

En alguns casos, s'especifica un preu determinat per 'CPU'. Això significarà que l'SGBD pot ser utilitzat per tants usuaris com permeti el servidor amb el nombre de processadors que consten a la llicència adquirida.

El preu de l'SGBD en modalitat 'per usuari' pot ser inicialment més baix, però en haver-se d'ampliar el nombre de llicencies d'ús (desprès dels 5 o 10 primers usuaris - habitualment ja inclosos en el preu inicial), s'haurà d'adquirir una llicència d'ús per a cadascun d'aquests (segons columna 'preu connexió individual').

Els preus relacionats només s'han de considerar indicatius, ja que hi han altres molts factors que poden fer variar aquest. Un mateix SGBD pot tenir diverses versions. Així

una pot ser bàsica, i una altra d'alt rendiment. S'ha posat com exemple d'aquest cas, els preus del Oracle Estandard Edition i del Oracle Enterprise Edition.

	Preus i Costos d'ús	
	Preu SGBD	**Preu connexió individual**
PostgreSQL	**0€**	**0€**
MySQL	**0€** (en utilització GPL)	**0€** (en utilització GPL)
Informix Dynamic Server Express Ed.	**3730€** (processadors il·limitats)	**95€**
DB2 Express Ed.	**3730$**	**75€**
Oracle Estandard Ed.	**4500€ (per 1 CPU)**	**130€**
Oracle Enterprise Ed.	**35000€ (per 1 CPU)**	**540€**
Microsoft SQL Server Standard	**6700€ (per 1 CPU)**	
Ingres	**0€**	**0€**

10.- Instal.lació PostgreSQL i utilitats

Donat que el present estudi ha de servir per a assegurar la idoneïtat d'ús d'aquest SGBD, s'ha considerat que part fonamental del mateix era assegurar la correcta instal·lació sobre els sistemes operatius: Linux, Windows 95a, 98se, ME, Windows 2000 server (sp4) i Windows XP(sp2). Aquestes instal·lacions s'han complementat amb els corresponents manuals d'instal·lació i ús, i amb els manuals suplementaris (instal·lació i ús) de l'editor SQL SQuirreL (versió Linux i Windows XP).

Per a realitzar els manuals (adjunts en format d'annex, del V al XI), s'ha procedit a fer la instal·lació de cada sistema operatiu en un disc dur, a fi efecte de poder fer un seguiment detallat de les incidències, reproduir les proves tal com calgués, i poder fer en un segon pas, la instal·lació de l'editor SQuirreL, per a comprovar que la connexió d'aquest amb l'SGBD, mitjançant JDBC, era correcta.

El maquinari utilitzat per a les proves d'instal·lació amb Windows 95a/98se/ME, ha estat:

- Pentium III 350Mhz, 256Mb de memòria RAM, amb els corresponents discs dur IDE (amb capacitat d'entre els 2 als 4Gb, un per a cada SO).

Per a les proves d'instal·lació amb Linux i Windows XP s'ha utilitzat:
- Portàtil Dell Latitude D505, amb processador Pentium M de 1.7Ghz, 768Mb de memòria RAM i disc dur de 60Gb.

Per la prova d'instal·lació amb Windows 2000 (server), s'ha utilitzat un equip clònic amb les següents característiques:
- Pentium 4 de 2,8 Ghz, amb 512Mb de memòria RAM i disc dur de 60Gb.

Les instal·lacions fetes sobre els Windows 95a/98se i ME s'han realitzat tenint únicament instal·lat en el disc dur el sistema operatiu i els últims afegits de seguretat corresponents, mentre els sistemes operatius Windows 2000 (server) i XP sobre els que s'han instal·lat les corresponents versions, ja contaven amb força temps d'ús i d'altra programari instal·lat.

Comptant amb el manual del PostgreSQL CD Live, s'han realitzat un total de sis guies d'instal·lació i ús:

- Manual d'instal·lació PostgreSQL per a Windows 95/98/ME.
- Manual d'instal·lació PostgreSQL per a Windows 2000/XP.
- Manual d'instal·lació PostgreSQL per a Linux.
- Manual d'instal·lació SQuirreL per a Windows.
- Manual d'instal·lació SQuirreL per a Linux.
- Manual de funcionament de PostgreSQL CD Live!

Remarcar que les guies d'instal·lació estan fetes pensant en que seran utilitzades per usuaris inicialment neòfits, pel que s'ha evitat relatar la complexitat inherent a la realització de les mateixes.

11.- Compatibilitat amb altre programari

A l'instal·lar-se els SGBD com un *dimoni* o un servei del propi sistema operatiu (quedant aquest a l'espera de rebre peticions per un port de comunicacions), la possibilitat de que es produeixin incompatibilitats amb altre programari pot ser alta.

Per aquest motiu s'ha ideat un joc de proves per a verificar el correcte funcionament del PostgreSQL en entorns on també hi ha altres tipus de programari:

- De seguretat (antivirus, firewalls i similars).
- De tipus estàndard (dels que també s'instal·len com a serveis), que es poden utilitzar en l'àmbit acadèmic al llarg d'una Enginyeria, o una Enginyeria Tècnica Informàtica.
- Altra programari d'ús personal 'estàndard'.

S'ha prioritzat les comprovacions del que es preveu que pot ser l'entorn més habitual d'execució, o sigui, el d'un PostgreSQL 8.01 sobre un sistema operatiu Windows XP.

La metodologia seguida per a fer les comprovacions ha estat (per a cada programa):

- Instal·lació del programari que es vol comprovar (opcions per omissió).
- Instal·lació de PostgreSQL 8.01
- Instal·lació SQuirreL
- Engegat del servei de l'SGBD.
- Execució de SQuirreL
- Connexió SQuirreL amb PostgreSQL mitjançant JDBC
- Creació d'una taula
- Inserció d'un registre
- Consulta de les dades inserides.
- Tancament de l'editor SQuirreL
- Parada servei de l'SGBD
- Desinstal.lació PostgreSQL
- Esborrat directori de dades de PostgreSQL
- Esborrat del directori del SQuirreL
- Desinstal.lació del programari que s'està provant

L'ordinador que s'ha utilitzat per a executar les proves és de tipus clònic, té processador Intel Pentium 4 de 2,4Ghz, 512Mb de memòria RAM i un disc dur de 80GB.

S'ha optat per utilitzar una instal·lació de Windows XP ja existent, per assegurar que altre programari estàndard (suites d'ofimàtica, java, utilitats, etc), i les habituals instal·lacions i desinstal·lacions que es fan al llarg del temps, poguessin aportar un grau més de seguretat en la cerca de possibles incompatibilitats.

La versió del Windows XP que s'ha utilitzat ha estat la 'Professional Corporate' (5.1.2600), amb tots els afegits de seguretat addicionals publicats per Microsoft fins a la data.

Simbologia utilitzada en la representació dels resultats a les taules:

√ Funcionament correcte

● No possible instal·lació/funcionament sense deshabilitar/reconfigurar prèviament.

11.1.- Software de seguretat

	PostgreSQL 8.01 (2000/XP)		SQuirreL	
	Instal·lació	**Execució**	**Instal·lació**	**Execució**
Firewall Windows XP (en mode estricte)	√	√	√	√
AntiSpyware Microsoft	√	√	√	√
Mcafee Desktop Firewall	√	√	√	√
Panda Antivirus platinium internet security 2005	√	√	√	√
AVG Antivirus 7.1.371	√	√	√	√
Norton Antivirus 2006	√	√	√	√
Zone Alarm Firewall	●[1]	√	√	√

[1] Si es configura en mode 'zona internet' cal permetre l'execució del programari com a servei, i la utilització dels ports de comunicació corresponents.

11.2.- Programari estàndard UOC

	PostgreSQL 8.01 (2000/XP)		SQuirreL	
	Instal·lació	**Execució**	**Instal·lació**	**Execució**
Synera 3.1.2	√	√	√	√
MySQL 4.1.13	√	√	√	√
Informix 7.33	√	√	√	√
Apache	√	√	√	√
Tomcat	√	√	√	√
Eclipse 3.0	√	√	√	√
MS asp.NET WebMatrix v0.6	√	√	√	√

11.3.- Altre programari

	PostgreSQL 8.01 (2000/XP)		SQuirreL	
	Instal·lació	**Execució**	**Instal·lació**	**Execució**
EMule 0.44d	√	√	√	√
Bittorrent 3.42	√	√	√	√
Kazaa	*Impossibilitat de provar per tancament de servei (querella DMCA)			
eDonkey2000 1.4.3	√	√	√	√
Acrobat Reader 7.0	√	√	√	√
WinZIP 9.01	√	√	√	√

12.- Benchmarks; PostgreSQL vs MySQL

En parlar de Benchmarks de SGBD, normalment fem referència a la comparativa entre les velocitats de resposta, o al nombre de transaccions per segon que és capaç d'executar cada programari.

Malgrat l'objectiu inicial d'aquest apartat de benchmarks no era fer un joc de proves per passar a cronometrar els temps de resposta, la nul·la informació de qualitat trobada en els diferents mitjans consultats, ha fet que l'opció de fer-ho amb detall es considerés d'interès, ja que:

- Les comparatives trobades no especificaven les relacions utilitzades, ni el volum de dades carregades a la base de dades, ni tant sols la utilització o no d'índexs.
- Les comparatives habitualment són entre MySQL amb el model ISAM (no ACID), i el PostgreSQL. Aquest fet invalida qualsevol resultat, ja que la primera no fa les comprovacions d'integritat referencial, ni molts dels blocats necessaris de registres. La comparativa de MySQL amb InnoDB (si ACID) i el PostgreSQL són inexistents!

Optar per fer les proves, s'ha demostrat molt encertat en revisar els resultats aconseguits.

12.1.- Metodologia

Maquinari utilitzat

Equip principal on s'han executat les proves:
Ordinador Dell Latitud D505:

- Pentium M de 1,7Ghz
- 768Mb de memòria RAM
- Disc dur 5400 rpm (22ms i 16Mb cache)

Equip secundari de validació de les proves:
Ordinador Clònic:

- Pentium 4 de 2,4Ghz
- 512Mb de memòria RAM
- Disc dur 7200 rpm (9ms i 2Mb cache)

Programari

- Sistema operatiu: Windows XP SP2, amb tots els updates fins a la data d'execució (14/10/05).
    - Serveis en execució supervisats (mínims).
    - Firewall desactivat.
    - Optimitzat per a prioritat d'aplicacions en primer pla.
    - AVG Antivirus instal·lat i actiu.
- PostgreSQL 8.03
- MySQL 4.1.13 amb InnoDB
- MySQL 4.1.13 amb MyISAM
- Client de SQL utilitzat per a fer totes les proves: SQuirreL (per a fer les carregues de dades al PostgreSQL ha calgut utilitzar el seu client psql).

NOTA: Les proves del MySQL InnoDB i MyISAM són totalment independents, tal com si fossin dos SGBD diferents.

Dades

Per a realitzar els benchmarks s'han carregat les taules amb un conjunt de dades sintètiques (facilitades a l'assignatura de SGBD[1] – veure annex IV), creades amb l'objectiu d'executar proves d'optimització de consultes amb l'Oracle.

Els dos fitxers de text pla que contenien les esmentades dades, tenien un format de 'punt i coma' delimited, amb les mateixes tancades entre cometes. Per a fer la càrrega s'han transformat a l'estàndard CSV (s'han separat amb el caràcter 'coma' i s'han eliminat les cometes).

Els dos fitxers que s'han carregat (clients.txt i trajectes.txt), tenen una mida de 8.862 i 44.323 bytes respectivament. El primer té un total de 59.458 files, i el segon un total de 459.267 files.

Descripció Taules

Carrega

Descripció: Utilitzada per la càrrega inicial (no s'utilitza a les consultes). Amb duplicats i no normalitzada.
Num Registres: 59.458

Carrega_Trajectes

Descripció: Utilitzada per la càrrega inicial (no s'utilitza a les consultes). Amb duplicats i no normalitzada.
Num Registres: 459.267

Client

Descripció: Conté els diferents NIF, nom, cognoms i adreça dels clients.
Clau Primària: DNI
Num Registres: 33.582

Cotxe

Descripció: Conté la matricula del cotxe, i la seva marca i model.
Clau Primària: Matricula
Num Registres: 35.497

Peatge

Descripció: Conté el nom del peatge, i la autopista a la que pertany.
Num Registres: 37

Teletac

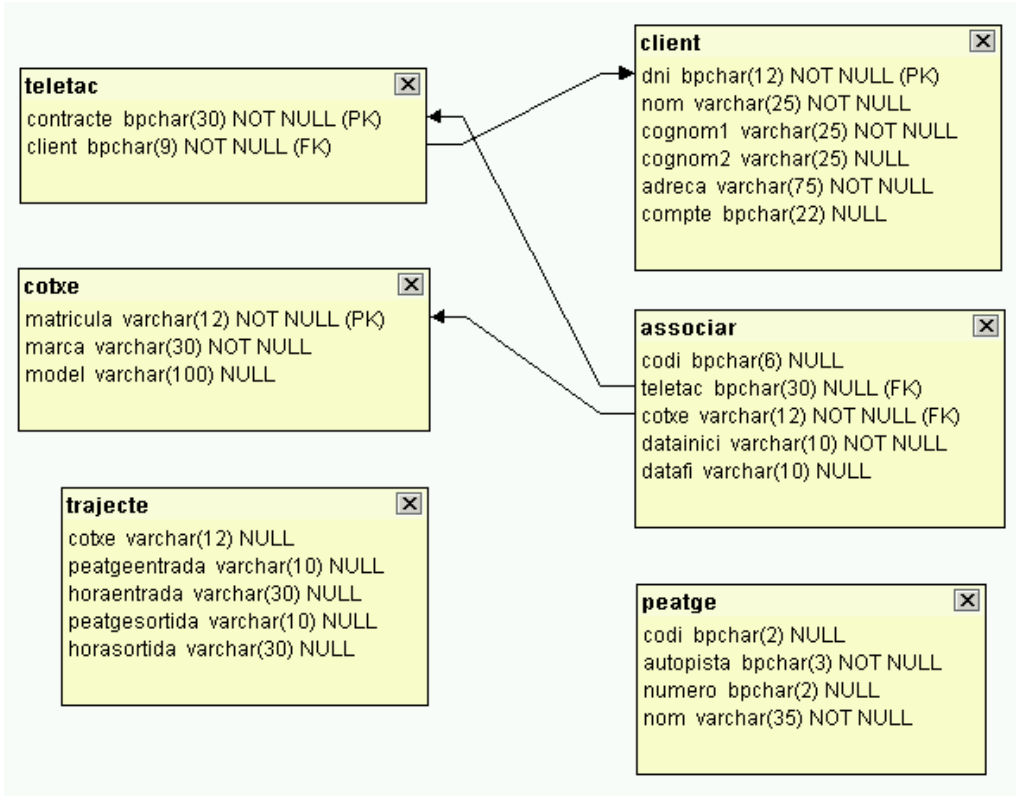
Descripció: Té el num de NIF i el codi del Teletac de cada client (NIF duplicats)
Clau Primària: Codi Teletac.
Clau Forana: client(DNI)
Num Registres: 41.866

Associar

Descripció: Relaciona (varis a varis) el Teletac, amb els cotxes que el poden utilitzar.
Clau Primaria: Contrate.
Clau Forana: client(DNI)
Num Registres: 59.458

[1] SGBD: Sistemes de Gestió de Bases de Dades, de la Enginyeria Informàtica de la Universitat Oberta de Catalunya.

Trajecte
Descripció: Històric dels recorreguts que han fet els vehicles per les autopistes controlades.
Num Registres: 459.267



Relacions de les diferents taules
(gràfic generat amb l'editor SQL SQuirreL)

Índexs addicionals
No s'han definit índexs addicionals.
Considerant que les consultes que es faran al sistema de proves són ad hoc, s'ha omès expressament l'índex a trajectes (per l'identificador del cotxe). Amb aquesta omissió (força important), es pretén comprovar si les optimitzacions que porta a terme l'SGBD són eficients. Aquesta comprovació es vital, ja que no és viable que tots els camps de una BD estiguin indexats, ni és possible preveure tots els tipus d'accés que faran els aplicatius a la BD (cadascun d'ells tindrà unes necessitats de recuperació d'informació diferent).

12.2.- Execució de les proves

Seguint una estructura de dades similar a la Pràctica de l'assignatura SGBD (primavera 2005), s'ha procedit a carregar les dades sintètiques en dues taules temporals.

No s'han creat disparadors amb la finalitat d'augmentar la compatibilitat de les proves entre els diferents SGBD, i per aconseguir que les diferències de rendiment fossin 'purament transaccionals'.

Des de les taules inicials de càrrega, s'ha fet mitjançant unes consultes les insercions múltiples a les taules de treball (s'ha anotat el temps d'aquestes operacions). Tot

seguit s'ha comprovat el nombre de registres de cada taula destinatària de les dades, per assegurar que s'havien carregat correctament.

El pas següent ha estat el d'executar les consultes, tot cronometrant el temps que cadascuna trigava. Addicionalment s'ha demanat el detall del pla d'execució que han utilitzat.

Desprès de realitzar totes les consultes s'ha executat el 'ANALYZE' segons la sintaxi de cada SGBD, per a passar a repetir les proves i comprovar així si s'havien triat diferents plans d'execució. En cada cas s'ha anotat el nou temps d'execució i els seus plans d'accés, tot comprovant les diferències amb els resultats inicials.

L'últim pas ha consistit en instal·lar tot el programari en l'ordinador secundari de validació, per passar a repetir tots el processos.

Estructura de les consultes

Consulta A
Consulta senzilla que retorna el resultat del contingut de quatre camps (cadascun d'una taula diferent). Fa la combinació de les quatre taules.
No s'utilitza la taula trajecte. És una primera prova de rendiment.

Consulta B
Similar a l'anterior, però retornant cinc columnes de cinc taules. En aquest cas cal fer la combinació amb la taula trajecte.
S'intenta observar amb aquesta consulta l'habilitat de l'optimitzador per a fer la unió de les taules segons un ordre lògic, el de la seva mida (totes tenen almenys un índex).

Consulta C
Similar a l'anterior (cinc columnes de cinc taules), afegint el filtratge pel DNI de la taula client (és una clau primària).
Si l'optimitzador funciona adequadament, hauria d'utilitzar l'índex que té creat.

Consulta D
És idèntic a l'anterior, però canviat la condició final. En lloc de cercar pel DNI sencer, es vol que retorni tots els que tenen la lletra N, forçant així a fer un recorregut seqüencial de la taula (full scan). Es vol comprovar si es fa aquesta operació en últim lloc.

Consulta E
Idèntica a l'anterior, però afegint un ordre en la representació dels resultats de sortida. Es vol comprovar si aquest afegit fa canviar el pla d'execució, i veure si hi ha sobre-càrrega per haver de tenir tots els resultats per a poder fer l'ordenació, o es fa l'ordenació en primer lloc, per desprès realitzar la resta de la consulta.

Consulta F
Similar a la C, però fent el filtrat per un camp que no té índex.

Consulta G
Similar a la C, però filtrat pel camp peatgeentrada de la taula trajecte. S'ha de notar que la taula trajecte no té cap índex creat, i que té un numero de files elevat.

12.3.- Comparativa de resultats

Temps d'execució amb ordinador Dell D505 (en segons)

Operació	MySQL InnoDB	MySQL MyISAM	PostgreSQL
Carregar Carrega	2,4	3	7,8
Carregar Carrega_Trajectes	23,2	18,9	37,5
Poblar Clients	2	2,5	4
Poblar Cotxes	1,7	1,7	3,8
Poblar Peatge	3,5	3,4	20,9
Poblar Teletac	24,1	15,8	6,6
Poblar Associar	53,6	38,6	10,2
Poblar Trajecte	20,1	21,1	19,8
Consulta A	6,9[1]	6,5[1]	4
Consulta B	3,4[1]	4,3[1]	44,4
Consulta C	3,6	3,9	4,4
Consulta D	34,6	35	24,9
Consulta E	**> 1800**[2]	**> 1800**[2]	22,7
Consulta F	**> 1800**[2]	**> 1800**[2]	23,7
Consulta G	359,5	358	6,4

[1] comptat el temps que ha trigat en treure els primers resultats (desprès es quedava el client fent el *fetch* de les restants files durant varis minuts).
[2] prova aturada per considerar excés de temps

Temps d'execució amb ordinador Pentium 4 Clònic (en segons)

Operació	MySQL InnoDB	MySQL MyISAM	PostgreSQL
Carregar Carrega	3,2	9,5	5,4
Carregar Carrega_Trajectes	17,9	17,6	31,8
Poblar Clients	3,1	5,5	6,6
Poblar Cotxes	2,2	2,1	3,9
Poblar Peatge	4,3	3,6	24,4
Poblar Teletac	12,3	13,5	20,4
Poblar Associar	15,8	15,4	15,9
Poblar Trajecte	11,3	11,4	18
Consulta A	2,3[1]	1,6[1]	6
Consulta B	1,9[1]	1,5[1]	85,4
Consulta C	4,5	3,9	14,4
Consulta D	6,5	3,4	29,3
Consulta E	325,25	333,1	24,1
Consulta F	270,1	363	24,2
Consulta G	28,9	67	8

[1] comptat el temps que ha trigat en treure els primers resultats

12.4.- Comparativa desprès de l'ANALYZE

L'única consulta que ha canviat el pla d'execució desprès de fer que els SGBD actualitzessin les estadístiques de les taules, ha estat la Consulta F del PostgreSQL. Aquesta ha passat de trigar 23,7 segons a només 2,4 segons a l'ordinador Dell D505, i de 24,2 a només 2 segons en l'ordinador clònic.

12.5.- Plans d'execució

S'inclouen a l'annex II.
S'ha de notar en els plans d'execució, que PostgreSQL utilitza un repertori més ampli de possibilitats de processament en les consultes.

En el MySQL, la consulta D surt greument perjudicada per una errada d'anàlisi, ja que en lloc d'ordenar cotxes per matricula (clau primària), realitza totes les operacions de combinació, per desprès ordenar els resultats. Així, en lloc de tenir els primers valors tal com es va executant la consulta, retorna els primers valors només quan ho acaba d'executar tot.

12.6.- Conclusions

L'execució dels jocs de prova es confirmen com a correctes, en haver una relació directa entre els resultats obtinguts amb el Dell D505 i l'ordinador P4 clònic.

Desprès d'executar el ANALYZE (o equivalent) a cada base de dades, s'observa que només el PostgreSQL té en compte les noves estadístiques per a modificar el pla d'accés de les consultes (només canvia el pla d'accés a la consulta F).

També s'observa que l'optimitzador de consultes del MySQL no té en compte la utilització del mòdul MyISAM o del InnoDB, ja que el resultat dels plans d'execució coincideixen en aquests dos modes.

Proves addicionals que caldrien fer

Caldria repetir els benchmark en un ordinador amb un sistema operatiu diferent;

- Es creu que podria ser significatiu repetir els passos d'aquest estudi, tenint instal·lats els SGBD en un ordinador funcionant sota Linux (per exemple).

Caldria tornar a fer les proves amb un ordinador multiprocessador:

- La paral·lerització de les diferents tasques que s'han de fer per a arribar a executar una consulta pot ser crític en un sistema 'real'. Els resultats d'una consulta es poden veure influenciats per la implementació del programa en el tractament dels fils d'execució que caldria crear. Aquest punt encara és més crític en haver-se d'executar els processos de diversos clients congruentment.

Caldria valorar com afecta al rendiment la utilització del client triat (SQuirreL), ja que al estar implementat en java, i utilitzar per la connexió el JDBC, podria influenciar de forma inadvertida en els resultats.

S'hauria d'implementar algun test addicionals per a valorar la velocitat d'invocació i execució dels disparadors i del procediments emmagatzemats.

També caldria repetir les proves un nombre més elevat de vegades, per a que els resultats finals fossin la mitjana dels valors obtinguts en les diferents execucions.

Com a conclusió final, i sense que aquests benchmarks es considerin definitius a falta de les proves addicionals necessàries anotades, dir que PostgreSQL sembla més ràpid en executar consultes complexes, donat que el seu analitzador sap aprofitar millor l'estructura de les taules i la mida d'aquestes, tot aconseguint prioritzar la utilització dels índexs, i la utilització de les taules petites, per aconseguir cardinalitats menors en fer combinacions.

Finalment i considerant que cap consulta feta amb el PostreSQL ha trigat més d'un minut en ser resolta, i que algunes consultes del MySQL (tant en MyISAM com en InnoDB) han trigat hores, no recomanaria MySQL de poder-se fer consultes ad hoc a l'SGBD, ni en entorns en que tingui que coexistir múltiple programari, utilitzant unes taules base genèriques, no optimitzades per a cadascun dels possibles aplicatius. Malgrat això podria ser una opció a considerar en sistemes senzills de tipus LAMP.

Conclusions addicionals

Durant la redacció final del present informe, tot contrastant les dades, s'ha observat que el temps d'execució d'algunes de les consultes variaven considerablement, segons el moment en que s'havia realitzat la consulta, o fins i tot segons la consulta anterior que s'havia fet.

En principi s'havia pensat en la possible influencia de l'editor SQL SQuirreL, però s'havia descartat, donat que en fer només de back-end, no havia d'influir en el funcionament del motor de la base de dades.

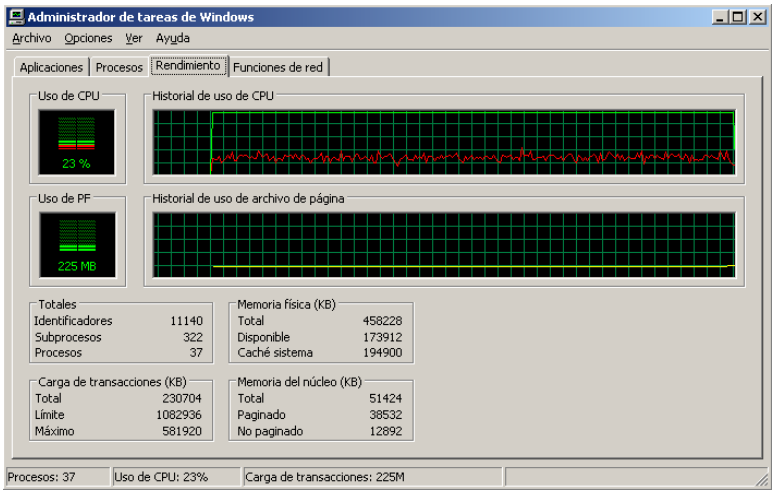
Fent un anàlisis més detallat dels possibles problemes que estava patint el model de proves, s'ha arribat a la conclusió que en recuperar consultes grans l'SQuirreL acaparava la memòria de l'ordinador (donat que el java no feia el oportú garbagge collection quan era degut), tot estrangulant l'espai de treball disponible pel motor de la base de dades, amb la conseqüent degradació aleatòria del rendiment.
Finalment s'ha optat per tornar a repetir totes les proves des dels clients propis de cada BD.

Malgrat tot, el SQuirreL ha estat una eina immillorable com a client, ja que treballar amb un únic editor amb diferents BD ha estat molt còmode. També s'ha de considerar que normalment, i per omissió, només retorna els 100 primers registres, cosa que faria (en un ús normal), que els problemes detectats no succeïssin.

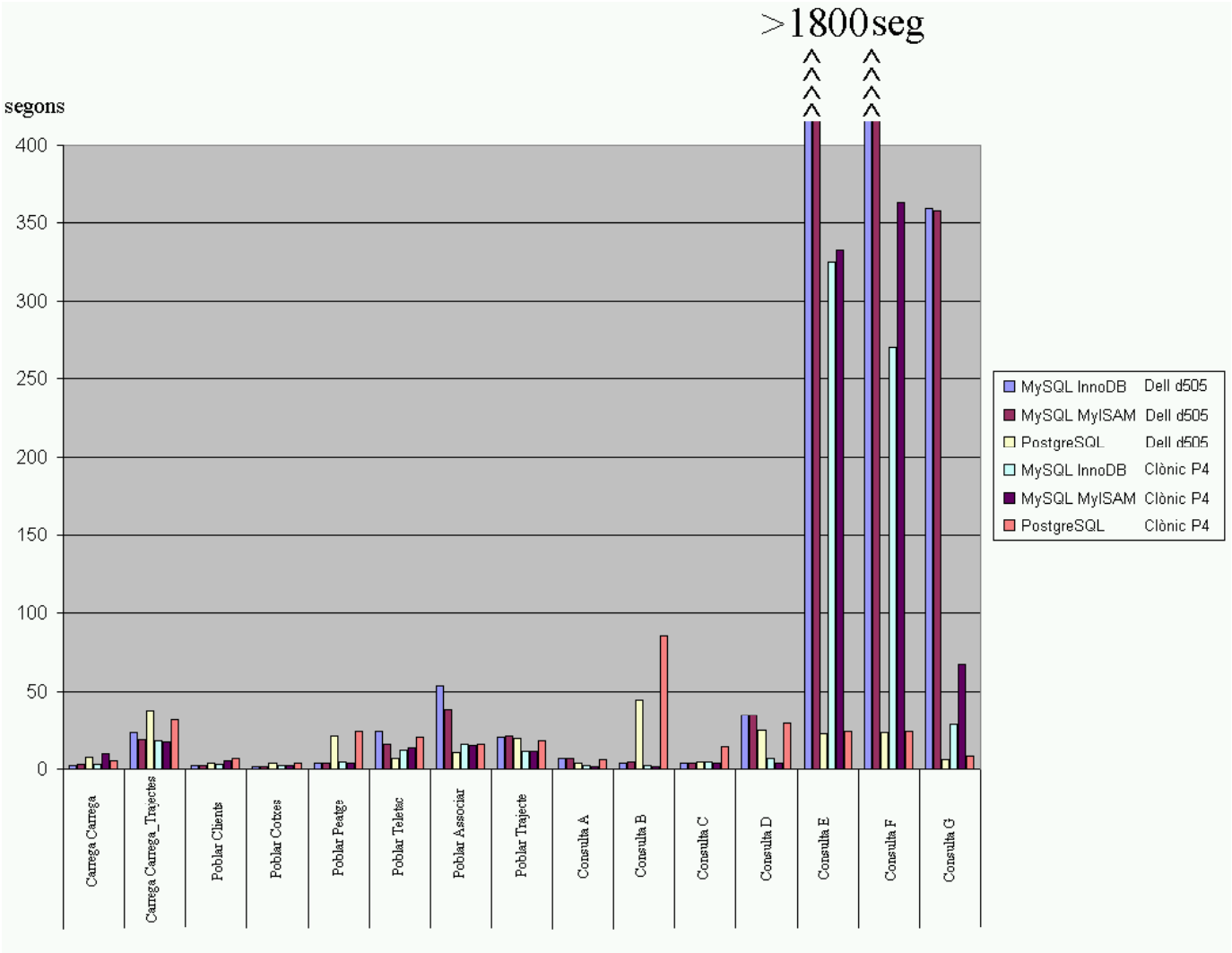
S'hauria de notificar com a possible error als desenvolupadors del MySQL, les diferències que s'observen en les consultes E i F en els diferents ordinadors. Durant l'execució de les mateixes en el P4 s'ha notat una mala gestió del ús del processador, ja que tot l'ordinador es congelava durant uns segons cada poc temps.

En el ordinador Dell (processador Pentium M), el ús del processador es mantenia al 100% però no produïa aquest efecte de 'pèrdua d'interrupcions' (i de pèrdua del control del punter del ratolí).

La gràfica d'ús de la CPU del P4 durant l'execució d'aquestes consultes ha estat la següent (important notar la línia vermella –cronologia del nucli – que sol correspondre al temps de processador ocupat per les transferències del bus IDE del disc dur).



12.7.- Gràfica comparativa dels temps d'execució.



S'ha d'observar que els resultats obtinguts varien de forma important segons a l'ordinador en que s'han executat els test. Així, els resultats del PostgreSQL amb el Dell

d505 són força millors que amb el P4 (possiblement degut a la major quantitat de memòria cau del seu disc dur, i al menor nombre de vies del processador – que fa que es tinguin que retrocedir menys instruccions a mig processar), mentre que amb el MySQL, les execucions amb el clònic P4 són clarament més ràpides (possiblement, al executar menys concurrentment les consultes, aprofiti millor l'alta freqüència i paral·lelisme de la seva CPU escalar).

13.- Manteniment de la base de dades

13.1.- Backup

Els fitxers on PostgreSQL emmagatzema la BD, són fitxers físics en disc des del punt de vista del sistema operatiu, pel que a mesura que s'incorpora informació, aquests van augmentant de mida.

Juntament amb la informació útil que es guarda en els fitxers de la BD, s'emmagatzemen dades relatives a l'ordre de la mateixa, al tipus de dades, i altres necessàries per a l'accés i al correcte funcionament de l'SGBD. Això fa que de forma estimada, els esmentats fitxers ocupin almenys el doble d'espai que les dades que emmagatzemen.

Per assegurar la durabilitat de totes aquestes dades, cal que l'administrador de la BD faci còpies de seguretat periòdiques, ja que malgrat PostgreSQL sigui molt estable, i fins i tot els discs durs tinguin redundància, res permetria recuperar les dades en cas de robatori físic de l'equip, d'incendi, o d'altres accidents.

Les còpies de seguretat 'totals' es poden fer d'almenys tres formes:

a. Còpia dels fitxers de la BD en fred.
b. Bolcat des de rutines pròpies.
c. Còpia en calent.

Per a fer una còpia en fred, s'ha d'aturar l'SGBD i salvaguardar els fitxers on PostgreSQL emmagatzema les taules i la resta d'informació. Aquest mètode és poc recomanable, ja que implica no poder utilitzar el sistema durant el temps en que s'executa aquest procés. La restauració posteriorment també pot ser complicada, ja que només es podrà recuperar la còpia de forma total, no essent possible fer recuperacions parcials.

L'opció d'utilitzar rutines pròpies per a fer el bolcat de les dades tampoc és gaire recomanable, ja que pot passar que no es tinguin permisos d'accés a totes les taules, o que en fer el bolcat, les dades resultants no tinguin integritat referencial, per haver accedit a les taules de forma seqüencial.

L'opció de fer una còpia de seguretat en calent, amb la utilitat pg_dump que el mateix PostgreSQL incorpora, sembla la més òptima. Aquest programa serveix per a fer un bolcat total del contingut de la BD pel canal estàndard de sortida del sistema operatiu. De fer una canalització a un fitxer, aquest desprès es podrà tornar a utilitzar per a fer la càrrega de dades. De tenir en funcionament dues bases de dades, també es podran utilitzar canalitzacions *pipe-out* i *pipe-in*, per a fer un traspàs directe de la informació entre aquestes. Aquesta última opció és especialment útil en fer migracions de versió, o en haver de substituir el maquinari, ja que els SGBD poden estar executant-se en diferents ordinadors.

Exemple de backup:
En primer lloc localitzarem l'executable pg_dump o pg_dumpall (en sistemes Windows) en el directori ..\bin d'allí on està instal·lat l'SGBD.

Tot seguit es pot utilitzar la instrucció amb els paràmetres;

```
pg_dumpall --host=\\nomServidor --port=5432 --username=postgres
--password >backup.txt
```

Les clàusules host i port són opcionals, si s'executa des del mateix ordinador on resideix la base de dades i si el port de comunicacions és l'estàndard.

En executar el bolcat, se'ns demanarà la contrasenya de l'usuari postgres.

El fitxer backup.txt contindrà el bolcat de les dades, i totes les instruccions necessàries per a refer les taules, procediments emmagatzemats, usuaris i tots aquells paràmetres necessaris per a aconseguir que es pugui reconstruir la BD original, en restaurar aquest fitxer sobre una BD buida.

A grans trets, l'ordre i contingut del fitxer backup.txt és (per blocs):

```
-- PostgreSQL database cluster dump
Connexió a la tablespace de la que es fa el bolcat.

-- Users
Instruccions de creació dels usuaris existents.

-- Database creation
Instruccions per la creació de la BD.

-- Users
Paràmetres addicionals dels usuaris de la BD.

-- PostgreSQL database dump
Paràmetres de la BD (valors dels 'SET').

-- Name: DATABASE [nomTablespace]; Type: COMMENT; Schema: -;
Owner:
Comentaris DATABASE

-- Name: SCHEMA public; Type: COMMENT; Schema: -; Owner:
[nomUsuari]
Comentaris primer SCHEMA existent.

-- Name: [nomFuncio](); Type: FUNCTION; Schema: public; Owner:
[nomUsuari]
Codi plpgsql de les funcions existents.

-- Name: [nomLlenguatge]; Type: PROCEDURAL LANGUAGE; Schema:
public; Owner:
Instruccions de creació dels llenguatges instal·lats.

-- Name: [nomTipus]; Type: TYPE; Schema: public; Owner:
[nomUsuari]
Creació dels tipus.

-- Name: auditoria; Type: TABLE; Schema: public; Owner:
[nomUsuari]; Tablespace:
Creació de taules.
```

```
-- Data for Name: test; Type: TABLE DATA; Schema: public; Owner: [nomUsuari]
Instruccions del bolcat de les dades de la taula.

-- Name: [nomVista]; Type: VIEW; Schema: public; Owner: [nomUsuari]
Creació de vistes.

-- Data for Name: [nomTaula]; Type: TABLE DATA; Schema: public; Owner: [nomUsuari]
Instrucció de bolcat de dades de les taules, i les dades d'aquestes.

-- Name: [nomResticcio]; Type: (PK/FK) CONSTRAINT; Schema: public; Owner: [nomUsuari]; Tablespace:
Modificació de taules per afegir les CONSTRAINT (claus primàries, índexs, etc).

-- Name: [nomDisparador]; Type: TRIGGER; Schema: public; Owner: [nomUsuari]
Definició de triggers

-- Name: public; Type: ACL; Schema: -; Owner: [nomUsuari]
Assignació de privilegis d'accés dels usuaris sobre les dades.

-- PostgreSQL database dump complete
Indicació de fi de bolcat de la BD.
```

L'avantatge d'aquest mètode de còpia de seguretat, és que es pot editar el fitxer de còpia per a cercar informació concreta. En aquest trobarem tant les dades contingudes en les taules, com les sentències de creació d'aquestes (amb les claus primàries i foranes), i fins i tot el codi plpgsql de les funcions i disparadors.

Exemple de restore:
El contingut d'una còpia de seguretat es pot restaurar total o parcialment.
En el cas de voler una restauració parcial, haurem d'editar el fitxer on s'ha emmagatzemat la còpia, i traspassar el que volem editar a un nou fitxer.
Utilitzarem aquest últim per a fer una restauració parcial.

La importació es realitzarà utilitzant una *pipe* del mateix sistema operatiu:

```
PGSQL NomBaseDades < backup.txt
```

Per a restaurar tot contingut d'una BD, primer l'esborraríem per assegurar no quedessin dades, desprès la crearíem, per finalitzar amb la restauració pròpiament dita.
Al igual que a l'exemple anterior, utilitzarem una *pipe* del sistema operatiu per a realitzar la importació de les dades:

```
DROPDB NomBaseDades
CREATEDB -E LATIN1 NomBaseDades
PGSQL NomBaseDades < backup.txt
```

13.2.- Processos periòdics

Tal com s'ha exposat al llarg del present treball, l'SGBD PostgreSQL gairebé no requereix de manteniment. No obstant, per assegurar el seu bon rendiment, caldrà executar periòdicament els processos següents:

- Eliminació dels registres marcats com a obsolets (generats degut a l'ús del MVCC).
- Regeneració dels cluster creats.
- Regeneració de la informació estadística de les taules.

Així, caldrà automatitzar l'execució de la seqüència de comandes següent:

```
VACUUM FULL;
CLUSTER;
ANALYZE;
```

De incloure aquestes instruccions en un fitxer (amb nom sql.txt per exemple), es podria automatitzar l'execució d'un script des del sistema operatiu de l'ordinador, per a que executes la línia:

```
..\rutaBinDelPostgreSQL\psql -U postgres -d UOC -f sql.txt  <password.txt
```

On *plsql* és la instrucció que permet executar les comandes, UOC és la base de dades, postgres és el nom d'usuari que ha d'executar les instruccions que hi han al fitxer sql.txt, i considerant que a l'interior del fitxer password.txt es troba la contrasenya de l'esmentat usuari.

13.3.- Espai d'emmagatzemament

A diferència d'altres SGBD, PostgreSQL no reserva espai pels fitxers on emmagatzema les dades. L'agrupació d'aquest fitxers formen el que s'anomena Cluster (no s'ha de confondre amb el cluster d'agrupació de dades de les taules), que és l'equivalent a les tablespaces de l'Oracle, o als dbspace del Informix.

Pel fet de que cada taula és físicament un o més fitxers, no cal supervisar l'ocupació d'aquestes zones (no existeix la possibilitat d'exhaurir l'espai intern del cluster), ja que els fitxers augmentaran de mida a mesura que s'insereixin dades.

Malgrat aquesta diferència de concepte amb altres SGBD, existeix la possibilitat d'indicar la ubicació física dels fitxers, a efecte d'optimitzar la concurrència en l'accés als disc on s'emmagatzemen les dades.

14.- Conclusions i futurs treballs

Al llarg del treball, malgrat no s'ha pogut reflectir totes les proves fetes, s'observa que de fer comparatives entre PostgreSQL i altres SGBD, MySQL s'ha de descartar per les seves poques possibilitats. Es comprova que PostgreSQL té almenys (si no les sobrepassa) les característiques de l'Informix, i que s'apropa a les funcionalitats que pot tenir Oracle.

PostgreSQL és pot instal·lar de forma molt senzilla en els sistemes operatius més actuals (des del Linux, al Windows XP, passant pel Windows 95), i no presenta incompatibilitats amb altres programaris d'ús corrent.

PostgreSQL és un SGBD complert, madur, escalable, amb un potent llenguatge nadiu (pl/pgSQL), amb llicencia BSD i amb un suport tècnic que es pot aconseguir des de la mateixa comunitat del programari lliure, o mitjançant empreses especialitzades en aquest SGBD.

El punt d'inflexió en la utilització de PostgreSQL en l'entorn empresarial ja s'ha sobrepassat tal com es demostra en consultar a www.postgresql.org les histories de projectes que utilitzen aquest programari, i a la mateixa web en general, al cercar qualsevol tipus d'informació. La noticia de que Sun Microsystems ha integrat PostgreSQL en el Solaris 10, i que donarà suport tècnic complert a aquest, significa que aquest programari encara tindrà més pes en l'entorn empresarial futur, i conseqüentment, hi haurà més necessitat d'especialistes qualificats.

De realitzar-se futurs Projectes de Fi de Carrera seguint la línia del present, es podria optar per:

a. Obrir un projecte per la millora dels índexs *hash*, ja que segons s'ha exposat, s'han de millorar. Aquest projecte podria ser un estudi teòric del funcionament actual (seguint fins i tot la codificació de la seva implementació), amb un informe final detallat amb les propostes per la millora dels mateixos, o fins i tot realitzant les reformes necessàries a l'algorisme.
b. Fer un estudi dels requisits que hauria de tenir un Live CD de PostgreSQL per a que s'adaptés a les necessitats reals de les assignatures de l'àrea de Bases de Dades, i crear l'esmentat Live CD (imprescindible persistència de dades entre les sessions). Es podria complimentar amb la posta en marxa d'un servidor web (amb un front-end php), que deixés (prèvia autenticació) executar comandes SQL, per a arribar a fer fins i tot el lliurament de pacs/pràctiques amb aquest mecanisme, o senzillament perquè servís per a practicar amb el PostgreSQL sense necessitat de fer una instal·lació local.
c. Fer un estudi paral·lel al present, tot justificant la possibilitat d'utilitzar altres SGBD (Interbase, Ingres, etc) per a portar a terme les tasques docents en entorns universitaris.

Referències

Llibres:

Douglas, Korry & Douglas, Susan. **PostgreSQL, a comprensive guide to building, programming, and administering PostgreSQL databases**. Editorial Sams Publishing (First Edition – February 2003)

Momjian, Bruce. **PostgreSQL Introduction and Concepts**. Editorial Addison-Wesley (Frist Printing. November 2000).

Worslley, John. **Practical PostgreSQL**. Editorial O'Relly (First Edition 2001)

PostgreSQL Global Development Group. **PostgreSQL 8.0.0 Documentation**. Editat per PostgreSQL Global Development Group (edició electrònica - 2005).

Gonzales, Jesús & Seoane, Joaquin & Robles, Gregorio. **Introducción al Software Libre**. Fundació per a la Universitat Oberta de Catalunya (Primera edició 2003)

Altres recursos:

Stallman, Richard. **El Proyecto GNU**:
http://www.gnu.org/gnu/thegnuproject.es.html

Raymond, Eric S. **La Catedral i el Bazar**:
http://www.sindominio.net/biblioweb/telematica/catedral.html

Mas, Jordi. **Programari lliure en el sector públic**. UOC, 2003:
http://www.uoc.edu/dt/20325/index.html

Altres recursos diversos d'interès:
http://www.postgresql.org/about/
http://en.wikipedia.org/wiki/Postgresql
http://en.wikipedia.org/wiki/Object-SQL_Impedance_Mismatch
http://www.sobl.org/traducciones/practical-postgres/

Novetats:

Sun MicroSystems integra PostgreSQL en el sistema operatiu Solaris 10:
http://www.sun.com/smi/Press/sunflash/2005-11/sunflash.20051117.1.html
http://www.sun.com/software/solaris/postgres.jsp

Glossari

ACID: Acrònim fet amb les inicials de les característiques que ha de complir una base de dades per a ser 'funcional'; Atomicitat en les transaccions, Consistència en les operacions, Isolació entre tasques, Durabilitat de les dades.

ARO: Army Research Office

BD: Base de Dades.

Benchmarks: Comparatives de rendiments entre dos o més productes, basades en l'execució del mateixos jocs de proves i cronometrat dels temps corresponents.

BSD: Inicials de Berkeley Software Distribution. El programari que s'acull a aquest tipus de llicència és denominat també 'software lliure', ja que en aquest model de distribució de software s'especifica que el codi font s'ha de distribuir lliurement amb l'executable.

CSV: Coma Separated Values. Acrònim d'un tipus d'organització (simple) de dades en fitxers de text. S'utilitzava aquest format com estàndard de facto per a exportar/importar dades.

DARPA: Defense Advanced Research Projects Agency

Dimoni: S'anomena així als programes de baix nivell que s'executen concurrentment amb el sistema operatiu. Poden ser programes que facin tasques de manteniment del sistema, o ser altres serveis que han d'assegurar una execució cada cert període de temps (servidors ftp, http, un SGBD, etc).

OO: Paradigma de la programació Orientada a l'Objecte.

LAMP: Acrònim descriptiu dels Sistemes informàtics lleugers (utilitzats habitualment per a fer pàgines d'internet dinàmiques), format per les inicials de Linux, Apache, MySQL i PHP, Perl o Python..

LAMP: Versió alternativa de l'acrònim (posterior a l'inicial). Aquesta definició és tendenciosa, ja que s'utilitzen les inicials de Linux, Apache, 'Middleware' i PostgreSQL.

Mainframe: Aquesta designació la reben els ordinadors centrals de grans empreses (típicament un banc), ja que són físicament grans, tenen uns rendiments que permeten suportar un nombre molt elevat de transaccions simultàniament (procedents de milers de terminals), a la vegada que són molt cars, i que requereixen de condicions especials de funcionament (climatització, etc).

MVCC: Multi-Version Concurrency Control. Mecanisme amb el qual la base de dades utilitza diverses versions dels registres, per a augmentar la concurrència, i millorar el funcionament.

NSF: National Science Foundation

NESL: Navy Electronic System Command

PDP11: Ordinador de 16 bits de Digital Equipament Corporation (DEC). Va ser una dels primers ordinadors amb el que van poder comptar els laboratoris i investigadors a l'època dels 70, tant per cost, com per disponibilitat i manteniment.

SGBD: Sistema de Gestió de Bases de Dades.

XML: És un acrònim anglès que prové de les paraules eXtensible Markup Language. És un format de dades etiquetades.

Annex I. Exemple índex calculat complex.

(es planteja l'exemple com un cas real)

Els mateixos propietaris de la xarxa d'autopistes que ens van fer optimitzar l'accés a les dades dels usuaris dels Teletac (enunciat pràctica SGBD segons annex IV i Benchmarks d'annex II), ara hauran d'accedir freqüentment a les durades dels trajectes, donat que per a complir amb la nova llei que fa que s'hagi de notificar els vehicles que han realitzat un trajecte superant la velocitat màxima permesa, s'haurà d'accedir a la taula trajecte per la durada d'aquest (camp inexistent).

L'objectiu és optimitzar l'accés a la taula trajecte, segons el temps que ha trigat l'usuari del Teletac en fer-lo. Un refinament posterior de la consulta (que no es realitzarà), agruparia els trajectes segons els peatges d'entrada/sortida, per a determinar la distancia entre ells, i poder determinar així la velocitat real.

Que és demana:
Plantejar l'optimització sense modificar l'estructura de les taules, o sigui, treballant únicament amb les possibilitats d'índex que ofereix l'SGBD PostgreSQL.

Solució:
S'observa que la taula trajecte té els camps horaentrada i horasortida. Aquests, per 'problemes' històrics són de tipus char, pel que s'haurien de convertir els camps a timestamp, o alternativament fer el tractament per a transformar-los mentre s'accedeix a ells.

Es proposaria de crear un índex calculat que donés com a resultat la resta entre l'hora de sortida i l'hora d'entrada. Amb això, tindríem els segons que s'ha trigat en fer el trajecte en qüestió.

L'operació de la resta de l'hora de sortida amb l'hora d'entrada la farem en una funció per a poder indicar que el resultat d'aquesta és de tipus IMMUTABLE. Aquesta clàusula és imprescindible per a informar a l'optimitzador que el resultat que retorna l'expressió sempre és la mateixa (per uns valors donats), i que un cop calculat el valor, el pot utilitzar com a índex.

```
CREATE OR REPLACE FUNCTION TempsTrajecte (trajecte.horasortida%TYPE,
  trajecte.horaentrada%TYPE) RETURNS INT8 AS '
BEGIN
RETURN
  (EXTRACT(EPOCH FROM to_timestamp($1, ''DD/MM/YYYY HH:MI:SS'')) -
  EXTRACT(EPOCH FROM to_timestamp($2, ''DD/MM/YYYY HH:MI:SS'')));
END;
' LANGUAGE 'plpgsql' STRICT IMMUTABLE;
```

Els paràmetres d'entrada són del mateix tipus que els camps horaentrada i horasortida de la taula trajecte. El valor de sortida serà de tipus enter llarg.
La funció fa una conversió de tipus char a timestamp de les hores d'entrada i sortida, i fa la resta entre elles. El valor resultant és el que es retorna.

Una vegada creada la funció, es pot passar a crear l'índex calculat:

```
CREATE INDEX idx_TempsTrajecte
      ON trajecte USING btree (TempsTrajecte(trajecte.horasortida,
         trajecte.horaentrada));
```

A partir d'aquest moment, quan fem una consulta tot específicant com a condició la durada del trajecte, s'utilitzarà l'índex creat. En el cas de fer l'explain de la següent consulta:

```
SELECT *
      FROM trajecte
      WHERE TempsTrajecte(trajecte.horasortida,
          trajecte.horaentrada) < 100;
```

Se'ns retornarà al pla d'accés:

```
Index Scan using idx_TempsTrajecte on trajecte  (cost=0.00..185.86 rows=46 width=77)
  Index Cond: (TempsTrajecte(horasortida, horaentrada) < 100)
```

En el cas d'haver executat aquesta mateix consulta abans de crear l'índex calculat, ens hagués tornat el següent pla d'execució.

```
Seq Scan on trajecte (cost=0.00..13953.01 rows=153089 width=77)
  Filter: (TempsTrajecte(horasortida, horaentrada) < 100)
```

Malgrat les diferències siguin molt notables i sembli imprescindible crear l'índex proposat, també caldria considerar com pot afectar aquest nou índex a les insercions i modificacions.
Una vegada tinguéssim els resultats d'aquesta última comprovació caldria decidir si crear l'índex, o alternativament, programar la consulta (si és per lliurar per ex. a la DGT una vegada a la setmana) per a que s'executés en hores de baixa activitat de l'SGBD, ni que sigui de forma poc òptima.

Annex II. Detalls execució Benchmarks.

Càrrega de dades

PostgreSQL; creació taules i càrrega de dades.

Creació de les taules inicials de càrrega

```
CREATE TABLE carrega (
      Dni CHAR(12) NOT NULL,
      Nom VARCHAR(25) NOT NULL,
      Cognom1 VARCHAR(25) NOT NULL,
      Cognom2 VARCHAR(25),
      Adreca VARCHAR(75) NOT NULL,
      Compte CHAR(22) NOT NULL,
      Teletac VARCHAR(30) NOT NULL,
      Data VARCHAR(10),
      Data2 VARCHAR(10),
      Matricula VARCHAR(12) NOT NULL,
      Marca VARCHAR(30) NOT NULL,
      Model VARCHAR(100));
```

```
CREATE TABLE carrega_trajectes (
      PeatgeEntrada VARCHAR(10) NOT NULL,
      NomEntrada VARCHAR(30) NOT NULL,
      HoraEntrada varchar(30) NOT NULL,
      PeatgeSortida VARCHAR(10),
      NomSortida VARCHAR(30),
      HoraSortida VARCHAR(30),
      TeleTac VARCHAR(15) NOT NULL,
      Cotxe VARCHAR(12) NOT NULL );
```

Creació de les taules del model de proves

```
CREATE TABLE client (
     DNI CHAR(12) NOT NULL,
     Nom VARCHAR(25) NOT NULL,
     Cognom1 VARCHAR(25) NOT NULL,
     Cognom2 VARCHAR(25),
     Adreca VARCHAR(75) NOT NULL,
     Compte CHAR(22),
     CONSTRAINT pk_client PRIMARY KEY(DNI));
```

```
CREATE TABLE cotxe (
     Matricula VARCHAR(12) NOT NULL,
```

```
      Marca VARCHAR(30) NOT NULL,
      Model VARCHAR(100),
      CONSTRAINT pk_matricula PRIMARY KEY (Matricula));
```

```
CREATE TABLE carrega_trajectes (
      PeatgeEntrada VARCHAR(10) NOT NULL,
      NomEntrada VARCHAR(30) NOT NULL,
      HoraEntrada VARCHAR(30) NOT NULL,
      PeatgeSortida VARCHAR(10),
      NomSortida VARCHAR(30),
      HoraSortida VARCHAR(30),
      TeleTac VARCHAR(15) NOT NULL,
      Cotxe VARCHAR(12) NOT NULL );
```

```
CREATE TABLE Peatge (
      Codi CHAR(2) ,
      Autopista CHAR(3) NOT NULL,
      Numero CHAR(2),
      Nom VARCHAR(35) NOT NULL);
```

```
CREATE TABLE TeleTac (
      Contracte CHAR(30) CONSTRAINT PK_TeleTac PRIMARY KEY,
      Client CHAR(9) NOT NULL CONSTRAINT FK_TeleTac
            REFERENCES Client(DNI));
```

```
CREATE TABLE Associar (
      Codi CHAR(6),
      TeleTac CHAR(30) CONSTRAINT FK_AssociarTeletac
                        REFERENCES TeleTac(Contracte),
      Cotxe VARCHAR(12) CONSTRAINT FK_AssociarCotxe
                        REFERENCES Cotxe(Matricula) NOT NULL,
      DataInici VARCHAR(10) NOT NULL,
      DataFi VARCHAR(10));
```

```
CREATE TABLE Trajecte (
      Cotxe VARCHAR(12),
      PeatgeEntrada VARCHAR(10),
      HoraEntrada VARCHAR(30) ,
      PeatgeSortida VARCHAR(10) ,
      HoraSortida VARCHAR(30));
```

Càrrega física dels fitxers de text en les taules

```
COPY carrega FROM 'c:/uoc/pfc/benchmarks/clients.csv'
```

```
     WITH DELIMITER ';';
```

```
COPY carrega_trajectes FROM 'c:/uoc/pfc/benchmarks/trajectes.csv'
     WITH DELIMITER ';';
```

MySQL InnoDB; creació taules i càrrega de dades.

Creació de les taules inicials de càrrega

```
CREATE TABLE carrega (
     Dni CHAR(12) NOT NULL,
     Nom VARCHAR(25) NOT NULL,
     Cognom1 VARCHAR(25) NOT NULL,
     Cognom2 VARCHAR(25),
     Adreca VARCHAR(75) NOT NULL,
     Compte CHAR(22) NOT NULL,
     Teletac VARCHAR(30) NOT NULL,
     Data VARCHAR(10),
     Data2 VARCHAR(10),
     Matricula VARCHAR(12) NOT NULL,
     Marca VARCHAR(30) NOT NULL,
     Model VARCHAR(100)) ENGINE=InnoDB;
```

```
CREATE TABLE carrega_trajectes (
      PeatgeEntrada VARCHAR(10) NOT NULL,
      NomEntrada VARCHAR(30) NOT NULL,
      HoraEntrada varchar(30) NOT NULL,
      PeatgeSortida VARCHAR(10),
      NomSortida VARCHAR(30),
      HoraSortida VARCHAR(30),
      TeleTac VARCHAR(15) NOT NULL,
      Cotxe VARCHAR(12) NOT NULL) ENGINE=InnoDB;
```

Creació de les taules del model de proves

```
CREATE TABLE client (
      DNI CHAR(12) NOT NULL,
      Nom VARCHAR(25) NOT NULL,
      Cognom1 VARCHAR(25) NOT NULL,
      Cognom2 VARCHAR(25),
      Adreca VARCHAR(75) NOT NULL,
      Compte CHAR(22),
      CONSTRAINT pk_client PRIMARY KEY(DNI)) ENGINE=InnoDB;
```

```
CREATE TABLE cotxe (
      Matricula VARCHAR(12) NOT NULL,
      Marca VARCHAR(30) NOT NULL,
      Model VARCHAR(100),
      CONSTRAINT pk_matricula PRIMARY KEY (Matricula))
            ENGINE=InnoDB;
```

```
CREATE TABLE carrega_trajectes (
      PeatgeEntrada VARCHAR(10) NOT NULL,
      NomEntrada VARCHAR(30) NOT NULL,
      HoraEntrada VARCHAR(30) NOT NULL,
      PeatgeSortida VARCHAR(10),
      NomSortida VARCHAR(30),
      HoraSortida VARCHAR(30),
      TeleTac VARCHAR(15) NOT NULL,
      Cotxe VARCHAR(12) NOT NULL) ENGINE=InnoDB;
```

```
CREATE TABLE Peatge (
      Codi CHAR(2) ,
      Autopista CHAR(3) NOT NULL,
      Numero CHAR(2),
      Nom VARCHAR(35) NOT NULL) ENGINE=InnoDB;
```

```
CREATE TABLE TeleTac (
      Contracte CHAR(30),
      Client CHAR(9) NOT NULL,
            FOREIGN KEY (Client) REFERENCES client(DNI),
            PRIMARY KEY (Contracte)) ENGINE=InnoDB;
```

```
CREATE TABLE Associar (
      Codi CHAR(6),
      TeleTac CHAR(30) NOT NULL,
      Cotxe VARCHAR(12) NOT NULL,
      DataInici VARCHAR(10) NOT NULL,
      DataFi VARCHAR(10),
              FOREIGN KEY (TeleTac)
                 REFERENCES TeleTac(Contracte),
              FOREIGN KEY (Cotxe)
                 REFERENCES Cotxe(Matricula)) ENGINE=InnoDB;
```

```
CREATE TABLE Trajecte (
      Cotxe VARCHAR(12),
      PeatgeEntrada VARCHAR(10),
      HoraEntrada VARCHAR(30) ,
      PeatgeSortida VARCHAR(10) ,
      HoraSortida VARCHAR(30)) ENGINE=InnoDB;
```

Càrrega física dels fitxers de text en les taules

```
LOAD DATA INFILE 'c:/uoc/pfc/benchmarks/clients.csv'
      INTO TABLE carrega
      FIELDS TERMINATED BY ';' LINES TERMINATED BY '\n';
```

```
LOAD DATA INFILE 'c:/uoc/pfc/benchmarks/trajectes.csv'
      INTO TABLE carrega_trajectes
      FIELDS TERMINATED BY ';' LINES TERMINATED BY '\n';
```

MySQL MyISAM; creació taules i càrrega de dades.

Creació de les taules inicials de càrrega

```
CREATE TABLE carrega (
      Dni CHAR(12) NOT NULL,
      Nom VARCHAR(25) NOT NULL,
      Cognom1 VARCHAR(25) NOT NULL,
      Cognom2 VARCHAR(25),
      Adreca VARCHAR(75) NOT NULL,
      Compte CHAR(22) NOT NULL,
      Teletac VARCHAR(30) NOT NULL,
      Data VARCHAR(10),
      Data2 VARCHAR(10),
      Matricula VARCHAR(12) NOT NULL,
      Marca VARCHAR(30) NOT NULL,
      Model VARCHAR(100));
```

```
CREATE TABLE carrega_trajectes (
      PeatgeEntrada VARCHAR(10) NOT NULL,
      NomEntrada VARCHAR(30) NOT NULL,
      HoraEntrada varchar(30) NOT NULL,
      PeatgeSortida VARCHAR(10),
      NomSortida VARCHAR(30),
      HoraSortida VARCHAR(30),
      TeleTac VARCHAR(15) NOT NULL,
      Cotxe VARCHAR(12) NOT NULL));
```

Creació de les taules del model de proves

```
CREATE TABLE client (
      DNI CHAR(12) NOT NULL,
      Nom VARCHAR(25) NOT NULL,
      Cognom1 VARCHAR(25) NOT NULL,
      Cognom2 VARCHAR(25),
```

```
      Adreca VARCHAR(75) NOT NULL,
      Compte CHAR(22),
      CONSTRAINT pk_client PRIMARY KEY(DNI));
```

```
CREATE TABLE cotxe (
      Matricula VARCHAR(12) NOT NULL,
      Marca VARCHAR(30) NOT NULL,
      Model VARCHAR(100),
      CONSTRAINT pk_matricula PRIMARY KEY (Matricula));
```

```
CREATE TABLE carrega_trajectes (
      PeatgeEntrada VARCHAR(10) NOT NULL,
      NomEntrada VARCHAR(30) NOT NULL,
      HoraEntrada VARCHAR(30) NOT NULL,
      PeatgeSortida VARCHAR(10),
      NomSortida VARCHAR(30),
      HoraSortida VARCHAR(30),
      TeleTac VARCHAR(15) NOT NULL,
      Cotxe VARCHAR(12) NOT NULL);
```

```
CREATE TABLE Peatge (
      Codi CHAR(2) ,
      Autopista CHAR(3) NOT NULL,
      Numero CHAR(2),
      Nom VARCHAR(35) NOT NULL);
```

```
CREATE TABLE TeleTac (
      Contracte CHAR(30),
      Client CHAR(9) NOT NULL,
            FOREIGN KEY (Client) REFERENCES client(DNI),
            PRIMARY KEY (Contracte));
```

```
CREATE TABLE Associar (
      Codi CHAR(6),
      TeleTac CHAR(30) NOT NULL,
      Cotxe VARCHAR(12) NOT NULL,
      DataInici VARCHAR(10) NOT NULL,
      DataFi VARCHAR(10),
            FOREIGN KEY (TeleTac)
                  REFERENCES TeleTac(Contracte),
            FOREIGN KEY (Cotxe)
                  REFERENCES Cotxe(Matricula));
```

```
CREATE TABLE Trajecte (
      Cotxe VARCHAR(12),
```

```
     PeatgeEntrada VARCHAR(10),
     HoraEntrada VARCHAR(30) ,
     PeatgeSortida VARCHAR(10) ,
     HoraSortida VARCHAR(30));
```

Càrrega física dels fitxers de text en les taules

```
LOAD DATA INFILE 'c:/uoc/pfc/benchmarks/clients.csv'
     INTO TABLE carrega
     FIELDS TERMINATED BY ';' LINES TERMINATED BY '\n';
```

```
LOAD DATA INFILE 'c:/uoc/pfc/benchmarks/trajectes.csv'
     INTO TABLE carrega_trajectes
     FIELDS TERMINATED BY ';' LINES TERMINATED BY '\n';
```

Població de dades i consultes

(estàndard per PostgreSQL, MySQL InnoDB i MySQL MyISAM)

Traspàs de dades a les taules del model de prova

```
INSERT INTO client
     SELECT DISTINCT DNI, Nom, Cognom1, Cognom2, Adreca
     FROM carrega;
```

```
INSERT INTO cotxe
     SELECT DISTINCT Matricula, Marca, Model
     FROM carrega;
```

```
INSERT INTO peatge ( nom, autopista)
     SELECT DISTINCT Nomentrada, SUBSTR(PeatgeEntrada,2,3)
     FROM carrega_trajectes;
```

```
INSERT INTO TeleTac
     SELECT DISTINCT(TeleTac), Dni
     FROM carrega;
```

```
INSERT INTO Associar (TeleTac, Cotxe, DataInici, DataFi)
     SELECT DISTINCT(TeleTac), Matricula, Data, Data2
     FROM carrega;
```

```
INSERT INTO Trajecte (cotxe, PeatgeEntrada, HoraEntrada,
           PeatgeSortida, HoraSortida)
      SELECT cotxe, peatgeEntrada, horaentrada, peatgeSortida,
           horaSortida
      FROM carrega_Trajectes ;
```

Consultes fetes

```
-- Consulta A

SELECT client.DNI, teletac.Contracte, associar.Teletac,
           cotxe.Matricula
      FROM Client, Teletac, Associar, Cotxe
      WHERE client.DNI = teletac.Client
           AND teletac.Contracte = associar.Teletac
           AND associar.Cotxe = cotxe.Matricula;
```

```
-- Consulta B

SELECT client.DNI, teletac.Contracte, associar.Teletac,
           cotxe.Matricula, trajecte.PeatgeEntrada,
           trajecte.HoraEntrada
      FROM client, teletac, associar, trajecte, cotxe
      WHERE client.DNI = teletac.Client
           AND teletac.Contracte = associar.Teletac
           AND associar.Cotxe = cotxe.Matricula
           AND cotxe.Matricula = trajecte.Cotxe;
```

```
-- Consulta C

SELECT client.DNI, teletac.Contracte, associar.Teletac,
           cotxe.Matricula, trajecte.PeatgeEntrada,
           trajecte.HoraEntrada
      FROM Client, Teletac, Associar, Trajecte, Cotxe
      WHERE client.DNI = teletac.Client
           AND teletac.contracte = associar.Teletac
           AND associar.cotxe = cotxe.Matricula
           AND cotxe.matricula = trajecte.Cotxe
           AND client.dni = '10258564N';
```

```
-- Consulta D

SELECT client.DNI, teletac.Contracte, associar.Teletac,
           cotxe.Matricula, trajecte.PeatgeEntrada,
           trajecte.HoraEntrada
      FROM client, teletac, associar, trajecte, cotxe
      WHERE client.dni = teletac.Client
           AND teletac.contracte = associar.Teletac
           AND associar.cotxe = cotxe.Matricula
           AND cotxe.matricula = trajecte.Cotxe
```

```
            AND client.DNI LIKE '%N%';
```

```
-- Consulta E

SELECT client.DNI, teletac.Contracte, associar.Teletac,
            cotxe.Matricula, trajecte.PeatgeEntrada,
            trajecte.HoraEntrada
      FROM client, teletac, associar, trajecte, cotxe
      WHERE client.dni = teletac.Client
            AND teletac.contracte = associar.Teletac
            AND associar.cotxe = cotxe.Matricula
            AND cotxe.matricula = trajecte.Cotxe
            AND client.DNI LIKE '%N%'
      ORDER BY matricula;
```

```
-- Consulta F

SELECT client.DNI, teletac.Contracte, associar.Teletac,
            cotxe.Matricula, trajecte.PeatgeEntrada,
            trajecte.HoraEntrada
      FROM client, teletac, associar, trajecte, cotxe
      WHERE client.DNI = teletac.Client
            AND teletac.Contracte = associar.Teletac
            AND associar.Cotxe = cotxe.Matricula
            AND cotxe.Matricula = trajecte.Cotxe
            AND client.Nom = 'Maria'
      ORDER BY matricula;
```

```
-- Consulta G

SELECT client.DNI, teletac.Contracte, associar.Teletac,
            cotxe.Matricula, trajecte.PeatgeEntrada,
            trajecte.HoraEntrada
      FROM client, teletac, associar, trajecte, cotxe
      WHERE client.DNI = teletac.Client
            AND teletac.Contracte = associar.Teletac
            AND associar.Cotxe = cotxe.Matricula
            AND cotxe.Matricula = trajecte.Cotxe
            AND trajecte.Peatgeentrada = 'AP7-35'
      ORDER BY matricula;
```

Plans d'execució PostgreSQL

Consulta A

```
Hash Join  (cost=6113.49..13018.77 rows=52338 width=100)
  Hash Cond: ("outer".teletac = "inner".contracte)
  ->  Hash Join  (cost=1031.71..4990.54 rows=52338 width=50)
        Hash Cond: (("outer".cotxe)::text = ("inner".matricula)::text)
        ->  Seq Scan on associar  (cost=0.00..1316.38 rows=52338 width=50)
        ->  Hash  (cost=734.97..734.97 rows=35497 width=16)
              ->  Seq Scan on cotxe  (cost=0.00..734.97 rows=35497 width=16)
  ->  Hash  (cost=4450.77..4450.77 rows=47600 width=50)
        ->  Hash Join  (cost=1181.78..4450.77 rows=47600 width=50)
              Hash Cond: ("outer".client = "inner".dni)
              ->  Seq Scan on teletac  (cost=0.00..952.00 rows=47600 width=47)
              ->  Hash  (cost=900.82..900.82 rows=33582 width=16)
                    ->  Seq Scan on client  (cost=0.00..900.82 rows=33582 width=16)
```

Consulta B

```
Merge Join  (cost=107359.98..1663658.41 rows=103549687 width=147)
  Merge Cond: (("outer".cotxe)::text = "inner"."?column4?")
  ->  Merge Join  (cost=16169.10..19091.38 rows=52338 width=116)
        Merge Cond: (("outer".matricula)::text = "inner"."?column5?")
        ->  Index Scan using idx_cotxe on cotxe  (cost=0.00..2048.47 rows=35497 width=16)
        ->  Sort  (cost=16169.10..16299.94 rows=52338 width=100)
              Sort Key: (associar.cotxe)::text
              ->  Hash Join  (cost=5081.77..9344.60 rows=52338 width=100)
                    Hash Cond: ("outer".teletac = "inner".contracte)
                    ->  Seq Scan on associar  (cost=0.00..1316.38 rows=52338 width=50)
                    ->  Hash  (cost=4450.77..4450.77 rows=47600 width=50)
                          ->  Hash Join  (cost=1181.78..4450.77 rows=47600 width=50)
                                Hash Cond: ("outer".client = "inner".dni)
                                ->  Seq Scan on teletac  (cost=0.00..952.00 rows=47600 width=47)
                                ->  Hash  (cost=900.82..900.82 rows=33582 width=16)
                                      ->  Seq Scan on client  (cost=0.00..900.82 rows=33582 width=16)
  ->  Sort  (cost=91190.88..92180.12 rows=395696 width=63)
        Sort Key: (trajecte.cotxe)::text
        ->  Seq Scan on trajecte  (cost=0.00..11022.96 rows=395696 width=63)
```

Consulta C

```
Hash Join  (cost=4238.76..48144.06 rows=518362 width=147)
  Hash Cond: (("outer".cotxe)::text = ("inner".cotxe)::text)
  ->  Seq Scan on trajecte  (cost=0.00..11022.96 rows=395696 width=63)
  ->  Hash  (cost=4238.11..4238.11 rows=262 width=116)
       ->  Nested Loop  (cost=1079.98..4238.11 rows=262 width=116)
            ->  Hash Join  (cost=1079.98..2660.67 rows=262 width=100)
                  Hash Cond: ("outer".teletac = "inner".contracte)
                  ->  Seq Scan on associar  (cost=0.00..1316.38 rows=52338 width=50)
                  ->  Hash  (cost=1079.39..1079.39 rows=238 width=50)
                      ->  Nested Loop  (cost=0.00..1079.39 rows=238 width=50)
                           ->  Index Scan using pk_client on client  (cost=0.00..6.01 rows=1 width=16)
                                 Index Cond: (dni = '10258564N'::bpchar)
                              ->  Seq Scan on teletac  (cost=0.00..1071.00 rows=238 width=47)
                                 Filter: ('10258564N'::bpchar = client)
              ->  Index Scan using idx_cotxe on cotxe  (cost=0.00..6.01 rows=1 width=16)
```

```
                    Index Cond: (("outer".cotxe)::text = (cotxe.matricula)::text)
```

Consulta D

```
Merge Join  (cost=107443.93..1663742.36 rows=103549687 width=147)
  Merge Cond: (("outer".cotxe)::text = "inner"."?column4?")
  ->  Merge Join  (cost=16253.05..19175.33 rows=52338 width=116)
        Merge Cond: (("outer".matricula)::text = "inner"."?column5?")
        ->  Index Scan using idx_cotxe on cotxe  (cost=0.00..2048.47 rows=35497 width=16)
        ->  Sort  (cost=16253.05..16383.90 rows=52338 width=100)
              Sort Key: (associar.cotxe)::text
              ->  Hash Join  (cost=5165.73..9428.56 rows=52338 width=100)
                    Hash Cond: ("outer".teletac = "inner".contracte)
                    ->  Seq Scan on associar  (cost=0.00..1316.38 rows=52338 width=50)
                    ->  Hash  (cost=4534.73..4534.73 rows=47600 width=50)
                          ->  Hash Join  (cost=1265.73..4534.73 rows=47600 width=50)
                                Hash Cond: ("outer".client = "inner".dni)
                                ->  Seq Scan on teletac  (cost=0.00..952.00 rows=47600 width=47)
                                ->  Hash  (cost=984.77..984.77 rows=33582 width=16)
                                      ->  Seq Scan on client  (cost=0.00..984.77 rows=33582 width=16)
```

Consulta E

```
Merge Join  (cost=107443.93..1663742.36 rows=103549687 width=147)
  Merge Cond: (("outer".cotxe)::text = "inner"."?column4?")
  ->  Merge Join  (cost=16253.05..19175.33 rows=52338 width=116)
        Merge Cond: (("outer".matricula)::text = "inner"."?column5?")
        ->  Index Scan using idx_cotxe on cotxe  (cost=0.00..2048.47 rows=35497 width=16)
        ->  Sort  (cost=16253.05..16383.90 rows=52338 width=100)
              Sort Key: (associar.cotxe)::text
              ->  Hash Join  (cost=5165.73..9428.56 rows=52338 width=100)
                    Hash Cond: ("outer".teletac = "inner".contracte)
                    ->  Seq Scan on associar  (cost=0.00..1316.38 rows=52338 width=50)
                    ->  Hash  (cost=4534.73..4534.73 rows=47600 width=50)
                          ->  Hash Join  (cost=1265.73..4534.73 rows=47600 width=50)
                                Hash Cond: ("outer".client = "inner".dni)
                                ->  Seq Scan on teletac  (cost=0.00..952.00 rows=47600 width=47)
                                ->  Hash  (cost=984.77..984.77 rows=33582 width=16)
                                      ->  Seq Scan on client  (cost=0.00..984.77 rows=33582 width=16)
                                            Filter: (dni ~~ '%N%'::text)
  ->  Sort  (cost=91190.88..92180.12 rows=395696 width=63)
        Sort Key: (trajecte.cotxe)::text
        ->  Seq Scan on trajecte  (cost=0.00..11022.96 rows=395696 width=63)
```

Consulta F

```
Merge Join  (cost=94960.29..104877.51 rows=518026 width=147)
  Merge Cond: (("outer".cotxe)::text = "inner"."?column4?")
  ->  Merge Join  (cost=3769.41..5910.55 rows=262 width=116)
        Merge Cond: (("outer".matricula)::text = "inner"."?column5?")
        ->  Index Scan using idx_cotxe on cotxe  (cost=0.00..2048.47 rows=35497 width=16)
        ->  Sort  (cost=3769.41..3770.06 rows=262 width=100)
              Sort Key: (associar.cotxe)::text
              ->  Hash Join  (cost=2178.18..3758.88 rows=262 width=100)
                    Hash Cond: ("outer".teletac = "inner".contracte)
                    ->  Seq Scan on associar  (cost=0.00..1316.38 rows=52338 width=50)
                    ->  Hash  (cost=2177.59..2177.59 rows=239 width=50)
```

```
                        ->  Hash Join  (cost=985.19..2177.59 rows=239 width=50)
                              Hash Cond: ("outer".client = "inner".dni)
                              ->  Seq Scan on teletac  (cost=0.00..952.00 rows=47600 width=47)
                              ->  Hash  (cost=984.77..984.77 rows=168 width=16)
                                    ->  Seq Scan on client  (cost=0.00..984.77 rows=168 width=16)
                                          Filter: ((nom)::text = 'Maria'::text)
  ->  Sort  (cost=91190.88..92180.12 rows=395696 width=63)
        Sort Key: (trajecte.cotxe)::text
        ->  Seq Scan on trajecte  (cost=0.00..11022.96 rows=395696 width=63)
```

Consulta F desprès de fer el ANALYZE

```
Sort  (cost=17768.48..17769.10 rows=248 width=129)
  Sort Key: cotxe.matricula
  ->  Hash Join  (cost=3801.09..17758.62 rows=248 width=129)
        Hash Cond: (("outer".cotxe)::text = ("inner".cotxe)::text)
        ->  Seq Scan on trajecte  (cost=0.00..11658.67 rows=459267 width=45)
        ->  Hash  (cost=3801.07..3801.07 rows=9 width=110)
              ->  Nested Loop  (cost=2088.86..3801.07 rows=9 width=110)
                    ->  Hash Join  (cost=2088.86..3773.84 rows=9 width=97)
                          Hash Cond: ("outer".teletac = "inner".contracte)
                          ->  Seq Scan on associar  (cost=0.00..1387.58 rows=59458 width=47)
                          ->  Hash  (cost=2088.85..2088.85 rows=7 width=50)
                                ->  Hash Join  (cost=984.79..2088.85 rows=7 width=50)
                                      Hash Cond: ("outer".client = "inner".dni)
                                      ->  Seq Scan on teletac  (cost=0.00..894.66 rows=41866 width=47)
                                      ->  Hash  (cost=984.77..984.77 rows=5 width=16)
                                            ->  Seq Scan on client  (cost=0.00..984.77 rows=5 width=16)
                                                  Filter: ((nom)::text = 'Maria'::text)
                    ->  Index Scan using idx_cotxe on cotxe  (cost=0.00..3.01 rows=1 width=13)
                          Index Cond: (("outer".cotxe)::text = (cotxe.matricula)::text)
```

Consulta G

```
Merge Join  (cost=28289.65..38229.77 rows=517885 width=147)
  Merge Cond: (("outer".cotxe)::text = "inner"."?column5?")
  ->  Merge Join  (cost=12120.56..14287.45 rows=1979 width=79)
        Merge Cond: (("outer".matricula)::text = "inner"."?column4?")
        ->  Index Scan using idx_cotxe on cotxe  (cost=0.00..2048.47 rows=35497 width=16)
        ->  Sort  (cost=12120.56..12125.50 rows=1979 width=63)
              Sort Key: (trajecte.cotxe)::text
              ->  Seq Scan on trajecte  (cost=0.00..12012.20 rows=1979 width=63)
                    Filter: ((peatgeentrada)::text = 'AP7-35'::text)
  ->  Sort  (cost=16169.10..16299.94 rows=52338 width=100)
        Sort Key: (associar.cotxe)::text
        ->  Hash Join  (cost=5081.77..9344.60 rows=52338 width=100)
              Hash Cond: ("outer".teletac = "inner".contracte)
              ->  Seq Scan on associar  (cost=0.00..1316.38 rows=52338 width=50)
              ->  Hash  (cost=4450.77..4450.77 rows=47600 width=50)
                    ->  Hash Join  (cost=1181.78..4450.77 rows=47600 width=50)
                          Hash Cond: ("outer".client = "inner".dni)
                          ->  Seq Scan on teletac  (cost=0.00..952.00 rows=47600 width=47)
                          ->  Hash  (cost=900.82..900.82 rows=33582 width=16)
                                ->  Seq Scan on client  (cost=0.00..900.82 rows=33582 width=16)
```

Plans d'execució MySQL InnoDB i MySQL MyISAM.

Consulta A

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	cotxe	index	PRIMARY,idx_cotxe	PRIMARY	12	<null>	35830	Using index
1	SIMPLE	associar	ref	TeleTac,Cotxe	Cotxe	12	mysql.cotxe.matricula	1	
1	SIMPLE	teletac	eq_ref	PRIMARY,Client	PRIMARY	30	mysql.associar.TeleTac1		
1	SIMPLE	client	eq_ref	PRIMARY	PRIMARY	12	mysql.teletac.Client1	Using where; Using index	

Consulta B

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	trajecte	ALL	<null>	<null>	<null>	<null>	459373	
1	SIMPLE	cotxe	eq_ref	PRIMARY,idx_cotxe	PRIMARY	12	mysql.trajecte.Cotxe	1	Using index
1	SIMPLE	associar	ref	TeleTac,Cotxe	Cotxe	12	mysql.cotxe.matricula	1	
1	SIMPLE	teletac	eq_ref	PRIMARY,Client	PRIMARY	30	mysql.associar.TeleTac	1	
1	SIMPLE	client	eq_ref	PRIMARY	PRIMARY	12	mysql.teletac.Client	1	Using where; Using index

Consulta C

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	client	const	PRIMARY	PRIMARY	12	const	1	Using index
1	SIMPLE	teletac	ref	PRIMARY,Client	Client	9	const	1	Using where; Using index
1	SIMPLE	associar	ref	TeleTac,Cotxe	TeleTac	30	mysql.teletac.Contracte	1	
1	SIMPLE	trajecte	ALL	<null>	<null>	<null>	<null>	459373	
1	SIMPLE	cotxe	eq_ref	PRIMARY,idx_cotxe	PRIMARY	12	mysql.associar.Cotxe	1	Using where; Using index

Consulta D

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	trajecte	ALL	<null>	<null>	<null>	<null>	459373	
1	SIMPLE	cotxe	eq_ref	PRIMARY,idx_cotxe	PRIMARY	12	mysql.trajecte.Cotxe	1	Using index
1	SIMPLE	associar	ref	TeleTac,Cotxe	Cotxe	12	mysql.cotxe.matricula	1	
1	SIMPLE	teletac	eq_ref	PRIMARY,Client	PRIMARY	30	mysql.associar.TeleTac	1	
1	SIMPLE	client	eq_ref	PRIMARY	PRIMARY	12	mysql.teletac.Client	1	Using where; Using index

Consulta E

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	trajecte	ALL	<null>	<null>	<null>	<null>	459373	Using temporary; Using filesort
1	SIMPLE	cotxe	eq_ref	PRIMARY,idx_cotxe	PRIMARY	12	mysql.trajecte.Cotxe	1	Using index
1	SIMPLE	associar	ref	TeleTac,Cotxe	Cotxe	12	mysql.cotxe.matricula	1	
1	SIMPLE	teletac	eq_ref	PRIMARY,Client	PRIMARY	30	mysql.associar.TeleTac	1	
1	SIMPLE	client	eq_ref	PRIMARY	PRIMARY	12	mysql.teletac.Client	1	Using where; Using index

Consulta F

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	trajecte	ALL	<null>	<null>	<null>	<null>	459373	Using temporary; Using filesort
1	SIMPLE	cotxe	eq_ref	PRIMARY,idx_cotxe	PRIMARY	12	mysql.trajecte.Cotxe	1	Using index
1	SIMPLE	associar	ref	TeleTac,Cotxe	Cotxe	12	mysql.cotxe.matricula	1	
1	SIMPLE	teletac	eq_ref	PRIMARY,Client	PRIMARY	30	mysql.associar.TeleTac	1	
1	SIMPLE	client	eq_ref	PRIMARY	PRIMARY	12	mysql.teletac.Client	1	Using where

Consulta G

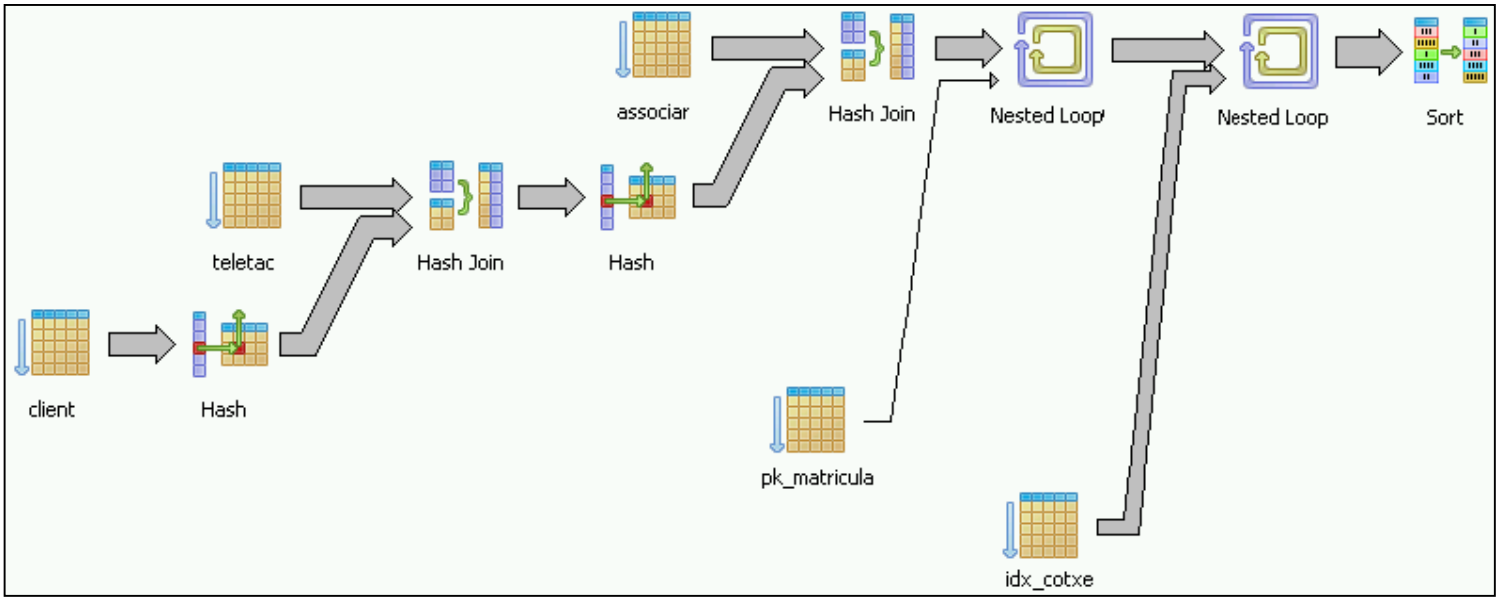
id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	trajecte	ALL	<null>	<null>	<null>	<null>	459373	Using where; Using temporary; Using filesort
1	SIMPLE	cotxe	eq_ref	PRIMARY,idx_cotxe	PRIMARY	12	mysql.trajecte.Cotxe	1	Using index
1	SIMPLE	associar	ref	TeleTac,Cotxe	Cotxe	12	mysql.cotxe.matricula	1	
1	SIMPLE	teletac	eq_ref	PRIMARY,Client	PRIMARY	30	mysql.associar.TeleTac	1	
1	SIMPLE	client	eq_ref	PRIMARY	PRIMARY	12	mysql.teletac.Client	1	Using where; Using index

Annex III. Optimització consulta F.

La revisió dels passos que el planificador elegeix per a executar les consultes, pot ajudar a resoldre els problemes de rendiment.

En el cas de la consulta F (tal com s'ha exposat en l'apartat de Benchmarks), s'hauria de crear un índex a la taula trajecte (pel camp cotxe). De fer-se així, i de crear-se també un cluster per aquest mateix índex, un cop fet de nou el analyze, la consulta passaria a utilitzar el pla d'accés següent:

```
Sort  (cost=5236.86..5237.46 rows=237 width=129)
  Sort Key: cotxe.matricula
  ->  Nested Loop  (cost=2088.86..5227.52 rows=237 width=129)
        ->  Nested Loop  (cost=2088.86..3801.07 rows=9 width=110)
              ->  Hash Join  (cost=2088.86..3773.84 rows=9 width=97)
                    Hash Cond: ("outer".teletac = "inner".contracte)
                    ->  Seq Scan on associar  (cost=0.00..1387.58 rows=59458 width=47)
                    ->  Hash  (cost=2088.85..2088.85 rows=7 width=50)
                          ->  Hash Join  (cost=984.79..2088.85 rows=7 width=50)
                                Hash Cond: ("outer".client = "inner".dni)
                                ->  Seq Scan on teletac  (cost=0.00..894.66 rows=41866 width=47)
                                ->  Hash  (cost=984.77..984.77 rows=5 width=16)
                                      ->  Seq Scan on client  (cost=0.00..984.77 rows=5 width=16)
                                            Filter: ((nom)::text = 'Maria'::text)
              ->  Index Scan using pk_matricula on cotxe  (cost=0.00..3.01 rows=1 width=13)
                    Index Cond: (("outer".cotxe)::text = (cotxe.matricula)::text)
        ->  Index Scan using idx_cotxe on trajecte  (cost=0.00..158.01 rows=39 width=45)
              Index Cond: ((trajecte.cotxe)::text = ("outer".cotxe)::text)
```

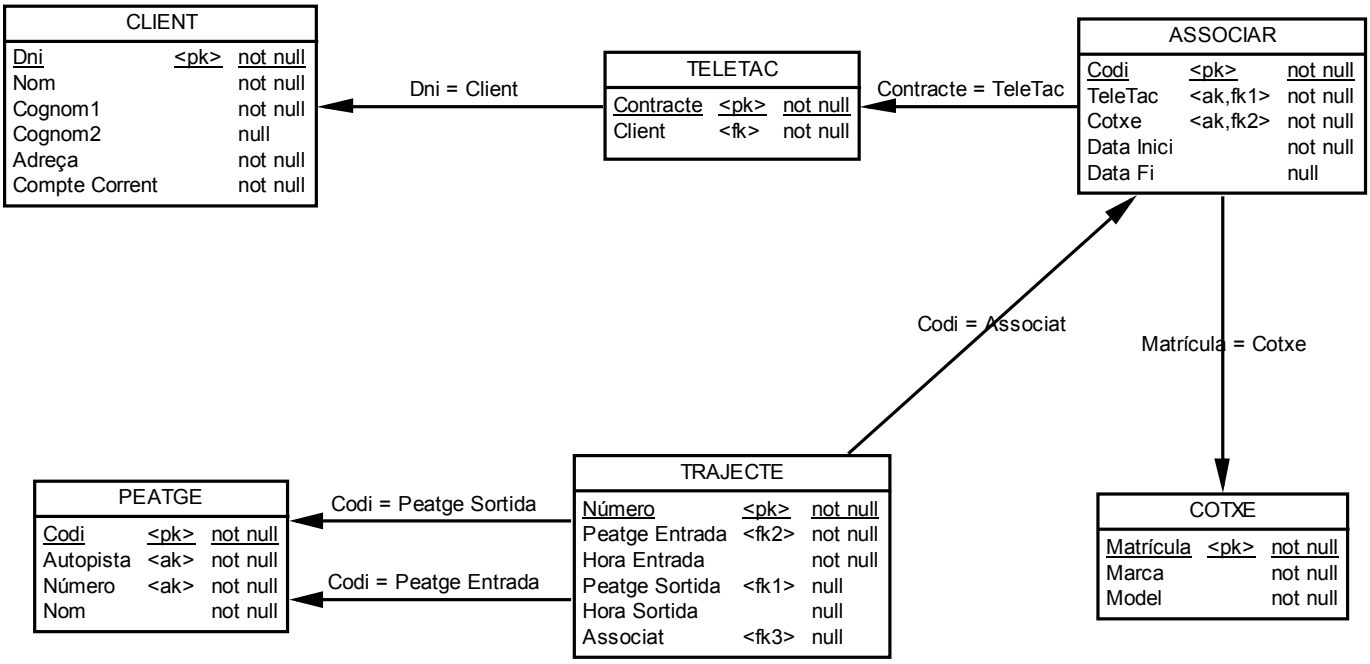


S'ha de fer notar que la consulta trigava inicialment 23,7 segons a executar-se. En utilitzar-se el Analyze per actualitzar les estadístiques va passar a fer-ho en només 2,4 segons. L'optimització final que ha permès que s'executi en només 0,4 segons, ha consistit en crear un índex i en clusteritzar les dades utilitzant el mateix ordre.

Annex IV. Dades sintètiques pràctica SGBD.

Tal com s'ha exposat a l'apartat de Benchmarks, les dades utilitzades per a fer les comprovacions de rendiment són les subministrades al segon semestre de 2005 a l'assignatura SGBD[2] per a fer la pràctica final.

L'estructura lògica inicial de les dades era:



La pràctica consistia bàsicament en aconseguir un òptim rendiment del sistema exposat (sobre Oracle 9i) per una consulta donada, sense que aquestes optimitzacions perjudiquessin altres possibles consultes de tipus ad-hoc.

L'enunciat demanava concretament:

d. *Decidir l'organització física més adequada per a cada taula "lògica".*
e. *Fer les suposicions oportunes per a determinar els atributs físics de cadascuna de les taules.*
f. *Determinar la clàusula d'emmagatzematge (storage clause) de cadascuna de les taules i dels índexs necessaris per implementar les restriccions (utilitzar les estadístiques o fer els càlculs oportuns)*
g. *Crear dos tablespaces (de la mida adequada), per a guardar en un d'ells les dades i a l'altre els índexs. Els dos hauran d'estar gestionats pel propi sistema utilitzant mapa de bits i un d'ells haurà de tenir gestió automàtica de l'espai de segments. El dos tablespaces hauran de poder créixer de forma automàtica.*
h. *Crear un nou usuari amb els permisos oportuns per a crear objectes als dos tablespaces anteriors i per a llegir les taules de càrrega.*

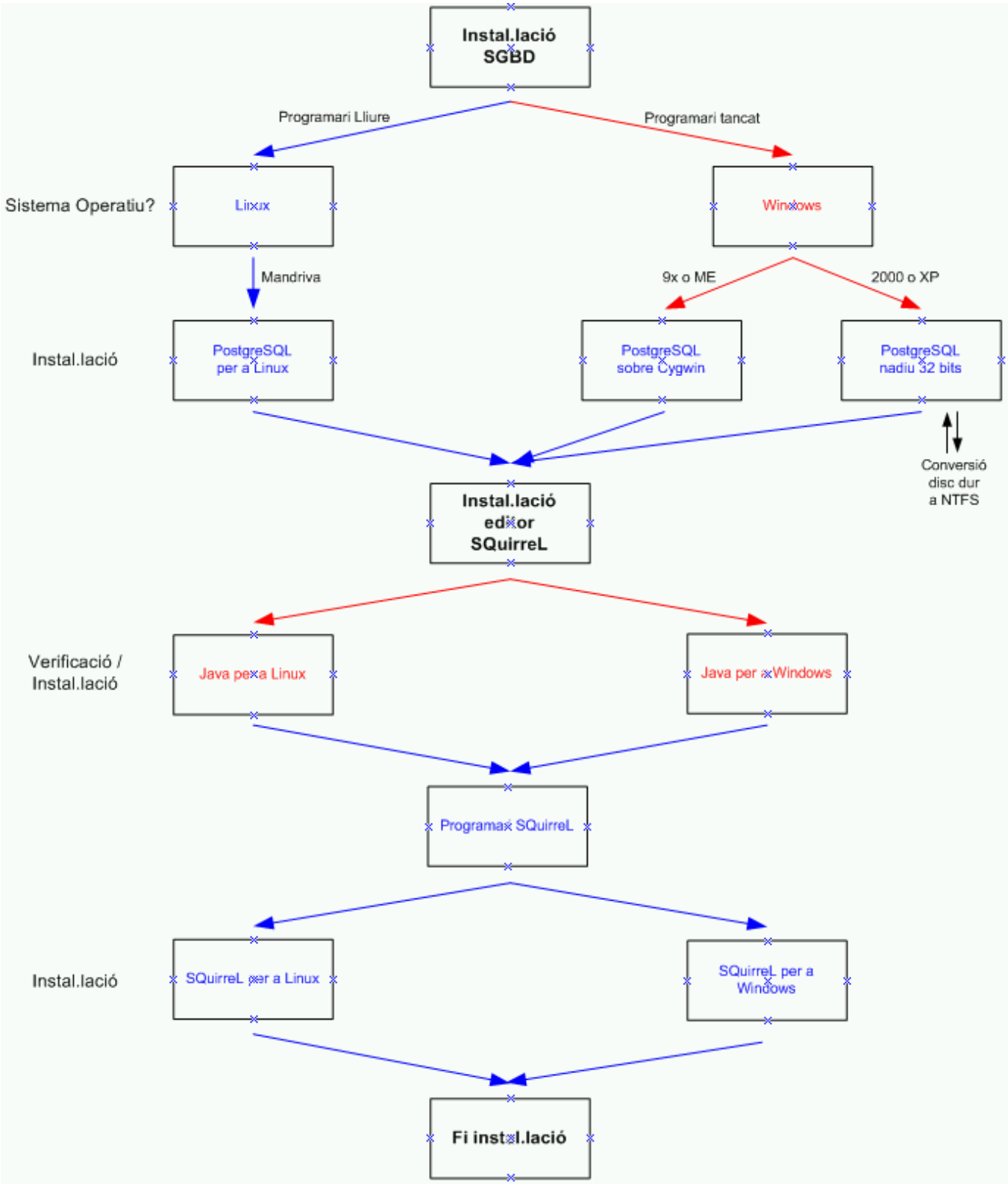
[2] de l'enginyeria informàtica de la Universitat Oberta de Catalunya (UOC).

i. *A partir de les decisions i dels càlculs de disseny anteriors, tornar a crear les taules, amb els paràmetres físics oportuns i en el tablespace que hi correspongui.*
j. *Decidir de forma raonada, els índex i de quin tipus (b-tree, mapa de bits o index-join) fan falta per fer la següent consulta més eficient (òptima);*
    a. *Nom dels peatges d'entrada i sortida, data (entrada) i matrícula que han fet servir els cotxes autoritzats per Na Iratxe Puigcerver Provenzals durant l'any 2002, ordenats per la matricula del cotxe i data (per a cada cotxe).*

Com s'observa, mentre que a l'esmentada pràctica es demana optimitzar un sistema manualment (mitjançant l'estudi de les característiques de les dades i dels tipus de consultes que s'han de fer), en l'apartat dels benchmarks es capgira el concepte, i s'intenta comprovar com d'intel·ligents són els algorismes 'planificadors' que tenen els SGBDs per a optimitzar l'accés a les taules, a fi de tornar uns resultats concrets sense cap mena d'optimització manual.

Annex V. Arbre instal·lació programari

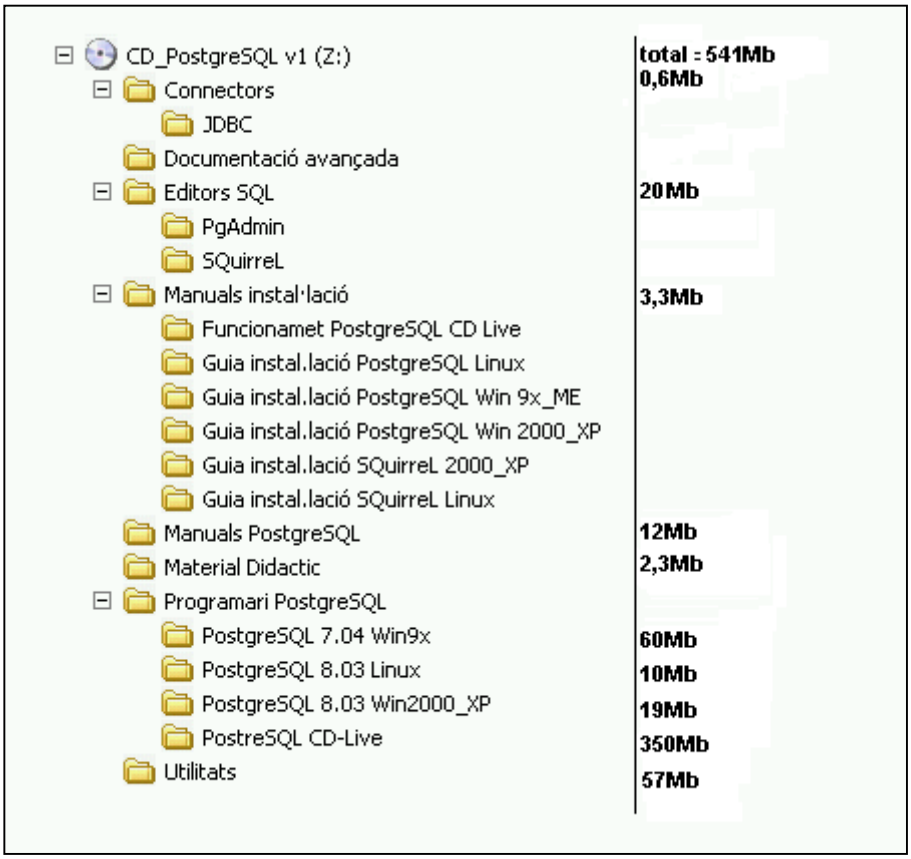
Depenent del sistema operatiu que utilitzi l'ordinador on s'hagi de fer funcionar l'SGBD PostgreSQL i l'editor SQL SQuirreL, s'haurà d'elegir entre les diferents possibilitats d'instal·lació.



Annex VI. Contingut CD programari PostgreSQL

Mentre es realitzaven els manuals d'instal·lació del PostgreSQL i l'SquirreL (adjunts en els annexos següents), es va recopilar tota aquella documentació, programari, i material en general, que pot ser necessari pels alumnes d'utilitzar-se les esmentades eines en un cicle formatiu.

La recopilació s'ha emmagatzemat en un CDROM, seguint la següent estructura:



Els continguts per directoris, són:

Connectors: Conté els drivers JDBC necessaris per a poder utilitzar el PostgreSQL des de l'SQuirreL, i per a poder utilitzar l'SGBD des d'aplicacions java pròpies.

Documentació avançada: S'han separat en un directori específic aquells documents que es creuen interessants, però que són prou complexos com per a ser 'opcionals', o de només lectura en estudis concrets.

Editors SQL: En aquest directori s'ha inclòs tant l'editor SQL SQuirreL (el fitxer executable *jar* serveix per a qualsevol entorn), com el PgAdmin. La versió 1.2 d'aquest últim s'instal·la automàticament amb el PostgreSQL, però la aquí inclosa (1.4) incorpora millores, entre elles, el generador gràfic dels plans d'execució de les consultes.

Manuals instal·lació: En aquest directori es recullen els manuals d'instal·lació del PostgreSQL pels sistemes operatius Microsoft Windows 95, 98, ME, 2000, XP i Linux Mandriva, i els manuals d'instal·lació del SQuirreL per al Windows 2000/XP (compatible almenys amb Windows 98/ME) i Linux Mandriva.

Manuals PostgreSQL: Recull el manual oficial del PostgreSQL 8.0 en anglès (més de mil tres-centes pàgines), una versió en castellà de la versió 6.5 (suficient per la majoria de consultes), el llibre Practical PostgreSQL tant en versió navegable com en versió imprimible (editorial O'Reilly), i unes guies bàsiques de plPgSQL.

Material didàctic: S'ha pensat d'emmagatzemar en aquest directori aquell material més d'estil 'multimedia', que pugui ser interessant (hi han dos presentacions fetes en flash).

Programari PostgreSQL: S'inclou els programes que permetés fer la instal·lació del PostgreSQL en els diferents sistemes operatius que es contemplen.

Utilitats: S'ha utilitzat l'espai sobrer del CDROM per a incloure el WinZip, l'Adobe Acrobat, i el java 1.5 per a Windows i Linux.

Annex VII. Manual d'instal.lació del programari PostgreSQL (versió Linux)

Manual d'instal·lació del programari PostgreSQL.

(versió Linux)

Introducció

Que és PostgreSQL?

PostgreSQL és un programari de Base de Dades (BD) amb llicència de distribució tipus BSD[3], o sigui, és un programa d'ús i distribució lliure. Del mateix se'n pot obtenir el programa font per a modificar-lo, ampliar-lo, o corregir segons les pròpies necessitats.

D'entre els Sistemes de Gestió de Bases de Dades (SGBD) actuals, destaca per la seva estabilitat, possibilitats i versatilitat, comptant amb característiques que fins i tot alguns SGBD comercials no incorporen a l'actualitat. L'haver estat desenvolupat des d'un inici en un entorn acadèmic (Universitat de Berkeley – 1974), fa que estigui concebut com a una plataforma d'exhibició de *l'estat del art* de la tecnologia en aquesta branca del programari. Al seu inici era una demostració del que haurien de ser els SGBDs futurs (incorporant per tant opcions que 'comercialment' no eren desenvolupables segons un estricte criteri de beneficis empresarials). Actualment encara destaca per tenir característiques molt avançades, que difícilment es troben en altres sistemes de la seva categoria (per exemple: el subsistema GisT, que és utilitzat per a facilitar l'emmagatzemament i indexació de polígons tridimensionals en sistemes cartogràfics i topogràfics).

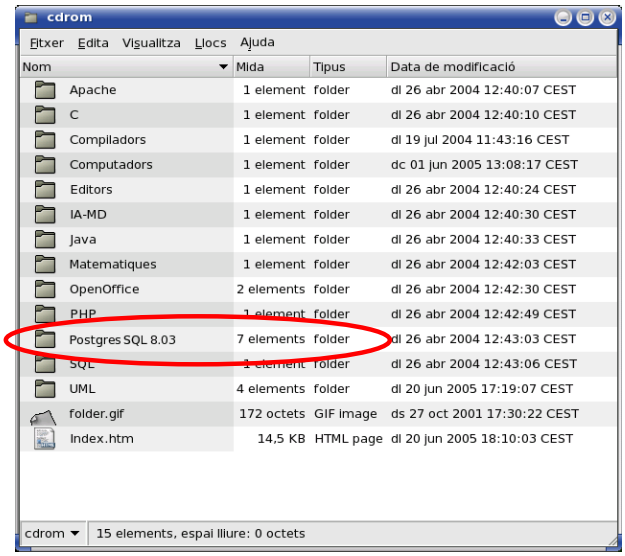
El constant creixement de la popularitat del programari lliure (tipus GNU, GPL o similars), fa que sigui imprescindible tenir una coneixença profunda d'aquests entorns per a desenvolupar els aplicatius que demanda el mercat actual. La constant aparició d'empreses que ofereixen suport al programari lliure (IBM, Novell, SUN, entre d'altres), fa que s'esvaeixin els dubtes de si aquest tipus de programari és viable en un futur, ja que en aquest nou paradigma es traslladen les inversions inicials de compra, al manteniment i a millors desenvolupaments.

La disponibilitat del codi font dels aplicatius amb llicència BSD, fa que no es depengui del fabricant del programari per la continuïtat del producte. Si el fabricant desaparegués, la inversió feta en el desenvolupament del programari propi (sobre un SGBD concret per exemple) estaria assegurada, ja que aquest *motor de base de dades* seguiria tenint el manteniment d'una comunitat d'usuaris (distribuïda probablement per tot el mon), que estaria interessada en solucionar els problemes que sorgissin, i fins i tot de continuar la seva evolució. També s'ha de considerar que el desenvolupament dels mòduls d'idioma o altres característiques que poden no ser rentables per un fabricant concret, pot ser desenvolupats pels 'tercers' que estiguin interessats.

[3] Existeixen altres BDs que es mouen a l'entorn del programari lliure, però que no ho són realment. MySQL per exemple, té llicència dual (gratuït per a ús privat i de pagament en usos comercials/empresarials).

Instal·lació de PostgreSQL (Linux)

La present guia d'instal·lació pressuposa uns coneixements bàsics dels Sistema Operatiu Linux i de les eines GNU. Malgrat no s'entri en els detalls de les opcions elegides per a fer la instal·lació del PostgresSQL 8.0.3, s'especifiquen els passos exactes a seguir per a portar a terme aquesta tasca (utilitzant la distribució de Linux Mandriva 10).
Se suposa que estan preinstal·lades les eines bàsiques per a descomprimir fitxers del tipus tar, que hi ha un usuari creat, i que es pot utilitzar l'ordinador des d'un *shell* com a usuari *root*.
El programari que instal·larem el trobarem al CDROM de PostgreSQL lliurat conjuntament amb els mòduls de l'assignatura (segons imatge adjunta), o alternativament es pot descarregar des de la pestanya Recursos de l'aula (la versió complerta ocupa uns 10Mb aproximadament).
Considerarem que hem descarregat o copiat el fitxer postgresql-8.0.3.tar.bz2 al directori /home/usuOrd/Desktop/



Per a començar la instal·lació haurem d'obrir una finestra de shell amb l'usuari **root** (amb la comanda *su*), i crear l'usuari **postgres**. Quan creem la contrasenya per a aquest usuari, l'haurem d'introduir i desprès confirmar.
Un cop fet això, canviarem al directori de destinació del postgreSQL que volem instal·lar, i descomprimirem el fitxer amb la comanda *tar*.

```
[usuOrd@mandrake ~]$ su
Password:
[root@mandrake usuOrd]# useradd postgres
[root@mandrake usuOrd]# passwd postgres
Changing password for user postgres.
New UNIX password:
Retype new UNIX password:
[root@mandrake usuOrd]# cd /usr/src/
[root@mandrake src]# tar -xjf /home/usuOrd/Desktop/postgresql-8.0.3.tar.bz2
```

Canviant al directori postgresql-8.0.3.tar.bz2 i fent un *ls*, comprovarem que els fitxers s'han descomprimit correctament.

```
[root@mandrake src]# cd postgresql-8.0.3/
[root@mandrake postgresql-8.0.3]# ls
aclocal.m4  configure*    contrib/   doc/            HISTORY  Makefile  src/
config/     configure.in  COPYRIGHT  GNUmakefile.in  INSTALL  README
[root@mandrake postgresql-8.0.3]# ./configure --prefix=/opt/pgsql --without-readline --without-zlib
```

Tal com es mostra a la imatge superior, tot seguit haurem de fer un *configure*, per a preparar la posterior compilació. És important posar tots els paràmetres tal com estan a l'exemple.

Al acabar el procés de *configure* (pot trigar alguns minuts), si tot va bé, veurem (segons la següent imatge) que acaba sense cap error.

```
config.status: linking ./src/backend/port/dynloader/linux.h to src/include/dynloader.h
config.status: linking ./src/include/port/linux.h to src/include/pg_config_os.h
config.status: linking ./src/makefiles/Makefile.linux to src/Makefile.port
[root@mandrake postgresql-8.0.3]# make install
```

Tot seguit hem de fer el *make install* (segons imatge anterior).

Aquest és el procés que pot trigar més estona de tota la instal·lació. En finalitzar ens apareixerà la frase: **PostgreSQL installation complete!**

```
make -C config install
make[1]: Entering directory `/usr/src/postgresql-8.0.3/config'
mkdir -p -- /opt/pgsql/lib/pgxs/config
/bin/sh ../config/install-sh -c -m 755 ./install-sh /opt/pgsql/lib/pgxs/config/install-sh
/bin/sh ../config/install-sh -c -m 755 ./mkinstalldirs /opt/pgsql/lib/pgxs/config/mkinstalldirs
make[1]: Leaving directory `/usr/src/postgresql-8.0.3/config'
PostgreSQL installation complete.
[root@mandrake postgresql-8.0.3]#
```

Ara hem de crear el directori on emmagatzemarem la base de dades (amb un *mkdir*), donar permisos d'accés al directori (amb el *chown*), canviar d'usuari (amb el *su*), i ja des d'aquest crear l'espai on s'emmagatzemaran les bases de dades.
Tot això ho farem segons es pot veure en la següent captura de pantalla:

```
[root@mandrake postgresql-8.0.3]# mkdir /opt/pgsql/data -p
[root@mandrake postgresql-8.0.3]# chown postgres /opt/pgsql/data
[root@mandrake postgresql-8.0.3]# chown postgres /usr/src/postgresql-8.0.3/
[root@mandrake postgresql-8.0.3]# su - postgres
[postgres@mandrake ~]$ /opt/pgsql/bin/initdb -D data/
```

El procés d'inicialització pot trigar una estona en executar-se.

En acabar, ens hauria d'indicar que ja ho podem engegar (similar a la següent pantalla):

```
vacuuming database template1 ... ok
copying template1 to template0 ... ok

WARNING: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the -A option the
next time you run initdb.

Success. You can now start the database server using:
    /opt/pgsql/bin/postmaster -D data
or
    /opt/pgsql/bin/pg_ctl -D data -l logfile start

[postgres@mandrake ~]$
```

Abans, però, crearem la base de dades de treball amb la instrucció **createdb**. Els paràmetres a utilitzar són els que hi han tot seguit:

```
[iplana@mandrake ~]$ /opt/pgsql/bin/createdb --username=postgres --password uoc
Password:
CREATE DATABASE
[iplana@mandrake ~]$
```

Ara ja podem engegar el *motor* de la base de dades;

```
[postgres@mandrake /]$ /opt/pgsql/bin/postmaster -D /home/postgres/data/
LOG:  database system was shut down at 2005-10-20 17:01:52 CEST
LOG:  checkpoint record is at 0/A32A30
LOG:  redo record is at 0/A32A30; undo record is at 0/0; shutdown TRUE
LOG:  next transaction ID: 544; next OID: 17230
LOG:  database system is ready
```

Un cop s'ha engegat correctament el *postmaster*, la finestra d'aquest shell es queda executant-ho (no ens retorna el control).
El programa que s'executa en aquesta cònsola, és pròpiament dit el motor de la base de dades. Aquest queda a l'espera de que es facin peticions pel port de comunicacions 5432, per a passar a recollir-les, processar-les, i tornar els resultats.

Per a fer consultes o altres operacions amb la base de dades, haurem d'obrir un nou shell. Un cop a aquest, obrirem l'editor segons l'exemple:

```
[iplana@mandrake /]$ /opt/pgsql/bin/psql --username=postgres --password uoc
Password:
Welcome to psql 8.0.3, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

uoc=#
```

Com que especifiquem que volem engegar la cònsola amb un usuari específic (postgres), ens demanarà la contrasenya en el procés.

En el mateix moment d'engegar la sessió d'edició SQL, hem triat la base de dades que utilitzarem (en aquest cas l'anomenada UOC).

La cònsola del **psql** que hem engegat ens servirà per a introduir 'rudimentàriament' comandes SQL bàsiques. En el document ***Instal·lació editor SQuirreL*** s'especifica com utilitzar un entorn de treball amb finestres, per a introduir i editar sentències SQL.

(per a aturar els processos que s'executen a les finestres que tenim engegades -postmaster i psql-, cal pitjar ctrl+q i ctrl+c respectivament).

Provant la instal·lació de PostgreSQL

Un cop tot instal·lat, especificarem els passos per a posar en marxa el motor de la base de dades, crear des de l'editor (psql) una taula, inserir unes dades i fer la consulta de verificació final.
En primer lloc reiniciarem l'ordinador. Un cop carregat el Linux obrirem una finestra del shell, i canviarem a l'usuari *postgres*:

su - postgres

Tot seguit se'ns demanarà la contrasenya. Una vegada introduïda, engegarem el motor de la base de dades amb la instrucció:

/opt/pgsql/bin/postmaster -D /home/postgres/data/

El shell no retornarà el control, ja que queda executant aquest programa.
Per arrancar l'editor psql, haurem d'engegar un nou shell, i introduir la següent línia:

/opt/pgsql/bin/psql --username=postgres --password uoc

Se'ns demanarà de nou la contrasenya. Una vegada introduïda, ja podrem treballar amb la base de dades.
En aquest punt hauríem de tenir en la finestra del shell, una pantalla segons es mostra:

```
[iplana@mandrake /]$ /opt/pgsql/bin/psql --username=postgres --password uoc
Password:
Welcome to psql 8.0.3, the PostgreSQL interactive terminal.
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit
uoc=#
```

La primera instrucció que podem executar, és la de creació d'una taula de prova:

create table test (nom varchar(60));

Ens hauria d'indicar que s'ha creat correctament.
La següent cosa a fer, és introduir dades:

insert into test values ('Registre de prova 1');

Hauria de retornar el missatge conforme s'han inserit correctament les dades.
La prova final, és fe una consulta:

select * from test;

Hauria de retornar el valor de un única registre, amb el text que acabem d'inserir.

```
uoc=# create table test (nom varchar(60));
CREATE TABLE
uoc=# insert into test values('Registre de prova 1');
INSERT 25424 1
uoc=# select * from test;
         nom
---------------------
 Registre de prova 1
(1 row)
uoc=#
```

(creació de la taula, inserció de les dades i posterior consulta)

Resolució de problemes i FAQ

Aquesta guia d'instal·lació serveix per a instal·lar PostgreSQL sobre un Linux Ubuntu, Red Hat, o altres?

- Si/No.
- No hi hauria d'haver gaires diferències entre els diferents sistemes, però pot ser que fins i tot dues instal·lacions de la mateixa distribució de Linux (si s'han triat diferents paquets durant la instal·lació inicial), ja tinguin diferències importants (pot ser que fins i tot, en un d'ells ja se'ns instal·les el PostgreSQL per defecte!)
- La guia s'ha d'agafar com a model. Es pot seguir pas a pas si es té la versió del SO específic, en cas contrari s'hauria d'utilitzar com una guia genèrica de referència.

Quins requeriments de maquinari té el PostgreSQL?

- Tal com l'utilitzem a les assignatures de l'àrea de BD, es pot executar amb el maquinari bàsic recomanat per la UOC. Un ordinador amb processador x86 (tipus Pentium, Athlon o similar), capaç d'executar el Linux sobre el que s'instal·la el programari, amb uns 100Mb d'espai de disc lliure (per la instal·lació bàsica, comptant la reserva d'espai per la base de dades), el podrà executar sense problemes.
- Si penseu en un ús empresarial (amb desenes o centenars de connexions), caldrà que el servidor disposi de discs durs SCSI (preferiblement en RAID 5), amb almenys 1Gb de memòria RAM, i preferentment amb un parell (o més) de nuclis de processament (s'ha de dimensionar segons el volum de transaccions que tingui que suportar – i això no és pas senzill!).

PostgreSQL funciona en diversos sistemes operatius?

- Si. Al menys funciona en una dotzena de sistemes operatius diferents (podeu trobar aquest mateix manual d'instal·lació pel Windows 95/98 i pel Windows 2000/XP). De fet, al tenir PostbreSQL llicència de distribució de tipus BSD, res impedeix compilar les fonts pel sistema operatiu que ens interessi (encara que per a això, cal estar bastant especialitzat).

Com pot ser que PostgreSQL sigui gratuït, si té característiques superiors a l'Informix, i fins i tot similars a l'Oracle?

- La resposta és senzilla, però complexa de respondre. Com que part de *l'encant* que envolta al programari lliure és la investigació, la recerca constant de millores i fins i tot la superació personal, res millor que entrar en aquest mon llegint dos articles ***clàssics***:
    - http://www.gnu.org/gnu/thegnuproject.es.html
    - http://www.sindominio.net/biblioweb/telematica/catedral.html

  (als dos articles s'utilitza el significat originari de la paraula hacker - que no us porti a confusió!).

Introduir les sentències SQL des de la cònsola del sistema no és gens còmode, hi han alternatives?

- Si. D'entre elles destaquem la utilització de l'editor SQL SQuirreL. Aquest es pot utilitzar amb el PostgreSQL i altres SGBD (només s'ha de configurar la cadena de connexió), és multi-plataforma (al estar fet en Java, funciona tant en Windows, com en Macintosh, com en Linux, etc), és força còmode, i també és Programari Lliure!

Puc exportar les dades i/o el disseny de les taules?

- Si. Amb instruccions especifiques (pg_dump) es poden exportar les dades i les definicions de les taules, procediments i disparadors, a fi efecte de poder disposar de còpies de seguretat, i/o de traspassar les dades a un altres servidor.

Annex VIII. Manual d'instal.lació del programari PostgreSQL (versió Windows 9x/ME)

Manual d'instal·lació del programari PostgreSQL.

(versió Windows 9x)

Introducció

Que és PostgreSQL?

PostgreSQL és un programari de Base de Dades (BD) amb llicència de distribució tipus BSD[4], o sigui, és un programa d'ús i distribució lliure. Del mateix se'n pot obtenir el programa font per a modificar-lo, ampliar-lo, o corregir segons les pròpies necessitats.

D'entre els Sistemes de Gestió de Bases de Dades (SGBD) actuals, destaca per la seva estabilitat, possibilitats i versatilitat, comptant amb característiques que fins i tot alguns SGBD comercials no incorporen a l'actualitat. L'haver estat desenvolupat des d'un inici en un entorn acadèmic (Universitat de Berkeley – 1974), fa que estigui concebut com a una plataforma d'exhibició de *l'estat del art* de la tecnologia en aquesta branca del programari. Al seu inici era una demostració del que haurien de ser els SGBDs futurs (incorporant per tant opcions que 'comercialment' no eren desenvolupables segons un estricte criteri de beneficis empresarials). Actualment encara destaca per tenir característiques molt avançades, que difícilment es troben en altres sistemes de la seva categoria (per exemple: el subsistema GisT, que és utilitzat per a facilitar l'emmagatzemament i indexació de polígons tridimensionals en sistemes cartogràfics i topogràfics).

El constant creixement de la popularitat del programari lliure (tipus GNU, GPL o similars), fa que sigui imprescindible tenir una coneixença profunda d'aquests entorns per a desenvolupar els aplicatius que demanda el mercat actual. La constant aparició d'empreses que ofereixen suport al programari lliure (IBM, Novell, SUN, entre d'altres), fa que s'esvaeixin els dubtes de si aquest tipus de programari és viable en un futur, ja que en aquest nou paradigma es traslladen les inversions inicials de compra, al manteniment i a millors desenvolupaments.

La disponibilitat del codi font dels aplicatius amb llicència BSD, fa que no es depengui del fabricant del programari per la continuïtat del producte. Si el fabricant desaparegués, la inversió feta en el desenvolupament del programari propi (sobre un SGBD concret per exemple) estaria assegurada, ja que aquest *motor de base de dades* seguiria tenint el manteniment d'una comunitat d'usuaris (distribuïda probablement per tot el mon), que estaria interessada en solucionar els problemes que sorgissin, i fins i tot de continuar la seva evolució. També s'ha de considerar que el desenvolupament dels mòduls d'idioma o altres característiques que poden no ser rentables per un fabricant concret, pot ser desenvolupats pels 'tercers' que estiguin interessats.

[4] Existeixen altres BDs que es mouen a l'entorn del programari lliure, però que no ho són realment. MySQL per exemple, té llicència dual (gratuït per a ús privat i de pagament en usos comercials/empresarials).

Instal·lació de PostgreSQL (Windows 95/98/ME)

La instal·lació de la versió del PostgreSQL per al Windows 95, 98 i ME consisteix bàsicament en descomprimir un fitxer que conté un emulador de Linux (Cygwin) i una còpia feta 'en fred' de la instal·lació d'un PostgreSQL 7.04 (amb una base de dades i una taula de prova ja creades).

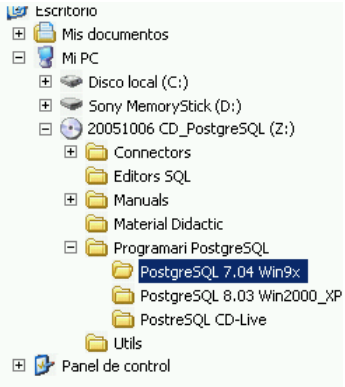
Requeriments de maquinari:
Punt de treball bàsic recomanat, segons especificacions UOC.

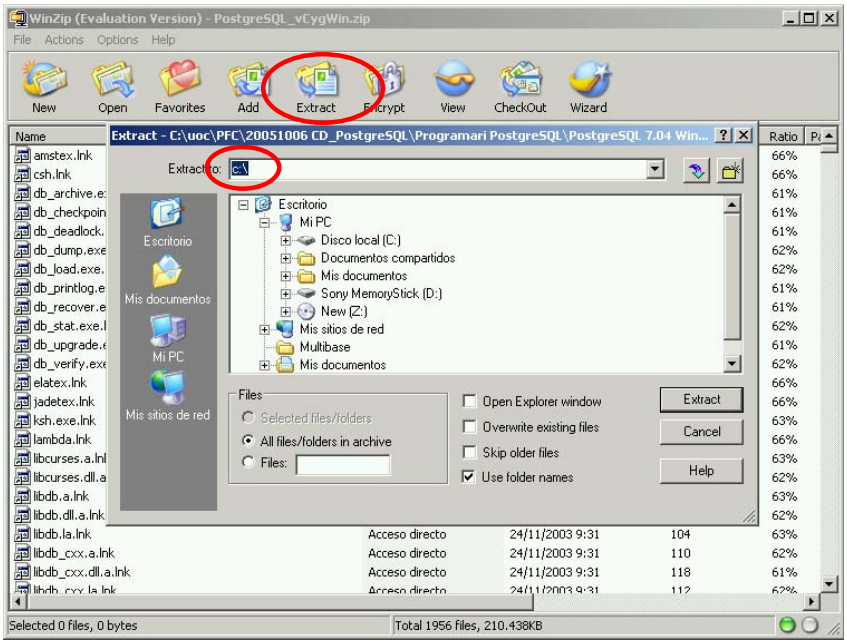
Requeriments de programari:
Utilització del Windows 95 (4.00.950), 98 (4.10.2222) o ME (4.90.3000).
Tenir instal·lat el programari Winzip o equivalent (de no tenir-lo, es pot instal·lar el que hi ha al directori *Utils* del CDROM).

Com que és impossible assegurar una compatibilitat total amb qualsevol altra tipus de programari instal·lat, durant la instal·lació i les proves inicials s'aconsella tenir des-instal.lat qualsevol programa que pugui interceptar les crides de xarxa (limitadors d'ampla de banda, antivirus, firewalls, etc).
Una vegada comprovat el bon funcionament del PostgreSQL es pot tornar a instal·lar el programari de seguretat que es cregui oportú (mentre no estigui aquest instal·lat, no s'hauria de permetre que l'ordinador es connectés a internet!).

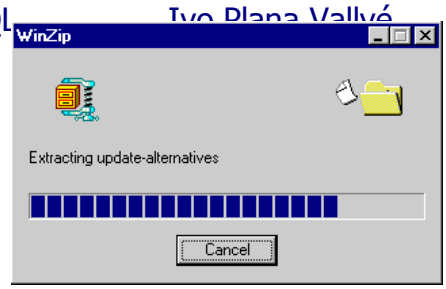
Instal·lació:

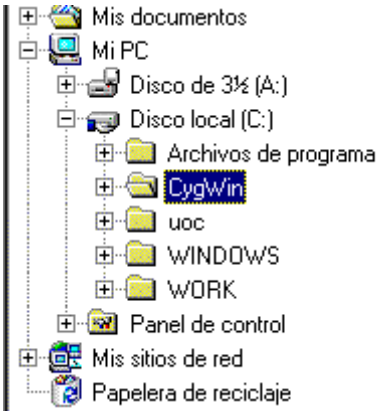
Accedirem al directori *PostgreSQL 7.04 Win9x* del CDROM lliurat conjuntament amb els mòduls de l'assignatura, i obrirem amb un doble-clic de ratolí el fitxer **PostgreSQL_vCygWin.zip**. Un cop fet això, s'obrirà el programa descompressor de fitxers zip que l'ordinador tingui associat a aquesta extensió.
En la finestra del Winzip (si no s'utilitza aquest programa, s'haurà de fer l'acció equivalent), s'haurà de elegir *Extract*.
El directori de destinació dels fitxers, ha de ser el 'c:\' o sigui, l'arrel del disc dur C.





L'estat del procés d'extracció dels fitxers s'indicarà mitjançant una barra de progrés.

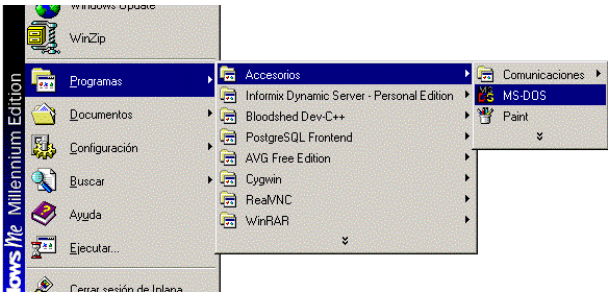




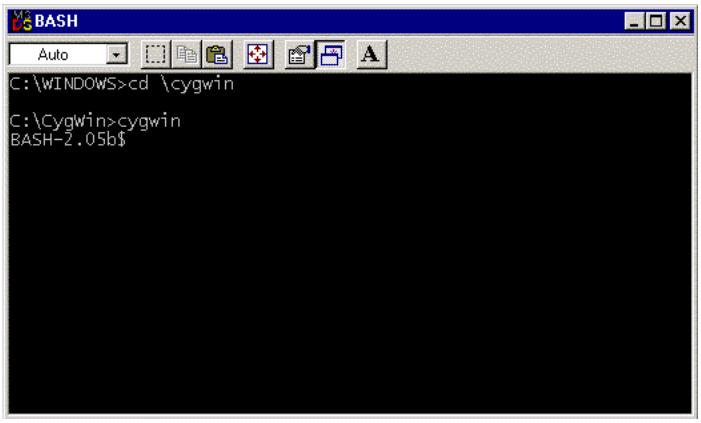
Una cop acabat el procés, trobarem el directori *Cygwin* a l'arrel del dics dur C.

Haurem de buscar al interior d'aquest directori el fitxer *cygwin.bat*, i executar-lo.

En el cas de que en obrir-se la pantalla del MSDOS, indiqui 'sense espai d'entorn' (o equivalent), tancarem la finestra, i obrirem el MSDOS des del menú *Inici* del Windows.



Una vegada al DOS, s'ha de canviar de directori (amb: cd \cygwin), i s'ha de carregar el programar *cygwin* (tal com es veu a les primeres línies de la imatge de la dreta).



En aquest punt, cal recordar que aquesta cònsola del sistema, és una emulació d'un shell de Linux, pel que si cal fer alguna operació, haurem d'utilitzar les seves comandes i sintaxis, però res impedeix seguir utilitzant l'explorador del windows per accedir al directori cygwin!

Prova de funcionament.

La posta en marxa del PostgreSQL consta de tres passos.

a) Engegar un 'dimoni' per afegir serveis de compatibilitat amb el Linux.
Ho farem teclejant a la finestra del shell del Cygwin, la comanda:

```
/bin/ipc-daemon2 &
```

(és important utilitzar les barres correctes, i no oblidar l'espai abans de posar el caràcter '&').

b) Engegar el motor de la base de dades. Teclejarem directament a la cònsola del shell:

```
/bin/postmaster –i –D /home/data
```

```
BASH-2.05b$ /bin/ipc-daemon2 &
[1] 636631
BASH-2.05b$ /bin/postmaster -i -D /home/data
LOG:  shmdt(0x845a6000) failed: Invalid argument
LOG:  database system was shut down at 2005-09-30 22:57:19
LOG:  checkpoint record is at 0/A01A70
LOG:  redo record is at 0/A01A70; undo record is at 0/0; shutdown TRUE
LOG:  next transaction ID: 543; next OID: 17146
LOG:  database system is ready
```

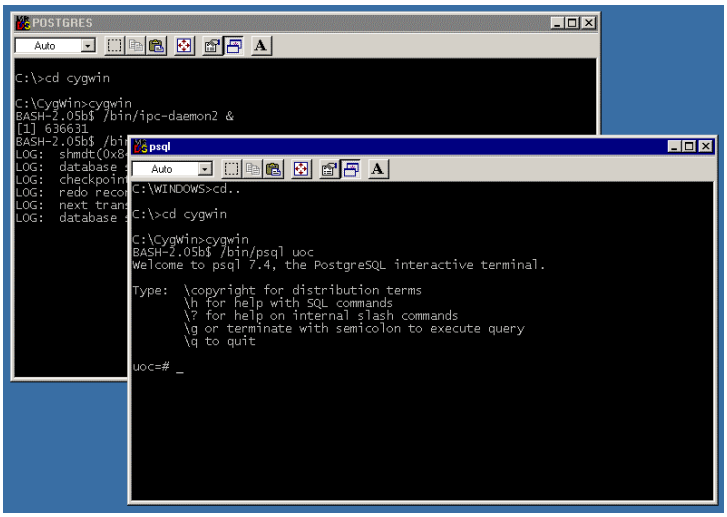
En aquest punt comprovarem que desprès d'executar la instrucció anterior, el cursors no 'torna'. Això és degut a que el servidor de l'SGBD queda permanentment en execució, a l'espera de rebre comandes SQL per la xarxa (malgrat totes les operacions les fem des del mateix ordinador, internament es comunica igual que si rebés peticions de *l'exterior*).

Per tal cosa, obrirem una nova finestra amb el cygwin, per a executar el pas C.

c) Des d'una nova finestra del Cygwin, obrirem la cònsola SQL amb la comanda:

```
/bin/psql uoc
```

A l'escriptori ens quedaran dues finestres obertes (una amb el 'motor de la base de dades' i l'altra amb l'editor SQL), segons es pot observar.



Des de la finestra de l'editor, introduirem la següent consulta SQL, per a comprovar que tot s'ha instal·lat com s'esperava:

```
select * from test;
```

Ens té que tornar el resultat que s'observa en la captura de pantalla següent:

```
uoc=# select * from test;
    nom
-----------
 ok !!!!!!
(1 row)

uoc=#
```

No busqueu d'on surten aquesta valors! (ja que van inclosos amb la instal·lació que s'ha extret del fitxer zip!)

Resolució de problemes i FAQ

Quins requeriments de maquinari té el PostgreSQL per a Windows 9x?

- Tal com l'utilitzem a les assignatures de l'àrea de BD, es pot executar amb el maquinari bàsic recomanat per la UOC. Un ordinador amb processador x86 (tipus Pentium, Athlon o similar), capaç d'executar el windows sobre el que s'instal·la el programari, amb uns 210Mb d'espai de disc lliure, el podrà executar sense problemes.

PostgreSQL funciona en diversos sistemes operatius?

- Si. Al menys funciona en una dotzena de sistemes operatius diferents (podeu trobar aquest mateix manual d'instal·lació pel Win95/98, Windows 2000/XP i pel Linux). De fet, al tenir PostbreSQL llicència de distribució de tipus BSD, res impedeix compilar les fonts pel sistema operatiu que ens interessi (encara que per a això, cal estar bastant especialitzat).

Com pot ser que PostgreSQL sigui gratuït, si té característiques superiors al Informix, i fins i tot similars al Oracle?

- La resposta és senzilla, però complexa de respondre. Com que part de *l'encant* que envolta al programari lliure és la investigació, la recerca constant de millores i fins i tot la superació personal, res millor que entrar en aquest mon llegint dos articles ***clàssics***:
    - http://www.gnu.org/gnu/thegnuproject.es.html
    - http://www.sindominio.net/biblioweb/telematica/catedral.html

  (als dos articles s'utilitza el significat originari de la paraula hacker - que no us porti a confusió!).

Puc exportar les dades i/o el disseny de les taules?

- Si. Amb instruccions especifiques (pg_dump) es poden exportar les dades i les definicions de les taules, procediments i disparadors, a fi efecte de poder disposar de còpies de seguretat, i/o de traspassar les dades a un altres servidor.

Annex IX. Manual d'instal.lació del programari PostgreSQL (versió Windows 2000/XP)

Manual d'instal·lació del programari PostgreSQL.

(versió Windows 2000/XP)

Introducció

Que és PostgreSQL?

PostgreSQL és un programari de Base de Dades (BD) amb llicència de distribució tipus BSD[5], o sigui, és un programa d'ús i distribució lliure. Del mateix se'n pot obtenir el programa font per a modificar-lo, ampliar-lo, o corregir segons les pròpies necessitats.

D'entre els Sistemes de Gestió de Bases de Dades (SGBD) actuals, destaca per la seva estabilitat, possibilitats i versatilitat, comptant amb característiques que fins i tot alguns SGBD comercials no incorporen a l'actualitat. L'haver estat desenvolupat des d'un inici en un entorn acadèmic (Universitat de Berkeley – 1974), fa que estigui concebut com a una plataforma d'exhibició de *l'estat del art* de la tecnologia en aquesta branca del programari. Al seu inici era una demostració del que haurien de ser els SGBDs futurs (incorporant per tant opcions que 'comercialment' no eren desenvolupables segons un estricte criteri de beneficis empresarials). Actualment encara destaca per tenir característiques molt avançades, que difícilment es troben en altres sistemes de la seva categoria (per exemple: el subsistema GisT, que és utilitzat per a facilitar l'emmagatzemament i indexació de polígons tridimensionals en sistemes cartogràfics i topogràfics).

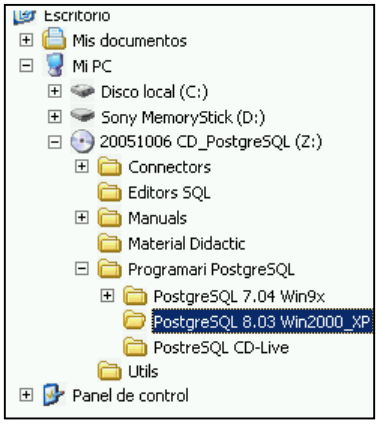
El constant creixement de la popularitat del programari lliure (tipus GNU, GPL o similars), fa que sigui imprescindible tenir una coneixença profunda d'aquests entorns per a desenvolupar els aplicatius que demanda el mercat actual. La constant aparició d'empreses que ofereixen suport al programari lliure (IBM, Novell, SUN, entre d'altres), fa que s'esvaeixin els dubtes de si aquest tipus de programari és viable en un futur, ja que en aquest nou paradigma es traslladen les inversions inicials de compra, al manteniment i a millors desenvolupaments.

La disponibilitat del codi font dels aplicatius amb llicència BSD, fa que no es depengui del fabricant del programari per la continuïtat del producte. Si el fabricant desaparegués, la inversió feta en el desenvolupament del programari propi (sobre un SGBD concret per exemple) estaria assegurada, ja que aquest *motor de base de dades* seguiria tenint el manteniment d'una comunitat d'usuaris (distribuïda probablement per tot el mon), que estaria interessada en solucionar els problemes que sorgissin, i fins i tot de continuar la seva evolució. També s'ha de considerar que el desenvolupament dels mòduls d'idioma o altres característiques que poden no ser rentables per un fabricant concret, pot ser desenvolupats pels 'tercers' que estiguin interessats.

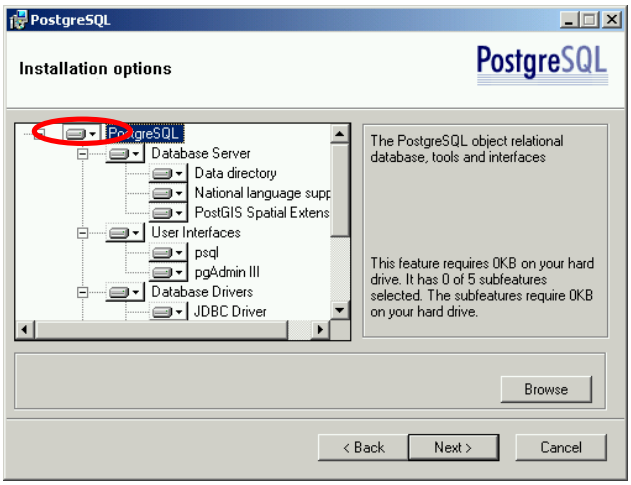
[5] Existeixen altres BDs que es mouen a l'entorn del programari lliure, però que no ho són realment. MySQL per exemple, té llicència dual (gratuït per a ús privat i de pagament en usos comercials/empresarials).

Instal·lació de PostgreSQL (Windows 2000/XP)

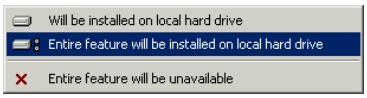
Per a començar haurem d'obtenir el programa executable d'instal·lació. Aquest el trobarem al CDROM de Programari Lliure (segons imatge adjunta), o alternativament es pot descarregar des de pestanya *Recursos* de l'aula de l'assignatura (la versió complerta ocupa uns 18Mb aproximadament).
Un cop tinguem el fitxer postgresql-8.0.3.zip, el descomprimirem a una carpeta del disc dur de l'ordinador.
Per motius de compatibilitat els dos fitxers de tipus msi (instal·lables), fan referència en el seu nom a la versió 8.0.
Abans d'executar el fitxer **postgresql-8.0.msi**, i per a evitar que falli la instal·lació, ens assegurarem de desactivar el firewall de l'ordinador (desconnectant-nos d'internet si cal!).

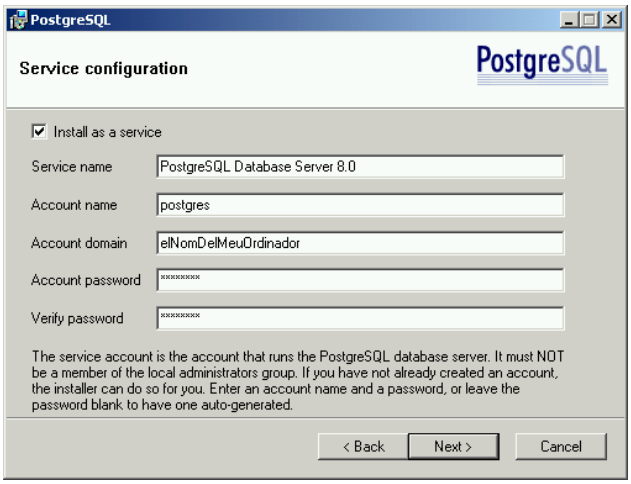


L'assistent del programa d'instal·lació ens donarà la benvinguda tot deixant-nos elegir l'idioma desitjat (de les opcions previstes). A la següent pantalla se'ns indicarà que és preferible tancar totes les aplicacions que estiguin obertes.
A la tercera pantalla ens apareix l'acord de llicencia. Desprès de pitjar sobre el botó Next, se'ns convidarà a elegir els mòduls que volem instal·lar;



Clicarem amb el ratolí sobre la 'caixa' superior (amb nom PostgreSQL), i elegirem del menú contextual que apareix l'opció: *Entire feature will be installed on local hard drive.*

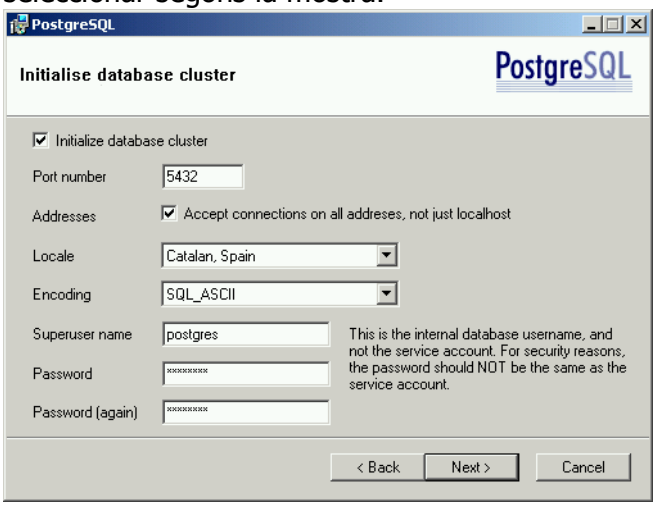




En pitjar el botó Next, ens trobarem a la finestra *Service Configuration*.

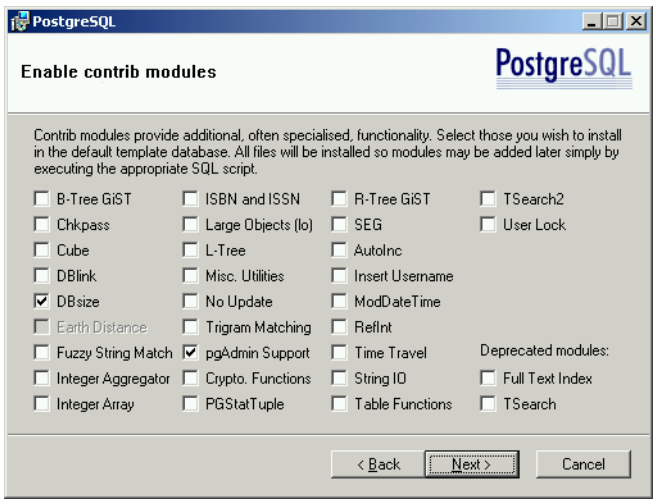
En aquesta ens apareixerà el nom del nostre ordinador a la casella *Account Domain* (ens hem d'assegurar que estigui bé). També se'ns demanarà que introduïm la contrasenya que voldrem que s'utilitzi per a engegar el servei del windows que executarà el motor de la base de dades (per a unificar instal·lacions, es recomana utilitzar com a contrasenya: **postgres**)

Les opcions de la pantalla de la configuració de la base de dades, les podem seleccionar segons la mostra.



La marca de l'opció *Accept connections on all addreses, not just localhost*, serveix per que puguem accedir a la base de dades des d'altres ordinadors (si tenim xarxa local), o fins i tot des d'internet.
Malgrat no ho recomanin, seguirem utilitzant (a fi efecte d'homogeneïtzar les instal·lacions) la contrasenya **postgres** (poseu una de més complexa si esteu utilitzant un ordinador compartit, o connectat a una xarxa).

Al tornar a pitjar sobre el botó Next, ens apareixerà la pantalla titulada *Enable procedural languages.* Clicarem de nou sobre el botó Next, un cop comprovat que el llenguatge PL/pgsql està marcat (opció per omissió).



A la pantalla següent tindrem oportunitat d'elegir els mòduls que volem que s'instal·lin conjuntament amb la base de dades.

Malgrat no marquem cap d'addicional (pitjarem Next amb les opcions que hi han marcades per omissió), cal destacar de les possibilitats que se'ns ofereixen:

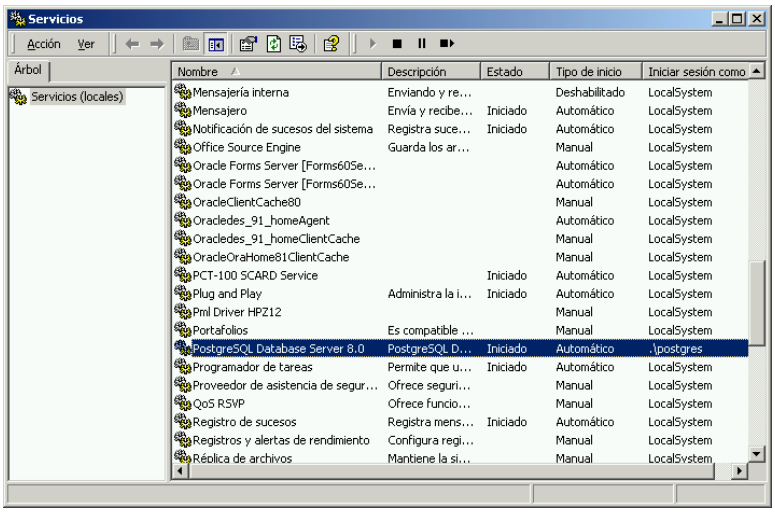
- Time travel: D'instal·lar aquesta opció, s'emmagatzemen automàticament les versions de les dades que hi han hagut al registres de la BD, podent-se fer una consulta de dades anteriors (sobre-escrites). Amb això podem consultar dades per les que no s'havia prevista guardar historial de canvis: Exemple de consulta: *Per l'aplicació multi-idioma de Comptabilitat, retorna l'idioma que estaven utilitzant els usuaris el 5 de gener de 2004 a les 14 hores.*
- R-Tree GisT: Tal com s'ha comentat, afegeix la facilitat d'emmagatzemament (i indexat) de figures geomètriques. Molt utilitzat en entorns cartogràfics, topogràfics, de CAD/CAE, etc

Un cop pitgem sobre el botó Next, l'ordinador passarà a completar la instal·lació tot copiant els programes al disc dur. Durant aquest procés se'ns mostrarà una barra de progrés (pot trigar uns minuts).

En acabar se'ns mostrarà la finestra final, amb el missatge d'***Installation complete!***

Provant la instal·lació de PostgreSQL

Abans de fer una prova de funcionament inicial, haurem de comprovar que el servei de l'SGBD s'ha engegat correctament.

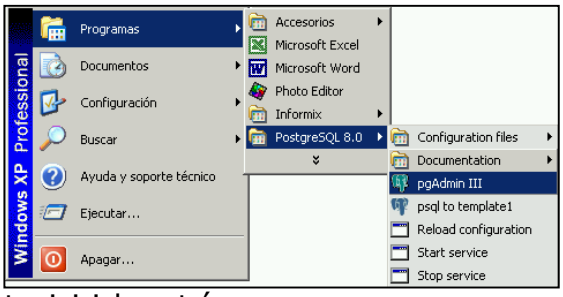


Des del *Panel de Control* del Windows, obrirem *Herramientas Administrativas*. Des d'aquí obrirem *Servicios*, i comprovarem que està *Iniciado*, i que el tipus d'inici està en *Automático*.

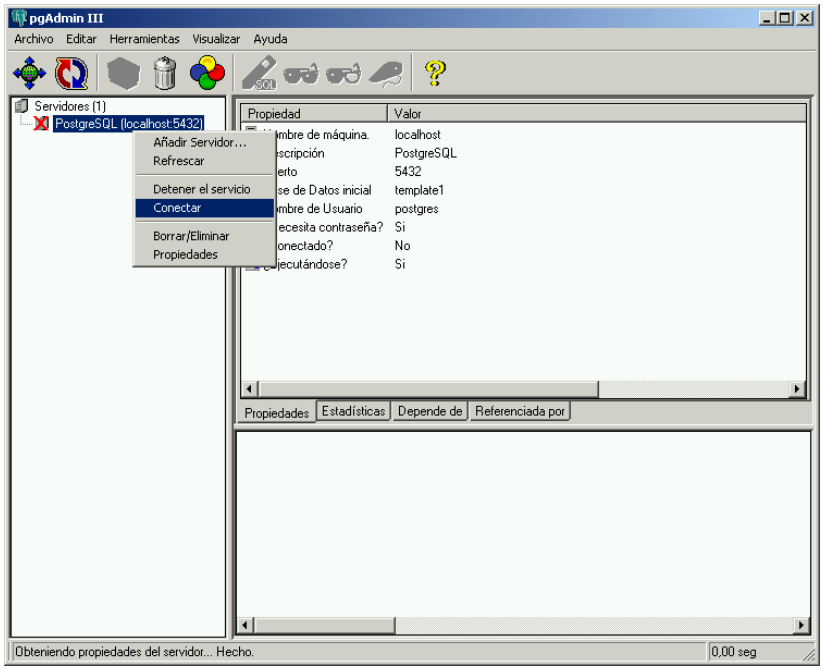
Des d'aquest mateix punt de control del servei, podrem canviar si ens cal (en obrir la finestra del servei concret amb un doble clic) el tipus d'inici, la contrasenya, i altres opcions més específiques.

Un cop finalitzada la instal·lació i comprovacions, ha arribat el moment de provar-ho!

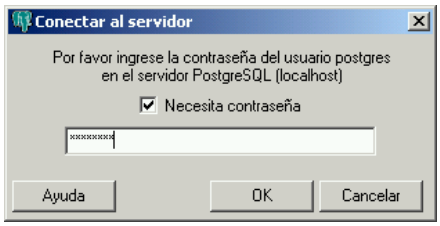
Engegarem la cònsola d'administració de la base de dades (*pgAdmin III*) amb la icona corresponent;



La pantalla que se'ns mostra inicialment és:

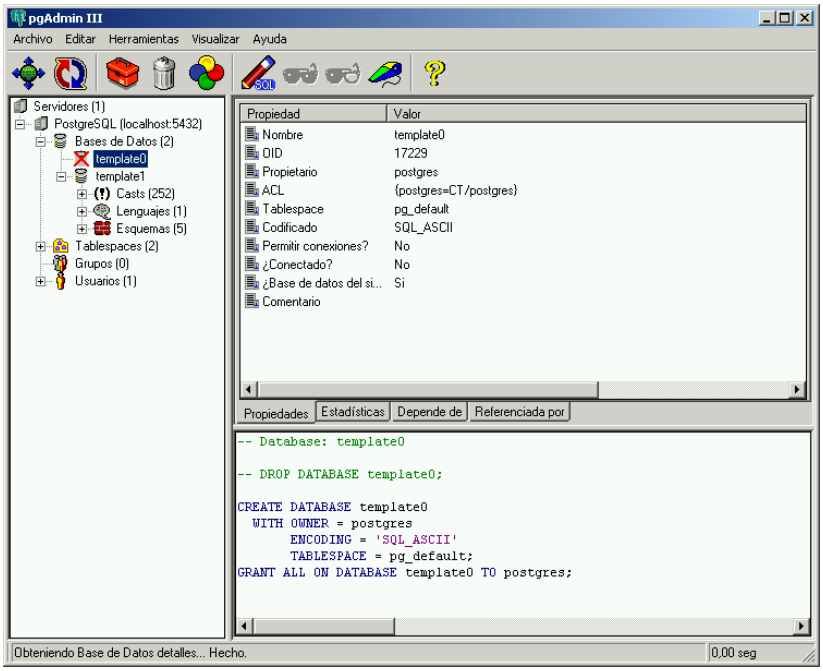


Només engegar observarem una creu vermella a la part de l'esquerra, a la branca de PosgreSQL (localhost:5432). Aquesta indica que l'eina d'administració no està connectada a la base de dades. Pitjarem amb el botó contrari del ratolí sobre la línia que té aquesta creu vermella, i elegirem l'opció Conectar.

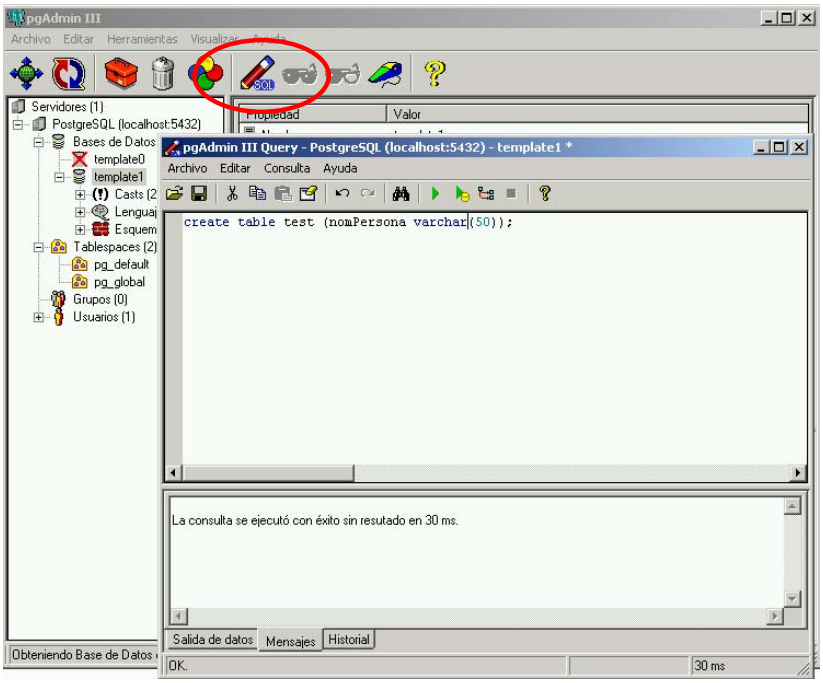


Un cop introduïda la contrasenya postgres, ens apareixerà la configuració de la BD, i se'ns il·luminaran les eines disponibles.

Ens hauríem de familiaritzar amb l'entorn, tot comprovant els ajuts contextuals que surten en posar-nos damunt de cada icona.

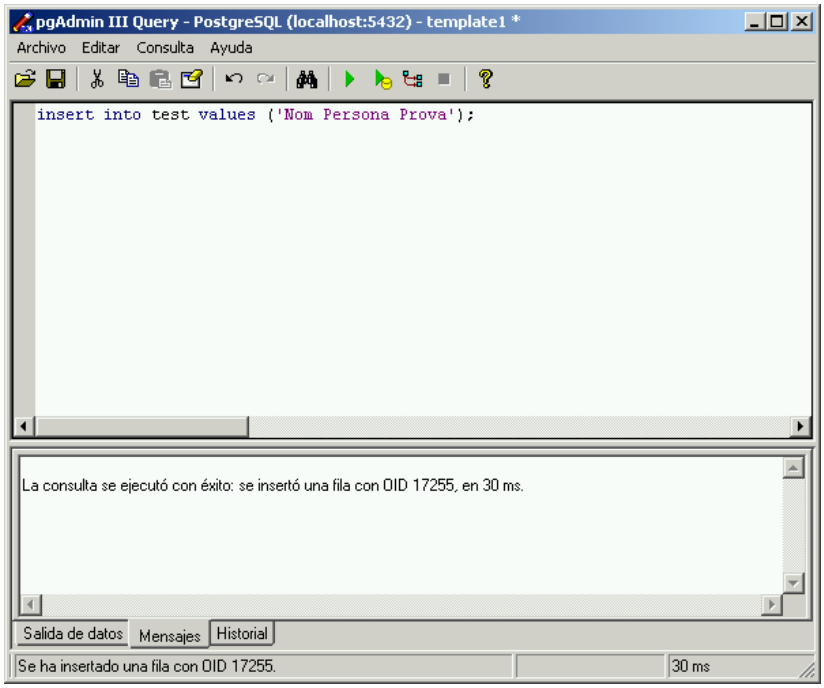


Per a crear una taula en primer lloc haurem de seleccionar el primer template disponible (en aquest cas el template1). Tot seguit obrirem l'editor SQL amb la icona superior amb forma de llapis.

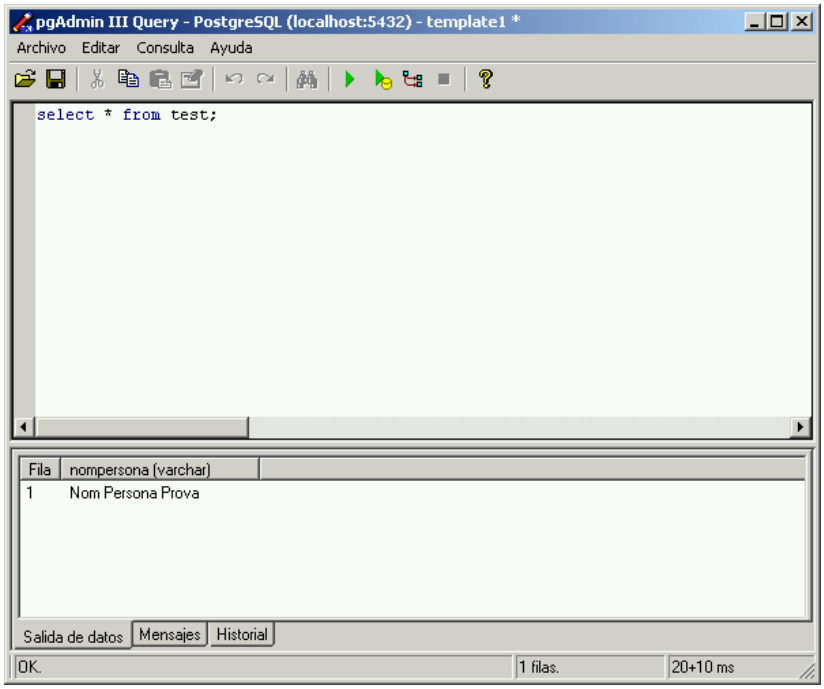


Provarem de crear la primera taula amb la instrucció :
create table test (nomPersona varchar(50));
(els resultats de les execucions sempre apareixen a la part inferior de la finestra SQL).

Tot seguit, provem de fer una inserció de dades:



La comprovació final consistirà en tancar el programa *pgAdmin III*, reiniciar l'ordinador, i tornar consultar les dades recent inserides.



Resolució de problemes i FAQ

En fer la instal·lació, diu que no pot continuar perquè el disc dur no és NTFS. Que puc fer?

- El format nadiu d'emmagatzemament de dades dels sistemes operatius Windows 2000 i XP és el format NTFS. El vell FAT32 es manté (i permet utilitzar) per motius de compatibilitat. El format NTFS permet assignar privilegis als directoris i a les dades, comprimir o encriptar carpetes, tenir control de l'espai utilitzat per a cada usuari (quotes), i altres moltes avantatges.
- Per varis motius, PostgreSQL requereix treballar sobre un disc dur que tingui format NTFS. Per a convertir una partició de FAT32 a NTFS, senzillament s'ha de posar una comanda en el *Simbolo del Sistema* de l'ordinador (i reiniciar-lo):

  CONVERT C: /FS:NTFS

  (la conversió és un procés segur, i no esborra cap dada del disc dur).

Puc tenir algun problema si converteixo la meva partició a NTFS?

- Normalment cap. Només en el cas d'utilitzar algun programa molt vell (i rar, i mal fet), o tenir altres sistemes operatius instal·lats a l'ordinador (una versió de Linux vella a una altra partició per ex.), pot ser que no es pugui accedir a la nova partició en format NTFS, sense aplicar abans l'*afegit* corresponents al Linux o l'actualització corresponent al programa, per assegurar la compatibilitat.

Quins requeriments de maquinari té el PostgreSQL?

- Tal com l'utilitzem a les assignatures de l'àrea de BD, es pot executar amb el maquinari bàsic recomanat per la UOC. Un ordinador amb processador x86 (tipus Pentium, Athlon o similar), capaç d'executar el windows XP sobre el que s'instal·la el programari, amb uns 100Mb d'espai de disc lliure (per la instal·lació bàsica, comptant la reserva d'espai per la base de dades), el podrà executar sense problemes.
- Si penseu en un ús empresarial (amb desenes o centenars de connexions), caldrà que el servidor disposi de discs durs SCSI (preferiblement en RAID 5), amb almenys 1Gb de memòria RAM, i preferentment amb un parell (o més) de nuclis de processament (s'ha de dimensionar segons el volum de transaccions que tingui que suportar – i això no és pas senzill!).

PostgreSQL funciona en diversos sistemes operatius?

- Si. Al menys funciona en una dotzena de sistemes operatius diferents (podeu trobar aquest mateix manual d'instal·lació pel Win95/98 i pel Linux). De fet, al tenir PostbreSQL llicència de distribució de tipus GPL, res impedeix compilar les fonts pel sistema operatiu que ens interessi (encara que per a això, cal estar bastant especialitzat).

Com pot ser que PostgreSQL sigui gratuït, si té característiques superiors al Informix, i fins i tot similars al Oracle?

- La resposta és senzilla, però complexa de respondre. Com que part de *l'encant* que envolta al programari lliure és la investigació, la recerca constant de millores i fins i tot la superació personal, res millor que entrar en aquest mon llegint dos articles ***clàssics***:
  - http://www.gnu.org/gnu/thegnuproject.es.html
  - http://www.sindominio.net/biblioweb/telematica/catedral.html

  (als dos articles s'utilitza el significat originari de la paraula hacker - que no us porti a confusió!).

La finestra que s'utilitza en el pgAdmin III per a introduir consultes SQL no és gens còmoda, hi han alternatives?

- La interfície d'introducció de sentències SQL que porta el pgAdmin és bàsicament per a fer consultes de manteniment o prova. Hi han moltes altres eines per a fer consultes sobre les BDs, d'entre elles cal destacar l'SQuirreL, ja que funciona amb diversos SGBD (només s'ha de configurar la cadena de connexió JDBC), és multi-plataforma (al estar fet en Java, funciona tant en Windows, com en Macintosh, com en Linux, etc), és força còmode, i també és Programari Lliure!

Puc exportar les dades i/o el disseny de les taules?

- Si. Amb instruccions especifiques (pg_dump) es poden exportar les dades i les definicions de les taules, procediments i disparadors, a fi efecte de poder disposar de còpies de seguretat, i/o de traspassar les dades a un altres servidor.

Annex X. Guia d'instal·lació del programari SQuirreL (versió Windows 2000/XP)

Guia d'instal·lació del programari SQuirreL.

(versió Windows 2000/XP)

Introducció

Que és SQuirreL SQL?

SQuirreL és un client per a executar consultes SQL sobre Sistemes de Gestió de Bases de Dades (SGBD), dissenyat sota conceptes moderns. Aquest fet fa que sigui independent d'un SGBD concret, que tingui una interfície gràfica àgil, i que sigui independent tant del Sistema Operatiu (SO) com del maquinari sobre el que s'executa.

SQuirreL SQL és independent del fabricant de l'SGBD, donat que està dissenyat específicament per a que es connecti amb aquests mitjançant els ponts JDBC/ODBC existents.

És independent del sistema operatiu i del maquinari donat que està programat en Java, pel que només cal tenir la maquina virtual (MV) adequada instal·lada per a que qualsevol ordinador sigui plenament compatible.

El programari té llicència del tipus GNU Lesser General Public License (en la pràctica és la GPL), pel que es poden obtenir les fonts del programa original per a corregir/millorar, i tornar a compilar. En ser GNU, no hi ha cap problema per a tornar a copiar/distribuir el programa original o el modificat.

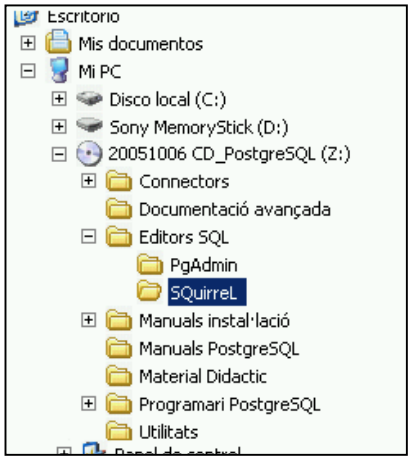
Instal·lació d'Squirrel SQL (Windows 2000/XP)

Instal·lació del programa.

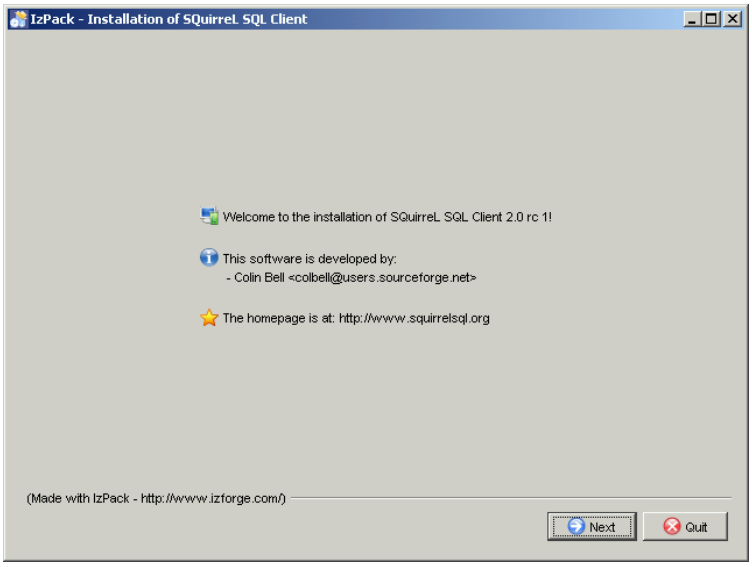
El primer pas és obtenir el programa executable d'instal·lació. Aquest el trobarem al CDROM lliurat amb el material de l'assignatura (segons imatge adjunta), o alternativament es pot descarregar des de la pestanya *Recursos* de l'aula (la versió complerta ocupa uns 7Mb aproximadament).

Un cop tinguem el fitxer squirrel-sql-2.0final-install.jar, l'executarem fent un 'doble clic' amb el ratolí sobre aquest, o col·locant-se en el directori on estigui el fitxer des de la finestra de *Simbolo del Sistema*, i teclejant:

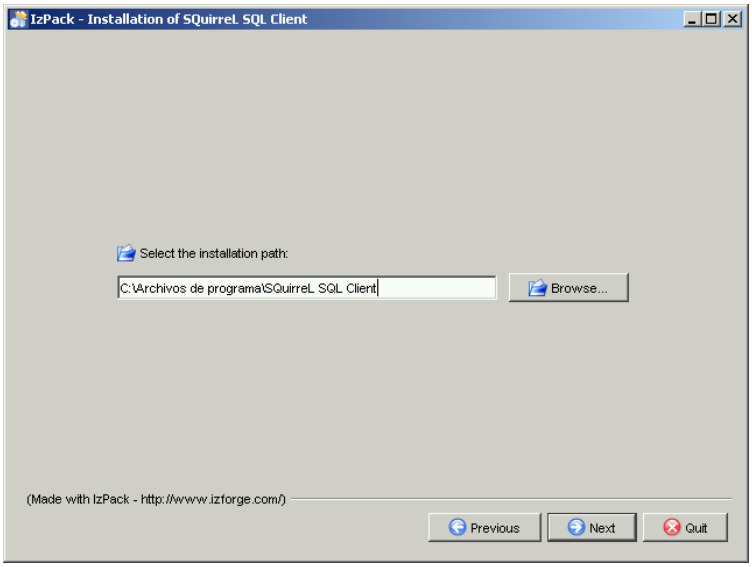
java -jar squirrel-sql-2.0final-install.jar



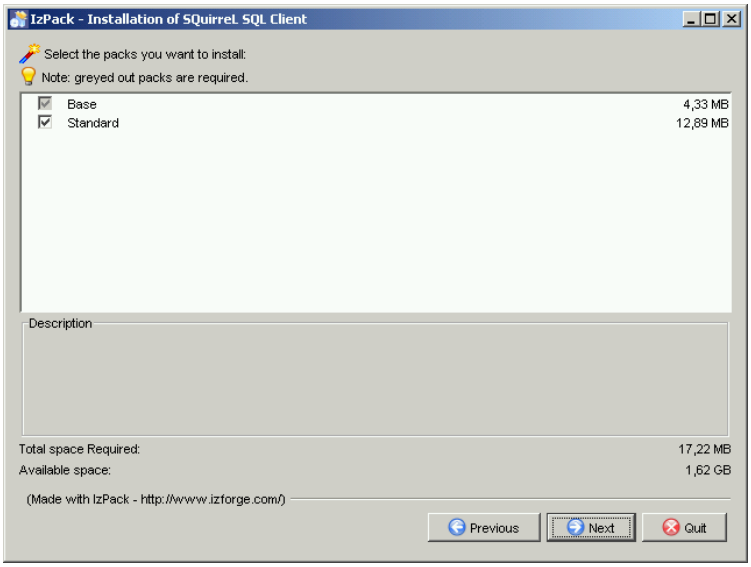
La pantalla de benvinguda que apareixerà és la següent:



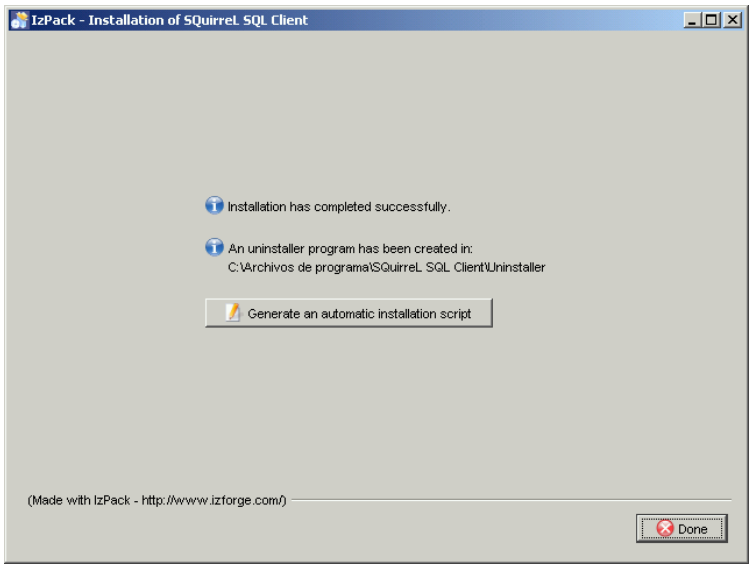
En pitjar al botó de *Next*, ens sortirà la pantalla de presentació del programa, en tornar a fer *Next* ens sortirà la finestra que ens indica que el programa té una llicència de tipus GNU, i que per a instal·lar-ho l'hem d'acceptar.



A la següent pantalla se'ns proposarà un camí d'instal·lació per omissió. Sense canviar-lo, pitjarem sobre el botó *Next*.



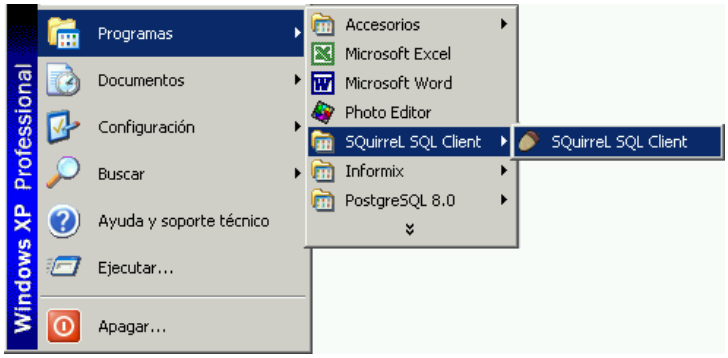
Els components triats a la instal·lació bàsica, són tots els que hi han al 'paquet', pel que només haurem de pitjar sobre *Next*.



Tot seguit es descomprimiran sobre el disc dur els fitxers necessaris.

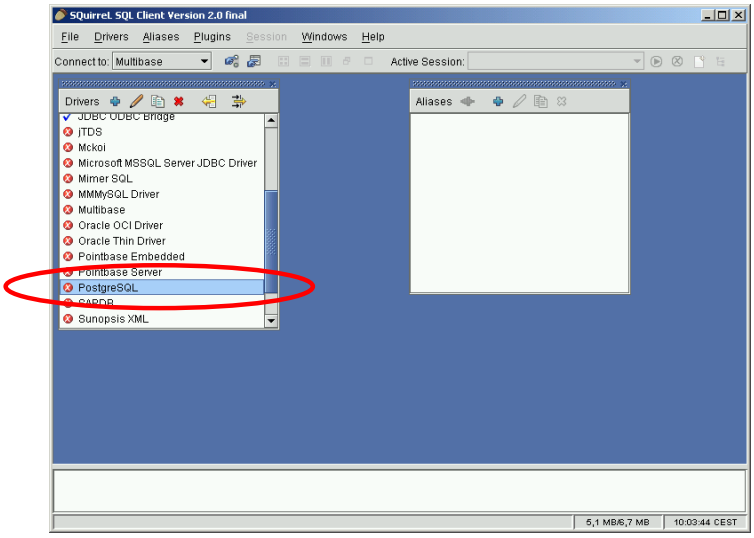
Quan les barres de progrés hagin arribat al 100%, ens apareixerà la pantalla que ens indica que la instal·lació ha acabat.

En fer la instal·lació sobre Windows, el programa crea automàticament les icones d'accés directe a l'escriptori i en el grup de programes.

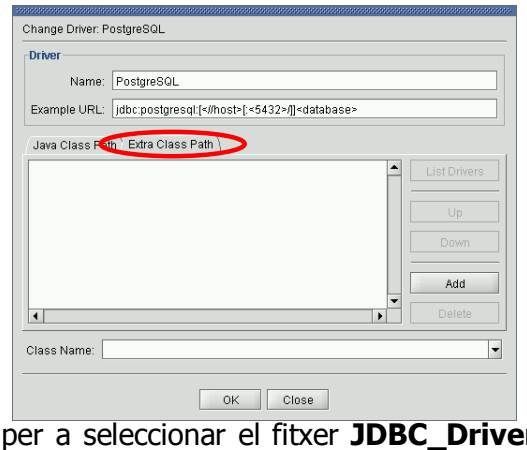


Configuració de la connexió JDBC.

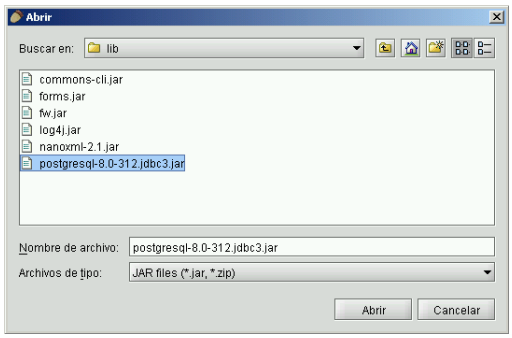
Una vegada tenim el programa instal·lat, cal preparar l'entorn per a que quan aquest busqui el 'pont' JDBC per l'SGBD PostgreSQL, el pugui trobar.
En engegar inicialment l'SQuirreL podem observar a la finestra de l'esquerra que la connexió pel PostgreSQL no està disponible.



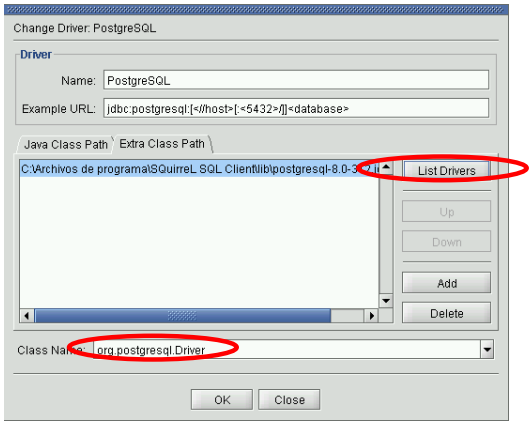
Els passos que haurem de seguir per a configurar-ho és:
- Copiar el drivers JDBC **JDBC_Driver_postgresql-8.1dev-401.jdbc3.jar** que hi ha al CDROM o a la pestanya *Recursos* de l'assignatura, al directori:
C:\archivos de programa\Squirrel SQL Client\lib
- Obrir amb un doble click la línia de PostgreSQL de la finestra **Drivers**, i anar a la pestanya **Extra Class Path**.



- Utilitzar el botó Add per a seleccionar el fitxer **JDBC_Driver_postgresql-8.1dev-401.jdbc3.jar** (al directori C:\archivos de programa\Squirrel SQL Client\lib).

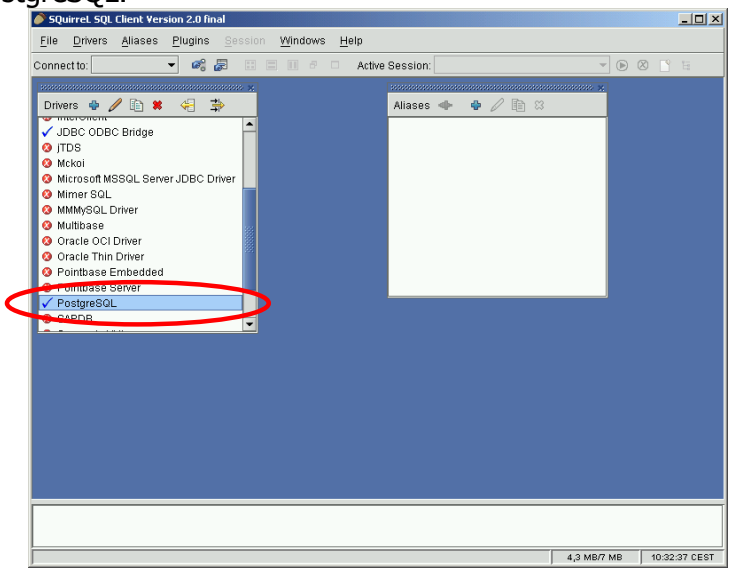


- Comprovar que a Class Name, tenim el nom del driver que hem carregat. Ha d'aparèixer: org.PostgreSQL.Driver. Si no apareix, haurem d'utilitzar el botó List Drivers per a que el busqui dins el fitxer jar del driver.

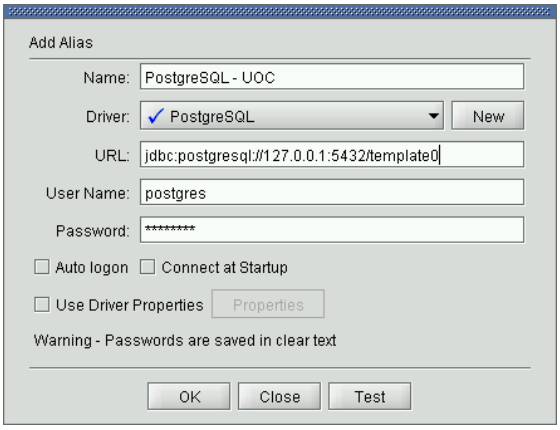


La configuració final, ha de quedar segons la imatge.

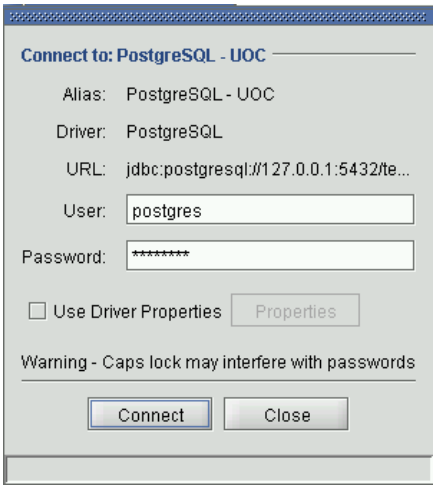
Ara, al tornar a engegar l'SQuirreL ja ha d'aparèixer la marca de 'vist' (disponible) al driver del PostgreSQL.



Com que ja tenim el 'connector' disponible, només caldrà fer la configuració amb els paràmetres del nostre *motor de base de dades* PostgresQL.
Per començar clicarem sobre la creu blava que hi ha a la part superior de la finestra **Aliases**.
L'haurem d'omplir segons la mostra;

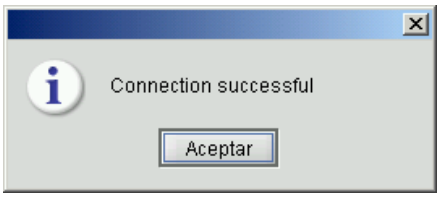


Al final de la línia URL, s'indicarà el template que hagueu vist disponible durant la instal·lació del PostgreSQL (habitualment template0 o template1).



Tot seguit pitjarem sobre el botó Test (ha d'estar el PostgreSQL instal·lat, comprovat, i en funcionament). En sortir la finestra de la dreta, pitjarem sobre el botó Connect.

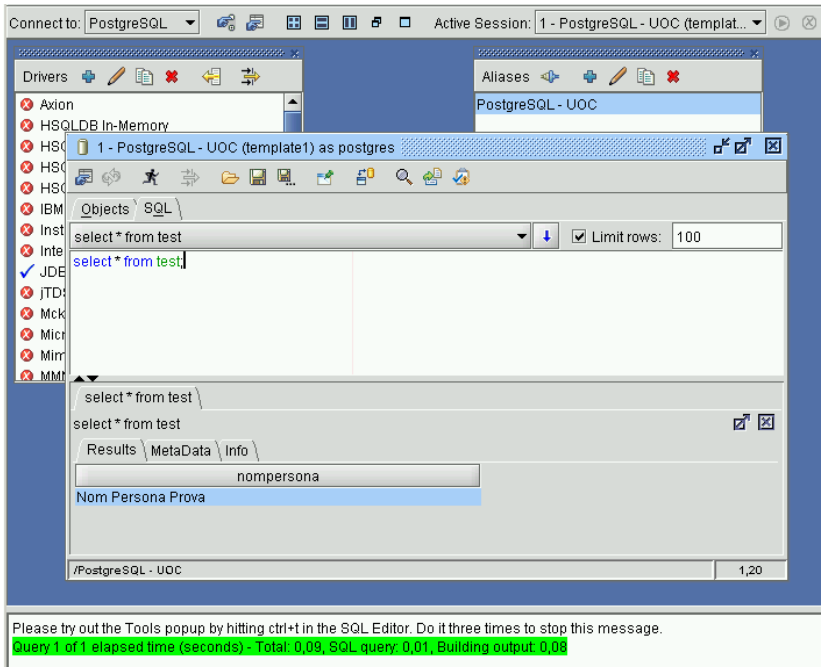
Si en surt el missatge de 'successful', haurem completat tota la instal·lació satisfactòriament.



Prova de funcionament

La prova final consisteix en fer la consulta de la taula que hem creat des del *pgAdmin III* (i no esborrat) durant la instal·lació del PostgreSQL.

La finestra d'edició SQL s'obra en fer un doble clic a la línia PostgreSQL – UOC de la finestra **Aliases**.
A la part d'edició superior, introduirem la consulta:
select * from test;



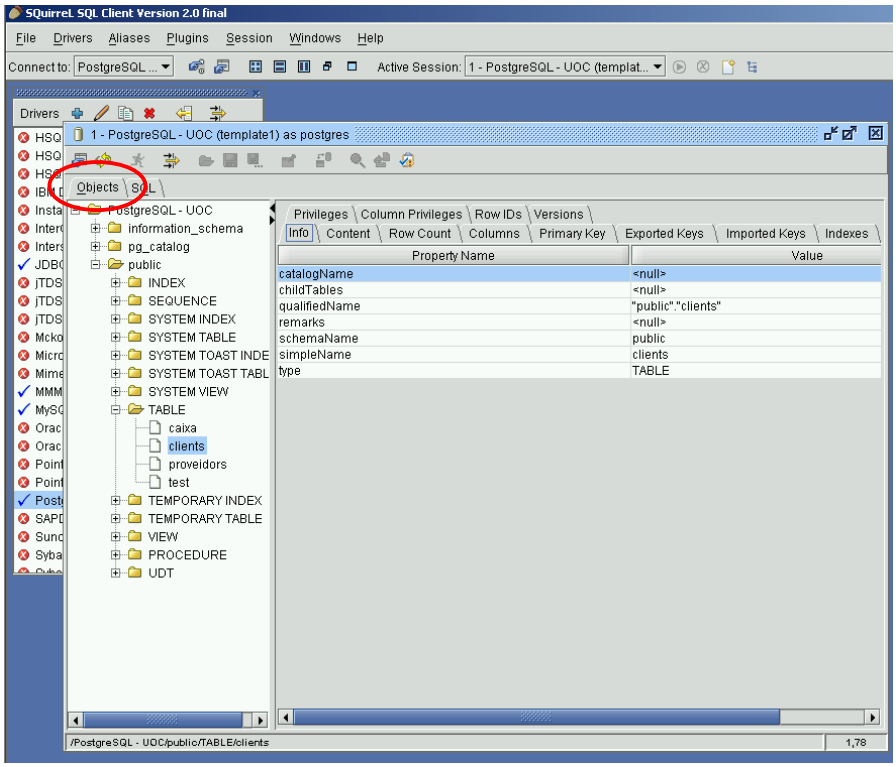
El resultat apareix a la part inferior, en la zona de pestanyes.
També podem observar les estadístiques de la execució de la consulta a la finestra d'estat de la part inferior.

Detalls de funcionament

Ara que tenim el client SQL SQuirreL instal·lat, seguirem un exemple de creació de taules i unes consultes, per a comprovar els detalls de funcionament que fan que sigui realment útil.

Partint de tenir l'SQuirreL executant-se, i d'estar connectats a la base de dades utilitzant la finestra **Aliases**, comprovarem primer *que és el que hi ha* a la base de dades.
Utilitzant la pestanya **Objects**, navegarem per l'estructura del catàleg de la BD, tot observant aquells detalls que puguin ser d'interès.

En la imatge adjunta hem obert la carpeta TABLE, per revisar la taula clients.
En fer això, podrem utilitzar les pestanyes de la finestra de la dreta per comprovar qualsevol informació d'aquesta (índexs, claus primàries, foranes, el seu contingut, etc).



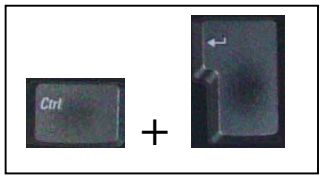
Si tornem a la pestanya SQL, farem dues insercions a la taula **test**.
Introduirem a la finestra SQL les següents línies tal com tenim tot seguit:

```
insert into test values ('Nom de la segona persona');
insert into test values ('Roberto d''Angelo');
```

És important observar que l'editor assigna diferents colors a les paraules, segons anem escrivint:
Blau per les instruccions.
Negre pel nom de les taules (i dels camps).
Morat pels literals de text.

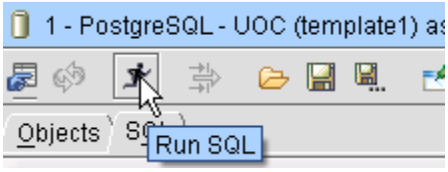
Abans d'executar, comprovarem que els colors són tal com surten a la part superior. Si no és així, haurem de comprovar que utilitzem la cometa senzilla (habitualment situada en la mateixa tecla que el tancament d'interrogant – a la dreta del zero), i que tenim dues cometes seguides en la segona instrucció, per a indicar que hi ha un apòstrof (cometa) en el mateix text!

Quan estiguem segurs de que tot està bé, utilitzarem la combinació de tecles CONTROL + RETURN per a executar-ho:

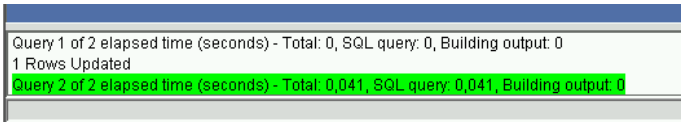


La utilització d'aquesta combinació de tecles és molt ergonòmica, i gairebé tots els editors SQL 'seriosos' la suporten.

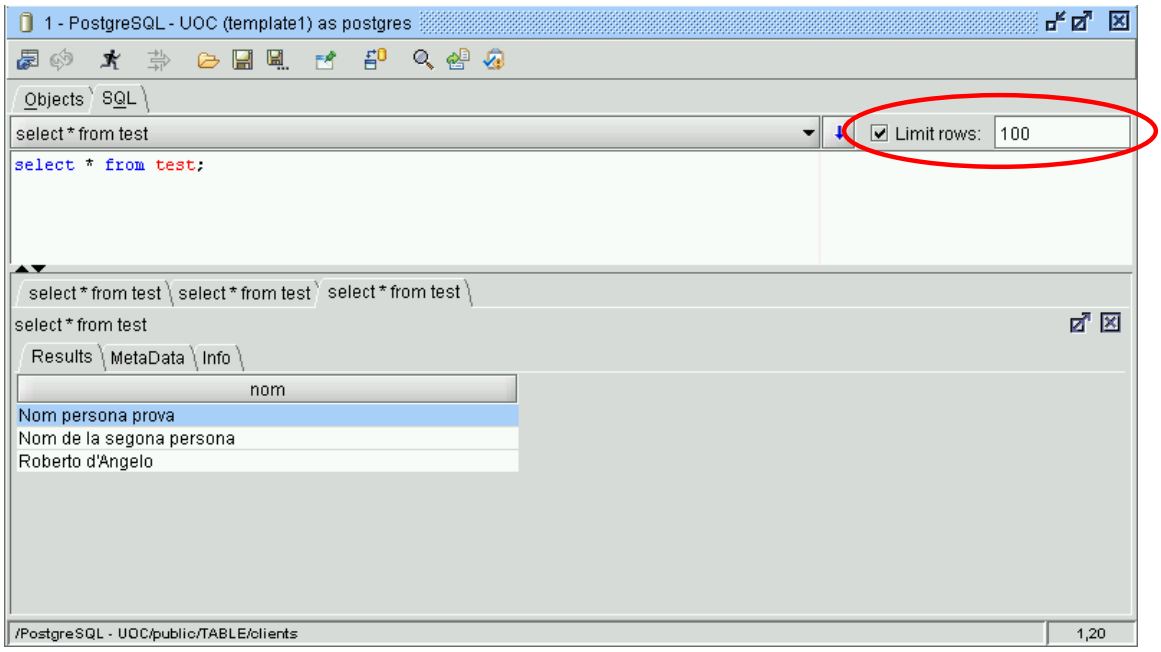
Alternativament, també ho podem executar les insercions amb un clic de ratolí. Picarem sobre el botó que hi ha a la part superior, amb forma d'home corrent:



Si s'ha executat correctament, a la línia de missatges de la part inferior de la pantalla, ens ha dir que s'han executat les dues insercions.



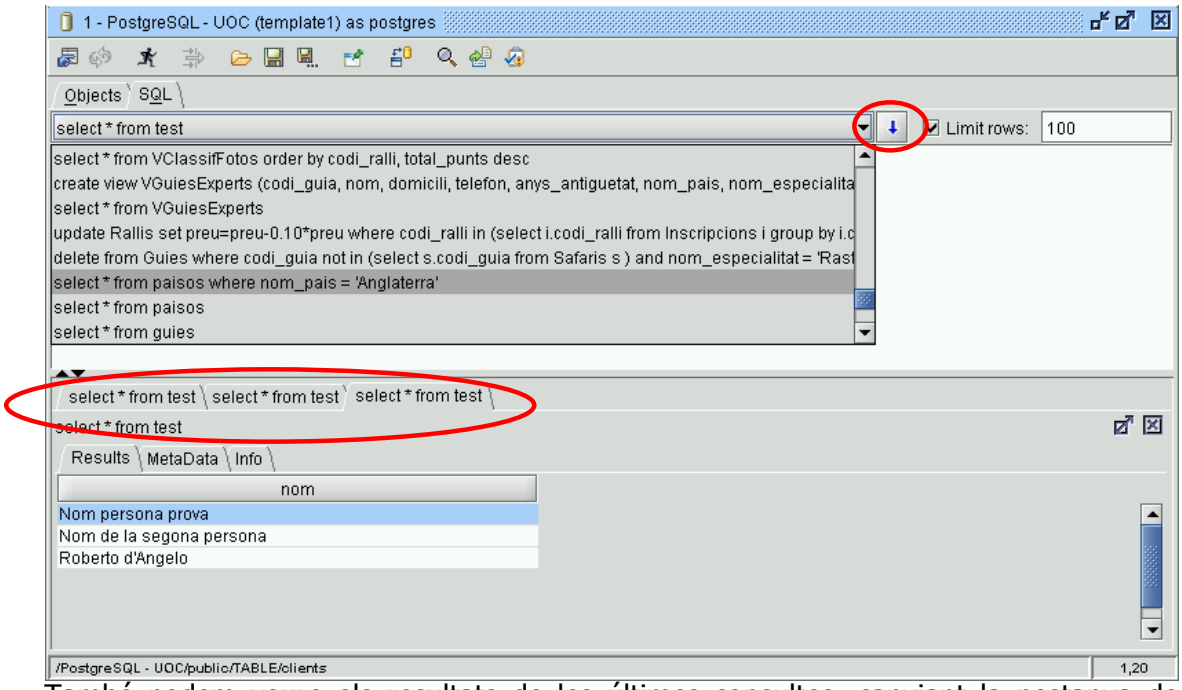
En fer una consulta de la taula **test**, ens han d'aparèixer les dues insercions que hem fet, junt amb la primera que ja hi havia:



És important observar (encerclat en vermell), que per omissió està marcat que només se'ns retornin les 100 primeres files que compleixin la condició. Això evita que per

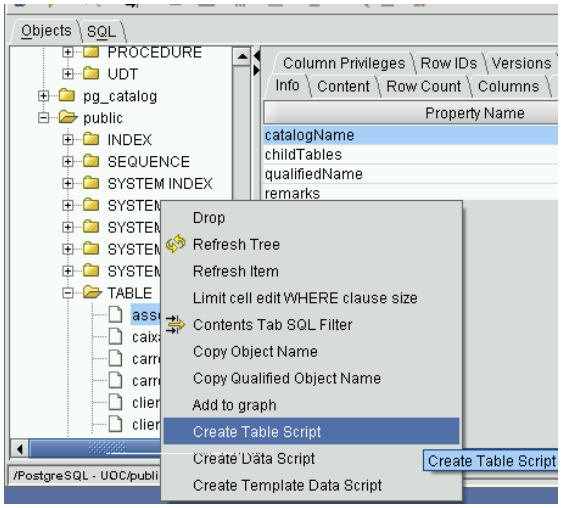
error, en utilitzar taules amb molts registres, fem treballar innecessàriament el servidor.

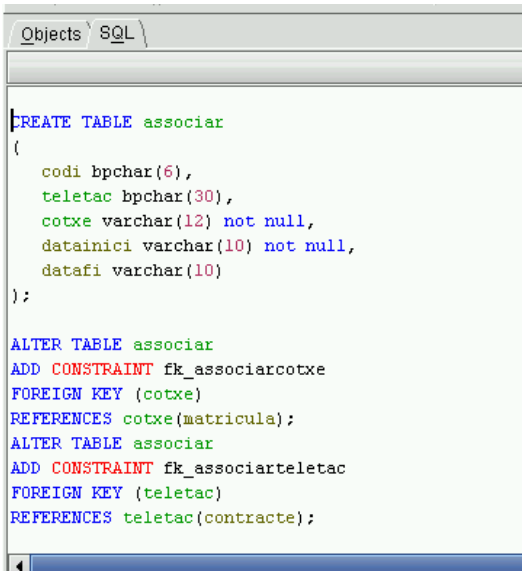
D'haver de tornar a executar una consulta recent, es pot utilitzar la fletxa marcada per a obrir la llista de les últimes instruccions executades.



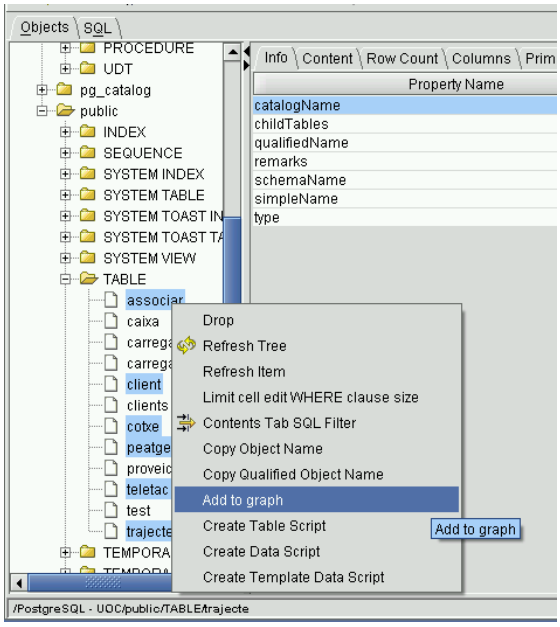
També podem veure els resultats de les últimes consultes, canviant la pestanya de visualització de resultats.

En el cas de voler recuperar l'SQL necessari per a tornar a recrear una de les taules que ja tenim al catàleg, la seleccionarem des de la pestanya **Objects**, i clicarem sobre aquesta amb el botó contrari del ratolí. Finalment elegirem *Create Table Script*.
Un cop fet això, haurem de tornar a la finestra SQL, i comprovarem que allí ens ha aparegut les sentències SQL que són necessàries per a tornar a crear aquesta taula.



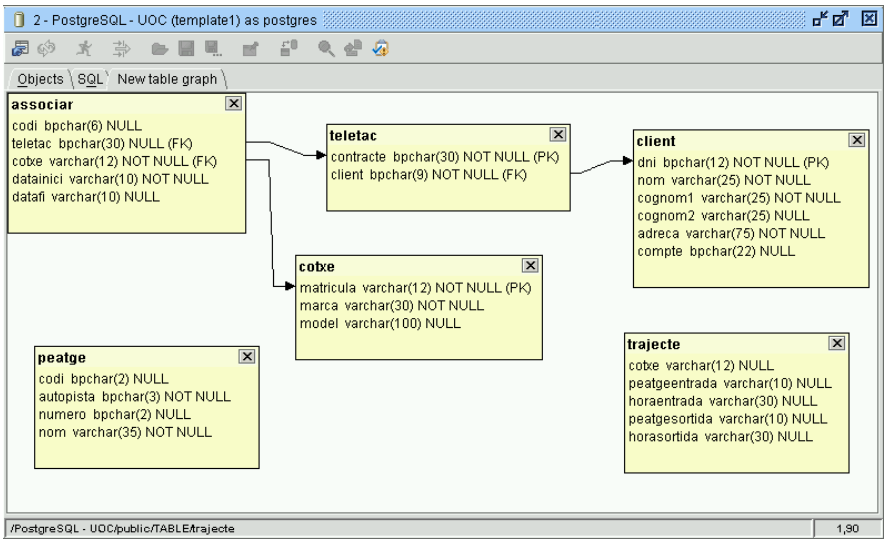


Una opció que també és molt interessant (més quan s'estan utilitzant diverses taules relacionades entre si), és la d'obtenir l'esquema d'aquestes, amb els corresponents vincles.
Per a fer això, des de la mateixa pestanya **Objects**, elegirem les taules que volem incloure a l'esquema (utilitzant les tecles shift i ctrl per a fer la selecció múltiple) .



Tot seguit farem clic amb el botó contrari del ratolí sobre aquestes, i elegirem del menú contextual que apareix, l'opció *Add to graph*.

El resultat se'ns presenta gràficament en una nova pestanya:



Annex XI. Guia d'instal·lació del programari SQuirreL (versió Linux)

Guia d'instal·lació del programari SQuirreL.

(versió Linux)

Introducció

Que és SQuirreL SQL?

SQuirreL és un client per a executar consultes SQL sobre Sistemes de Gestió de Bases de Dades (SGBD), dissenyat sota conceptes moderns. Aquest fet fa que sigui independent d'un SGBD concret, que tingui una interfície gràfica àgil, i que sigui independent tant del Sistema Operatiu (SO) com del maquinari sobre el que s'executa.

SQuirreL SQL és independent del fabricant de l'SGBD, donat que està dissenyat específicament per a que es connecti amb aquests mitjançant els ponts JDBC/ODBC existents.

És independent del sistema operatiu i del maquinari donat que està programat en Java, pel que només cal tenir la maquina virtual (MV) adequada instal·lada per a que qualsevol ordinador sigui plenament compatible.

El programari té llicència del tipus GNU Lesser General Public License (en la pràctica és la GPL), pel que es poden obtenir les fonts del programa original per a corregir/millorar, i tornar a compilar. En ser GNU, no hi ha cap problema per a tornar a copiar/distribuir el programa original o el modificat.

Instal·lació d'SQuirreL SQL (Linux)

Requisits per a fer la instal·lació.

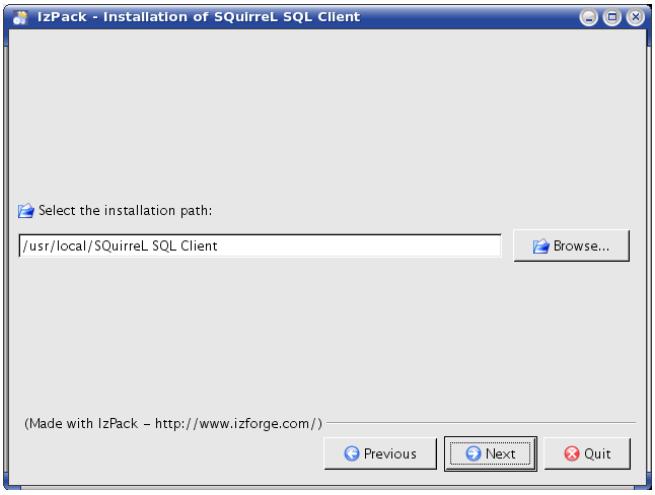
- Tenir instal·lada la versió de Java; jre1.5.0_04.
- Tenir instal·lat i funcionant el PostgreSQL 8.0.3.

Instal·lació del programa.

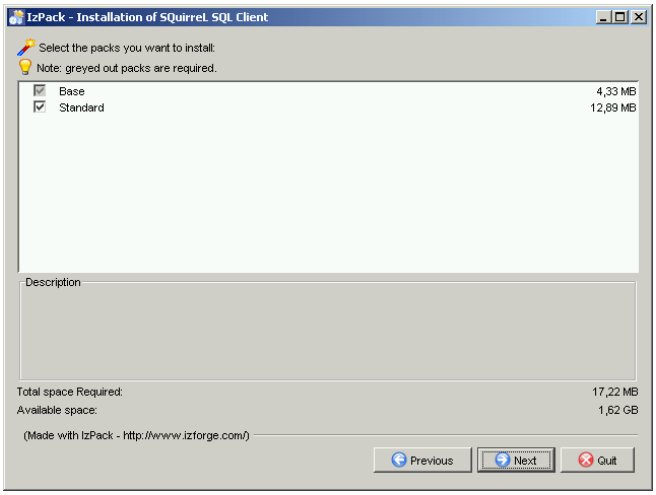
El primer pas és obtenir el programa executable d'instal·lació. Aquest el trobarem al CDROM lliurat amb el material de l'assignatura, o alternativament es pot descarregar des de la pestanya *Recursos* de l'aula (la versió complerta ocupa uns 7Mb aproximadament).
Un cop tinguem el fitxer squirrel-sql-2.0final-install.jar en el directori /home/nomUsuari/ (per ex.), ens situarem en un shell al mateix directori *home*, i executarem la instrucció:
/usr/java/jre1.5.0_04/bin/java –jar /home/nomUsuari/squirrel-sql-2.0final-install.jar

En el cas que no tinguem el java instal·lat en el camí que s'indica en la instrucció anterior, haurem de posar-ho segons ho tinguem.
La pantalla de benvinguda que apareixerà és la següent:

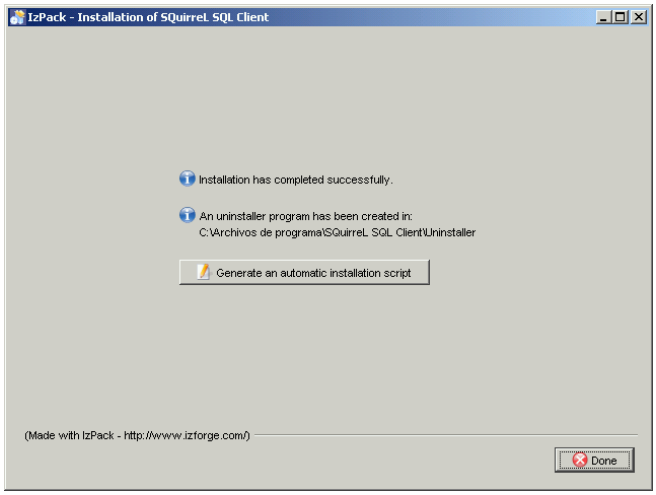
En pitjar al botó de *Next*, ens sortirà la pantalla de presentació del programa, en tornar a fer *Next* ens sortirà la finestra que ens indica que el programa té una llicència de tipus GNU, i que per a instal·lar-ho l'hem d'acceptar.



A la següent pantalla se'ns proposarà un camí d'instal·lació per omissió. Sense canviar-lo, pitjarem sobre el botó *Next*.



Els components triats a la instal·lació bàsica, són tots els que hi han al 'paquet', pel que només haurem de pitjar sobre *Next*.



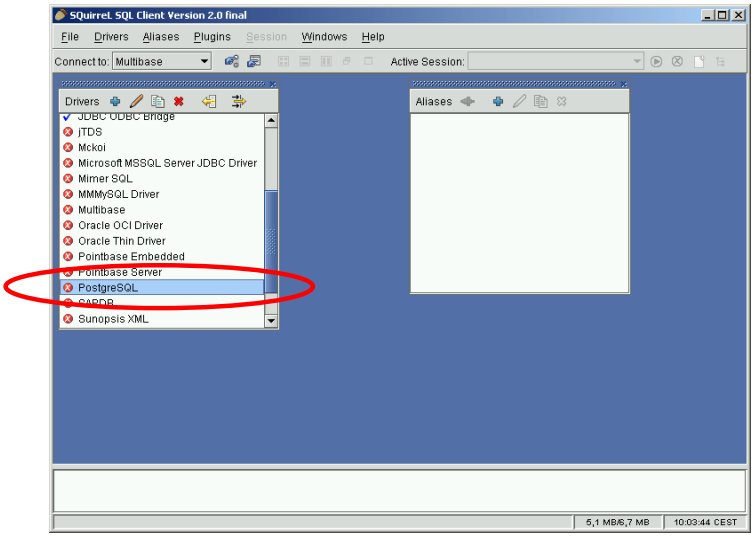
Tot seguit es descomprimiran els fitxers, copiant-se al disc dur. Quan les barres de progrés hagin arribat al 100%, ens apareixerà la pantalla que ens indica que la instal·lació ha acabat.

Un cop acabada la instal·lació, ja el podem executar des del shell amb la comanda:
/usr/java/jre1.5.0_04/bin/java –jar /home/nomUsuari/SQuirrel\ SQL\ Client/squirrel-sql.jar

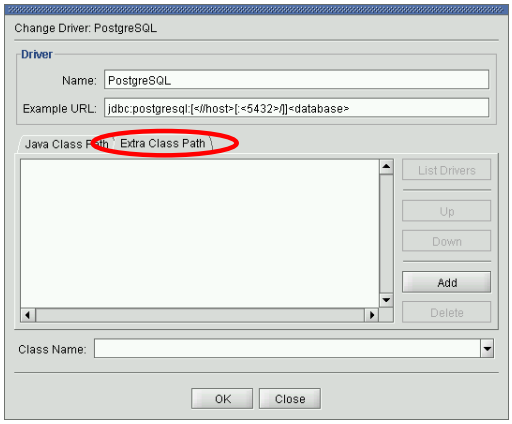
considerant que el camí del java és el que s'ha utilitzat fins al moment, i que l'SQuirreL està al *home* de l'usuari.

Configuració de la connexió JDBC.

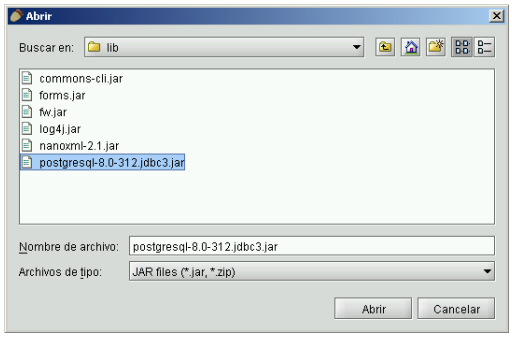
Una vegada tenim el programa instal·lat, cal preparar l'entorn per a que quan aquest busqui el 'pont' JDBC pel PostgreSQL, el pugui trobar.
En engegar inicialment l'SQuirreL podem observar a la finestra de l'esquerra que la connexió pel PostgreSQL no està disponible.



Els passos que haurem de seguir per a configurar-ho és:
- Copiar el drivers JDBC **JDBC_Driver_postgresql-8.1dev-401.jdbc3.jar** que hi ha al CDROM de l'assignatura o la pestanya *Recursos* de l'aula, al directori:
/home/nomUsuari/SQuirreL\ SQL\ Client/lib
- Obrir amb un doble click, la línia de PostgreSQL de la finestra **Drivers** i anar a la pestanya **Extra Class Path**.

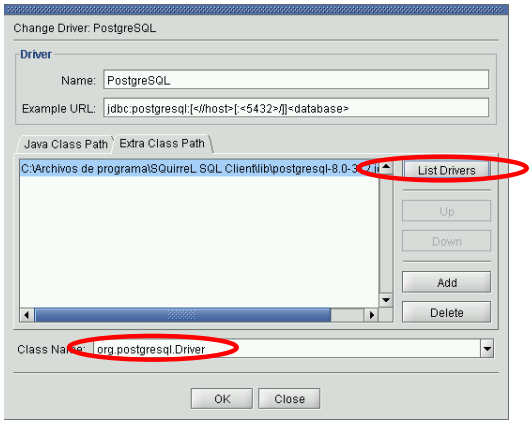


- Utilitzar el botó Add per a seleccionar el fitxer **JDBC_Driver_postgresql-8.1dev-401.jdbc3.jar** (al directori /home/nomUsuari/SQuirreL\ SQL\ Client/lib).



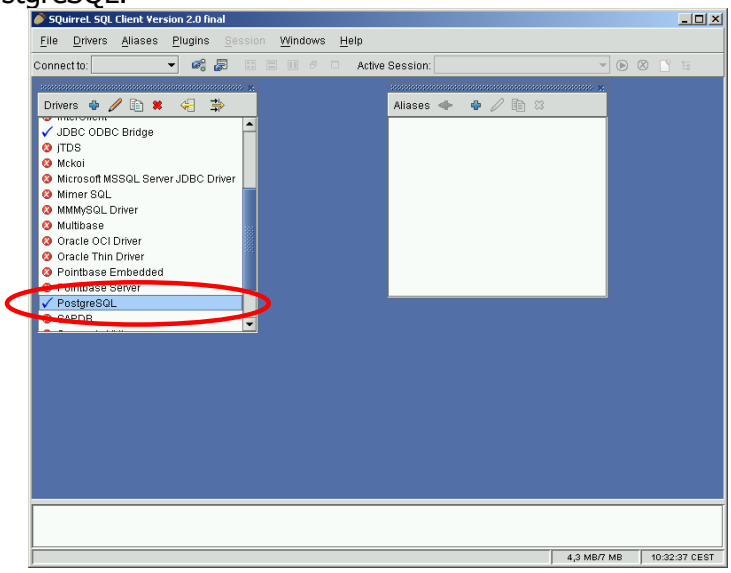
- Comprovar que a Class Name, tenim el

nom del driver que hem carregat. Ha d'aparèixer: org.PostgreSQL.Driver. Si no apareix, haurem d'utilitzar el botó List Drivers per a que el busqui dins el fitxer jar del driver.

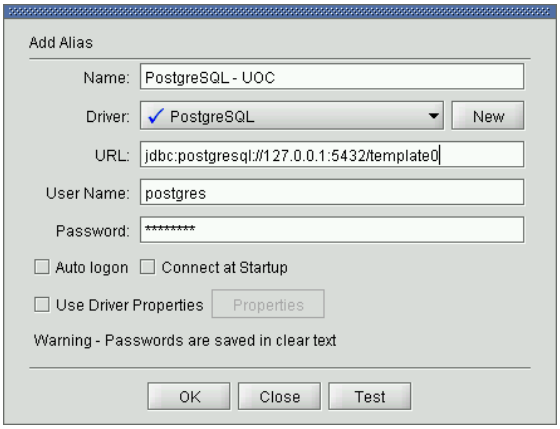


La configuració final, ha de quedar segons la imatge.

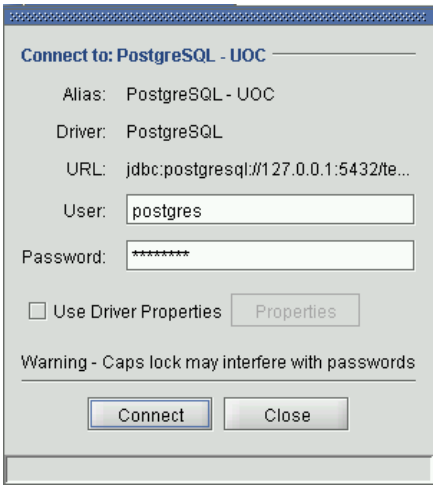
Ara, al tornar a engegar l'SQuirreL ja ha d'aparèixer la marca de 'vist' (disponible) al driver del PostgreSQL.



Com que ja tenim el 'connector' disponible, només caldrà fer la configuració amb els paràmetres del nostre *motor de base de dades* PostgresQL.
Per començar clicarem sobre la creu blava que hi ha a la part superior de la finestra **Aliases**.
L'haurem d'omplir segons la mostra;

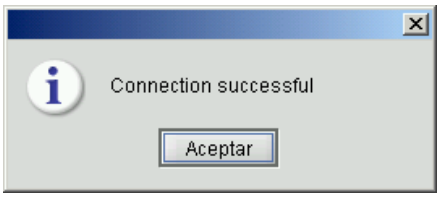


Al final de la línia URL, s'indicarà el template que hagueu vist disponible durant la instal·lació del PostgreSQL (habitualment template0 o template1).



Tot seguit pitjarem sobre el botó Test (ha d'estar el PostgreSQL instal·lat, comprovat, i en funcionament). En sortir la finestra de la dreta, pitjarem sobre el botó Connect.

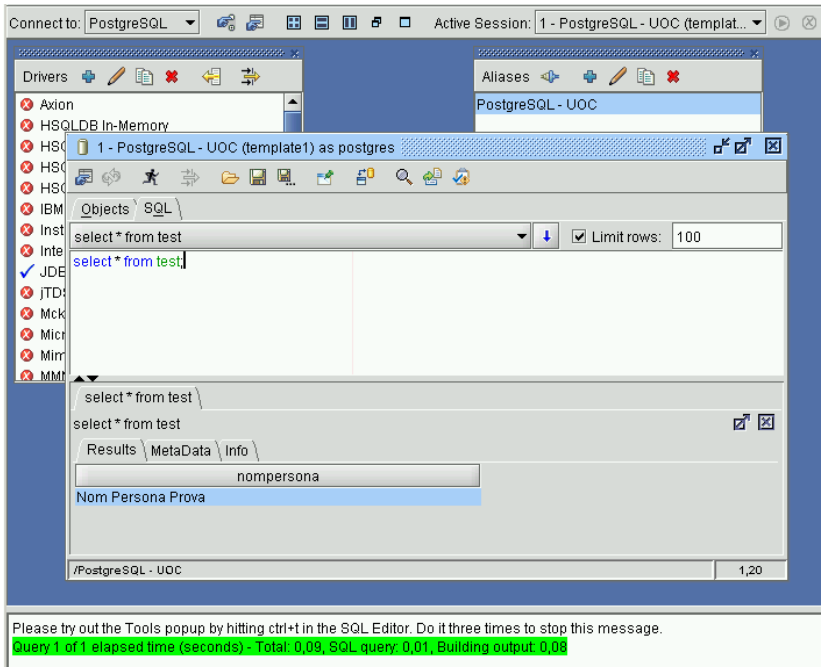
Si en surt el missatge de 'successful', haurem completat tota la instal·lació satisfactòriament.



Prova de funcionament

La prova final consisteix en fer la consulta de la taula que hem creat durant la instal·lació de la base de dades (i que no hem esborrat).

La finestra d'edició SQL s'obra en fer un doble clic a la línia PostgreSQL – UOC de la finestra **Aliases**.
A la part d'edició superior, introduirem la consulta:
select * from test;



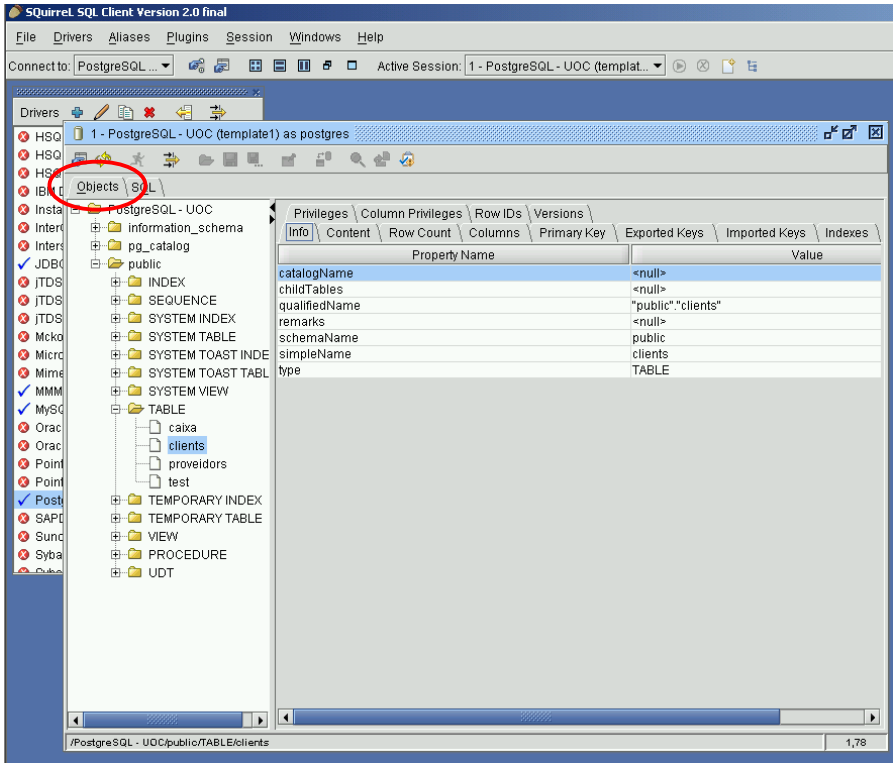
El resultat apareix a la part inferior, en la zona de pestanyes.
També podem observar les estadístiques de la execució de la consulta a la finestra d'estat de la part inferior.

Detalls de funcionament

Ara que tenim el client SQL SQuirreL instal·lat, seguirem un exemple de creació de taules i unes consultes, per a comprovar els detalls de funcionament que fan que sigui realment útil.

Partint de tenir l'SQuirreL executant-se, i d'estar connectats a la base de dades utilitzant la finestra **Aliases**, comprovarem primer *que és el que hi ha* a la base de dades.
Utilitzant la pestanya **Objects**, navegarem per l'estructura del catàleg de la BD, tot observant aquells detalls que puguin ser d'interès.

En la imatge adjunta hem obert la carpeta TABLE, per revisar la taula clients.
En fer això, podrem utilitzar les pestanyes de la finestra de la dreta per comprovar qualsevol informació d'aquesta (índexs, claus primàries, foranes, el seu contingut, etc).



Si tornem a la pestanya SQL, farem dues insercions a la taula **test**.
Introduirem a la finestra SQL les següents línies tal com tenim tot seguit:

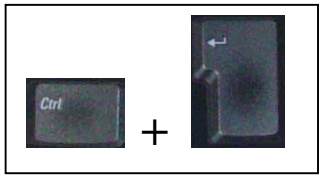
```
insert into test values ('Nom de la segona persona');
insert into test values ('Roberto d''Angelo');
```

És important observar que l'editor assigna diferents colors a les paraules, segons anem escrivint:
Blau per les instruccions.
Negre pel nom de les taules (i dels camps).
Morat pels literals de text.

Abans d'executar, comprovarem que els colors són tal com surten a la part superior.

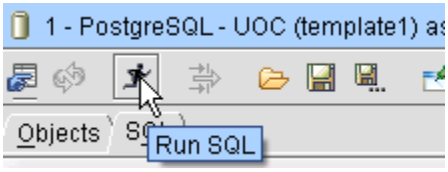
Si no és així, haurem de comprovar que utilitzem la cometa senzilla (habitualment situada en la mateixa tecla que el tancament d'interrogant – a la dreta del zero), i que tenim dues cometes seguides en la segona instrucció, per a indicar que hi ha un apòstrof (cometa) en el mateix text!

Quan estiguem segurs de que tot està bé, utilitzarem la combinació de tecles CONTROL + RETURN per a executar-ho:

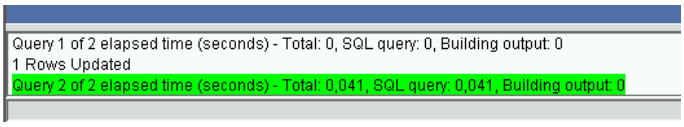


La utilització d'aquesta combinació de tecles és molt ergonòmica, i gairebé tots els editors SQL 'seriosos' la suporten.

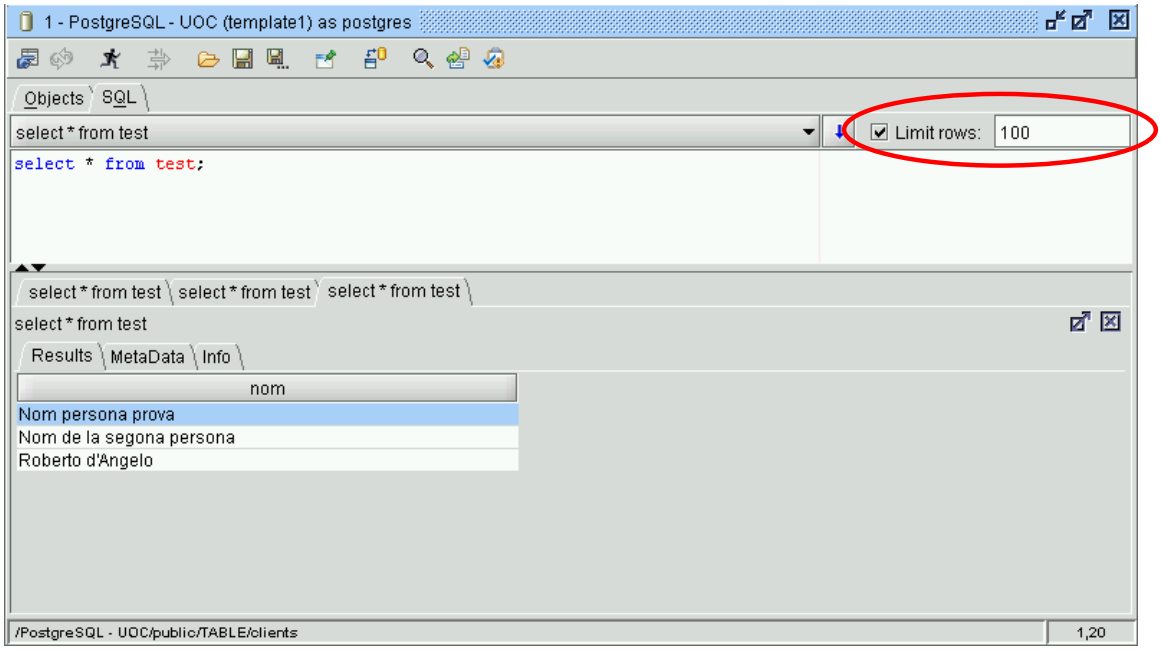
Alternativament, també ho podem executar les insercions amb un clic de ratolí. Picarem sobre el botó que hi ha a la part superior, amb forma d'home corrent:



Si s'ha executat correctament, a la línia de missatges de la part inferior de la pantalla, ens ha dir que s'han executat les dues insercions.



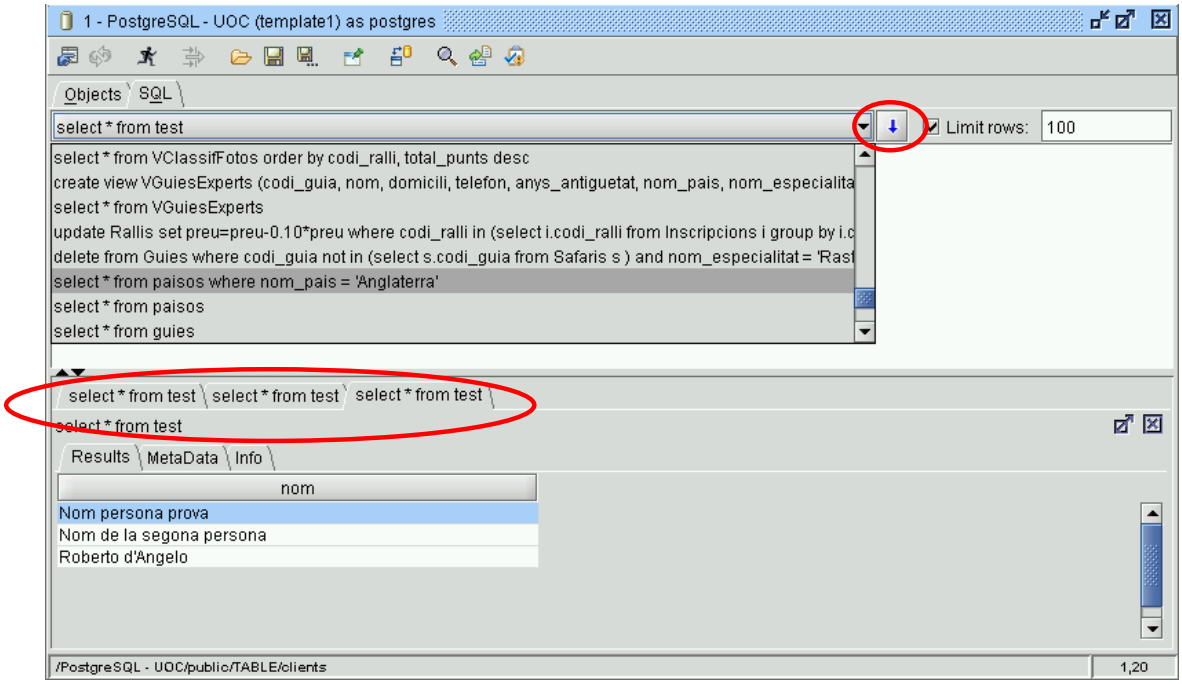
En fer una consulta de la taula **test**, ens han d'aparèixer les dues insercions que hem fet, junt amb la primera que ja hi havia:



És important observar (encerclat en vermell), que per omissió està marcat que només se'ns retornin les 100 primeres files que compleixin la condició. Això evita que per

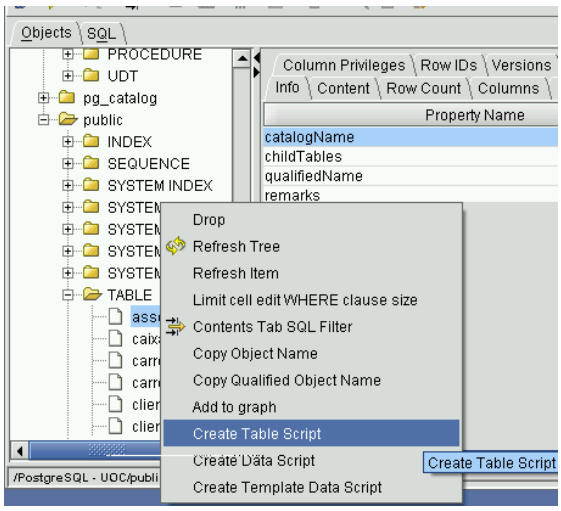
error, en utilitzar taules amb molts registres, fem treballar innecessàriament el servidor.

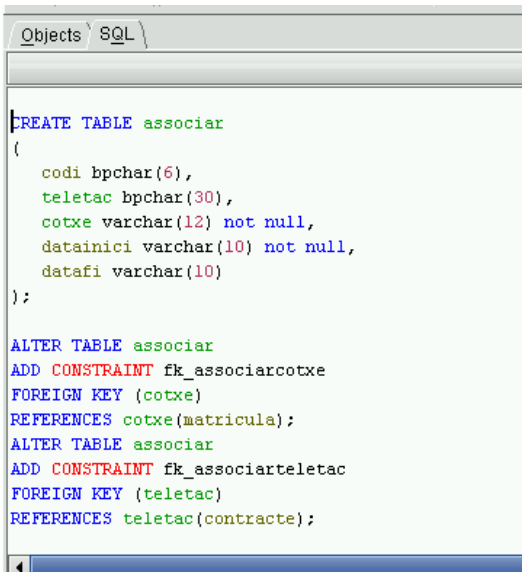
D'haver de tornar a executar una consulta recent, es pot utilitzar la fletxa marcada per a obrir la llista de les últimes instruccions executades.



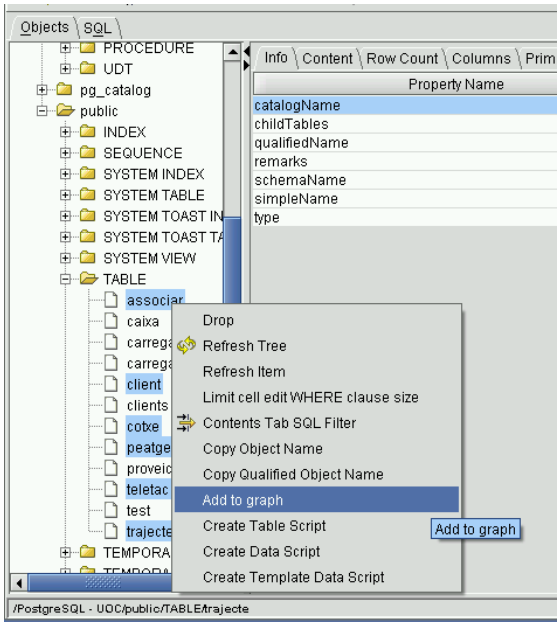
També podem veure els resultats de les últimes consultes, canviant la pestanya de visualització de resultats.

En el cas de voler recuperar l'SQL necessari per a tornar a recrear una de les taules que ja tenim al catàleg, la seleccionarem des de la pestanya **Objects**, i clicarem sobre aquesta amb el botó contrari del ratolí. Finalment elegirem *Create Table Script*.
Un cop fet això, haurem de tornar a la finestra SQL, i comprovarem que allí ens ha aparegut les sentències SQL que són necessàries per a tornar a crear aquesta taula.



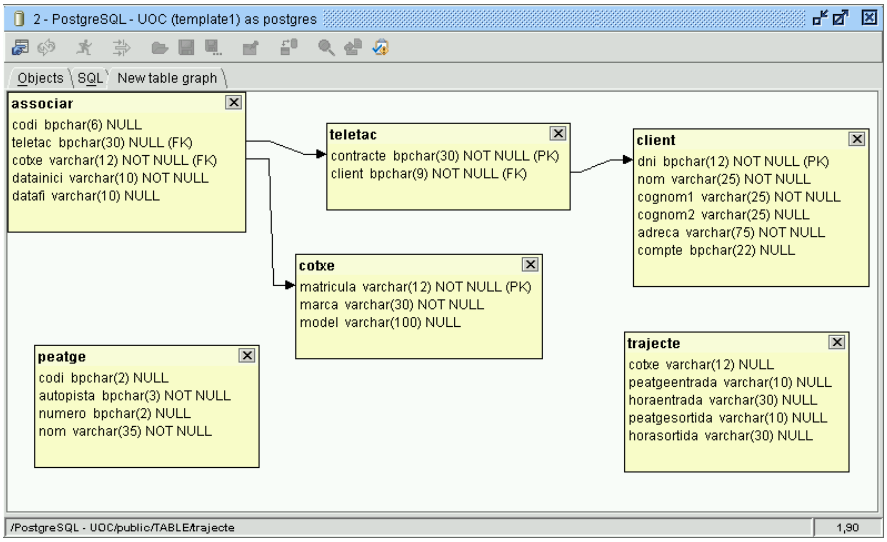


Una opció que també és molt interessant (més quan s'estan utilitzant diverses taules relacionades entre si), és la d'obtenir l'esquema d'aquestes, amb els corresponents vincles.
Per a fer això, des de la mateixa pestanya **Objects**, elegirem les taules que volem incloure a l'esquema (utilitzant les tecles shift i ctrl per a fer la selecció múltiple) .



Tot seguit farem clic amb el botó contrari del ratolí sobre aquestes, i elegirem del menú contextual que apareix, l'opció *Add to graph*.

El resultat se'ns presenta gràficament en una nova pestanya:



Annex XII. Manual funcionament PostgreSQL CDLive!

Manual funcionament PostgreSQL CD Live!

Introducció

Que és PostgreSQL?

PostgreSQL és un programari de Base de Dades (BD) amb llicència de distribució tipus BSD[6], o sigui, és un programa d'ús i distribució lliure. Del mateix se'n pot obtenir el programa font per a modificar-lo, ampliar-lo, o corregir segons les pròpies necessitats.

D'entre els Sistemes de Gestió de Bases de Dades (SGBD) actuals, destaca per la seva estabilitat, possibilitats i versatilitat, comptant amb característiques que fins i tot alguns SGBD comercials no incorporen a l'actualitat. L'haver estat desenvolupat des d'un inici en un entorn acadèmic (Universitat de Berkeley – 1974), fa que estigui concebut com a una plataforma d'exhibició de *l'estat del art* de la tecnologia en aquesta branca del programari. Al seu inici era una demostració del que haurien de ser els SGBDs futurs (incorporant per tant opcions que 'comercialment' no eren desenvolupables segons un estricte criteri de beneficis empresarials). Actualment encara destaca per tenir característiques molt avançades, que difícilment es troben en altres sistemes de la seva categoria (per exemple: el subsistema GisT, que és utilitzat per a facilitar l'emmagatzemament i indexació de polígons tridimensionals en sistemes cartogràfics i topogràfics).

El constant creixement de la popularitat del programari lliure (tipus GNU, GPL o similars), fa que sigui imprescindible tenir una coneixença profunda d'aquests entorns per a desenvolupar els aplicatius que demanda el mercat actual. La constant aparició d'empreses que ofereixen suport al programari lliure (IBM, Novell, SUN, entre d'altres), fa que s'esvaeixin els dubtes de si aquest tipus de programari és viable en un futur, ja que en aquest nou paradigma es traslladen les inversions inicials de compra, al manteniment i a millors desenvolupaments.

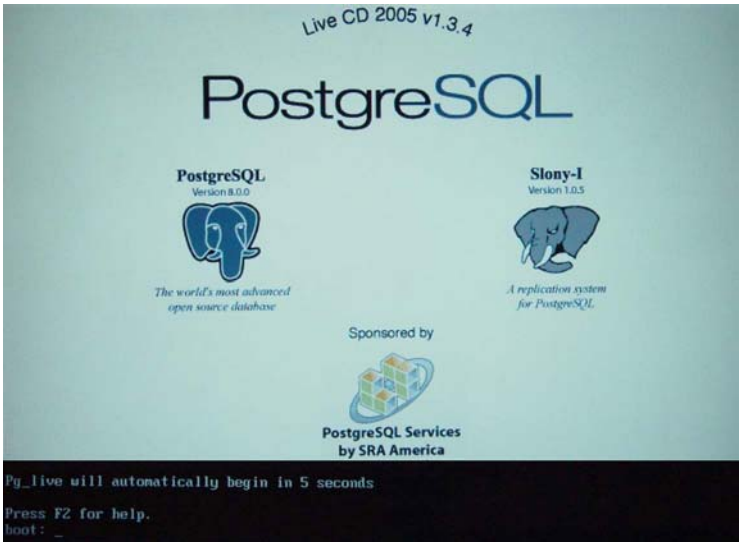
La disponibilitat del codi font dels aplicatius amb llicència BSD, fa que no es depengui del fabricant del programari per la continuïtat del producte. Si el fabricant desaparegués, la inversió feta en el desenvolupament del programari propi (sobre un SGBD concret per exemple) estaria assegurada, ja que aquest *motor de base de dades* seguiria tenint el manteniment d'una comunitat d'usuaris (distribuïda probablement per tot el mon), que estaria interessada en solucionar els problemes que sorgissin, i fins i tot de continuar la seva evolució. També s'ha de considerar que el desenvolupament dels mòduls d'idioma o altres característiques que poden no ser rentables per un fabricant concret, pot ser desenvolupats pels 'tercers' que estiguin interessats.

[6] Existeixen altres BDs que es mouen a l'entorn del programari lliure, però que no ho són realment. MySQL per exemple, té llicència dual (gratuït per a ús privat i de pagament en usos comercials/empresarials).

Requeriments PostgreSQL CD Live

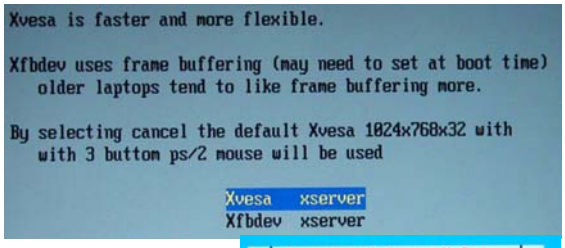
Punt de treball bàsic recomanat, segons especificacions UOC.

Arrancada:
Amb l'ordinador engegat, s'introdueix el CDROM PostgreSQL Live a la unitat òptica principal de l'ordinador, i es reinicia aquest.
Si l'ordinador està configurat per a que comprovi l'existència d'un CD arrancable en la unitat durant el procés de posta en marxa, apareixerà la següent pantalla:

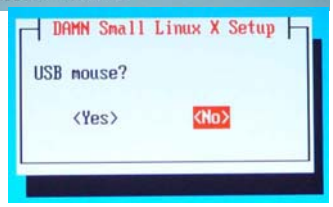


En cas que de que l'ordinador engegui el sistema operatiu instal·lat (tal com faria normalment), caldrà seguir els passos de l'apèndix A, abans de continuar seguint la guia.
La primera vegada que provem el sistema CD Live, esperarem els 5 segons que s'indiquen a la pantalla inicial, per a provar si l'ordinador on s'executa és plenament compatible amb la distribució Linux que incorpora el CDROM.
Durant el procés de càrrega, se'ns faran quatre preguntes:

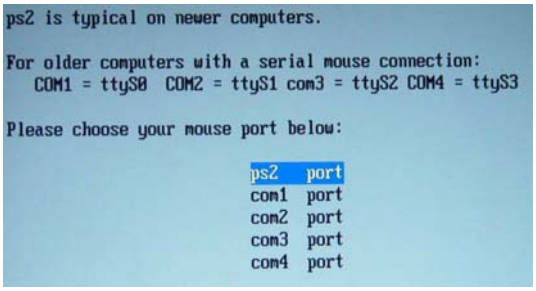
a) Si l'ordinador suporta l'estàndard Xvesa. La primera vegada contestarem afirmativament.

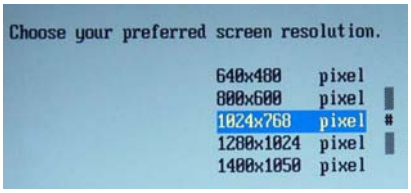


b) Si l'ordinador té un ratolí de tipus USB.
Segons el tipus de connector del ratolí, elegirem si és USB o no. Si pitgem **No**, posteriorment podrem triar entre PS2 i Sèrie (si indiquem que no té 'wheel').

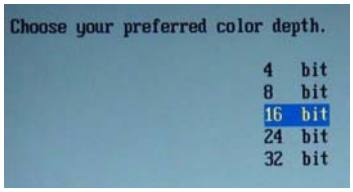



USB PS2 Sèrie

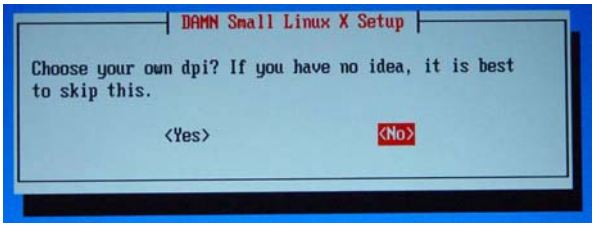




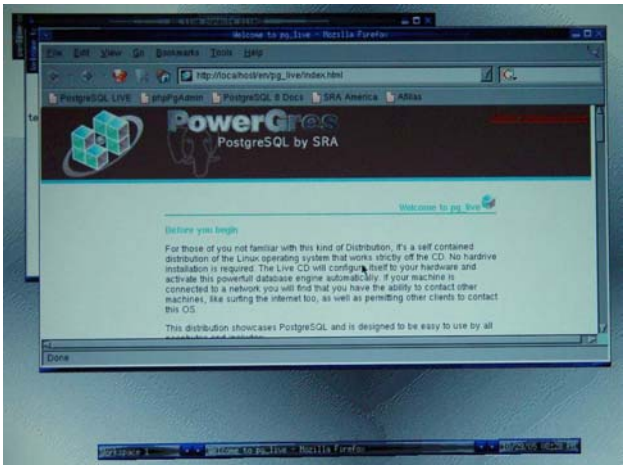
c) Sobre la resolució a la que volem treballar (ha d'estar suportada per la tarja gràfica i el monitor!), i el nombre de colors.
4 bits = 2^4 = 16 colors
8 bits = 2^8 = 256 colors
16 bits = 2^{16} = 655536 colors
...
La primera vegada optarem per triar una resolució i un nombre de colors baixos.



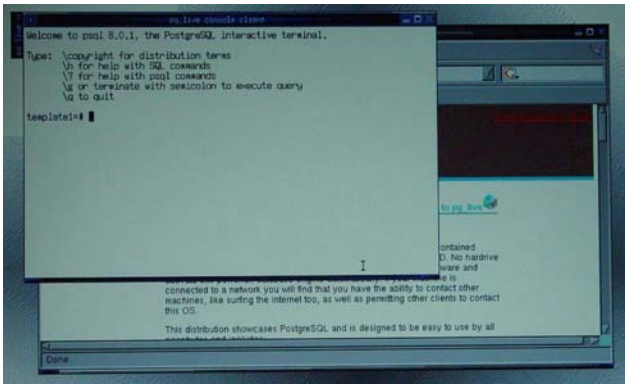
d) La pregunta sobre els dpi (dot per inch) no ens influenciarà. Es pot contestar **No**.



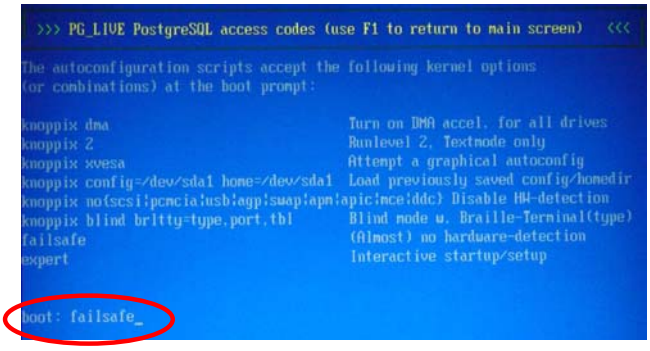
Si tot el procés ha anat bé, ens apareixerà la pantalla inicial de la distribució PowerGres, del PostgreSQL 8.01.



Darrera de la pantalla de benvinguda (amb les explicacions de la distribució), tenim la cònsola per a introduir instruccions SQL. Podem passar a aquesta seleccionant la finestra amb el ratolí, o amb la combinació de tecles ALT+Tabulador.
Si apareix aquesta tal com es veu a la imatge, es pot passar directament a l'apartat d'aquesta guia de 'Prova de funcionament inicial'



Si durant el procés d'arrancada l'ordinador s'ha aturat per algun problema mentre reconeixia els dispositius del maquinari, caldrà reiniciar-lo.
En tornar a engegar amb el CDROM posat, s'haurà de pitjar F2 en la pantalla inicial.
A la pantalla d'opcions d'arrancada s'haurà d'introduir el paràmetre '**failsafe**', per a que faci una arrancada amb el mínim de dispositius.



Prova de funcionament inicial.

Introduirem les comandes tal com es mostren a continuació: (veure apèndix B - sobre distribució del teclat anglès)

```
create table test (nom varchar(60));
```

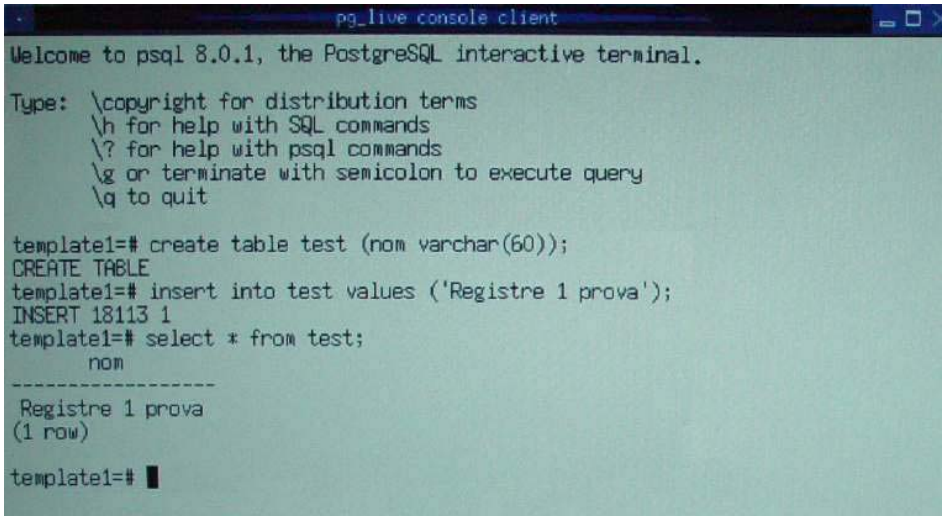
```
insert into test values ('Prova numero 1');
```

```
select * from test;
```

Ha de respondre;

```
nom
--------------------------
Prova numero 1.
```

Es poden comprovar els detalls a la imatge adjunta:



Resolució de problemes i FAQ

El CD-ROM facilitat juntament amb el material de l'assignatura no fa el que s'indica en aquesta guia!

- Si únicament s'ha facilitat el CD-ROM amb la recopilació dels manuals d'instal·lació de PostgreSQL, el mateix programari (pels diferents SO), i altres continguts, haureu de preparar vosaltres mateixos el CD-ROM del PostgreSQL CD-Live!. Això es fa a partir de la imatge ISO que hi ha al CD-ROM al directori: ***Programari PostgreSQL*****PostreSQL CD-Live**, i que té per nom: ***pg_live.1.3.4-SRAA.iso***
- El procediment a seguir variarà segons el programa de gravació que utilitzeu, però bàsicament consisteix en indicar que es vol obrir el fitxer .iso, i tot seguit indicar que es vol enregistrar un nou CD-ROM a partir d'aquesta imatge.

Annex A

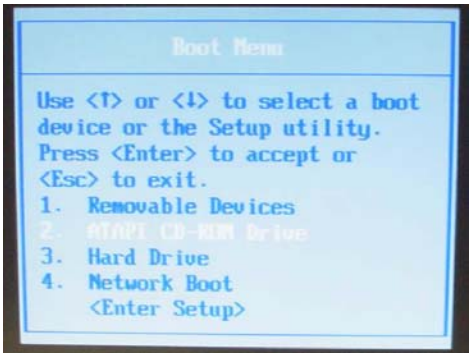
Configuració ordinador per a arrancar des de CDROM.

Segons la marca i/o model de l'ordinador (i fins i tot de la configuració del mateix) ens podem trobar en varis casos:

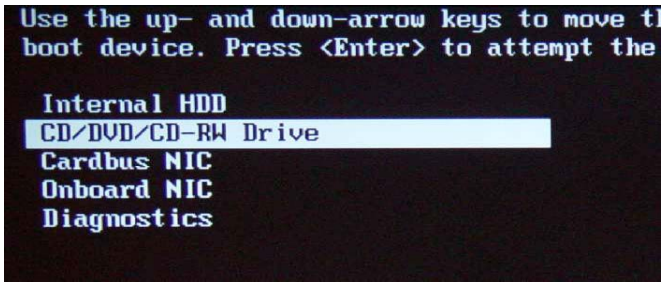
a) L'ordinador engega correctament des del CDROM/DVD, si hi ha un suport òptic amb auto-arrancada.
b) Fa falta pitjar una tecla durant l'arrancada de l'ordinador per a que aquest comprovi si hi ha un suport òptic a la unitat de CDROM/DVD, per passar a arrancar d'aquesta si té auto-arrancada.
c) L'ordinador no està configurat per a engegar des de CDROM/DVD.

En el primer cas no tindrem cap problema per arrancar al PostgreSQL live CD.

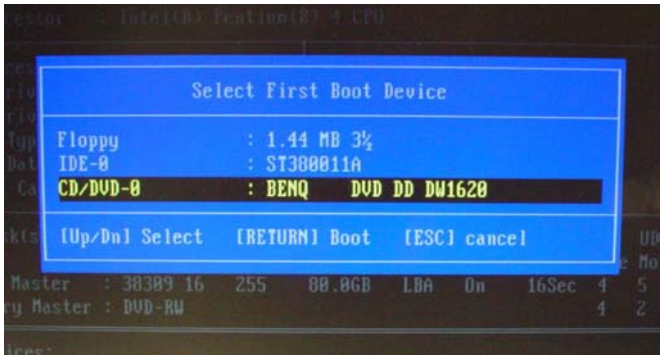
En el cas **b**, haurem d'estar alerta durant el procés d'arrancada de l'ordinador, per si indica quina és la 'magic key' que fa que comprovi l'unitat de CDROM.
Malgrat que segons el fabricant de l'ordinador, la tecla a pitjar pot diferir, les més 'habituals' són F12 (Dell), tecla ESC (HP), tecla F11 (ordinador amb BIOS AMI – molts clónics), tecla F2 (Toshiba).
En cas que reconegui que s'ha pitjat la tecla corresponen per a forçar l'esmentat 'boot' des d'una unitat alternativa al disc dur, hauria d'aparèixer un menú similar als següents (amb les diverses possibilitats).



Hewlett-Packard



Dell



AMI (BIOS American Megatrend)

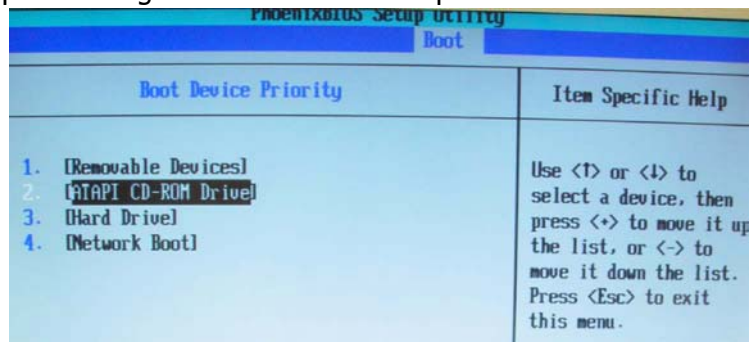
En tots els casos caldrà seleccionar l'opció que indica CDROM/DVD.

Si cap tecla permet arrancar des d'una unitat alternativa al disc dur principal, caldrà configurar a les opcions de la BIOS (Basical Input Output System), aquesta possibilitat.

ALERTA: És un procés delicat, ja que de tocar una opció que no correspon, l'ordinador pot deixar d'arrancar!

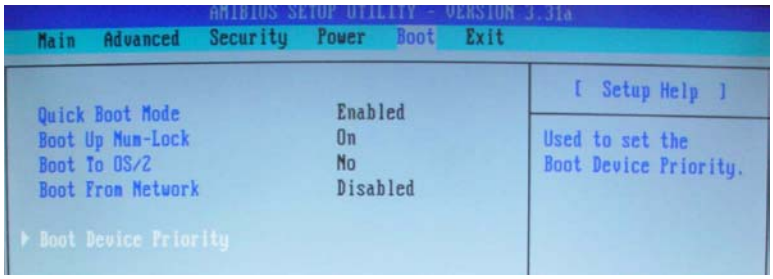
Segons el fabricant de l'ordinador, la BIOS tindrà una interfície més o menys 'amigable'. Sigui com sigui, hauríem de cercar l'opció de 'dispositiu d'arrancada', en algun apartat que faci referència al procés de 'BOOT', per a especificar quin volem que sigui el dispositiu principal.

EXEMPLE:
Entrem a configurar la BIOS d'un ordidor HP (fabricant de BIOS: Phoneix), tot pitjant la tecla SUPR durant l'arrancada.
Un cop comprovades les opcions que surten a la pantalla, anem a l'opció de BOOT, i ens trobem que podem elegir entre diverses amb possibilitats:

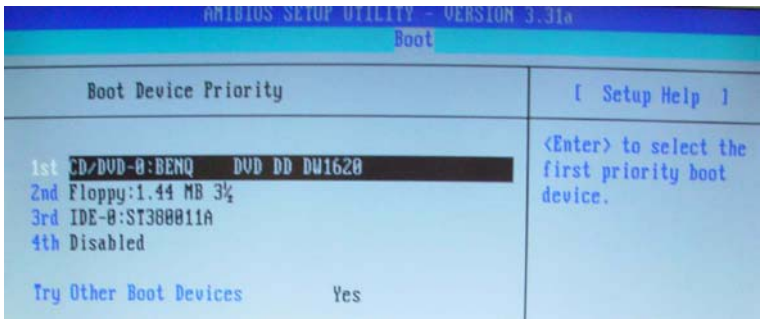


Elegirem CDROM, i en pitjar la tecla ESC, se'ns indicarà com podem sortir d'aquests menús de configuració, tot guardant els canvis fets.

En el cas que la BIOS sigui AMI, ens trobarem que podem anar al menú BOOT:



Un cop en aquest, triarem Boot Device Priority:



Farem que el primer dispositiu que es comprovi per arrancar, sigui el CDROM/DVD.

Més que veure un cas concret de com configurar-ho, el més important és veure que en tots els ordinadors hi ha una pantalla on es pot configurar l'ordre en que l'ordinador comprova els dispositius d'emmagatzemament durant el procés d'arrancada.

Annex B

Disposició del teclat QWERTY angles

El programari PostgreSQL CD live està desenvolupat als Estats Units, pel que la distribució del teclat que ens trobem en editar qualsevol sentencia SQL, segueix la disposició anglesa.

Es pot comprovar que els parèntesis estan desplaçats una posició a la dreta, l'asterisc està sobre la tecla del vuit, i el punt i coma està on tenim habitualment la lletra eñe.