UNIVERSITAT OBERTA DE CATALUNYA

Ingeniería Técnica Informática Gestión

AREA BASES DE DATOS

EVALUACIÓN DEL ESTANDARD SQL:1999 RESPECTO A LAS CARACTERISTICAS ORIENTADAS AL OBJETO QUE SOPORTA Y SU IMPLEMENTACION A LOS SGBD OBJECT-RELATIONAL.

Alumno: Ángel del Río Medina

Dirigido por: Gladys Utrera Iglesias

Curso: 2003-2004

Gladys Utrera Iglesias

Consultor - Informática i multimedia,
Colaborador Docente - UOC, CV UOC –
Consultor UOC - Informática de gestión –

ÁNGEL Del Río Medina

Estudiante - Informática de sistemas
Estudiante - Informática de gestión
Simpatizante – UOC

TFC –Area Bases de datos
Alumno: Ángel del Río Medina
Dirigido por: Gladys Utrera Iglesias
Curso: 2003-2004

Índice

- 1. Dedicatoria y agradecimientos 7
- 2. Presentación 7
  - 2.1. Descripción 7
- 3. Objetivos 8
  - 3.1. Objetivos generales 8
  - 3.2. Objetivos particulares 8
- 4. Tareas a realizar 8
  - 4.1. Plan de trabajo 8
  - 4.2. Investigación: Análisis del estándar SQL 1999 8
  - 4.3. Comparación con dos SGBD comerciales 9
  - 4.4. Implementación y demostración practica 9
- 5. Productos 9
  - 5.1. Productos 9
- 6. Calendario 9
  - 6.1. Calendario tabular con las tareas 9
  - 6.2. Diagrama de gantt 9
- 7. Los SGBD comerciales 10
- 8. Contenidos de SQL:1999 12
- 9. Características relacionales 13
- 10. Nuevos tipos de datos 14
  - 10.1. Predicados 16
    - 10.1.1. Semántica del SQL:1999 17
  - 10.2. Seguridad 18
  - 10.3. Base de datos activa 19
- 11. Orientación de objetos 20
- 12. Tipos estructurados definidos por el usuario 21
- 13. Funciones vs métodos 23
- 14. Notaciones funcionales y de punto 24
- 15. Objetos 25

- 16. Utilizando los tipos REF 26
- 17. Futuro del SQL 27
- 18. Introducción a SQL:2003..... 28
    - 18.1. Nuevos tipos de datos..... 28
- 19. PostgreSQL 30
- 20. Tipos de datos relevantes en PostgreSQL..... 30
- 21. Algunas características de PostgreSQL..... 32
    - 21.0.1. Tablas internas 32
    - 21.0.2. Cursores 35
  - 21.1. Lenguajes procedurales 36
    - 21.1.1. Usando PL/pgSQL..... 36
    - 21.1.2. Estructura de PL/pgSQL..... 36
    - 21.1.3. Algunos ejemplos simples de funciones en PL/pgSQL..... 37
  - 21.2. Triggers 38
    - 21.2.1 Ejemplos de Trigger 39
  - 21.3. Herencia en PostgreSQL 40
- 22. Oracle como BD Relacional Orientada a Objeto..... 42
  - 22.1. Ventajas..... 42
  - 22.2. Inconvenientes 42
  - 22.3. Elementos de Oracle para implantación de BDROO..... 42
- 23. Diferencias entre la herencia en PostgreSQL y Oracle 45
- 24. Comparación entre PostgreSQL y Oracle 9i..... 46
  - 24.1. Características Generales..... 46
  - 24.2. Características especificas..... 47
- 25. Otro gigante: Informix..... 48
  - 25.1. Algunas diferencias con Oracle 49
  - 25.2. Objetivos que se han cubierto en Oracle..... 50
  - 25.3. Posibilidades de ampliación..... 51
  - 25.4. Tabla comparativa de Oracle con Informix 52

- 26. Caso práctico ... 53
- 27. Enunciado ... 53
    - 27.1. PARTIDAS ... 53
    - 27.2. HEROES ... 53
    - 27.3. PARTICIPACIONES ... 53
- 28. Modelo E-R ... 54
    - 28.1. PostgreeSQL ... 55
    - 28.2. Oracle ... 56
    - 28.3. Informix ... 58
- 29. Comparación de las características del SQL:1999 entre los productos Oracle y PostgreeSQL ... 59
    - 29.1. UDT ... 59
    - 29.2. Colecciones ... 60
    - 29.3. Tipo referencia ... 61
    - 29.4. Tabla de tipos ... 61
    - 29.5. Herencias en las características previas ... 61
- 30. Conclusiones y Futuros trabajos a desarrollar ... 62
- 31. Bibliografía ... 63
    - 31.1. Artículos ... 63
    - 31.2. Documentos ... 63
    - 31.3. Libros ... 63
    - 31.4. Enlaces ... 64
    - 31.5. TFC ... 64

1. Dedicatoria y agradecimientos

Este trabajo esta dedicado a mi esposa María Victoria, por su paciencia y comprensión conmigo durante estos interminables años de carrera.

También quiero agradecer el cariño y el apoyo de mi hermana y de mis padres. A los cuales también dedico este trabajo.

2. Presentación

2.1. Descripción del área[1]

La información se ha convertido en un recurso de importancia primordial en todas las organizaciones, independientemente de cuáles sean sus ámbitos de negocio y actuación. Por eso, las organizaciones invierten grandes esfuerzos en el desarrollo de sistemas de información para la producción y gestión de la información.
Una de las funciones de las bases de datos es facilitar la descripción y manipulación de los datos que los sistemas de información requieren.
Durante los estudios de Ingeniería Técnica Informática, el estudiante ha adquirido conocimientos sobre las bases de datos relacionales clásicas. Actualmente, el área de bases de datos está evolucionando frente a la aparición de nuevos retos en la gestión de la información. Por ejemplo, la proliferación de la Web comporta la necesidad de tratar datos semiestructurados y de integrar datos que provienen de diferentes fuentes de información. La necesidad de recopilar, resumir y analizar datos para ofrecer apoyo a la toma de decisión está presente en los almacenes de datos. Adicionalmente, las bases de datos relacionales siguen evolucionando, incorporando nuevas características presentes en otros campos, como en el de la orientación a objetos.

3. Objetivos

3.1. Objetivos generales:

Realización de un trabajo de síntesis de los conocimientos adquiridos en diferentes asignaturas en la etapa final de los estudios. Normalmente el trabajo fin de carrera (TFC) es un trabajo eminentemente practico y vinculado al ejercicio profesional de la informática aunque en algunos casos puede ser un trabajo de investigación.

3.2. Objetivos particulares:

- Analizar y sintetizar las especificaciones del estándar SQL 1999
- Comparar el estandar con tres gestores de bases de datos comerciales, como en este caso son PostgreeSQL, Informix y Oracle.
- Planificar y estructurar de un caso practico que sirva para analizar las similitudes y diferencias entre Oracle, PostgreeSQL, Informix y el estándar.
- Elaborar una memoria del proyecto
- Elaborar una presentación del desarrollo y resultados finales del proyecto.

4. Tareas

Las tareas hasta obtener la memoria son las siguientes

4.1. Plan de trabajo (PEC1)

Elaboración de un documento en el cual estén identificadas y descritas las tareas necesarias para llevar a cabo el proyecto y su distribución de tiempos. Este documento deberá ser capaz de responder cuestiones como: ¿Qué se quiere hacer? ¿Qué aspira tener cuando finalice el trabajo? ¿Qué se espera encontrar? ¿Cómo lograr el objetivo general? ¿Cómo se logran los objetivos específicos?

[1] Extracto del Área 02 (Bases de datos) de los TFC del año 2003-2004

4.2. Investigación: Análisis del estándar SQL1999 (PEC2)

Para lo cual se analizaran los documentos ("The Object-Oriented Data base Manifesto" y "Third generation database manifesto").

Haciendo especial énfasis en estos tres apartados:

- **GC1**: Más allá de los servicios tradicionales de gestión de datos, la tercera generación de SGBD debe proveer soporte para reglas y estructuras de objetos complejos.
- **GC2**: La tercera generación de SGBD debe absorber y mantener la segunda generación de SGBD, es decir los SGBD relacionales clásicos.
- **GC3**: La tercera generación de SGBD debe ser abierta a otros subsistemas.

4.3. Comparación de los tres SGBD (PEC2)

Comparación del estándar con PostgreeSQL, comparación de estándar con Oracle, comparación entre Oracle y PostgreeSQL.
En menor medida también se realizaran comparaciones con Informix.

Haciendo énfasis en el GC1 que es el apartado más relevante, concretamente en las siguientes características:

- Especificación de tipos abstractos de datos
- Restricciones y disparadores
- Identificadores de objetos
- Métodos
- Herencia
- Polimorfismo
- Encapsulamiento
- Todas las demás facilidades normalmente asociadas con orientación a objetos

4.4. Implementación y demostración practica (PEC3)

Realización de una pequeña implementaron que ejemplifique y resalte las diferencias entre los gestores de bases de datos y el estándar.

5. Productos

5.1. Productos

Los productos a obtener son dos:

Memoria del proyecto, que sintetiza el trabajo realizado en el TFC y tiene que demostrar claramente que se han alcanzado los objetivos propuestos.

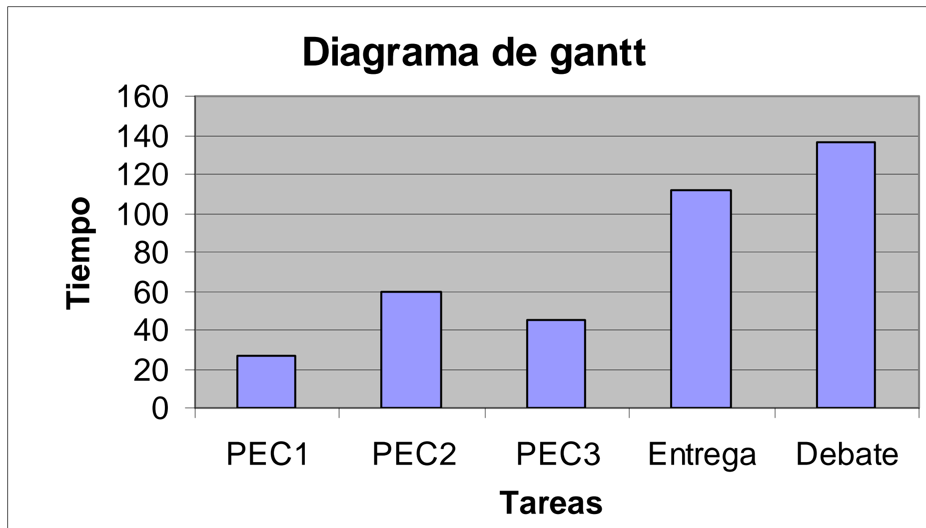
Presentación virtual, que debe sintetizar de forma clara y concisa el trabajo realizado a lo largo del semestre y los resultados obtenidos.

6. Calendario

6.1. Calendario tabular con las tareas

Nombre	Inicio	Entrega
PEC1	24/02/2004	22/03/2004
PEC2	01/03/2004	30/04/2004
PEC3	30/04/2004	14/06/2004
Entrega Final: Memoria y Presentación Virtual	24/02/2004	15/06/2004
Cierre Debate Virtual	15/02/2004	30/06/2004

6.2. Diagrama de gantt



7. Los SGBD comerciales[2]

Ya conocéis a grandes rasgos la historia de la evolución de los SGBD a lo largo de los años. Recordad que la mayoría de los SGBD relacionales actuales aparecieron antes de 1985. Actualmente hay en el mercado más de un centenar de marcas diferentes de SGBD relacionales y unas cuantas más de otros tipos de SGBD (orientados a objetos, especializados en documentos, especializados en multimedia, especializados en gestión geográfica, etc.).

Aunque la funcionalidad básica de todos los productos relacionales es muy parecida y aunque todos se basan en el SQL, cada producto tiene sus particularidades y una arquitectura (interna y externa) diferente, que se adapta a los tipos de entornos a los que va enfocado.
Actualmente todos los SGBD del mercado están pensados para que se puedan integrar en entornos cliente-servidor, y tienen que dar servicio en situaciones de operación muy diversas.

Los SGBD más conocidos

Los SGBD relacionales más conocidos son, sin ningún tipo de duda, los siguientes:

• Oracle.
• DB2 de IBM.
• Informix.
• SQL-Server de Microsoft.

Sin embargo hay muchos otros bastante conocidos como, por ejemplo, Adaptive Server de Sybase (antes se llamaba SQL Server y fue el origen del de Microsoft) o el OpenIngres de CA (antes se llamaba Ingres, y se le considera el padre de la mayoría de los sistemas actuales).

Hay SGBD específicos para ser utilizados como herramienta monousuario sobre un pequeño ordenador personal. Este ordenador puede estar aislado o bien conectado como cliente en una red C/S. En este último caso, en el equipo cliente sólo tendríamos la BD local y el SGBD local, con el cual también podríamos acceder a los SGBD remotos.

Hay servidores de BD muy potentes para dar servicio a millares de usuarios simultáneamente. Estos SGBD, que a menudo son especializados en paralelismo, funcionan generalmente sobre SO propietarios. Por ejemplo, DB2/MVSo Teradata de NCR.

Access

Un ejemplo típico de SGBD utilizado como herramienta monousuario es el Access de Microsoft, aunque sería más correcto decir el Jet, que es el nombre del motor. El Access es, de hecho, una herramienta de acceso a BD que puede trabajar con cualquier gestor de datos que tenga previsto el ODBC; es decir, prácticamente con todos.

Por razones de disponibilidad y precio nos puede interesar tener, en lugar de un servidor muy potente, todo un conjunto de servidores de menos potencia, quizás distribuidos por la red. Un ejemplo de producto para este tipo de entornos sería el SQL-Server de Microsoft.

Muchos de los SGBD del mercado están pensados para poder funcionar en un gran abanico de entornos y potencias.

[2] Extracto del modulo: Introducción a los sistemas de gestión de bases de datos (Asignatura Bases de datos II). Autor: Rafael Camps Paré - P03/75053/02083

Estos tipos de SGBD se venden con diferentes presentaciones o versiones. Es frecuente disponer de tres versiones: una versión monousuario de uso personal, una multiusuario para entornos C/S y otra para grandes servidores multiprocesadores paralelos.

> **Por ejemplo,...** .los sistemas Oracle y Informix fueron concebidos inicialmente para plataformas de tipo medio, pero posteriormente han ido ampliando el alcance hacia arriba y hacia abajo en la gama de potencias.

EL SGBD que los estudiantes utilizáis para las prácticas de esta asignatura es una versión monousuario del sistema Informix, que tiene versiones ideadas para funcionar en equipos de gran potencia con paralelismo masivo.

Obviamente, hay una cierta relación entre el precio de un SGBD y su potencia.
Los precios pueden oscilar entre algunos cientos de euros y unos cuantos cientos de millares.
Las tendencias actuales de los SGBD relacionales del mercado se pueden resumir en los puntos siguientes:

- Facilidad de crecimiento mediante el funcionamiento con plataformas de paralelismo masivo.
- Funcionalidad pensada para llevar a cabo análisis masivos de datos. (Por ejemplo, *data warehousing* y *data minin*g.)
- Ampliación de los tipos de datos y posibilidad que el usuario informático los pueda definir según sus necesidades.
- Incorporación al mundo de la orientación a objetos.
- Incorporación al mundo de Internet.
- Incorporación a las arquitecturas de objetos distribuidos.(Como CORBA o COM++.)

8. Contenidos de SQL:1999

Este texto solo contempla aquellas características que son nuevas en la más reciente generación del SQL standard (SQL:1999) dando por supuestas las características del estándar SQL-92.

Las características del SQL:1999 pueden ser divididas en dos categorías: “características relacionales” y “características de orientación a objetos”.

9. Características relacionales

Aunque llamamos a esta categoría de características “relacionales”, rápidamente reconoceremos que más apropiadamente deberían estar catalogadas como “características que hacen relación al SQL tradicional y modelo de datos”...una frase menos contundente. Las características aquí no están estrictamente limitadas al modelo relacional, pero no están relacionadas a la orientación de objetos.

Estas características están a menudo divididas dentro de cinco grupos: nuevos tipos de datos, nuevos predicados, semántica elevada, seguridad adicional y bases de datos activas. Miraremos uno por uno.

10. Nuevos tipos de datos

SQL:1999 tiene cuatro nuevos tipos de datos (aunque uno de ellos tiene algunas variantes identificables). El primero de estos tipos es LARGE OBJECT, o tipo LBO.

Éste es el tipo con variantes: CHARACTER LARGE OBJECT (CLOB) y BINARY LARGE OBJECT (BLOB).

CLOB actúa mucho como cadena de carácter, pero tiene restricciones que no permiten el uso en PRIMARY KEY o predicados UNIQUE, en FOREIGN KEY, y en comparaciones más que puramente igualan o desigualan textos.

BLOB tiene restricciones similares. (Por implicación, LOB no puede ser utilizado en GROUP BY o en ORDER BY tampoco.) Las aplicaciones no se transferirían enteras hacia los valores LOB desde la base de datos (después de un almacenaje), pero manipularía los valores LOB a través de un lado de cliente especial llamado localizador LOB.

En SQL:1999, un localizador es un valor binario único que actúa como una clase de sustituto para un valor cogido dentro de la base de datos; los localizadores pueden ser usados en operaciones (tales como SUBSTRING) sin generalizar la transferencia de un valor entero LOB de la interfaz del cliente-servidor.

Otro nuevo tipo de datos es BOOLEAN, el cual permite a SQL grabar directamente valores verdaderos true, false y unknown. Combinaciones complejas de predicados también están expresados de manera que son más manejables que la clase normal de reestructuración, podríamos hacer las siguientes:

```
WHERE   COL1 > COL2 AND
        COL3   = COL4 OR
        UNIQUE (COL6) IS NOT FALSE
```

SQL:1999 también tiene dos nuevos tipos de composición: ARRAY y ROW. El tipo ARRAY permite almacenar colecciones de valores directamente en una columna de tabla de bases de datos. Por ejemplo:

```
WEEKDAYS  VARCHAR(10)  ARRAY¨[7]
```

Permitiría almacenar los nombres de los siete días de la semana directamente en una sencilla fila en la base de datos. ¿Esto significa que SQL:1999 permite a las bases de datos que no satisfacen la forma de la primera norma?. Realmente lo hace, en el sentido que este permite “reparar grupos”, los cuales la primera forma prohibe.

El tipo ROW en SQL:1999 es una extensión del (anónimo) tipo row que SQL ha tenido siempre. Este da a la base de datos diseños de poder adicional para almacenar valores estructurados en columnas simples de la base de datos:

```
CREATE TABLE employee (
    EMP_ID    INTEGER,
    NAME              ROW (
          GIVEN       VARCHAR(30) ,
          FAMILY      VARCHAR(30)  ) ,
    ADDRESS ROW (
          STREET      VARCHAR(50) ,
```

```
                    CITY                VARCHAR(30) ,
                    STATE     CHAR(2)  )
            SALARY   REAL )

        SELECT   E.NAME.FAMILY
        FROM          employee  E
```

Mientras se podría argumentar que esto viola la primera norma, la mayoría de los observadores lo reconocen sólo como otro tipo de dato que se puede descomponer.

SQL:1999 añade aún otro tipo de datos relacionado fácilmente, llamado "distinct types".

Reconociendo que es generalmente poco probable que una aplicación quisiera, decir que directamente compare la talla de zapato de un empleado con su IQ, el lenguaje permite a los programadores declarar SHOE_SIZE e IQ para cada dato "basado en" INTEGER, pero prohibe directamente mezclar a aquellos dos tipos en expresiones. Por consiguiente, una expresión como:

```
WHERE   MY_SHOE_SIZE >  MY_IQ
```

(dónde la variable del nombre implica su tipo de dato) sería reconocido como un error de sintaxis. Cada uno de esos dos tipos podrían ser "representados" como INTEGER, pero el sistema SQL no les permite mezclarlos.

```
SET  MY_IQ  =  MY_IQ  *  2
```

En vez de esto, los programas tendrán que expresar implícitamente sus intentos deliberados cuando estén haciendo tales mezclas:

```
WHERE  MY_SHOE_SIZE  >
CAST  (MY_IQ  AS SHOE_SIZE)
SET  MY_IQ  =
MY_IQ  *  CAST (2  AS  IQ)
```

Añadiendo a estos tipos, SQL:1999 ha introducido también tipos de usos definidos, pero ellos caen en la lista de características de orientación a objetos.

10.1. Predicados

SQL:1999 tiene tres nuevos predicados, uno de los cuales se considera como característica orientada a objetos. Los otros dos son de SIMILAR predicado y de DISTINCT predicado.

Desde la primera versión de SQL standard, la búsqueda de cadena de caracteres ha estado limitada a comparaciones muy simples (like =, > , or <>) y el modelo menos rudimentario uniendo capacidades del predicado LIKE:

```
WHERE NAME LIKE ´% SMIT_´
```

Cuando une el valor NAME que tiene cero o más caracteres precediendo los cuatro caracteres ´SMIT´ y exactamente un carácter después de ellos (tal como SMITH o HAMMERSMITS).

Reconociendo que las aplicaciones a menudo requieren capacidades más sofisticadas que son todavía cortas de operaciones de texto completo, SQL:1999 ha introducido el predicado SIMILAR que dan los programas UNIX como expresiones regulares para el uso de comparar modelos. Por ejemplo:

```
WHERE NAME SIMILAR TO
        ´(SQL- (86 | 89 | 92 | 99) ) | (SQL (1 | 2 | 3) ) ´
```

(el cual compararía los nombres dados al SQL standard durante los años transcurridos). Es un poco desafortunado que la sintaxis de las expresiones regulares utilizadas en el predicado SIMILAR no comparen, todo lo que debiesen, la sintaxis de las expresiones regulares de UNIX, pero algunos de los caracteres de UNIX ya fueron utilizados para otras propuestas en SQL.

El otro nuevo predicado, DISTINCT, es muy parecido en operación al corriente SQL predicado UNIQUE; la diferencia importante es que dos valores nulos son considerados no iguales a otro y satisfaría al predicado UNIQUE, pero no todas las aplicaciones requieren ser el caso.

El predicado DISTINCT considera a dos valores nulos no son distintos uno del otro y aquellos dos valores nulos causarían un predicado DISTINCT que no sería correcto.

10.1.1. Semántica del SQL:1999

Es difícil saber exactamente dónde trazar la línea cuando hablamos de la nueva semántica en SQL:1999, sin embargo este texto intenta exponer los aspectos más relevantes del lenguaje.

Muchos ambientes usan vistas como mecanismo de seguridad y/o simplificador de la vista de unas aplicaciones de la base de datos. Sin embargo, si muchas vistas no son actualizadas, entonces, estas aplicaciones a menudo tienen que “escapar” del mecanismo de vistas y confiar en la actualización directa de las tablas subyacentes; lo cual es una situación insatisfactoria.

SQL:1999 ha crecido significativamente el rango de vistas que pueden ser actualizadas directamente, utilizando sólo las facilidades que se proveen en el standard. Esto depende en gran medida de las dependencias funcionales para determinar que vistas adicionales pueden ser actualizadas, y cómo hacer cambios a la tabla de base subyacente para efectuar dichas actualizaciones.

Otro defecto de SQL, ha sido su incapacidad para la repetición de defectos de las aplicaciones tales como proceso de programa de material.
Bien, SQL:1999 ha dado una facilidad llamada RECURSIVE QUERY (repetición de pregunta) para satisfacer sólo esta clase de requerimiento. Escribiendo una repetición de pregunta conlleva escribir la expresión query que se quiera repetir y darle un nombre, entonces utilizando el nombre asociado a la repetición de pregunta:

```
WITH RECURSIVE
Q1 AS SELECT...FROM...WHERE...,
Q2 AS SELECT...FROM...WHERE...
SELECT...FROM Q1, Q2 WHERE...
```

Ya hemos mencionado localizadores tales como el valor client-side que pueden representar un valor LOB almacenado en el server side. Los localizadores se pueden usar de la misma manera para representar valores ARRAY, aceptando el hecho de que (como los LOB) los ARRAY pueden a menudo ser demasiado largos para pasar entre una aplicación y la base de datos. Los localizadores también se pueden usar para representar tipos de valores definidos de los usuarios, los cuales tienen también el potencial de ser grandes y poco manejables.

Finalmente, SQL:1999 ha añadido la noción de savepoints, implementados en productos. Un savepoint es como una substracción en la cual una aplicación puede deshacer las aplicaciones hechas antes del comienzo del savepoint sin deshacer todas las acciones de una transacción entera. SQL:1999 permite ROLLBACK TO SAVEPOINT y RELEASE SAVEPOINT, los cuales actúan destinados a la substracción**.**

10.2. Seguridad

El SQL:1999 aumenta su seguridad. Los privilegios se pueden garantizar para actuar sólo con sus funciones predefinidas. Esta estructura anidada puede simplificar enormemente a menudo el trabajo difícil de manejar la seguridad en un ambiente de una base de datos.
Las actuaciones han sido sabiamente implementados por los productos SQL durante varios años (ocasionalmente bajo diferentes nombres) y finalmente se ha recuperado el standard.

10.3. Base de datos activa

SQL:1999 reconoce la noción de la base de datos activa, si bien algunos años después lo hizo la implementación. Esta facilidad se provee a través de una característica conocida como triggers. Un trigger, es una facilidad que permite a los diseños de la base de datos ordenar al sistema de esta, hacer ciertas operaciones cada vez que una aplicación realiza operaciones específicas en tablas particulares.
Por ejemplo, los triggers se pueden usar para entrar al sistema todas las operaciones que cambian los salarios en una tabla de empleados:

```
CREATE TRIGGER log_salupdate
        BEFORE UPDATE OF salary
                ON employees
        REFERENCING OLD ROW as oldrow
                        NEW ROW as newrow
        FOR EACH ROW
        INSERT INTO log_table
          VALUES (CURRENT_USER,
                    oldrow. salary,
                    newrow. salary)
```

Los triggers se pueden usar para muchas propuestas, no sólo para entrar datos en el sistema. Por ejemplo, se puede escribir triggers que guarden un presupuesto económico reduciendo dinero de las adquisiciones si algún empleado nuevo es contratado...e incrementando una excepción si no hay dinero disponible para hacer esto.

11. Orientación de objetos

Añadiendo a las características más tradicionales de SQL, el desarrollo de SQL:1999 fue enfocado ampliamente, algunos observadores dirían que demasiado, añadiendo soporte para los conceptos de la orientación de objetos al lenguaje.

Algunas de las características que caen dentro de esta categoría fueron primero definidas en el standard SQL/PSM publicado a finales de 1996, específicamente, soportado por funciones y procedimientos del extracto de SQL. Las posibilidades de SQL:1999 que habilitan esto, se llaman rutinas SQL-invoked.

12. Tipos estructurados definidos por el usuario

La facilidad más fundamental en SQL:1999 que da soporte a la orientación de objetos es el tipo estructurado definido por el usuario; la palabra “estructurado” distingue esta característica de los distintos tipos (los cuales son “user-defined “ también, pero están limitados en SQL:1999 para estar basados en “built-in types” en SQL y esto no tiene la estructura asociada con ellos). Los tipos estructurados tienen una serie de características, las más importantes son:

- Podrían estar definidas para tener uno o más atributos, cada uno de los cuales puede ser cualquier tipo de SQL, incluyendo built-in types como INTEGER, tipos de colección como ARRAY, u otros tipos estructurados.

- Todos los aspectos de sus comportamientos se proveen a través de métodos, funciones y procedimientos.

- Sus atributos están encapsulados mediante el uso de “system-generated” con las funciones de observar y mutar (“get” y “set”) que proveen el acceso sólo a sus valores. Sin embargo, estas funciones “system-generated” no pueden ser sobrecargadas; sin embargo, todas las otras funciones y métodos si podrían.

- Las comparaciones de sus valores están hechas sólo a través de las funciones “user-defined”.

- Podrían participar en tipos de jerarquías, en las cuales, tipos más especializados (subtypes) tienen todos los atributos de utilizar todas las rutinas asociadas con los tipos más generales (supertypes), pero podrían añadir nuevos atributos y rutinas.

Veamos un ejemplo de una definición de tipo estructurado:

```
CREATE TYPE emp_type
  UNDER person_type
AS ( EMP_ID INTEGER,
       SALARY       REAL )
INSTANTIABLE
NOT FINAL
REF ( EMP_ID )
INSTANCE METHOD
  ( ABS_OR_PCT BOOLEN,
    AMOUNT  REAL )
       RETURNS REAL
```

Este nuevo tipo, es un subtipo de otro tipo estructurado que se podría usar para describir personas en general, incluyendo atributos comunes como nombre y direcciones; el nuevo atributo emp_type incluye “plain old persons” que no tiene el ID del empleado ni el salario.

Hemos declarado este tipo para que sea instantáneo y que permita tener subtipos definidos (NOT FINAL). Además, que no hay referencias a este tipo son derivados del valor del empleado ID. Finalmente hemos definido un método (lo veremos más tarde) que se puede aplicar a las instancias de este tipo.

SQL:1999, después de un extenso coqueteo con la herencia múltiple (en la cual a los subtipos se les permite tener más de un supertipo inmediato), ahora provee un modelo tipo cercano ligado con la herencia simple de Java. A los definidores de tipo se les permite especificar que cualquier tipo es instantáneo (en otras palabras, los valores de otro tipo específico pueden ser creados) o no instantáneo (equivalente para abstraer types de otros lenguajes de programación). Y, naturalmente, en ningún lugar, tal como en una columna, donde un valor de tipos estructurados sea permitido, un valor de cualquiera de sus subtipos puedan aparecer; esto provee la clase exacta de sustitución de “object-oriented” de los cuales dependen los

programas.

Algunos “object-oriented” de los lenguajes de programación, tales como C++ , permiten a los definidores de tipo especificar el grado en los cuales los tipos son encapsulados: un nivel de encapsulación de PUBLIC aplicado a un atributo significa que cualquier usuario de este tipo puede acceder al atributo, PRIVATE significa que ningún otro código más que el que está en uso para implementar el método del tipo puede acceder al atributo, y PROTECTED significa que sólo el método del tipo y el método de cualquier subtipo del tipo puede acceder al atributo.

SQL:1999 no tiene este mecanismo, aunque se intentó definir; se anticipa que sea propuesto para una futura revisión del standard.

13. Funciones vs métodos

SQL:1999 hace una distinción importante entre las funciones y los métodos de SQL-invoked. En resumen, un método es una función con varias restricciones y elevaciones. Resumamos las diferencias entre los dos tipos de rutinas:

- Los métodos están asociados a un solo tipo definido por el usuario; las funciones no.

- El tipo definido por el usuario al cual está limitado un método es el tipo de datos de un argumento distinguido para el método (el primero, argumento no declarado); una función no tiene argumentos distinguidos en este sentido.

- Las funciones pueden ser polimórficas (sobrecargadas), pero en tiempo de compilación se elige una función específica, examinando los tipos declarados de cada argumento en una invocación de función y escogiendo el "mejor emparejamiento" entre las funciones candidatas (que tienen el mismo nombre y el mismo número de argumentos). Los métodos también pueden ser polimórficos, pero el tipo más específico de sus argumentos distinguidos, determinado en tiempo de ejecución, permite definir hasta la ejecución la selección del método concreto a invocar, todos los demás argumentos se resuelven en tiempo de compilación en base a los tipos declarados de los argumentos.

- Los métodos deben ser almacenados en el mismo esquema en el cual se almacena la definición de sus tipos estructurados asociados; las funciones no están limitadas a un determinado esquema.

- Tanto las funciones como los métodos, pueden ser escritos en SQL (usando las sentencias computacionalmente completas del SQL/PSM) o en cualquiera de los diversos lenguajes de programación más tradicionales, incluyendo Java.

14. Notaciones funcionales y de punto

Acceder a los atributos de los tipos “user-defined” se pude hacer usando cualquiera de las dos notaciones. En muchas situaciones, las aplicaciones podrían parecer más naturales cuando usan “notaciones de punto”:

```
WHERE  emp.salary  >  10000
```

Mientras en otras situaciones, una notación funcional podría ser más natural:

```
WHERE  salary (emp)  >  10000
```

SQL:1999 soporta ambas notaciones; de hecho, son definidas para ser variaciones sintácticas de lo mismo, tanto como “emp” está en una entidad de almacenaje (como una columna o variable) cuyos tipos declarados son algún tipo estructurado con un atributo nombrado “salary”...o exista una función nombrada “salary” con un argumento simple cuyo tipo de dato es el tipo estructurado de emp (apropiado).

Los métodos son ligeramente menos flexibles que las funciones en este caso: Sólo las notaciones de punto se pueden usar para las invocaciones de método, al menos para las propuestas de la especificación del argumento distinguido.

Por ejemplo:

```
emp.salary
```

Otro método diferente, puede combinar la notación funcional y de punto:

```
emp.give_raise (amount)
```

15. Objetos

Se habrá podido observar que se ha usado la palabra "object" (objeto) a lo largo del texto sobre la descripción de los tipos estructurados.
Esto es porque, además de ciertas características como la herencia de tipos, encapsulación y más adelante las instancias de los tipos estructurados de SQL:1999 son valores simples, sólo como ejemplos de la construcción en tipos del lenguaje.
Por supuesto, el valor de employee es más complejo (tanto en apariencia como en comportamiento) que un ejemplo de INTEGER, pero es todavía un valor sin ninguna otra identidad que la dada por su valor.

Para mejorar las características que permite SQL para proveer objetos, habría que tener algún sentido de identidad que puedan tener referencia en una variedad de situaciones. En SQL:1999, esa capacidad es sustituida permitiendo a la base de datos diseñar para especificar que ciertas tablas están definidas para ser "typed tables"... esto es, las definiciones de sus columnas se derivan de los atributos de un tipo estructurado:

```
CREATE TABLE empls OF employee
```

Tales tablas tienen una columna para cada atributo del tipo estructurado subrayado. Las funciones, métodos y procedimientos definidos para operar en ejemplos del tipo row, opera en rows de la tabla!. Los rows de la tabla, entonces, son valores de , o ejemplos de, el tipo.

Se le da a cada row una única identidad que se comporta sólo como un OID (objeto identificador)...es único en espacio (esto es dentro de la base de datos) y tiempo (la vida de la base de datos).

SQL:1999 provee un tipo especial, llamado REF type, cuyos valores son aquellos identificadores únicos. Se da siempre un REF type con un tipo estructurado especificado.

Por ejemplo, si tuviéramos que definir una tabla conteniendo una columna llamada "manager" cuyos valores hicieran referencia a los rows en una typed table de empleados, aparecería algo como esto:

```
manager        REF (emp_type)
```

Un valor de un tipo REF cualquiera identifica un row en una tabla de tipos (de un específico tipo estructurado, por supuesto) o este no especificará nada en absoluto, lo cual podría significar que es un "dangling reference" (referencia colgada) dejada después del row una vez el identificado fuese borrado.

Todos los tipos REF están "scoped" (alcanzados) así que la tabla a la que hacen referencia se conoce al compilar en tiempo. Durante el desarrollo de SQL:1999, se hicieron esfuerzos para permitir a los tipos REF que fueran más generales que esto (por ejemplo, ninguna de las varias tablas podían estar en el scope (alcance), o ninguna tabla del propio tipo estructurado estaría en el scope incluso si la tabla fuera creada después de que el tipo REF fuese creado también); sin embargo, se encontraron varios problemas que no podían ser resueltos sin retrasar la publicación del standard, así que se adoptó esta restricción. Por un lado el efecto de la restricción, posiblemente un efecto beneficioso, es que los row de los tipos REF ahora soportan mucho mejor la integridad referencial, posiblemente facilitando la tarea de implementar en algunos productos!

16. Utilizando los tipos REF

No nos debería sorprender aprender que los tipos REF se pueden usar de formas un poquito más sofisticadas que solamente almacenándolos y recuperándolos.

SQL:1999 provee sintaxis "seguir una referencia" para acceder a los atributos del valor de un tipo estructurado:

```
SELECT  emps.manager->last_name
```

La notación pointer (→) se aplica al valor de unos tipos REF y es entonces "seguido" dentro del valor identificador del tipo estructurado asociado, el cual, por supuesto, es realmente un row de la tabla de tipos que está al alcance del tipo REF.

Estos tipos estructurados están ambos asociados con el tipo REF de la columna manager en la tabla emps y el tipo de la otra tabla (cuyo nombre requerido no aparece en esta expresión).

Sin embargo, esta tabla estructurada debe tener un atributo llamado last_name, y la tabla de tipo entonces tiene una columna con ese nombre.

17. Futuro del SQL

Es difícil saber que traerá el futuro, pero tanto los grupos ANSI como ISO están destinados a evitar el largo proceso que tuvo como resultado el SQL:1999. Todos creemos que 6 años es demasiado tiempo, especialmente con el mundo trabajando en "el mundo web" cada vez más.

En cambio, los planes están siendo desarrollados para que se revisen los resultados más o menos cada tres años, incluso si los realces técnicos están en algún lugar más modesto que los de SQL:1999.

Además para desarrollar las partes principales del standard de SQL:1999, partes adicionales de SQL están siendo desarrolladas para dirigir temas como temporal data (fecha temporal), la relación con Java y la gestión de fechas externas con SQL data.

18. Introducción a SQL:2003

SQL:2003 hace revisiones a todas las partes de SQL:1999 y añade un nuevo estilo, Parte 14:SQL/XML (Especificaciones relacionadas con XML). Añadiendo, hay una reorganización ligera de las partes heredadas del SQL:1999. Un trozo sustancial de la Parte 2 del SQL:1999: SQL/Foundation que comercializado con Information Schema y Definition Schema está separado dentro de su propia parte, la Parte 11: SQL/Schematra en el SQL:2003.
La Parte 5 de SQL:1999 es decir SQL/Bindings ha sido eliminada en SQL:2003 fusionando todo el material contenido en dicha parte dentro de la Parte 2 de SQL:2003 llamada SQL/Foundation.

Las siguientes características son nuevas, introducidas en el SQL:2003, SQL/Foundation:

- Nuevos tipos de datos
- Aumentos a las rutinas invocadas de SQL
- Extensiones al estado de cuentas de CREATE TABLE
- Una nueva fusión (MERGE) al estado de cuentas,
- Un nuevo schema object (objeto de esquema) – generador de secuencias
- Dos nuevas clases de columnas – columna de identidad y columna generada.

Las futuras características a implementar serán tales como: un chequeo retrospectivo de limitaciones OLAP (on-line analytical processing) extensiones de la forma de new built – in functions (nueva construcción en funciones) y una nueva cláusula WINDOW en las expresiones de pregunta (publicada anteriormente como una corrección en SQL:1999), soportado con el uso de los datos probados para una mejor actuación, también son mejorados los manejadores para salvar punteros, etc...

18.1. Nuevos Tipos de Datos

SQL:2003 retiene todos los tipos de datos que existían en SQL:1999 con la excepción de los tipos BIT y BIT VARYING. Estos fueron borrados del standard debido a la falta de soporte en las bases de datos de los motores de SQL, y también a la falta del soporte esperado en el futuro. SQL:2003 introduce tres nuevos tipos de datos: BIGINT, MULTISET y XML. Desde que el tipo de dato XML es parte de SQL/XML no comentaremos nada más a cerca de él aquí.
Los nuevos tipos de datos son de primera clase, significando que ellos pueden ser utilizados en todos los contextos igual que los otros tipos de datos existentes en SQL, ejemplo: como en una columna de tipos, parámetros y tipos de las rutinas invocadas de SQL, ETC...

El tipo de dato BIGINT es un nuevo tipo numérico similar a los existentes SMALLINT e INTEGER, sólo con una precisión mayor. Pensado como una precisión en particular para un tipo INTEGER (o cualquiera de los otros tipos de datos numéricos) no es mandado en el standard de SQL, las implementaciones actuales en general soportan valores 32- bit INTEGER. Estas implementaciones entonces normalmente soportan valores 64-bit BIGINT. Sin embargo, una implementación conforme podría también elegir cualquier otra precisión (incluso la elección de un decimal o un binario). El tipo BIGINT soporta las mismas operaciones aritméticas como el tipo INTEGER, ejemplo: +, -, ABS, MOD, etc...

El nuevo tipo de dato MULTISET es una nueva colección de tipos similar a los existentes en el tipo ARRAY, pero sin una orden implementada. Conceptualmente, un multiset es una colección de elementos sin ordenar, todos del mismo tipo (el tipo elemento), con duplicaciones permitidas. El tipo elemento puede ser cualquier otro soportado por los tipos de datos de SQL. Por ejemplo, INTEGER MULTISET denota el tipo de un valor multiset cuyo tipo de elemento es INTEGER y cuyo valor actual podría ser: 5, 30, 100, 45, -8, 9.

El tipo MULTISET puede ser creado tanto enumerando los elementos individuales como sustituyendo los elementos a través de una expresión query (pregunta); ejemplo:

```
MULTISET [1, 2, 3, 4] ó
MULTISET (SELECT grades FROM courses)
```

El último ejemplo muestra como una tabla o una parte de la tabla puede ser convertida en multiset. A la inversa, un valor multiset puede ser utilizado tanto en una referencia a tabla en la cláusula FROM utilizando el operador UNNEST.

El ejemplo muestra como trabajaría:

Ejemplo1: Utilizando UNNEST para hacer referencia a multiset de la cláusula FROM.

```
SELECT   T. A,   T. A*2  AS TIMES_TWO
FROM  UNNEST  (MULTISET [4,  3,  2,  1] )
        AS  T (A)
```

Se vuelve lo siguiente:

A	TIMES_TWO
4	8
3	6
2	4
1	2

19. PostgreSQL

20. Tipos de datos relevantes en PostgreSQL

Como todos los gestores de bases de datos, PostgreSQL implementa los tipos de datos definidos para el estándar SQL3 y aumenta algunos otros.

Tipos de datos del estándar SQL3 en PostgreSQL		
Tipo en Postgres	Correspondiente en SQL3	Descripción
bool	boolean	valor lógico o booleano (true/false)
char(n)	character(n)	cadena de caracteres de tamaño fijo
date	date	fecha (sin hora)
float4/8	float(86#86)	número de punto flotante con precisión 86#86
float8	real, double precision	número de punto flotante de doble precisión
int2	smallint	entero de dos bytes con signo
int4	int, integer	entero de cuatro bytes con signo
int4	decimal(87#87)	número exacto con 88#88
int4	numeric(87#87)	número exacto con 89#89
money	decimal(9,2)	cantidad monetaria
time	time	hora en horas, minutos, segundos y centésimas
timespan	interval	intervalo de tiempo
timestamp	timestamp with time zone	fecha y hora con zonificación
varchar(n)	character varying(n)	cadena de carácteres de tamaño variable

Tipos de datos extendidos en PostgreSQL	
Tipo	Descripción
box	caja rectangular en el plano
cidr	dirección de red o de *host* en IP versión 4
circle	círculo en el plano
inet	dirección de red o de *host* en IP versión 4
int8	entero de ocho bytes con signo
line	línea infinita en el plano
lseg	segmento de línea en el plano
path	trayectoria geométrica, abierta o cerrada, en el plano
point	punto geométrico en el plano
polygon	trayectoria geométrica cerrada en el plano
serial	identificador numérico único

Tipo	Descripción
SET	conjunto de tuplas
abstime	fecha y hora absoluta de rango limitado (Unix system time)
aclitem	lista de control de acceso
bool	booleano '*true*'/'*false*'
box	rectángulo geométrico '(izquierda abajo, derecha arriba)'
bpchar	caracteres rellenos con espacios, longitud especificada al momento de creación
bytea	arreglo de bytes de longitud variable

char	un sólo carácter
cid	*command identifier type*, identificador de secuencia en transacciones
cidr	dirección de red
circle	círculo geométrico '(centro, radio)'
date	fecha ANSI SQL 'aaaa-mm-dd'
datetime	fecha y hora 'aaaa-mm-dd hh:mm:ss'
filename	nombre de archivo usado en tablas del sistema
float4	número real de precisión simple de 4 bytes
float8	número real de precisión doble de 8 bytes
inet	dirección de red
int2	número entero de dos bytes, de -32k a 32k
int28	8 números enteros de 2 bytes, usado internamente
int4	número entero de 4 bytes, -2B to 2B
int8	número entero de 8 bytes, 90#9018 dígitos
line	línea geométrica '(pt1, pt2)'
lseg	segmento de línea geométrica '(pt1, pt2)'
macaddr	dirección MAC
money	unidad monetaria '$d,ddd.cc'
name	tipo de 31 caracteres para guardar identificadores del sistema
numeric	número de precisión múltiple
oid	tipo de identificación de objetos
oid8	arreglo de 8 *oid*s, utilizado en tablas del sistema
path	trayectoria geométrica '(pt1, ...)'
point	punto geométrico '(x, y)'
polygon	polígono geométrico '(pt1, ...)'
regproc	procedimiento registrado
reltime	intervalo de tiempo de rango limitado y relativo (Unix delta time)
smgr	manejador de almacenamiento (*storage manager*)
text	cadena de caracteres nativa de longitud variable
tid	tipo de identificador de tupla, localización física de tupla
time	hora ANSI SQL 'hh:mm:ss'
timespan	intervalo de tiempo '@ <number> <units>'
timestamp	fecha y hora en formato ISO de rango limitado
tinterval	intervalo de tiempo '(abstime, abstime)'
unknown	tipo desconocido
varchar	cadena de caracteres sin espacios al final, longitud especificada al momento de creación
xid	identificador de transacción

21. Algunas características de PostgreSQL

21.0.1. Tablas internas

En las tablas presentadas en esta sección se muestra el contenido de las tablas que PostgreSQL utiliza como catálogos para mantener el sistema. Es en base a la idea de mantenerlo todo en tablas que PostgreSQL, a diferencia de otros gestores de bases de datos, es extensible. La mayoría de la información presentada en esta sección ha sido extraída de examinar el código fuente de PostgreSQL y por esta razón se halla incompleta. Sin embargo se ha procurado mostrar al menos lo más importante para los desarrolladores.

Catálogo del sistema PostgreSQL. Cada base de datos tiene estas mismas tablas, salvo por la primera que es única, que almacenan cada una de las partes que componen la base de datos

Nombre del catálogo	Descripción
pg_database	Bases de datos
pg_class	Clases o tablas
pg_attribute	Atributos o campos de la clase o tabla
pg_index	Índices secundarios
pg_proc	Procedimientos (en C y en SQL)
pg_type	Tipos de datos (del sistema y definidos por el usuario)
pg_operator	Operadores (del sistema y definidos por el usuario)
pg_aggregate	Agregados y funciones agregadas
pg_am	Métodos de acceso
pg_amop	Operadores de métodos de acceso
pg_amproc	Funciones de soporte para métodos de acceso
pg_opclass	Clases de operadores de métodos de acceso

Tabla que contiene todas las bases de datos que existen en el sistema. Vale la pena consultarla en aplicaciones que impliquen explorar diversos aspectos

Table = pg_database			
Field	Type	Length	Description
datname	name	32	Nombre de la base
datdba	int4	4	Uid del dababase admin
datpath	text	var	El path para llegar hasta la base

Tabla que contiene todas las tablas en la base de datos actual

Table = pg_class			
Field	Type	Length	Description
relname	name	32	Nombre de la tabla
reltype	oid	4	El Object Id de la tabla
relowner	oid	4	El UID (postgres) del dueño de la tabla
relam	oid	4	
relpages	int4	4	Número de páginas ocupadas por la tabla
reltuples	int4	4	Número de tuplas en la tabla
relhasindex	bool	1	Verdadero si tiene al menos un índice
relisshared	bool	1	
relkind	char	1	
relnatts	int2	2	Número de atributos
relchecks	int2	2	

reltriggers	int2	2	Número de *triggers* asociados
relhasrules	bool	1	Verdadero si tiene reglas asociadas
relacl	aclitem[]	var	

Tabla que contiene los atributos de los atributos de todas las tablas en la base a ctual.

Table = pg_attribute			
Field	**Type**	**Length**	**Description**
attrelid	oid	4	OID del atributo
attname	name	32	Nombre del atributo
atttypid	oid	4	Oid del tipo definido para el atributo
attdisbursion	float4	4	
attlen	int2	2	Longitud en bytes del atributo
attnum	int2	2	
attnelems	int4	4	
attcacheoff	int4	4	
atttypmod	int2	2	
attbyval	bool	1	
attisset	bool	1	
attalign	char	1	
attnotnull	bool	1	
atthasdef	bool	1	

Para saber que bases de datos hay en el sistema:

```
SELECT * FROM pg_database;
```

Para saber que tablas tengo en la base de datos actual:

```
SELECT * FROM pg_class;
```

Lo mismo, pero sólo las definidas por el usuario, excluyendo las del sistema:

```
SELECT * FROM pg_class WHERE relname !~~ 'pg%';
```

Si sólo queremos saber cuantos registros tiene una tabla, basta con preguntar:

```
SELECT relname,reltuples FROM pg_class WHERE relname=mitabla';
```

donde mitabla es el nombre de la tabla de la cual nos interesa saber el número de registros.

A continuación, se presenta una forma de extraer información de las tablas de la base de datos. pg_database tiene las bases de datos, pg_class tiene las tablas, y pg_attribute tiene los campos. Si lo que queremos saber es el número de registros en una tabla determinada, basta con preguntar lo siguiente, conociendo la tabla, adicionalmente se obtienen los campos. Para, además, saber los tipos de los campos, habría que hacer un *query* a pg_type con los oids de los campos. Por ejemplo:

```
SELECT relname,reltuples,attname,attnum
```

```
    FROM pg_class,pg_attribute
    WHERE pg_class.relname='mitabla'
        AND pg_attribute.attrelid=pg_class.oid
        AND attnum > 0
ORDER BY attnum;
```

Si lo que se desea es saber todo de todas las tablas de la base que no sean del sistema, sino del usuario, por supuesto ordenadas por oid, se hace lo siguiente, pero con el problema de que incluye los índices:

```
SELECT pg_class.oid,relname,reltuples,attname
    FROM pg_class,pg_attribute
    WHERE pg_class.relname !~~ '%pg%'
        AND pg_attribute.attrelid=pg_class.oid
        AND attnum > 0
    ORDER BY pg_class.oid;
```

Para saber cuales son los campos de todas las tablas que se tienen en una base de datos:

```
SELECT pg_class.relname,pg_attribute.attname,pg_class.reltuples,pg_type.typname
        FROM pg_attribute
        WHERE attrelid = pg_class.oid
            AND attnum > 0
            AND atttypid=pg_type.oid
            AND pg_class.oid IN
                (SELECT oid FROM pg_class WHERE relname !~~ 'pg%')
            AND pg_type.oid IN (SELECT oid FROM pg_type);
```

21.0.2. Cursores

El concepto de cursores es el de un índice que se desliza sobre el resultado de una consulta y que permite traer unas cuantas tuplas dentro del total de las resultantes. Se utiliza siempre dentro de una transacción para garantizar permanencia.

```
BEGIN;
DECLARE ccur CURSOR FOR SELECT * FROM mitabla;
FETCH 1 IN ccur;            # lee la siguiente tupla
FETCH 5 IN ccur;            # lee las siguientes cinco tuplas
FETCH FORWARD 5 IN ccur;    # lee las siguientes cinco tuplas
FETCH BACKWARD 1 IN ccur;   # lee la tupla anterior
FETCH ALL IN ccur;          # lee las tuplas restantes
END;
```

21.1. Lenguajes procedurales

A partir de la versión 6.3 de PostgreSQL se permite la inclusión de lenguajes procedurales. El DBMS no sabe interpretar las funciones o *triggers* escritos en estos lenguajes, sino que pasa el control al *parser/ejecutor*, llamado manejador, del lenguaje pertinente. El gestor es un módulo compartido que se carga bajo demanda.

Los dos lenguajes que son distribuidos junto con PostgreSQL, pero no integrados son PL/pgSQL y PL/Tcl. Al momento de hacer la instalación, sólo PL/pgSQL se instala y se incluye en el directorio de librerías. Para PL/Tcl es necesario reconfigurar el DBMS y recompilarlo.

21.1.1. Usando PL/pgSQL

Antes de hablar del lenguaje en sí, hay que dar crédito al autor del lenguaje: Jan Wieck.
Las metas del diseño de PL/pgSQL como lenguaje procedural son:

- que pueda ser utilizado para crear funciones y *triggers*;
- añadir estructuras de control al lenguaje SQL;
- que sea capáz de realizar cómputos complejos;
- que herede todas las definiciones de usuario como tipos, funciones y operadores;
- que sea confiable para correr dentro del DBMS;
- de fácil empleo.

El gestor de PL/pgSQL analiza el código fuente de las funciones y produce un árbol de instrucciones en código binario interno la primera vez que es llamado por el DBMS. El código binario producido es identificado dentro del gestor por el OID de la función. Esto garantiza que al cambiar la función con una secuencia DROP/CREATE, el cambio tendrá efecto sin necesidad de realizar una nueva conexión al servidor.

Para todas las expresiones y secuencias SQL utilizadas en la función, el intérprete de código PL/pgSQL binario crea y prepara un plan de ejecución usando los controladores SPI SPI_prepare() y SPI_saveplan(). Esto se hace la primera vez que cada secuencia individual es procesada en la función PL/pgSQL. Así, una función con código condicional que contiene varias secuencias para las cuales un plan de ejecución será requerido, sólo prepararán y salvarán aquellos planes que realmente serán utilizados durante la vida de la conexión al DBMS.

Incluso es posible crear funciones de cómputo condicional complejo y utilizarlas posteriormente para definir operadores o utilizarlas en índices funcionales. Sin embargo, para conversiones de entrada o salida o cálculos sobre tipos de datos definidos por el usuario, no es posible realizarlos en PL/pgSQL.

21.1.2. Estructura de PL/pgSQL

El lenguaje PL/pgSQL es no hace diferencia entre mayúsculas y minúsculas. Todas las palabras reservadas e identificadores pueden aparecer en una mezcla de mayúsculas y minúsculas.

PL/pgSQL es un lenguaje por bloques. Se define un bloque como:

```
[<<etiqueta>>]
[DECLARE
   declaraciones]
BEGIN
   aserciones
END;
```

Puede haber cualquier número de bloques anidados en la sección de aserciones de un bloque. Los bloques anidados pueden ser utilizados para ocultar variables de la parte exterior del bloque de aserciones. Las variables declaradas en la sección de declaraciones que precede a

un bloque son inicializadas al valor por omisión cada vez que el control pasa al interior del bloque, no solamente cada vez que la función es invocada.

Es importante no confundir la estructura de control BEGIN ... END para agrupar aserciones de PL/pgSQL con los comandos del DBMS para control de transacciones. Las funciones y los *triggers* no pueden comenzar o hacer commit de transacciones y PostgreSQL no tiene transacciones anidadas.

21.1.3. Algunos ejemplos simples de funciones en PL/pgSQL

Las siguientes dos funciones en PL/pgSQL son realmente simples y se muestran su ejecución para mayor claridad del mecanismo.

```
CREATE FUNCTION add_one (int4) RETURNS int4 AS '
   BEGIN
      RETURN $1 + 1;
   END;
' LANGUAGE 'plpgsql';
mancha=> select add_one(5);
add_one
-------
     6
(1 row)
```

```
CREATE FUNCTION concat_text (text, text) RETURNS text AS '
   BEGIN
      RETURN $1 || $2;
   END;
' LANGUAGE 'plpgsql';
mancha=> select concat_text ('Hola ', 'Lola') as periquita;
periquita
---------
Hola Lola
(1 row)
```

21.2. Triggers

Se pueden automatizar acciones que usualmente se ejecutan antes o después de una consulta, de tal manera que al llevar a cabo la consulta se ``dispare'' la ejecución de determinadas acciones, sin intervención humana.

Los *trigger* se pueden escribir en cualquiera de los siguientes lenguajes: C, PL/pgSQL y PL/Tcl.

PostgreSQL tiene varias particularidades respecto a *triggers* que deben de ser tomadas en cuenta. La primera de ellas es la creación automática de ciertas variables que son visibles a la función:

NEW
Variable de tipo RECORD y que contiene el registro nuevo en el caso de un *trigger* disparado con INSERT o UPDATE a nivel de renglones.

OLD
Variable de tipo RECORD y que contiene el registro anterior en el caso de un *trigger* disparado con DELETE o UPDATE a nivel de renglones

TG_NAME
Variable de tipo name que contiene el nombre del *trigger* disparado.

TG_WHEN
Variable de tipo text que contiene la cadena `BEFORE' o la cadena `AFTER' dependiendo de la definición del *trigger*.

TG_LEVEL
Variable de tipo text que contiene la cadena `ROW' o la cadena `STATEMENT' dependiendo de la definición del *trigger*.

TG_OP
Variable de tipo text que contiene alguna de las cadenas `INSERT', `UPDATE' o `DELETE' indicando que operación disparó al *trigger*.

TG_RELID
Variable de tipo oid que contiene el OID de la tabla que disparó el *trigger*.

TG_RELNAME
Variable de tipo name que contiene el nombre de la tabla que disparó el *trigger*.

TG_NARGS
Variable de tipo integer que contiene el número de argumentos dados al *trigger* durante la creación del mismo.

TG_ARGV[]
Variable de tipo arreglo de text que contiene los argumentos dados al momento de crear el *trigger*. El índice comienza en cero y puede ser indicado como una expresión. Índices no válidos (94#94) regresan NULL. Este es el único mecanismo para pasar argumentos a un

trigger, ya que como se ha mencionado anteriormente la función activada no puede tener argumentos.

En segundo lugar, deben de regresar o NULL o un registro o renglón conteniendo exactamente la misma estructura de la tabla con la que fué disparado el *trigger*. Los *triggers* disparados por una acción AFTER siempre deberán de regresar el valor NULL sin ningún efecto. Los *triggers* disparados por una acción BEFORE sólo deberán de regresar el valor NULL cuando se deba de invalidar la acción para el renglón que lo dispara. De otra manera, el renglón o registro regresado reemplaza al insertado o actualizado por la operación. Es posible reemplazar valores en particular directamente en NEW y regresar éste o construir un renglón o registro completamente nuevo que suplante al original.

21.2.1. Ejemplos de Trigger

Tenemos una tabla de archivos (llamada archivos), en la que uno de los campos es un MD5 sobre el contenido del archivo (campo llamado dmd5). En una tabla anexa (llamada aux), necesitamos llevar la cuenta de cuántos registros tenemos con el mismo MD5, para esto tenemos dos campos: dmd5 y cont. Así, cada vez que insertamos un registro en archivos debemos de actualizar el contador correspondiente en aux. De igual manera, cuando borramos un registro de archivos, deberemos de decrementar el contador correspondiente en aux y en caso de que sea el único, eliminar el registro.

Obviamente esto se puede hacer desde el mismo programa de actualización, pero resulta, y es este caso en particular lo que motivó el empleo de *triggers*, que son varios programas los que actualizan esta tabla, así que mantener cada uno de ellos se vuelve un tanto cuanto complejo.

La mejor solución es emplear *triggers*. Cabe señalar que el ejemplo que se muestra a continuación funciona sólo de la versión de PostgreSQL 6.5.3 en adelante, dado que emplea el *Procedure Language* PL/pgSQL y algunas particularidades integradas a partir de esa versión.

```
DROP FUNCTION inc_aux ();
CREATE FUNCTION inc_aux () RETURNS OPAQUE AS '
DECLARE
    myrec  record;
BEGIN
  SELECT * INTO myrec FROM aux WHERE aux.dmd5 = NEW.dmd5;
  IF NOT FOUND THEN
    INSERT INTO aux VALUES (NEW.dmd5, 1);
  ELSE
    UPDATE aux SET cont=cont+1 WHERE dmd5 = NEW.dmd5;
  END IF;
  RETURN NEW;
END;
' LANGUAGE 'plpgsql';
DROP TRIGGER ins_arc ON archivos;
CREATE TRIGGER ins_arc BEFORE INSERT ON archivos FOR EACH ROW
 EXECUTE PROCEDURE inc_aux();
```

Los DROPs antes de crear la función y el *trigger*, son para garantizar que las funciones no existen previamente.
Recordemos que los *triggers* no pueden recibir argumentos y *siempre* tienen que regresar un valor *opaco*.

21.3. Herencia en PostGreSQL

La herencia es un concepto de las bases de datos orientadas a objetos. Esta descubre nuevas posibilidades interesantes al diseño de las bases de datos.

Creemos dos tablas: la tabla ciudades y la tabla capitales. Naturalmente, capitales son también ciudades, así que, queremos de alguna forma mostrar explícitamente las capitales cuando hacemos listas de todas las ciudades.

```
CREATE TABLE capitales (
  nombre text,
  poblacion real,
  altitud int,
  comunidad_autonoma char(2)
);

CREATE TABLE no_capitales (
  nombre text,
  poblacion real,
  altitud int
);

CREATE VIEW ciudades AS
  SELECT nombre, poblacion, altitud FROM capitales
    UNION
  SELECT nombre, poblacion, altitud FROM no_capitales;
```

Esto funciona igual que las consultas, pero se pone feo cuando necesitamos actualizar varias filas, para nombrar una sola cosa.

Una solución mejor sería esta:

```
CREATE TABLE ciudades (
  nombre text,
  poblacion real,
  altitud   int
);

CREATE TABLE capitales (
  comunidad_autonoma char(2)
) INHERITS (cities);
```

En este caso, una fila de *capitales* hereda todas las columnas (*nombre*, *población* y la *altitud*) de su padre, *ciudades*. El tipo de columna *nombre* es *texto*, un tipo nativo del tipo PostgreSQL para la longitud variable de las cadenas de caracteres. Las capitales de los Comunidad_autonomas tienen una columna extra, *comunidad_autonoma*, que muestra el suyo propio. En PostgreSQL, una tabla puede heredar desde cero ó más tablas.

Por ejemplo, la siguiente consulta encuentra los nombres de todas las ciudades, incluyendo las capitales de los Comunidad_autonomas, que son localizadas a una altitud por encima de los 500 pies.

```
SELECT nombre, altitud
 FROM ciudades
```

```
              WHERE altitud > 500;
```

Las cuales se vuelven:

```
nombre    | altitud
-------------+----------
 Alicante  |    2174
 Castellon |    1953
 Valencia  |     845
(3 rows)
```

Por otra parte, la siguiente consulta encuentra todas las ciudades que no son capitales de Comunidad_autonoma y están situadas a una altitud de 500 pies ó más altas.

```
SELECT nombre, altitud
   FROM ONLY ciudades
   WHERE altitud > 500;
```

```
  nombre    | altitud
-------------+----------
 Alicante  |    2174
 Castellon |    1953
(2 rows)
```

Aquí el ONLY antes de CIUDADES indica que la consulta debería ser ejecutada sólo sobre la tabla *ciudades,* y no en tablas por debajo de ella en la jerarquía de herencias. Muchos de los comandos de los que ya hemos hablado –SELECT, UPDATE, y DELETE—soportan la notación ONLY.

22. Oracle como BD Relacional Orientada a Objeto

- o El modelo relacional-orientado a objeto constituye una alternativa para añadir características del paradigma de orientación a objeto a las bases de datos relacionales tradicionales.

- o Incrementa la capacidad para tratar estructuras de datos más complejas sin, por ello, renunciar a su contrastada capacidad para tratar con datos convencionales. En definitiva, supone una evolución del modelo relacional que no renuncia al ingente legado tradicional.

22.1. Ventajas:

- Mejora de la capacidad de representación y tratamiento de datos. Datos mas complejos y lógica de programa asociados a los mismos

- Mejor integración con lenguajes de aplicación basados en la orientación a objeto

- Se pueden abordar problemas difícilmente abordables con las estructuras de datos tradicionales

- Se dota a los objetos generados con los lenguajes OO de todas las capacidades de persistencia, gestión y administración proporcionadas por los SGBB

- Se oferta para cada problema el mecanismo de representación y tratamiento más adecuado a su complejidad estructural

- Los datos estructurados se benefician de mecanismos de consulta más simples y uniformes que los proporcionados por BDOO puras

- El usuario se beneficia de la contrastada fiabil dad de los sistemas de BDR sobre los que se evoluciona

22.2. Inconvenientes:

- Existen varios estándares a la hora de conducir la implantación de un modelo relacional orientado a objeto. Fundamentalmente:

- El modelo propugnado por el OMG Group y que se materializa en la formalización de una estructura de datos gestionada mediante la utilización de OQL. El enfoque de este modelo parte de una aproximación de los modelos OO a los modelos relacionales orientados a objeto

- El modelo propugnado desde el estándar SQL, que se materializa en una estructura de datos gestionada mediante la utilización de SQL3, una evolución de SQL.

- Como suele ocurrir casi siempre que no existe un estándar fuertemente implantado, cada fabricante adopta las características de cada uno que mejor le parecen e incorpora funcionalidades de su propia cosecha. Esto redunda en una pérdida de uniformidad.

- En la actualidad el nivel de implantación de características del paradigma de OO en las bases ROO no es tan completo como en las BDOO puras (en concreto en algunos productos no se han implantado todavía las características de herencia). Eso implica que existen problemas inabordables o difícilmente abordables mediante BDROO que sí lo son con BDOO puras.

22.3. Elementos de Oracle para Implantación de BDROO:

– Tipos Objeto (Object Types) TO. Son elementos de la estructura de la BD
que vienen a corresponderse con el concepto de Tipo de Dato Abstracto (TDA)

- Componentes de los Tipos de Objeto:
  - Nombre, que identifica al objeto unívocamente dentro del esquema de una BD
  - Atributos, que modelizan la estructura y estado de una entidad del mundo real. Pueden contener tipos de datos nativos (números, cadenas, etc.) u otros Tipos de Objeto
  - Métodos, funciones o procedimientos que implementan operaciones y comportamiento sobre los datos
- **Objetos.** Instancias de los TO, almacenadas en una variable o en la BD mediante tablas o atributos de las mismas.
- **Tablas Objeto (Object Tables).** Tipo especial de tabla que puede almacenar un Objeto en cada una de sus filas (Row Objects) o en alguno de sus atributos (Column Objetcs)
- **Vistas Objeto (Object Views).** Elementos para modelizar características ROO sobre un esquema relacional puro.
- **EL T.O. REF**. Es un puntero a un Row Object. Los REFs y colecciones de REFs permiten modelizar relaciones muchos a uno reduciendo la necesidad de claves externas. Se le puede asignar un valor nulo.
- **Colecciones (VARRAY y Tablas anidadas).** T.O. para modelizar relaciones uno a muchos
- **VARRAY.** Es un T.O. con estructura similar a un vector (indizado), con un tamaño fijo en su declaración, aunque puede variarse de forma dinámica:

```
CREATE TYPE precios AS VARRAY(10) OF NUMBER(12,2);
```

- Pueden utilizarse como tipo de dato de un columna de una tabla relacional, como tipo del atributo de un T.O.; o, en PL/SQL, como variable, parámetro, valor de retorno de una función
- **Tablas Anidadas (Nested Tables).** Tipo Colección no ordenada de elementos del mismo tipo (nativo o T.O.)

```
CREATE TYPE tabla_lineas AS TABLE OF linea_pedido;
```

- Pueden utilizarse como tipo de dato de un columna de una tabla relacional, como tipo del atributo de un T.O.; o, en PL/SQL, como variable, parámetro, valor de retorno de una función

```
CREATE TABLE tabla_pedidos OF pedidos NESTED

TABLE lineas_pedidos STORE AS tabla_lineas_pedidos;
```

tabla_lineas_pedidos es la tabla anidada que almacena los valores de todos los atributos lineas_pedidos para todos los objetos pedidos de la tabla tabla_pedidos

- Herencia. En la actualidad (Oracle9i), soporta directamente la herencia (Las anteriores versiones no la soportaban, aunque era posible simularla en algunos casos concretos).

23. Diferencias entre la herencia en PostgreSQL y Oracle

Es de destacar que en Oracle se pueden usar las dos posibilidades (herencia de tipos y de tablas), mientras que en PostgreSQL solo se puede usar la herencia entre tablas.

PostgreSQL	Oracle 9i
Creamos cuatro tablas: La primera contendrá los datos de donde se ubica: CREATE TABLE direccion (direccion VARCHAR(30), codigo_postal VARCHAR(5), ciudad VARCHAR(15), provincia VARCHAR(15), pais VARCHAR(15)); Despues la tabla contenedora de las propiedades de cualquier sitio: CREATE TABLE residencia (id INTEGER, numero_habitaciones INTEGER, numero_baños INTEGER, metros_cuadrados INTEGER, PRIMARY KEY (id)); La tabla piso, tiene atributos de especialización e incorpora los de las tablas anteriores. CREATE TABLE piso (gastos_comunidad INTEGER)) INHERITS (residencia, direccion); La tabla atico, tiene atributos de especialización, e incorpora los de las tablas iniciales: CREATE TABLE atico (numero_plantas INTEGER, metros_terreno INTEGER)) INHERITS	Creamos un tipo domicilio: CREATE TYPE domicilio AS OBJECT (direccion VARCHAR(30), codigo_postal VARCHAR(5), ciudad VARCHAR(15), provincia VARCHAR(15), pais VARCHAR(15)); Ahora un objeto abstracto del que heredan los otros dos: CREATE TYPE residencia AS OBJECT (id INTEGER, numero_habitaciones INTEGER, numero_baños INTEGER, metros_cuadrados INTEGER) NOT FINAL NOT INSTANTIABLE; Ahora dos objetos que hereden del primero: CREATE TYPE piso UNDER residencia (gastos_comunidad INTEGER)) FINAL INSTANTIABLE; CREATE TYPE atico UNDER residencia (numero_plantas INTEGER, metros_terreno INTEGER)) FINAL INSTANTIABLE; Por ultimo dos tablas con estos nuevos tipos: CREATE TABLE t piso OF piso (PRIMARY KEY (codigo)); CREATE TABLE t_atico OF atico (PRIMARY KEY (codigo));

24. Comparación entre PostgreSQL y Oracle 9i

24.1. Características Generales

SQL3	PostgreSQL	Oracle 9i
UDT	No	Si
Tipos colecciones	Si	Si
Tipos referencia	No	Si
Tablas Tipadas	No	Si
Herencia	SI: solo de tablas Soporta herencia multiple	Si: de tipos y de tablas

24.2. Características especificas

Comparativa	PostgreSQL	Oracle 9i
Tipo de producto	Desarrollado en entornos académicos	Producto comercial
Versión probada	Versión 7.3 y 7.4	Versión 9i.
Precio	Distribución gratuita. El tipo de licencia de uso, hace que las nuevas versiones también tengan que ser de libre distribución.	Precio de adquisición alto. Se ha de pagar por cada cliente que utiliza el motor de la base de datos. Cada nueva versión conlleva una nueva adquisición.
Soporte técnico (privado)	Existen empresas especializadas que ofrecen un buen servicio técnico	El fabricante ofrece un buen servicio técnico.
Otros tipos de soporte	Soporte en forums de discusión mantenidos por la propia comunidad de desarrollo.	Al tener Oracle un buen soporte en su pagina web se echan en falta links no dependientes del fabricante.
Cantidad y calidad de la documentación	Los propios desarrolladores crean la documentación oficial que pegan en la web del proyecto. Los propios usuarios desarrollan casos prácticos y documentación adicional.	Oracle es un producto muy extenso, en consecuencia existe mucha documentación. Eso hace que cueste encontrar lo que se esta buscando.
Robustez	Es una BD que cumple las propiedades ACID para las transacciones.	Es una BD que cumple las propiedades ACID para las transacciones.
Otras características	Tiene procedimientos, disparadores, restricciones, replicaciones y soporta(con diferente sintaxis), todas las posibilidades del ANSI de SQL-92.	Tiene procedimientos, disparadores, restricciones, replicaciones y soporta(con diferente sintaxis), todas las posibilidades del ANSI de SQL-92.
Escalabilidad	No soporta bases de datos distribuidas pero funciona con sistemas de ato rendimiento y multiprocesadores	Soporta bases de datos distribuidas
Complejidad de la instalación y del mantenimiento del motor de la BD	Siguiendo las instrucciones que facilitan los desarrolladores del producto, la instalación sobre un SO Linux no tendría que tener ninguna dificultad. El mantenimiento posterior, se tendría que limitar a realizar las copias de seguridad, dado que a partir de la versión del PostgreSQL 7.4 el VACUUM se ejecuta periódicamente de forma automática.	No tiene ninguna dificultad y se puede hacer con una con una formación mínima.
Sistemas operativos sobre los que funciona	Unix, Linux, Aix, (entre otros) y la posibilidad de ejecutarse en el entorno Windows (con ciertas restricciones)	Funciona el los principales sistemas operativos del mercado (Windows, unix, Linux, Aix, etc.)

25. Otro gigante: Informix

Informix es uno de los cuatro grandes de las bases de datos junto DB2 de IBM,SQL Server de Microsoft y Oracle.

Aunque en muchos aspectos es mejor que Oracle, no se ha sabido mover en el terreno del marketing. Oracle capturó la mayor parte del mercado y Informix no se recuperó de las perdidas económicas. DB2 y SQL Server tenían grandes compañías detrás con otros negocios que les permitió aguantar la política agresiva de Oracle. Recientemente IBM adquirió Informix con lo que el mercado de las bases de datos comerciales en UNIX (Linux) quedó entre IBM y Oracle.

Puedes encontrar una infinidad de información sobre Oracle sobre Linux en Internet, pero muy poca sobre Informix. La poca información es debido a la poca comunidad Internet que tiene Informix, al menos comparada con la de Oracle. Y es que, hoy en día, las documentaciones oficiales, de tan sencillas que quieren ser, cada vez son más confusas e incompletas. Sin duda, el mejor soporte técnico que hay para un producto es su comunidad de usuarios en Internet. Informix por desgracia no ha sabido crearla. Una búsqueda de "oracle linux" en Google devuelve unas 972.000 páginas, mientras que "informix linux"143.000.

25.1. Algunas diferencias con Oracle

Oracle siempre ha sido considerada una base de datos para uso más general que Informix. Informix por su lado, se especializó más en aplicaciones tipoGIS (datos geográficos), Datawarehouse y Datamining. Sin duda a los gurús, les agrada más Informix que Oracle.

En cuanto a precios, Informix tiene tendencia a ser más caro que Oracle en configuraciones parecidas.

En la practica, como buenos enemigos acérrimos, ambas tienen parecidas características y funcionalidades. Cada una tiene las típicas ampliaciones que permiten especializar la base de datos a un cierto tipo de aplicación (en Informix se llaman DataBlades). Para competir, en cada nueva versión que sacaban, las dos iban añadiendo de serie muchas de las extensiones que en la versión anterior eran opcionales (de pago por separado).

De cara a una instalación y configuración básicas, las diferencias frente a Oracle apreciables son dos:

- En Oracle, tienes que definir los usuarios dentro la base de datos (gestión interna de usuarios). En cambio, Informix utiliza los mismos usuarios de Linux (los que creas con adduser), simplificando la administración.

- Para conectar externamente vía TCP/IP a Oracle, hacía falta colgar un daemon llamado listener de un puerto. Luego el listener traducía las llamadas al SQLNet para hablar con la base de datos. Luego había dos procesos: el daemon de la base de datos (oracle), y el daemon que escuchaba el puerto (listener). En Informix, el mismo daemon de la base de datos (ONINIT) atiende los puertos. De esta manera Informix ocupa menos memoria y recursos.

Con estas diferencias ya podemos ver, que Informix es una base de datos más moderna y integrada con Linux que Oracle. Pero los precios y el marketing ha influido notablemente.

25.2. Objetivos que se han cubierto en Oracle

La sintaxis del lenguaje es casi idéntica a la de Informix-4GL. Se han implementado los tipos de datos básicos, los de fecha y hora, intervalos y "date-time".

Las sentencias de asignación, presentación en pantalla, estructuras repetitivas (bucles while y for) y condicionales (if..else..), generación de menús y listados se han implementado completamente.

El manejo de los formularios de entrada de datos se ha implementado siguiendo la sintaxis original de Informix-4GL, sin embargo se ha modificado la sintaxis de la definición de los formularios para dar cabida a nuevos tipos de objetos, como botones o casillas de verificación.

Se ha introducido una nueva sentencia (browse), que permite visualizar todas las tuplas devueltas por una consulta y 'navegar' por ellas mediante movimientos del cursor.

La sintaxis de algunas sentencias se ha ampliado, permitiendo nuevas funcionalidades no encontradas en Informix-4GL, como asignaciones múltiples. Estas modificaciones están explicadas con detalle en el manual de usuario.

25.3. Posibilidades de ampliación

Se ha generado únicamente una interfaz en modo texto. Sin embargo, la implementación de las funciones de manejo de formularios de entrada se ha realizado de manera independiente de la implementación a bajo nivel (curses).Gracias a esto es posible desarrollar librerías de funciones que gestionen los formularios de entrada de manera gráfica, generando interfaces en X-Window.

La generación de código está actualmente muy enfocada a la interfaz de PostgreSQL. Una posible ampliación sería la generalización de este código, para permitir el desarrollo de aplicaciones que accedan a datos almacenados en otro tipo de servidores, o incluso usando ODBC (el API de acceso a bases de datos definido por Microsoft y que se ha convertido en estándar).

25.4. Tabla comparativa de Oracle con Informix

Puntos de Interés	Oracle	Informix
Robustez	Muy estandarizado en la industria	No tan robusto pero muy bueno
Vida del producto	Está en proceso. La documentación está planeada para el 2010.	Cuestionable. IBM muestra el plan para el 2004.
Tiempo de configuración	Se requiere más tiempo que con Informix, pero todavía es razonable.	Mínimo.
Bloqueos	Ninguno en "Sirsi" una vez que se esté ejecutando la base de datos..	Muy pocos para "Sirsi". Pero ha habido unos cuantos.
Mantenimiento de la base de datos	Mínima para el usuario final. La mayoría del mantenimiento se hace como parte de "Sirsi Software Upgrades".	Mínima para el usuario final. La mayoría del mantenimiento se hace como parte de "Sirsi Software Upgrades"
Acceso API	Acceso vía Sirsi API.	Acceso vía Sirsi API.
Acceso a consultas vía SQL	Si, ambos leen y escriben en la próxima versión liberada (Primavera 2003).Trabajará con una amplia variedad de otros grupos de productos.	No, no está planeado abrir SQL a llamadas o escrituras.
Acceso limitado de SQL	Ficheros de seguridad y accesos privilegiados.	Ficheros de seguridad y accesos privilegiados.
Acceso a lectura y escritura	Si, vía API. Si, vía Oracle, ambos leen y escriben en la próxima versión liberada.	Si, vía API. Lee sólo vía Informix en otros grupos de productos. No está planeado que escriba.
Unicode para CJK	Si	No
Coste	$ 15,000.00	Incluido en el precio base.
Mantenimiento anual	$ 2,850.00	Incluido en el coste de mantenimiento base.

26. Caso practico

Ahora se implementa un caso práctico que muestra lo planteado y discutido en el trabajo previo.

27. Enunciado

Un club de rol ha diseñado una pequeña BD que le permite guardar datos destinados a la gestión de las distintas partidas que juegan los miembros del club. En concreto, se han diseñado las siguientes tablas:

27.1. PARTIDAS

Esta tabla guardará información de las partidas que se juegan en las distintas jornadas. Concretamente, cada partida se identifica por un título. Además deseamos saber en qué fecha se juega, cuántos héroes participan (no existen límites en el número de participantes de una partida), cuántos sobreviven (este dato se desconoce hasta que la partida no se ha jugado), y cuántos puntos de experiencia puede tener como máximo un héroe en las partidas jugadas anteriormente para poder participar en ésta. Las columnas de la tabla son:

Título	CHAR(80)	PRIMARY KEY
Fecha	DATE	NOT NULL
Num_participantes	INTEGER	NOT NULL
Num_sobrevivientes	INTEGER	
Puntos_máximos	INTEGER	NOT NULL

27.2. HEROES

Esta tabla contiene todos los personajes que pueden participar en las partidas organizadas en las jornadas. Para cada héroe, además de su nombre, queremos saber su clase (guerrero, mago, ladrón, etc.), raza (humano, elfo, duende, etc.), y (si procede y se conoce) el arma en la cual está especializado (espada, arco, hacha, etc.) Sus columnas son:

Codigo	INTEGER	PRIMARY KEY
Nombre	CHAR(30)	NOT NULL
Clase	CHAR(10)	NOT NULL
Raza	CHAR(10)	NOT NULL
Arma	CHAR(10)	

Las subtablas que heredan de esta son los diferentes tipos de personajes pertenecientes a cualquie rjuego de rol: Clerigo, Mago,Guerrero y Ladrón.

27.3. PARTICIPACIONES

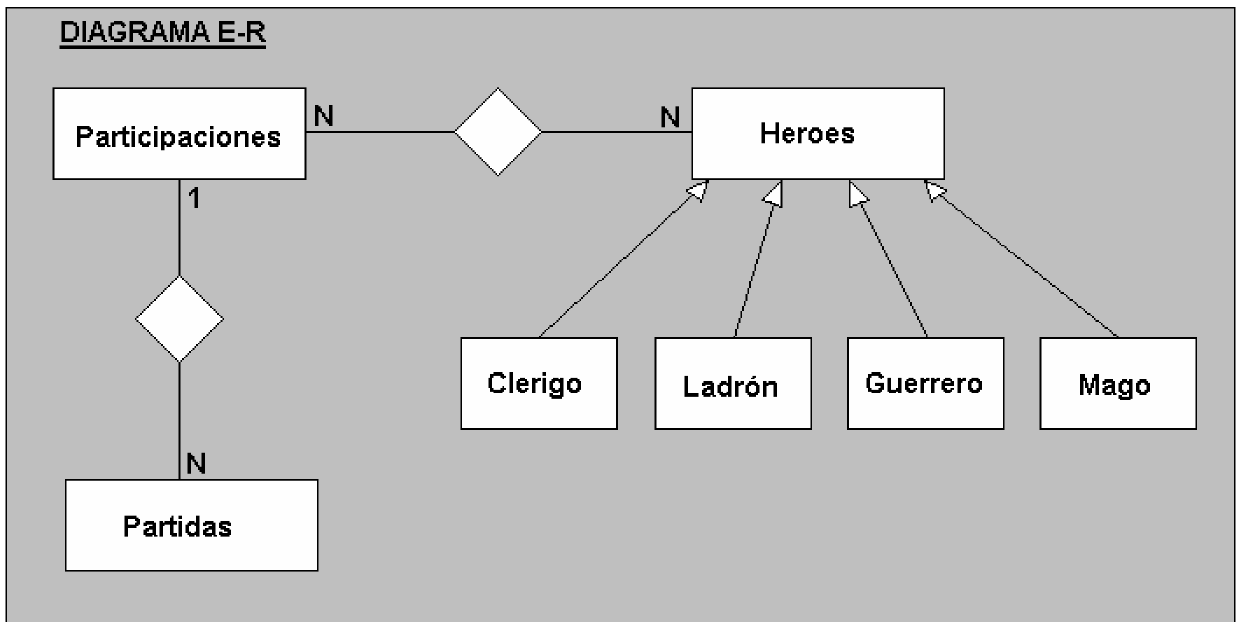
En esta tabla se almacenan las participaciones de cada héroe en las diferentes partidas, los puntos de experiencia que ha conseguido en la partida y si ha muerto o no tras jugar la partida. Sus columnas son:

Heroe	CHAR(30)	clave foránea a HÉROES
Partida	CHAR(80)	clave foránea a PARTIDA
Puntos	INTEGER	
Muerto	CHAR(1)	toma valores S (ha muerto), N (no ha muerto) o null

PRIMARY KEY(Heroe, Partida)

Por lo tanto, la tabla participaciones guarda inicialmente en qué partidas están inscritos los diferentes héroes. Mientras la partida no se ha jugado, las participaciones de héroes a la partida considera valores nulos para los atributos puntos y muerto. Una vez la partida se ha jugado, se registra cómo ha acabado la participación de los diferentes héroes inscritos en la partida. Esto implica, para cada héroe que ha participado en la partida, dar valores a los atributos puntos y muerto.

28. Modelo E-R



Para dar ejemplo de las posibilidades y carencias de las tres bases de datos, Se inserta seguidamente el código necesario para crear una estructura de datos que sirve para almacenar información referente a la practica.

El ejemplo se ha simplificado hasta dejar nada más los atributos suficientes que sirven para comprobar el funcionamiento de los tipos y de la herencia.

Se puede observar que Oracle ha conseguido utilizar ambas posibilidades, mientras que PostgreSQL e Informix solamente han podido utilizar la herencia entre las tablas.

28.1. PostgreSQL

Creamos una serie de tablas:

```
CREATE TABLE heroe
(
Nombre VARCHAR(30),
Clase VARCHAR(10),
Raza VARCHAR(10),
)
```

Ahora la tabla contenedora:

```
CREATE TABLE armas
(
codigo INTEGER,
arma VARCHAR(10)
);
```

Las siguientes tablas tienen atributos de especialización e incorpora los datos de las tablas iniciales:

```
CREATE TABLE mago
(
Nivel INTEGER,
Tunica VARCHAR(10)
)
INHERITS (heroe, armas);
```

```
CREATE TABLE guerrero
(
Nivel INTEGER
)
INHERITS (heroe, armas);
```

```
CREATE TABLE clerigo
(
Nivel INTEGER,
dios VARCHAR(10)
)
INHERITS (heroe, armas);
```

```
CREATE TABLE ladron
(
Nivel INTEGER,
clan VARCHAR(10)
)
INHERITS (heroe, armas);
```

28.2. Oracle

Creo un tipo héroe

```
CREATE TYPE HeroeType AS OBJECT
(
Nombre VARCHAR(30),
Clase VARCHAR(10),
Raza VARCHAR(10),
)
```

Creo un objeto abstracto del que hereda el resto

```
CREATE TYPE heroe AS OBJECT
(
codigo NUMBER
Arma VARCHAR(10)
h HeroeType
)
NOT FINAL NOT INSTANCIABLE
```

Creo cuatro objetos que heredan del primero

```
CREATE TYPE mago UNDER heroe
(
nivel NUMBER,
tunica VARCHAR(10)
)
FINAL INSTANCIABLE;
```

```
CREATE TYPE guerrero UNDER heroe
(
nivel NUMBER
)
FINAL INSTANCIABLE;
```

```
CREATE TYPE clerigo UNDER heroe
(
nivel NUMBER,
dios VARCHAR(10)
)
FINAL INSTANCIABLE;
```

```
CREATE TYPE ladron UNDER heroe
(
nivel NUMBER,
clan VARCHAR(10)
)
FINAL INSTANCIABLE;
```

Creo cuatro tablas con estos nuevos tipos

```
CREATE TABLE m mago OF mago
(
PRIMARY KEY(codigo)
);
```

```
CREATE TABLE g guerrero OF guerrero
(
PRIMARY KEY(codigo)
);
```

```
CREATE TABLE c clerigo OF clerigo
(
PRIMARY KEY(codigo)
);
```

```
CREATE TABLE l ladron OF ladron
(
PRIMARY KEY(codigo)
);
```

28.3. Informix

Creamos una serie de tablas:

```
CREATE TABLE heroe
(
codigo INTEGER,
Nombre CHAR(30),
Clase CHAR(10),
Raza CHAR(10),
)
```

Ahora la tabla contenedora:

```
CREATE TABLE armas
(
codigo INTEGER,
arma CHAR(10)
);
```

Las siguientes tablas tienen atributos de especialización e incorpora los datos de las tablas iniciales:

```
CREATE TABLE mago
(
Codigo INTEGER,
Nivel INTEGER,
Tunica VARCHAR(10)
)
```

```
CREATE TABLE guerrero
(
Codigo INTEGER,
Nivel INTEGER
)
```

```
CREATE TABLE clerigo
(
Codigo INTEGER,
Nivel INTEGER,
dios VARCHAR(10)
)
```

```
CREATE TABLE ladron
(
Codigo INTEGER,
Nivel INTEGER,
clan VARCHAR(10))
```

29. Comparación de las características del SQL:1999 entre los productos Oracle y PostgreeSQL

29.1. UDT

Tal como se ha comentado anteriormente SQL:1999 define los UDT como una característica para poder encapsular unos ciertos tipos de datos.

Hemos observado que en Oracle 9i esta característica nos da muchas posibilidades, desde adaptar a nuestras necesidades un tipo existente, a confeccionar un tipo con diferentes atributos relacionados.

En el primer caso, tendríamos un buen ejemplo si definiésemos dos tipos, para diferenciar los metros lineales de los metros cuadrados, dado que querríamos evitar las operaciones entre ellos.

En el segundo caso, tendríamos un nuevo tipo definido, para tener todas las propiedades de una dirección (por ejemplo) englobadas en un sólo tipo.

A diferencia de Oracle, PostgreSQL no tiene la capacidad de crear UDTs. Ni tan sólo soporta la creación de UTDs para poder distinguir dos atributos que comparten el mismo tipo de datos, pero que conceptualmente describen conceptos diferentes de la realidad. Habría que poder distinguir dos atributos de tipo entero que representen cosas diferentes (longitudes y peso, por ejemplo).

En PostgreSQL también se ha encontrado en falta el poder definir columnas de tipo complejo. La facilidad que proporciona el disponer de UTDs es grande, ya que por ejemplo, en el mundo real al hablar de una dirección, no se piensa en un tipo de calle (avenida, carretera), en el nombre de la calle, en el código postal , en la población y provincia dónde está situada. Al hacer mención de una calle, se está pensando en una ubicación que trae incorporada todos los atributos nonbrados, y que abstractamente se describe.

Al nombrar la implementación del caso práctico en PostgreSQL, se ha tenido en cuenta la utilización de los UTDs para describir tipos complejos de datos, quedando la nombrada implementación muy similar al equivalente en el modelo relacional.

29.2. Colecciones

Estos tipos están soportados por los dos productos y al observar el funcionamiento de las colecciones en Oracle 9i, podemos comprobar que sigue bastante fielmente las recomendaciones hechas en SQL:1999. En el caso práctico se ha utilizado en dos ocasiones las raices y ha sido de utilidad para poder mejorar el modelo de datos sin tener que hacer que éste gane en complejidad, ya que se ha evitado la utilización de tablas adicionales. Por otro lado podemos tener un atributo multievaluado, aún que no cumpla una de las reglas fundamentales del modelo relacional, en el que se dice que cada atributo tiene que ser atómico.

Se tiene que observar que durante la implementación en PostgreSQL se ha descubierto un error de funcionamiento en la base de datos. PostgreSQL siempre crea vectores de longitud limitada, aún que se indique la longitud máxima.

La creación de las colecciones siguen las recomendaciones del standard SQL:1999, y no se han observado divergencias significativas, excepto en la implementación que se ha hecho del caso desarrollado, muy útiles, ya que ha permitido simplificar las relaciones del caso descrito.

29.3. Tipo referencia

En PostgreSQL también echamos en falta el tipo referencia. Este tipo se utilizaría en los casos en que se teniese que identificar las columnas de alguna tabla de tipos. Un valor de tipo referencia puede ser almacenado en una tabla y utilizado en una fila específica de otra tabla, como si fuera un puntero lógico.

29.4. Tabla de tipos

Tal como ya se ha observado, PostgreSQL no soporta tipos complejos de datos. Como resultado de esta limitación, no soporta las tablas de tipo.

En Oracle 9i si que es posible implementar este tipo de tablas dado que éstas sirven para instanciar tipos abstractos de datos.

29.5. Herencias en las características previas

Tanto en Oracle 9i como en PostgreSQL coinciden en el hecho de tener herencia de tablas. Adicionalmente Oracle 9i soporta herencia de tipos. Este hecho es importante, ya que los desarrollos actuales dan importancia a este hecho, haciendo un uso de esta característica.

El funcionamiento en PostgreSQL diverge notablemente, ya que funciona en forma de árbol jerárquico de tablas. A las tablas que heredan se les añade los atributos de las tablas padre. Esta sencillez de funcionamiento diverge del descrito en es standard SQL:1999, pero simplifica enormemente el diseño. Por otra parte, un diseño sencillo, no quiere decir que sea óptimo, ya que se puede comprobar al intentar implementar algunos casos más complejos, que nos siguen faltando los UDT.
En este punto, se tiene que recordar que PostgreSQL tiene un error en la implementación de la herencia de las tablas, que hace que se puedan insertar datos duplicados en una columna UNIQUE o en una clave primaria de una tabla padre, desde una tabla hija.
En Oracle 9i tenemos la posibilidad de limitar si una tabla o un tipo intermedio en la jerarquía de la herencia es instanciable o es una clase final ya que existen las cláusulas FINAL y NOT FINAL, y las de INSTANCIABLE y NOT INSTANCIABLE. Esta diferenciación no se puede llevar a termino en PostgreSQL. En este se puede hacer en cualquier momento que una tabla sea padre de una tabla nueva, que sería hija. Este hecho también conlleva que se pueda insertar datos en cualquier tabla de la jerarquía de la herencia, haciendo difícil el mantenimiento de la coherencia de datos, ya que al crear unas tablas nuevas hijas, dependiendo de una principal, se tendrá que tener especial cuidado de que se puedan o no insertar datos en la primera.

30. Conclusiones y futuros trabajos a desarrollar

Actualmente la tendencia en el mundo de la informática en general y los sistemas gestores de datos en particular, es la orientación a objetos, tal y como se ha visto durante el trabajo.

La herencia, el polimorfismo y la reutilización son algunas de las ventajas de este paradigma de la programación que abre nuevas e innumerables posibilidades, de las cuales solamente se han explorado algunas en este proyecto.

Se han dejado muchas otras sin ver que podrían dar lugar a otros interesantes trabajos, como por ejemplo: la siguiente versión de SQL:2003, de la que en este proyecto sólo se ve una breve introducción, o la multimedia en el entorno de las bases de datos, por nombrar algunos.

Estas ventajosas características de la orientación a objetos quedan patentes al comparar los tres SGBD, Oracle, Informix y PostgreSQL, con el estándar SQL3.

Aunque de los tres, el que sobresale por implementar de forma completa la herencia múltiple es Oracle.
Informix, que cumple la mayor parte de las características del SQL3, ha quedado en un segundo plano, después de ser absorbido por parte de IBM. Se puede hacer hincapié en la importancia del marketing y la publicidad en determinados entornos de la informática.

También hemos podido observar algunas características relevantes entre Oracle e Infomix como por ejemplo la estandarización de la primera en la industria frente a su desventaja en robustez de la segunda, o el tiempo de configuración en Informix que es mínimo. Aunque como acérrimos enemigos también tienen cosas en común, como por ejemplo el mínimo mantenimiento de ambos.

Como hemos podido ver, Oracle se ajusta a la definición del estándar más que PostgreeSQL, que al ser un producto desarrollado por una comunidad de código abierto está todavía en desarrollo y con algunas características sin implementar o con algunos errores, como es el caso de la herencia o la herencia múltiple, todavía sin documentar.

Oracle y Postgree tienen en común respecto al estándar los tipos colecciones y la herencia solo en tablas. Queda muy claro la falta de desarrollo en la herencia en PostgreSQL, además de otras características como por ejemplo, las tablas tipadas, tipos referencias o UDT.

Una posible continuación de este trabajo podría ser la documentación y ampliación de PostgreeSQL en alguna de sus características menos desarrolladas.

31. Bibliografía

31.1. Artículos

SQL:1999, formerly known as SQL3.
Andew Eisenberg
Sybase, Condord, MA 01742
andrew.eisenberg@sybase.com
Jim Meltom
Sandy, UT 84093
jim.melton@acm.org
SIGNOD Record, Vol. 28, Nº 1, March 1999

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

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

31.2. Documentos

Material de la asignatura: Prácticas de Modelos Avanzados de BD - Ingeniería uperior de Informática.
Autores: Juan Miguel Medina Rodríguez, Olga Pons Capote, M. Amparo Vila Miranda
Dpto. Ciencias de la Computación e I. A.Universidad de Granada

31.3. Libros

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

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

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

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

31.4. Enlaces

Documentación sobre PostgreSQL

http://www.postgresql.com
Web oficial del equipo de desarrollo del SGBD PostgreSQL.

http://es.tldp.org/Postgresql-es/web/navegable/todopostgresql/postgres.htm
Manual del PostgreSQL en castellano (format HTML), pegado en la web del equipo LUCAS (trabajan para conseguir publicar documentos, normalmente de de software GNU, en castellano).

Documentación sobre Oracle

http://www.oracle.com
Web oficial de soporte para Oracle

Documentación sobre Informix

http://www.ibm.com
Web oficial de soporte para Informix

http://www.ilustrados.com
Web sobre trabajos de investigación y afines

31.5. TFC

TCP - Avaluació de l'standard SQL:1999 respecte les característiques orientades a l'objecte que suporta i la seva implementació al SGBD object-relational.
autora: Miracle Cerón Mercadé
dirigit per: M. Elena Rodríguez González
Curs: Setembre 2003 - Gener 2004

TCP - Avaluació de l'standard SQL:1999 respecte les característiques orientades a l'objecte que suporta i la seva implementació al SGBD object-relational.
autor: Ivo Plana Vallvé
dirigit per: Alex Alfonso Minguillon
Curs: Setembre 2003 - Gener 2004