UNIVERSITAT OBERTA DE CATALUNYA

Enginyeria Tècnica Informàtica de Gestió

AVALUACIÓ DE SQL: 1999 RESPECTE LES CARACTERÍSTIQUES *ORIENTADES A OBJECTES* QUE SUPORTA I LA SEVA IMPLEMENTACIÓ EN SISTEMES DE GESTIÓ DE BASES DE DADES

Alumne: Manel González Luque
Dirigit per: Josep Joaquim Navarro Juaní
CURS: Febrer 2004 – Juny 2004

Resum

El treball realitzat analitza com l'estàndard SQL:1999 ha incorporat a la seva especificació una sèrie de característiques pròpies de la tecnologia d'orientació a l'objecte. D'aquesta forma, l'estàndard tracta de proporcionar als Sistemes de Gestió de Bases de Dades algunes de les funcionalitats exigides als sistemes de tercera generació. Aquests sistemes han d'incorporar altres característiques, es troben recollides a l'article "Third Generation Database System Manifesto".

Iniciem l'estudi amb l'evolució que han seguit els SGBDs, on han sorgit dos models clars, els Sistemes de Gestió de Bases de Dades Orientats a l'Objecte, que prenen com a base el model de dades d'orientació a l'objecte i que han de tenir al menys les característiques que detalla l'article "The Object-Oriented Database Manifest", i els Sistemes de Gestió Objecte-Relacionals. Aquest darrer pretén incorporar les característiques pròpies del sistemes de tercera generació mitjançant extensions del model relacional.

Es realitza un estudi de l'estàndard SQL: 1999 centrat en les característiques especificades al primer grup de proposicions de l'article "Third Generation Database System Manifesto", especialment els mecanismes de construcció de nous tipus, l'herència i les funcions.

L'estàndard SQL: 1999 ha estat implementat per molts fabricants a diversos productes, però les implementacions poden ser diferents. S'han estudiat tres productes comercials per tal de veure com han implementat l'estàndard únicament pel que fa a les característiques que són objecte del nostre treball.

Per estudiar la implementació concreta de l'estàndard en dos dels sistemes de bases de dades comercials estudiats, Oracle9i i PostgreSQL, s'ha desenvolupat un cas pràctic.

Índex

1 Introducció

1.1 Context del treball

Els sistemes de gestió de bases de dades (SGBD, en anglès DBMS, Database Management Systems) han anat evolucionant al llarg dels darrers 30 anys. Aquesta evolució ha estat lligada a las possibilitats tècniques de cada moment, així com a l'evolució de les necessitats de les aplicacions informàtiques.
Durant els anys 60 i inicis dels 70 els SGBD que imperaven eren els jeràrquics i els de xarxa. Aquests tipus de SGBD configuraven el que s'anomena la primera generació de sistemes de bases de dades. Es podien considerar així perquè eren els que primer van proporcionar una sèrie d'eines o utilitats per la manipulació, creació, gestió d'emmagatzematge, accés i protecció de les dades de manera unificada.

Aquest sistemes presentaven com a desavantatges principals la necessitat d'escriure programes complexes per fer, fins i tot, consultes simples, hi havia una mínima independència de les dades i no es disposaven de fonaments teòrics abastament acceptats.

A finals dels 70 i primers dels 80 van aparèixer els SGBD basats en el model de dades relacional primerament exposat per Codd a l'any 1970. Aquests SGBDs van formar el que s'anomena la segona generació de sistemes de bases de dades.

Els Sistemes de Gestió de Bases de Dades Relacionals (SGBDR), entre d'altres característiques, utilitzen un llenguatge de manipulació de dades no procedimental i proveeixen d'un grau substancial d'independència de les dades.

Malgrat que aquesta segona generació de sistemes va suposar un avenç respecte dels anteriors, i comercialment va experimentar un gran èxit, l'augment de la complexitat de les aplicacions han mostrat que presenten una sèrie de desavantatges. Efectivament, per a moltes aplicacions aquests sistemes són inadequats, especialment pel que fa a la seva limitada capacitat per al modelat de dades. Aplicacions del tipus CAD, CASE, GIS, etc.

Aquestes limitacions no sols s'han fet evidents en aplicacions complexes, sinó també en aplicacions de negocis tradicionals que necessiten recollir certes característiques dels objectes del mon real que no són adaptables als tipus de dades predefinits que proporcionen els sistemes de bases de dades relacionals.

Les carències més destacades són:

- El model de dades relacional és massa simple per modelar entitats complexes anidades, com ara els objectes d'aplicacions de disseny i enginyeria.

- Suport d'un conjunt limitat de tipus de dades atòmics (com ara sencers, cadenes, etc.). No suporten tipus de dades com els que es poden trobar en llenguatges de programació d'alt nivell.
- El model de dades no recull suficients conceptes semàntics, que han de ser explícitament tractats pels programadors de les aplicacions.
- Diferencies importants entre el llenguatge de la base de dades i els llenguatges de programació, tan pel que fa al model de dades subjacent com a les estructures de dades.
- Model de transaccions no apropiat per a transaccions de llarga durada.

La qüestió de l'emmagatzematge dels objectes no és fàcil, atès que la tecnologia d'objecte expandeix enormement el significat del termini dada. Amés dels tipus de dades usuals, els objectes poden contenir mètodes, estructures de dades complexes, informació multimèdia, referències a d'altres objectes i altres tipus de dades que no són fàcilment emmagatzemats a bases de dades.

Un intent d'aportar una resposta a l'increment de complexitat de les aplicacions de bases de dades ha donat com a resultat dos nous models de dades: el Model de Dades Orientat a l'Objecte (MDOO) i el model de dades Objecte-Relacional (MDOR). Aquesta evolució representa els SGBDs de tercera generació.

Els partidaris dels Sistemes de Gestió de Bases de Dades Orientats a l'Objecte (SGBDOO), consideren els sistemes relacionals adients per a aplicacions estàndards de negocis, però insuficients per suportar aplicacions complexes. Proposen desenvolupar noves tecnologies de bases de dades partint d'un model de dades d'orientació a l'objecte juntament amb un llenguatge de bases de dades adient, que permetin capacitats addicionals.

Els defensors dels sistemes relacionals creuen que la tecnologia relacional és una part necessària de tot sistema de gestió de bases de dades i que les aplicacions complexes és podrien manegar amb extensions al model relacional.

Els sistemes Objecte-Relacional es basen en un enfocament més evolutiu que integra conceptes de la orientació a l'objecte amb els sistemes de bases de dades que estenen el model de dades relacional i un llenguatge de consulta relacional.

Independentment de quin sigui el model de dades en el que es basin, ja sigui el model d'orientació a l'objecte o bé el model relacional estes, els sistemes de la propera generació han d'incloure una sèrie de noves capacitats, tot respectant les funcionalitats dels sistemes de les generacions prèvies que s'han mostrat d'utilitat.

Un intent d'unificar criteris i d'establir quines característiques ha de tenir un SGBD per ser considerat com a Orientat a l'Objecte ha estat l'article "The Object-Oriented Database Manifesto" de Atkinson, M.P.; Bancilhon, F.; DeWitt, S.J.; Dittich, K.R.; Maier, D; Zdonik, S.B., publicat a l'any 1989.

Per altra banda també s'ha tractat de determinar els principis que han de guiar el desenvolupament d'un SGBD perquè es pugui considerar com de tercera generació. Això s'ha fet a l'article publicat a l'any 1990, titulat "Third Generation Database System Manifesto" de Stonebraker, M.; Rowe, L.A.; Lindsay, B.G.; Gray, J.; Carey, M.J.; Brodie, M.L.; Berstein, P.A.; Beech, D. (Commitee for Advanced DBMS Function).

1.2 Objectius del treball

L'objectiu del nostre treball és analitzar les característiques d'orientació a l'objecte que un SGBD hauria d'incorporar per ser considerat com de Tercera Generació, tal i com es descrivien al segon dels articles esmentats a l'apartat anterior i estudiar en profunditat com l'estàndard SQL:1999 incorpora aquestes característiques.

No queda dins l'abast del treball estudiar la sintaxi de l'estàndard SQL:1999 a través de la que s'implementen aquests elements. Només s'entrarà en el detall per tal d'ajudar a una milloria en l'explicació dels exemples exposats, amb els quals es pretén millorar la comprensió dels conceptes.

Després d'identificar els elements d'orientació a l'objecte que incorpora l'estàndard SQL:1999, s'estudiarà com alguns dels SGBDs més coneguts del mercat (Oracle 9i, Informix i PostgreSQL) han implementat aquestes característiques als seus 'dialectes' del llenguatge de consulta.

Com a complement de tot aquest anàlisis es desenvoluparà un cas pràctic que permetrà posar de manifest com dos dels sistemes comercials analitzats han portat a terme la implementació dels elements d'orientació a l'objecte.

1.3 Estructura del treball

Després d'aquesta introducció, el capítol 2 comença resumint les principals característiques de la tecnologia d'orientació a l'objecte i alguns dels seus avantatges més reconeguts. Això ens permetrà situar-nos respecte de les característiques d'orientació a l'objecte que es tracten d'incorporar als SGBD de tercer generació, ja sigui en sistemes d'orientació a l'objecte pur o bé al objecte-relacional.

El següent apartat del capítol 2 enumerarà breument les principals característiques que han de tenir els SGBDs per ser considerats com Sistemes de Gestió de Bases de Dades Orientats a l'Objecte (SGBDOO) segons l'article "The Object-Oriented Database Manifesto".

El capítol 2 finalitza amb una explicació que pretén ser breu dels elements que un SGBD ha d'incorporar per se considerat un Sistema de Tercera Generació tal com es

defineix al Third-Generation Database Manifesto. Al capítol següent s'explicaran amb detall les característiques més importants des del punt de vista de l'objectiu d'aquest treball.

Al capítol 3 es realitza l'estudi de l'estàndard SQL:1999, centrat en com la seva especificació incorpora els elements d'orientació a l'objecte detallats a l'últim apartat del capítol anterior. Aquest capítol 3 té dos apartats: una breu descripció de l'estàndard SQL:1999 i un estudi de cadascun dels elements d'orientació a l'objecte detallats al "Third-Generation Database Manifesto", aprofundint més pel que fa als mecanismes de funcions i regles, analitzant en quin grau han estat recollits a l'estàndard SQL.

El capítol 4 estudia les característiques tres dels sistemes de gestió de bases de dades objecte relacionals comercials, especialment el seu tractament de regles i funcions. Els sistemes analitzats són: Oracle 9i, Informix 9.4 i PostgreSQL 7.3.

Per finalitzar, al capítol 5 s'ha desenvolupat un cas pràctic per posar de manifest la implementació de las característiques estudiades en dos dels sistemes analitzats al capítol precedent: Oracle 9i i PostgreSQL.

2 Els Sistemes de Gestió de Bases de Dades i la Tecnologia d'Orientació a l'Objecte

2.1 Tecnologia d'Orientació a l'Objecte

És important conèixer quins són els aspectes més importants que sustenten la tecnologia de l'orientació a l'objecte.

Primerament hauríem de definir el que és un objecte. Una definició possible és la donada per Connolly: "Entitat identificable de forma unívoca que conté tant els atributs que descriuen l'estat d'un objecte del món real com les accions associades a ell". Podríem dir que un objecte és un paquet de software que conté una col·lecció de procediments i dades relacionades.

Mentre que Loomis ens proporciona una bona descripció del que és la tecnologia que tracta amb objectes: "La tecnologia de l'Objecte suposa el disseny i la implementació de software en components modulars ben definits, els quals interactuen els uns amb els altres per proporcionar una funcionalitat desitjada".

Aquesta tecnologia ha generat moltes expectatives en el món del desenvolupament del software. Algunes de les millores que es pretenen amb la seva aplicació són:

- una millora de qualitat del software, fiabilitat i extensibilitat;
- uns temps de desenvolupament més curts
- una major reusabilitat del codi.

Un benefici clau d'aquesta tecnologia és la possibilitat de dissenyar sistemes de negoci adaptables, es a dir que es puguin modificar ràpidament en resposta a noves oportunitats i demandes.

A continuació relacionem els principis bàsics d'aquesta tecnologia:

- Abstracció, encapsulació i ocultació de la informació. L'abstracció tracta de modelar els objectes de la vida real identificant els seus aspectes essencials (des del punt de vista de l'aplicació). El software es desenvolupa en petits mòduls comprensibles de dades i operacions permeses sobre aquestes dades. L'encapsulació i ocultació d'informació fan una distinció clara entre les interfícies de l'objecte, que descriu el que l'objecte fa, i la implementació d'aquest objecte, que defineix com ho fa. Els detalls interns de l'objecte es troben ocults al món exterior.
- Identitat de l'objecte. En un sistema d'orientació a l'objecte cada objecte té assignat un identificador d'objecte. S'exigeix que l'objecte sigui únic en tot el sistema.

- Mètodes i missatges. Les funcions que encapsulen un objecte i que defineixen el seu comportament s'anomenen mètodes. Els objectes es comuniquen entre si mitjançant els missatges. Amb aquests sol·liciten a d'altres objectes que executin algun dels seus mètodes.
- Classes. Els objectes que tenen els mateixos atributs i responen als mateixos missatges poden ser agrupats en classes. Es poden tractar com del mateix tipus.
- Herència. Per definir nous tipus poden emprar-se tipus definits prèviament als quals s'hi afegeixen algunes característiques addicionals. L'herència permet que una classe sigui definida com cas especial (subclasse) d'una classe més general (superclasse).
- Redefinició i sobrecàrrega. La redefinició és la possibilitat de que un mètode heretat d'una superclasse pugui ser modificat en la seva implementació a la subclasse. Mentre que la sobrecàrrega és la possibilitat de que un mateix mètode pugui fer coses diferents en funció de quin objecte rep el missatge i de quins paràmetres es passen al mètode.
- Polimorfisme. És un cas més general de sobrecàrrega. Es pot utilitzar el mateix missatge per fer diferents funcions depenent de l'objecte que rep el missatge. El sistema reconeix aquest tipus d'objecte en temps d'execució i executa la rutina que correspon.

2.2 Els Sistemes de Gestió de Bases de Dades Orientats a l'Objecte

Al punt anterior hem descrit breument els principis bàsics subjacents a la tecnologia d'orientació a l'objecte. De l'enfocament que vol integrar aquests conceptes amb els sistemes de bases de dades surten els Sistemes de Gestió de Bases de Dades Orientades a l'Objecte (SGBDOO).

Malgrat que aquests sistemes prenen com a base el model d'orientació a l'objecte, quan van començar a desenvolupar-se no hi havia una interpretació comú sobre quines funcions bàsiques s'hi havien de proporcionar. A diferència del model de dades relacional no existeix un model de dades d'orientació a l'objecte universalment acceptat.

Un intent d'arribar a un acord respecte a la definició d'un SGBDOO és el "Object-Oriented Database System Manifesto". Aquest document estableix una sèrie de definicions que han de servir com a referència per determinar si un SGBD es pot considerar un SGBDOO, a més de ser punt de partida per a futures discussions sobre la matèria.

L'article proposa tretze característiques d'obligatori acompliment per a un SGBDOO. Aquestes característiques es basen en dos criteris: el sistema ha de ser orientat a l'objecte i ha de ser un SGBD. Les vuit primeres característiques fan referència a l'orientació a l'objecte i les cinc següents són relatives a les funcionalitats cal que tingui un SGBD.

A més a més d'aquestes regles hi ha dos grups més. Un grup de característiques que són opcionals, poden millorar el sistema però no són obligatòries, i un altre grup de característiques obertes, on el dissenyador pot escollir entre un nombre de solucions diferents acceptables i que tampoc són obligatòries..

A continuació es relacionen aquestes característiques sense entrar en detalls de cadascuna:

Obligatòries	
Orientació-Objecte	SGBD
1. Objectes complexes	9. Persistència
2. Identitat de l'objecte	10. Gestió de l'emmagatzematge secundari
3. Encapsulació	11. Concurrència
4. Tipus i classes	12. Recuperació
5. Herència	13. Dispositiu de consultes *ad hoc*
6. Sobrecàrrega	
7. Extensibilitat	
8. Completessa de càlculs	
Opcionals	
1. Herència múltiple	
2. Comprovació i inferència de tipus	
3. Distribució	
4. Disseny de transaccions	
5. Versions	
Obertes	
1. Paradigma de programació	
2. Sistema de representació	
3. Sistema de tipus	
4. Uniformitat	

Encara que no hi ha un consens generalitzat sobre un model de dades d'Orientació a l'Objecte, hi ha un grup, l'Object Management Group (OMG), que no crea estàndards reconeguts com l'ISO o l'ANSI però que pretén desenvolupar estàndards de facto. Aquest grup vol definir mecanismes estàndards basats en l'objecte per tal d'incorporar funcionalitats avançades. Sobre els treballs d'aquest grup es va formar l'Object Data Management Group (ODMG), amb l'objectiu de definir els estàndards dels SGBDOOs.

2.3 Manifest dels Sistemes de Gestió de Dades de Bases de Tercera Generació.

Tal com s'explicava a la introducció, l'article "Third Generation Database System Manifesto" tracta d'establir quines són les característiques que han tenir els SGBDs per ser considerats com de tercera generació. Aquest enfocament no considera estrictament

necessari dissenyar nous SGBD basats en el model de dades d'orientació a objecte, sinó que es poden afegir extensions als SGBD relacionals per tal d'assolir les funcionalitats desitjades.

Les característiques proposades es sustenten en tres principis bàsics:

- A més dels serveis tradicionals de gestió de dades, els SGBDs de tercera generació han de proporcionar un suport per a estructures de dades i regles més riques.
  Aquest principi engloba les característiques que han de permetre l'emmagatzenament i manipulació de dades no tradicionals i també la possibilitat d'especificar un conjunt de regles sobre les dades, registres i col·leccions.

- Els SGBD de tercera generació han de contenir les funcionalitats oferides pels SGBDs de segona generació. Es a dir, no es poden comprometre els avenços aconseguits. El que es fa és afegir noves característiques i millores.

- Els SGBDs de tercera generació han de ser oberts a d'altres subsistemes.

Proposicions relatives a la gestió de regles i objectes.

1. Sistema ric en tipus.

   Són desitjables els següents:
   a. Sistema de tipus de dades abstractes per construir-ne de nous tipus. Permet afegir nous tipus als ja predefinits pel sistema.
   b. Un constructor de tipus de matrius.
   c. Un constructor de tipus de seqüències.
   d. Un constructor de tipus de registres.
   e. Un constructor de tipus de conjunts.
   f. Funcions com a tipus.
   g. Un constructor de tipus d'unió.
   h. Composició recursiva dels constructors anteriors (per poder suportar objectes complexes que tenen una estructura interna).

   El llenguatge de consulta ha de poder tractar amb aquests tipus.

2. Herència. Considera que aquests sistemes han de possibilitar la creació de jerarquies de tipus. A més d'herència simple és desitjable l'herència múltiple per modelar millor certes realitats.

3. Funcions i encapsulament. L'encapsulament promou la modularitat i permet registrar les funcions junt amb les dades que encapsula. També proporciona avantatges en quant a rendiment del sistema.
   Les funcions han de poder ser heretades. Els usuaris haurien d'escriure les funcions en un llenguatge d'alt nivell i obtenir accés al SGBD mitjançant un

llenguatge d'accés no procedimental. D'altra banda els tipus no haurien de fer-se opacs als llenguatges de consulta.

4. Identificadors únics. La clau primària és un identificador únic assignat per l'usuari i que té un significat per a ell. Només si no és pot assignar una clau d'aquest tipus el sistema ha d'assignar necessàriament un identificador únic.

5. Regles (disparadors, restriccions) seran importants característiques dels futurs sistemes. No haurien de associar-se amb funcions o col·leccions específiques, per tal d'evitar la duplicació d'esforços al codificar les funcions i facilitar les modificacions en les regles en cas necessari. Per altra banda si s'associen amb funcions és fa difícil consultar les regles que en un moment donat estan definides al sistema.

Proposicions relatives a l'increment de funció de SGBD

6. Tots els accessos de programes a la base de dades haurien de fer-se mitjançant un llenguatge d'alt nivell no procedimental. A curt termini això es pot fer incrustant un llenguatge de consulta en un llenguatge de programació convencional. A llarg termini es farà afegint elements de llenguatges de consulta en un llenguatge de programació persistent.
7. Caldria oferir la possibilitat d'especificar les col·leccions de dues maneres: una mitjançant l'enumeració de membres i l'altra utilitzant el llenguatge de consulta per especificar la pertinença.
8. Són essencials les vistes actualitzables. Oferir vistes actualitzables permet realitzar canvis als esquemes definits sense afectar de forma important el funcionament de la base de dades. S'ha de mantenir la independència de les dades, el que vol dir que els detalls de la implementació física han de quedar ocults al programador.
9. El model de dades no té gaire relació amb els indicadors de rendiment del sistema.

Proposicions resultants de la necessitat d'un sistema obert.

10. Els SGBDs de tercera generació han de ser accessibles des de múltiples llenguatges d'alt nivell.
11. S'han de proveir d'interfícies estàndards per accedir als diferents SGBDs.
12. SQL és el llenguatge de consulta més estes al món dels SGBDs i per tant, cal introduir les extensions necessàries perquè es puguin implementar les noves funcionalitats.
13. El nivell més baix de comunicació entre el client i el servidor ha de ser les consultes SQL i les respostes resultants.

3 L'estàndard SQL:1999 i les característiques d'Orientació a l'Objecte

3.1 Breu descripció de SQL:1999

El llenguatge SQL (Structures Query Language) es va desenvolupar per suportar el model de dades relacional, model que té els seus orígens als articles de Codd de l'any 1970. D'ençà, s'han desenvolupat molts SGBDs basats en SQL.

Per assolir la portabilitat entre els diferents sistemes, l'American National Standard Institut (ANSI) va dur a terme un projecte de desenvolupament d'unes especificacions del llenguatge. El resultat va ser l'estàndard de l'any 1986. A l'any 1987 es publica l'estàndard inicial de l'International Standards Organization (ISO), que va ser modificat a l'any 1989, donant com a resultat un segon estàndard amplament adoptat, l'SQL-83 o SQL1.

A l'any 1992 es va procedir a una important revisió de l'estàndard ISO, resultant el SQL-92 o SQL2.

A l'any 1999 es formalitza la nova versió de l'estàndard, anomenada SQL3 (ISO 1999a). Aquesta versió conté funcionalitats addicionals per suportar la gestió de dades amb orientació a l'objecte.

La naturalesa constantment evolutiva de l'estàndard SQL ha donat peu a nombrosos *dialectes* entre els diferents fabricants i productes.

L'estàndard SQL:1999 consta de les següents parts:

- Framework. Defineix l'esquema conceptual en que es basen les altres parts i els termes així com la notació que s'utilitzen.
- Foundation. Inclou les definicions de dades i la seva manipulació amb els nous tipus de dades. També inclou: funcions, suport a objectes, regles, triggers, seguretat i transaccions.
- Persistent Stored Modules (PSM). Permet la definició de funcions i procediments en llenguatges d'alt nivell i SQL, i el seu emmagatzematge a la Base de Dades.
- Call-Level Interface (CLI). Conjunt de especificacions de llenguatge utilitzats pels fabricants de SGBDs per permetre l'accés a SQL a través de crides a rutines completament especificades.
- Bindings. Permet crides dinàmiques de codi SQL inserit dins d'un llenguatge amfitrió.

3.2 Elements d'Orientació a l'Objecte

Aquest apartat tracta amb més detall el grup de proposicions agrupades sota el primer principi exposat a l'article "Third Generation Database System Manifesto" en relació a l'estàndard SQL: 1999. De totes les característiques que són desitjables per a un SGBD segons aquest article, ens interessaran sobretot les que fan referència als elements d'orientació a l'objecte i especialment les proposicions 1.3 i 1.5 sobre funcions i regles.

3.2.1 Sistema ric de tipus

Els nous tipus que s'han afegit sobre l'estàndard SQL92 són:

- Tipus predefinits: Booleà, BLOB (Binary Large Object) i CLOB (Character Large Object).
- Tipus REF.
- Tipus ARRAY
- Tipus ROW
- Tipus definits per l'usuari (User-Defined Types, UDT).

Dels tipus col·lecció que segons l'article serien desitjables (array, sequence, set), a l'estàndard SQL: 1999 només s'hi ha especificat el tipus ARRAY.

Els tipus de dades que donen suport a l'orientació a l'objecte són els UDTs i el tipus Referència (REF).

No entrarem massa en detall en els sistema de tipus, atès que ha estat l'objecte de treballs anteriors i ha quedat abastament documentat.

3.2.1.1 Tipus ROW

És una seqüència de parells de nom de camp/tipus de dada que permet a un tipus de dades representar files d'una taula, de manera que files complertes es poden emmagatzemar en variables. Això permet que les columnes d'una taula continguin valors del tipus ROW, sent com niuar una taula dins d'una altra.

3.2.1.2 Tipus ARRAY

És una col·lecció ordenada d'elements, els quals poden ser referenciats per la seva posició ordinal dins l'array. El tipus array es parametritza amb el tipus dels elements que pot contenir.

3.2.1.3 Tipus definits per l'usuari

Dins dels tipus definits per l'usuari trobem dues categories:

- Tipus diferenciats

- Tipus estructurats

Tipus diferenciats

És l'UDT més simple. Són tipus que es construeixen a partir d'un tipus predefinit (per exemple un char, integer, etc.). Aquests tipus permeten introduir més semàntica que la permesa per la definició dels dominis al SQL92 (que també es mantenen al SQL: 1999). Els dominis només permeten restringir el conjunt de valors que pot prendre un atribut d'un tipus predefinit. Aquests tipus poden tenir definides altres funcions i mètodes. Això també els fa molt més potents que els dominis.

Tipus estructurats

Aquesta categoria és la més propera al concepte de classe de l'orientació a l'objecte. L'estàndard SQL: 1999 permet a l'usuari crear qualsevol número de tipus estructurats que poden ser utilitzats com els tipus proveïts pel propi llenguatge. Poden ser el tipus de dades d'una columna o d'una variable, per exemple.

Un tipus estructurat és defineix com una o més definicions d'atributs i cap o varies declaracions de mètodes. Tot UDT ve proveït de manera automàtica d'uns mètodes com són, el mètode constructor que permet obtenir una instància del tipus, els mètodes observer i mutator, definits per a cadascun dels atributs i que permeten la recuperació i modificació del valor de l'atribut respectivament. A banda d'aquest, es poden definir d'altres, de tal manera que es pugui dotar al tipus la semàntica desitjada. Es veurà això al l'apartat de funcions.

3.2.1.4 Taules tipificades

A més dels tipus estructurats, SQL: 1999 ha afegit la possibilitat de definir una taula basada en un tipus estructurat. Les taules així definides s'anomenen taules tipificades (typed tables).

Cadascuna de les files d'aquesta taula serà una instància del tipus estructurat el que es basa. Així doncs, cada columna de la taula es correspon amb un dels atributs del tipus.

Cada taula, a més de les columnes corresponents als atributs del tipus, pot tenir una columna addicional que s'anomena columna auto-referenciada (self-referencing column). El valor emmagatzemat en aquesta columna per a una fila donada és un valor que la identifica de manera unívoca en tota la bases de dades (aquest és un punt a tractar a l'apartat de l'identificador únic, que es veurà més endavant). El tipus de dades de la columna auto-referenciada és REF, el tipus referència. Quan una taula tipificada té aquesta columna es diu que és una taula referenciable.

Els mètodes associats amb el tipus de la taula queden associats amb aquesta, pel que cridar un mètode d'una instància del tipus és el mateix que cridar el mètode a una fila de la taula.

3.2.1.5 Tipus referència (REF)

És un tipus que permet referenciar des d'un lloc a un altre. Un taula pot emmagatzemar un valor del tipus referència, que s'utilitzarà com a una referència directa a una fila d'una taula determinada. Però, aquesta taula cal que sigui una taula tipificada i el tipus de la taula ha de ser el mateix tipus amb el que s'ha definit el tipus referència.

3.2.2 Herència

Com s'explicava abans, l'herència és una de les característiques més important de la tecnologia d'orientació a l'objecte. Dèiem que l'herència permet que una classe sigui definida com cas especial (subclasse) d'una classe més general (superclasse), essent possible afegir atributs nous o mètodes nous i modificar els mètodes existents.

L'estàndard SQL: 1999 suporta el mecanisme d'herència. Es permès definir:

- Una jerarquia de tipus.
- Una jerarquia de taules.

Jerarquia de tipus

Un UDT pot participar d'una jerarquia de subtipus/supertipus. Un subtipus hereta tots els atributs i mètodes del seu supertipus i a més pot definir atributs i mètodes addicionals. També pot redefinir les funcions heretades del supertipus.

SQL:1999 no suporta l'herència múltiple i sols els tipus estructurats poden formar una jerarquia de tipus.

Jerarquia de taules

Les taules també poden participar d'una jerarquia de subtaula/supertaula. Es a dir, podem definir una taula com a subtaula d'una altra que s'anomena supertaula.

Les jerarquies de taules s'estableixen sobre taules tipificades. Totes les taules d'una determinada jerarquia han basar-se en tipus que formen part de la mateixa jerarquia de tipus. Al voler recuperar informació d'una taula el sistema realitza una unió d'aquesta taula i de totes les seves subtaules.

3.2.3 Funcions i encapsulament

Els procediments i funcions permeten portar al SGBD part de la lògica de l'aplicació, de manera que es pugui minimitzar l'intercanvi de dades entre aquesta i la base de dades, i també poder facilitar la portabilitat i el manteniment d'aquesta lògica en diferents sistemes. La tecnologia d'orientació a l'objecte es caracteritza per implementar les funcionalitats d'una aplicació mitjançant objectes. Això implica que aquests objectes tenen un comportament, el qual s'implementa associant-li unes determinades rutines (mètodes, funcions i procediments).

Un SGBD proveeix de moltes funcions predefinides que permeten realitzar operacions diverses, però des del punt de vista de l'orientació a l'objecte i, com hem dit abans, en relació a la incorporació de part de la lògica de l'aplicació, ens interessen les rutines definides per l'usuari.

Una de les parts de l'estàndard SQL:1999 és l'SQL/PSM (Persistent Stored Modules), que permet escriure procediments i rutines definides per l'usuari en un llenguatge de tercera generació o en SQL i emmagatzemar-los a la base de dades.
Aquesta part estandaritza per primera vegada la possibilitat per als programadors de SQL d'escriure funcions i procediments (SQL-invoked routines) per tal de poder utilitzar-los dins del seu codi SQL. Aquestes rutines poden ser escrites en SQL, però molts productes comercials han desenvolupat els seus propis dialectes de SQL (com per exemple PL/SQL d'Oracle). També poden ser escrites en llenguatges com C o Java.

L'estàndard SQL:1999 permet crear rutines associades als tipus definits per l'usuari, però també separadament com a part d'un esquema. Segons SQL/PSM, es diu que una rutina es dependent d'un tipus definit per l'usuari si és creada durant la declaració del tipus. Aquestes rutines no poden eliminar-se directament sinó que s'eliminen de manera implícita quan s'elimina el tipus (amb una sentència DROP DATA TYPE).

Quan s'han explicat els tipus definits per l'usuari, hem dit que consistien en una o més definicions d'atributs i en zero o més declaracions de rutines. Aquestes rutines especifiquen el comportament dels UDTs. El comportament d'un tipus són les accions que pot realitzar una instància del tipus.

Les rutines definides per l'usuari poden ser procediments, funcions o mètodes (aquest darrer tipus afegit per l'estàndard SQL:1999). Les característiques de cadascuna són:

- Procediments. Son rutines que tornen valors només per mitjà de paràmetres. Poden tenir zero o més paràmetres, els quals poden ser d'entrada (IN), de sortida (OUT) o d'entrada i sortida (INOUT). Són cridats amb una sentència CALL.
- Funcions. Rutines que tornen un valor sense necessitat d'utilitzar un paràmetre. Només poden tenir paràmetres d'entrada i només es pot tornar un valor des d'una funció. Les funcions (i els procediments) no estan necessàriament associades a un UDT específic, poden definir-se en altres esquemes de la base de dades. Per a la seva invocació s'utilitza una notació funcional.

- Mètodes. Són casos especials de funcions. Estan associats a un UDT específic i per tant s'han de definir a l'esquema en el que s'ha definit el seu tipus associat. La sintaxi d'invocació és la notació amb punt (dot notation).

Així doncs, un mètode, a diferència d'una funció o un procediment, està estretament lligat a un tipus definit per l'usuari.

En SQL la signatura del mètode es declara com a part de la definició del tipus, però la seva implementació està en una altra part. Després es poden afegir o esborrar mètodes amb la clàusula ALTER TYPE.

Exemple

```
CREATE TYPE persona (
        Dni     VARCHAR(9),
        Nom     VARCHAR(25),
        DataNeix        DATE)
NOT FINAL
METHOD edat()                   /* declarem el mètode amb la seva signatura */
        RETURNS INTEGER

/* Ara es defineix el mètode */
CREATE INSTANCE METHOD edat ()
        RETURNS INTEGER
        FOR empleat
        /* aquí aniria el codi del mètode */
```

Si per exemple joan fos una instància de persona, invocaríem el mètode de la següent manera: *joan.edat*.

Posat que la rutina sigui externa, és a dir proveïda externament en un llenguatge de programació estàndard (com C, Java o C++), s'ha de definir especificant una clàusula externa que identifiqui el corresponent codi compilat dins del sistema d'arxius del sistema operatiu.

Exemple:

```
CREATE FUNCTION calcul_descompte (cost REAL, percent INTEGER)
        RETURNS REAL
        LANGUAGE C
        SPECIFIC tenda.article.descompte_en_c
        DETERMINISTIC
        NO SQL
        EXTERNAL NAME 'file://C:/rutines/descomptes.bin'
        PARAMETER STYLE GENERAL
```

A l'exemple anterior la clàusula LANGUAGE indica el llenguatge en que està escrit el mètode (per defecte és SQL). NO SQL vol dir que el mètode no conté sentències SQL. Que és DETERMINISTIC vol dir que retorna el mateix resultat en resposta a un conjunt de valors dels arguments per un estat donat de la base de dades.

Els mètodes poden ser de dos tipus: mètodes estàtics i mètodes d'instància. Els primers es declaren utilitzant la paraula clau STATIC a la seva declaració i operen sobre el tipus mateix, mentre que els segons es declaren utilitzant la paraula clau INSTANCE i operen sobre una instància del tipus als quals estan associats.

Un mètode es pot declarar amb paràmetres o sense, però de forma implícita el SQL sempre crea un paràmetre implícit que precedeix al primer paràmetre declarat (si n'hi ha). Aquest paràmetre implícit s'anomena SELF i sempre és del tipus associat al mètode. Tothora podem invocar un mètode fent referència a la instància de l'UDT mitjançant el nom SELF (similar al *this* del llenguatge JAVA). Les funcions no tenen aquest paràmetre implícit.

A SQL:1999 quan s'invoca una funció, es busca al SQL-path, que és una llista de esquemes de base de dades en els que pot estar localitzada la funció. Cada rutina té associat un SQL-path que hereta de l'esquema en el que està definida.

Encapsulació

Les dades emmagatzemades als atributs d'un tipus no poden ser accedides directament. Aquesta estructura de dades només es pot veure i modificar mitjançant mètodes definits al tipus.
Els comportaments d'un tipus també estan encapsulats mitjançant les rutines que els implementen. Així definim una interfície del tipus. Poden modificar la implementació dels comportament i les aplicacions que utilitzen la base de dades no es veuran afectades (sempre que no es modifiqui la interfície).

A SQL: 1999 a tot tipus estructurat es defineixen, de forma automàtica, els següents mètodes:

- Dos mètodes d'instància per a cada atribut del tipus.
  El mètode **Observer**, per llegir el valor de l'atribut. Quan s'invoca torna el valor actual de l'atribut.
  El mètode **Mutator** per modificar el valor de l'atribut.
- Un mètode constructor. Té el mateix nom que el tipus. Aquest mètode serveix per crear noves instàncies del tipus.

Els mètodes anteriors poden ser sobrecarregats (overloaded) però no redefinits (overriden).
Altres mètodes d'un tipus definit per l'usuari són els que estableixen certes propietats d'ordenació de les instàncies del tipus. Aquests mètodes serveixen per fer comparacions. L'ordenació es pot realitzar utilitzant mètodes que són qualificats com:

- RELATIVE. Mètode que torna 0 per iguals, un valor negatiu per menor que, i un valor positiu per major que.
- MAP. Utilitza una funció que pren un sol argument de l'UDT i torna tipus de dades predefinit.
- STATE. Compara els atributs dels operands per determinar l'ordre.

Polimorfisme

Ja hem vist que SQL: 1999 suporta el mecanisme d'herència. L'herència permet definir subtipus a partir de supertipus. Els subtipus creats tindran tots els atributs i rutines del supertipus. A més es poden afegir nous atributs i mètodes, així com modificar els mètodes existents.

Els mètodes es poden redefinir (override), el que significa que mantenint la signatura del mateix (nom i arguments) es pot modificar la implementació alhora es poden sobrecarregar (overload), el que significa que es poden definir nous mètodes amb el mateix nom als ja existents però diferenciats respecte del número i/o tipus dels arguments.

Per tant, és possible l'existència de diferents rutines amb el mateix nom. A SQL: 1999 totes les rutines tenen dos noms:

- El nom específic, és un nom ordinari qualificat per l'esquema. Així s'assegura que no hi hauran dues rutines amb el mateix identificador en un esquema donat i que és únic a una base de dades.
- El nom de la rutina, es podria dir el nom d'invocació. És el que s'utilitza per invocar un mètode o funció. Aquest és el nom que poden tenir diferents rutines a la base de dades.

Per determinar quina de les rutines del mateix nom és invocada s'utilitza la llista d'arguments de la rutina.

En SQL: 1999 les funcions i mètodes són polimòrfics. El polimorfisme té a veure amb la relació que s'estableix entre la invocació a un mètode i el codi que efectivament s'associa amb aquesta crida. Quan els noms són els mateixos aquesta vinculació es pot fer en temps d'execució (late binding). Això fa que amb un mètode declarat en un supertipus i amb els seus subtipus redefinits, la determinació de quina definició del mètode s'utilitzarà es deixi fins al moment de l'execució, quan ja es coneixerà el tipus més específic de l'argument implícit del mètode.

3.2.4 Identificadors únics

La tecnologia d'orientació a l'objecte proporciona un identificador únic per cadascuna de les instàncies dels objectes que es creïn al sistema. Aquest identificador cal que sigui independent dels valors dels atributs de l'objecte (el seu estat). Aquest identificador tampoc pot canviar al llarg de la vida de l'objecte i no pot ser reutilitzat quan aquest és esborrat. D'aquesta manera es proporciona automàticament la integritat d'entitat requerida.

A l'estàndard SQL: 1999 quan creem un tipus podem determinar que el sistema assigni un valor REF únic associat a qualsevol instància del tipus.

Els valors assignats pel sistema són invariables. Mentre aquesta instància existeixi, serà únic en tota la base de dades i no serà reutilitzat.

En SQL: 1999 les taules tipificades poden tenir una columna addicional, anomenada columna auto-referenciada, que conte valors del tipus REF i que serveixen per identificar cada fila de manera única en tota la base de dades.

Els valors REF poden ser:

- Generats pel sistema. Aquest cas és el que s'adaptaria als requeriments de l'identificador únic que demana la tecnologia d'orientació a l'objecte.
- Generats per l'usuari. S'obté a partir de valors proporcionats per l'usuari.
- Derivats de dades emmagatzemades a un o varis atributs de la instància del tipus. En aquest cas, aquests atributs cal que formin part d'una restricció d'unicitat (restricció UNIQUE).

3.2.5 Regles: disparadors i restriccions

Segons el "Third Generation Database System Manifesto", el fet de que les regles s'implementen mitjançant rutines que estan associades als tipus o taules pot provocar dos problemes. El primer que el programador quan implementa una funció sigui conscient de que s'ha d'invocar la funció que implementa una regla determinada. Això no garantiria que el sistema forci el compliment de la regla. El segon problema és que es dificulta conèixer totes les regles definides dins del sistema.

SQL:1999 proporciona una forma d'implementar regles fora de la definició dels tipus o de les taules, que són els disparadors (triggers). Un disparador permet als dissenyadors de bases de dades donar instruccions al SGBD perquè realitzi certes operacions cada vegada que una aplicació faci operacions específiques sobre algunes taules.

Els disparadors permeten forçar el compliment de les regles dins de la base de dades. Això també permet reduir la complexitat de les aplicacions basades en la base de dades. Anteriorment era responsabilitat de cada programa d'aplicació assegurar el compliment de les regles i restriccions.

Els disparadors són com rutines de SQL però, en lloc de ser invocades, s'executen de manera implícita si s'escau un determinat esdeveniment, com pot ser inserir una fila a una taula o esborrar o actualitzar una fila existent (INSERT, DELETE i UPDATE).

L'ús dels triggers té alguns desavantatges com poden ser: afegir complexitat al disseny i implementació de la base de dades, ocultar funcionalitats a l'usuari i la reducció del rendiment de la base de dades, atès que cada acció requereix comprovar si s'ha d'executar un disparador i en cas afirmatiu executar-se.

Exemple de creació d'un disparador:

```
CREATE TRIGGER control_sou
        UPDATE OF sou ON empleat
                REFERENCING NEW AS nou_sou
                                OLD AS antic_sou
                FOR EACH ROW
                        WHEN antic_sou.sou > nou_sou.sou
                        (EXECUTE PROCEDURE refusar_insercio)
ENABLED;
```

Impedeix una modificació a la baixa del sou d'un empleat.

Tot disparador té tres components principals: esdeveniment del disparador (INSERT, UPDATE o DELETE), condició temporal (BEFORE, AFTER o FOR EACH ROW) i acció del disparador (INSERT, DELETE, UPDATE i EXECUTE PROCEDURE).

Els triggers no són l'únic mecanisme proporcionat per SQL per tal de definir regles dins del sistema. Abans de l'estàndard SQL:1999 ja s'havien proporcionat funcionalitats per definir restriccions d'integritat o d'altres tipus.

Les restriccions d'integritat protegeixen la base de dades d'arribar a situacions inconsistents. Són de cinc tipus:

1. Dades requerides. Algunes columnes han de contenir un valor vàlid; no poden contenir nuls.
2. Restriccions de domini. Cada columna té un domini és a dir, pot prendre un conjunt de valors possibles. SQL permet crear explicitament dominis amb la sentència CREATE DOMAIN. Val la pena recordar aquí la possibilitat que ofereix SQL:1999 de crear tipus diferenciats (DISTINCT TYPES), que introdueixen més semàntica que els dominis.
3. Restriccions d'integritat d'entitat. La clau primària d'una taula ha de contenir un valor únic i no nul per a cada fila.
4. Restriccions d'integritat referencial. Si la clau externa d'una taula conté un valor, aquest s'ha de referir-se a una fila existent i vàlida de la taula relacionada.
5. Restriccions de negoci. Permeten establir requeriments i regles del negoci suportat per la base de dades. D'aquesta manera es força el compliment de la regla sense que el codi de l'aplicació es vegi obligat a fer-ho constantment. Es poden especificar utilitzant les clàusules CHECK i UNIQUE de les sentències CREATE i ALTER TABLE i la sentència CREATE ASSERTION. Aquesta darrera restricció no esta directament vinculada a la definició d'una taula. Els triggers també es poden fer servir per definir regles de negoci.

4 L'estàndard SQL:1999 i els Sistemes de Gestió de Bases de Dades Objecte-Relacionals comercials

4.1 Productes analitzats

4.1.1 Oracle 9i

La versió d'Oracle estudiada en aquest treball és la 9i. Es tracta d'un servidor de base de dades Objecte-Relacional no gratuït, desenvolupat per l'empresa Oracle Corporation. Esta pensat per treballar en entorns client/servidor amb alt grau de concurrència i necessitats d'emmagatzematge de dades molt importants.

És un producte amb un alt grau d'adherència als estàndards de la indústria, tant pel que fa al llenguatge d'accés a les dades, com als sistemes operatius, interfícies d'usuari i protocols de comunicació en xarxa.

4.1.2 Informix

Informix és un producte no gratuït de l'empresa IBM. La versió que hem utilitzat és Informix Dynamic Server 9.4.

L'origen d'aquest gestor de base de dades es troba a inicis dels 80. Va néixer com un SGBD relacional dins de l'entorn UNIX (el seu nom prové de INFORMation + unIX). En 1990 s'afegeixen moltes funcionalitats que formen part del nucli dels SGBDR. En 1992 s'afegeixen els disparadors (triggers).

En 1999 es fusionen les versions 7.x (Informix Dynamic Server) i 9.x (Universal Data Option) donant com a resultat Infromix Dynamic Server 2000, que comença amb la versió 9.2. Aquesta versió proporciona extensió per objectes.

4.1.3 PostgreSQL 7.3

PostgreSQL és un SGBD de codi obert i gratuït. Els seus orígens es troben a l'Universitat de Califòrnia a Berkeley, on neix el sistema de base de dades Ingres a finals dels 70. Em 1986, Michael Stonebraker va afegir funcionalitats d'Orientació a l'Objecte i la nova versió es va anomenar Postgre. Més tard, a mitjans dels 90, es va afegir suport a SQL, i el nom va canviar a PostgreSQL. Actualment aquests SGBD es desenvolupa per un grup internacional de software de codi obert anomenat PostgreSQL Global Development Group.

Les característiques d'aquest sistema són:

- Objecte-Relacional.
- Compliment amb l'estàndard SQL 92 i bastants funcionalitats de l'estàndard SQL:1999.
- Codi obert.
- Processament de transaccions.
- Integritat referencial. Implementa integritat referencial complerta. Permet establir regles de negoci i restriccions.
- Utilització de múltiples llenguatges procedimentals.
- Extensibilitat. Possibilitat de definir nous tipus de dades, noves funcions i nous operadors

4.2 Característiques d'Orientació a l'Objecte i implementació als SGBD Objecte-Relacionals analitzats

4.2.1 Sistema de tipus

4.2.1.1 Oracle

Oracle defineix dues categories de tipus definits per l'usuari:

- Tipus Objecte (Object Types)
- Tipus Col·lecció (Collection Types)

Tipus Objecte

Els tipus objecte és l'equivalent al User Defined Type de l'estàndar SQL:1999. Oracle anomena objecte a les instàncies d'aquests tipus definits per l'usuari.

Taules Objecte

Una taula objecte (Object Table) és un tipus especial de taula que conté objectes (instàncies de tipus objecte) i proporciona una vista relacional dels atributs d'aquests objectes. Són l'equivalent de les taules tipificades de l'estàndard SQL:1999.

Tipus REF

Com el tipus REF definit per l'estàndard SQL, Oracle també ofereix un tipus que permet referenciar un objecte fila d'un determinat tipus objecte. . Aquest tipus permet expressar relacions de molts a un, quan el costat un és un objecte fila.

Tipus col·lecció

Un tipus col·lecció d'Oracle descriu una unitat de dades composta d'un nombre indefinit d'elements del mateix tipus de dades.

Els tipus col·leccions oferts són: tipus VARRAY i taules niuades (nested tables)

El tipus VARRAY és com el tipus ARRAY de l'estàndard, però pot tenir un nombre variable d'elements, per això el seu nom té una V a l'inici.

Una nested table és un conjunt d'elements de dades no ordenats, tots del mateix tipus de dades. Té una sola columna, i el tipus d'aquesta columna és un tipus predefinit o bé un tipus definit per l'usuari.

Vistes Objecte (Object Views)

Utilitzant les vistes objecte es poden crear taules objecte virtuals amb dades (objecte o predefinits) emmagatzemats en les columnes de taules objecte o taules relacionals de la base de dades.

Proporciona la flexibilitat de contemplar la mateixa dada objecte o relacional de més d'una manera. Així es poden utilitzar diferents representacions de l'objecte en memòria per a diferents aplicacions sense canviar la manera en que les dades s'emmagatzemen a la base de dades.

Aquest mecanisme facilita molt la utilització de dades relacionals en aplicacions orientades a l'objecte i provar tècniques de programació d'orientació a l'objecte sense convertir les taules existents.

4.2.1.2 Informix

Informix també ofereix un sistema ampliat de tipus de dades. Hi han dos grups: els tipus complexes i els tipus definits per l'usuari.

Els tipus complexes són els creats a partir d'una combinació d'altres tipus de dades. Informix proporciona els tipus col·leccions, amb LIST, SET i MULTISET, i els tipus fila (ROW type).

Dintre dels tipus definits per l'usuari diferencia els tipus opacs (Opaque types)i els diferenciats (Distinct types).

Col·leccions

Depenent de si els elements han de estar ordenats o no, o si poden existir duplicats, utilitzarem un tipus SET, LIST o MULTISET.

El tipus LIST és l'equivalent a l'ARRAY de l'estàndard. Emmagatzema una col·lecció d'elements a una posició implícita i pot contenir valors duplicats.

A més ofereix tipus col·lecció no implementats per l'estàndard com són els SET i MULTISET. El tipus SET emmagatzema una col·lecció de valors sense una posició implícita i no permet valors duplicats (són conjunts). El tipus MULTISET emmagatzema una col·lecció de valors que no tenen una posició implícita, però admet valors duplicats.

Tipus fila (ROW)

Hi han dos tipus fila: amb nom i sense nom. Els primers tenen un nom definit i propietats d'herència i es poden utilitzar per construir una taula tipificada. Els segons no tenen nom definit ni propietats d'herència.

Tipus definits per l'usuari

Com l'estàndard, Informix permet definir tipus diferenciats (Distinct Types).

L'altra tipus que permet definir és el tipus opac (Opaque Type). Aquest és un tipus de dades quina estructura interna és desconeguda pel servidor de la base de dades. Aquesta estructura és definida per l'usuari, així com les funcions de suport i les operacions. Aquests tipus es poden utilitzar com els tipus predefinits pel sistema.

Taules tipificades

Informix també permet la creació de taules tipificades (typed tables). Només les taules tipificades poden conformar una jerarquia de taules.

4.2.1.3 PostgreSQL

PostgreSQL, a més del sistema de tipus predefinits que subministra i que són els tipus oferts per l'estàndard SQL:1999, també ofereix uns tipus predefinits exclusius, que permeten tractar amb objectes complexes de tipus geomètric i entitats d'Internet.

PostgreSQL permet definir nous tipus de dades (els Custom Data Type). Crear un nou tipus requereix determinar la sintaxi requerida pels valors de literals, el format per a l'emmagatzematge intern de les dades, el conjunt d'operadors suportats pel nou tipus, i el conjunt de funcions que poden operar amb valors d'aquest tipus.
La definició d'aquests nous tipus es fa per sota del llenguatge SQL. Crear un nou tipus suposa implementar funcions per operar amb el tipus en un llenguatge de baix nivell, normalment en C. Una vegada creat el tipus, es poden declarar funcions addicionals per proveir d'operacions útils sobre el tipus.

És possible crear un tipus compost, que s'especifica mitjançant una llista de noms d'atributs i tipus de dades. De fet, és com el tipus ROW que hem vist a l'estàndard SQL:1999.

4.2.2 Herència

4.2.2.1 Oracle

Oracle ha implementat les característiques d'herència definides a l'estàndard. A l'igual que aquest, Oracle només permet l'herència simple.

En Oracle trobem la possibilitat d'establir jerarquies de tipus i jerarquies de vistes objecte. Un vista objecte pot ser creada com una subvista d'una altra vista objecte. El tipus de la supervista ha de ser el supertipus immediat del tipus de la vista objecte que estem creant.

Exemple:

```
CREATE VIEW persona_v OF persona_t
        WITH OBJECT ID(dni) AS
        SELECT dni, nom, adreça FROM totespersones
        WHERE tipusid = 1;

CREATE VIEW estudiant_v OF estudiant_t UNDER persona_v AS
        SELECT dni, nom, adreça, depart_id, tutor FROM totespersones
        WHERE tipusid = 2;
```

La subvista hereta l'identificador d'objecte de la seva supervista.

4.2.2.2 Informix

Aquets SGBD permet l'herència de tipus i de taules. A l'igual que l'estàndard només suporta l'herència simple.

A Informix l'herència de tipus només s'aplica a tipus de files amb nom. Les rutines associades als tipus es poden sobrecarregar.

4.2.2.3 PostgreSQL

PostgreSQL només suporta l'herència de taules

4.2.3 Funcions

4.2.3.1 Oracle 9i

Oracle permet definir els comportaments dels objecte mitjançant la definició de mètodes. Els mètodes poden ser:

- Mètode membre (member method). És una funció o procediment que sempre té un paràmetre implícit, SELF, com el seu primer paràmetre i es del tipus de l'objecte que el conté.
- Mètode estàtic (static method). Es una funció o procediment sense el paràmetre implícit SELF. Afecta al tipus mateix i no a un objecte instanciat.
- Mètode de comparació (comparison method). Per comparar instàncies de tipus objecte. Aquests mètodes poden ser map (map methods), que aprofiten la possibilitat de comparació dels tipus predefinits pel sistema o d'ordre (order methods), que utilitzen la seva pròpia lògica interna.

Tot tipus objecte té un mètode constructor definit pel sistema. Aquest mètode crea un nou objecte d'acord amb l'especificació del tipus objecte. A Oracle però, el sistema no proporciona automàticament uns mètodes *observer* i *mutator* per cada atribut de l'UDT, és l'usuari qui els ha de definir.

Oracle permet que l'usuari adapti les funcions d'agregació (MAX, MIN, SUM, etc.) per poder operar amb els nous tipus definits. També permet crear funcions d'agregació completament noves.

Els mètode poden implementar-se en PL/SQL, Java i C. Es possible la sobrecàrrega i la redefinició dels mètodes. Com a l'estàndard els mètodes són polimòrfics.

Exemple de definició d'un mètode a Oracle 9i:

```
CREATE TYPE rectangle AS OBJECT (
        base    INTEGER,
        alt     INTEGER,
        MEMBER FUNCTION area);

CREATE TYPE BODY rectangle AS
        MEMBER FUNCTION area  RETURN INTEGER IS
        var INTEGER;
        BEGIN
                var := SELF.base * SELF.alt;
                RETURN var;
        END;
END;
```

Però a més dels mètodes a Oracle es poden definir funcions i procediments. Les diferències entre funcions, procediments i mètodes són les mateixes que hem descrit dins de l'apartat de funcions de l'estàndard SQL:1999.

En Oracle es possible crear paquets (packages) que són col·leccions de procediments, funcions, variables i sentències SQL que s'agrupen juntes i s'emmagatzemen com una unitat de programa única.

No podem acabar aquest punt sense explicar que és PL/SQL. De forma breu podem dir que és una extensió procedimental de SQL. Afegeix conceptes com declaracions de variables i constants, estructures de control, maneig d'excepcions, sentències

condicionals i modularització, que SQL no té. PL/SQL és un llenguatge de programació amb la seva pròpia sintaxi, les seves pròpies regles i els seu compilador.

4.2.3.2 Informix

Hem vist que un usuari pot definir nous tipus, com són els tipus opacs. Informix permet definir les funcions necessàries que la base de dades necessita per interactuar amb un tipus de opac definit per l'usuari. D'aquesta manera s'implementa l'encapsulació, ja que només es poden accedir a les dades de l'estructura interna del tipus mitjançant rutines de l'usuari.

Els tipus opacs són definits fora de la base de dades, però es manté dins de la mateixa una llista amb els tipus i una referència a les seves rutes d'accés. El tipus de dades opac no és visible per a la base de dades. Les funcions utilitzades amb els tipus opacs s'han d'escriure en llenguatge C. La funció és registrada en la base de dades utilitzant la sentència CREATE FUNCTION.

A més de funcions d'accés a l'estructura del tipus es poden definir funcions de conversió i funcions o procediments que realitzen operacions comuns sobre el tipus.

També poden crear funcions que s'associen a tipus fila amb nom (ROW TYPES). Aquestes funcions defineixen les regles específiques del tipus. En aquest cas les funcions no estan lligades explícitament als tipus, tal com els mètodes en el cas de l'estàndard SQL:1999. Però les funcions poden ser enllaçades dinàmicament amb els tipus mitjançant els paràmetres que aquestes funcions utilitzen.

La sobrecàrrega i redefinició de funcions es possible. Els llenguatges pels que el servidor de bases de dades Informix dona suport en la definició de rutines són: SPL (Llenguatge de procediments emmagatzemats), C i Java. A l'igual que Oracle amb PL/SQL, Informix ha creat una extensió de SQL que permet crear procediments emmagatzemats, es tracta de l'SPL.

A Informix les rutines definides per l'usuari són polimòrfiques.

4.2.3.3 PostgreSQL

Es possible definir noves funcions. Les funcions que trobem a PostgreSQL són de quatre tipus:

- Funcions escrites en SQL. Executen un llista arbitrària de sentències SQL, tornant el resultat de la darrera consulta de la llista.
- Funcions en llenguatge procedimental, com PL/pgSQL o PL/Tcl.
- Funcions internes. Són funcions escrites en llenguatge C i que s'han enllaçat estàticament amb el servidor PostgreSQL. Han de ser declarades durant la inicialització del cluster de la base de dades (*initdb*).

- Funcions d'extensió. Es compilen en objectes carregables dinàmicament (com les llibreries dll) i són carregades sota demanda del sistema. Es creen en dos fases: primer s'escriu en un llenguatge de programació (com C o C++) i es compilen en un mòdul objecte dinàmic (.dll o .so) i segon, es declara la funció en PostgreSQL, amb el comando CREATE FUNCTION.

No hem vist la possibilitat de declarar mètodes, funcions o procediments associats a l'esquema d'un tipus, tal com es pot fer a Oracle i Informix.

Les funcions en PostgreSQL permeten la sobrecàrrega

4.2.4 Disparadors i restriccions

4.2.4.1 Oracle 9i

Oracle permet definir restriccions de negoci de diferents maneres: utilitzant SQL amb les clàusules CHECK i CONSTRAINT, procediments emmagatzemats , funcions, mètodes i disparadors.

Oracle implementa els disparadors especificats a l'estàndard SQL:1999. Els disparadors a Oracle suporten l'execució de blocs de codi PL/SQL i també de procediments escrits en PL/SQL i Java.

Els disparadors que es poden construir a Oracle estan associats a una taula, però també poden estar associats a una vista, a un esquema o la base de dades mateixa. Així trobem:

- Disparadors DML (data Manipulation Language) sobre taules.
- Disparadors INSTEAD OF sobre vistes.
- Disparadors de sistema sobre DATABASE o SCHEMA. Els de DATABASE s'executen per tots els esdeveniments i tots els usuaris; amb SCHEMA, s'executen per cada esdeveniment d'un usuari específic.

Els disparadors INSTEAD OF i els disparadors d'esdeveniments de sistema són propis d'Oracle. Els primers proporcionen una manera transparent de modificar vistes que no podrien ser modificades directament mitjançant sentències UPDATE, INSERT i DELETE. Els últims poden executar-se sobre esdeveniments de sistema com són l'inici o el tancament de la base de dades i esdeveniments d'usuari com són logging in o logging off.

Sobre les restriccions associades a una taula, Oracle implementa la major part del que s'ha vist a l'estàndard SQL. El que Oracle no suporta és la possibilitat de crear dominis i assercions.

Les restriccions poden ser immediates o diferides. Les primeres actuen tan aviat com l'operació d'escriptura afecta a la columna amb restricció d'una taula. Es diferida si la restricció actua quan es completa la sentencia SQL que causa el canvi a la columna amb restricció.

4.2.4.2 Informix

L'implementació dels disparadors a Informix no és significativament diferent a la d'Oracle.

Com Oracle, Informix també proporciona un disparador INSTEAD OF per utilitzar amb les vistes. El que no hem vist a Informix és la possibilitat de crear disparadors sobre altres esquemes en la mateixa base de dades.

Les accions que es poden executar són les ja estudiades, INSERT, UPDATE, DELETE, EXECUTE PROCEDURE o EXECUTE FUNCTION. També es permeten accions escrites en SPL, C i Java.

Respecte les restriccions sobre taules, Informix tampoc implementa la creació de dominis ni assercions (CREATE DOMAIN i CREATE ASSERTION)

4.2.4.3 PostgreSQL

No hi han diferències significatives entre PostgreSQL i els altres sistemes analitzats. PostgreSQL implementa les restriccions d'integritat referencial de l'estàndard.

També permet crear disparadors (triggers), que junt amb les restriccions (tant a nivell de columna com de taula) serveixen per definir regles de negoci dins la base de dades. Els conceptes de DOMAINS i ASSERTIONS de l'estàndard no són suportats directament per PostgreSQL.

A diferència dels altres sistemes, PostgreSQL també permet la creació de regles (RULE), que són molt semblants als disparadors, amb la diferència que aquests es refereixen exclusivament a la taula sobre la que s'opera, mentre que les regles actuen sobre taules externes. D'altra banda el disparador realitza una acció addicional a l'acció que el provoca i la regla pot executar una acció en substitució de l'acció que l'activa (amb la clàusula INSTEAD).

5 Desenvolupament d'un cas pràctic

5.1 Objectiu

Als capítols anteriors hem identificat les principals característiques d'Orientació a l'Objecte dins l'estàndard SQL:1999, especialment pel que fa referència a funcions i regles. També hem vist com tres dels SGBD Objecte-Relacionals comercials més coneguts han implementat aquestes característiques.

En aquest capítol es pretén mostrar a través d'un cas pràctic en quina mesura dos dels SGBD analitzats, Oracle 9i i Postgresql 7.3, permeten dissenyar bases de dades utilitzant els principis de l'Orientació a l'Objecte.

En aquest cas es tracta de mostrar especialment les funcions com a mecanisme per modelar el comportament dels tipus definits per l'usuari i també com a forma d'aportar noves funcionalitats a la base de dades. Tractarem de mostrar les propietats d'encapsulament i polimorfisme. Altre aspecte que es tractarà de mostrar de forma pràctica és la creació de regles.

5.2 Descripció del cas

Dissenyarem una base de dades per emmagatzemar la informació d'un petit negoci de tendes de discos.

Es vol guardar la informació bàsica relativa a una **tenda**. Cada tenda té un identificador, una descripció, una adreça i un codi postal. El **codi postal** té tres elements d'informació: la ciutat, la província i el país.

De cada tenda ens interessa mantenir informació de totes les existències de discos que té, és a dir, el seu **inventari**. La informació que es manté d'un inventari és l'identificador de la tenda, l'identificador de cada disc en existència, la quantitat de cada producte i la data del darrer moviment de cada disc. Cada tenda té un sol inventari. Un disc pot estar en diferents inventaris, però cada inventari sols manté un registre per cada disc.

Els **discos** són els articles que ven el negoci i que constitueixen l'inventari de cada tenda. D'un disc ens interessa el seu identificador, l'editor, el format (que pot ser CD, DVD o MiniDisc) i el preu del article. Un disc es composa de una o més **cançons**, de les quals ens interessa: el número de cançó, l'autor de la cançó, el títol i la durada.

D'altra banda es vol guardar informació sobre les transaccions que realitza el negoci. Per això s'emmagatzema informació sobre els **clients**. La informació bàsica que interessa d'un client és l'identificador del client, l'adreça per enviar-li les factures i

altra informació, el telèfon i el tipus de client. Hi han dos tipus de clients, el **client habitual** que ha fet més d'una compra i les **empreses**. D'aquest darrer tipus interessa emmagatzemar el nom de l'empresa i el límit de crèdit i del client habitual interessa el nom i els cognoms. Segons el tipus de client es calculen els descomptes de manera diferent. El client habitual té un percentatge fix, mentre que l'empresa té un descompte segons el volum de compres.

Les transaccions es reflexen a través de les **comandes** que fan els clients. D'una comanda interessa el seu identificador, que és una seqüència generada automàticament pel sistema, la data, l'identificador del client, el seu status. Una comanda pot estar en status 'pendent' si no hi han suficients existències d'un producte demanat i si s'ha servit sencera tindrà l'status 'servida'. Un client pot tenir zero o més comandes, però una comanda només correspon a un client.

Les comandes es composen d'un o més **ítems**, cada ítem té la següent informació: identificador de la comanda, número d'ítem, identificador del disc i l'import total d'aquest ítem.

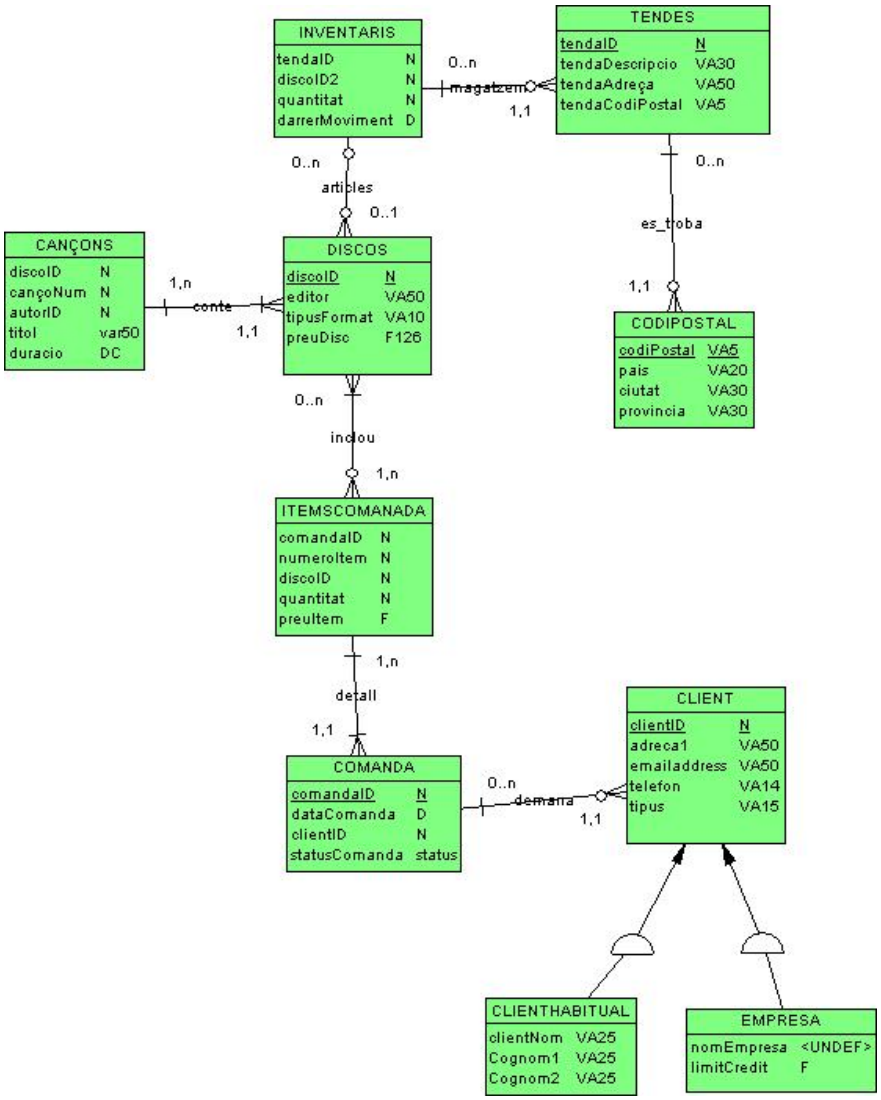
Algunes regles que caldria observar són:

- Els preus dels discos han de ser major o igual que 0. El valor per defecte es 0.
- D'un determinat disc no es pot servir en una comanda una quantitat major que la que hi ha en existències.
- La quantitat d'un disc en existències és 0 o superior. Per defecte és 0.
- Quan es crea un ítem d'una comanda automàticament s'ha de rebaixar l'inventari del disc en la quantitat servida i s'actualitza l'import total de la comanda.
- Si s'esborra una comanda s'han d'esborrar tots els ítems que la composen.
- El import total de la comanda no pot superar el límit de crèdit d'un client.

Algunes funcions que cal que el sistema proporcioni són:

- Una funció que permeti calcular la duració d'un disc.
- Un mètode que permeti calcular els descomptes de cada client.
- Un mètode que proporcioni una cadena amb dades bàsiques del client.
- Un procediment per actualitzar l'inventari d'un article.

Per fer el diagrama del model conceptual del cas hem utilitzat l'eina PowerDesigner de Sybase. El diagrama del model entitat-relació seria:



Es veu la jerarquia d'herència de l'entitat client, que s'especialitza en les entitats client habitual i empresa.

Sobre aquest model conceptual realitzarem modificacions per afegir algunes de les característiques d'orientació a l'objecte que permet l'estàndard SQL:1999. Després es veurà si aquestes característiques que permet l'estàndard poden ser implementades per Oracle 9i i per Postgresql 7.3.

En el desenvolupament del cas no explorarem totes les possibilitats d'orientació a l'objecte que permet l'estàndard SQL:1999; especialment pel que es refereix al sistema de tipus.

5.3 Implementar el model amb l'estàndard SQL:1999

Primer, implementarem el model conceptual segons l'estàndard SQL, com si fos per un SGDB completament basat en l'esmentat estàndard. Això ens permetrà fer una comparació amb la manera en que es pot implementar a Oracle i a PostgreSQL.

5.3.1 Definició de les entitats

Les entitats que utilitzarem al nostre disseny són:

- Tenda
- CodiPostal
- Inventari
- Cançó
- Disc
- Client
- ClientHabitual
- Empresa
- Comanda
- ItemComanda

Malgrat que en un model d'Orientació a l'Objecte pur definiríem les entitats com a UDTs, al nostre cas només ho farem amb algunes, la resta es definiran directament com a taules, al igual que en un model relacional.

CodiPostal

Crearem un UDT que ens permetrà definir columnes d'aquest tipus dins d'una taula.

```
CREATE TYPE CodiPostalType AS (
        codiPostalId        VARCHAR(5) NOT NULL,
        pais                VARCHAR(20) NOT NULL,
        ciutat              VARCHAR(30) NOT NULL,
        provincia           VARCHAR(30) NOT NULL)
NOT FINAL INSTANTIABLE;
```

Tenda

Creem la taula Tendes, que té com a clau primaria l'atribut tendaId.

```
CREATE TABLE Tendes (
        tendaId             NUMERIC NOT NULL,
        codiPostal          codiPostalType NOT NULL,  /* utilitzem el tipus creat abans */
        adreçaTenda          VARCHAR(25),
PRIMARY KEY (tendaId));
```

Disc

En aquest cas hem creat un nou tipus que ampliarà les possibilitats de generar especialitzacions, contemplant la possible evolució del negoci i que surtin nous tipus d'enregistraments.

```
CREATE TYPE discType AS (
        Editor          VARCHAR(50) NOT NULL,
        tipusFormat     formats NOT NULL,
        preuDisc        FLOAT NOT NULL DEFAULT 0 CHECK (preuDisc >= 0))
NOT FINAL
INSTANTIABLE
REF IS SYSTEM GENERATED;
```

Observem que encarreguem al sistema la generació d'un valor REF únic associat a cada instància de discType.

Hem establert les restriccions que els preus dels discos no poden ser negatius i que el valor per defecte és 0.

D'altra banda la columna tipusFormat fa referència a un domini que creem de la següent manera:

```
CREATE DOMAIN formats AS VARCHAR(10)
        DEFAULT 'CD'
        CHECK (VALUE IN ('CD', 'DVD', 'MD'));
```

Per emmagatzemar instàncies d'aquest tipus creem la següent taula tipificada:

```
CREATE TABLE discos OF discType (
        REF IS discId SYSTEM GENERATED);
```

Es crearà una columna discId que contindrà per cada fila el seu valor REF.

Cançó

Per aquesta entitat definim un taula.

```
CREATE TABLE cançons (
        discId          REF (discType) SCOPE discos
                        REFERENCES ARE CHECKED ON DELETE CASCADE,
        cançoNum        NUMERIC NOT NULL CHECK( cançoNum >0),
        autor           VARCHAR(50) NOT NULL,
        titol           VARCHAR(50) NOT NULL,
        durada          DECIMAL NOT NULL default 0 CHECK (durada >=0),
PRIMARY KEY (discId, cançoNum));
```

Fem referència al disc que conté la cançó mitjançant la columna discId definida com un valor REF que cal que estigui en la taula discos.

Inventari

En aquest cas creem una taula per recollir totes les existències que té una determinada tenda.

```
CREATE TABLE inventaris (
discoId          REF (discType) SCOPE discos
                 REFERENCES ARE CHECKED ON DELETE CASCADE,
tendaId          NUMERIC NOT NULL,
quantitat        NUMERIC NOT NULL CHECK (QUANTITAT >= 0),
darrerMoviment   DATE NOT NULL DEFAULT CURRENT_DATE,
PRIMARY KEY (discoId, tendaId),
FOREIGN KEY (tendaId) REFERENCES tendes (tendaId)
);
```

Veiem com hem evitat una clau forana, fent que la columna discId sigui del tipus REF, de forma que cada valor és una referència a la fila de discos on es troba el disc.

A la definició de la taula s'han definit com a regles que un article no pot tenir existències negatives i que per defecte la data del darrer moviment és la del dia en que es crea la fila.

Client

En aquest cas considerem apropiat crear un nou tipus, sobre el qual definirem una especialització per obtenir dos subtipus de client. Aquest plantejament ens permetrà comprovar les propietats de sobrecàrrega (overloading), redefinició (overriding) i polimorfisme de les rutines definides pe l'usuari que modelen el comportament dels tipus.

```
CREATE TYPE clientType AS (
        NIF             VARCHAR(10) NOT NULL,
        Adreça1 VARCHAR(50) NOT NULL,
        emailAddress    VARCHAR(50),
        telefon         VARCHAR(14))
NOT FINAL
METHOD toString() RETURNS VARCHAR(100)
METHOD descompte() RETURNS DECIMAL;
```

Aquest tipus declara dos mètodes: *toString* que proporciona una cadena amb el NIF del client i *descompte* que torna el descompte que s'aplica a un client. Podíem haver utilitzat un atribut per recollir el descompte associat a un client, però no s'ha fet perquè podria ser que segons el tipus de client hagin formes diferent de càlcul i també per provar la sobrecàrrega de mètodes.

A continuació creem la jerarquia de tipus:

```
CREATE TYPE ClientHabitual UNDER ClientType AS (
        clientNom       VARCHAR(25) NOT NULL,
        cognom1         VARCHAR(25) NOT NULL,
        cognom2         VARCHAR(25) NOT NULL)
```

```
NOT FINAL
OVERRIDING METHOD toString() RETURNS VARCHAR (100)



CREATE TYPE Empresa UNDER ClientType AS (
        nomEmpresa      VARCHAR (50) not null,
        limitCredit     FLOAT DEFAULT 500 CHECK (limitCredit >=0))
NOT FINAL
OVERRIDING METHOD toString() RETURNS VARCHAR(25)
METHOD descompte (pImport FLOAT) RETURNS DECIMAL;
```

En la definició dels subtipus hem declarat que hi ha una redefinició del mètode toString i hem sobrecarregat el mètode descompte.

A continuació creem les següents taules tipificades per emmagatzemar les instàncies dels tipus creats.

```
CREATE TABLE clients OF ClientType (
        REF IS clientID SYSTEM GENERATED);

CREATE TABLE clientsHabituals OF ClientHabitual UNDER Clients

CREATE TABLE empreses OF Empresa UNDER Clients;
```

Segons la definició anterior s'ha creat una jerarquia de taules que es correspon amb la jerarquia del tipus client.

Comanda

Creem un taula per emmagatzemar totes les comandes fetes pels clients. Cada comanda tindrà un identificador únic. Cada comanda només pot ser d'un sol client. En lloc d'una restricció de clau forana amb la taula clients, definim una columna amb el tipus de dades REF que referència files de la taula clients.

```
CREATE TABLE comandes (
        comandaId       NUMERIC NOT NULL UNIQUE,
        clientId        REF (ClientType) SCOPE Clients
                        REFERENCES ARE CHECKED ON DELETE CASCADE,
        dataComanda     DATE NOT NULL DEFAULT CURRENT_DATE,
        statusComanda    statusType NOT NULL,
        import          FLOAT NOT NULL DEFAULT 0 CHECK (import >=0),
PRIMARY KEY (comandaId));
```

La columna statusComanda pertany al domini statusType definit de la següent manera:

```
CREATE DOMAIN statusType AS VARCHAR(25)
        DEFAULT 'pendent'
        CHECK (VALUE IN ('pendent', 'servida'));
```

ItemsComanda

Una comanda pot contenir diferents articles (ítems), cadascun d'ells és una línia de la comanda. També definim una taula per emmagatzemar les totes les línies de totes les comandes. Cada línia s'identifica plenament amb l'identificador de comanda i el número de línia.

```
CREATE TABLE ItemsComanda (
        comandaId       NUMERIC NOT NULL;
        numeroItem      INTEGER NOT NULL CHECK (numeroItem >=0),
        discId          REF (discType) SCOPE discos
                        REFERENCES ARE CHECKED ON DELETE CASCADE,
        quantitatINTEGER NOT NULL DEFAULT 0;
        preuItem        FLOAT NOT NULL DEFAULT 0,
PRIMARY KEY (comandaId, numeroItem)
FOREIGN KEY (comandaId) REFERENCES Comandes (comandaId) ON DELETE CASCADE);
```

5.3.2 Funcions i restriccions

L'estàndard permet tres tipus de rutines definides per l'usuari: mètodes, funcions i procediments.

Mètodes

Hem definit els mètodes toString i descompte(), lligats als tipus clientType. Dins de la definició dels tipus s'han declarat els mètodes. La definició d'aquests mètodes és com segueix:

```
CREATE INSTANCE METHOD toString()
        RETURNS VARCHAR(100)
        FOR ClientType
        RETURN NIF

CREATE INSTANCE METHOD toString()
        RETURNS VARCHAR(100)
        FOR ClientHabitual
        RETURN NIF || ' ' || clientNom || ' ' || cognom1 || ' ' || cognom2

CREATE INSTANCE METHOD toString()
        RETURNS VARCHAR(100)
        FOR Empresa
        RETURN NIF || ' ' || nomEmpresa
```

L'operador || concatena dues cadenes de caràcters. El mètode toString definit pel supertipus ClientTypus s'ha redefinit als subtipus per mostrar una informació diferent.

```
CREATE INSTANCE METHOD descompte()
        RETURNS DECIMAL
        FOR ClientType
        RETURN 0.03

CREATE INSTANCE METHOD descompte(pImport FLOAT)
        RETURNS DECIMAL
        FOR empresa
        IF  pImport > 1000
                THEN RETURN 0.05
```

```
        ELSE RETURN 0.03
        END IF;
```

Pel subtipus empresa hem creat altre mètode descompte amb un paràmetre addicional, la quantitat comprada, de manera que torna un percentatge diferent segons aquesta quantitat.

Funcions

Les funcions a diferència dels mètodes no cal que siguin declarades dintre de l'esquema d'un UDT. En aquest cas hem definit una funció que ben bé podria ser un mètode del tipus discType, però amb idea d'exemplificar l'hem definit fora del seu esquema.

```
CREATE FUNCTION duradaDisc (pDisc DiscType)
        RETURNS DECIMAL
        RETURN ( SELECT SUM(c.durada)
                        FROM cançons c
                        WHERE c.discId = pDisc.discID
                        GROUP BY c.discID);
```

Procediments

Hem definit dos procediments que ens serviran per definir les restriccions a través dels disparadors. Al cas hi haurà un disparador que funcionarà quan es vulgui actualitzar la taula itemsComanda.

El procediment que actualitza el import total d'una comanda és:

```
CREATE PROCEDURE actualitzaComanda (pImport INTEGER, PComandaId INTEGER)
        BEGIN
                UPDATE Comandes
                SET import = import + pImport
                WHERE comandaId = pComandaId;
        END;
END PROCEDURE;
```

El procediment per actualitzar l'inventari és:

```
CREATE PROCEDURE actualitzaInventari (pDiscId INTEGER, pQuantitat NUMERIC)
        DECLARE Stock INTEGER;
        BEGIN
                SELECT quantitat INTO stock
                FROM inventaris WHERE discId = pDiscId;
                IF stock < pQuantitat THEN
                        UPDATE Inventaris
                        SET quantitat = quantitat + pQuantitat
                        WHERE discId = pDiscId;
                END IF
        END;
END PROCEDURE;
```

5.4 Implementar el model en Oracle

La creació de les entitats no ha estat molt diferent respecte del model basat completament en l'estàndard SQL que hem definit al punt anterior.

Les principals diferències en implementar-lo en Oracle han estat:

- No hem utilitzat columnes de tipus REF en les taules cançons, inventaris, comandes i itemsComanda. La raó és purament pràctica, ja que Oracle permet definir columnes d'aquest tipus i fer referències a files d'una taula objecte. S'ha fet perquè una columna d'aquest tipus no es pot utilitzar com part d'una clau primària i que això dificulta el seu ús dins de la definició de funcions i procediments (només es pot utilitzar la funció DEREF dins d'una consulta i no dins d'un bloc de PL/SQL).
- Oracle no admet la definició de dominis i assercions, per tant hem implementat els dominis que afecten a les columnes formatDisc de discos i statusComanda de comandes com a restriccions de columna.
- Oracle permet també definir una jerarquia de vistes objectes (equivalent a una jerarquia de taules tipificades), però a efectes de l'objectiu d'aquest exemple, que és veure el funcionament de funcions i regles, no s'ha considerat necessària la seva implementació.

Mostrem el codi SQL que ens ha permès crear les entitats del model:

Tipus codi postal	Taula tendes
CREATE TYPE CodiPostalType AS OBJECT(codiPostalId VARCHAR2(5), pais VARCHAR2(20), ciutat VARCHAR2(30), provincia VARCHAR2(30)) NOT FINAL INSTANTIABLE;	CREATE TABLE Tendes (tendaId NUMERIC NOT NULL, codiPostal codiPostalType NOT NULL CHECK (codiPostal.codiPostalId IS NOT NULL), CHECK (codiPostal.pais IS NOT NULL), CHECK (codiPostal.ciutat IS NOT NULL), CHECK (codiPostal.provincia IS NOT NULL), adreçaTenda VARCHAR2(25) NOT NULL, PRIMARY KEY (tendaId));
Tipus disc	**Taula objecte discos**
CREATE TYPE discType AS OBJECT(discId INTEGER, Editor VARCHAR2(50), tipusFormat VARCHAR2(5), preuDisc FLOAT) NOT FINAL INSTANTIABLE;	CREATE TABLE discos OF discType (Editor NOT NULL, tipusFormat NOT NULL CHECK tipusFormat IN ('CD', 'DVD', 'MD')), preuDisc NOT NULL, CONSTRAINT preuDisc_min CHECK (preuDisc >=0), PRIMARY KEY (discId)) OBJECT IDENTIFIER IS PRIMARY KEY;

Destaquem com al definir la taula **tendes** establim les restriccions dels atributs del tipus CodiPostalType.

Taula cançons	Taula inventaris
CREATE TABLE cançons (discId INTEGER NOT NULL, cançoNum NUMERIC NOT NULL CHECK(cançoNum >0), autor VARCHAR2(50) NOT NULL, titol VARCHAR2(50) NOT NULL, durada DECIMAL NOT NULL CHECK (durada >=0), PRIMARY KEY (discId, cançoNum), FOREIGN KEY (discId) REFERENCES discos (discId));	CREATE TABLE inventaris (discId INTEGER NOT NULL, tendaId NUMERIC NOT NULL, quantitat NUMERIC NOT NULL CHECK (QUANTITAT >= 0), darrerMoviment DATE DEFAULT SYSDATE NOT NULL, PRIMARY KEY (discId, tendaId), FOREIGN KEY (discId) REFERENCES discos (discId), FOREIGN KEY (tendaId) REFERENCES tendes (tendaId));
Supertipus Client	**Subtipus ClientHabitual**
CREATE TYPE clientType AS OBJECT(clientId NUMBER, NIF VARCHAR2(10), Adreça1 VARCHAR2(50), emailAddress VARCHAR2(50), telefon VARCHAR2(14), MEMBER FUNCTION descompte RETURN NUMBER, MEMBER FUNCTION toString RETURN VARCHAR2) NOT FINAL;	CREATE TYPE ClientHabitual UNDER ClientType (clientNom VARCHAR2(25), cognom1 VARCHAR2(25), cognom2 VARCHAR2(25), OVERRIDING MEMBER FUNCTION toString RETURN VARCHAR2)
Subtipus Empresa	**Taula Objecte Clients**
CREATE TYPE Empresa UNDER ClientType (nomEmpresa VARCHAR2 (50), limitCredit FLOAT, OVERRIDING MEMBER FUNCTION toString RETURN VARCHAR2, MEMBER FUNCTION descompte (pImport FLOAT) RETURN NUMBER);	CREATE TABLE clients OF ClientType (clientId PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY KEY;
Taula Objecte ClientsHabituals	**Taula Objecte Empreses**
CREATE TABLE clientsHabituals OF ClientHabitual (clientNom NOT NULL, cognom1 NOT NULL, cognom2 NOT NULL, clientId PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY KEY;	CREATE TABLE empreses OF Empresa (nomEmpresa NOT NULL, limitCredit DEFAULT 500 CHECK (limitCredit >=0), clientId PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY KEY;

Assenyalem que quan es defineixen els tipus de clients, s'inclou la declaració dels mètodes que tenen associats. En el cas dels subtipus, s'indica amb la clàusula OVERRIDING que es redefineix el mètode heretat (*toString*) del supertipus **ClientType**. També es pot veure com el subtipus **Empresa** sobrecarrega el mètode *descompte* afegint un altre argument.

També remarquem que les restriccions sobre els atributs dels tipus es defineixen en les columnes corresponents de la taula objecte basada en el tipus.

Taula comandes	Taula itemsComanda
CREATE TABLE comandes (comandaId NUMERIC NOT NULL, clientId INTEGER NOT NULL, dataComanda DATE DEFAULT sysdate NOT NULL, statusComanda VARCHAR2(25) DEFAULT 'pendent' CHECK (statusComanda IN ('pendent', 'servida')), import FLOAT DEFAULT 0 NOT NULL CHECK (import >=0), PRIMARY KEY (comandaId));	CREATE TABLE ItemsComanda (comandaId INTEGER NOT NULL, numeroItem INTEGER NOT NULL CHECK (numeroItem >=0), discId INTEGER NOT NULL, quantitat INTEGER DEFAULT 0 NOT NULL, preuItem FLOAT DEFAULT 0 NOT NULL, PRIMARY KEY (comandaId, numeroItem), FOREIGN KEY (comandaId) REFERENCES Comandes (comandaId), FOREIGN KEY (discId) REFERENCES discos (discId));

A continuació s'han definit els mètodes que s'havien declarat dins del tipus client i dels seus subtipus.

```
CREATE OR REPLACE TYPE BODY ClientType AS
MEMBER FUNCTION toString RETURN VARCHAR2 AS
resultat VARCHAR2(10);
BEGIN
resultat := SELF.NIF;
RETURN (resultat);
END;
 MEMBER FUNCTION descompte RETURN NUMBER AS
        BEGIN
                RETURN 0.03;
        END;
END;

CREATE OR REPLACE TYPE BODY ClientHabitual AS
 OVERRIDING MEMBER FUNCTION toString RETURN VARCHAR2 AS
        BEGIN
                RETURN SELF.NIF || SELF.clientNom || SELF.cognom1 || SELF.cognom2;
        END;
END;

CREATE OR REPLACE TYPE BODY Empresa AS
 overriding MEMBER FUNCTION toString RETURN VARCHAR2 AS
 BEGIN
 RETURN SELF.NIF || ' ' || SELF.nomEmpresa || ' ' || SELF.limitCredit || ' ' || SELF.emailaddress;
 END;
 MEMBER FUNCTION descompte(pImport FLOAT) RETURN number AS
 BEGIN
        IF  pImport > 1000 THEN
                RETURN 0.05;
        ELSE
                RETURN 0.03;
        END IF;
```

```
 END;
END;
```

Veiem com el mètode toString no té declarat cap argument de manera explícita, però el sistema implícitament afegeix el argument SELF, que és la instància de l'objecte que invoca el mètode.

S'implementen tot seguit les funcions i procediments que havíem definit al nostre model.

```
CREATE FUNCTION duradaDisc (pDisc INTEGER) RETURN DECIMAL IS
        resultat DECIMAL;
        BEGIN
        SELECT SUM(c.durada) INTO resultat
                        FROM cançons c

                        WHERE c.discId = pDisc
                        GROUP BY c.discID;
                RETURN (resultat);
        END;

CREATE FUNCTION stocDisc (pDisc INTEGER)
        RETURN NUMBER
        IS resultat NUMBER;
        BEGIN
                SELECT quantitat INTO resultat
                FROM inventaris
                WHERE discId = pDisc;
                RETURN (resultat);
        END;

CREATE PROCEDURE actualitzaComanda (pImport IN FLOAT, PComandaId IN INTEGER) AS
        BEGIN
                UPDATE Comandes
                SET import = import + pImport
                WHERE comandaId = pComandaId;
        END;

CREATE PROCEDURE actualitzaInventari (pDiscId IN INTEGER, pQuantitat IN NUMERIC) AS
STOCK number;
BEGIN
        SELECT quantitat INTO stock
        FROM inventaris
        WHERE discId = pDiscId;
                IF stock < pQuantitat THEN
                        Raise_application_error(-20101, 'Inventari insuficient per comanda');
                ELSE
                        UPDATE Inventaris
                        SET quantitat = quantitat - pQuantitat
                        WHERE discId = pDiscId;
                END IF;
END actualitzaInventari;
```

Creem un disparador que ens permetrà actualitzar l'import d'una comanda i l'inventari cada vegada que s'insereix una línia de comanda nova. El disparador fa ús dels procediments creats.

```
CREATE TRIGGER insercio_Item
AFTER INSERT OR UPDATE OF preuItem, quantitat ON ItemsComanda
FOR EACH ROW
BEGIN
        actualitzaComanda(:new.preuItem, :new.comandaId);
        actualitzaInventari(:new.discId, :new.quantitat);
END;
```

5.4.1 Funcionament del model implementat en Oracle

Un cop hem definit el model, s'han introduït un conjunt de dades per comprovar el funcionament de les funcions i restriccions establertes.

Les dades bàsiques introduïdes són:

```
INSERT INTO tendes VALUES (1, codiPostalType('08013', 'Espanya', 'Barcelona', 'Barcelona'), 'Lepanto 145');
```

Es veu com per introduir un valor a la columna CodiPostal de la taula **tendes** es fa servir el mètode constructor que automàticament és proveït pel sistema en crear un nou tipus.

```
INSERT INTO discos VALUES (1, 'Virgin', 'CD',10);
INSERT INTO discos VALUES (2, 'Emy', 'CD',14);
```

```
INSERT INTO cançons VALUES (1, 1, 'Culture Club', 'Do you really want to hurt me?', 4.23);
INSERT INTO cançons VALUES (1, 2, 'Culture Club', 'White Boy', 4.40);
INSERT INTO cançons VALUES (1, 3, 'Culture Club', 'Church of the poison mind', 3.32);
INSERT INTO cançons VALUES (2, 1, 'Carlos Embade', 'Unión de Reyes', 3.21);
INSERT INTO cançons VALUES (2, 2, 'AfroCuba', 'El Cangrejo', 8.18);
INSERT INTO cançons VALUES (2, 3, 'Puntillita', 'El guaguancó de Marta', 2.43);
```

```
INSERT INTO inventaris (discId, tendaId, quantitat) VALUES (1,1,15);
INSERT INTO inventaris (discId, tendaId, quantitat) VALUES (2,1,2);
```

Al inserir les dades a inventaris no s'ha posat cap data. Si es fa una consulta, es comprovarà que el sistema ha posat per defecte la data del moment de la inserció (SYSDATE).

```
INSERT INTO ClientsHabituals VALUES (1,'35082597F', 'Lepanto 147',NULL, '932461914', 'Manuel', 'Gonzalez', 'Luque');
```

```
INSERT INTO empreses VALUES (2,'40256180G', 'Aribau 47','disc@myHost.es', '932226914', 'Discos castello', 2000);
```

```
INSERT INTO comandes (comandaId, clientId) VALUES (1, 1);
```

Al inserir una comanda no s'ha posat la data, l'status i l'import; el sistema per defecte assigna aquests valors (SYSDATE, 'pendent' i 0 respectivament). L'import s'anirà

modificant a mesura que es creen noves línies de comanda (items de comanda), gràcies al disparador creat. Així, si s'introdueix la següent línia de la comanda 1:

```
INSERT INTO itemsComanda VALUES (1,1,1,2,20);
```

Abans d'inserir la línia fem la consulta següent:

```
SQL> select * from comandes;

 COMANDAID   CLIENTID DATACOMA STATUSCOMANDA                    IMPORT
---------- ---------- -------- ------------------------- ----------------------------------------------------------
         1          1 16/05/04 pendent                                 0
```

Després de la inserció es repeteix la consulta:

```
SQL> select * from comandes;

 COMANDAID   CLIENTID DATACOMA STATUSCOMANDA                    IMPORT
---------- ---------- -------- ------------------------- ----------------------------------------------------------
         1          1 16/05/04 pendent                                20
```

Per altra banda el mateix disparador que actua quan s'insereix la línia de comanda , ha actualitzat l'inventari. Es recorda que del disc 1 hi havien 15 unitats. Si es fa una consulta a inventaris, es comprovarà que s'ha rebaixat en la quantitat indicada a la línia de comanda nova (2 unitats).

```
SQL> SELECT * FROM inventaris WHERE discId = 1;

    DISCID    TENDAID  QUANTITAT DARRERMO
---------- ---------- ---------- ----------------------------------
         1          1         13 16/05/04
```

El disparador també donaria error si es vol crear una línia de comanda amb una quantitat d'un disc més gran que la que queda en inventari. Per exemple, si del disc 1 queden 13 unitats i es crea la següent línia:

```
INSERT INTO itemsComanda VALUES (1,2,1,15,100);
```

El sistema dona el resultat:

```
SQL> INSERT INTO itemsComanda VALUES (1,2,1,15,100);
INSERT INTO itemsComanda VALUES (1,2,1,15,100)
            *
ERROR en línea 1:
ORA-20101: Inventari insuficient per comanda
```

Es pot comprovar que les restriccions de domini, que han sigut implementades com a restriccions de columna funcionen correctament. Si s'insereix un disc amb un format 'LP' obtindrem la següent resposta:

```
SQL> INSERT INTO discos VALUES (4,'DRO','LP',12);
INSERT INTO discos VALUES (4,'DRO','LP',12)
*
ERROR en línea 1:
```

```
ORA-02290: restricción de control (SYSTEM.SYS_C002726) violada
```

Es pot obtenir la durada d'un disc determinat utilitzant la funció duradaDisc. Per exemple si es vol saber la durada total del disc amb identificador 1, el sistema ens dona:

```
SQL> select duradaDisc(discId)
  2  FROM discos WHERE discId = 1;

DURADADISC(DISCID)
------------------
                11
```

Sobre els mètodes, cal comprovar que es donen les característiques de sobrecàrrega i redefinició pròpies d'un model d'Orientació a l'Objecte.

Comprovem la redefinició (overriding) cridant al mètode toString del client tipus ClientHabitual inserit. S'obté com resultat:

```
SQL> SELECT VALUE(p).toString() FROM clientsHabituals p;

VALUE(P).TOSTRING()
-------------------------------------------------------------
35082597F Manuel Gonzalez Luque
```

Si es crida el mateix mètode d'un objecte del tipus empresa s'obtindrà el següent resultat:

```
SQL> SELECT VALUE(p).toString() FROM empreses p;

VALUE(P).TOSTRING()
-----------------------------------------------------------------------
40256180G Discos castello 2000 disc@myHost.es
```

Podem comprovar la sobrecàrrega del mètode descompte. Al tipus empresa hem definit dos mètodes amb el mateix nom, descompte () heretat del supertipus i descompte(pImport FLOAT) definit al subtipus. Si es crida als dos mètodes els resultats són:

```
SQL> SELECT VALUE(P).descompte() FROM empreses p;

VALUE(P).DESCOMPTE()
--------------------
                ,03

SQL> SELECT VALUE(p).descompte(10000) FROM empreses p;

VALUE(P).DESCOMPTE(10000)
-------------------------
                   ,05
```

5.5 Implementar el model en PostgreSQL

PostgreSQL permet definir nous tipus de dades que després poden ser utilitzats de la mateixa manera que els tipus predefinits que proporciona el sistema. Però, donada la necessitat de codificar totes les rutines en un llenguatge de programació (C o C++), aquesta facilitat és més pròpia dels desenvolupadors del sistema que dels usuaris finals. Per aquest motiu, al implementar el model en aquest SGBD hem obviat la creació de nous tipus, el que fa que la implementació del model en PostgreSQL sigui molt similar a la d'un model relacional.

Per altra banda no existeixen les taules tipificades, malgrat que si que es pot crear una jerarquia de taules normals (amb la clàusula INHERITS).

Per l'absència de tipus definits per l'usuari (en la manera que ho permet Oracle), no es poden definir mètodes, funcions o procediments associats als mateixos que ens permeti modelar el seu comportament o encapsular les seves dades.

Pel que fa a les restriccions i disparadors, PostgreSQL ha implementat la major part de les funcionalitats especificades a l'estàndard SQL, i en aquest apartat no trobem diferències significatives con respecte a la implementació feta en Oracle.

No s'adjuntarà el codi SQL que s'ha fet servir per crear les taules a PostgreSQL ja que no hi han diferències significatives pel que fa a la implementació de les restriccions de taules i de columnes. Tornem a recordar que PostgreSQL no suporta la creació de dominis i assercions, per tant les restriccions s'han definit de manera idèntica a com s'ha fet a Oracle.

Val la pena però, exposar el codi relatiu a la jerarquia de taules de clients. El codi és el següent:

```
CREATE TABLE clients (
        clientId        INTEGER NOT NULL,
        NIF             VARCHAR(10),
        Adreça1         VARCHAR(50),
        emailAddress    VARCHAR(50),
        telefon         VARCHAR(14)
        PRIMARY KEY (clientId));

CREATE TABLE ClientsHabituals (
        clientNom       VARCHAR(25),
        cognom1         VARCHAR(25),

        cognom2         VARCHAR(25))
        INHERITS (clients);

CREATE TABLE empreses (
        nomEmpresa      VARCHAR(50),
        limitCredit     FLOAT DEFAULT 0 NOT NULL CHECK (limitCredit >=0))
        INHERITS (clients);
```

Com no hi han tipus definits per l'usuari, tampoc s'ha pogut declarar ni definir els mètodes toString() i descompte(), que hem fet servir per analitzar les propietats de sobrecàrrega i redefinició. No obstant això, cal dir que PostgreSQL permet la sobrecàrrega de funcions i procediments.

S'ha implementat la funció duradaDisc() per exemplificar la possibilitat de crear funcions definides per l'usuari. El codi és el següent:

```
CREATE FUNCTION duradaDisc (pDisc INTEGER) RETURNS DECIMAL
AS '
        DECLARE resultat DECIMAL;
        BEGIN
    SELECT SUM(c.durada) INTO resultat
                        FROM can?ons c
                        WHERE c.discId = $1
                        GROUP BY c.discID;
                RETURN (resultat);
        END;
' LANGUAGE plpgsql;
```

Si es vol saber la durada del disc número 1, obtenim el següent resultat:

```
tfc=# select duradaDisc(1);
 duradadisc
---------------
         11.95
(1 row)
```

Hem implementat també el disparador que s'encarrega d'actualitzar l'inventari i la comanda corresponent quan s'insereix o s'actualitza la taula itemsComanda. A diferència d'Oracle o de l'estàndard on era possible utilitzar qualsevol procediment emmagatzemat en la base de dades, en PostgreSQL cal definir un procediment que es lliga amb el disparador. Una funció de disparador és una funció que no té arguments i que torna un tipus especial de dades (OPAQUE).

A diferència de l'estàndard un disparador pot tenir associat més d'un esdeveniment d'activació, com per exemple inserir i actualitzar (INSERT OR UPDATE).

En lloc de definir els procediments actualitzaInventari i actualitzaComanda, s'ha definit la funció de disparador següent:

```
CREATE FUNCTION trig_insert_update_item () RETURNS opaque AS
' DECLARE stock INTEGER;
 BEGIN
        SELECT quantitat INTO stock
        FROM inventaris
        WHERE discId = NEW.discId;
        IF stock < NEW.quantitat THEN
         RAISE EXCEPTION "Inventari insuficient per comanda";
ELSE
```

```
        UPDATE Inventaris
        SET quantitat = quantitat - NEW.quantitat
        WHERE discId = NEW.discId;
END IF;
        UPDATE Comandes
                SET import = import + NEW.preuItem
                WHERE comandaId = NEW.comandaId;
        RETURN NULL;
END;
' LANGUAGE plpgsql;
```

I el codi del disparador al que s'associa la funció és:

```
CREATE TRIGGER insercio_Item
AFTER INSERT OR UPDATE ON itemsComanda
FOR EACH ROW EXECUTE PROCEDURE trig_insert_update_item ();
```

Realitzem la mateixa prova que hem fet a l'implementació en Oracle. Afegim una nova línia de comanda, comprovant com s'actualitza l'inventari i la comanda que correspon a la nova línia.

```
tfc=# select * from inventaris;
 discid | tendaid | quantitat | darrermoviment
--------+---------+-----------+----------------
      1 |       1 |        15 | 2004-05-18
      2 |       1 |         2 | 2004-05-18
      3 |       1 |         1 | 2004-05-18
(3 rows)

tfc=# select * from comandes where comandaId = 1;
 comandaid | clientid | datacomanda | statuscomanda | import
-----------+----------+-------------+---------------+--------
         1 |        1 | 2004-05-18  | pendent       |      0
(1 row)
```

inserim una línia de comanda

```
tfc=# INSERT INTO itemsComanda VALUES (1,1,1,2,20);
INSERT 17179 1

Comprovem l'estat en inventari

tfc=# select * from inventaris;
 discid | tendaid | quantitat | darrermoviment
--------+---------+-----------+----------------
      1 |       1 |        13 | 2004-05-18
      2 |       1 |         2 | 2004-05-18
      3 |       1 |         1 | 2004-05-18
(3 rows)

tfc=# select * from comandes where comandaId = 1;
 comandaid | clientid | datacomanda | statuscomanda | import
-----------+----------+-------------+---------------+--------
         1 |        1 | 2004-05-18  | pendent       |     20
(1 row)
```

5.6 Comparació de la implementació en els dos sistemes

Exposarem en una taula els aspectes més significatius de la implementació del cas pràctic en els dos sistemes.

Oracle 9i	PostgreSQL
Sistema de tipus	
No era objectiu del treball estudiar el sistema de tipus dels productes analitzats. Només es fa referència als aspectes d'aquest sistema de tipus que ens interessen, com són els tipus definits per l'usuari. Oracle ens ha permet definir fàcilment els tipus que hem considerat necessaris. Ha estat possible establir una jerarquia de tipus, de forma que s'ha comprovat el mecanisme d'herència.	PostgreSQL no facilita a l'usuari, com fa Oracle, la possibilitat de definir nous tipus. Per tant al nostre exemple no hem implementat nous tipus i la implementació de les entitats ha sigut igual que qualsevol model relacional. El que si ens ha permès PostgreSQL és definir una jerarquia de taules.
Funcions	
S'han definit mètodes associats als nous tipus creats. Això ens ha permet comprovar el mecanisme d'herència en relació a les funcions. S'ha vist el funcionament de la redefinició de mètodes dins d'una jerarquia de tipus i la sobrecàrrega de funcions. Si bé no hem implementa la encapsulació, aquesta és perfectament possible. A diferència de l'estàndard, el sistema no proveeix de manera automàtica dels mètodes *observer* i *mutator*, però si el mètode constructor. També s'ha vist la possibilitat de definir funcions associades a altres esquemes (la base de dades en el nostre cas).	Donat que no s'han creat nous tipus, només s'ha comprovat al possibilitat de crear funcions dins de la base de dades i com es admesa la sobrecàrrega.

Oracle 9i	PostgreSQL
Regles (restriccions i disparadors)	
Hem implementat moltes de les restriccions que forcen la integritat referencial de les dades, així com s'han definit restriccions que defineixen regles de negoci. El mecanisme de definició de disparadors és molt semblant al definit per l'estàndard.	Aquí les diferències amb Oracle són molt petites. La definició de regles de PostgreSQL s'ajusta bé a l'estàndard. Hem comprovat que la definició dels disparadors és una mica diferent a la d'Oracle i potser no és tan fàcil.

6 Conclusions

Dins del món del desenvolupament del software hi ha un grau important de reconeixement dels avantatges que pot aportar la tecnologia d'Orientació a l'Objecte. Moltes aplicacions necessiten mecanismes per tractar amb nous tipus complexes i per treballar en entorns distribuïts de manera més eficient i segura. El món dels sistemes de gestió de bases de dades no pot quedar al marge de les necessitats de las noves aplicacions i noves maneres de treballar (entorns client/servidor, concurrència, processos distribuïts, etc.).

En aquest treball hem estudiat com s'han format grups de treball per tal d'orientar als fabricants de bases de dades sobre quines característiques han d'incorporar els seus productes per afrontar aquestes noves necessitats. Concretament hem analitzat els principis orientadors definits pel Third Generation Database System Manifesto.

Hem comprovat que l'estàndard SQL:1999 ha tingut en compte una part important de les recomanacions fetes a l'article esmentat. Pel que respecta a les funcions i possibilitat de establir regles, l'estàndard presenta un elevat grau de compliment de les propostes.

SQL:1999 permet als usuaris definir funcions, procediments i mètodes. L'encapsulació és una característica que es pot assolir gràcies a la possibilitat de crear rutines per part de l'usuari. Hem comprovat que aquestes funcions poden ser heretades i que la redefinició i sobrecàrrega són possibles.

Altre dels aspectes de l'estàndard que hem analitzat, ha estat la possibilitat de definir regles. Hem comprovat que SQL:1999 disposa dels mecanismes suficients com per complir sense problemes la proposició 1.5 de l'article esmentat abans.

En l'anàlisi comparatiu entre els tres productes, s'ha vist que Oracle i Informix, incorporen totes les funcionalitats que s'han estudiat a l'estàndard SQL:1999. Mentre que PostgreSQL, encara no disposa d'algunes de les característiques avançades definides a l'estàndard, com és la creació de nous tipus i la possibilitat de definir funcions que modelin el comportament dels tipus. Cal destacar que Oracle és el que més fàcilment permet aplicar les noves funcionalitats d'Orientació a l'Objecte, des de el punt de vista de l'usuari.

El desenvolupament del cas pràctic en el sistema Oracle i en PostgreSQL, ha posat de manifest el que s'ha dit anteriorment. Assenyalem però, que no hi ha diferències importants en quant a les possibilitats d'utilitzar restriccions i disparadors per crear regles tant en un sistema com en altra, i que els dos s'adapten bastant bé a l'estàndard.

7 Futures línies de treball

El món de les tecnologies de la informació evoluciona a una gran velocitat. La aparició de la Web ha accelerat molt els canvis, ja que ha modificat també les maneres d'enfocar els problemes i les solucions.

Les bases de dades també han d'evolucionar amb l'entorn. SQL:1999 ha estat un pas important per l'adaptació dels sistemes basats en el model relacional. Però encara hi ha noves funcionalitats per afegir. Moltes d'aquestes són les que formen part del proper estàndard SQL:2003.

Alguns sistemes comercials ja han incorporat alguns dels canvis que figuraran en el nou estàndard, mentre que d'altres encara han d'implementar algunes de les funcionalitats que ja trobem al actual estàndard. Per tant l'evolució pel que fa als productors de bases de dades no té el mateix ritme.

Em sembla clar que les noves funcionalitats que contempla l'estàndard SQL:2003 són un bon material de estudi per futures línies de treball. Des de una òptica semblant a la del nostre treball, es pot comprovar en quina mesura el nou estàndard s'adapta a les proposicions del Manifest de Bases de Dades de Tercera Generació, afegint aquelles característiques que encara no s'havien incorporat.

8 Agraïments

A la meva família, sense el seu ajut i la seva inacabable paciència mai hagués pogut començar i finalitzar aquests estudis.

Al Josep Navarro, consultor d'aquest treball, que m'ha ajudat sempre que ho he necessitat i m'ha donat els ànims suficients per no quedar-me en el camí.

A la UOC, per fer possible que gent com jo pugui trobar una manera de realitzar-se sense perdre treball i família en l'intent.

9 Glossari

ANSI:	American National Standards Institute.
BD:	Base de Dades. Conjunt estructurat de dades que representa entitats i les seves interrelacions.
Clau primària:	Atribut o conjunt d'atributs escollits per identificar les files d'una taula.
DBMS:	Database Management System. Acrònim angles de SGDB.
Disparador:	Procediment emmagatzemat a la base de dades que s'executa automàticament quan es du a terme una operació d'actualització sobre una taula concreta.
Esquema:	Es la descripció o definició de la base de dades o dels seus elements (taules, tipus, etc.).
ISO:	International Standards Organization.
Model d'Orientació a l'Objecte:	Model de dades basat en la tecnologia d'orientació a l'objecte. El seu component fonamental es l'objecte.
Model de Dades:	Conjunt de components i d'eines conceptuals que una determinada tecnologia per a modelitzar una realitat.
Model Objecte-Relacional:	Ampliació del model relacional que afegeix la possibilitat que els tipus de dades siguin tipus abstractes de dades definits per l'usuari.
Model Relacional:	Model de dades que utilitza com a element fonamental per modelitzar les relacions (taules al món dels SGBD).
Objecte:	Paquet de software que conté una col·lecció de procediments i dades relacionats
Orientació a l'Objecte:	Tecnologia que es basa en objectes que proporcionen encapsulació de procediments i dades, missatges que suporten polimorfisme entre els objectes i classes que implementen herència dintre de jerarquies de classes.

ROW:	Fila o registre d'una base de dades. Es un tipus proporcionat per l'estàndard SQL:1999.
SGBD:	Sistema de Gestió de Bases de Dades. Programari especialitzat que gestiona i controla una base de dades.
SGBDOO	SGBD d'Orientació a l'Objecte. En angles OODBMS.
SGBDOR	SGBD Objecte-Relacional. En angles ORDBMS.
SGBDR	SGBD Relacional. En angles RDBMS.
SQL	Structured Query Language. Llenguatge estructurat de consultes. Permet crear, actualitzar i consultar la base de dades.
Transacció:	Conjunt d'operacions simples que s'executen com una unitat.
UDT:	User-Defined Types. Tipus definits per l'usuari.

10 Bibliografia

Llibres

Melton, J.; Eisenberg, A. *Understanding SQL and Java Together*; Morgan Kauffman: 2000 .

Loomis, M.E.S. *Object Databases. The Essentials*; Addison-Wesley: 1995.

Kim, W. *Introduction to Object-Oriented Databases*; The MIT Press: 1990.

Date, C.J. *Introducción a los Sistemas de bases de datos*; Addison-Wesley: México, 1998.

Connolly, T.; Begg, C. *Database Systems: A practical approach to design, implementation and management*; Third edition Addison-Wesley: 2002 .

Taylor, A.D. *Object Technology*; Addison-Wesley: 1997

Douglas, K.; Douglas, S. *PostgreSQL* , Sams Publishing 2003.

Stinson, B *PostgreSQL Essential Reference*, New Riders Publishing 2001.

Flannery, R. *The Informix Handbook*, Prentice Hall 2000.

Beaulieu, A.; Mishra, S. *Mastering Oracle SQL*, O'Reilly 2002.

Greenwald, R.; Stackowiak, R.; Stern, J. *Oracle Essentials: Oracle 9i, Oracle 8i & Oracle 8.*, 2ond Edition, O'Reilly 2001.

Articles

Atkinson, M.; Bancilhon, F.; DeWitt, S.; Dittich, K.; Maier, D.; Zdonik, S. *The Object-Oriented Database Manifesto*; Dedictive and Object-Oriented Databases, Proceedings of the First International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan, 1989 . http://www.cs.cmu.edu/People/clamen/OODBMS/Manifesto/htManifesto/Manifesto.html

Stonebraker, M.; Rowe, L.; Lindsay, B.; Gray, J.; Carey, M.; Brodie, M.; Berstein, P.; Beech, D. (Commitee for. Advanced. DBMS. Function). *Third Generation Database System Manifesto. SIGMOD Record* 1990, vol.19, num.3, pàg. 31-44. http://www.db.ucsd.edu/cse132B/Thirdmanifesto.pdf

Altres documents

Laux, F. *Object-Oriented Features of SQL:1999*; Reutilingen University: 2001 . http://www-inf.fh-reutlingen.de/forschung/laux/SQL1999ooAspects.pdf

Melton, Jim *SQL:1999 Major Features & Current Work* 1999, CS32-Data Management & Interchange, Standards Australia.

Enllaços a Internet

http://www.service-architecture.com/database/articles/sql1999.html. Informació relativa a bases de dades orientades a l'objecte.

http://www.acm.org/crossroads/xrds7-3/ordbms.html. Informació sobre bases de dades objecte-relacionals.

http://www.wiscorp.com/SQLStandards.html. Especificació de l'estandard SQL:1999.

http://www-306.ibm.com/software/data/informix/pubs/library/ . Pàgina de manuals d'Informix. Es possible descarregar documentació en format pdf. D'aquests són particularment útils els següents:

- IBM Informix Guide to SQL Reference.
- Database Design and Implementation guide.
- Guia del desenvolupador de rutines i de tipus de dades definits per l'usuari.

http://otn.oracle.com/pls/db901/db901.homepage . Pàgina d'acccés a l'extensa documentació que ofereix Oracle. S'han consultat els documents següents:

- Application Developer's Guide – Object-Relational Features.
- Database concepts.
- SQL Reference.

http://www.postgresql.org/docs/ . Pàgina de documentació sobre PostgreSQL.