UNIVERSITAT OBERTA DE CATALUNYA

ENGINYERIA TÈCNICA INFORMÀTICA DE SISTEMES

AVALUACIÓ DE L'ESTÀNDARD SQL:1999 RESPECTE LES CARACTERÍSTIQUES ORIENTADES A L'OBJECTE QUE SUPORTA I LA SEVA IMPLEMENTACIÓ ALS SGBD OBJECT-RELATIONAL.

Alumne: Ivo Plana Vallvé
Dirigit per: Alex Alfonso Minguillon
CURS: Setembre 2003 - Gener 2004

Índex

1. Resum

En aquest treball s'intenta realitzar una síntesis de les especificacions aportades per l'estàndard definit com a SQL:1999, tot analitzant les ampliacions realitzades que fan referència a la nova orientació a l'objecte i a la incorporació de l'herència com a principal element diferenciador d'aquest.

Es comença fent un repàs als escrits publicats, sobre el que es creia que hauria de ser el camí evolutiu de les bases de dades relacionals, per a convertir-se en les bases de dades relacionals orientades a l'objecte, i dels manifests on s'especificaven les característiques que es creien necessàries d'implementar en les noves bases de dades orientades a l'objecte (no relacionals).

La segona part d'aquest treball està dedicat a analitzar la funcionalitat de la base de dades PostgreSQL, tot comprovant fins a quin grau implementa les especificacions del SQL:1999. Les conclusions sobre aquesta base de dades són molt positives, malgrat tingui mancances en l'apartat de la implementació dels mínims necessaris per a poder indicar que acompleix amb les definicions de l'any 99, però altrament es recorda que PostgreSQL és una base de dades complerta i potent, i de codi font obert, es a dir, de cost zero i de lliure distribució.

En l'última part del treball es compara les característiques del PostgreSQL amb les del Oracle 9i, per comprovar fins a quin punt divergeixen en la implementació, de les especificacions del SQL:1999.

2. Introducció

L'accés a dades és una de les principals necessitats en el tractament de la informació, pel que corroborem que els processos informàtics no funcionen de forma isolada, si no que pel contrari interactuen amb uns paràmetres o variables que són obtingudes de l'exterior en temps real o diferit.

Els primers pseudo-sistemes informàtics tenien integrada la funcionalitat o programa en el seu cablejat, fent falta refer les connexions entre els components, en cas de voler que la *maquina de càlcul* executes un procés diferent.

Donat que tot es basava en la interconnexió, les dades també apareixien en aquells temps, com a presencia o absència de valor lògics en certs punts, degut precisament al connexionat amb la resta de components.

En el moment en que aquests primers computadors, van ser capaços de manipular més que un senzill numero com a resultat de portar a terme les seves operacions, va fer falta un suport que permetés en primer lloc representar de forma permanent els resultats (impressores de paper), i en segon lloc, un suport que permetés emmagatzemar l'esmentada informació resultant, però amb la possibilitat de que aquesta pugues tornar a ser introduïda en un sistema informàtic, com a *input* d'un nou procés.

En les primeries, la informació s'emmagatzemava en tubs de raigs catòdics, posteriorment en suports perforats, més endavant en suports magnètics (en les seves diverses variants), per continuar evolucionant fins als suports òptics.

Un cop aconseguits uns mecanismes prou robustos, ràpids i fiables per a emmagatzemar la informació, s'ha passat a millorar progressivament la forma en que aquesta s'emmagatzema en els esmentats suports d'emmagatzemament, per a poder posteriorment recuperar-la de la manera més senzilla i ràpida possible.

Si bé es pot considerar com una base de dades el primer cens dels Estats Units (realitzat en fitxes perforades), s'hauria de considerar com a inici de les bases de dades, l'època en que es varen començar a utilitzar els suports magnètics amb accés directe.

Les primeres aplicacions no van anar més enllà de guardar fitxers de text pla per a posteriorment poder recuperar-los i bolcar així les dades en la memòria (ja fossin aquestes dades valors de variables o de taules). En aquest moment no hi havia diferencia entre llegir aquest fitxer i un de seqüencial, malgrat que es podia llegir un fitxer qualsevol dels emmagatzemats.

El següent pas important va ser el poder accedir a aquests fitxers plans directament, es a dir, a partir d'un punt desitjat. Amb aquesta possibilitat, i amb la creació d'un o diversos fitxers addicionals, per indicar la posició, i per tant l'ordre de les dades atòmiques contingudes en el fitxer, ja es va poder fer

cerques complexes. Podem dir que en el moment de desenvolupar-se aquesta fita, va començar l'historia moderna de les bases de dades.

L'evolució informàtica de la branca del programari de 'les bases de dades', va passar de desenvolupar solucions mono-ordinador, fins als moderns Sistemes de Gestió de Bases de Dades Distribuïdes Orientades a l'Objecte.

En sistemes mono-ordinador i mono-procés, la base de dades únicament s'havia d'encarregar d'emmagatzemar ordenadament la informació, per després poder recuperar-la segons uns certs criteris. En aquest punt, més que base de dades, teníem el que ara podríem anomenar 'fitxer contenidor de dades'.

Aquests limitats i senzills sistemes van tenir que ser ràpidament millorats per a poder suportar concurrència d'accessos. Així dons, en el nou escenari, una base de dades centralitzada, era accedida per diversos ordinadors simultàniament.

En aquest cas, cada ordinador verificava en els fitxers índexs, la posició dels valors requerits, per posteriorment accedir a aquests, si no estaven blocats per un altra procés/usuari.

Els problemes de fiabilitat i rendiment van ser els colls d'ampolla que van fer sorgir la necessitat de crear un nou concepte en aquest camp; Els Sistemes de Gestió de Bases de Dades Transaccionals (SGBD).

En el nou escenari apareixien diversos conceptes nous:

- El servidor, motor de la base de dades
- Els clients, que consultaven la base de dades
- El llenguatge de consulta.
- El protocol de comunicació entre els clients i el servidor.

Amb el sistema exposat, un client pot enviar una petició de 'feina' al servidor, per a que aquest un cop comprovada la viabilitat d'execució de la tasca demanada, passí a planificar-la, optimitzar-la i finalment a executar-la si no entra en conflicte amb cap altra tasca en curs o fins i tot pendent. Un cop executada correctament la tasca, el servidor transmet la informació resultant al client corresponent.

Es pot observar diferencies significatives en el funcionament d'aquest nou esquema gestor de dades;

- Existeix un propietari de les dades (el servidor, motor de la base de dades), que està executant peticions de diversos clients, per passar a resoldre-les i enviar els resultats a aquests.
- La seguretat de les dades s'incrementa, ja que al existir un programari que recull peticions (es pot considerar una certa abstracció de les dades), i que cap client les pot corrompre per una fallada de comunicacions, o per una errada en el procés de bloqueig de registres, per ex. El servidor en tot moment té en compta la situació de totes les peticions en curs, per a poder avançar els treballs en la posició de les cues de tasques pendents, tot planificant l'ordre d'execució, o fins i tot

rebutjant-los si es produeixen interferències entre diversos d'aquests. Es fa servir la política d'atomicitat en l'execució de les tasques (o es porten completament a terme, o en cas contrari no es realitza cap acció).

- La velocitat en l'accés a les dades també es veu millorada amb aquest nou sistema, ja que el client únicament ha d'enviar una petició de l'operació que proposa que es realitzi. Amb ben pocs bytes circulant pel canal de comunicació, es pot fer realitzar una tasca complexa al servidor. Un cop el servidor ha processat la petició declarativa, torna el resultat al client. Un altre cop podem comprovar que el nombre de dades que circulen per la línia de comunicacions són molt poques, ja que únicament pot tornar un resultat concret, encara que aquest hagi requerit el moviment de centenars o milers de registres de la base de dades.

L'evolució d'aquests sistemes ha estat constant, i ha calgut anys per a poder considerar-los madurs.

L'evolució d'altres tipus d'aplicatius ha fet que el que ja consideràvem com un bon repositori de dades de caràcter general, hagi passat a no poder emmagatzemar tipus de dades complexos, pertanyent per exemple, als valors dels sistemes CAD, o d'enginyeria, fet provocat principalment per l'ortogonalitat del sistema d'emmagatzemament relacional.

La rigidesa de la esmentada ortogonalitat de les bases de dades relacionals, ve donada principalment per la seva evolució. Tot el funcionament definit en el SQL-92 esta basat *en les files i columnes de les taules*. Aquesta mateixa rigidesa de funcionament que fa que les dades siguin fàcilment manipulables quan s'està tractant amb grans quantitats de dades d'un mateix tipus, amb un nombre de interrelacions mesurable en temps de disseny, fa que sigui difícil de fer encabir informació amb un nombre de relacions no previsible en temps de disseny (en entorns d'aplicacions d'Intel·ligència Artificial, en sistemes CAD/CAE, per exemple), o amb uns atributs no coneguts en nombre, nom o tipus.

Les taules són ideals per a guardar gran quantitat d'informació amb unes característiques conegudes i dimensionades.

Així en la definició del magatzem per un tipus d'informació, es defineix el tipus de dada, la mida que tindrà cadascuna d'aquestes dades, i si cal propietats avançades per a optimitzar l'accés, o fins i tot per a poder distingir diferents aparicions entre elles.

La definició del tipus de dades que s'emmagatzemarà, es realitza en temps de creació de la taula, o sigui que, a priori s'ha de saber que s'hi guardarà.

En sistemes en que s'han d'emmagatzemar les especificacions, mides, interrelacions entre components i molts altres paràmetres que van creixent segons els prototips avancen, no es saben a priori les dades necessàries per a crear les taules per a contenir aquesta informació.

En tot cas, la complexitat en les relacions de les diferents taules que caldrien per a tenir tota la informació lligada, seria tant gran, que el sistema restaria inviable.

Davant aquests fets, s'està treballant en l'evolució natural de les bases de dades relacionals, per evolucionar-les a nous sistemes relacions, amb ampliacions, per a poder encabir aquests tipus de dades, gràcies a l'existència de tuples, herència i d'altres mecanismes.

3. Definició de les ORDBMS segons el 'The Object-Oriented Database Manifesto'

Aquest article va aparèixer amb la voluntat de ser un punt de reflexió del que haurien de ser en un futur, les bases de dades orientades a l'objecte.

Com que fins al moment de la seva publicació no hi havia cap especificació que fes referència a possibles sistemes de bases de dades relacionals orientades a l'objecte, aquest manifest hauria de ser un dels punt sobre el que es desenvoluparien els debats per a fer les especificacions/normes d'aquests productes.

L'article divideix les característiques que ha de tenir les ORDBMS en tres grans blocs, segons els autors entenen la necessitat, idoneïtat o possibilitats d'aquestes;

Obligatòries

Són aquelles característiques necessàries que ha de tenir un sistema de base de dades, per a poder ser denominat Sistema Relacional de Base de Dades Orientada a l'Objecte (ORDBMS).

Opcionals

Són les característiques que serien desitjables, però que per la seva complexitat o pel seu petit valor marginal, no es creu necessari que s'implementin en una primera etapa, malgrat si es considerin importants en ser assolits tots els objectius 'obligatoris'.

Obertes

Són aquelles parts que els fabricants poden incloure lliurement en les ORDBMS per a millorar la seva funcionalitat, o per a especialitzar-la en algun dels seus aspectes.

La necessitat de disposar de bases de dades orientades a l'objecte, sorgeix d'ambients on la informació no era propensa de ser emmagatzemada en el paradigma tradicional de les files i columnes, essent un clar exemple els entorns de treball CAD, CAE, que tenen particularitats tant especials, que fan que les seves dades siguin complexes de manipular i encara més d'emmagatzemar.

Així, certs atributs, tenen més complexitat en la seva ubicació i dependència, que no pas en el seu valor, cosa que fa evolucionar l'emmagatzemament d'aquests, fins aproximar-se al llenguatge que els tracta. Arriba a ser tant necessari tenir la mateixa especialització en la jerarquia alhora de fer les dades persistents, com en el moment de tractar-ho com a herència en el llenguatge de programació.

El primers punts a considerar en parlar d'un sistema d'emmagatzemament, són els següents:

- Cal que hi hagi persistència de les dades.
- Calen eines i gestió per manipular l'emmagatzemament secundari.
- Cal que diversos usuaris i processos puguin funcionar simultàniament sense interferències.
- Calen sistemes de recuperació consistents per evitar pèrdues de dades.
- Cal que es pugui resoldre qualsevol tipus de consulta.
- Calen mecanismes de consulta potents.

Tots aquests punts, són els que ha de tenir qualsevol SGBD estàndard del mercat.

Els punts que hauria de tenir una base de dades, en ser dissenyada com una ampliació de les SGBD, per arribar a ser una ORDBMS són:

- Possibilitat de treballar amb objectes complexes.
- Que cada objecte sigui únic.
- Que existeixi l'encapsulació.
- Que hi hagin tipus i classes.
- Que existeixi el concepte d'herència, encara que no cal que sigui de tipus 'múltiple'.
- Que existeixi la possibilitat d'anul·lació combinada amb unió posterior.
- Que existeixi l'extensibilitat.
- Que s'asseguri la completessa de càlculs.

3.1 Característiques obligatòries

Desenvoluparem aquests punts independentment:

3.1.1 Objectes complexos

Cal que es puguin construir models de dades útils per l'entorn en que es desenvolupi el sistema. Per tal cosa, cal que a partir de dades simples, es puguin crear estructures de dades noves, per a contenir de forma lògica i ordenada, aquelles famílies o estructures amb significat.

Aquests objectes, poden ser des de senzilles llistes, fins a complexes matrius o tuples de tipus.

Tots els objectes creats, han de mantenir la propietat de ser ortogonals, no fent falta d'aquesta forma tenir que considerar característiques particulars, evitant-se excepcions.

Es podria definir un Objecte Complex de tipus Adreca. En aquest cas, es podria definir un atribut de tipus adreca, contenint aquest d'aquesta forma, totes les propietats que s'han assignat a l'esmentat Objecte Complex.

Adreca	
TipusVia	INTEGER
NomVia	VARCHAR(40)
Numero	VARCHAR(9)
CodPostal	VARCHAR(5)
Poblacio	VARCHAR(40)
Provincia	VARCHAR(20)

3.1.2 Identitat dels objectes

Dos objectes poden ser iguals (el mateix), o bé poden tenir el mateix valor. A partir d'aquesta premissa, cal poder distingir els objectes entre sí, pel que cal que cadascun estigui identificat inequívocament. Per aconseguir això, cal que cada objecte tingui un identificador únic, havent la possibilitat de que aquest sigui introduït pel mateix sistema, o pel contrari, fent que aquest identificador sigui responsabilitat del usuari.

3.1.3 Encapsulació

Gran part de l'èxit del nou paradigma de la programació basada en objectes, es deu a les millores que s'introdueixen en el reaprofitament del codi, i en les millores aconseguides en la seguretat dels nous sistemes.

El reaprofitament i la nova robustesa són conseqüència directa d'haver aconseguit allunyar diferents capes de programació i/o de dades entre si. Amb l'encapsulació, aconseguim que els programes accedeixin a les dades mitjançant una interfície declarada, per la dada concreta.

Així per exemple, per a llegir el valor d'un salari d'un treballador, haurem de cridar una rutina (mètode) que serà l'encarregada de passar-nos el valor que és del nostre interès. En el cas de que per exemple el tipus de moneda canviï, no cal canviar l'aplicació ni les dades, únicament cal declarar un mètode de lectura nou per aquest valor, retornant-lo després de haver fet la conversió corresponent.

3.1.4 Tipus i classes

Les dos definicions s'apropen molt en concepte, però hi ha subtils diferències entre aquestes.

Els tipus poden ser estructures de dades, que podran ser reconegudes en temps de compilació, i per tant, molt menys propensos a error i/o fallades en temps d'execució.

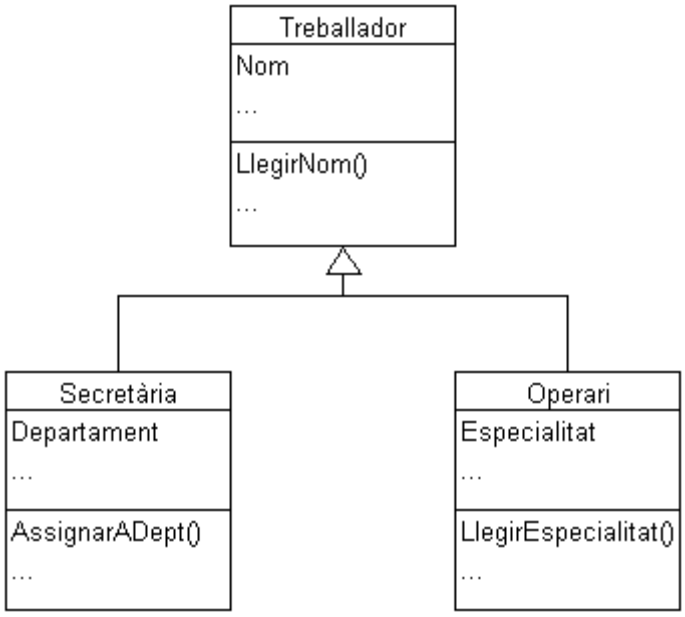
Diem que pot ser reconegut en temps de compilació, quan l'estructura creada ha de ser validada per ser introduïda en la SGBD. En el moment en que es produeix aquesta validació, seran realitzades les comprovacions necessàries per assegurar la correcció del nou tipus.

Les classes consten del constructor i les dades pròpiament dites. Són instanciables i manipulables, pel que difícilment són totalment comprovables en temps de compilació. Les esmentades comprovacions s'hauran de fer en temps d'execució, ja que durant aquesta, es podran crear i eliminar.

3.1.5 Herència

L'herència permet especificar el comportament i les propietats d'un objecte, amb el senzill procediment de l'especialització, de la substitució, de la inclusió o de les restriccions.

Així, es podrà definir un objecte amb unes determinades característiques, per recollir sota aquest, altres amb més restriccions, alhora que podrem fer que uns tercers siguin fills d'aquest, amb el que encara podrem acotar més una estructura de característiques en forma d'arbre.



Exemple d'herència per generalització.

Diem que existeix herència per substitució si es poden realitzar més operacions sobre el objecte fill que sobre el pare, mentre que si la herència és per inclusió, cada objecte fill està inclòs dintre dels objectes que tenen les característiques del pare.

El concepte d'herència per restricció, la trobem en aquells casos, en que els objectes que hereten, tenen més restriccions que els objectes pare.

Quan parlem d'herència per especialització, estem indicant que els objectes fills són més específics que els pares, tenint per exemple mètodes addicionals per accedir a dades que no existeixen en l'objecte d'on hereta.

3.1.6 Anul·lació, sobrecarrega i unió posterior

Bàsicament estem contemplant el tenir diversos mètodes d'igual nom, però diferenciats internament en el sistema pels seus paràmetres.

Un exemple fàcilment observable, és quan es fa una suma de variables del tipus numèric o de tipus alfanumèric. En l'últim cas, s'està realitzant una concatenació, mentre que en el primer s'està fent un càlcul matemàtic.

Observem que una mateixa operació realitza crides a funcions diferents, dependentment dels tipus dels paràmetres, simplificant així en gran mesura la tasca de programació i la complexitat del sistema final.

3.1.7 Completessa de càlculs

Cal que el llenguatge sigui prou ric i extens, com per a poder accedir als recursos del sistema i desenvolupar amb aquest aplicacions complexes.

Aquestes seran més extenses que les aplicacions tradicionals amb accés a les bases de dades.

3.1.8 Extensibilitat

Cal que els tipus bàsics i els predefinits puguin ser ampliats amb els tipus definits pels propis usuaris.

Els nous tipus hauran de ser igual als interns (des del punt de vista de la base de dades), no havent diferencia entre aquests i els natius.

3.1.9 Persistència

Cal poder preservar les dades, durant i després de l'execució del programa.

Cal que aquesta persistència sigui ortogonal, ja que el tipus de dades no ha d'importar per a que es puguin emmagatzemar.

El mateix procediment o instrucció ha de servir tant per a emmagatzemar les dades d'una tupla, com per a emmagatzemar les dades d'un objecte (per exemple), no havent de tenir en compta excepcions, segons el tipus de dades amb el que s'estigui tractant.

3.1.10 Emmagatzemament secundari

El motor de la base de dades ha de tenir mecanismes automàtics per a optimitzar les consultes, creant índexs si cal, o utilitzant mecanismes estadístics o alternatius per a millorar i assegurar el rendiment d'aquesta. Tots aquests sistemes han de ser transparents al usuari, i han d'afegir l'abstracció necessària per a separar la part física de la lògica.

3.1.11 Concurrència

Cal assegurar que la base de dades manté la coherència i la funcionalitat en executar varies peticions simultànies (de varis processos o usuaris).

Així cal que executi atòmicament les operacions, comprovant que no hi ha dependències entre les diverses peticions.

Cal en tot moment considerar les necessitats de complir amb el que s'anomena regles d'integritat ACID (Atomicitat, Consistència, Isolament i Definitivitat de les dades).

Ens podríem trobar amb un exemple en que dos processos intentessin actualitzar una taula i que donada la simultaneïtat d'aquestos, es crees una inconsistència de dades. En aquest cas, s'estaria incomplint la propietat de Consistència, pel que caldria assegurar l'ordre d'execució de les instruccions.

Tot seguit es pot observar com s'executen dues tasques simultàniament, sense que hi hagin els esmentats problemes de consistència.

Transacció 1 (confirmació d'una factura)

```
begin work;
set transaction isolation level serializable;
update FACTURES set confirmada = 'S' where numero = 999;
commit work;
```

Transacció 2 (revisió d'una factura)

```
begin work;
set transaction isolation level serializable;
select confirmada from FACTURES where numero = 999;
update FACTURES set revisada = 'S' where numero = 999;
commit work;
```

3.1.12 Recuperació

En tot moment s'ha de poder recuperar la informació d'una base de dades. Incloent els casos en que s'espatlla un disc dur, o el propi processador.

Caldrà establir polítiques de còpies de seguretat i mecanismes de tipus dietari, per a poder restaurar en un punt la base de dades, mantenint la integritat d'aquesta.

3.1.13 Facilitats de consulta

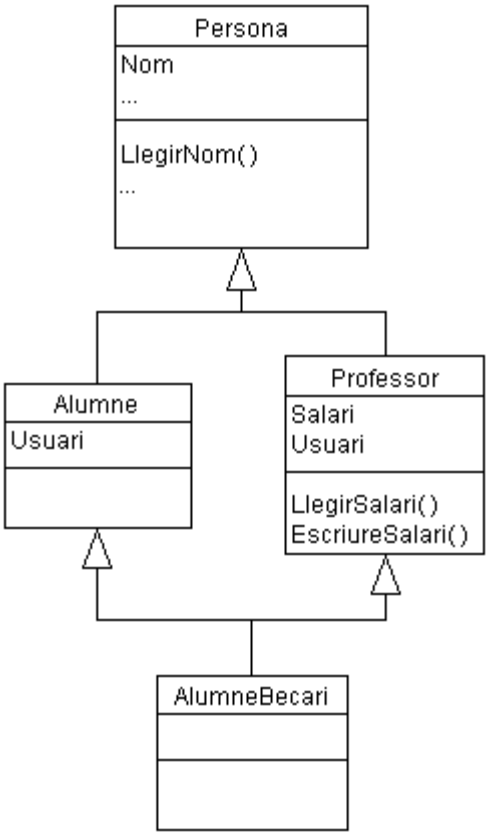
Cal que la base de dades disposi d'un sistema de consulta prou simple i potent, com per a extreure la informació necessària de forma ràpida i senzilla.

El propi motor de base de dades ha d'analitzar les consultes fetes, a fi d'optimitzar aquestes, segons l'estructura de les dades que s'han de consultar.

3.2 Característiques Opcionals

3.2.1 Herència múltiple:

Una de les característiques que seria preuada en les bases de dades Orientades a l'objecte, seria el suportar l'herència múltiple, resolent de forma transparent els possibles conflictes d'herència repetida.



Exemple d'herència repetida.

En l'esquema, es pot observar el cas d'una classe anomenada AlumneBecari, que hereta d'Alumne i de Professor. En aquest cas, s'hauria de solucionar el problema d'heretar per dos costats els atributs de Persona.

3.2.2 Comprovacions en temps de compilació:

En el cas de que s'aconseguís que tots els tipus foren reconeguts en temps de compilació, s'evitarien possibles errors en temps d'execució, ja que aquests haguessin estat detectats en una primera instància, millorant aquest fet la fiabilitat i manteniment dels sistemes.

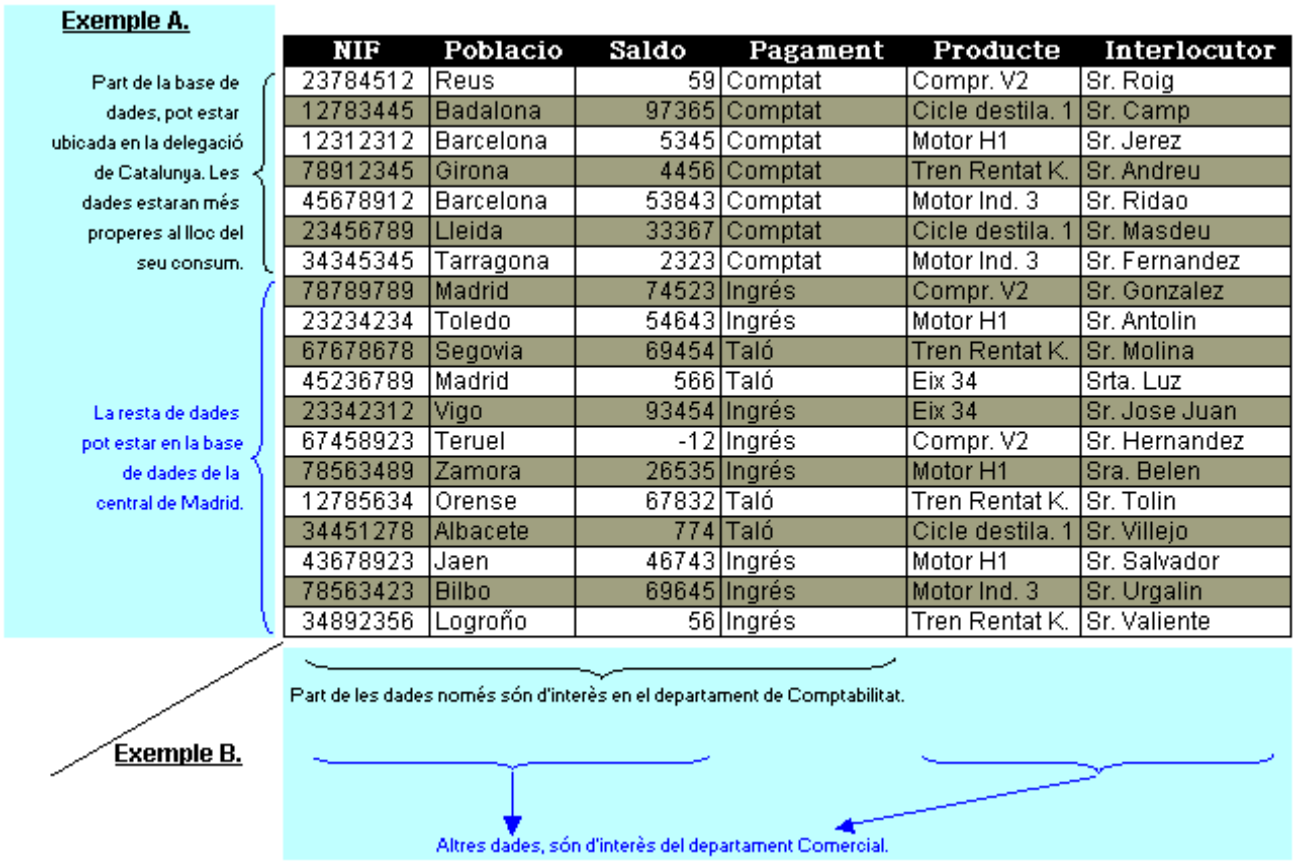
3.2.3 Distribució de la base de dades:

També fora aconsellable que la base de dades estigues repartida físicament entre maquinari divers, aconseguint així una millor qualitat de servei i un augment en la fiabilitat d'aquesta.

Per augmentar encara més la fiabilitat, es podria optar per fer que aquest entorn distribuït fos mixt, composant-se de maquinari de fabricants diversos, executant cadascun diferents Sistemes Operatius o versions d'aquests, assegurant així que una fallada de maquinari o del SO no afectes a tots els nodes.

També es pot fer que una base de dades sigui distribuïda, per a eliminar trànsit de xarxa.

En el cas de tenir centres de treball distribuïts geogràficament, pot ser raonable fer que part de la base de dades que s'utilitza en determinada seu, estigui ubicada en el servidor de la mateixa, augmentant així la velocitat de resposta del sistema.



NIF	Poblacio	Saldo	Pagament	Producte	Interlocutor
23784512	Reus	59	Comptat	Compr. V2	Sr. Roig
12783445	Badalona	97365	Comptat	Cicle destila. 1	Sr. Camp
12312312	Barcelona	5345	Comptat	Motor H1	Sr. Jerez
78912345	Girona	4456	Comptat	Tren Rentat K.	Sr. Andreu
45678912	Barcelona	53843	Comptat	Motor Ind. 3	Sr. Ridao
23456789	Lleida	33367	Comptat	Cicle destila. 1	Sr. Masdeu
34345345	Tarragona	2323	Comptat	Motor Ind. 3	Sr. Fernandez
78789789	Madrid	74523	Ingrés	Compr. V2	Sr. Gonzalez
23234234	Toledo	54643	Ingrés	Motor H1	Sr. Antolin
67678678	Segovia	69454	Taló	Tren Rentat K.	Sr. Molina
45236789	Madrid	566	Taló	Eix 34	Srta. Luz
23342312	Vigo	93454	Ingrés	Eix 34	Sr. Jose Juan
67458923	Teruel	-12	Ingrés	Compr. V2	Sr. Hernandez
78563489	Zamora	26535	Ingrés	Motor H1	Sra. Belen
12785634	Orense	67832	Taló	Tren Rentat K.	Sr. Tolin
34451278	Albacete	774	Taló	Cicle destila. 1	Sr. Villejo
43678923	Jaen	46743	Ingrés	Motor H1	Sr. Salvador
78563423	Bilbo	69645	Ingrés	Motor Ind. 3	Sr. Urgalin
34892356	Logroño	56	Ingrés	Tren Rentat K.	Sr. Valiente

3.2.4 Transaccions a mida

Poder elegir el tipus i mida de les transaccions entre els clients i els servidors, amb objectiu de fer que aquestes siguin més adients segons cada cas.

3.2.5 Manteniment de versions

Una altra característica desitjable, estaria el poder realitzar consultes i operacions amb la base de dades, tenint en compta el valor dels registres al llarg de l'historia de la base de dades.

Aquest fet que ens permetria executar una ordre de l'estil; restaura l'estat de la BD a dia X, hora Y, minut Z.

L'espai necessari per mantenir tots els diferents valors que ha tingut la base de dades al llarg del temps, és gran, però pot ser molt valuós ja que la pròpia BD ens pot facilitar moltes comparatives anuals (periòdiques) de dades.

```
SELECT NomPoblacio, Mitjana FROM Ciutats['epoch', 'now']
    WHERE Nom = 'Barcelona';
```

De treballar amb una base de dades que disposes d'aquesta característica, fora possible executar la instrucció de l'exemple, per aconseguir les xifres de poblacions que ha tingut la ciutat consultada al llarg del temps.

3.3 Característiques Obertes

3.3.1 Paradigma de programació

Com que no hi ha raó per a obligar a utilitzar un tipus de paradigma de programació concret, seria desitjable que es pugues utilitzar qualsevol dels paradigmes més o menys tradicionals: programació lògica, programació funcional, programació imperativa, programació múltiple, etc.

4. SQL:1999 – Versió 3

L'última versió de SQL es coneguda com a SQL:1999, per evitar la possible futura confusió (en el cas d'anomenar-la SQL3), amb una versió SQL 2003.

Els nous paradigmes de programació i les noves necessitats, van impulsar la normalització del nou estàndard per part dels comitès ANSI X3H2 i ISO DBL.

El fet de definir noves característiques pel SQL, estava encaminat a afegir el suport per a poder treballar amb objectes. Els paradigmes mínims que s'havien d'assolir per a completar els objectius eren:

- Que hi hagués persistència per a objectes,
- Que s'implementes l'herència simple.
- Disponibilitat de taules tipades.
- Existència de tipus definits pel usuari.
- Existència tipus 'col·lecció'.
- Nous tipus fila i referència.
- Millores en el suport d'objectes grans.
- Introducció de la recursivitat.

4.1 Herència

L'herència consisteix en la capacitat de definir un objecte a partir de les diferències que té respecte d'un altre, i és el principal concepte que fa diferenciar la definició del SQL:1999 de la del 92.

L'herència és primordial per a poder dissenyar aplicacions en les que les columnes de les taules no estan clarament identificades, tal com passa en certs sectors d'activitat que funcionen amb tipus de dades que no són fàcilment col·locables en les estructures de dades tradicionals (taules amb registres).

En sectors en que la uniformitat de la informació és molt baixa, cal defugir de l'ús de les SGBD tradicional per varis motius;

- Costos elevats de disseny del sistema.
- Baixa utilització (en volum) de les estructures dissenyades.
- Altíssima complexitat (gran quantitat de taules).
- Ampliabilitat i manteniment molt costosos.

L'herència és el concepte bàsic que permet solventar tots els punts que s'han enumerat com inconvenients habituals per a certs sectors. Per altres entorns, la nova tecnologia pot ser també un revulsiu de millora i estalvi, tal com s'analitzarà tot seguit.

Malgrat no ho sembli inicialment, les Bases de Dades Objecte-Relacional (ORDBMS), no tenen com a unitat d'emmagatzemament l'objecte, sinó que mantenen el concepte de fila-columna (matriu) del model relacional, però a

diferencia de les RDBMS, permeten incorporar objectes com a camps o columnes del model relacional.

Podríem dir que l'herència consisteix en la capacitat de definir un objecte a partir de les diferències que té respecte d'un altre, i que permet definir entitats per agregació o especialització.

La primera limitació que cal observar en l'herència definida en el SQL:1999, és que no és del tipus herència múltiple, ja que això hagués complicat excessivament l'apropament de les SGBD al mon dels objectes (s'hagués tingut que implementar tots els mecanismes necessaris per resoldre les possibles incoherències que pot generar l'herència múltiple).

El cas il·lustratiu que utilitzarem com exemple, serà el de PERSONA. Podem considerar en el nostre exemple, aquesta entitat com a 'pare' de la entitat TREBALLADOR.

Podríem definir les característiques bàsiques que tindrà qualsevol persona com atributs en PERSONA. Els detalls que són necessaris per definir TREBALLADOR s'afegiran en aquesta altra entitat.

En el moment de definir TREBALLADOR, s'ha d'indicar que és *inherit* (hereta) de PERSONAL. Un cop està així definit tot, podrem realitzar operacions amb TREBALLADOR (insert, update, delete) i la pròpia SGBD s'encarregarà de propagar les operacions necessàries a les taules vinculades. En les instruccions que utilitzem per interactuar amb la base de dades, no caldrà indicar de cap manera que ens estem referint a un element d'una taula (que està relacionada amb una altra). En referir-nos a TREBALLADOR, ens referim a tots els atributs que aquest té, des de la possible arrel de jerarquia d'herències.

Podríem portar aquesta teoria al mon real, dintre de l'àmbit del sector industrial, pensant per exemple en un *motor* constituït per molts altres components. Si les relacions d'herència estan prou ben implementades, es podrà fer un recorregut pels detalls dels components d'aquest enginy, fins arribar, per exemple, al proveïdor dels cargols més petits.

4.2 UDT / Tipus definits estructurats

En el paradigma de les DBMS tradicionals, els tipus existents eren els que els fabricants havien incorporat durant el disseny de cadascuna d'aquestes SGBD, mentre que en les especificacions del SQL1999, ens trobem amb noves especificacions, per les quals els usuaris es poden definir els seus tipus propis.

Es va decidir que d'apostar-se per l'orientació a objectes, hi haurien menys errors en els desenvolupaments, milloraria la intel·ligibilitat de les aplicacions, augmentaria la reusabilitat, i fins i tot el rendiment en descarregar els catàlegs, de les nombroses taules que caldrien.

Un nou tipus definit pot ser de qualsevol tipus disponible nativament en la base de dades, o altrament pot estar format per un nou tipus definit per l'usuari.

Al definir un tipus, podem crear un contenidor per les característiques d'un ens. Així el nou tipus pot ser per ex. un tipus que defineixi una adreça. Aquesta estarà composta pel tipus d'adreça (carrer, via, avinguda, etc.), pel nom d'aquesta, pel numero de la porta, pel codi portal, per la població, i per tot un seguit de detalls.

Un cop definit el tipus adreça, aquest podrà ser utilitzada tant per a contenir l'adreça dels treballadors (que estarien emmagatzemats a la taula treballadors), com per a contenir l'adreça dels clients o proveïdors. Aquest cas senzill, ja deixa entreveure l'estalvi en el disseny de l'aplicació, ja que fins i tot es reutilitzarà el tipus en diferents situacions, si aquest està prou ben dissenyat.

En el cas de que el nou tipus fos incomplet, es podria ampliar creant un altre tipus amb atributs addicionals, i incloent l'inicial com un dels seus atributs.

Segons el SQL:1999, utilitzaríem la següent sintaxis per a crear els esmentats tipus estructurats:

```
CREATE TYPE NomDelTipus [UNDER NomDelSuperTipus]
[<external Java type clase>]
[AS representation]
[ [NOT] INSTANTIABLE]
[NOT] FINAL
[ReferenciaALEspecificacioDelTipus]
[<referencia a l'opció de conversió>]
[opcions de conversió]
[llistat de mètodes]
```

Es pot observar a l'exemple que hi ha moltes clàusules no obligatòries. Un exemple pràctic (i bàsic) podria ser:

```
CREATE TYPE adreca AS
   (
   CarrerNom  CHARACTER VARYING(30),
   Numero     CHARACTER(6),
   Ciutat     CHARACTER VARYING(30)
   )
   NOT FINAL;
```

4.3 Tipus definits distint

De vegades no cal definir nous tipus, si no que únicament cal diferenciar els 'tipus' d'unitats que s'utilitzen.

Encara que dos unitats del mon real utilitzin els enters (per exemple) per quantificar una de les seves característiques, aquestes no tenen per que ser ni comparables ni sumables.

Així, podríem dir que estaria un error sumar el pes i el volum (per exemple) de pomes i de totxanes, malgrat tot fos indicat numèricament.

Per evitar aquests errors, declararem cadascuna de les famílies d'unitats aïlladament, eliminant així les operacions entre elles.

Per crear els esmentats tipus distint, utilitzaríem la següent sintaxis;

```
CREATE DISTINCT TYPE <NomTipus>
  AS <tipus sql>
  FINAL
```

I un exemple representatiu, on primer creem els tipus i després la taula que els utilitza:

```
CREATE TYPE t_pes      AS INTEGER FINAL;
CREATE TYPE t_llargada AS INTEGER FINAL;
```

```
CREATE TABLE prova
  (
  id       INTEGER,
  pes      t_pes,
  llargada t_llargada
  )
```

En el moment d'intentar fer l'operació:

```
select id+pes from prova where id = 1234;
```

La SGBD ens indicaria d'una errada per diferencia de tipus.

4.4 Col·leccions de tipus

S'utilitzen per a emmagatzemar en una única columna múltiples valors, o un vector. Amb aquest mecanisme, es poden crear les anomenades *nested tables,* que no són més que taules contingudes dintre d'una columna d'una taula.

Tornarem a utilitzar l'exemple de la taula treballador, per a il·lustrar la utilització d'una taula col·lecció; En aquest cas, podríem crear una taula on constes un identificador únic, el nom i els cognoms, i tot seguit un vector amb 12 posicions, on podríem emmagatzemar el salari per cadascun dels mesos de l'any.

En una columna, podem tenir un vector, una llista, un set (col·lecció sense duplicats) o un multiset (una col·lecció amb duplicats).

Les col·leccions de tipus es poden utilitzar ortogonalment per a qualsevol tipus estàndard dels definits pel SQL, a excepció de ells mateixos. Per fer referència a un dels elements de la col·lecció, es pot utilitzar l'ordinal que li

correspon en l'ordre, o un rang d'ells per a fer operacions sobre diversos d'aquests.

Segons la sintaxis del SQL:1999, una taula amb col·leccions de tipus s'hauria de definir de la següent forma:

```
CREATE TABLE exemple
  (
  id     INTEGER NOT NULL PRIMARY KEY,
  valors INTEGER ARRAY[3][3][3]
  )
```

En aquest cas, en lloc de definir un vector, s'ha definit un cub que permetrà emmagatzemar 27 valors. Per fer referència a un dels valors, s'hauran d'utilitzar les coordenades x,y,z de la seva ubicació, tal com es mostra tot seguit:

```
select id, valors[1][1][1] from exemple where id = 1;
```

4.5 Tipus ROW

És un tipus constructor, similar al tipus *collection*, però en aquest cas, tenim dos o més atributs en una única columna d'una taula.

Podríem definir una columna que tingues per nom persona (per exemple), que tingues com a detall cognoms i nom. D'aquesta forma, evitem definir (segons l'exemple) persona com a tipus, simplificant el manteniment de l'estructura de taules.

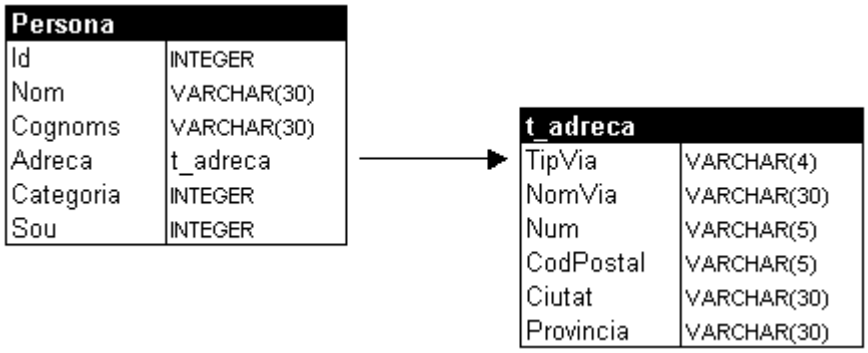
4.6 Tipus referència

Ens referim a un tipus de caràcter 'referència', quan aquest descriu perfectament totes les característiques de l'ens que representa.

Aquest tipus referència només podrà contenir valor que referencien instàncies de un tipus especificat, i pot contenir múltiples atributs (files). No són constructors.

4.7 Taula tipada

Podem dir que una taula és tipada, quan aquesta té com a únic atribut un tipus referència. Així si en declarar la taula, indiquem en l'únic atribut que és un tipus referència. La aula únicament es referirà a l'objecte o ens al que fa referència el tipus que hem indicat.



El detall al que podem arribar amb les taules tipades és important, ja que per un costat es pot fer que hereti d'una taula, i per l'altra, podem fer que el seu 'únic' atribut sigui un descriptor perfecte de la diferència amb el objecte que estem especialitzant. Així, podríem crear una nova taula cotxe, que hereta d'una taula vehicle, que estigues únicament construïda amb un atribut, que alhora seria un tipus que contindria totes les descripcions necessàries per definir aquest tipus de vehicle.

5. Disseny d'un cas pràctic

S'ha triat un cas de complexitat reduïda per posar en pràctica les possibilitats que defineix l'estàndard SQL1999.

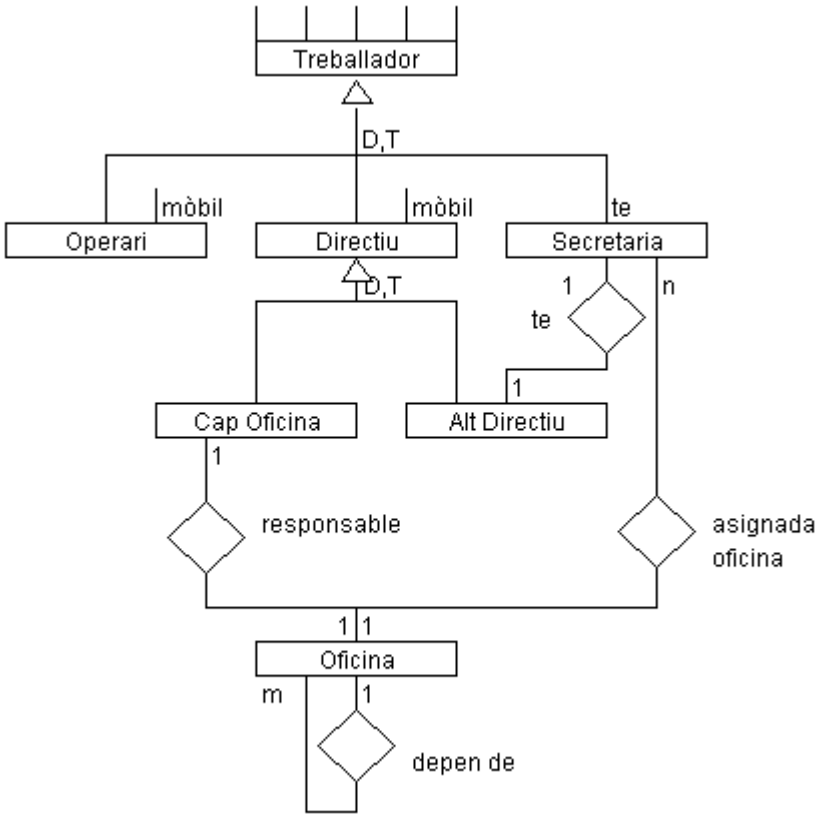
Per poder copsar adequadament les avantatges que incorpora aquesta versió de la definició del SQL, vers la del 92, es desenvoluparà per separat el mateix cas en mode relacional pur, i en mode orientat a objectes. Després es posarà en pràctica amb una base de dades comercial, tenint en compta les possibilitats que suporti l'esmentada base de dades (en aquest cas PostgreSQL).

El cas en qüestió, descriu el funcionament d'una agencia de transports. Aquest cas és un *subset* del presentat a l'assignatura Bases de Dades I com a pràctica

El desenvolupament es centra en la part jeràrquica dels treballadors. Aquest fet es degut a què amb la complexitat elegida, es poden portar a terme les operacions del SQL:1999 que es volen comprovar. A més, és un cas tant genèric, que es pot trobar pràcticament en qualsevol aplicació que incorpori la part de personal d'una empresa.

5.1 Utilitzant un sistema relacional

Seguint l'esquema conceptual, comprovarem les relacions necessàries (claus primàries i foranies), que hem de tenir per poder explotar adequadament el model.



Tenim una taula anomenada **Treballador**, amb una clau primària *Codi_Treballador*. Aquesta clau és mantinguda per la pròpia aplicació, evitant utilitzar DNI amb l'objectiu de prescindir de les comprovacions necessàries (la lletra, si és DNI, si és NIE, etc.), i amb el propòsit addicional de poder posteriorment experimentar amb els dispensadors o seqüències.

Operari és una taula que referència a **Treballador**, ja que gran part dels atributs que foren necessaris per descriure a un treballador, estan en aquesta primera taula. L'atribut *codi* d'aquesta entitat, és el vincle que permet lligar el treballador amb **Treballador**. La relació és senzilla, i un **Operari** és obligatòriament un **Treballador** (ha d'existir un registre amb el mateix *codi* que a **Operari**), i ser un **Operari**, és una de les tres possibles opcions per a un **Treballador**.

La taula **Directiu** té una relació similar amb **Treballador** (amb clau primària *codi*), però posteriorment serà relacionat per **Cap_Oficina** i per **Alt_Directiu**.

Secretària està relacionada amb **Treballador** per la clau primària *codi*. Estem en un cas similar a **Operari**, únicament que aquest cop tindrem que altres relacions faran referència a aquesta taula. Aquí ja s'ha de contemplar el fet de que una *Secretaria*, pot fer tasques per a varis *Alts_Directius*, pel que caldrà una taula que permeti lligar l'esmentada relació.

Cap_Oficina. És una taula relacionada per una banda amb *Directiu*, i per una altra amb *Oficina*. Com que cada *Cap_Oficina*, pot ser-ho de varies, caldrà una taula intermèdia per emmagatzemar aquesta relació. La clau principal segueix essent codi, fet que era previsible, ja que caldrà per lligar-ho amb **Treballador**.

Alt_Directiu és un **Directiu**, que és **Treballador**. Aquest pot ser ajudat per una secretària (que té en exclusivitat). Per poder realitzar aquestes relacions, cal la clau primària *codi*.

La taula **Oficina**, conté la informació de cadascuna de les seus de l'empresa. Aquestes estan enumerades amb un codi únic (*codi_oficina*), que ens servirà com a clau primària. Els atributs són els mínims necessaris per demostrar la validesa del model, i únicament cal distingir l'atribut *depende_oficina*, que servirà per relacionar-ho amb la mateixa taula, per si una oficina depèn d'una altra.

La taula **Responsable** s'implementa en el model per a poder emmagatzemar la relació múltiple entre *Cap_Oficina* i **Oficina**.

Cada *Cap_Oficina*, pot tenir vàries **Oficines** sota la seva responsabilitat, alhora que pot ser que una **Oficina** hagi tingut al llarg del temps, diferents *Caps_Oficina*.

En tots els casos, cal emmagatzemar la data d'inici d'un responsable en una oficina determinada, pel que la taula que ens serveix per a realitzar les relacions, tindrà a més atributs addicionals.

El codi SQL per crear les taules esmentades, és el següent;

```
CREATE TABLE OFICINA
( codi_oficina       CHAR(10),
  adreca             CHAR(50),
  fax                CHAR(9),
  telefono           CHAR(9),
  depende_de_oficina CHAR(10),
PRIMARY KEY
 (codi_oficina),
FOREIGN KEY
 (depende_de_oficina) REFERENCES OFICINA
);
```

```
CREATE TABLE SECRETÀRIA
( codi             CHAR(10),
  oficina          CHAR(10) NOT NULL,
PRIMARY KEY (codi),
FOREIGN KEY (codi) REFERENCES TREBALLADOR,
FOREIGN KEY (oficina) REFERENCES OFICINA
);
```

```
CREATE TABLE RESPONSABLE
( oficina          CHAR(10),
  data_inici       DATE,
  Cap_Oficina      CHAR(10),
  data_fi          DATE,
PRIMARY KEY (oficina, data_inici),
FOREIGN KEY (oficina) REFERENCES OFICINA,
FOREIGN KEY (Cap_Oficina) REFERENCES CAP_OFICINA,
UNIQUE (data_inici, Cap_Oficina)
);
```

```
CREATE TABLE ALT_DIRECTIU
( codi             CHAR(10),
  secretària       CHAR(10) NOT NULL,
PRIMARY KEY (codi),
FOREIGN KEY (codi) REFERENCES DIRECTIU,
FOREIGN KEY (secretària) REFERENCES SECRETÀRIA
);
```

```
CREATE TABLE CAP_OFICINA
( codi             CHAR(10),
PRIMARY KEY (codi),
FOREIGN KEY (codi) REFERENCES DIRECTIU
);
```

```
CREATE TABLE DIRECTIU
( codi             CHAR(10),
  mobil            CHAR(9) NOT NULL,
PRIMARY KEY (codi),
FOREIGN KEY (codi) REFERENCES TREBALLADOR
);
```

```
CREATE TABLE OPERARI
( codi              CHAR(10),
  mobi              CHAR(9) NOT NULL,
PRIMARY KEY (codi),
FOREIGN KEY (codi) REFERENCES TREBALLADOR
);
```

Les dades mínimes i bàsiques de prova que s'han d'introduir per comprovar la correctessa en el procés de creació de les taules, són:

```
INSERT INTO OFICINA VALUES ('O-0001', 'C/Margarita, 15, Madrid',
   '916242400', '916242401', NULL);
INSERT INTO Treballador VALUES ('E-0001', 'JOSE', 'SANCHEZ MARTINEZ',
   '1111111111', '01-01-2000', '00000000001111111111');
INSERT INTO Operari VALUES ('E-0001', '666112233');
INSERT INTO Directiu VALUES ('E-0002', '612345678');
INSERT INTO Secretària VALUES ('E-0003','O-0001');
INSERT INTO Cap_Oficina VALUES ('E-0006');
INSERT INTO Alt_Directiu VALUES ('E-0002', 'E-0007');
INSERT INTO Responsable VALUES ('O-0001', '01/01/2000',
   'E-0006','12/31/2002');
```

S'han elegit una *select* especialment pensada per a comprovar el correcta funcionament de les relacions:

```
select nom, cognoms
  from operari
  join treballador on (treballador.codi_treballador=operari.codi)
  where operari.mobil='666112233';
```

El resultat retornat és;

```
 nom                  | cognoms
----------------------+--------------------------------
 JOSE                 | SANCHEZ MARTINEZ
```

Comprovant així que la creació de les taules i de les relacions ha estat correcta.

Altres proves addicionals que s'han realitzat, però que no s'inclouen aquí, han estat;

- Carrega múltiple de dades.
- Verificació del correcta funcionament de les claus primàries.
- Verificació de que el UNIQUE funciona correctament.
- Verificació de que les seqüències funcionen correctament.

5.2 Utilitzant orientació a l'objecte segons SQL:1999

En el moment d'interpretar el disseny elegit per a fer l'esquema segons l'estàndard definit al SQL 1999, ens trobem amb varies particularitats que cal definir:

Model que s'analitza:
- S'utilitza el mateix cas emprat en el model relacional. Una part del disseny de l'empresa de transport és suficient per a demostrar la viabilitat i l'eficiència, de les implementacions en model Object/Relational.

Transformacions del model que es realitzaran per a una millor comprensió del model Orientat a Objectes:
- Es realitzen petites variacions en els atributs de les entitats, alhora que es crea una de nova com a arrel de la jerarquia. Es crea l'entitat **Persona** per a poder generalitzar més el cas, i no hipotecar futures ampliacions del sistema aquí descrit.

Anàlisi del model final:
- La versió final, s'analitza amb totes les modificacions esmentades, per a poder aprofitar millor el model Object/Relacional. Aquest model teòric, patirà posteriorment de diversos problemes d'implementació, ja que segons el disseny proposat (fet seguint les especificacions que hauria de tenir una base de dades que complís amb el estàndard SQL1999), no serà implementable degut a algunes mancances de la base de dades emprada. En aquest punt no s'ha cregut oportú considerar les limitacions comentades, per evitar limitar el model teòric.

Així les entitats que utilitzarem en el disseny, són les següents (els atributs es definiran posteriorment):
- Persona
- Treballador
- Operari
- Directiu
- Secretària
- Alt Directiu
- Cap Oficina
- Oficina
- Responsable

5.2.1 Elements implementables del SQL:1999, en el cas descrit

UDT

Trobem que existeixen varies propietats de les entitats que es repeteixen a diferents llocs, i que addicionalment el projecte millorarà tant en claredat com en l'apartat de manteniment, en el cas de fer que aquests siguin tipus.

- *Adreça*: Es pot considerar un UDT compost per *Tipus Via* (carrer, avinguda, carretera, etc.), el nom de la via, el numero d'aquesta on està l'element al que fa referència.
- *Població*: És el UDT compost per *Codi Postal*, *Població*, *Província*, *País*

La implementació en SQL 1999 del UDT Adreça seria;

```
CREATE TYPE t_Adreca AS
(
  TipusAdreca    CHARACTER VARYING(4),
  Nom            CHARACTER VARYING(20),
  Numero         INTEGER
  Poblacio       CHARACTER VARYING(20),
  Provincia      CHARACTER VARYING(20),
  CodPostal      CHARACTER(5),
)
NOT INSTATIABLE
NOT FINAL
```

Rows

En determinats casos, utilitzar un UDT per implementar una estructura de dades relacionades és massa complex. Per a no desvincular dades relativament properes, evitant la utilització dels UDT, es pot utilitzar la facilitat d'inserir en una única Row (columna) dos o més atributs.

- Evitarem en el cas de *Nom* i *Cognom*, utilitzar un UDT, ja que sembla més lògic (donat que la taula és a l'arrel de la jerarquia) inserir aquí, els dos atributs en una única columna.

L'exemple es podria desenvolupar de la següent forma;

```
CREATE TABLE PERSONA
(
  IdPersona   INTEGER,
  nomPer      ROW (
                   nom     CHARACTER VARYING(20),
                   cognoms CHARACTER VARYING(20)),
  nif         INTEGER,
  Adreca      t_Adreca,
  nss         CHARACTER VARYING(10) UNIQUE,
  d_naixament DATE,
);
```

Dominis

Les restriccions i regles que es poden indicar amb els dominis, afecten a tot el repositori de la base de dades, pel que no caldrà considerar en el codi de les aplicacions cadascuna de les restriccions de cadascun dels atributs. Únicament ens haurem d'assegurar de recollir els possibles errors de retorn que ens indiqui la base de dades, per assegurar-nos de que l'operació executada ha finalitzat correctament, o en cas contrari, per recollir el codi indicador de la causa de l'error. Alguns dominis que es creuen necessari implementar són:

- *Codi_Oficina*: Ens permetrà definir un valor inicial i una restricció que evitarà de poder inserir un null.
- *Codi_Postal*: Definirà algunes de les característiques que ha de tenir. Entre altres, el seu valor per defecte i que no pot ser null.

Seguint l'estàndard definit al SQL:1999, la codificació del exemple *Codi_Postal* seria;

```
CREATE DOMAIN Codi_Postal CHARACTER VARYING(5)
  CHECK (VALUE IS NOT NULL) DEFAULT '00000';
```

Assertions

Mitjançant aquesta facilitat, podrem definir restriccions genèriques a nivell de taula, per evitar així tenir que contemplar certes particularitats en el moment d'operar amb aquestes.

- En el moment de definir la taula **Responsable**, s'ha copsat la necessitat de realitzar la comprovació de que la data de inici i de fi, en que un responsable es fa càrrec d'una oficina, no es sobreposi amb les dates en que un altra responsable la té assignada. Si es cometes aquest error alhora d'introduir les dates d'un responsable, comportaria que una oficina constes en una determinada data, amb dos (o més) responsables, podent causar un error greu en el cas de fer una consulta que estigues preparada per un retorn d'una única fila.
- Podríem utilitzar un altra assertion per evitar que una secretària sigui compartida per masses directius. En aquest cas, únicament caldria fer un count de les vegades que apareix el Codi de la Secretària en la taula Alt_Directiu.

```
CREATE ASSERTION a_Secretària
   CHECK ( ( SELECT COUNT(*) FROM Alt_Directiu
             WHERE oldSecretària = Secretària) > 5);
```

Col·leccions de tipus

Trobem alguns casos en els que seria útil tenir la possibilitat d'emmagatzemar varis valors en la mateixa columna. Aquest cas pot ser el dels idiomes que té una secretària, o les xifres de varis acumulats. Sense tenir que fer una relació (amb claus primàries i el que comporta alhora de fer consultes), tenim accessibles uns valors que s'han desnormalitzat d'un disseny que estaria més basat en les relacions, però que permet una major agilitat, ja que pot evitar en una aplicació de mida mitjana, unes desenes de petites taules poc poblades.

- *Idiomes* secretària: En aquest cas no cal implementar una taula amb tots els idiomes possibles, ja que el nombre de secretaries és molt petit, i la possibilitat d'idiomes molt alta. Senzillament introduirem el nom de l'idioma que correspongui, quedant aquest com una descripció d'una característica de l'esmentat element.
- *Telèfons*: Donat que cada cop hi ha més diversitat de números de contacte, senzillament introduirem el nombre de números que calgui (ja siguin fixes o mòbils) en el vector que s'ha pensat per aquesta fi.

Si ho utilitzéssim a Oficina, ho hauríem de fer d'aquesta forma;

```
CREATE TABLE OFICINA
(
   codi        INTEGER,
   adreca      CHARACTER VARYING(50),
   tel         CHARACTER VARYING(9) [5],
   fax         CHARACTER VARYING(9) [5],
   dep_oficina CHARACTER VARYING(10)
);
```

En aquest exemple, s'indica que la mida màxima del vector és cinc.

Tipus distinct;

Podem determinar quins tipus són compatibles entre si, fent per exemple que determinats *integers* no es puguin arribar a sumar, per pertànyer aquests a dos tipus de dades diferents.

- No es aplicable a cap cas del model descrit.

Nested Tables

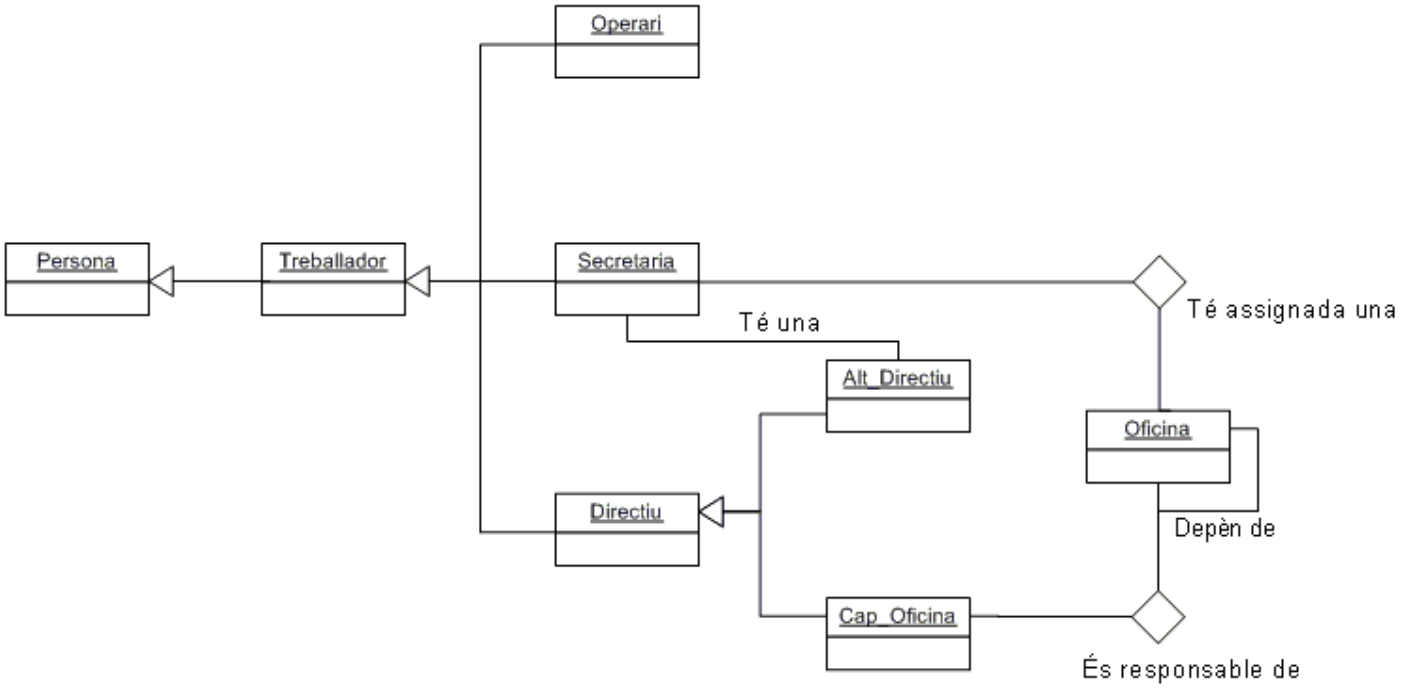
Trobaríem que aquesta característica és implementable en el cas de voler indicar quina característica, de un conjunt acotat, té un element d'una taula.

- En el cas Operari, podríem tenir una característica que indiques la especialització d'aquest. Com que el nombre d'especialitzacions es preveu baixa, es pot implementar amb les nested tables. Així *Especialitzacio* podria ser de *TipEspecialització*. Aquest tipus emmagatzemaria l'esmentat atribut.

Un cop vistes les particularitats que s'utilitzaran en la implementació de cada entitat, passarem a descriure les relacions d'herència entre aquestes. Després només caldrà formalitzar tot aquest disseny, amb les instruccions que caldrien de SQL:1999 per a poder-ho implementar en el cas de tenir una ORDBMS que acomplís totes les especificacions.

5.2.2 Herències necessàries

El model que estem desenvolupant, una vegada inclosa l'entitat arrel Persona, és:



Persona

És la primera entitat de l'estructura que es desenvolupa, i haurà de tenir tots els atributs bàsics que calen per definir, distingir i localitzar una persona.

Cal observar que no tots els atributs són obligatoris, ja que contemplem un cas d'ús futur més ampli (ja es contempla, per exemple, que hi podran haver persones menors d'edat, o sigui, sense DNI).

Treballador

Tot Treballador depèn de una persona. Aquesta entitat contindrà els atributs addicionals a Persona, que calen per definir les característiques d'aquesta entitat. Així, *data_inici* és necessari per a tenir constància de la data d'alta de la persona en l'empresa.

Sempre que es vulgui consultar les dades bàsiques (com a persona), d'un treballador, podrem consultar aquesta taula, ja que contindrà les dades especialitzades, més les inicials de Persona.

Operari

És una entitat dos nivells per sota de Persona, i un nivell per sota de Treballador. Hereta els atributs de Persona, els definits a Treballador, i afegeix *Zona*.

Directiu

Aquest cas és complex, ja que al igual que Operari hereta de Persona i de Treballador, per després ser el pare de herència posterior, per a *Cap_Oficina* i *Alt_Directiu*. Com a Directiu s'afegeixen els atributs *Data_Alta*.

Secretària

Es semblant a Directiu en complexitat, però addicionalment s'ha de tenir en compta que si bé hereta de Treballador (i per tant de Persona), després està relacionada amb Alt_Directiu i amb Oficina. Secretària podrà estar relacionada amb varis Alts_Directius, però només podrà tenir assignada una Oficina.

Cap_Oficina

Hereta de Directiu i té relació amb Oficina. Es pot comprovar que és necessària una entitat addicional (Responsable), ja que sense aquesta, no podria tenir relació amb més d'una oficina. També cal per a fer constar la data en que s'ha establert i la data en que finalitza aquesta responsabilitat.

Responsable

Emmagatzema les dates d'inici i de fi de la relació *Cap_Oficina* amb Oficina, alhora que permet que un únic *Cap_Oficina*, sigui el responsable de més d'una Oficina.

Oficina

S'elegiran els atributs bàsics per la seva implementació. No hereta de cap taula, però té relació amb Responsable i amb Secretària.

S'ha de fer notar en aquest cas, que a Oficina no hi ha cap indicació de quina és la Secretària assignada a aquesta.

Observacions:

Es creu que és necessari dotar d'una clau única interna i automàtica a les entitats Persona i Treballador, ja que segons s'ha exposat, es pot generalitzar l'ús de Persona, fins arribar a emmagatzemar en aquesta entitat persones sense l'esmentat document d'identitat.

Es creu que serà òptim, arribat el cas, de crear una seqüència per a fer que s'enumerin automàticament les Persones que s'insereixin a la taula.

Sempre es podrà fer una consulta utilitzant com a referència el DNI, però ens assegurarem en el cas de que aquest no existeixi, de poder diferenciar dos persones que tinguin el mateix nom.

La segona seqüència que es creu necessària, és la de Treballador. En aquest cas, utilitzarem com a clau primària el numero de matricula del

treballador. Aquest nombre és seqüencial (segons l'ordre en que s'incorporen els treballadors a l'empresa), pel que coincideix amb l'ordre en què es dispensaran els nombres amb el mètode proposat.

Tal com s'ha exposat, i donat que en el model SQL 1999 no existeix l'herència múltiple, caldrà dues relacions amb clau primària.

Secretària haurà de tenir com a clau primària *IdTreballador*, que com s'observarà és l'identificador únic de la entitat de la que hereta.

Amb Directiu i Cap_Oficina creem igualment una clau primària, utilitzant *IdTreballador* de l'entitat de les que hereten immediatament, o fins i tot a dos nivells per damunt.

5.2.3 Implementació del cas en model Object/Relational

Arribat a aquest punt d'estudi del disseny, s'ha comprovat que malgrat el model a implementar es senzill, permet incorporar moltes de les característiques definides a l'estàndard del SQL 1999.

La base de dades que s'utilitza en aquest cas, PostgreSQL, és una base de dades de llarga tradició (va aparèixer la primera versió al 1977), i curiosament, amb llicència de tipus BSD (similar a la GNU), es a dir, és d'ús gratuït. Aquest tipus de llicència, gairebé limita únicament als usuaris del programa, a haver de distribuir les possibles millores que facin en aquest, pel que es pot ja intuir, que estem parlant d'un sistema amb una gran comunitat de desenvolupadors que participen tant en el desenvolupament del mateix programa, com en el ús d'aquest, o sigui, en el desenvolupament d'aplicacions que utilitzen aquesta ORDBMS.

El model de llicència BSD, és pràcticament idèntic al que utilitza la distribució del sistema operatiu Linux, pel que es constata, un cop comprovada la popularitat i nivells d'ús als que està arribant, que és un model perfectament vàlid pel futur, assegurant l'existència d'aquest producte, independentment de marques i fabricants.

PostgreSQL basa gairebé tota la seva Orientació a Objectes, en el fet de considerar la fila d'una taula com un objecte.

Les taules poden heretar d'altres taules els seus atributs, afegint els seus atributs propis al resultat final.

Aquesta senzilla (i limitada) funcionalitat, facilita en molts casos la creació de sistemes, però al no poder-se definir en una columna més de un atribut (*row*), o fer que una columna faci referència a un tipus, o a una altra taula, limita l'ús d'aquestes funcionalitats, fins al punt de utilitzar-les com a simples possibilitats o extensions del Sistema de Gestió de Base de Dades (SGBD).

5.3 Proposta del model en PostgreSQL

Basant-nos en el disseny fet per a una ORDBMS que complís amb el estàndard SQL 1999, desenvoluparem el cas descrit, comprovant així les diferencies finals entre el model teòric i el pràctic.

En primer lloc crearem les seqüències i els dominis que utilitzarem;

```
CREATE DOMAIN od_CODI_OFICINA char(4) NOT NULL DEFAULT '0000';
CREATE DOMAIN od_CODI_POSTAL char(5) NOT NULL DEFAULT '00000';
```

Aquests dos dominis evitaran la inserció de valors NULL, i crearan els elements, en cas de no indicar valor per els camps Codi, amb valors zero (facilitarà la seva cerca posterior).

```
CREATE SEQUENCE os_IdPersona MINVALUE 0;
CREATE SEQUENCE os_IdTreballador MINVALUE 0;
```

La seqüència pel identificador de persona, permetrà tenir una clau única per aquesta entitat, encara que inserim a gent que no tingui document d'identitat.

La seqüència que creem per l'identificador del treballador, servirà per assignar inequívocament el nombre de matricula del treballador.

Després d'això crearem l'entitat oficina. La creem en primer lloc, ja que no hereta de cap altra entitat;

```
CREATE TABLE ot_OFICINA
(  codi_oficina       od_CODI_OFICINA,
   adreca             varchar(50) NOT NULL,
   tel                varchar(9) [ ],
   fax                varchar(9) [ ],
   depende_oficina    varchar(10),
   PRIMARY KEY(codi_oficina),
   FOREIGN KEY(depende_oficina) REFERENCES ot_OFICINA(codi_oficina)
);
```

S'ha d'observar que té clau primària, i que la *foreign key* apunta a ella mateixa. Aquest fet es deu a que una oficina pot dependre jeràrquicament d'una altra.

En aquesta taula, ja s'han implementat el primer element descrit al SQL 1999. Tant a l'atribut *tel* com a *fax*, s'ha utilitzat col·leccions de tipus. S'ha pensat que no era pràctic fer una estructura complexa per a poder emmagatzemar tots els possibles números de telèfon, i que no era viable crear tants atributs com números de telèfon tingues l'oficina amb el nombre més alt de línies telefòniques.

La solució de compromís, és molt vàlida fins a un nombre de telèfons no molt gran, i permet en aquest cas, simplificar el model i addicionalment facilita el manteniment.

La taula persona serà la base de les taules que fan referència a les persones que treballen a l'empresa (Treballador).

Tal com ja s'ha raonat, es creu convenient tenir un identificador únic. Aquest cal que estigui controlat per la pròpia base de dades. Així ja en la creació de la taula definim que aquest valor sigui agafat de la seqüència definida per aquest cas.

```
CREATE TABLE ot_PERSONA
( IdPersona    INT DEFAULT nextval('os_IdPersona'::text),
  nom          VARCHAR(20) NOT NULL,
  cognoms      VARCHAR(30) NOT NULL,
  nif          INT         NOT NULL,
  adreca       VARCHAR(40) NOT NULL,
  cod_postal   od_CODI_POSTAL,
  poblacio     VARCHAR(25) NOT NULL,
  nss          VARCHAR(10) UNIQUE,
  d_naixament  DATE,
  PRIMARY KEY(IdPersona)
);
```

No caldria que aquest identificador únic fos una *primary key*, però com que en un sistema dissenyat amb Orientació a Objectes, és altament costos modificar certes propietats de les entitats de més baix nivell, ja es preveu la creació d'aquesta clau primària, per poder ampliar futurament el sistema amb noves funcionalitats.

En aquest punt del disseny, s'ha comprovat que PostgreSQL no implementa les característiques que permetrien crear com a ROWS o com a UDT els conjunts de dades que constitueix l'adreça o el nom i cognoms.

PostgreSQL tampoc permet fer referència a una taula com si fos un tipus d'atribut, pel que l'expressivitat del disseny es veurà força reduïda.

L'entitat Treballador és la següent de la jerarquia;

```
CREATE TABLE ot_TREBALLADOR
( IdTreballador INT DEFAULT nextval('os_IdTreballador'::text),
  d_inici    DATE,
  cc         VARCHAR(20) NOT NULL,
  PRIMARY KEY(IdTreballador)
)
INHERITS(ot_PERSONA);
```

Al igual que a persona, s'utilitza una seqüència per assignar valor a IdTreballador.

No s'utilitza l'identificador únic heretat que tenim a persona, ja que en un futur, no heretarà en exclusiva Treballador de persona. Altrament, amb la idea

de que la clau única sigui el nombre de matricula del treballador a l'empresa, cal que aquest nombre comenci de zero, i sigui consecutiu.

En la creació d'aquesta taula, també s'ha d'observar una de les característiques definides al SQL 1999; l'herència. Malgrat el PostgreSQL no segueixi estrictament l'estàndard, es pot comprovar que en declarar que la taula és INHERITS, es fa que aquesta quedi declarada com a taula que hereta de ot_PERSONA.

La forma en que s'implementa és bàsica, senzilla, però no per això menys potent.

Si considerem que tenim una taula B que hereta d'una taula A, podríem explicar el seu funcionament de la següent manera:

- En el moment de la creació de la taula B, s'indica de quina altra hereta (A). Conceptualment s'ha d'entendre com una suma d'atributs. Tenim en la nova taula, els atributs de la taula en qüestió, més els de la que hereta.
- Així si féssim una consulta de la taula B, se'ns retornaria a_1, a_2, b_1, b_2, essent els subíndex, les columnes de les taules.
- En el cas d'inserir valors en la taula arrel (A), ens adonaríem en fer una consulta que els valors existeixen únicament en la taula en qüestió. El resultat de la consulta a la taula B ens indicaria que aquesta no ha esta modificada per l'acció d'inserció esmentada.
- En el cas d'inserir valors en la taula B, podrem verificar que aquests apareixen tant en la A com en la B, ja que la base de dades té relacionades ambdues taules, i crea un vincle d'herència entre elles.

Posteriorment, en inserir les dades de prova i fer les consultes adients, es veurà amb detall tot aquest funcionament.

La taula Operari té únicament un atribut, zona.

En passar del model relacional al model Object Relational ens adonem de que la clau única ja no cal. Aquest fet es degut a que la relació amb Treballador ja la mantindrà la mateixa base de dades, ja que s'indica que hereta d'aquest.

Altrament, i per mantenir aquesta entitat pel cas d'exemple que estem realitzant, s'ha inserit l'atribut zona, ja que s'ha cregut interessant.

```
CREATE TABLE ot_OPERARI
( zona  VARCHAR(20) NOT NULL)
INHERITS(ot_TREBALLADOR);
```

Es interessant observar que la indicació de que hereta de Treballador, es fa fora de la sintaxis original SQL92, tot just després de tancar els parèntesis del Create.

La taula Directiu, en aquest punt, és tant interessant com l'anterior taula.

Destacar que hereta de Treballador, i que s'ha eliminat la clau primària que tenia en el model Relacional, ja que hereta la que té Treballador.

```
CREATE TABLE ot_DIRECTIU
( d_alta    DATE)
INHERITS(ot_TREBALLADOR);
```

La taula Secretària és la que tanca el trio de taules que hereten de Treballador. Al igual que en les altres, s'ha eliminat la seva clau primària (ja que hereta la de Treballador), i s'ha utilitzat un domini en l'atribut *codi_oficina.*

S'ha de recordar que una secretària només pot tenir una oficina assignada.

```
CREATE TABLE ot_SECRETARIA
( codi_oficina   od_CODI_OFICINA,
  idioma         VARCHAR(30) [],
  PRIMARY KEY(IdTreballador),
  FOREIGN KEY(codi_oficina) REFERENCES ot_OFICINA(codi_oficina)
)
INHERITS(ot_TREBALLADOR);
```

L'atribut *idioma* s'ha definit com una col·lecció de tipus, per evitar crear una estructura de dades que permeti emmagatzemar n idiomes per secretària.

Si comprovem amb detall el codi emprat per crear la taula, observarem un detall d'importància: s'està utilitzant un atribut heretat com a clau primària.

La taula Secretària, per ella mateixa, no té l'atribut anomenat *IdTreballador*, però no obstant es fa referència a aquest per a indicar que és l'atribut que s'utilitza com a clau primària.

```
CREATE TABLE ot_CAP_OFICINA
( plus      INT,
  PRIMARY KEY(IdTreballador)
)
INHERITS(ot_DIRECTIU);
```

Si recorrem les relacions d'herència i comprovem els atributs de Treballador, trobarem l'esmentada columna.

La taula Cap_Oficina és la primera que trobem per sota de les que hereten de Treballador. Aquesta hereta de Directiu, que heretava de Treballador.

S'ha eliminat el atribut codi que tenia en el model relacional, ja que en aquest cas, fem que hereti els de l'entitat Directiu, que és hereva de Treballador (i per tant de Persona).

Utilitzem IdTreballador de Treballador com a clau primària de la taula.

La taula Alt_Directiu està relacionada directament amb Directiu, per l'herència, i amb Secretària relacionalment, ja que té un atribut (secretària) per permetrà establir l'esmentada relació.

```
CREATE TABLE ot_ALT_DIRECTIU
( secretària   INT,
 PRIMARY KEY(IdTreballador),
 FOREIGN KEY(secretària) REFERENCES ot_TREBALLADOR(IdTreballador)
)
INHERITS(ot_DIRECTIU);
```

Malgrat pogués semblar en un principi que Responsable hagués d'heretar de Cap_Oficina, es pot comprovar que no existeix cap especialització ni altra motiu per a fer l'herència.

S'ha conservat el model pensat pel model relacional, ja que en aquest cas, i donades les possibilitats del PostgreSQL, és vàlid.

La taula es relaciona amb Oficina mitjançant la seva clau primària (codi_oficina), i amb Secretària, fent referència a la clau heretada d'aquesta (de Treballador).

```
CREATE TABLE ot_RESPONSABLE
( oficina       od_CODI_OFICINA,
  d_inici       DATE,
  Cap_Oficina   INT,
  d_fi          DATE,
 PRIMARY KEY (oficina, d_inici),
 FOREIGN KEY(oficina)      REFERENCES ot_OFICINA,
 FOREIGN KEY(Cap_Oficina) REFERENCES ot_CAP_OFICINA,
 UNIQUE(d_inici, Cap_Oficina)
);
```

5.3.1 Detalls de funcionament del model Object/Relational

Un cop definides totes les entitats, totes les seqüències, i els dos dominis, s'ha procedit a inserir dades en el sistema.

Les dades de prova, bàsiques, han estat;

```
INSERT INTO ot_persona (nom, cognoms, nif, adreca,cod _postal,
  poblacio, nss) values ( 'Nom Per 1', 'Cognoms Per 1', 11, 'Adreca
  Per 1','0011','Pobla Per 1','11');
```

```
INSERT INTO ot_oficina values ('0011', 'direc oficina 0011', '{"tel
  011"}', '{"fax 011"}');
```

```
INSERT INTO ot_secretària (nom, cognoms, nif, adreca, cod_postal,
```

```
  poblacio, nss, cc, codi_oficina, idioma)
  values ('Nom Secre 1', 'Cognoms Secre 1', 11, 'Adreca Secre
  1','0022','Pobla Secre 1','nS',1,'0011','{"castella","'catala"}');
```

```
INSERT INTO ot_directiu (idpersona, nom, cognoms, nif, adreca,
  cod_postal, poblacio, nss, d_naixament, d_inici, cc, d_alta)
  values (11, 'Nom Direc 1', 'Cognom Direc 1', 11, 'Adreca Direc
  1','0022','Poblac Direc 1', 1, '03/03/1968', '03/03/2003', 'nD',
  '03/003/2001');
```

```
INSERT INTO ot_cap_oficina (nom, cognoms, nif, adreca, cod_postal,
  poblacio, nss, d_naixament, d_inici, cc, d_alta, plus)
  values ('Nom JfOf 1', 'Cognom JfOf 1', 11, 'Adreca JfOf 1',
  '0022', 'Pobla JfOf 1', 1, '03/03/1968', '03/03/2003', 'n3',
  '03/003/2001', 99);
```

```
INSERT INTO ot_responsable (oficina, f_inicio, cap_oficina, d_fi)
  values ('0011','03/01/2000',15,null);
```

A partir d'aquestes dades, s'han fet un seguit de consultes, comprovant la funcionalitat del model.

En fer una consulta amb SQL relacional 'tradicional' de persona, observem el resultat:

```
SELECT * FROM ot_persona;

idper|  nom      | cognoms      |nif|  adreca      |cpost|  poblacio     |nss|fnaixament
-----+-----------+--------------+---+--------------+-----+--------------+---+-----------
  10 |Nom Per 1  |Cognom Per 1  | 11|Adreca Per 1  | 0011 |Pobla Per 1   |11 |
  11 |Nom Secre 1|Cognom Secre 1| 11|Adreca Secre 1| 0022 |Pobla Secre 1 |nS |
  11 |Nom Direc 1|Cognom Direc 1| 11|Adreca Direc 1| 0022 |Poblac Direc 1|1  |1968-03-03
  12 |Nom JfOf 1 |Cognom JfOf 1 | 11|Adreca JfOf 1 | 0022 |Pobla JfOf 1  |1  |1968-03-03
```

Observem que ha retornat 4 files.

Comprovant les dades de prova, s'observa que la fila que té per nom idper, i que té valor 10, pertany a les dades introduïdes en la taula persona. La fila amb idper amb valor 11 pertany a les dades introduïdes en la taula Secretària, la següent fila pertany a les dades introduïdes en la taula Directiu, i l'última, a les dades introduïdes a la taula Cap_Oficina.

Observem amb aquesta primera consulta, que en inserir dades en les taules que hereten de Persona, s'afegeixen també a la taula d'on hereta, bé directament o bé indirectament, tal com s'observa per les dades introduïdes a Cap_Oficina, ja que aquesta no hereta directament de Persona.

Podem distingir les dades que hi han a la taula Persona, segons la seva procedència?

```
SELECT * FROM ONLY ot_persona;

idpersona| nom     |  cognoms    |nif| adreca     |cod_postal| poblacio  |nss|d_naixament
---------+---------+-------------+---+------------+----------+-----------+---+-----------
  10     |Nom Per 1|Cognoms Per 1| 11|Adreca Per 1|   0011   |Pobla Per 1| 11|
```

Afegint la clàusula **ONLY** a la consulta, coincidint amb la definició que es fa al SQL1999, la base de dades retorna únicament les files que s'han introduït a la taula persona.

Si consultem la taula Treballador, podem observar que;

```
SELECT * FROM ot_treballador;

 idpersona |    nom         |     cognoms      | nif   |    adreca         | cod_postal |
poblacio| nss | d_naixament | idtreballador |  d_inici  | cc
-----------+-------------+-----------------+-----+----------------+------------+---------
--------+-----+-------------+---------------+-----------+----

       11 | Nom Secre 1 | Cognoms Secre 1 |  11 | Adreca Secre 1 | 0022        | Pobla
Secre 1  | nS  |              |         12 |            | 1

       11 | Nom Direc 1 | Cognom Direc 1   |  11 | Adreca Direc 1  | 0022       | Poblac
Direc 1 | 1   | 1968-03-03   |         14 | 2003-03-03 | nD

       12 | Nom JfOf 1  | Cognom JfOf 1    |  11 | Adreca JfOf 1   | 0022        | Pobla
JfOf 1   | 1   | 1968-03-03   |         15 | 2003-03-03 | n3
```

Únicament apareixen les dades que s'han introduït en les taules que hereten d'aquesta. La fila amb IdPersona amb valor 10 no apareix, ja que s'ha introduït directament en la taula Persona.

Si tornem a fer la consulta amb la clàusula ONLY, ens trobem que;

```
SELECT * FROM ONLY ot_treballador;

 idpersona | nom     | cognoms    | nif | adreca     | cod_postal | poblacio | nss   |
d_naixament | idtreballador | d_inici | cc
-----------+--------+-----------+-----+-----------+------------+-----------+-----+-------
-----------+---------------+---------+----
(0 rows)
```

No retorna cap valor, ja que no s'ha introduït cap valor directament a la taula Treballador. Totes les dades que havia tornat la primer vegada, eren les introduïdes des de les instàncies que hereten d'aquesta.

Un cop feta la comprovació anterior, s'ha de fer notar que al igual que en aquest cas en que no obtenim cap resultat de la consulta, no encertarem cap registre en el cas de fer un UPTDATE, pel que és impossible afegir els atributs necessaris a Persona per a convertir-lo en un Treballador.

En introduir dades en la taula Directiu, ha sorgit un problema que no s'ha localitzat en el mateix instant. Comprovem les dades que hi tenim;

```
SELECT * FROM ot_directiu;

 idpersona |    nom         |    cognoms        | nif |    adreca         | cod_postal |
poblacio    | nss | d_naixament | idtreballador |
 d_inici  | cc |   d_alta
-----------+-------------+----------------+-----+---------------+------------+----------
------------+-----+-------------+---------------+----------------------------------------
----------+----+------------

        11 | Nom Direc 1 | Cognom Direc 1 |  11 | Adreca Direc 1 | 0022        | Poblac
Direc 1 | 1   | 1968-03-03   |         14 |
2003-03-03 | nD | 2001-03-03

        12 | Nom JfOf 1  | Cognom JfOf 1  |  11 | Adreca JfOf 1  | 0022        | Pobla
JfOf 1   | 1   | 1968-03-03   |         15 |
2003-03-03 | n3 | 2001-03-03
```

Si ara tornem a comprovar el resultat de la consulta Treballador, observarem que tenim dos persones amb IdPersona amb valor 11.

Si tenim en compta que IdPersona és una clau primària, observarem que **hem creat una inconsistència de dades en la clau primària** de la taula Treballador.
En el moment d'introduir les dades a Directiu, s'ha afegit el valor a IdTreballador manualment, evitant que el NUL permetés que la seqüència assignes el valor corresponent. Aquest fet, posa de manifest que al inserir dades en una taula que hereta atributs d'una altra, no es fan les comprovacions de si les dades inserides violen les regles definides en la taula arrel.

La consulta a la taula Secretària, ens pot permetre veure com es tracten les col·leccions tipus;

```
SELECT * FROM ot_secretària;

 idpersona |    nom      |    cognoms      | nif |    adreca        |cod_postal|    poblacio
|
nss | d_naixament | idtreballador |d_inici | cc | codi_oficina |      idioma
-----------+-------------+-----------------+-----+---------------+------------+---------
--------+-----+-------------+---------------+-----------------------------------------
--------+----+--------------+----------------

        13 |Nom Secre 1|Cognoms Secre 1|  11 | Adreca Secre 1 |0022       | Pobla Secre 1
|
 nS  |            |         16 |           | 1  | 0011         | {castella,catala}
```

S'observa que en fer una consulta d'una entitat que té col·leccions tipus, en el resultat consten totes els valors inserits en l'atribut que té aquesta característica.

És important observar que s'obvia un dels primers paradigmes de les bases de dades relacionals, un atribut pot tenir més d'un valor.

En el cas de voler consultar únicament un dels valors del vector que té l'atribut idioma, es pot fer una consulta del tipus;

```
SELECT nom, idioma[1] FROM ot_secretària WHERE codi_oficina = '0011';

   nom        |  idioma
--------------+----------
 Nom Secre 1 | castella
```

En aquest cas, s'observa que només es consulta el primer idioma del vector. Notar que el vector comença en la posició u (només tenim dos idiomes, i el segon és el català).

En el cas de tenir que modificar els valors que estan inserits en aquest vector, únicament cal fer referència a aquests, conjuntament amb la posició que ocupen, per modificar-los normalment.

6. La base de dades PostgreSQL

La primera versió del Sistema de Gestió de Base de Dades Objecte-Relacional (ORDBMS) PostgreSQL *va sortir al mercat* a l'any 1977. En aquell moment, el sistema ni era orientat a Objectes, ni era gairebé una Base de dades Relacional, ni tenia per nom PostgreSQL.

La primera versió, de nom Ingres, va ser desenvolupada a la Universitat de Califòrnia, Berkeley, com a demostració acadèmica del que haurien de ser les bases de dades futures, per passar posteriorment a l'entorn empresarial (de la ma de l'empresa Relational Technologies/Ingres Corporation).

El projecte acadèmic va ser continuat amb força a l'any 1986 per un equip de persones dirigides per en Michael Stonebraker. El repte era millorar el producte, per transformar-lo en una base de dades objecte-relacional.

La versió resultant d'aquest esforç es va anomenar Postgres95, i va ser un exemple a referenciar, en parlar de bases de dades avançades.

La següent evolució del producte, va afegir les sigles SQL al nom original, passant-se a denominar PostgreSQL. La diferencia evolutiva entre aquesta nova versió i l'anterior, va ser molt major que en d'altres ocasions, degut principalment al fet de que es va alliberar el codi font, passant aquest a ser de domini públic (*open source*).

La conseqüència de l'alliberament del codi font de la base de dades, va ser l'esperada. Arreu del mon es va crear una comunitat de desenvolupadors, que a partir d'aquell moment col·laboraren activament en desenvolupar encara més el sistema.

Encara que PostgreSQL es considerada la base de dades més avançada del món Open Source (codi font de lliure distribució), no existeixen gran nombre de base de dades amb la que es pugui comparar en 'aquesta catalogació'. No obstant, s'ha de considerar que acompleix amb les regles d'integritat ACID, que disposa de disparadors (*triggers*) i de procediments emmagatzemats (*stored procedures*), de transaccions, de *drivers* JDBC i d'altres possibilitats que fan que sigui una RDBMS complerta, amb la que es pot desenvolupar qualsevol aplicatiu, per complex i gran que sigui.

Malgrat ser PostgreSQL *open source*, existeixen versions de pagament. Aquesta contradicció és possible, ja que exactament no es ven la base de dades. Les versions comercials, inclou lleugeres modificacions amb característiques addicionals (instal·ladors gràfics, per exemple), documentació impresa i suport tècnic per a un període de temps determinat.

PostgreSQL es pot instal·lar pràcticament en qualsevol sistema Unix/Linux. També es pot instal·lar amb l'ajut del programa CygWin sobre Windows, encara que en aquest cas el rendiment es baix, i només s'aconsella fer-ho per realitzar proves i/o formació.

Pràcticament totes les distribucions actuals de Linux incorporen la base de dades PostgreSQL com a paquet instal·lable, pel que durant la instal·lació de qualsevol de les versions d'aquest Sistema Operatiu, només caldria seleccionar dels paquets a instal·lar (de la categoria de Bases de Dades) aquesta opció, per passar a tenir un potent (i econòmic) servidor de bases de dades, perfectament comparable amb d'altres solucions comercials d'alt, altíssim, cost.

Encara que estigui fora dels objectius d'aquest document, cal comentar que una altra solució que ens permetria disposar d'un servidor de bases de dades de baix cost, estaria el MySQL.

Malgrat no ho sembli, el MySQL no és open source, i encara que es permeti el seu ús en entorns no comercials, no és una solució gratuïta. Si a això afegim que no és una base de dades ACID, i que no té algunes característiques que s'haurien de considerar com a imprescindibles (triggers, stored procedures, etc.), fa que el seu ús en desenvolupaments de mida mitjana o gran, no sigui aconsellable.

6.1 Característiques no documentades: Herència Múltiple

Malgrat no es defineixi l'herència múltiple en el SQL:1999, es pot comprovar experimentalment que PostgreSQL si la suporta, encara que l'esmentada característica no estigui documentada.

Malgrat que de moment no es cregui convenient utilitzar l'herència múltiple en aplicacions realitzades amb aquesta SGBD (donat que una futura versió no té per què implementar-ho - ja que no està documentat), es creu prou interessant per posar un exemple bàsic del seu funcionament.

Crearem una jerarquia de taules per poder emmagatzemar tipus d'animals:

```
CREATE TABLE animal (id INTEGER, nom VARCHAR(15));
```

Els mamífers i els ovípars s'emmagatzemaran en taules diferents. El *nom*, donat que és un atribut bàsic dels dos, està en la taula d'on hereten.

```
CREATE TABLE mamifer (num_potes INTEGER) INHERITS (animal);
```

```
CREATE TABLE ovipar (posta_max INTEGER) INHERITS (animal);
```

Hi ha animals rars (que són mamífers i ovípars alhora), que caldrà emmagatzemar en una taula que hereti de les dos anteriors.

```
CREATE TABLE altres (descrip VARCHAR(50)) INHERITS (mamifer, ovipar);
```

En crear aquesta taula, PostgreSQL ens adverteix que s'està creant una herència múltiple, resolent transparentment la situació.

```
NOTICE:    CREATE  TABLE:  merging  multiple  inherited  definitions  of
attribute "id"
NOTICE:    CREATE  TABLE:  merging  multiple  inherited  definitions  of
attribute "nom"
```

Fem la inserció d'un cas especial, inserim l'animal Ornitorrinco.

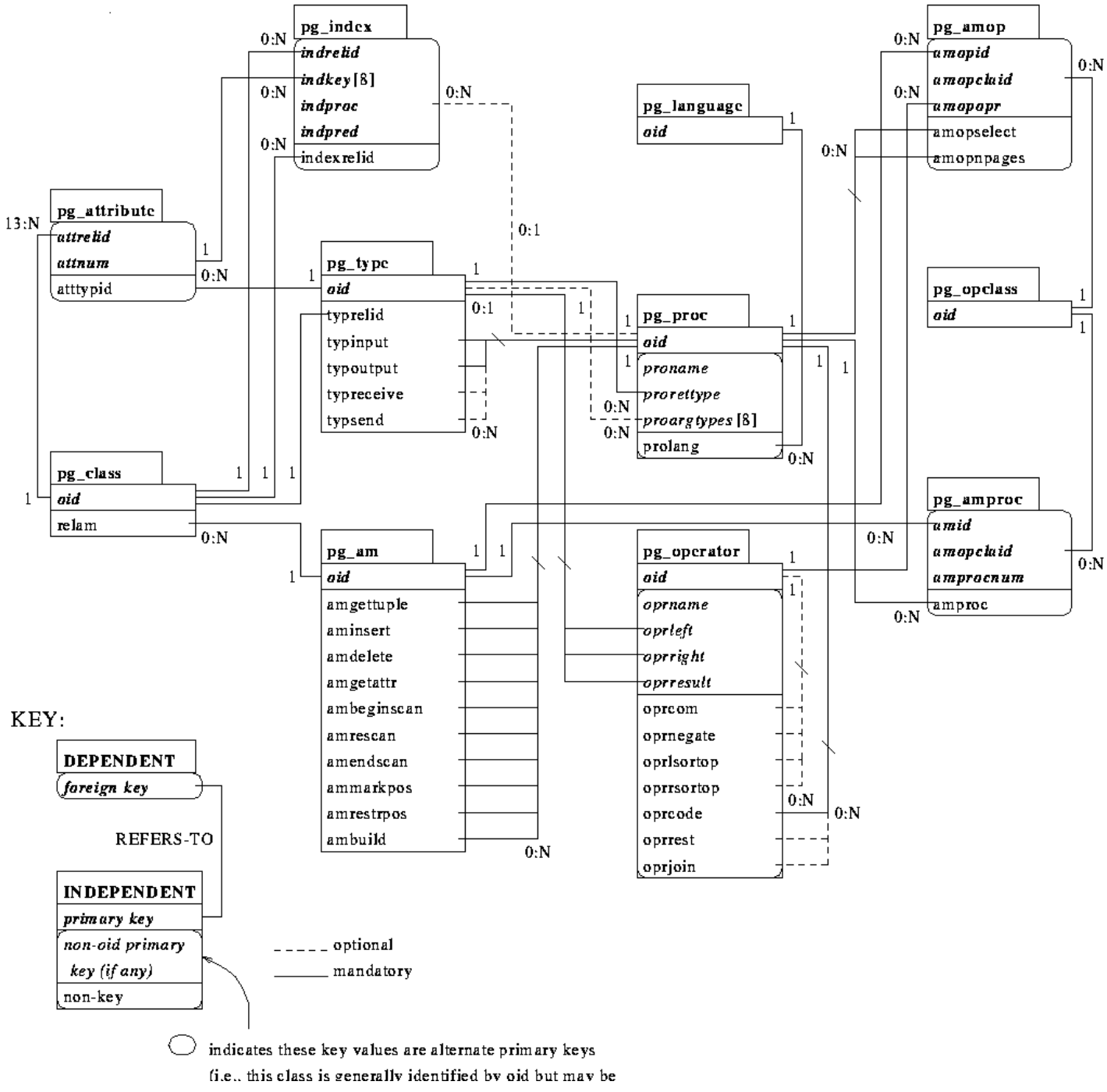
```
INSERT INTO altres VALUES (99, 'ornitorrinco', 4, 1, 'animal raro');
```

En fer la consulta de les dades inserides, comprovem que funciona correctament.

```
SELECT * FROM altres;
 id |     nom      | num_potes | posta_max |   descrip
----+--------------+-----------+-----------+-------------
 99 | ornitorrinco |         4 |         1 | animal raro
(1 row)
```

6.2 Implementació interna de les taules, els catàlegs

PostgreSQL té una important avantatge sobre altres bases de dades: Gran part del seu funcionament està parametritzat pels valors continguts en les seves pròpies taules. A aquestes taules d'ús intern, se les anomena catàlegs i estan molt ben documentades.



7. Diferències entre PostgreSQL i Oracle 9i

En aquest punt relacionarem les característiques que s'han implementat de l'SQL:1999 en les bases de dades *object-relational* Oracle 9i i en PostgreSQL.

S'han constatat notables diferències en les esmentades implementacions. Aquestes són degudes a diferents motius; per un costat tenim que Oracle 9i ha de mantenir una perfecta compatibilitat amb les versions anteriors i ha d'assegurar un entorn d'explotació molt robust, donat el mercat al que va destinat. Per l'altra banda PostgreSQL és desenvolupat per una comunitat informàtica que té precisament com a objectiu, el desenvolupament d'aquest i d'altres productes. Al ser un producte de codi obert està en constant evolució, donat que en part del món acadèmic en que es desenvolupa, es pren com a model del que haurien de ser les bases de dades *object-relational*, encara que per a que ho sigui completament fa falta que evolucioni molt més.

COMPARATIVA	Oracle 9i	PostgreSQL
Tipus de producte	Producte comercial.	Desenvolupat en entorns acadèmics.
Versió provada	Versió 9i	Versió 7.3 i 7.4
Cost	Preu d'adquisició alt. S'ha de pagar per cada client que utilitzi el motor de la base de dades. Cada nova versió comporta una nova adquisició.	Distribució gratuïta. El tipus de llicència d'ús, fa que les noves versions també hagin de ser de lliure distribució.
Suport tècnic (de pagament)	El fabricant ofereix molt bon servei tècnic.	Existeixen empreses especialitzades que ofereixen suport tècnic.
Altres tipus de suport	Al tenir Oracle un bon suport en la seva web (metalink) es troben a faltar webs no dependents del fabricant.	Suport en fòrums de discussió mantinguts per la pròpia comunitat de desenvolupadors.
Qualitat i quantitat de la documentació	Oracle és un producte molt extens, pel que existeix molta documentació. Això fa que costi trobar la que s'està cercant.	Els propis desenvolupadors creen la documentació oficial que pengen en la web del projecte. Els propis usuaris desenvolupen casos pràctics i documentació addicional.
Robustesa	És una BD que compleix les propietats ACID per les transaccions.	És una BD que compleix les propietats ACID per les transaccions.
Altres característiques	Té disparadors, procediments	Te disparadors, procediments

	emmagatzemats, restriccions, replicació i suporta (amb diferent sintaxi), totes les possibilitats definides en l'ANSI de l'SQL-92.	emmagatzemats, restriccions, replicació i quasi bé totes les possibilitats definides en l'ANSI de l'SQL-92.
Escalabilitat	Suporta BD distribuïdes.	No suporta BD distribuïdes però funciona amb sistemes d'alt rendiment i multiprocessadors .
Complexitat d'instal·lació i del manteniment del motor de la BD	No té gaire dificultat i és pot fer amb una formació mínima (provat en entorn Windows)	Seguint les instruccions que faciliten els desenvolupadors del producte, la instal·lació sobre un SO Linux no hauria de tenir cap dificultat. El manteniment posterior, s'hauria de limitar a realitzar les còpies de seguretat, donat que a partir de la versió del PostgreSQL 7.4 el VACUUM s'executa periòdicament de forma automàtica.
Sistemes operatius sobre els que funciona	Gairebé en el principals sistemes operatius del mercat (Windows, Unix, Linux, Aix, etc.)	Unix, Linux, Aix, (entre d'altres) i possibilitat d'executar en l'entorn Windows (amb certes restriccions).

7.1 Avaluació

Al llarg de l'avaluació entre tots dos SGBD, constatarem que fins i tot el concepte d'objecte és completament diferent i està portat a terme amb criteris desiguals. S'observarà que els objectes de l'Oracle 9i segueixen el paradigma tradicional de l'herència, tal com es fa en C++, Java o Eiffel (per exemple). Existeixen constructors i es poden especificar taules que hereten d'unes altres. En PostgreSQL el concepte de disseny basat en objectes el deixen en part, en l'àmbit teòric, fins a considerar que una fila d'una taula és una instància, simple, sense possibilitats de que l'atribut pugui ser un tipus definit per l'usuari. Malgrat aquesta última afirmació s'ha de dir que un atribut si que pot ser un tipus definit per l'usuari però en aquest cas s'ha de fer tota la codificació (en C++ habitualment) de les rutines que suporten el mateix. Donada la feina de crear un nou tipus, es considera que és responsabilitat de l'equip de manteniment de PostgreSQL la seva creació, més que no pas de l'usuari final.

En el moment de comparar el model relacional amb l'*object-relational* s'han constatat les diferències entre els dos models, comprovant que el model *object-relational* incorpora moltes noves característiques adreçades a tenir una major reusabilitat de les dades, i a incrementar la distància entre la representació

física i la representació lògica d'aquestes, creant abstraccions més properes a models reals. Per tant, el model *object-relational*, des d'un punt de vista semàntic, és més ric que el model relacional.

S'observa que les abstraccions encaminades a emmagatzemar les dades d'una forma més similar al món real, es porta a terme de diferents maneres depenent del SGBD utilitzat, aconseguint amb l'Oracle 9i un major apropament a aquesta abstracció.

7.2 Comparació de les característiques del SQL:1999 en els dos productes

7.2.1 UDT

Tal com s'ha comentat anteriorment l'SQL:1999 defineix els UDT com una característica per poder encapsular uns certs tipus de dades.

Hem observat que en Oracle 9i aquesta característica ens dóna moltes possibilitats, des d'adaptar a les nostres necessitats un tipus existent, a confeccionar un tipus amb diferents atributs relacionats.

En el primer cas, tindríem un bon exemple si definíssim dos tipus, per diferenciar els metres lineals dels metres quadrats, donat que voldríem evitar les operacions entre ells.

En el segon cas, tindríem un nou tipus definit, per tenir totes les propietats d'una adreça (per exemple) englobades en un sol tipus.

A diferència de l'Oracle, PostgreSQL no té la capacitat de crear UDTs. Ni tant sols suporta la creació d'UDTs per a poder distingir dos atributs que comparteixen el mateix tipus de dades, però que conceptualment descriuen conceptes diferents de la realitat. Caldria poder distingir dos atributs de tipus enter que representin coses diferents (longituds i pes, per exemple).

En el PostgreSQL també s'ha trobat a faltar el poder definir columnes de tipus complex. La facilitat que proporciona el disposar de UDTs és gran, ja que per exemple, en el món real en parlar d'una adreça, no es pensa en un tipus de carrer (via, avinguda, carretera), en el nom del carrer, en el codi postal i en la població i província on està situat. En fer menció a un carrer, s'està pensant en una ubicació que porta incorporats tots els atributs esmentats, i que abstractament ens descriu un lloc.

En portar a terme l'implementació del cas pràctic en PostgreSQL, s'ha tingut que obviar la utilització dels UDTs per descriure tipus complexes de dades, quedant l'esmentada implementació molt similar a l'equivalent en model relacional.

7.2.2 Col·leccions

Aquests tipus estan suportats pels dos productes i en observar el funcionament de les col·leccions en l'Oracle 9i, podem comprovar que segueix bastant fidedignament les recomanacions fetes en l'SQl:1999. En el cas pràctic

s'han utilitzat en dues ocasions els arrays i ha sigut d'utilitat per poder millorar el model de dades sense haver de fer que aquest guanyés en complexitat, ja que s'ha evitat la utilització de taules addicionals. Per altra banda podem tenir un atribut multiavaluat, encara que no compleixi una de les regles fonamentals del model relacional, en que es diu que cada atribut ha de ser atòmic.

S'ha d'observar que durant la implementació en PostgreSQL s'ha descobert un error de funcionament en la base de dades. PostgreSQL sempre crea vectors de longitud il·limitada, encara que s'indiqui una longitud màxima.

La creació de les col·leccions segueixen les recomanacions del estàndard SQL:1999, i no s'han observat divergències significatives, essent en la implementació que s'ha fet del cas desenvolupat, molt útils, ja que ha permès simplificar les relacions del cas descrit.

7.2.3 Tipus referència

En el PostgreSQL també trobem a faltar el tipus referència. Aquest tipus s'utilitzaria en els casos en que s'haguessin d'identificar les columnes de alguna taula tipada. Un valor de tipus referència pot ser emmagatzemat en una taula i utilitzat en una fila especifica d'una altra taula, com si fos un punter lògic.

7.2.4 Taules tipades

Tal com ja s'ha observat, PostgreSQL no suporta tipus complexos de dades. A resultes d'aquesta limitació, no suporta les taules tipades.

En Oracle 9i sí que és possible implementar aquest tipus de taules donat que aquestes serveixen per instanciar tipus abstractes de dades.

7.2.5 Herència en les característiques prèvies

Tant el Oracle 9i com el PostgreSQL coincideixen en el fet de tenir herència de taules. Addicionalment Oracle 9i suporta herència de tipus. Aquest fet és important, ja que els desenvolupaments actuals donen importància a aquest fet, fent un ús intens d'aquesta característica.

El funcionament en PostgreSQL divergeix notablement, ja que funciona en forma d'arbre jeràrquic de taules. A les taules que hereten se'ls hi afegeix els atributs de les taules pare. Aquesta senzillesa de funcionament divergeix del descrit en el estàndard SQL:1999, però simplifica enormement el disseny. Per altra costat, un disseny senzill, no vol dir que sigui òptim, ja que es pot comprovar en arribar a intentar implementar alguns casos més complexos, que ens segueixen faltant els UDT.

En aquest punt, s'ha de recordar que PostgreSQL té una errada en la implementació de l'herència de les taules, que fa que es puguin inserir dades duplicades en una columna UNIQUE o en una clau primària d'una taula pare, des de una taula filla.

En l'Oracle 9i tenim la possibilitat de limitar si una taula o un tipus intermedi en la jerarquia de l'herència és instanciable o és una classe final ja que existeixen les clàusules FINAL i NOT FINAL, i les de INSTANCIABLE i

NOT INSTANCIABLE. Aquesta diferenciació no es pot portar a terme en PostgreSQL. En aquest es pot fer en qualsevol moment que una taula sigui pare d'una nova taula, que seria filla. Aquest fet també comporta que es pugui inserir dades en qualsevol taula de la jerarquia de la herència, dificultant el manteniment de la coherència de dades, ja que en crear unes noves taules filles, depenent d'una principal, s'haurà de tenir especial cura de que es puguin o no inserir dades en la primera.

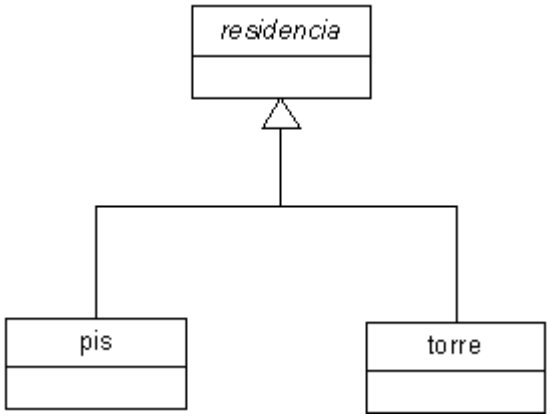
7.3 Taula comparativa de les característiques dels dos productes

Característica SQL:1999	Oracle 9i	PostgreSQL
UDT	SI	NO
Tipus Col·leccions	SI	SI
Tipus referència	SI	NO
Taules tipades	SI	NO
Herència	SI: de tipus i de taules	SI: sols de taules
Herència Múltiple	NO	SI (experimentalment)

7.4 Exemple pràctic d'utilització de tipus i herència

Per exemplificar les possibilitats i mancances d'ambdues bases de dades, s'insereix tot seguit el codi necessari per a crear una estructura de dades que serveixi per emmagatzemar informació de la residència de les persones.

L'exemple s'ha simplificat fins deixar només els atributs mínims que serveixen per comprovar el funcionament dels tipus i de l'herència.



Es pot observar en que Oracle s'ha aconseguit utilitzar ambdues possibilitats, mentre que en PostgreSQL només s'ha pogut utilitzar l'herència entre les taules.

Oracle 9i	PostgreSQL

Creo un tipus domicili:
 	
`CREATE TYPE DomiciliType AS OBJECT`
`  (`
`  adreça  VARCHAR (30),`
`  Codi_postal  VARCHAR (5),`
`  Ciutat  VARCHAR (15),`
`  Provincia  VARCHAR (15),`
`  Pais  VARCHAR (15)`
`  );`
 	
Creo un objecte abstracte del que heretaran els altres dos:
	Creem 4 taules:
	
	La primera és la taula que contindrà les dades d'on s'ubica:
	
	`CREATE TABLE adreca`
	`  (`
	`  adreca  VARCHAR(30),`
	`  codi_postal  VARCHAR(5),`
	`  ciutat  VARCHAR(15),`
	`  provincia  VARCHAR(15),`
	`  pais  VARCHAR(15)`
	`  );`

```
CREATE TYPE residencia AS OBJECT
  (
  codi              NUMBER,
  domicili          DomiciliType,
  num_habitacions   NUMBER,
  num_banys         NUMBER,
  m_quadrats        NUMBER
  )
NOT FINAL NOT INSTANTIABLE;
```

Creo dos objectes que hereten del primer:

```
CREATE TYPE pis UNDER residencia
  (
  despeses_comunitat NUMBER
  )
FINAL INSTANTIABLE;
```

```
CREATE TYPE torre UNDER residencia
  (
  num_plantes NUMBER,
  metres_terreny NUMBER
  )
FINAL INSTANTIABLE;
```

Creo dues taules amb aquests nous tipus:

```
CREATE TABLE t_pis OF pis
  (
  PRIMARY KEY(codi)
  );
```

```
CREATE TABLE t_torre OF torre
  (
  PRIMARY KEY(codi)
  );
```

Després la taula contenidora de les propietats de qualsevol llar;

```
CREATE TABLE residencia
  (
  id              INTEGER,
  num_habitacions INTEGER,
  num_banys       INTEGER,
  m_quadrats      INTEGER,
  PRIMARY KEY(id)
  );
```

La taula pis, té atributs d'especialització, i incorporà els de les taules inicials;

```
CREATE TABLE pis
  (
  despeses_comunitat INTEGER)
  )
  INHERITS(residencia, adreca);
```

La taula torre, té atributs d'especialització, i incorporà els de les taules inicials;

```
CREATE TABLE torre
  (
  num_plantes    INTEGER,
  metres_terreny INTEGER,
  )
  INHERITS(residencia, adreca);
```

8. El futur SQL:200n, el nou estàndard

El nom que tindrà el nou estàndard ja es va decidir en el moment de publicar les especificacions del SQL:1999, serà el de: SQL:200n. El nom triat, amb el prefix de la primera desena d'anys del nou mil·leni, va ésser elegit pel significat que té, ja que indica que no passaran masses anys fins la nova publicació. Es va considerar que era positiu donar a entendre que les millores s'havien de periodificar sobre un calendari de cicle més curt, per aconseguir una major evolució d'aquests productes, tot intentant tornar al cicle habitual de publicar les millores cada tres o quatre anys, evitant així períodes llargs com el donat entre la publicació del SQL-92 i el SQL:1999.

Part de l'interès en fer que les especificacions es publiquen més sovint, és degut a que es vol evitar els esforços de desenvolupament sobtats, a que es veuen obligats els equips de disseny de les SGBD en sortir aquestes.

També s'ha de considerar que un nombre elevat de canvis simultanis, ajuda a crear majors controvèrsies que no pas canvis graduals fets al llarg del temps, i que la retro-realimentació de les noves característiques vers les futures especificacions, també són importants, ja que en augmentar el ritme de publicació de les propostes, es té possibilitat de comprovar com afecten aquestes a l'estàndard, tot fent que les complementarietats sorgeixin naturalment.

La futura definició del SQL:200n tindrà tres parts: la primera serà una correcció d'errates, la segona part descriurà les millores que s'haurien de fer sobre el model relacional, i la tercera descriuria les millores que s'haurien d'incorporar al model orientat a objectes.

Part dels participants en l'estandardització del SQL:1999, van coincidir en dir que els ARRAY ajudaven a solucionar part dels problemes del mode relacional, ja que aportava noves possibilitats. Malgrat aquest nou tipus, s'està demanant d'incloure en el SQL:200n, models de dades propers al SET, al MULTISET (bossa) i fins i tot a les LLISTES.

Amb motiu de paral·leritzar l'evolució del SQL amb la de la resta de tecnologies informàtiques, també es creu necessari facilitar l'emmagatzemament de les estructures de dades XML en les bases de dades ORDBMS, calent per aquesta fita, flexibilitzar les possibilitats de les col·leccions de tipus.

Altres característiques que sembla que siguin importants per a parts dels ens que realitzen les descripcions de les estandarditzacions, són relatives a poder crear disparadors en la utilització de vistes i el aconseguir que un procediment emmagatzemat pugui retornar múltiples valors com a resultat de la seva execució.

Entre les millores que s'han de proposar en la part d'orientació a l'objecte, està la de suportar herència múltiple. Durant la versió SQL:1999 es va evitar descriure-ho com a especificació, donada la complexitat necessària per

implementar-ho, evitant així possibles allargaments de terminis per a tenir les primeres SGBD que acomplissin aquest estàndard en el mercat.

L'herència múltiple es complicada d'implementar quan cal assegurar la integritat de les dades, més si hi ha la possibilitat de que aquesta herència funcioni en un entorn distribuït. Sembla que el mode de funcionament que té l'herència en el Java s'imposarà finalment com a model en el SQL:200n evitant els esmentats problemes, però possibilitant l'herència sense tots els inconvenients que es preveien.

9. Conclusions

Tal com s'ha demostrat en el món de la programació, l'orientació a l'objecte és necessària pel fet de que permet la reutilització, i pel fet de què fa augmentar la fiabilitat de les aplicacions.

En el món de les bases de dades, l'orientació a l'objecte ha servit per manllevar alguns problemes que encara no estaven resolts en el paradigma relacional. Certs sectors es trobaven fortament limitats per les restriccions que imposaven les 'files i columnes' tradicionals, donat que el tipus de dades amb les que treballen són de tipus molt diversos, alhora que n'hi han molt poques amb les mateixes característiques, a diferencia de sectors més tradicionals (banca per exemple), on hi han pocs tipus de dades, però gran quantitat d'aquestes.

Altrament, la utilització de llenguatges de programació orientats a l'objecte, crea la necessitat de fer persistents les dades amb les que es treballen. Serà més senzill emmagatzemar les esmentades dades en un sistema proper al funcionament d'aquests llenguatges, que fer complexes transformacions i traduccions d'aquestes, per emmagatzemar-les en sistemes relacions.

En la comparativa entre Oracle i PostgreSQL, s'ha vist que ambdues són perfectament vàlides per a ser utilitzades en entorns de producció d'elevats requeriments i fiabilitat, encara que en la mateixa s'ha pogut constatar que Oracle està molt més propera a l'estàndard SQL:1999 que PostgreSQL, donat el seu ampli suport als tipus i a l'herència.

PostgreSQL encara no arriba a tenir algunes característiques avançades que es defineixen a l'estàndard SQL:1999, però en contrapartida té algunes característiques que la fan encara més desitjable (donat que solucionar les mancances actuals només és qüestió de temps): És una base de dades de codi font obert, o sigui, qualsevol programador pot modificar el seu codi font i tornar a compilar tot el sistema. Qualsevol persona pot utilitzar la base de dades lliurement, sense haver de pagar llicencies, ni preocupar-se de quantes connexions simultànies manté el servidor, o de fins a quantes CPUs es pot instal·lar en el servidor, per a no infringir les llicencies d'ús.

Finalment cal recordar que PostgreSQL té un petit defecte en la implementació de l'herència (que se soluciona fàcilment), i que experimentalment suporta herència múltiple. Aquesta última característica no documentada (encara no s'hauria d'utilitzar en desenvolupaments), sembla indicar que en les properes versions d'aquesta SGBD s'han d'esperar forces canvis i millores.

10. Futurs treballs a desenvolupar

L'oportunitat que s'ha tingut per comprovar l'evolució de les bases de dades i les seves noves característiques ha estat única, ja que arribar a copsar les noves possibilitats requereix d'un temps d'estudi i de pràctica, així com d'un cert procés d'adaptació. Les regles a seguir en dissenyar bases de dades canvien notablement.

Les noves possibilitats que aporta SQL:1999 a les bases de dades, només es comparable a la incorporació dels disparadors o dels procediments emmagatzemats, o fins i tot, a la introducció del SQL com a llenguatge de consulta en aquestes. El salt qualitatiu vers la versió anterior (SQL-92), és tant gran, que queda reflectit pels set anys de desenvolupament de la nova definició.

Durant el procés de cerca d'informació del funcionament del PostgreSQL, s'ha estat molt a prop de la comunitat de programadors que el desenvolupa. De fet, s'ha notat que existeix tot un grup de gent (molt propera als grups que desenvolupen el Linux i altres projectes de codi obert), que treballen en múltiples àrees per fer que el producte evolucioni. Hi ha gent que codifica, gent que aporta idees de disseny, gent que lliura tota una nova funcionalitat, gent que produeix la documentació, gent que la tradueix i fins i tot gent crítica que aporta el contrapunt del com fer les coses.

Durant el procés d'investigació de sí el PostgreSQL suportava definir una tupla per a utilitzar-ho com atribut, es va arribar a la conclusió (després de seguir varis fòrums de discussió i de revisar tota la documentació), de que no es podia, però per acabar d'assegurar-ho es va escriure a un dels esmentats fòrums. La contesta va ser ràpida i molt professional, confirmant les proves realitzades i el que s'havia contrastat en els manuals, però el més interessant del cas, és que es va provar el sistema de suport tècnic d'aquests tipus de productes de codi obert, tot arribant a la conclusió de que funciona.

En la resposta del fòrum de suport de PostgreSQL van afegir un petit paràgraf del estil; ...*está contemplado que lo soporte en un futuro, ¿te animas a colaborar en el desarrollo?*...

Una possible continuació d'aquest estudi sobre el SQL:1999 i la seva implementació en la base de dades PostgreSQL, podria passar per ampliar aquesta SGBD, per a que s'apropés més a la seva definició. L'estudi del funcionament dels arrays del PostgreSQL, amb la posterior ampliació i modificació per a permetre que cadascun dels elements enumerats poguessin ésser anomenats pel seu indicador, tot permeten que aquests fossin de tipus diferents, no hauria de ser un projecte excessivament complicat. Bàsicament s'hauria de fer que en aquests arrays s'inserissin els OIDs de les taules que definissin així aquests nous tipus, encara que tot això hauria de ser estudiat amb detall, tot demanant supervisió a programadors que ja haguessin desenvolupat parts d'aquesta base de dades.

11. Annexes

I. 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 i que no ha permès que caigués en el desànim. 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 quatre anys.

A la persona que em va animar a fer una enginyeria. A la persona a la que li dec el futur que m'espera.

II. Referències

http://www.postgresql.org
Web oficial del equip de desenvolupament de la SGBD PostgreSQL.

http://www.unne.edu.ar/cyt/2002/06-Biologicas/B-043.pdf
Notes d'interès de la implementació d'un projecte, amb explicacions del perquè de fer-ho orientat a l'objecte i en tres capes.

http://es.tldp.org/Postgresql-es/web/navegable/todopostgresql/postgres.htm
Manual del PostgreSQL en castellà (format HTML), penjat en la web del equip LUCAS (treballen per aconseguir publicar documentació, habitualment de software GNU, en castellà).

http://techdocs.postgresql.org
Enllaç amb gran quantitat de manuals i amb eines i utilitats addicionals.

http://www.commandprompt.com/ppbook/
Empresa ubicada a Oregon (USA) dedicada a desenvolupar professionalment amb PostgreSQL, i en donar suport tècnic sobre aquesta SGBD.

http://www.service-architecture.com/database/articles/sql1999.html
Definicions que calen del SQL:1999

http://www.cygwin.com
Utilitat per emular sobre un Windows un entorn Linux, sobre el que es pot instal·lar un PostgreSQL de proves.

http://www.service-architecture.com/object-relational-mapping/articles/
Bona font de notes tècniques sobre el funcionament intern de les BDs.

http://www.gnu.org/gnu/thegnuproject.es.html
Historia de que és, i de com es va inicial el software GNU.

III. Bibliografia

Articles

Atkinson, M.P. [et al] The Object-Oriented Database Manifesto. Deductive and Object-Oriented Databases, Proceedings of the Firt International Conference on Deductive and Object-Oriented Databases (DOOD'89), Kyoto Research Park, 1989. North-Holland/Elsevier Science Publishers 1990.

M. Stonebraker [et al.]. Third Generation Database System Manifesto. Comitee for Advanced DBMS Function, 1990. SIGMOD Record, volum 19, número 3.

Llibres

Jim Melton i Alan R. Simon. SQL: 1999. Understanding Relational Language Components. Ed. Morgan Kaufmann. 2001.

Jim Melton. Advanced SQL: 1999. Understanding Object-Relational and Other Advanced Features. Ed. Morgan Kaufmann. 2002.

John C. Worsley, Joshua D. Drake . Practical PostgreSQL. Ed. O'Reilly Unix. 2003.

François Bancilhon, Claude Delobel, Paris Kanellakis. Building an Object-Oriented Database System. Ed. Morgan Kaufmann Publishers, Inc. 2002.

IV. Glossari

ACID: Son les sigles de les propietats bàsiques que ha complir una transacció d'una base de dades. S'ha d'executar de forma Atòmica (o tot o res), de forma Consistent (si s'executa correctament, ha de quedar emmagatzemat), de forma Isolada (sense estar influenciada per altres transaccions en curs), i de forma Durable (s'ha de poder recuperar les dades encara que es produeixi una fallada del sistema).
Base de dades distribuïda: És una base de dades emmagatzemada en diversos servidors, pel que físicament està fraccionada en parts. Els servidors interactuen entre si, per transmetent les peticions i actualitzacions. Amb aquest mètode s'intenta fer que les dades estiguin en un lloc més proper al seu consum.
BD: Base de Dades.
BSD: És un tipus de llicencia d'us de programari molt similar a la llicencia d'ús GNU.
CAD: Computer-Aided Design, traduït com a Disseny Assistit per Ordinador.
CPU: El terme *Central Processing Unit* ve a definir el cor de l'ordinador. Aquest està format per varis milions de transistors (en les CPUs modernes), que s'encarreguen d'executar les instruccions llegides de la memòria.
DBMS: Data Base Management System, traduït com a Sistema de Bases de Dades.
Disparador: És un programa que s'emmagatzema en el servidor on resideix la base de dades i que s'executa en aquest mateix, en ocorrer un esdeveniment que s'ha definit.
GNU: És un tipus de llicencia d'ús de programari, pel qual es permet copiar, modificar, distribuir els programes, sense més obligació que la de mantenir el tipus de llicencia en les noves versions, o versions modificades del programa. De vegades s'utilitza l'expressió *de codi font obert*, per a indicar que es faciliten les fonts del programa, per a que es puguin portar a terme les esmentades modificacions/millores que calguin.
ORDBMS: Object Relational Database Management System, es a dir, Base de Dades Relacional Orientada a l'Objecte.
Procediments emmagatzemats: És un programa que s'executa en el servidor de la base de dades i que està emmagatzemat en aquest mateix. La principal avantatge dels procediments emmagatzemats és que es poden realitzar tasques complexes (cursors), sense necessitat de que les dades surtin del servidor (màxima velocitat d'execució, ja que els resultat intermitjos no han de viatjar fins al client).
ROW: És una fila o registre de la base de dades. Conté tants elements com els definits en les columnes. Cada columna té un nom, i s'ha definit el tipus de dades que pot contenir. Fins a la definició del SQL-92 cada element de cada columna era atòmic, permetent-se a partir del SQL:1999 multiplicitat de valors en cadascun.
SGBDOO: Sistema de Gestió de Bases de Dades Orientades a l'objecte.
SQL: *Structured Query Language*. Llenguatge de consulta estructurat. Defineix la sintaxis del llenguatge d'interrogació de les bases de dades relacionals.
Stored Procedure: Veure Procediments Emmagatzemats.
Transacció: Està compost per un conjunt d'instruccions, que s'han d'executar de forma sencera. En cas de que qualsevol de les accions no es pugui dur a

terme, la transacció no s'ha de portar a terme, havent de quedar el sistema en l'estat anterior al del començament de l'execució d'aquesta.
Trigger: Veure disparador.
XML: Extensible Markup Language. És una definició encaminada a definir l'estàndard de codificació d'estructures de dades que puguin entendre tant els ordinadors com les persones.