Universidad Nacional Autónoma de México
Facultad de Contaduría y Administración
Sistema Universidad Abierta y Educación a Distancia

Licenciatura en Informática

Desarrollo de Aplicaciones en Manejadores de Bases de Datos Relacionales

Apunte electrónico

SUAyED

COLABORADORES

DIRECTOR DE LA FCA
Dr. Juan Alberto Adam Siade

SECRETARIO GENERAL
L.C. y E.F. Leonel Sebastián Chavarría

COORDINACIÓN GENERAL
Mtra. Gabriela Montero Montiel
Jefe de la División SUAyED-FCA-UNAM

COORDINACIÓN ACADÉMICA
Mtro. Francisco Hernández Mendoza
FCA-UNAM

COAUTORES
Lic. Armando Carlos Rojas Marín
Lic. Carlos Francisco Mendez Cruz

DISEÑO INSTRUCCIONAL
Lic. Paola Hernández León

CORRECCIÓN DE ESTILO
Mtro. Francisco Vladimir Aceves Gaytán

DISEÑO DE PORTADAS
L.CG. Ricardo Alberto Báez Caballero
Mtra. Marlene Olga Ramírez Chavero
L.DP. Ethel Alejandra Butrón Gutiérrez

DISEÑO EDITORIAL
Mtra. Marlene Olga Ramírez Chavero

OBJETIVO GENERAL

Al finalizar el curso, el alumno será capaz de desarrollar aplicaciones con un manejador de base de datos, haciendo uso de los conceptos teóricos correspondientes.

TEMARIO OFICIAL

(64 horas)

	Horas
1. Planeación de la base de datos	8
2. Construcción de la base de datos	12
3. Características avanzadas	12
4. Consultas	12
5. Administración	10
6. Construcción de la aplicación	10
TOTAL	**64**

INTRODUCCIÓN

El PostgreSQL fue desarrollado originalmente en la Universidad de California en Berkeley. Está basado en Postgres release 4.2. El proyecto Postgres, lidereado por el Profesor Michael Stonebraker, fue financiado por diversos organismos oficiales u oficiosos de los EE.UU: la Agencia de Proyectos de Investigación Avanzada de la Defensa de los EEUU (DARPA), la oficina de investigación de la Armada (ARO), la Fundación Nacional para la Ciencia (NSF) y ESL, Inc.

¿Qué es Postgres?

Los sistemas de mantenimiento de Bases de Datos relacionales (DBMS) soportan un modelo de datos que consiste en una colección de relaciones con nombre, que contienen atributos de un tipo específico. En los sistemas comerciales actuales, los tipos numéricos posibles incluyen: de punto flotante, enteros, cadenas de caracteres, cantidades monetarias y fechas. El modelo relacional sustituyó modelos previos en parte por su “simplicidad”. Sin embargo, como se ha mencionado, esta simplicidad también hace muy difícil la implementación de ciertas aplicaciones. Postgres ofrece una fortaleza adicional al incorporar los siguientes cuatro conceptos adicionales básicos en una vía en la que los usuarios pueden extender fácilmente el sistema.

- Clases
- Herencia
- Tipos
- Funciones

Otras características aportan eficacia y flexibilidad adicional:

- Restricciones (*Constraints*)
- Disparadores (*Triggers*)
- Reglas (*Rules*).

Integridad transaccional

Estas características colocan a Postgres en la categoría de las Bases de datos identificadas como objeto-relacionales. Nótese que éstas son diferentes de las referidas como orientadas a objetos, que en general no son bien aprovechadas para soportar lenguajes de Bases de Datos relacionales tradicionales. Postgres tiene algunas características que son propias del mundo de las bases de datos orientadas a objetos. De hecho, algunas Bases de Datos comerciales han incorporado recientemente características en las que Postgres fue pionera.

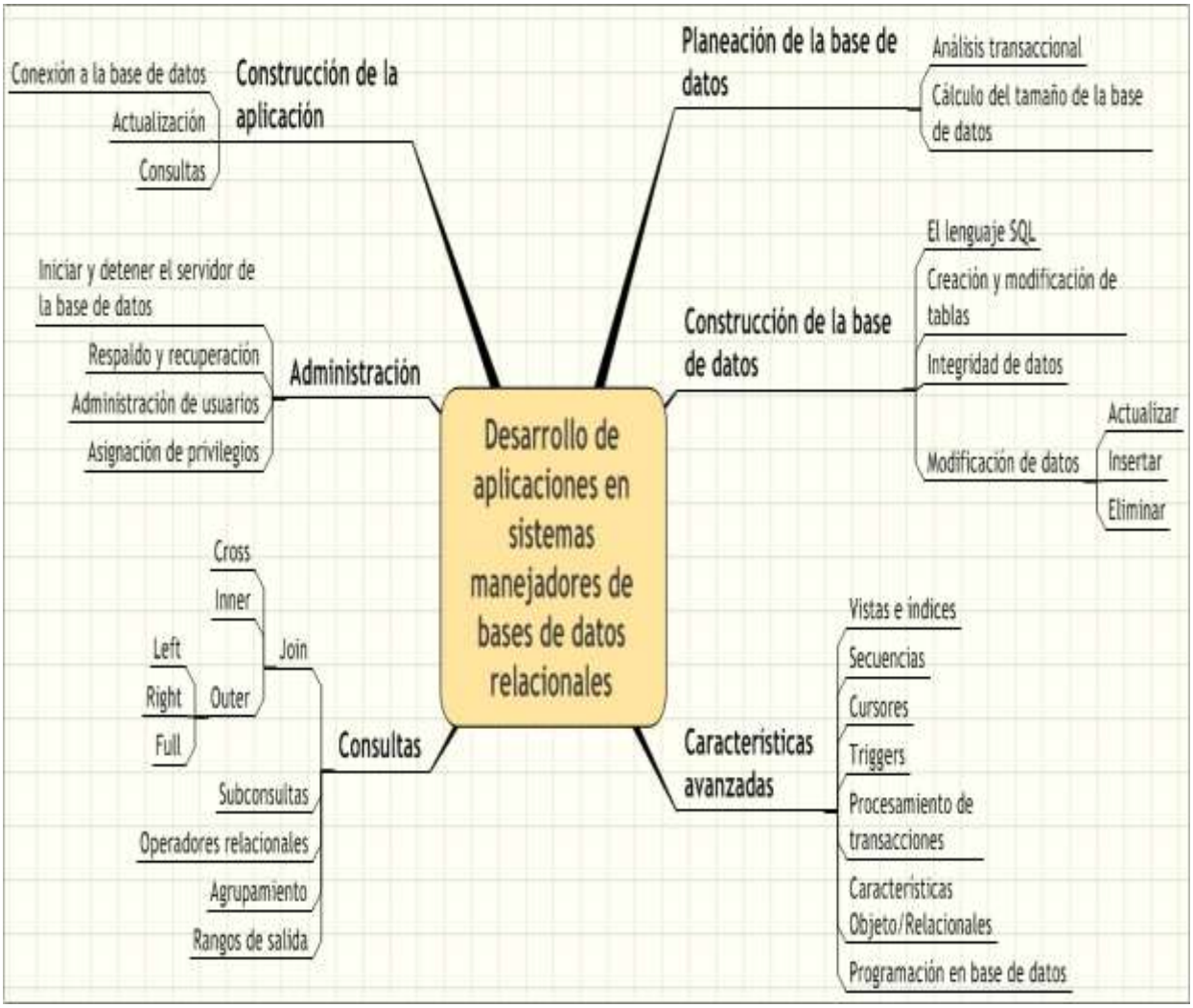
Breve historia Postgres

El Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos conocido como PostgreSQL (y brevemente llamado Postgres95) está derivado del paquete Postgres escrito en Berkeley. Con una década de desarrollo tras él, PostgreSQL es el gestor de bases de datos de código abierto más avanzado hoy en día, ofreciendo control de concurrencia multi-versión, soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones, tipos y funciones definidas por el usuario), contando también con un amplio conjunto de enlaces con lenguajes de programación (incluyendo C, C++, Java, perl, tcl y python).

El proyecto Postgres de Berkeley

La implementación del DBMS Postgres comenzó en 1986. Los conceptos iniciales para el sistema fueron presentados en The Design of Postgres y la definición del modelo de datos inicial apareció en The Postgres Data Model. El diseño del sistema de reglas fue descrito en ese momento en The Design of the Postgres Rules System. La lógica y arquitectura del gestor de almacenamiento fueron detalladas en The Postgres Storage System.

ESTRUCTURA CONCEPTUAL



Unidad 1
Planeación de la base de datos

OBJETIVO PARTICULAR

El alumno planeará la base de datos de una empresa para el desempeño eficiente de sus actividades, anticipando lo que debe de hacerse, cuándo y quién lo hará.

TEMARIO DETALLADO

(8-horas)

1. Planeación de la base de datos

1.1. Análisis transaccional

1.2. Cálculo del tamaño de la base de datos

INTRODUCCION

A diferencia de la mayoría de otros sistemas de bases de datos que usan bloqueos para el control de concurrencia, Postgres mantiene la consistencia de los datos con un modelo multiversión. Esto significa que mientras se consulta una base de datos, cada transacción ve una imagen de los datos (una versión de la base de datos) como si fuera tiempo atrás, sin tener en cuenta el estado actual de los datos que hay por debajo. Esto evita que la transacción vea datos inconsistentes que pueden ser causados por la actualización de otra transacción concurrente en la misma fila de datos, proporcionando aislamiento transaccional para cada sesión de la base de datos.

La principal diferencia entre multiversión y el modelo de bloqueo es que en los bloqueos MVCC derivados de una consulta (lectura) de datos no entran en conflicto con los bloqueos derivados de la escritura de datos y de este modo la lectura nunca bloquea la escritura y la escritura nunca bloquea la A diferencia de la mayoría de otros sistemas de bases de datos que usan bloqueos para el control de concurrencia, Postgres mantiene la consistencia de los datos con un modelo multiversión. Esto significa que mientras se consulta una base de datos, cada transacción ve una imagen de los datos (una versión de la base de datos) como si fuera tiempo atrás, sin tener en cuenta el estado actual de los datos que hay por debajo. Esto evita que la transacción vea datos inconsistentes que pueden ser causados por la actualización de otra transacción concurrente en la misma fila de datos, proporcionando aislamiento transaccional para cada sesión de la base de datos.

La principal diferencia entre multiversión y el modelo de bloqueo es que en los bloqueos MVCC derivados de una consulta (lectura) de datos no entran en conflicto con los bloqueos derivados de la escritura de datos y de este modo la lectura nunca bloquea la escritura y la escritura nunca bloquea la lectura.

1.1. Análisis transaccional

***Multi-Version Concurrency Control* (Control de la Concurrencia Multi Versión), MVCC,**

Es una técnica avanzada para mejorar las prestaciones de una base de datos en un entorno multiusuario. Vadim Mikheev ha proporcionado la implantación para Postgres.

Aislamiento transaccional

El estándar ANSI/ISO SQL define cuatro niveles de aislamiento transaccional en función de tres casos que deben ser tomados en cuenta transacciones concurrentes. Estos eventos no deseados son:

Lecturas “sucias”

Una transacción lee datos escritos por una transacción no esperada. No están en proceso.

Lectura no repetible

Una transacción vuelve a leer datos que previamente había leído y encuentra que han sido modificados por una transacción en proceso.

Lectura “fantasma”

Es una transacción que regresa a ejecutar una consulta, devolviendo un conjunto de filas que satisface una condición de búsqueda y encuentra que otras filas que satisfacen la condición han sido insertadas por otra transacción procesada.

Los cuatro niveles de aislamiento y sus correspondientes acciones se describen más abajo.

	Lectura “sucia”	Lectura no repetible	Lectura “fantasma”
Lectura no procesada	Posible	Posible	Posible
Lectura procesada	No posible	Posible	Posible
Lectura repetida	No posible	No posible	Posible
Secuencial	No posible	No posible	No posible

Tabla 1. Niveles de aislamiento de Postgres

Postgres ofrece lectura procesada y niveles de aislamiento secuencial.

Nivel de lectura procesada

Lectura procesada es el nivel de aislamiento por dafault en Postgres. Cuando una transacción se ejecuta en este nivel, la consulta solo ve datos procesados antes de que la consulta inicie y nunca ve ni datos “sucios” ni los cambios en transacciones concurrentes procesados durante la ejecución de la consulta.

Si una fila es devuelta por una consulta mientras se ejecuta una declaración UPDATE (o DELETE, o SELECT, o FOR UPDATE) está siendo actualizada por una transacción concurrente no procesada, entonces la segunda transacción que intente actualizar esta fila esperará a que la otra transacción se procese o pare. En caso de que pare, la transacción que espera puede proceder a cambiar la fila. En caso de que se procese (y si la fila todavía existe, por ejemplo, no ha sido borrada por la otra transacción), la consulta será reejecutada para esta fila y se comprobará que la nueva fila satisface la condición de búsqueda de la consulta. Si la nueva versión de la fila satisface la condición será actualizada (o borrada, o marcada para ser actualizada).

Hay que tener en cuenta que los resultados de la ejecución de SELECT o INSERT (con una consulta) no se verán afectados por transacciones concurrentes.

Nivel de aislamiento secuencial

La secuencia proporciona el nivel más alto de aislamiento transaccional.

Cuando una transacción está en el nivel secuencial, la consulta solo ve los datos procesados antes de que la transacción comience y nunca ve ni datos sucios ni los cambios de transacciones concurrentes procesados durante la ejecución de la transacción. Por lo tanto, este nivel emula la ejecución de transacciones en serie, como si las transacciones fueran ejecutadas una detrás de otra, en serie, en lugar de concurrentemente.

Si una fila es devuelta por una consulta durante la ejecución de una declaración UPDATE (o DELETE, o SELECT FOR UPDATE) está siendo actualizada por una transacción concurrente no procesada, la segunda transacción que trata de actualizar esta fila esperará a que la otra transacción se procese o pare. En caso de que pare, la transacción que espera puede proceder a cambiar la fila. En una transacción concurrente que se procese, una transacción secuencial será parada con el mensaje:

ERROR: Can`t seríalize access due to concurrent update

Porque una transacción secuencial no puede modificar filas cambiadas por otras transacciones después de que la transacción secuencial haya empezado.

Hay que tener en cuenta que los resultados de la ejecución de SELECT o INSERT (con una consulta) no se verán afectados por transacciones concurrentes.

1.2. Cálculo del tamaño de la base de datos

El tamaño de la base de datos depende de su aplicación, así como del número de usuarios y elementos.

Un espacio de tabla de 6 GB es más que suficiente para la mayoría de las instalaciones. Muchos usuarios pueden tener una instalación funcionando con espacios de tablas menores. Es necesario un administrador de base de datos Oracle (DBA) experimentado para evaluar el tamaño requerido. La siguiente fórmula debe usarse para determinar el tamaño de la base de datos requerida:

192 KB por sistema cliente
64 MB por canal

Por ejemplo, un servidor empresarial con 10 canales que está sirviendo 10,000 sistemas requeriría 1.92 GB para sus clientes y 640 MB para sus canales. Si se van a establecer canales personalizados para probar y mostrar los paquetes, éstos deben ser incluidos en esta fórmula.

Recuerda, las necesidades de almacenamiento de la base de datos puede crecer rápidamente dependiendo de los siguientes factores:

- El número de paquetes públicos importados (generalmente: 5000)
- El número de paquetes privados a ser administrados (generalmente: 500)
- El número de sistemas por administrarse (generalmente: 1000)

- El número de paquetes instalados en el sistema promedio (generalmente: 500)

Aunque debes ser generoso a la hora de estimar el tamaño de tu base de datos, considera asimismo que el tamaño afecta el tiempo de ejecución en la conducción de copias de seguridad y añade cargas de trabajo a otros recursos del sistema. Si la base de datos está siendo compartida, tu hardware y necesidades de espacio dependen enteramente de los otros elementos que la utilizan.

La base de datos Oracle debe tener un usuario asignado con acceso completo a DDL y DML a ese espacio de tabla por default del usuario. El usuario necesitará información de conexión estándar para la base de datos al momento de la instalación.

Los niveles de acceso requeridos por el usuario Oracle son los siguientes:

- ALTER SESSION
- CREATE SEQUENCE
- CREATE SYNONYM
- CREATE TABLE
- CREATE VIEW
- CREATE PROCEDURE
- CREATE TRIGGER
- CREATE TYPE
- CREATE SESSION

Entre los requerimientos adicionales para la base de datos se incluyen:

- Identificadores de seguridad (SID)
- Puertos de escucha

- Nombre de usuario
- Tamaño de extensión uniforme
- Administración automática de los espacios de segmento
- Juego de caracteres UTF-8

La disposición de disco en el servidor de la base de datos es independiente enteramente del usuario.

Arquitectura de ORACLE

La arquitectura de ORACLE tiene tres componentes elementales, la estructura de memoria para almacenar los datos y el código ejecutable, los procesos que corre el sistema de base de datos y las tareas de cada usuario conectado a la base de datos, y los archivos que sirven para el almacenamiento físico, en disco, de la información de la base de datos.

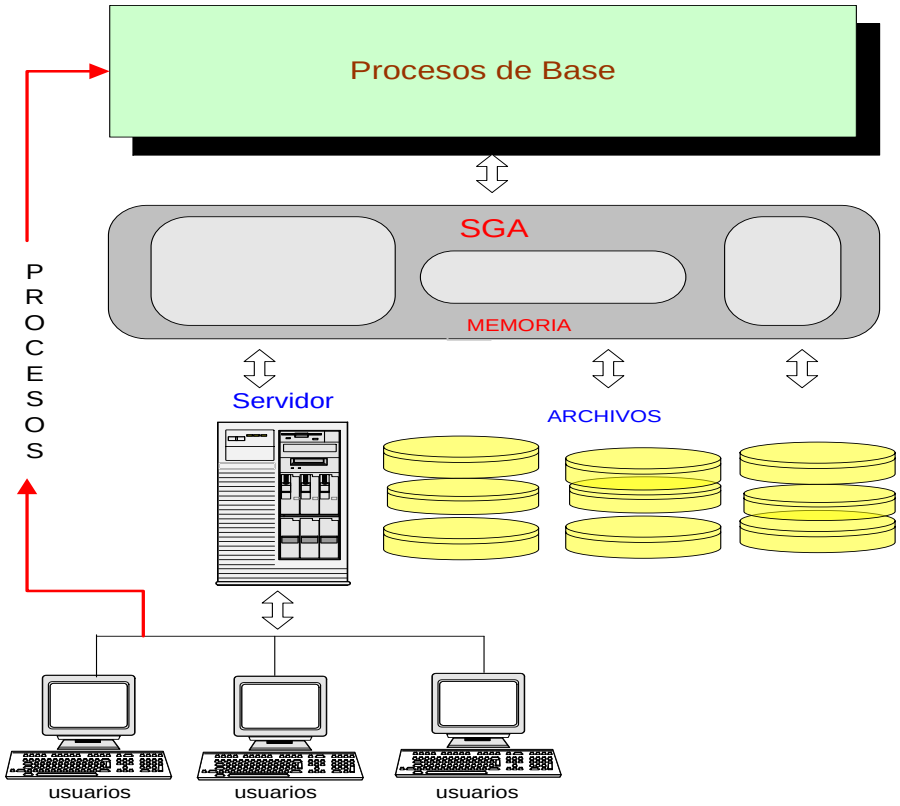


Figura 1.1. Arquitectura de ORACLE

Estructura de memoria

Hay dos clases de memoria, una de ellas compartida por todos los usuarios conectados y otra dedicada al trabajo de cada uno de ellos

El área global del sistema o SGA (*Systen Global Area*) es el área compartida por todos los usuarios y se divide en tres partes:

- Fondo común compartido (*Shared Spool*), en ella se conserva el diccionario de datos y las áreas compartidas de las órdenes SQL que se solicitan para su procesamiento.
- Área de memoria rápida (*Database Buffer Cache*), donde permanecen los datos traídos por las órdenes SQL de los usuarios conectados a la base de datos.
- Área de registros restaurados (*Redo Log Buffer*), aquí se registran los cambios hechos a la base de datos.

Por cada sesión de usuario se crea también, en memoria, un área específica llamada área global de programa o PGA (Programa Global Area), esta área no se comparte con las otras sesiones de usuario

Los archivos

Los archivos que maneja ORACLE se clasifican en cuatro grupos.

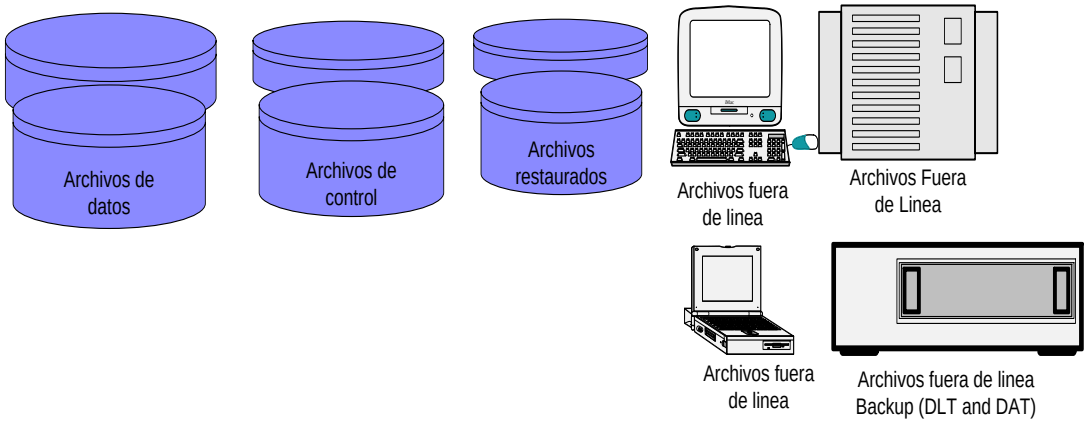


Figura 1.2. Tipos de archivos

Los archivos de datos (*Datafiles*): estos archivos sirven para el almacenamiento físico de las tablas e índices o agrupamientos (*clusters*) y procedimientos. Estos archivos son los únicos que contienen los datos de los usuarios de la base de datos.

Las unidades lógicas más grandes manejadas por ORACLE, para el almacenamiento de los datos, son llamados espacios de tablas (tablespaces), que le permite manejar o controlar el espacio en los discos.

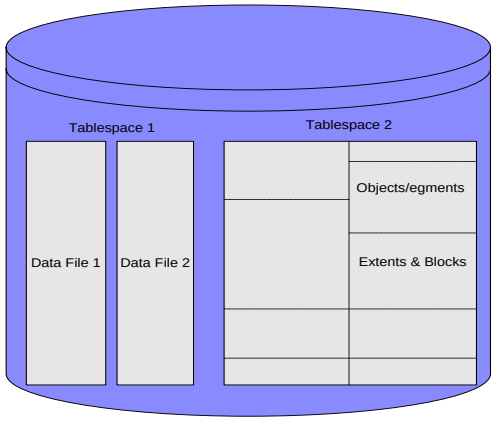


Figura 1.3. Archivos de datos

No es necesario que todos los espacios de tablas estén en un mismo disco. Cuando se crean en distintos discos se busca un mejor desempeño y mejor manejo del espacio de almacenamiento.

Una base de datos puede tener un solo espacio de tablas; pero, por las razones anteriores, se recomienda varios espacios de tablas. Como mínimo, se debe tener un espacio de tablas del sistema, un espacio de tablas por cada aplicación, un espacio de tablas para los usuarios y otro espacio de tablas para los índices.

El espacio de tablas SYSTEM se crea automáticamente cuando se crea una base de datos. Allí se guardan los archivos de control y el diccionario de datos y toda la información de los procedimientos almacenados.

El DBA puede crear un espacio de tablas con una orden, como la siguiente:

```
CREATE TABLESPACE indices
datafile ‘discod/db/datos1.dbf’ size 300m
```

Los archivos de datos (datafiles) almacenan los datos del usuario. Solo se requiere de uno para una base de datos. Sin embargo, los archivos de datos son fijos en tamaño e inalterables; cuando no haya espacio se debe adicionar más para incrementar el espacio de almacenamiento.

Cuando se agota el espacio, un DBA tiene dos alternativas:

a) Adicionar un nuevo archivo de datos, con la orden ALTER TABLESPACE:

```
ALTER TABLESPACE indices
Add datafile ‘discod/db/datos3.dbf’ size 150m;
```

b) Crear un Nuevo espacio de tablas, como se mostró previamente.

En el momento de la creación de una base de datos, el DBA debe planear o estimar los requerimientos de almacenamiento y también el nombre, tamaño y localización de los archivos de datos, junto con el número máximo de archivos de datos permitido para la base de datos.

El DBA puede crear varios espacios de tablas (*tablespace*) en discos separados para planear el crecimiento de la base de datos y hacer una mejor administración de la base de datos.

Un objeto de datos, por su parte, es una estructura lógica que puede ser una tabla, un archivo de índices, un archivo temporal, un archivo desorganizado o un cluster. Estos objetos se almacenan físicamente en segmentos que se componen de extensiones (*extents*).

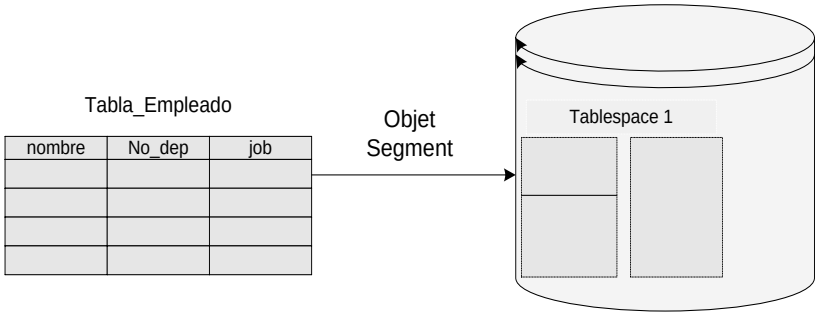


Figura 1.4. Espacio en tablas

A su vez, una extensión está hecha de bloques que, de acuerdo con el sistema operativo, puede tener un número determinado de bytes y que el DBA especifica en el momento de la creación de la base de datos. El tamaño del bloque es dependiente del sistema operativo y nunca puede ser menor al que éste maneja.

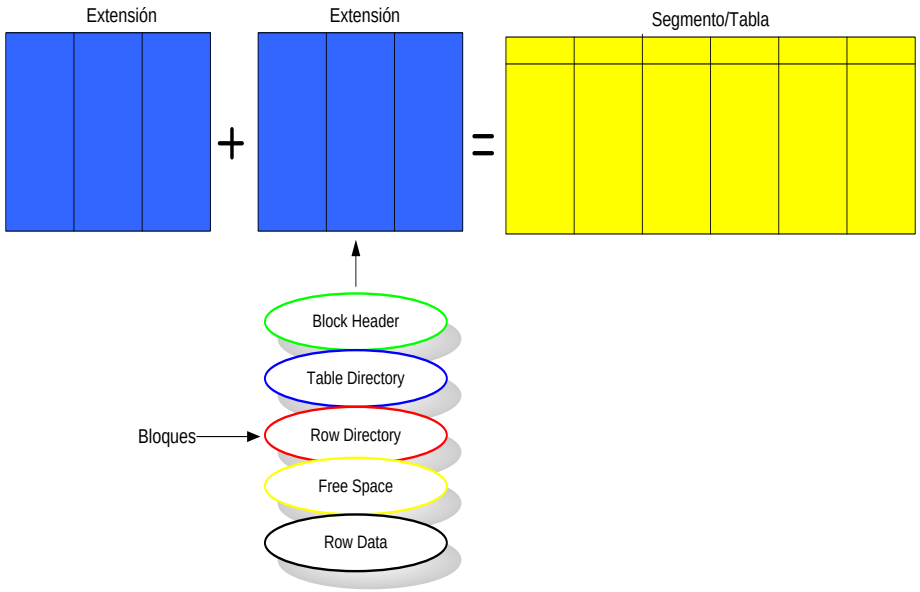


Figura 1.5. Extensión de bloques

En una base de datos pueden existir otros objetos que no contienen datos como las vistas, los sinónimos y las secuencias. Sin embargo, todo objeto independiente si contiene datos, o no, debe pertenecer a un esquema. Por eso, una colección de objetos de un usuario se denomina esquema.

Un objeto se puede crear en un esquema de tres formas:

1. Si un usuario da una orden de creación de un objeto, por default, el sistema lo crea en su propio esquema.

2. Copiando el objeto de otro usuario (al nombre de un objeto siempre se le antepone el nombre del esquema, por ejemplo Moisés_empleado) con una orden como:
   Create table empleado as select * from Moises_empleado;

3. otro usuario lo crea para uno, como en la orden:
   Create table paulina_proyecto (codigo number primary key…)

Tablespace planeación
Storage (inicial 1000 next 1000 minextents 1 maxextents 6…….)

Reglas para el almacenamiento de objetos en la base de datos

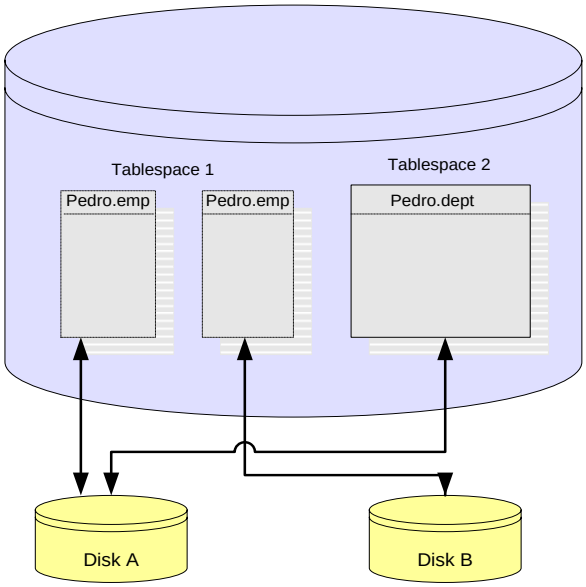


Figura 1.6. Almacenamiento de objetos

1. Un objeto puede almacenarse en uno o más archivos de datos (*datafiles*) pero de un solo espacio de tablas (*tablespace*).

2. Dos objetos diferentes de un esquema pueden estar en distintos *tablespace*.

3. Los objetos pueden almacenarse en múltiples discos. Por ejemplo, parte de Pedro_emp es almacenado en el archivo de datos 1 sobre el disco A y parte en el archivo de datos 2 sobre el disco B.

Archivos de control (Control Files): tienen la descripción física y dirección de los archivos de la base de datos y de los archivos restaurados, para el arranque correcto de la base de datos. En estos archivos se especifican cuáles datafiles conforman la base de datos para poder tener acceso a los datos o para poder recuperar la base de datos, ante una falla.

Los archivos de control se crean automáticamente cuando se da una orden CREATE DATABASE y no son editables, pues también se actualizan automáticamente.

Archivos restaurados (redo log files), tienen los cambios hechos a la base de datos para la recuperación ante fallas o para el manejo de las transacciones. Conservan los valores antes de una transacción, la orden ejecutada y opcionalmente, el valor después de la transacción. El principal propósito de estos archivos es servir de respaldo de los datos en la memoria RAM. Este conjunto de archivos debe estar conformado por dos grupos como mínimo y se recomienda que cada grupo esté almacenado en discos separados. El DBMS utiliza la técnica de ir sobrescribiendo sobre la información más antigua, cuando se agota el espacio en estos grupos de archivos.

Archivos fuera de línea (*archived files*) son archivos opcionales donde se guarda información antigua de los archivos restaurados, muy convenientes para los respaldos de la base de datos.

Los procesos

Los procesos son programas que se ejecutan para permitir el acceso a los datos. Los procesos se cargan en memoria y son transparentes para los usuarios. Los procesos se clasifican en tres grupos: procesos de base, de usuario y procesos servidores.

Los procesos de Base o de Soporte

Los procesos de base (*background*) son los que se encargan de traer datos desde y hacia la SGA; mejorando el desempeño al consolidar las tareas que son impartidas por todos los usuarios. Cada proceso de base tiene su propia área de memoria. Los procesos de base o soporte son los siguientes.

DBWR: (*Database writer*) se encarga de manejar los “buffers” de memoria cache para que los procesos del usuario siempre se encuentren algunos de ellos disponibles. Es un proceso obligatorio que además escribe los bloques de datos modificados por los usuarios, en los archivos de datos que componen la B.D. cuando el proceso LGWR le envía el mensaje de hacerlo.

LGWR: (*Log writer*) este proceso escribe datos desde la SGA a los archivos restaurados (*redo log files*) que sirven en caso de fallas en la instancia. Este proceso es obligatorio y es el único encargado de escribir y leer en estos archivos. El proceso de saturación de estos archivos es circular, por lo tanto antes de iniciar a sobrescribir en uno de ellos, se marca un punto de verificación y LGWR envía la orden de escritura en los *datafiles* al proceso DBWR.

LCKn, Lock: (*lock process*) El bloqueo es un proceso opcional. Efectúa los bloqueos entre instancias, en caso de ambientes con servidores paralelos (hasta con 10 servidores).

CKPT: (*Check point*) El punto de comprobación es un proceso opcional que ocurre cuando los usuarios conectados a la base de datos, hacen solicitudes de exámenes de datos.

SNPn: (*Snapshot process*) se encarga de actualizar los *snapshot* o réplicas de tablas que se usan principalmente en ambientes distribuidos.

SMON :(*System monitor*) recupera el sistema ante una falla de la instancia.

RECO. (*Recovery*) recupera ante las fallas, en una transacción en ambientes distribuidos.

ARCH: (*Archive*) copia los registros restaurados de la memoria RAM en archivos de datos que permite la recuperación cuando se presentan fallas de los medios magnéticos.

PMON: (*Process Monitor*) recupera ante una falla de un proceso de usuario; libera los recursos del proceso que fallo.

Procesos del Usuario

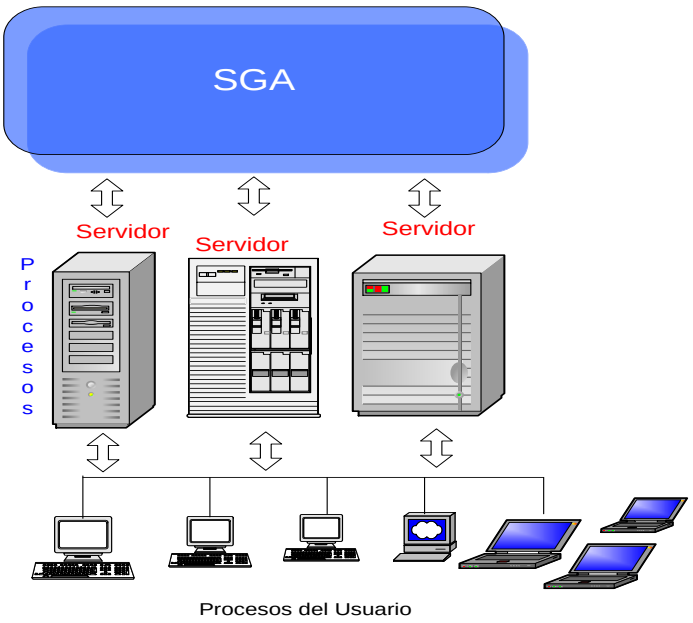


Figura 1.7. Procesos de usuario

Cuando un usuario se conecta a la base de datos, se crea un proceso de usuario que se encarga de ejecutar el código de aplicación del usuario y manejar el perfil

del usuario con sus variables de ambiente. Los procesos de usuario no se pueden comunicar directamente con la base de datos, únicamente lo hacen a través de procesos servidores.

Procesos Servidores

Ejecutan las órdenes SQL de los usuarios y llevan los datos al “database buffer cache”, para que los procesos del usuario puedan tener acceso a los datos. Se pueden tener distintas arquitecturas para trabajar en ORACLE, según los tipos de servidores: dedicados o multiusuario.

Instancia de ORACLE

Se denomina instancia al conjunto de estructuras de memoria y procesos de fondo que acceden a los archivos de base de datos. Es posible que una misma base de datos sea accedida por múltiples instancias; cada una de ellas reside en servidores diferentes (esta es la opción de servidores paralelos de ORACLE).

El sistema de base de datos ORACLE, cuando inicia, sigue los siguientes pasos que se detallan a continuación.

1. **Iniciar la instancia**. Para hacer este paso ORACLE lee el archivo de parámetros y configura la instancia, con base en ellos. En ese momento se crea la SGA y se activan los procesos de base; pero aun no se puede hacer nada. Es como encender un auto en neutral, listo para empezar a trabajar.

2. **Montar la base de datos**. Consiste en preparar el sistema para uso, trayendo a la RAM el diccionario de datos; es como poner el sistema en primera, listo para recibir algunas órdenes del DBA.

3. **Abrir la base de datos**. En este momento se abren los archivos y los usuarios ya pueden tener acceso a los datos.

De acuerdo con la anterior definición de instancia, ORACLE, a través de sus parámetros, puede determinar qué tan eficaz y espacioso es el motor. Los parámetros se definen en el archivo INIT.ORA, entre ellos se puede mencionar:

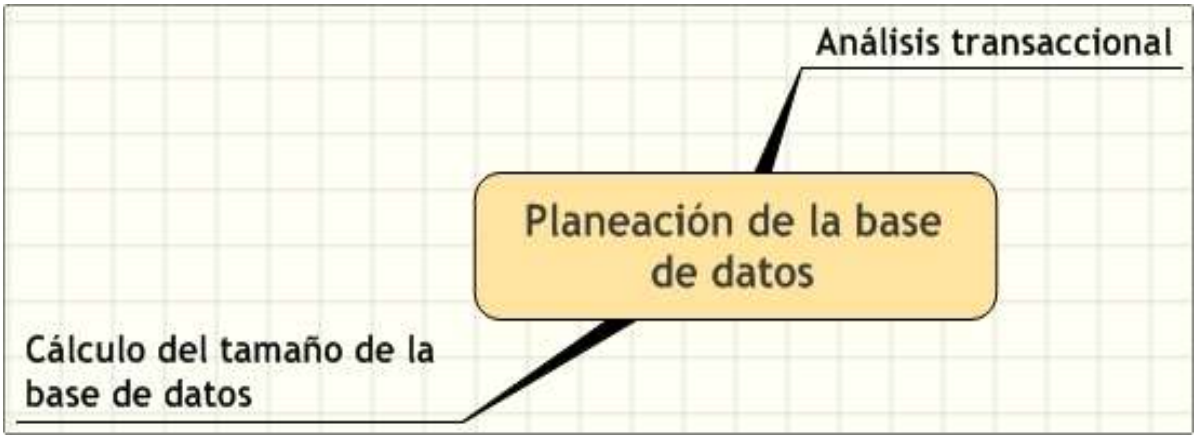
- db_block_buffers = numero de bloques de base de datos en la SGA. Existirá un buffer por cada bloque.
- db_block_size = tamaño del bloque de la base de datos.
  shared_spool_size = tamaño del area compartida “shared spool”, en bytes.

Además, allí se especifica el número de usuarios concurrentes, el número de transacciones concurrentes y los nombres de los archivos de control para la base de datos.

Estos parámetros se pueden ajustar, durante el proceso de afinamiento, porque ellos inciden en el desempeño del sistema. Algunos de los parámetros son específicos a una base de datos y por lo tanto deben cambiarse antes de crear una base de datos. Se incluyen en estos:

- database_name = nombre de la base de datos.
- db_block_size = tamaño del bloque.

RESUMEN


Análisis transaccional
Planeación de la base de datos
Cálculo del tamaño de la base de datos

BIBLIOGRAFÍA

SUGERIDA

Date, C. J. 2001. Sistemas de Bases de Datos. 7ª, México: Pearson.

Elmasri, Ramez. 2002. Fundamentos de sistemas de bases de datos. México: Pearson Educación, Addison-Wesley. Worsley C. y Joshua D.Drake. 2002. Practical PostgreSQL. Sebastopol, CA: O'Reilly. (Disponible en http://www.faqs.org/docs/ppbook/book1.htm).

Silberschatz, A., H. Korth y S. Sudarshan. 2006. Fundamentos de bases de datos. 5ª, Madrid, España: McGraw-Hill. Planning a database implementation, en http://publib.boulder.ibm.com/infocenter/rbhelp/v6r3/index.jsp?topic=/com.ibm.redbrick.doc6.3/wag/wag41.htm Visitada el 03/08/2009.

Estimating the Size of a Table, en http://msdn.microsoft.com/en-us/library/aa933068(SQL.80).aspx. Visitada el 03/08/2009

Unidad 2

Creación de la base de datos

OBJETIVO PARTICULAR

Al terminar el tema, el alumno será capaz de crear y manejar la base de datos así como su almacenamiento, manipulación y recuperación sin importar la computadora que se utilice

TEMARIO DETALLADO

(10-horas)

2. Creación de la base de datos

- 2.1. El lenguaje SQL
- 2.2. Creación de tablas
- 2.3. Modificación de tablas
- 2.4. Integridad
- 2.5. Modificación de datos
  - 2.5.1. Actualizar
  - 2.5.2. Insertar
  - 2.5.3. Eliminar

INTRODUCCIÓN

SQL es el lenguaje que sirve para trabajar con bases de datos, con independencia de la plataforma hardware y software donde se ejecuten.

Si nos atenemos al dicho “la información es poder”, podemos afirmar sin ninguna duda que la capacidad para obtener esa información otorga una cierta autoridad.

Saber cómo recuperar la información adecuada es algo fundamental, sin importar la función que desempeñemos en nuestra actividad diaria. Por ello el conocimiento del lenguaje SQL puede ser considerado como algo básico en el campo de la informática, una preparación indispensable y primordial tanto para un usuario de aplicaciones ofimáticas como para un programador, un diseñador de soluciones Web y, por supuesto, un administrador de base de datos.

Independientemente de cuál sea nuestro perfil, como usuario de informática, es muy posible que en un momento u otro tengamos que vérnoslas con este lenguaje. No es algo que deba intimidarnos ya que, como podrás experimentar por ti mismo, es más fácil de lo que pueda parecer en un principio.

2.1. El lenguaje SQL

¿Qué es SQL?

El acrónimo SQL procede de Structured Query Language, que podríamos traducir como lenguaje estructurado de consultas. Se trata, por tanto de un lenguaje de computadora, un grupo de palabras, a las que se llama normalmente instrucciones u órdenes, que se combinan conforme a unas reglas gramaticales para dar forma a frases conocidas como sentencias. No hay mucha diferencia, consecuentemente, con una lengua hablada, salvo por la gran simplicidad de SQL, puesto que el número de palabras y reglas es muy pequeño si lo comparamos con una lengua como podría ser el castellano. Otra diferencia, lógica y obvia, es que SQL es una lengua artificial, creada a medida para un determinado propósito: facilitar la comunicación entre las personas y los programas que gestionan las bases de datos.

La historia del lenguaje SQL va unida inevitablemente a la de los sistemas de bases de datos, concepto surgido a finales de los años 60 en la empresa IBM y que dio origen a un proyecto, conocido como System R, cuyo objetivo era crear un sistema de base de datos relacional que permitiera a un usuario desarrollar desde una Terminal el trabajo que, hasta el momento, requería la intervención de un buen número de personas especializadas en el archivo e indización de información. El fruto de ese proyecto fue SQL/DS, un producto de IBM que, con posterioridad, se convertiría en DB2. El lenguaje que entendía ese producto era SEQUEL, el precursor de SQL que, en la década de los 80, quedó bajo el control de ANSI, la organización americana de estandarización, apareciendo la primera versión estándar a mediados de los 80.

Actualmente SQL es un estándar ISO, algo que representa una cierta garantía en cuanto a la uniformidad y consistencia de la implantación del lenguaje en los productos de distintos fabricantes.

Desde 1989, año en que se publicó el estándar SQL1, el lenguaje ha ido evolucionando dando lugar a varias versiones o ediciones distintas.

- **SQL2 o SQL/92:** Fue la primera edición del estándar que recogía todas las características necesarias para la manipulación de datos, en comparación con SQL1 que era una edición incompleta.
- **SQL3 o SQL/99:** Es el estándar que hay en uso en la actualidad. Divide SQL en distintos niveles: núcleo del lenguaje, interfaz de llamadas, procedimientos almacenados, enlaces con lenguaje, etc. Cada parte es una especificación en sí misma y, como usuarios de productos basados en SQL, la única que nos interesaría sería el núcleo.
- **SQL/2003:** Se trata de una versión aun en desarrollo y que aporta a la edición anterior aspectos como una mayor colección de tipos de datos, integración con lenguaje XML, etc.

El nivel básico de SQL/92 es muy similar al núcleo de SQL/99, lo cual significa que el lenguaje tiene una sintaxis prácticamente idéntica en todos los productos que se ajustan a uno de estos dos estándares. Esto, no obstante, no significa que no existan ciertas diferencias entre las implementaciones de los distintos fabricantes. Las tres variantes mas importantes de SQL, por el número de instalaciones en las que se utilizan, son Oracle PL/SQL, IBM DB2 SQL y Microsoft Transact-SQL, existiendo muchas otras como las usadas en MySQL, PostgreSQL, Informix, etc.

Aplicaciones de SQL

El lenguaje SQL puede aplicarse a cualquier escenario en el que sea preciso trabajar con información que, de una forma u otra, sea representable en una

estructura tridimensional: columnas, filas y tablas. No es, por tanto, exclusivo para la operatividad sobre grandes bases de datos y sirve igualmente para tratar datos de una Mac que ejecuta FileMaker, recuperar información de una hoja de cálculo Microsoft Excel o Lotus, incluso, manipular los datos de un archivo de texto si estos se encuentra convenientemente delimitados.

Como conocimiento, el lenguaje SQL puede permitir a los usuarios de este tipo de aplicaciones: Access, FileMaker, Excel, Lotus, etc., llevar a cabo operaciones sobre los datos que no están predefinidas en algún tipo de formulario o informe que les haya preparado un técnico. Los programadores de aplicaciones necesitan conocer SQL para saber cómo recuperar la información con la que trabajarán sus programas, recuperándola de su origen y enviando a este las modificaciones que procedan. Aquellos que diseñan aplicaciones web, con herramientas como por ejemplo Dreamweaver, GoLive o Flash, necesitan SQL para recuperar la información que aparecerá en las páginas de sus sitios. Los administradores de bases de datos, lógicamente, son los más interesados en conocer SQL, puesto que en ellos recaen tareas vitales como la creación de las estructuras en las que se almacenará la información, la definición de procesos que se ejecutan en la base de datos o su mantenimiento.

Intérpretes de SQL

Dependiendo del tipo de programa que gestione los datos y nuestras posibilidades de acceso a estos, por cuestiones de seguridad, se utilizarán unas herramientas u otras para transmitir las sentencias SQL desde la computadora en que se trabaja hasta la base de datos.

Por regla general, todos los programas que entienden el lenguaje SQL cuentan con un intérprete, un programa que permite escribir las órdenes de manera interactiva, enviándolas a la base de datos y recibiendo de manera inmediata el resultado que generan.

El intérprete puede ser una aplicación basada en texto. Un ejemplo de ello sería la consola de la base de datos MySQL ejecutándose en GNU/Linux o Mac OS X. también puede contar con una interfaz de usuario gráfica más elaborada, tal como ocurre con productos como Microsoft Access o el cliente de Oracle.

En cualquier caso, indistintamente del tipo de interfaz, sistema operativo y base de datos que se emplee, la operatividad básica es siempre la misma: se introduce una sentencia SQL completa que se transmite a la base de datos que, tras su proceso, devuelve un resultado que aparecerá en la pantalla.

Al utilizar el término base de datos hay que tener en cuenta que se puede estar haciendo referencia al conjunto de información que forma la base de datos, las tablas, filas y columnas, pero también al programa que se encarga de gestionar esa información, lo que se conoce normalmente como **RDBMS (Relational Database Management System/Sistema de administración de bases de datos relacionales).** Por ello, y aunque, como se ha dicho antes, las bases de datos pueden ser también una hoja de cálculo, un archivo de texto y en general, cualquier tipo de archivo de datos [1]

Tipos de RDBMS

Los RDBMS, programas que se encargan de manipular los datos según nuestras indicaciones, se clasifican en dos grandes categorías: locales y servidores. Un RDBMS local se caracteriza por almacenar la información en el propio equipo del usuario que accede a ella. El caso más común es el administrativo que usa Microsoft Access en su equipo de escritorio. En este caso el RDBMS es Microsoft Access, un programa que, a demanda del usuario, recupera y manipula los datos que están almacenados en el disco de la propia computadora.

[1] En estas notas se usará el término 'base de datos' para hacer referencia a la información y el término RDBMS cuando se quiera señalar el programa que administra la base de datos.

En contraposición a estos, se encuentran los RDBMS que actúan como servidores. Se trata de programas que se ejecutan en una computadora central, denominado **servidor de datos**, al que se conectan los equipos de los usuarios, que se denominan **clientes**. Estos tienen en sus computadoras un software que les permite comunicarse con el servidor de datos, enviando sus sentencias SQL. De esta manera es posible contar con un único depósito de información común a un grupo de usuarios o, incluso, a una empresa entera, facilitando tareas como la seguridad en el acceso a la información, la realización de copias de seguridad y otras labores de mantenimiento.

Por regla general, no tendrá limitación alguna para operar sobre bases de datos que tengan alojadas en un RDBMS local, en su propio equipo, lo cual le permitirá ejecutar cualquier tipo de sentencia SQL. Al trabajar con un servidor de datos, sin embargo, la seguridad es un aspecto mucho más importante y se requieren unos permisos para poder efectuar ciertas acciones. Salvo que sea el administrador de bases de datos o bien un usuario con experiencia, normalmente no tendrá acceso directo al intérprete SQL de un RDBMS remoto. Ésto, sin embargo, no significa que no puedas probar tus conocimientos del lenguaje SQL en productos como Oracle, SQL Server, MySQL. Lo único que tienes que hacer es instalarlos en tu propio equipo o bien en uno que puedas hacer las funciones de servidor de datos de pruebas.

2.2. Creación de tablas

Las bases de datos son adecuadas para almacenar prácticamente cualquier tipo de datos e información por su capacidad para adaptarse a las necesidades de cada aplicación, ajustando los datos a la estructura que sea precisa. La estructura, nombre de cada tabla, columnas, tipos, etc., se estableció al ejecutar el guión que se encarga de crear la base de datos.

Dependiendo la función que tengamos asignada, puede ser que siempre operemos sobre bases de datos creadas previamente, limitándonos a extraer información y actualizarla, o, por el contrario, podemos vernos en la situación de tener que definir nosotros mismos la estructura de las bases de datos. Con este objetivo utilizaríamos DDL (*Data Definition Language*), la parte del lenguaje SQL dedicada a la definición de datos.

Nuestro objetivo es conocer los elementos necesarios de DDL para poder concretar la estructura de una base de datos, estableciendo el nombre de cada tabla, el de sus columnas, tipos de datos y restricciones asociadas, como puede ser la clave primaria o valor único.

Para llevar a cabo este trabajo conoceremos las tres sentencias esenciales del DDL: CREATE, ALTER y DROP. La primera se emplea para crear objetos en la base de datos, la segunda para modificarlas y la tercera para eliminarlas, siendo en este sentido equivalentes a las sentencias INSERT, UPDATE y DELETE de DML. La diferencia es que los objetos sobre los que se actúan no son filas de datos, sino tablas, vistas e índices y en general, cualquier tipo de metainformación que el RDBMS sea capaz de alojar en el interior de la base de datos.

Creación de una tabla

Asumiendo que tenemos creada en el RDBMS una nueva base de datos, estaremos en disposición de ir definiendo la estructura de cada una de las tablas que vayan a formar parte de ella. En este sentido y dependiendo de la complejidad de la base de datos, a menos que se use una herramienta de diseño específico, lo mejor es partir de un esbozo en papel apuntando la información que deberá alojar cada tabla, los nombres de las columnas, sus tipos, las relaciones que existieran entre las tablas, etc. Es el momento en que puede verse, antes de crear estructura alguna, si es necesario añadir o eliminar columnas, agrupar algunas columnas en una tabla independiente para evitar redundancia, etc.

Teniendo ya clara cuál será la estructura, iremos creando las distintas tablas con la sentencia CREATE TABLE, cuya sintaxis general es la siguiente:

```
CREATE [LOCAL | GLOBAL TEMPORARY] nombre_tabla
[CONSTANT | UPDATABLE]
(  nombrecolumna1    tipo1    [restriccion],
    nombrecolumna2    tipo2    [restriccion]    ….,
     [restricciones de tabla])
[ON COMMIT PRESERVE | DELETE]
```

Como puedes apreciar, muchas de las cláusulas de esta sentencia son opcionales. Algunos de los elementos, como la restricción aplicable a cada columna o a la tabla, pueden ser relativamente complejos.

El formato más sencillo de esta sentencia sería el se muestra a continuación:

```
CREATE TABLE nombre_tabla (
columna1 tipo1,
```

columna2 tipo2.....
)

Con una sentencia de este tipo estableceríamos exclusivamente el nombre de la tabla, el nombre de cada columna y su tipo.}
CREATE TABLE
create table prestamos (id_prestamo integer not null primary key,
nif_prestamos varchar (10) not null,
codigo varchar (5) not null,
prestamo date not null);
NOTICE: CREATE TABLE / PRIMARY KEY creara el indice impicito «préstamos_pkey» para la tabla «préstamos»

2.3. Modificación de tablas

Tras haber definido la estructura de una tabla, con la sentencia CREATE TABLE que ya conocemos, si necesitamos modificarla, añadiendo columnas, eliminándolas, modificando su valor, agregando o excluyendo restricciones, etc., tenemos dos posibilidades: eliminar la tabla, con una sentencia DROP TABLE, y volver a crearla partiendo desde cero, o bien emplear la sentencia ALTER para llevar a cabo los cambios sin perder toda la información contenida en la tabla.

Los elementos que pueden cambiarse en la estructura de una tabla varían de un RDBMS a otro, si bien en el estándar se definen las siguientes operaciones:

- Añadir nuevas columnas.
- Eliminar columnas existentes.
- Eliminar y establecer el valor por default asociado a una columna.
- Eliminar el dominio asociado a una columna.

- Añadir y eliminar restricciones al nivel de tabla.

Algunos RDBMS permiten operaciones adicionales, como la modificación del nombre de las columnas, cambiar su tipo, agregar y eliminar restricciones asociadas a las columnas, etc. Ninguna de éstas, sin embargo, forma parte de la sintaxis estándar, por lo que se tendrá que recurrir a la documentación específica de cada producto al no poder usar la misma sentencia en todos los casos.

La sintaxis general de la sentencia ALTER es esta:

```
ALTER TABLE nombre_tabla
ADD | DROP  COLUMN | CONSTRAINT  nombre  [atributos]
ALTER  COLUMN  nombre  SET  DEFAULT | DROP  DEFAULT
```

Veamos en la práctica cómo llevar a cabo algunas de las operaciones de modificación de estructura más usuales.

Añadir y eliminar columnas

Añadir una nueva columna a una tabla existente es una operación contemplada por todos los RDBMS, con pocas diferencias en la sintaxis que debe emplearse. La sentencia a usar sería similar a la siguiente:

```
ALTER TABLE nombre_tabla
ADD  [COLUMN]  nombre_columna  tipo  [atributos]
```

Además del nombre de la columna y su tipo, datos imprescindibles, también pueden incluirse otros atributos como el valor por default y restricciones de columna. La palabra COLUMN es opcional, podemos incluirla o no.

A la hora de eliminar una columna de la tabla:

ALTER TABLE nombre_tabla
DROP COLUMN nombre_columna.

ALTER TABLE libros
DROP COLUMN disponible;

Esta sentencia eliminará la columna que habíamos añadido. La eliminación de una columna conlleva, como es fácil de suponer, a la pérdida de los datos que hubiese almacenados en dicha columna en todas las filas de la tabla.

Modificar una columna

Teóricamente es posible modificar ciertos atributos de una columna, sin necesidad de eliminarla y volver a añadirla, empleando la sentencia ALTER TABLE con la cláusula ALTER COLUMN.

Esta posibilidad evita la pérdida de los datos contenidos en dicha columna, pero existen diferencias ostensibles entre la sintaxis y operaciones que permiten cada uno de los RDBMS.

ALTER TABLE nombre_tabla
ALTER COLUMN nombre_columna SET DEFAULT ‘S’

ALTER TABLE libros
ALTER COLUMN disponible SET DEFAULT ‘S’
Alter table préstamos drop column id;
ALTER TABLE

arojas=> alter table prestamos add column id_prestamos integer not null primary key;

NOTICE: ALTER TABLE / ADD PRIMARY KEY crearÃ¡ el Ã-ndice implÃ-cito Â«préstamos_pkeyÂ» para la tabla Â«préstamosÂ»
ALTER TABLE

Añadir y eliminar restricciones

Las restricciones con que cuenta una tabla, siempre que conozcamos sus nombres, pueden ser eliminadas, existiendo también la posibilidad de añadir otras nuevas. Estas son las operaciones fundamentales, si bien algunos RDBMS habilitan también su modificación. La sintaxis general, para eliminar una restricción sería la siguiente:

ALTER TABLE nombre_tabla
DROP CONSTRAINT nombrerestriccion
ALTER TABLE tabla
ADD PRIMARY KEY (columnas)

ALTER TABLE tabla
DROP PRIMARY KEY

La adición de una nueva restricción requiere, después de la cláusula ADD CONSTRAINT, el nombre que se le dará a los atributos que correspondan. En general solo podremos añadir restricciones al nivel de tabla, como FOREIGN KEY Y PRIMARY KEY y no de columna, como NOT NULL o CHECK

2.4. Integridad

Catalogación de los datos

Una de las grandes ventajas del modelo relacional, frente a otros de los que han ido utilizándose o surgiendo con el tiempo, es el hecho de que la estructura de los datos no está preestablecida en un programa ni es inamovible por la propia arquitectura del modelo.

En una base de datos relacional, lo que se conoce como el catálogo de datos meta-datos, información que describe la estructura de los datos, está almacenada también en la propia base de datos. Es más, ese catálogo debe aparecer como una parte más y por tanto, ser accesible mediante la misma mecánica.

El catálogo de una base de datos es, por ende, una descripción de la estructura de dicha base. Dependiendo del RDBMS que se emplee, el nombre del catálogo en sí puede cambiar será el método de acceso a su contenido, mediante consultas SQL, ni tampoco la información almacenada en el: nombre de tablas, nombres y tipos de columnas y otros atributos.

A las tablas que forman el catálogo se le conoce habitualmente como tablas de sistema, en contraposición a las tablas de datos propiamente dichas, que son conocidas como tablas de usuario.

Integridad de los datos

Otra de las reglas establecidas por Codd en su definición de sistema relacional de bases de datos indica que la integridad de la información debe ser mantenida en la propia base de datos, no en las aplicaciones que la utilicen. Los atributos necesarios

para fijar esa integridad se almacenarán en el catálogo o diccionario de la base de datos, conjuntamente con la información de las tablas, índices, vistas y cualquier otro objeto que pudiera existir.

Básicamente encontraremos dos categorías que afectan a la integridad de la información: la integridad de dominio y la integridad referencial. La primera especifica el tipo de Información que se puede alojar en cada columna, su longitud, si puede quedar o no sin completar, etc. Con la segunda se asegura que las referencias existentes entre tablas sean válidas y que, por ejemplo, una clave externa de la tabla libros usada antes como ejemplo no haga referencia a un código de editorial inexistente.

Los mecanismos expuestos por la base de datos para garantizar la integridad de la información deben ser, además, de aplicación obligatoria.

En la integridad de dominio son fundamentales aspectos como el tipo de datos asociado a una columna y los atributos derivados de dicho tipo: longitud máxima de una secuencia de caracteres, rango de valores permitidos en un número, lista de términos validos para una enumeración, posibilidad de que el dato quede como nulo,

Nombre	Información que puede contener
Boolean	Sí o no, verdadero o falso. Únicamente uno de dos valores posibles.
Char	Una secuencia de caracteres de longitud fija
Varchar	Una secuencia de caracteres de longitud variable.
Int	Un número entero, sin parte decimal.
Money	Un número relativo a importes económicos
Float	Un número de punto flotante, con parte decimal.
Date	Una fecha
Binary	Cualquier secuencia de bytes

etc.

Tabla 2. Tipos de datos más usuales en una base de datos

Integridad referencial

Se denomina integridad referencial a los mecanismos de la base de datos enfocados a garantizar que un dato referenciado desde una cierta tabla esté disponible en otra tabla de esa misma base de datos. Por una parte están los atributos que establecen el enlace entre ambas tablas, vinculando una columna de la primera con la clave primaria de la segunda y por otra las comprobaciones internas que debe efectuar el propio RDBMS para asegurar que esa integridad se mantiene.

En la figura siguiente, por ejemplo, la columna Editorial de la tabla Libros contiene un código que debe coincidir con la clave primaria, Id, de alguna de las filas de la tabla Editoriales.

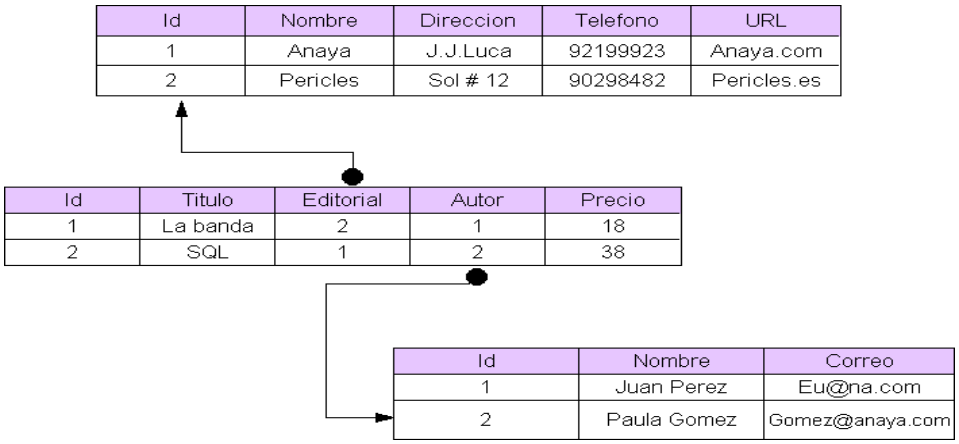


Figura 2.1 Ejemplo de integridad referencial

Relaciones entre tres tablas de una base de datos

En este caso concreto Libros.Editorial es la FK o clave externa, mientras que Editoriales: Id es la PK o clave primaria. Este tipo de relaciones se declaran explícitamente en el momento en que se define la estructura de las tablas, usando para ello el lenguaje SQL.

La existencia de estas relaciones entre tablas da sentido al modelo relacional, siendo también las responsables de que se evite la redundancia de datos. Si en la tabla libros almacenamos, en lugar de un código numérico, el nombre de la editorial, dirección, teléfono, etc., por cada libro, estaríamos inevitablemente repitiendo datos, ya que habrá múltiples libros que pertenezcan a una misma editorial, por tanto, que contengan los mismos datos. Separando la información de las editoriales en una tabla propia y estableciendo la relación mencionada, utilizando solamente un código numérico, se reduce considerablemente el tamaño de las bases de datos al disminuir los datos que son redundantes.

Tratamiento de valores nulos

En una base de datos relacional el contenido de una determinada columna en una fila de una tabla puede presentar tres estados diferentes: contener un valor, estar vacía o ser nula. En realidad los dos primeros son idénticos, la particularidad es que hay ciertos valores, como el 0 en el caso de los números o la cadena “ ” las secuencias de caracteres, que pueden considerarse como vacías.

Los valores nulos, representados a menudo como Null, son un caso especial. La presencia de Null indica que el contenido de la columna es **desconocido**. Por ello no debe confundirse Null con 0 ó “ “. No es lo mismo una columna que contiene una secuencia vacía de caracteres, por ejemplo al haber eliminado un contenido previo, que aquella que nunca tuvo contenido, por tanto, es Null.

El tratamiento de valores nulos es importante ya que condiciona la selección de datos, algo que es necesario tener en cuenta para no encontrarnos con sorpresa o resultados inesperados. Si, por ejemplo, buscamos en la tabla Libros puesta antes como ejemplo todas aquellas filas en las que la editorial sea 0, para corregir la asociación de cada libro con su editorial, no obtendríamos en el resultado aquellas filas cuya columna Editorial fuese Null, quedando así libros sin corregir esa asociación.

2.5. Modificación de datos

Lenguajes de comunicación con la base de datos

Una de las reglas más importantes de Codd en su definición de sistema relacional, y responsable de la existencia de SQL, es la que indica que, con independencia de que el RDBMS ofrezca los lenguajes específicos que desee para operar sobre las bases de datos, debe existir siempre un lenguaje con una sintaxis estándar, cuyas sentencias puedan ser expresadas textualmente y que permita efectuar cualquiera de los siguientes tipos de operaciones:

- Manipulación de datos.
- Definición de datos.
- Definición de vistas.
- Administración de seguridad.
- Control de transacciones.

Aunque Codd no indicaba explícitamente que ese lenguaje SQL, ya que por entonces éste no existía como tal, esta regla fue la causante de que SQL se convirtiera rápidamente en el lenguaje estándar para trabajar con bases de datos.

Para ajustarse a las distintas necesidades que representan las operaciones antes enumeradas, el lenguaje SQL se divide en tres partes bien diferenciadas:

- DDL: *Data Definition Language* - Lenguaje de definición de datos. Es la parte de SQL que se emplea para generar las estructuras de los datos, creando tablas, índices, atributos de integridad referencial, en general, cualquier

información que vaya a formar parte del catálogo o diccionario de la base de datos.

- DML: *Data Manipulation Language* - Lenguaje de manipulación de datos. Se trata de la parte más conocida de SQL, está formada por todas las sentencias que permiten seleccionar conjuntos de datos, eliminar información, actualizarla, filtrarla, agruparla, etc.
- CCL: *Data Control Language* – Lenguaje de control de datos. Está formado por las sentencias de SQL destinadas a controlar el acceso a los datos, definiendo privilegios de los usuarios, administrar las transacciones.

A estas tres partes, suele añadirse **una cuarta que es la que contiene el lenguaje utilizado para crear los procedimientos almacenados y funciones alojadas en la base de datos**.

Es en esta parte donde más diferencias hay entre distintos productos RDBMS, existiendo realmente lenguajes diferentes como PL/SQL (Oracle), T-SQL (SQL Server).

Tratamiento de conjunto de datos

El lenguaje utilizado para operar sobre las bases de datos es declarativo, no procedimental y debe estar orientado a operar sobre conjuntos de datos, en lugar de hacerlo de manera repetitiva sobre datos individuales.

Que el lenguaje sea declarativo significa que lo usaremos para indicar qué queremos hacer y no cómo lo queremos hacer. Al utilizar SQL, por ejemplo, compondremos sentencias comunicando al RDBMS que queremos modificar un dato en todas las filas que se ajusten a un cierto criterio, pero en ningún caso especificaremos cómo llevar a cabo ese proceso.

El modelo relacional de bases de datos está basado en la teoría de conjuntos y las operaciones algebraicas con éstos.

El lenguaje de comunicación con la base de datos debe contemplar estas operaciones, facilitando la selección de conjuntos de datos, su unión con otros conjuntos, la intersección, etc.

Normalizacion de bases de datos

Directamente relacionadas con el modelo relacional de bases de datos, aunque realmente no formen parte de él, existe una serie de reglas formales para la normalización de las estructuras de las bases de datos.

Hay en total seis formas normales o reglas de normalización, si bien las más conocidas y destacables son las tres primeras.

La normalización de una base de datos persigue varios objetivos, principalmente reducir la redundancia de datos y simplificar las dependencias entre columnas, aplicándose de manera acumulativa. Eso quiere decir que la segunda forma normal incluye a la primera, la tercera a la segunda y así sucesivamente. Una base de datos que esté en segunda forma normal, por tanto, cumplirá las dos primeras reglas de normalización.

Primera forma normal

La primera regla de normalizacion se expresa generalmente en forma de dos indicaciones separadas.

- Todos los atributos, valores almacenados en las columnas, deben ser indivisibles.
- No deben existir grupos de valores repetidos.

El valor de una columna debe ser una entidad atómica, indivisible, excluyendo así las dificultades que podría conllevar el tratamiento de un dato formado de varias partes.

Supón que tienes en una tabla una columna Dirección para almacenar la dirección completa, dato que se compondría del nombre de la calle, el número exterior, el número interior (puerta), el código postal, el estado y la capital.

Es el caso de la tabla que aparece en la parte superior de la siguiente figura.

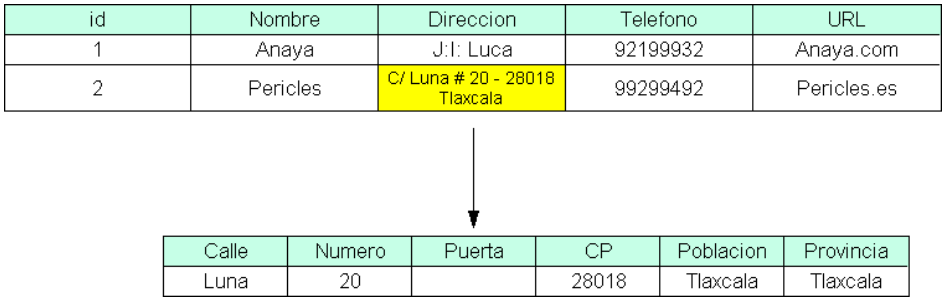


Figura 2.2. Tabla con un atributo divisible en varias partes

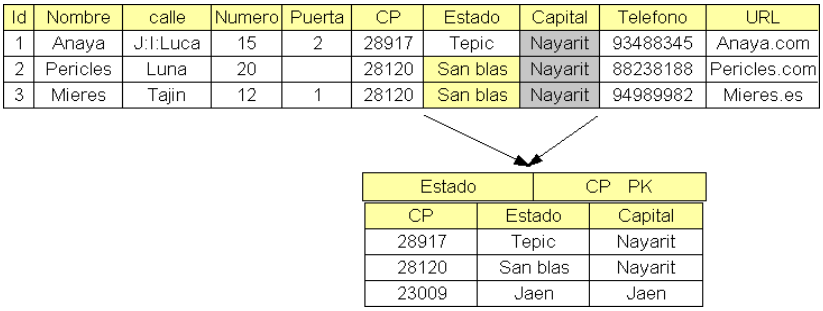
Una tabla con esta estructura plantea problemas a la hora de recuperar información. Imagina que necesitas conocer todas las entradas correspondientes a una determinada población, o que quieres buscar por el código postal. Al ser la dirección completa una secuencia de caracteres de estructura libre, no resultaría nada fácil.
La solución está en dividir el atributo Dirección en los atributos indivisibles que aparecen en la parte inferior de la figura.

Existirán más columnas, pero cada una de ellas contendrá un valor simple e indivisible que facilitará la realización de las operaciones antes mencionadas.

En cuanto a la segunda indicación, básicamente la resolvimos en el ejemplo de un punto previo, al evitar la repetición de los datos de la editorial en cada una de las

filas de la tabla Libros. Siempre que al muestrear la información de una tabla aparezcan datos repetidos, como ocurre en la tabla de la parte superior de la siguiente figura, existe la posibilidad de crear una tabla independiente con ellos, la que se encuentra en la parte inferior. Cuantas más entradas existan en la primera tabla, potencialmente con datos duplicados, tanto más efectiva será esta división.

Si el diseño de nuestra base de datos cumple estas premisas, está preparada para pasar de la primera a la segunda forma normal.



Id	Nombre	calle	Numero	Puerta	CP	Estado	Capital	Telefono	URL
1	Anaya	J:I:Luca	15	2	28917	Tepic	Nayarit	93488345	Anaya.com
2	Pericles	Luna	20		28120	San blas	Nayarit	88238188	Pericles.com
3	Mieres	Tajin	12	1	28120	San blas	Nayarit	94989982	Mieres.es

Estado	CP PK

CP	Estado	Capital
28917	Tepic	Nayarit
28120	San blas	Nayarit
23009	Jaen	Jaen

Figura 2.3 Aislamos los datos repetitivos de una tabla en otra tabla independiente

Segunda y tercera forma normal

Además de cumplir con las dos reglas del punto previo, la segunda forma normal añade la necesidad de que no existan dependencias funcionales parciales. Esto significa que todos los valores de las columnas de una fila deben depender de la clave primaria de dicha fila, entendiendo por clave primaria los valores de todas las columnas que la formen en caso de ser más de una.

Las tablas que están ajustadas a la primera forma normal, y además, disponen de una clave primaria formada por una única columna, con un valor indivisible, cumplen ya con la segunda forma normal. Ésta afecta exclusivamente a las tablas en las que la clave primaria está formada por los valores de dos o más columnas, debiendo

asegurarse, en este caso, de que todas las demás columnas son accesibles a través de la clave completa y nunca mediante una parte de esa clave.

En cuanto a la tercera forma normal, ésta indica que no deben existir dependencias transitivas entre las columnas de una tabla, lo cual significa que las columnas que no forman parte de la clave primaria deben depender sólo de la clave, nunca de otra columna no clave. En la tabla que hay en la parte superior del punto anterior, por ejemplo, las columnas Estado y Capital pueden ser determinadas mediante el valor de la columna CP, sin necesidad de la clave primaria que es el Id. Esto denota que no cumple con la tercera forma normal. En la tabla inferior, sin embargo, CP ha pasado a ser la clave primaria de una tabla independiente, mientras que Estado y Capital son columnas informativas dependientes de esa clave primaria.[2]

2.5.1. Actualizar

Introducción

Las operaciones que sobre una base de datos nos permiten llevar a cabo las instrucciones SELECT e INSERT, únicas que conocemos hasta el momento, no son en ningún caso destructivas, es decir, difícilmente causaremos con ellas la pérdida de información ya alojada en la base de datos.

Aunque la inserción de datos podría considerarse también una actualización de la información contenida en una base de datos, vamos a tratar las dos operaciones que más se identifican con el término actualización: la modificación de los datos que existe en las filas y su eliminación. Con este fin utilizaremos dos nuevas sentencias del lenguaje SQL: UPDATE y DELETE.

[2] Si estás interesado en profundizar en las reglas de normalización, el proceso de desnormalización y, en general, el diseño de bases de datos relacionales, se recomienda la lectura de textos específicos sobre este tema dada la profundidad y complejidad.

- **Modificación de datos**

En las tablas de una base de datos se almacenan dos categorías de información: aquella que puede ser considerada invariable y la susceptible de sufrir cambios a lo largo del tiempo. En la primera categoría estarían datos como el nombre y apellidos de una persona, el título de un libro o bien el nombre de su autor, mientras que en la segunda podrían tener cabida la dirección donde vive el socio, la disponibilidad de un libro o la fecha en que fue prestado por última vez.

Incluso los datos que son considerados invariables, como el nombre de una persona o el título de un libro, pueden necesitar una modificación en caso de que se hubiesen introducido inicialmente con algún tipo de error. En todos estos casos deberemos recurrir a la sentencia UPDATE, parte del lenguaje de manipulación de datos de SQL. Su sintaxis general es la siguiente:

```
UPDATE tabla
SET col1=valor1, col2=val2…
WHERE condicion.
```

Observa que la cláusula WHERE no se ha introducido entre corchetes, aunque en realidad se trata de una cláusula opcional de la sentencia UPDATE según el estándar. El riesgo de no incluirla, sin embargo, es muy grande, ya que la modificación que indicásemos se aplicaría a todas las filas de la tabla.

La finalidad de la cláusula WHERE en este caso, por tanto, es seleccionar las filas cuyas columnas se asignarán los valores indicados por SET.

Cambiar una columna de una fila

Uno de los casos de uso más habituales de la sentencia UPDATE se da cuando necesitamos cambiar el dato contenido en una columna de una fila determinada, en esta situación suele utilizarse una sentencia del tipo:

update socios set fecha_alta_socios = '2002-04-18' where id_nif_socios = '62877137F';
UPDATE 1

El RDBMS se limita a indicarnos el número de filas que se han visto afectadas, una información valiosa porque nos permite saber que la actualización ha afectado solamente a una fila, como debía ser, y no a varias.

Ejecutemos una consulta para comprobar cuál es el estado, fecha_alta_socios, en la tabla socios:

SELECT nif, nombre, apellido_paterno, fecha_alta_socios
FROM socios;

La consulta anterior, obteniendo nif, apellido_paterno y fecha_alta_socios de todos los socios, será la confirmación definitiva de que todo ha ido bien.

- **Cambiar varias columnas de una fila**

El procedimiento para modificar el contenido de varias columnas pertenecientes a una misma fila, por ejemplo la dirección y el código postal de un socio, es similar al que acaba de describirse en el punto previo. La diferencia estriba en que tras la cláusula SET, separadas por comas, irán facilitándose las parejas de nombres de columna y nuevo valor por asignar:

SET col1 = valor1, col2 = valor2…

Obviamente, cada uno de los valores debe ser del tipo adecuado según la columna a la que vaya a asignarse. La respuesta del RDBMS, nuevamente, se limitará a

indicarnos el número de filas que se han visto afectadas. Normalmente recurriremos de nuevo a utilizar la clave primaria de la tabla en la cláusula WHERE, asegurándonos así de no modificar nada más que la fila que corresponda.

La sentencia mostrada a continuación modificará las columnas dirección y cp de una fila concreta de la tabla socios, fila que identificamos a partir de la columna nif que es la clave primaria. En la figura siguiente puede verse una consulta previa a la modificación y otra posterior, apreciándose así el cambio en las columnas mencionadas.

update socios set direccion = 'Las Flores # 12', cp = '23021' where nif = '63273827G';
UPDATE 1

arojas=>select * from socios where apellido_paterno like 'Charte%';

nif	nombre	apellido_paterno	apellido_materno	direccion	cp	alta_socios
62877137F	Alejandro	Charte	Luque	Las Flores # 12	23021	2002-04-10
74381725T	Francisco	Charte	Ojeda	Las Flores # 12	23021	2005-06-09
63273827G	David	Charte	Luque			2005-06-29

(3 filas)

arojas=>select * from socios where apellido_paterno like 'Charte%';

nif	nombre	apellido_paterno	apellido_materno	direccion	cp	alta_socios
62877137F	Alejandro	Charte	Luque	Las Flores # 12	23021	2002-04-10
74381725T	Francisco	Charte	Ojeda	Las Flores # 12	23021	2005-06-09
63273827G	David	Charte	Luque	Las Flores # 12	23021	2005-06-29

(3 filas)

Modificamos dos columnas de una misma fila.
Nota: Según el estándar SQL, la sentencia UPDATE puede utilizarse tanto para actualizar una tabla como los datos generados por una vista.

- **Modificación de datos en varias filas**

Cuando necesitamos asignar el mismo valor a las mismas columnas de varias filas, no es preciso que usemos una sentencia UPDATE individual para cada fila utilizando la clave principal como referencia de selección, en su lugar, sustituiremos el filtro de búsqueda de la cláusula WHERE por uno adecuado que nos permita actuar sobre las filas que nos interesen. Este es un aspecto en el que debe ponerse la mayor atención, ya que un error en dicho filtro podría causar la pérdida de información en las filas que no deberían haberse visto afectadas. Lo mejor que podemos hacer, a fin de asegurarnos anticipadamente de que no vamos a manipular las filas inadecuadas, es ejecutar una consulta con el criterio de selección que pretendemos utilizar en la actualización. Las filas obtenidas como resultado serán las mismas que se actualicen con UPDATE.

Veamos un ejemplo partiendo de la consulta siguiente, en la que obtenemos el nif, apellido paterno y dirección de todos los socios y en la que se aprecia fácilmente que muchos de ellos no tienen una dirección. Quedando las columnas dirección y cp de las nuevas filas con valor NULL.

arojas=> select apellido_paterno, direccion from socios where direccion is not null;

apellido_paterno	direccion
Arias	Betania # 7
Moreno	Juan Rincon # 2
Charte	Las Flores # 12
Charte	Las Flores # 12
Charte	Las Flores # 12
Lopez	
Lopez	
Garcia	
Garcia	

Perez
Perez
(11 filas)

Lo que pretendemos hacer es almacenar el valor desconocido en la columna dirección de todas aquellas filas en las que no tienen valor. Necesitamos, por lo tanto, definir una condición en la cláusula WHERE que nos permita elegir esas filas y no otras.

Puesto que la consulta nos devuelve precisamente las filas que queremos, podemos sustituir la sentencia SELECT por una sentencia UPDATE en la que se mantenga la cláusula WHERE:

arojas=> update socios set direccion = 'Desconocida' where nif = '25856122E' or nif = '25856323T' or nif = '26323122E' or nif = '26323323T' or nif = '27548122E' or nif = '27548832H';
UPDATE 6

El resultado de la ejecución de esta sentencia es la notificación de que se han actualizado 6 filas, aquellas que no tenían una dirección, algo que podemos comprobar volviendo a usar la misma consulta original.

arojas=> select apellido_paterno,direccion from socios;

apellido_paterno	direccion
Arias	Betania # 7
Moreno	Juan Rincon # 2
Charte	Las Flores # 12
Charte	Las Flores # 12
Charte	Las Flores # 12

Lopez	Desconocida
Lopez	Desconocida
Garcia	Desconocida
Garcia	Desconocida
Perez	Desconocida
Perez	Desconocida

(11 filas)

Lógicamente, esta versión de la sentencia UPDATE, en la que se seleccionan varias filas a modificar, puede combinarse con la asignación de valores a más de una columna.

2.5.2. Insertar

Introducción

Hasta ahora todas las sentencias SQL han tenido como finalidad la extracción de información existente en una base de datos, llamada biblioteca, creada automáticamente.

Aunque la mayor parte de las operaciones que se lleva a cabo sobre una gran parte de las bases de datos empresariales son de consulta, a fin de elaborar informes, gráficos y cualquier otro tipo de resultado, también es habitual que desde aplicaciones de proceso de transacciones en línea se efectúen operaciones de inserción, modificación y borrado, enfocadas todas ellas a mantener actualizada la información que reside en la base de datos.

Conoceremos la sintaxis de la sentencia INSERT, cuyo objetivo es agregar filas a las tablas de una base de datos.

- **La sentencia INSERT**

Para agregar filas a una tabla, o en las columnas obtenidas mediante una vista si ésta es actualizable, nos serviremos de la sentencia INSERT que, al igual que SELECT, forma parte del subconjunto de SQL conocido como DML o de manipulación de datos.

La sintaxis general de esta sentencia es la siguiente:

INSERT INTO tabla [col1, col2 …)]
VALUES (val1, val2…)

Tras INSERT dispondremos la palabra INTO y, a continuación, el nombre de la tabla a la que van a añadirse datos. Esta irá seguida, opcionalmente entre paréntesis de los nombres de las columnas donde quieren insertarse los valores enumerados, tras la palabra clave VALUES. Habrá tantos valores como columnas se indiquen, o existan en la tabla, agregándose una nueva fila conteniendo esos valores. Si alguno de ellos incumple una restricción, por ejemplo duplicando un valor que no puede estar repetido o bien dejando como nula una columna que no puede ser NULL, la operación completa fallará, obtendremos un mensaje de error y no producirá ningún cambio en la tabla.

- **Inserción de valores por posición**

Si omitimos la indicación de los valores de columnas donde van a insertarse los valores, la sentencia INSERT tomará el formato mostrado a continuación:

INSERT INTO tabla
VALUES (val1, val2 …)

Los valores facilitados tras VALUES han de aparecer necesariamente en el mismo orden en que se definieron las columnas de la tabla, existiendo tantos valores como columnas existen en la tabla.

Esos valores, además deben coincidir en tipo y no incumplir ninguna de las restricciones que pudieran haberse establecido.

Suponiendo que quisiéramos agregar un nuevo libro a la tabla libros, deberíamos facilitar un valor para la columna código, otro para signatura, título, autor y disponible, en ese mismo orden. Por ejemplo:

INSERT INTO libros
VALUES (12,’G SHA inc’, ‘La incognita Newton’, ‘Shaw, Catherine’, ‘S’);

De entregar menos valores de los necesarios, el RDBMS nos informará del error con un mensaje similar al obtenido en la siguiente sentencia:

arojas=> insert into libros values (13,'La incognita Newton', 'Shaw, Catherine');

ERROR: el valor es demasiado largo para el tipo character varying(10)

Algo similar ocurrirá al incumplir una restricción, por ejemplo si intentamos introducir en la columna código un valor que ya exista. Esta columna es la llave principal de la tabla y como tal, no admite valores duplicados. El RDBMS nos lo hace saber con un mensaje de error:

arojas=> insert into libros values (12,'G SHA inc', 'La incognita Newton', 'Shaw, Catherine','S');

ERROR: llave duplicada viola restricciÃ³n unique Â«libros_pkeyÂ»

En caso de que necesitemos agregar varias filas a una misma tabla, como por ejemplo varios libros nuevos, escribiríamos una serie de sentencias INSERT, si bien algunos RDBMS permiten agregar varias filas en único paso.

- **Inserción de valores por nombre**

Si no conocemos la posición de las columnas en la tabla, para facilitar los valores en el orden correcto, o bien si vamos a aportar menos valores que columnas existentes en la tabla, será necesario que tras el nombre de ésta, y entre paréntesis, indiquemos los nombres de las columnas donde van a introducirse los valores.

Utilizando esta técnica, la sentencia siguiente añade una nueva fila a la tabla socios aportando valores solamente para parte de las columnas que la componen.

arojas=> insert into socios (nif,nombre,apellido_paterno,apellido_materno,direccion,cp,alta_socios) values ('63273827H','Manuel','Cid','Luque','Juan Carlos I # 23','23008',current_date);

Observa cómo se utiliza la funcion CURRENT_DATE para introducir en la columna alta_socios la fecha actual, la que indique el propio RDBMS.

- **Obtener la estructura de una tabla**

Cuando va a trabajarse sobre una base de datos cuya estructura no es ajena, por no haberla definido desde un principio, es habitual comenzar obteniendo dicha estructura, lo que se conoce comúnmente como información de esquema. Cada RDBMS implementa un mecanismo distinto para esta tarea, pudiendo ir desde una consulta directa sobre la tabla de sistema que almacena esa información hasta el uso de una sentencia específica para tal fin.

Todos los RDBMS almacenan la información de estructura en una serie de tablas de sistema que, por medio de la misma sentencia SELECT que ya conocemos,

podemos consultar obteniendo el nombre, tipo, longitud y otros atributos de cada columna, así como el nombre de cada tabla, vista, procedimiento almacenado y cualquier otro objeto que pueda existir en la base de datos. También es usual que los RDBMS cuenten con vistas y procedimientos almacenados que facilitan el acceso a esa información o, incluso, con una instrucción específica.

arojas=> \d libros;
Tabla Â«public.librosÂ»

Columna	Tipo	Modificadores
codigo	integer	not null
signatura	character varying(10)	not null
titulo	character varying(50)	not null
autor	character varying(40)	not null
ndices:ble	character(1)	

Â«libros_pkeyÂ» PRIMARY KEY, btree (codigo)
Â«

2.5.3. Eliminar

Aunque no es una tarea que se repita con la frecuencia del borrado o de la modificación de las filas de datos, la modificación de la estructura y eliminación completa de las tablas de una base de datos, también entran dentro de las funciones de un administrador.

La sentencia que usaremos para eliminar cualquier objeto de estructura de la base de datos es DROP, cuya sintaxis es muy sencilla:

DROP tipo nombre_objeto

Tras DROP indicaremos el tipo de objeto por eliminar, siendo este TABLE para las tablas y a continuación el nombre de dicho objeto. Para eliminar la tabla libros creada usaremos esta sentencia:

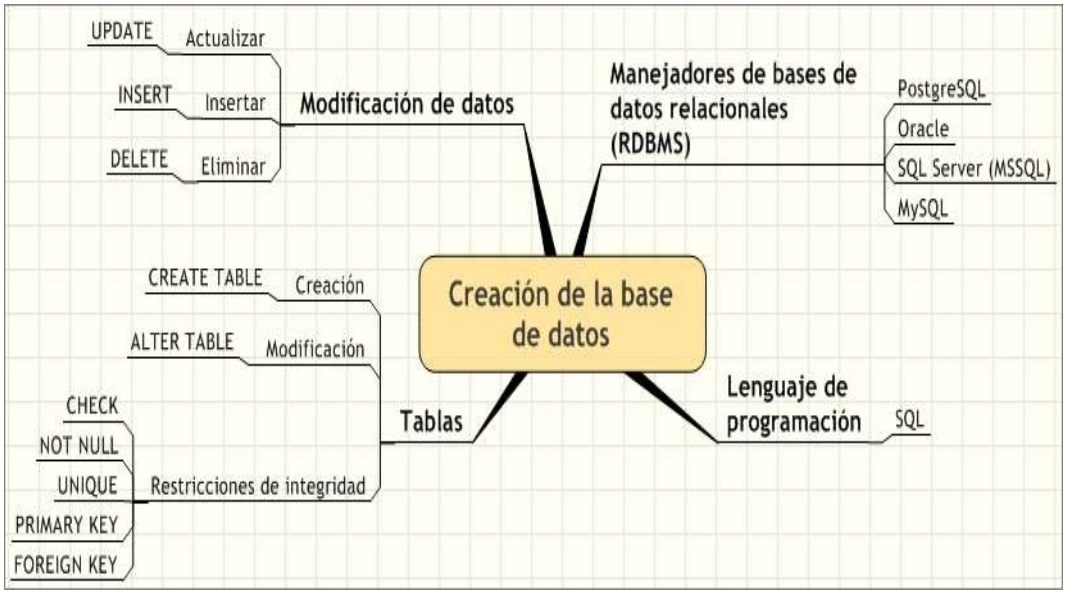
arojas=> drop table libros;
DROP TABLE

DROP TABLE ON DELETE CASCADE;

DROP TABLE DELETE CASCADE;

Atención: antes de ejecutar una operación así, hemos de estar completamente seguros de lo que hacemos, ya que en muchos RDBMS no hay vuelta atrás, es decir, no podemos recuperar ni la información que contenía la tabla ni tampoco su estructura.

RESUMEN


Creación de la base de datos
Modificación de datos
UPDATE
Actualizar
INSERT
Insertar
DELETE
Eliminar
Manejadores de bases de datos relacionales (RDBMS)
PostgreSQL
Oracle
SQL Server (MSSQL)
MySQL
Tablas
CREATE TABLE
Creación
ALTER TABLE
Modificación
Restricciones de integridad
CHECK
NOT NULL
UNIQUE
PRIMARY KEY
FOREIGN KEY
Lenguaje de programación
SQL

BIBLIOGRAFÍA

SUGERIDA

Abbey, Michael, y Michael J. Corey, Oracle guia para el principiante, México, McGraw Hill, 1996.

Oracle, México, McGraw Hill, 1995.Charte Ojeda, Francisco, SQL, Madrid, Anaya, 2005.

Connolly, Thomas M., y Carolyn E. Begg, Sistemas de bases de datos, 4ª ed., México, Addison Wesley, 2005 Dinerstein, Nelson T., Sistemas de manejo de archivos y bases de datos para microcomputadoras, México, CECSA, 1989.

Houlette, Forrest, Fundamentos de SQL, México, McGraw Hill, 2003.Koch, George, Oracle 7 manual de referencia, México, McGraw Hill, 1995.

Mendelzon / Ale Introducción a las bases de datos relacionales, México, Pearson Educación, 2000.

Oracle Education, Oracle7 RDBMS Backup and Recovery Strategias, Mexico, 1996

Piattini, Mario G., y Esperanza Marcos, Tecnología y diseño de bases de datos, Madrid, Alfaomega, 2007.
http://www.postgresql.com

Documentación en línea de PostgreSQL versión 8.4, en http://www.postgresql.org/docs/8.4/interactive/index.html (visitada el 10/07/09)

Unidad 3

Características avanzadas

OBJETIVO PARTICULAR

El alumno identificará las claves principales de la base de datos para elaborar vistas e índices, consultas para la vista creada, a través del lenguaje de programación SQL.

TEMARIO DETALLADO

(12-horas)

3. Características avanzadas

3.1. Vistas e índices
3.2. Secuencias
3.3. Cursores
3.4. Triggers
3.5. Procesamiento de transacciones
3.6. Características Objeto/Relacionales
3.7. Programación en base de datos

INTRODUCCIÓN

Existen diferentes maneras de efectuar consultas sobre varias tablas, llevando a cabo uniones, intersecciones, productos cartesianos o, en la mayoría de los casos, uniones naturales.

3.1 Vistas e índices

Definición de una vista

Una vista es una consulta prefabricada que se guarda en la propia base de datos, como se guarda la información de las filas de una tabla. Cada vista tiene un nombre, que se establece en el momento de la definición y una vez creada actúa básicamente como si de una tabla más se tratara, logrando ejecutar sobre ella cualquier sentencia de consulta.

La sintaxis que se utiliza para crear una nueva vista es la siguiente:

CREATE VIEW nombre_vista as consulta

Inicialmente la consulta puede ser de cualquier tipo, haciendo referencia incluso a otras vistas que fueron definidas con anterioridad. El siguiente ejemplo crea una vista llamada socios_préstamos.

```
arojas=> create view socios_prestamos as select apellido_paterno,
apellido_materno, prestamos
arojas-> from socios inner join prestamos on prestamos.nif = socios.nif;
CREATE VIEW
```

Uso de una vista e índices

Tras la creación, una vista aparece a nuestros ojos como una tabla más y por lo tanto, podemos usarla en una consulta como haríamos con cualquier otra tabla. Esta consulta, por ejemplo, recupera todas las filas y columnas generadas por la vista.

```
arojas=> select * from socios_prestamos;
 apellido_paterno | apellido_materno |         prestamos

        arojas=> select * from socios_prestamos;
         apellido_paterno     apellido_materno          prestamos
         Charte               Luque               (1,62877137F,4,2005-07-02)
         Arias                Trera               (2,23727319S,7,2005-07-03)
```

```
 Charte          Ojeda          (3,74381725T,10,2005-07-05)
(3 filas)
```

Podemos aplicar filtros, combinaciones y en general, usar las cláusulas habituales de una sentencia SELECT. La comodidad es que la vista no es una tabla que esté duplicando información existente en otras tablas de la base de datos, sino una consulta guardada que se ejecuta en el momento en que accedemos a ella y por tanto, siempre nos ofrece la información actualizada.

Creación de índices

Se denomina índice a una secuencia ordenada de datos, únicos o no, procedentes de una o más columnas de una tabla. El RDBMS almacena el índice internamente, muchas veces creándolos de manera automática, y lo utiliza para acelerar un conjunto de operaciones, especialmente la búsqueda de datos cuando en una cláusula WHERE se emplean las columnas que forman el índice, la agrupación y la ordenación.

Los índices ocupan espacio en la base de datos; cuantas más filas haya en la tabla a la que está asociado, mayor será el número de entradas en el índice y, por tanto, su tamaño. En cambio, operaciones que podrían necesitar varios minutos, cuando se opera con tablas que contienen decenas o cientos de miles de filas, se ejecutan en pocos segundos. Debe existir un cierto equilibrio entre la ocupación en disco y la mejora de rendimiento, relativamente fácil de alcanzar si reducimos el uso de índices a aquellas columnas que se utilicen con mayor frecuencia en las búsquedas y selección de filas.

El estándar SQL no cuenta con ninguna sentencia que facilite la creación o la eliminación de índices a demanda, ya que deben ser los RDBMS los que automáticamente generen los índices allí donde sea necesario. Por regla general, se crearán índices asociados a todas las restricciones PRIMARY KEY y UNIQUE,

puesto que dichas columnas son las perfectas candidatas para realizar la selección y búsqueda de filas. En la tabla libros, por ejemplo, la columna código es la clave primaria y sólo por ello, el RDBMS generará un índice ordenado con sus valores.
A pesar de no existir un acuerdo de estándar, la mayoría de los RDBMS que permiten la creación de índices a demanda del usuario cuentan con la sentencia CREATE INDEX ajustada a la siguiente sintaxis.

```
CREATE [UNIQUE] INDEX nombreindice
ON tabla (columnas)
```

La opción UNIQUE indica que la columna o columnas de donde se tomarán los datos para producir el índice no contendrán valores duplicados, por lo que puede asumirse que existirá únicamente una entrada por valor en el índice. Sería el caso del índice creado con la sentencia mostrada a continuación.

```
CREATE UNIQUE INDEX indsignatura
ON libros (signatura);
```

Este índice sería una secuencia ordenada con todas las signaturas de los libros, un dato que sabemos no se repite. Gracias a este índice, la búsqueda de un libro a través de su signatura, en lugar del código que actúa como clave principal, sería mucho más ágil.

También en columnas que tienen valores duplicados un índice puede resultar de utilidad. Supón que con bastante frecuencia tienes que efectuar búsquedas en la tabla libros para encontrar todos los títulos de un determinado autor, usando para ello la columna autor en una cláusula WHERE. Puesto que esa columna no es la clave primaria ni tiene una restricción UNIQUE, de hecho puede repetir su valor ya que un mismo autor puede tener varios libros, por defecto no tendrá un índice. Esto significa que para encontrar todas sus obras el RDBMS tendrá que recorrer de

principio a fin todas las filas de la tabla, comparando la columna autor en cada una de ellas para realizar si procede incluirla en el resultado.

Una operación como ésta puede acelerarse considerablemente creando un índice así:

CREATE INDEX indautor
ON libros (autor);

Ahora, al efectuar la búsqueda citada, el RDBMS usará el índice para localizar el autor, no mediante una búsqueda secuencial sino con un algoritmo mucho más rápido ya que el índice es una lista ordenada de valores. De la entrada del índice obtendrá la posición de todas las filas que corresponden a ese autor, incluyéndolas en el resultado de la consulta o vista. Como es fácil suponer, esta operación requiere mucho menos tiempo para recorrer la tabla de principio a fin.3

3.2. Secuencias

La mayoría de los RDBMS disponen de mecanismos propios que facilitan la generación automática de secuencias numéricas.

Sabemos que podemos obtener el máximo valor existente en la columna código de libros con una consulta del tipo SELECT MAX (código) FROM libros y también que una sencilla operación aritmética bastaría para incrementar ese valor máximo en una unidad. El método más directo consistiría en introducir esa subconsulta en la lista de valores por insertar en la fila, de la siguiente manera:

[3] Algunos de los RDBMS cuentan también con una sentencia ALTER INDEX, que permite modificar y regenerar el índice, y casi todos ellos disponen de la sentencia DROP INDEX que hace posible la eliminación de un índice siempre que se conozca su nombre.

arojas=> insert into libros values (((select max (codigo) from libros) + 1),'T CHA exc', 'Excel 2003', 'Charte, Francisco', 'S');
INSERT 143401 1

3.3. Cursores

Hasta ahora hemos estado utilizando variables de tipos simples, como INTEGER o VARCHAR, capaces de alojar en su interior un único valor. A estos tipos se añaden otros algo más complejos, entre los cuales destacan CURSOR y TABLE. Con ellos es posible operar sobre conjuntos de datos, en lugar de datos individuales.

Una variable de tipo TABLE es como una tabla en memoria, pero no es necesario usar la sentencia CREATE TABLE sino que se define como cualquier otra variable, con la sentencia DECLARE.

Como las demás variables, éstas se destruyen automáticamente en cuanto el procedimiento almacenado o función finaliza, no almacenándose en la base de datos como ocurre con las tablas normales.

Por su parte una variable de tipo CURSOR permite recorrer una a una las filas de un conjunto de datos, generalmente en el interior de un loop WHILE.
La variable declarada como CURSOR actúa a modo de apuntador, señalando la fila sobre la que se actuará en cada momento.

Los cursores y las sentencias específicas para su manipulación como DECLARE CURSOR FOR, forman parte del estándar SQL.

Declaración de un cursor

Las variables de tipo CURSOR se declaran como cualquiera otra, siguiendo la sintaxis:

DECLARE @variable CURSOR

Es al asignar valor a esta variable donde encontraremos la diferencia, ya que en ella no se guardará un valor puntual sino el apuntador de acceso a un conjunto de datos. La asignación se efectúa de la siguiente manera:

```
SET @variable = CURSOR FOR
        SELECT columnas FROM tabla | vista
        [otras clausulas]
```

Tras la palabra FOR disponemos de una consulta, una sentencia SELECT en la que tienen cabida todas las cláusulas y parámetros que hemos visto.

```
arojas=> fetch forward 5 in cursor_libros;
 codigo     signatura              titulo                    autor              disponible
1           I PIL cap       El capitan calzoncillos      Pilkey, Dav           S
2           I MAS mis       El misterio del perro secuestrado  Masters, M      S
3           I LI, sec       El secreto de los pirats     Li                    S
4           I DIE, mia      Mi amigo agapito             Diez Barrio, German   N
5           I FAR, unt      Un cesto lleno de lapices    Farias, Juan          S
(5 filas)
```

```
arojas=> fetch backward 1 in cursor_libros;

 codigo   signatura        titulo              autor              disponible
 4        I DIE, mia    Mi amigo agapito   Diez Barrio, German   N
 (1 fila)
```

```
arojas=> close cursor_libros;
CLOSE CURSOR
arojas=> commit work;
COMMIT
```

3.4. Triggers

Tanto los procedimientos almacenados como las funciones representan grupos de sentencias que son ejecutadas por el RDBMS a demanda de las aplicaciones, en el momento en que tiene lugar la invocación a través del nombre que se les ha asignado. En ningún caso una función o un procedimiento se ejecutarán por sí mismos.

Un desencadenador, disparador o *trigger*, todas estas denominaciones son equivalentes, es un procedimiento que se ejecuta automáticamente antes o después de que se produzca un cierto evento: la inserción de una nueva fila, la eliminación de una fila o una actualización. Por su propia naturaleza, este tipo de procedimiento no puede tomar parámetros de entrada ni tampoco devolver resultado alguno, pero a cambio pueden acceder a los valores de las columnas asociadas a la operación que ha desencadenado su ejecución, en el caso de las actualizaciones tanto a los nuevos como a los antiguos.

Mediante un desencadenador (*trigger*) que se ejecuta antes de completar la operación a la que está asociado es posible efectuar comprobaciones, aunque en la práctica éstas siempre resultan más eficientes si se implementan en forma de restricciones en la definición de la tabla. Los desencadenadores ejecutados tras completarse la operación suelen utilizarse para efectuar tareas adicionales, como el registro histórico de datos, la actualización paralela de otra información, etc. También es posible escribir desencadenadores que sustituyan la acción de la sentencia original.

Definición de un desencadenador (trigger)

Los desencadenadores se definen mediante la sentencia CREATE TRIGGER, tras la cual habrá que indicar el nombre que deseamos darle, el momento en que debe producirse, la acción y tabla a la que va asociado.

La sintaxis estándar es la siguiente:

```
CREATE TRIGGER nombre
BEFORE | AFTER| INSTEAD OF
INSERT   | DELETE| UPDATE [OF columnas]
ON nombre_tabla
[REFERENCING alias]
[FOR EACH ROW | STATEMENT]
[WHEN (condicion)]
```

Sentencias SQL

Como puede observarse, la notación es algo más compleja que en el caso de los procedimientos almacenados y funciones.

3.5. Procesamiento de transacciones

Al igual que la inserción de datos, la modificación y eliminación son operaciones que se verán afectadas por las transacciones en curso que pudieran existir en el RDBMS, cualquier cambio que efectuemos se perderá si no terminamos ejecutando la sentencia COMMIT.

Ante operaciones potencialmente peligrosas, como lo son la sustitución de datos por otros o la eliminación de estos, su inclusión en el contexto de una transacción puede evitarnos un serio disgusto. Puesto que ya conocemos las sentencias

COMMIT y ROLLBACK únicamente necesitamos saber cómo iniciar una transacción a petición para proteger de fallas las actualizaciones y dar de baja. Según el estándar SQL la sentencia se que usa es START TRANSACTION.

Tras iniciar una transacción, cualquier operación que efectuemos no será realmente confirmada hasta que no usemos la sentencia COMMIT. Esto significa que si de forma accidental, por un error, modificamos o eliminamos las filas inadecuadas, siempre podemos usar ROLLBACK para cancelar las operaciones que forman parte de la transacción, volviendo a dejar la base de datos en el estado previo a STAR TRANSACTION.

Supón que quieres eliminar de la tabla préstamos una serie de filas, correspondientes a los libros entregados hoy por los socios en la biblioteca, usando para ello una sentencia DELETE con un criterio de selección asociado. Sin embargo, al escribir la sentencia:
DELETE FROM préstamos;

Introduces el punto y coma al final accidentalmente, antes de haber escrito la cláusula WHERE y la condicional de selección, provocando que todas las filas de la tabla préstamos se pierdan. Nada habría ocurrido, si con anterioridad, hubiese iniciado una transacción. Observa la secuencia de sentencias ejecutadas a continuación y los resultados que generan:

Transacciones y concurrencia

Has usado ocasionalmente las sentencias COMMIT Y ROLLBACK para confirmar o cancelar una transacción con el fin de escribir los cambios efectuados en las tablas. Ciertos RDBMS, como en el caso de PostgreSQL se inician de una forma explicita, con la sentencia START TRANSACTION.

Con las transacciones, utilizadas adecuadamente, se garantiza la integridad de la información, una integridad que puede verse comprometida en dos situaciones diferentes:

Cuando una operación conlleva modificaciones en más de una tabla de la base de datos.
Cuando dos o más usuarios acceden de manera simultanea al mismo dato para editarlo.

Supón que está registrando el préstamo de un libro, operación que consta de varios pasos asignación del valor N a la columna disponible del libro prestado y regresado en el mismo día. Si entre estas dos operaciones, la sentencia UPDATE libros, se produce algún tipo de error en el servidor de datos, por ejemplo un problema de alimentación, el libro podría aparecer como prestado pero los datos del préstamo no han llegado a registrarse

Este problema se soluciona introduciendo las dos acciones, de actualización en el contexto de sendas transacciones. Esto garantiza el carácter atómico de la operación, al ejecutarse los pasos. Por ejemplo: Ventana emergente (pop-up) a partir de la palabra “ejemplo” que aparecerá en botón que cambia de color de manera intermitente, el cambio de color no debe ser muy rápido.

```
arojas=>
arojas=> start transaction;
START TRANSACTION
arojas=> commit;
COMMIT
arojas=> update libros
arojas-> set disponible = 'N'
arojas-> where codigo = 1;
UPDATE 1

arojas=> select * from libros;
```

codigo	signatura	titulo	autor	disponible
2	I MAS mis	El misterio del perro secuestrado	Masters, M	S
3	I LI, sec	El secreto de los pirats	Li	S
4	I DIE, mia	Mi amigo agapito	Diez Barrio, German	N
5	I FAR, unt	Un cesto lleno de lapices	Farias, Juan	S
6	T CHA, sse	Fabulas mitologicas	Charte, Francisco	S
7	T CHA, php	Leyendas de las calles de la ciudad de mexico	Dios, Juan de	N
8	T CHA, htm	Platero y yo	Ramon, Juan	S
9	G ESL, uni	En busca del unicornio	Eslava Galan, Juan	S
10	G MUN, ven	Ventanas de Nueva York	Munoz Molina, Antonio	N
11	G BAL, cli	Cuentos libertinos	Balzak, H.	
12	G SHA inc	La incognita Newton	Shaw, Catherine	S
13	T CHA exc	Excel 2003	Charte, Francisco	S
14	T CLI sch	Curso de Linux	Schroder	S
15	A INF fca	Curso de Rdbms	Armando Rojas Marin	S
1	I PIL cap	El capitan calzoncillos	Pilkey, Dav	N

(15 filas)

```
arojas=> start transaction;
START TRANSACTION
arojas=> commit;
COMMIT
arojas=> update libros
arojas-> set disponible = 'S'
arojas-> where codigo = 1;
UPDATE 1
```

```
arojas=> select * from libros;
```

codigo	signatura	titulo	autor	disponible
2	I MAS mis	El misterio del perro secuestrado	Masters, M	S
3	I LI, sec	El secreto de los pirats	Li	S
4	I DIE, mia	Mi amigo agapito	Diez Barrio, German	N
5	I FAR, unt	Un cesto lleno de lapices	Farias, Juan	S
6	T CHA, sse	Fabulas mitologicas	Charte, Francisco	S
7	T CHA, php	Leyendas de las calles de la ciudad de mexico	Dios, Juan de	N
8	T CHA, htm	Platero y yo	Ramon, Juan	S
9	G ESL, uni	En busca del unicornio	Eslava Galan, Juan	S
10	G MUN, ven	Ventanas de Nueva York	Munoz Molina, Antonio	N
11	G BAL, cli	Cuentos libertinos	Balzak, H.	
12	G SHA inc	La incognita Newton	Shaw, Catherine	S
13	T CHA exc	Excel 2003	Charte, Francisco	S
14	T CLI sch	Curso de Linux	Schroder	S

```
15      A INF fca       Curso de Rdbms                  Armando Rojas Marin     S
1       I PIL cap       El capitan calzoncillos         Pilkey, Dav             S
(15 filas)

arojas=> start transaction;
START TRANSACTION
arojas=> commit;
COMMIT
arojas=> begin;
BEGIN
arojas=> delete from prestamos;
DELETE 4
arojas=> rollback;
ROLLBACK

arojas=> select * from prestamos;
 id_prestamo            nif             codigo          fecha_prestamo
2       23727319S  7       2005-07-03
3       74381725T  10      2005-07-05
1       62877137F  4       2005-07-02
4       23727319S  1       2007-06-22
(4 filas)
```

3.6. Características Objeto/Relacionales

Introducción

La orientación a objetos es una técnica de desarrollo de software que ha mostrado considerables capacidades para resolver algunos de los problemas clásicos de la ingeniería de software. El concepto inferior a la tecnología de objetos es que todo software debe construirse utilizando componentes estándar y reutilizables siempre que sea posible. Tradicionalmente, la ingeniería de software, la gestión de base de datos eran disciplinas separadas. La tecnología de base de datos se ha concentrado

en los aspectos estáticos del almacenamiento de la información, mientras que la ingeniería de software lo ha hecho en modelar los aspectos dinámicos del software. Con la llegada de la tercera generación de sistemas de gestión de bases de datos, los denominados sistemas de gestión de bases de datos orientados a objetos (SGBDOO) y sistemas de gestión de bases de datos objeto-relaciones (SGBDOR), las dos disciplinas se han combinado para permitir el modelado concurrente tanto de los datos como de los procesos que actúan sobre los mismos.

Sin embargo, existe actualmente una importante discusión en lo que respecta a esta siguiente generación de sistemas SGBD. El éxito de los sistemas relacionales en las dos décadas anteriores es evidente y los tradicionalistas consideran que es suficiente ampliar el modelo relacional con capacidades adicionales (de orientación a objetos). Otros piensan que un modelo relacional inferior resulta inadecuado para gestionar aplicaciones complejas, como el diseño asistido por computadora, la ingeniería del software asistida por computadora y los sistemas de información geográfica.

Conceptos de orientación a objetos

Abstracción, encapsulación y ocultación de la información.

La **abstracción** es el proceso de identificar los aspectos esenciales de una entidad e ignorar las propiedades no importantes. En la ingeniería del software, esto significa que nos concentramos en lo que un objeto es y en lo que hace antes de decidir cómo debe implementarse de esta forma, diferimos los detalles de implementación lo más posible, evitando así asumir compromisos que puedan resultar demasiado restrictivos en una etapa posterior. Hay dos aspectos fundamentales dentro de la abstracción: la encapsulación y la ocultación de información.

El concepto de **encapsulación** significa que un objeto contiene tanto la estructura de los datos como el conjunto de operaciones que pueden usarse para manipular el

objeto. El concepto de **ocultación de la información** significa que separamos los aspectos externos de un objeto de sus detalles internos, que quedan ocultos a ojos del mundo exterior. De esta forma, podemos modificar los detalles internos de un objeto sin afectar a las aplicaciones que lo utilizan, siempre y cuando los detalles externos sigan siendo iguales. Esto impide que una aplicación sea tan interdependiente que un pequeño cambio tenga enormes efectos en cascada. En otras palabras, la ocultación de la información proporciona un cierto tipo de independencia de los datos.

Estos conceptos simplifican la construcción y mantenimiento de aplicaciones mediante mecanismos de **modularización**. Un objeto es una 'caja negra' que puede construirse y modificarse independientemente del resto del sistema, suponiendo que la interfaz externa no se modifique. En algunos sistemas, por ejemplo Smalltalk, las ideas de encapsulación y de ocultación de la información están combinadas. En Smalltalk, la estructura de los objetos siempre está oculta y sólo es visible la interfaz de operación con los objetos. De esta manera, la estructura de los objetos puede modificarse sin afectar a ninguna aplicación que utilice el objeto.

Existen dos formas de contemplar la encapsulación: el punto de vista de los lenguajes de programación orientados a objetos (OOPL, *object-oriented programming language*) y la adaptación a las bases de datos de dicho paradigma. En algunos OOPL, la encapsulación se consigue mediante los denominados **tipos abstractos de datos** (ADT, *Abstract Data Types*). Desde este punto de vista, un objeto tiene una interfaz y una implementación. La interfaz proporciona una especificación de las operaciones que pueden realizarse sobre el objeto; la implementación está compuesta por la estructura de los datos del ADT y las funciones que implementa la interfaz. Lo único que resulta visible para otros objetos o usuarios es la interfaz. Desde el punto de vista de base de datos, se consigue una encapsulación apropiada garantizando que los programadores sólo tengan acceso a la interfaz. De esta manera, la encapsulación proporciona un cierto tipo de

independencia lógica de los datos. Podemos cambiar la implementación interna de un ADT sin cambiar ninguna de las aplicaciones que utilicen dicho ADT (Atkinson et al., 1989):

Objetos y atributos

Muchos de los conceptos importantes de orientación a objetos surgen del lenguaje de programación Simula, desarrollado en Noruega a mediados de la década de 1960 para soportar la simulación de procesos del 'mundo real' (Dahl y Nygaard, 1966), aunque la programación orientada a objetos no surgió como nuevo paradigma de programación sino hasta el desarrollo del lenguaje Smalltalk (Goldberg y Robson, 1983). Los módulos de Simula no están basados en procedimientos, como sucede en los lenguajes de programación convencionales, sino en los objetos físicos que hay que modelar en la simulación. Este enfoque parecía particularmente adecuado, ya que los objetos eran la clave de la simulación: cada objeto debe mantener una cierta información acerca de su **estado** actual, además tiene una serie de acciones (**comportamiento**) que es preciso modelar. Podemos tomar de Simula la definición del concepto de objeto.

Objeto	Una entidad inequívocamente identificable que contiene tanto los atributos que describen el estado de un objeto del "mundo real" como las asociadas con él.

El concepto de objeto es simple, pero al mismo tiempo muy poderoso: cada objeto puede definirse y mantenerse independientemente de los demás. Sin embargo, un objeto encapsula tanto un estado como un comportamiento, mientras que una entidad solo modela el estado.

El estado actual de un objeto se describe mediante uno o más **atributos** (**variables de instancia**). Por ejemplo, la sucursal situada en la calle 20 de noviembre puede

tener unos atributos que se muestran en la siguiente tabla. Los atributos pueden clasificarse como simples o complejos. Un **atributo simple** puede ser un tipo primitivo, como un entero, cadena, número real, etc., que adopte valores literales; por ejemplo, AldamaN° en la tabla es un atributo simple con el valor literal 'B003'. **Un atributo complejo** puede contener colecciones y/o referencias. Por ejemplo, el atributo personal de ventas es una **colección** de objetos de ventas. Un **atributo de referencia** representa una relación entre objetos y contiene un valor, o una colección de valores, que son ellos mismos objetos (por ejemplo personal de ventas es, para ser exactos, una colección de referencia a objetos ventas). Un atributo de referencia es conceptualmente similar a una clave externa del modelo de datos relacional o a un indice de lenguaje de programación. Un objeto que contenga uno o más atributos complejos se denomina **objeto complejo**

Atributo	**Valor**
aldamaNo	B003
calle	20 de noviembre
ciudad	Tijuana
Código postal	91120
Personal de ventas	Melany Maya;David Ford
Director	Patricia González

Tabla 3.1. Atributos de objeto para una instancia de sucursal

Para hacer referencia a los atributos se utiliza generalmente la notación 'de puntos'. Por ejemplo para hacer referencia al atributo calle de un objeto aldama se utilizaría:

aldamaObject.calle

Identidad de los objetos

Una parte clave de la definición de un objeto es que debe poseer una identidad unívoca. En un sistema orientado a objetos, a cada objeto se le asigna un

identificador de objetos (OID, Object Identifier) en el momento de su creación; dicho identificador:

- Es generado por el sistema.
- Es exclusivo de dicho objeto.
- Es invariante, en el sentido de que no puede alterarse durante la vida del objeto. Una vez que se crea el objeto, no se reutilizará este OID para ningún otro objeto, ni siquiera después de borrar el objeto.
- Es independiente de los valores de sus atributos (es decir, de su estado). Dos objetos podrían tener el mismo estado pero seguirían teniendo diferentes identidades.
- Es invisible al usuario (idealmente).

De este modo, la identidad de los objetos garantiza que un objeto pueda ser siempre unívocamente identificado, proporcionando así automáticamente una integridad de las entidades. De hecho como la identidad de los objetos garantiza la unicidad en todo el sistema, proporciona una restricción más fuerte que la integridad de entidades empleada en el modelo de datos relacional, que solo requiere la unicidad dentro de una relación. Además, los objetos pueden contener, o hacer referencia a otros objetos, utilizando las correspondientes identidades. Sin embargo, para cada OID al que se haga referencia dentro del sistema deberá siempre haber presente un objeto que se corresponda con dicha OID, es decir, no debe haber ninguna **referencia dependiendo**. Por ejemplo, tenemos la relación Aldama Personal de ventas. Si alojamos cada objeto sucursal en el objeto de empleado correspondiente, nos encontraremos con los problemas de la redundancia de la información y de las anomalías de actualización. Sin embargo, si lo que hacemos es incluir la OID del objeto sucursal en el objeto de empleado correspondiente, continuará habiendo una sola instancia de cada objeto sucursal en el sistema y podrá mantenerse la coherencia mucho más fácilmente. De esta forma, los objetos pueden compartirse y las OID pueden usarse para mantener la integridad referencial.

Hay distintas maneras en las que puede implementarse la identidad de los objetos. En un SGBDR, la identidad de los objetos está basada en un valor. Se utiliza la clave principal para garantizar la unicidad de cada campo de una relación. Las claves principales no proporcionan el tipo de identidad de objeto que se necesita en los sistemas orientados a objetos. En primer lugar, como ya hemos indicado, la clave principal sólo es unívoca dentro de una relación, no en todo el sistema. En segundo lugar, la clave principal se elige generalmente a partir de los atributos de la relación, lo que hace que sea dependiente del estado del objeto. Si una clave potencial está sujeta a cambios, la identidad tendrá que ser simulada mediante identificadores unívocos, como por ejemplo el número de sucursal aldamaNO, pero como estos no están bajo control del sistema, no existe ninguna garantía de protección frente a violaciones de la identidad. Además, las claves simuladas tales como B001, B002, B003, tienen escaso significado semántico para el usuario.

Otras técnicas frecuentemente utilizadas en los lenguajes de programación para soportar la definición de identidades, son los nombres de variables y los índices (o direcciones de memoria virtuales), pero estos sistemas también ponen en riesgo la identidad de los objetos. Por ejemplo, en ‘C’ y C++, una OID es una dirección física dentro del espacio de memoria de un proceso. Para la mayoría de los propósitos para los que se utiliza una base de datos, este espacio de direcciones es demasiado pequeño: las consideraciones de escalabilidad requieren que las OID sean válidas para los distintos volúmenes de almacenamiento y posiblemente para las distintas computadoras que componen un SGBD distribuido. Además, cuando se borra un objeto, la memoria anteriormente ocupada por el mismo debe reutilizarse, por lo que podría crearse un nuevo objeto y asignársele el mismo espacio que ocupaba el objeto borrado. Todas las referencias al antiguo objeto, que habían dejado de ser válidas después del borrado, pasan de nuevo a ser válidas, pero desafortunadamente hacen referencia al objeto equivocado. De forma similar, si se mueve un objeto de una dirección a otra, se invalida su identidad. Lo que

necesitamos es un identificador lógico del objeto que sea independiente tanto del estado como de la ubicación.

La utilización de identificadores OID como mecanismos de definición de la identidad de los objetos presenta varias ventajas:

- Son eficientes. Las OID requieren un espacio mínimo de almacenamiento dentro de un objeto complejo. Normalmente, son más pequeñas que los nombres textuales, que las claves externas o que otras referencias de carácter semántico.
- Son rápidas. Las OID apuntan a una dirección real o a una ubicación dentro de una tabla que proporciona la dirección del objeto referenciado. Esto significa que pueden localizarse los objetos rápidamente, con independencia de si están almacenados actualmente en memoria local o en disco.
- No pueden ser modificadas por el usuario. Si las OID son generadas por el sistema y se mantienen invisibles, o al menos de solo lectura, el sistema puede garantizar más fácilmente la integridad de entidades y la integridad referencial. Además, esto evita que el usuario tenga que mantener la integridad por sí mismo.
- Son independientes del contenido. Las OID no dependen de ninguna manera de los datos contenidos en el objeto. Esto permite que se modifique el valor de cada atributo de un objeto, pero que el objeto siga siendo el mismo objeto con la misma OID:

Observa la potencial ambigüedad que puede surgir debido a esta última propiedad: dos objetos pueden parecer iguales a los ojos del usuario (todos los valores de los atributos son iguales) pero tienen diferentes OID, por lo que se tratará de diferentes objetos. Si las OID son invisibles, ¿Cómo distingue el usuario entre estos dos objetos? Podemos concluir de esto que seguirán siendo necesarias las claves principales para permitir que los usuarios distingan entre los objetos. Con esta técnica de designación de un objeto, podemos distinguir entre dos conceptos

diferentes: identidad de los objetos (algunas veces denominada equivalencia de objetos) e igualdad de los objetos. Dos objetos serán **idénticos** (equivalentes) sí y solo si son el mismo objeto (lo que se denota mediante '='), es decir, si sus OID son iguales. Dos objetos serán **iguales** si sus estados son iguales (lo que se denota mediante '=='). También podemos distinguir entre igualdad somera e igualdad profunda: los objetos tienen una **igualdad somera** si sus estados contienen los mismos valores cuando excluimos las referencias a otros objetos; los objetos mostrarán una **igualdad profunda**, si sus estados contienen los mismos valores y si los objetos relacionados también contienen los mismos valores.

Métodos y mensajes

Un objeto encapsula datos como funciones en un paquete autocontenido. En la tecnología orientada a objetos, las funciones se suelen denominar **métodos**. La siguiente figura proporciona una representación conceptual de un objeto, con los atributos interiores protegidos del exterior mediante los métodos: estos definen el **comportamiento** del objeto. Pueden utilizarse para cambiar el estado del objeto modificando los valores de sus atributos, o para consultar los valores de una serie de atributos seleccionados. Por ejemplo, podemos tener métodos para añadir un nuevo inmueble en alquiler en una sucursal, para actualizar el salario de un empleado o para imprimir los detalles relativos a un empleado.

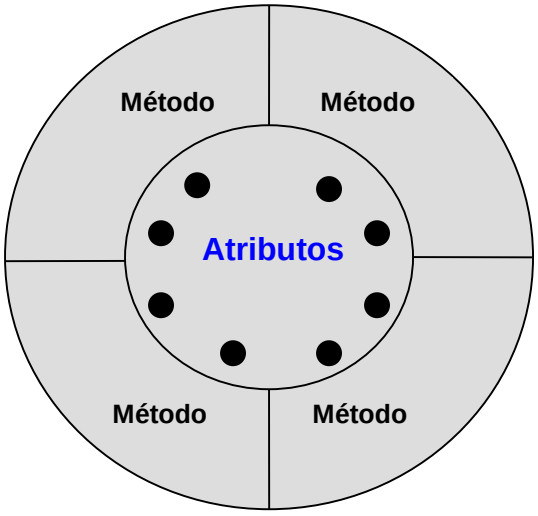


Figura 3.1. Un objeto que muestra tantos atributos como métodos

Un método está compuesto de un nombre y de un cuerpo que implementa el comportamiento asociado con el nombre del método. En un lenguaje orientado a objetos, el cuerpo está compuesto de un bloque de código que implementa la funcionalidad requerida, la figura siguiente representa el método para actualizar el salario de un empleado. El nombre del método es update Salario, con un parámetro de entrada increment, que se añade a la **variable de instancia** salario para generar el nuevo salario.

```
Method void update Salario (float incremento)
{
Salario = salario + incremento;
}
```

Figura 3.2. Ejemplo de método

Los **mensajes** son los medios por los que los objetos se comunican. Un mensaje es simplemente una solicitud que un objeto (el emisor) envía a otro objeto (el receptor) pidiendo ese segundo objeto que ejecute uno de sus métodos. El emisor y el receptor pueden ser el mismo objeto. De nuevo, se utiliza generalmente la

notación de puntos para acceder a los métodos. Por ejemplo, para ejecutar el método update salario sobre un objeto ventas, ventasObject, y pasar al método un valor de incremento igual a 1000, escribiríamos:

ventasObject. Update salario (1000)

En un lenguaje de programación tradicional, un mensaje se escribiría como una llamada a una función:

update salario(ventasObject, 1000)

Clases

En Simula, las clases son patrones que sirven para definir conjuntos de objetos similares. Así, los objetos que tienen los mismos atributos y responden a los mismos mensajes pueden agruparse para formar una **clase**. Los atributos y métodos asociados se definen una única vez para la clase, en lugar de definirlos separadamente para cada objeto. Por ejemplo, todos los objetos sucursales se describirían mediante una única clase Aldama. Los objetos en una clase se llaman **instancias** de la clase. Cada instancia tiene su propio valor o valores para cada atributo, compartiendo los mismos nombres de atributos y los mismos métodos con las demás instancias de la clase, como se ilustra más abajo en la Figura 3.3.

En la literatura, a menudo se usan como sinónimos los términos ‘clase’ y ‘tipo’, aunque algunos autores hacen una distinción entre los dos términos, como a continuación se describe. Un tipo corresponde a la noción de un modelo abstracto de datos (Atkinson y Buneman, 1989). En los lenguajes de programación, las variables se declaran como de tipo concreto. El compilador puede usar dicho tipo para comprobar que las operaciones realizadas con la variable son compatibles con el modelo, ayudando así a garantizar la corrección del software. Por el contrario, una clase es un patrón para la creación de objetos y proporciona métodos que

pueden aplicarse a dichos objetos. Así, las referencias a las clases se realizan en tiempo de ejecución, en lugar de hacerse en tiempo de compilación.

En algunos sistemas orientados a objetos, las clases son también objetos y tienen sus propios atributos y métodos, a los que se denominan **atributos de clase** y **métodos de clase**, respectivamente. Los atributos de clase describen las características generales de la clase, como los valores totales o promedios; por ejemplo, en la clase Aldama podemos tener un atributo de clase que nos dé el número total de sucursales. Los métodos de clase se utilizan para cambiar o consultar el estado de los atributos de clase. También hay métodos de clase especiales para crear nuevas instancias de la clase y destruir aquellas que ya no sean necesarias. En un lenguaje orientado a objetos, las nuevas instancias se crean normalmente mediante un método denominado new. Dichos métodos se denominan habitualmente **constructores**. Los métodos para destruir objetos y reclamar el espacio ocupado por los mismos se suelen denominar **destructores**. Los mensajes enviados a un método de clase se envían a la clase, en lugar de a una instancia de la misma, lo que implica que la clase es una instancia de nivel superior, denominada **metaclase**.

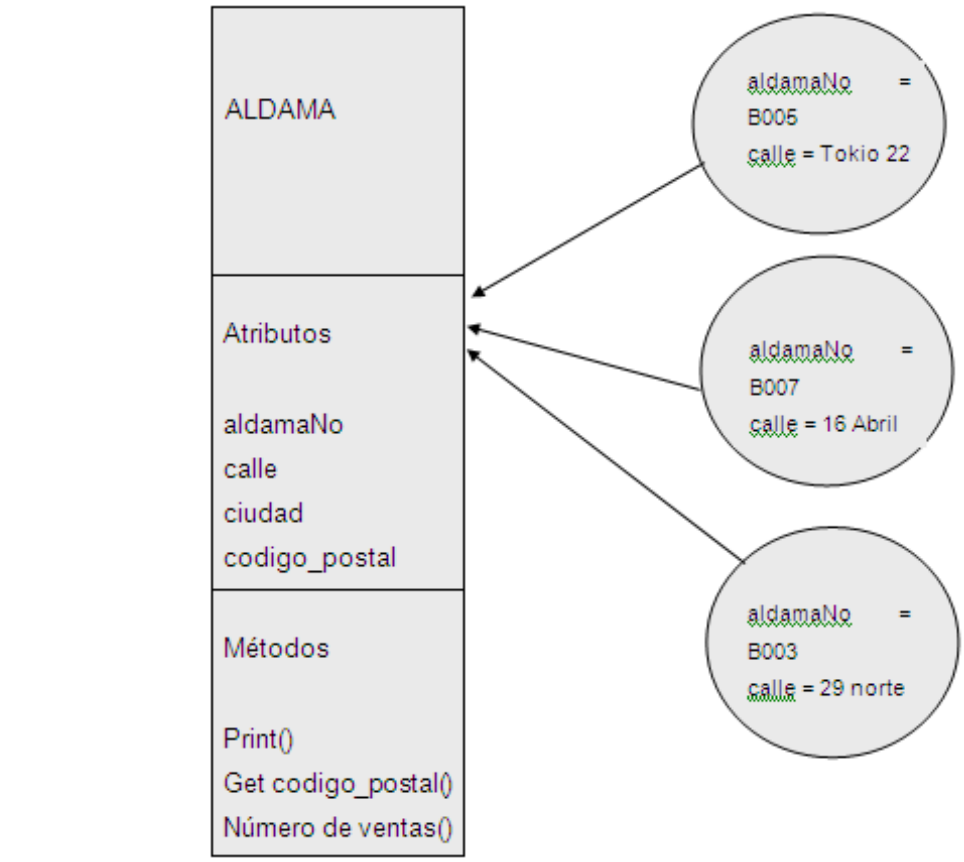


Figura 3.3. Las instancias de una clase comparten los atributos y los métodos

Subclases, superclases y herencias

Algunos objetos pueden tener atributos y métodos similares, pero no idénticos. Si hay un alto grado de similitud, sería útil poder compartir las propiedades comunes (atributos y métodos). La **herencia** permite definir una clase como un caso especial de otra clase más general. Estos casos especiales se conocen con el nombre de **subclases**, mientras que los casos más generales se llaman **superclases.** El proceso de formar una superclase se denomina **generalización** y el proceso de formar una subclase, **especialización**. De manera predeterminada, las subclases

heredan todas las propiedades de sus superclases, definiendo adicionalmente sus propias propiedades distintivas. Sin embargo, como veremos en breve, las subclases también pueden redefinir las propiedades heredadas. Todas las instancias de las subclases son también instancias de la superclase. Además, el principio de sustitubilidad dice que podemos utilizar una instancia de la subclase en todos aquellos casos en que un método o una estructura de programación esté esperando una instancia de la superclase.

Los conceptos de superclase, subclase y herencia son similares a los presentados para el modelo EER (*Enhanced Entity-Relationship*), salvo porque en el paradigma de orientación a objetos la herencia cubre tanto el estado como el comportamiento. La relación entre la subclase y la superclase se denomina en ocasiones relación **A KIND OF** (AKO, UN TIPO DE), pudiéndose decir por ejemplo que Director es AKO Ventas. La relación entre una instancia y su clase se denomina en ocasiones **IS-A**; por ejemplo Susana Brand IS-A Director.

Hay diversas formas de herencia: herencia simple, herencia múltiple, herencia repetida y herencia selectiva. La Figura 3.4. Muestra un ejemplo de **herencia simple**, en que las subclases director y personal de ventas heredan las propiedades de la superclase Ventas. El término ‘herencia simple’ hace referencia al hecho de que las subclases heredan sus atributos de una única superclase. La superclase Ventas podría ser a su vez una subclase de otra superclase, Person, formándose así una **jerarquía de clases**.

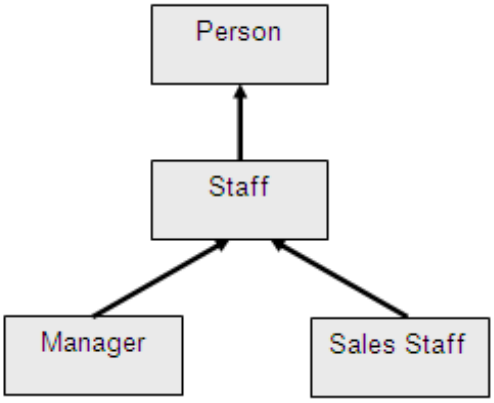


Figura 3.4. Herencia simple

La Figura 3.5 muestra un ejemplo de **herencia múltiple**, en el que la subclase Director de Ventas hereda propiedades tanto de la superclase Director como de Personal de Ventas. La provisión de un mecanismo de herencia múltiple puede ser bastante problemática, ya que se necesita proporcionar una manera de tratar con los conflictos que surgen cuando las superclases contienen los mismos atributos o métodos. No todos los lenguajes orientados a objetos y sistemas SGBD soportan la herencia múltiple en principio. Algunos autores afirman que la herencia múltiple introduce un nivel de complejidad difícil de solucionar de manera segura y coherente.

La Figura 3.5 muestra un ejemplo de **herencia múltiple**, en el que la subclase Director de Ventas hereda propiedades tanto de la superclase Director como de Personal de Ventas. La provisión de un mecanismo de herencia múltiple puede ser bastante problemática, ya que se necesita proporcionar una manera de tratar con los conflictos que surgen cuando las superclases contienen los mismos atributos o métodos. No todos los lenguajes orientados a objetos y sistemas SGBD soportan la herencia múltiple en principio. Algunos autores afirman que la herencia múltiple introduce un nivel de complejidad difícil de solucionar de manera segura y coherente.

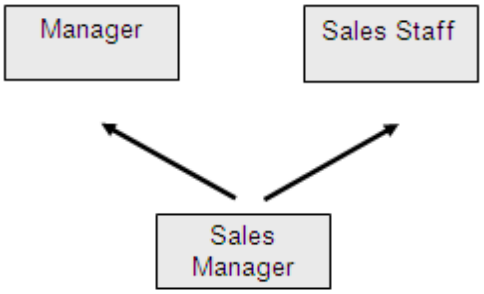


Figura 3.5. Herencia múltiple

Otros argumentan que es absolutamente necesaria para modelar el 'mundo real', como en este ejemplo. Los lenguajes que soportan la herencia múltiple, tratan de resolver los conflictos de diversas formas, como, por ejemplo:

Incluir ambos nombres de atributos/métodos y utilizar el nombre de la superclase como cualificador. Por ejemplo, si bonus es un atributo tanto de Manager como SalesStaff, la subclase SalesManager podría heredar bonus tanto de ambas superclases y cualificar la instancia de bonus en SalesManager como Manager.bonus o SalesStaff.bonus.

Hacer la jerarquía de herencias y utilizar la herencia simple para evitar conflictos. Con esta técnica, la jerarquía de herencia de la Figura 3.5 se interpretaría como:

SalesManager → Manager → SalesStaff

O bien:

SalesManager → SalesStaff →Manager

Con el ejemplo anterior, SalesManager heredaría una instancia del atributo bonus, que provendría de Manager en el primer caso y de SalesStaff en el segundo caso.

Requerir al usuario que redefina los atributos o métodos conflictivos

Generar un error y prohibir la definición hasta que el conflicto se resuelva.

La **herencia repetida** es un caso especial de herencia múltiple en el que las superclases heredan de otra superclase común. Ampliando el ejemplo anterior, las clases Manager y SalesStaff pueden heredar propiedades de una superclase común Staff, como se ilustra en la Figura 3.6. En este caso, el mecanismo de herencia debe garantizar que la clase SalesManager no herede propiedades de la clase Staff dos veces. Los conflictos pueden resolverse como hemos explicado para el caso de la herencia múltiple.

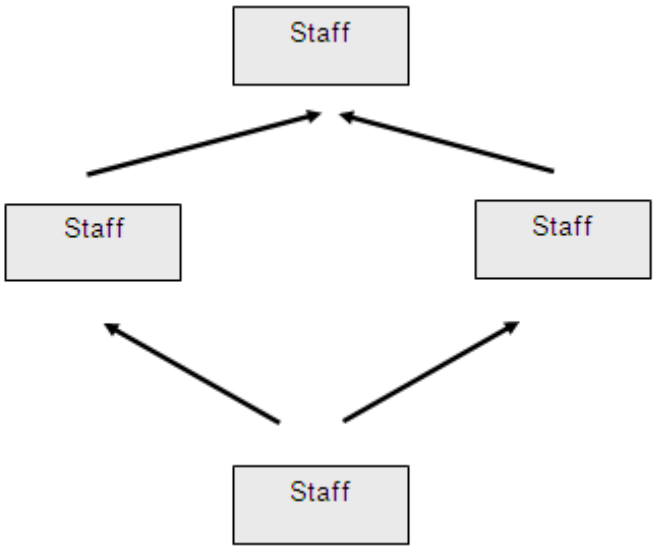


Figura 3.6. Herencia repetida

La **herencia selectiva** permite que una subclase herede un número limitado de propiedades de la superclase. Esta característica puede proporcionar una funcionalidad similar al mecanismo de vistas, al restringir el acceso a ciertos detalles, pero no a otros.

Polimorfismo y enlace dinámico

La sobrecarga es un caso especial del concepto más general de **polimorfismo**, que proviene de la palabra griega que significa 'tener múltiples formas'. Hay tres tipos de polimorfismo: de operación, de inclusión y paramétrico (Cardelli y Wegner, 1985). La sobrecarga, es un tipo de **polimorfismo de operación** (o **polimorfismo ad hoc**). Un método definido de una superclase y heredado por sus subclases es un ejemplo de **polimorfismo de inclusión**. Finalmente, el **polimorfismo paramétrico** o **genericidad**, como algunas veces se le denomina, utiliza tipos como parámetros en las declaraciones genéricas de tipos o de clases. Por ejemplo, la siguiente definición de patrón:

```
template <type T>
T max(x:T, y T) {
If(x > y)              return x;
else                   return y;
}
```

Define una función genérica max que toma dos parámetros de tipo T y devuelve el máximo de los dos valores.

Este fragmento de código no define en la práctica ningún método. En lugar de ello, esta descripción genérica actúa como plantilla para el posterior establecimiento de uno o más métodos diferentes con tipos distintos.

Los métodos reales se distanciarían de la siguiente forma:

```
int max(int, int)          // instanciar la funcion max para dos tipos enteros
real max(real, real)  // instanciar la funcion max para dos tipos reales
```

El proceso de seleccionar el método apropiado basándose en el tipo de objeto se denomina **enlace**. Si la determinación del tipo de un objeto puede diferirse hasta el momento de la ejecución (en lugar de hacerla en tiempo de compilación), esta selección se denomina **enlace dinámico** (o **tardío**). Por ejemplo, considera la jerarquía de clases de Staff, con las subclases Manager y SalesStaff, como se muestra en la Figura 3.6 y supón que cada clase tiene su propio método print para imprimir los detalles relevantes. Supón también que tenemos una lista compuesta por un número arbitrario de objetos, por ejemplo n objetos, pertenecientes a esta jerarquía. En un lenguaje de programación convencional, necesitaríamos una instrucción CASE o una instrucción IF anidada para imprimir los detalles correspondientes:

```
FOR i = 1 TO n DO
SWITCH (list[i]. type) {
CASE staff: printStaffDetails(list[i].object); break;
CASE manager:          printManagerDetails(list[i].object); break;
CASE salesperson:      printSalesStaffDetails(list[i].object); break;
}
```

Si se añadiera un nuevo tipo a la lista, tendríamos que ampliar la instrucción CASE para gestionar el nuevo tipo, lo que nos obligaría a recompilar este fragmento de software. Si el lenguaje soporta los mecanismos de enlace dinámico y de sobrecarga, podemos sobrecargar los métodos print utilizando un único nombre y sustituir la instrucción CASE por la línea:

```
list[i].print()
```

Además, con esta técnica podemos añadir cualquier número de nuevos tipos a la lista y, siempre y cuando continuemos sobrecargando el método print, no hará falta

volver a compilar este fragmento de código. Por tanto, el concepto de polimorfismo es ortogonal a (es decir, es independiente de) la herencia.

Almacenamiento de objetos en una base de datos relacional

Una técnica para conseguir la persistencia en un lenguaje de programación orientado a objetos como C++ o Java consiste en utilizar un SGBDR como motor de almacenamiento subyacente. Esto requiere establecer una correspondencia entre las instancias de las clases (es decir, los objetos) y uno o más campos distribuidos entre una o más relaciones. Para los propósitos de nuestra explicación, considera la jerarquía de herencia que se muestra en la Figura 3.7, que tiene una superclase Staff y tres subclases: Manager, SalesPersonnel y Secretary.

Para gestionar este tipo de jerarquías de clases, tenemos que realizar dos tareas básicas:

- Diseñar las relaciones que habrán de representar la jerarquía de clases.
- Diseñar el modo en que se accederá a los objetos, lo que significa:

    - escribir el código para descomponer los objetos en campos y almacenar los objetos descompuestos en relaciones;
    - escribir el código para leer los campos de las relaciones y reconstruir los objetos.

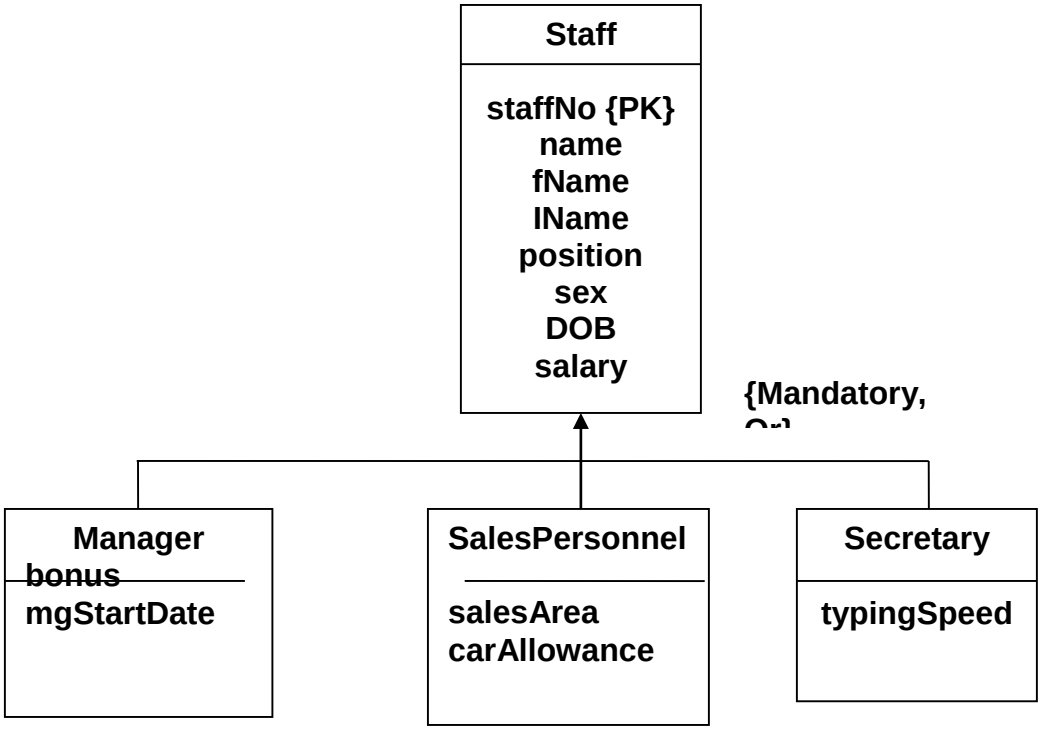


Figura 3.7. Jerarquía de herencia de ejemplo para Staff

3.7. Programación en base de datos

Este tema será uno de los más importantes. Aprender a resolver los problemas que se nos presentan a lo largo de la existencia utilizando una computadora establecida, el algoritmo adecuado y la lógica, nos lleva a un desarrollo pleno en todos los ámbitos: en el hogar, en la escuela y en el lugar de trabajo.

'Problema' es una palabra que tiene varias acepciones. Las más empleadas son:

1. Situación difícil que debe resolverse.
2. Asunto que se trata de aclarar o resolver.
3. Punto en que hay algo que averiguar o alguna dificultad.

4. Caso en la que se conocen algunos datos, mediante los cuales es posible encontrar otro que se busca.
5. Posición que nos presenta la necesidad de cambiar algo que tenemos por algo que deseamos; es decir, la satisfacción de las necesidades es una solución de los problemas.

Como se puede ver, no todos los problemas representan situaciones difíciles que han de resolverse. Algunos problemas son tan triviales como ir a la oficina todos los días. La serie de pasos lógicos que han de llevarse a cabo para hacerlo, es lo que se conoce como algoritmo.

Los problemas pueden resolverse de diversas formas; sin embargo, los que nos atañen, que requieren de las computadoras y un lenguaje de programación para proporcionar soluciones a los usuarios, siempre deben resolverse utilizando algoritmos y la lógica, ya que los circuitos de una computadora trabajan de esa manera; utilizando los operadores lógicos del algebra de Boole.

Algoritmos

Es una serie de pasos lógicos que deben seguirse para resolver un problema. Muchas palabras relativas a la aritmética provienen del árabe, debido a sus grandes adelantos en esta materia. Mohammed ben Musa (780-850 d.c.), padre del algebra, era conocido con el seudónimo de Al Jwarizmi, término que con el transcurso del tiempo degeneraría en algorismo; para convertirse finalmente en algoritmo por la influencia y similitud fonética con aritmética.

Métodos para la solución de problemas con computadora

La mejor manera de solucionar los problemas utilizando la computadora, es llevar a cabo una serie de pasos lógicos.

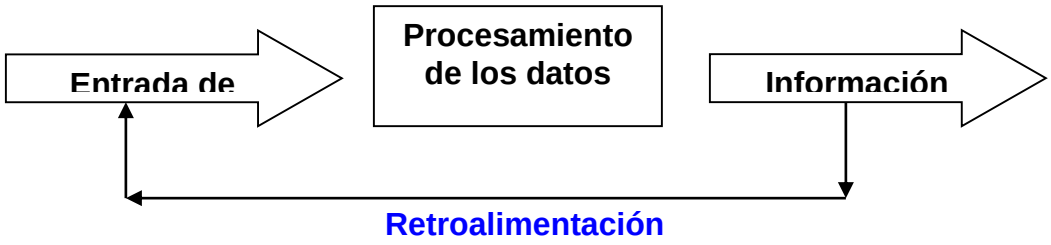
Esto permite encontrar una metodología adecuada para cada tipo de problema. La computadora, como elemento físico, no puede llevar a cabo ninguna tarea si no cuenta con un programa o instrucciones bien definidas para ello.

El desarrollador de programas de computación deberá tomar en cuenta las siguientes consideraciones, cada vez que tenga un problema que resolver.

1. **Especificación o análisis**. Es lo primero que se debe hacer antes de iniciar cualquier programa. Se debe conocer claramente y a profundidad cuál es el problema, de qué datos se dispone y qué objetivos se persiguen.
2. **Diseño del algoritmo**. Cada problema es diferente, por lo tanto, la creación del algoritmo que lo resolverá representa una gran parte del éxito o fracaso del proyecto. Los algoritmos deben ser muy **precisos**; es decir evitar rodeos innecesarios, **consistentes** para lograr obtener los mismos resultados para el mismo problema y **finitos**; es decir, que cuenten con una rutina de terminación.
3. **Codificación o programación**. Es la etapa de llevar el algoritmo a un lenguaje de programación para crear el código fuente.
4. **Depuración y verificación**. Es el proceso de ejecutar el programa para corregir los errores que pudieran presentarse y verificar los resultados con pruebas de escritorio, para certificar su buen funcionamiento.
5. **Compilación.** Una vez que se ha dado por terminada la elaboración de un programa, si se cuenta con un lenguaje de programación que incluya entre sus herramientas, un compilador, se puede crear un archivo ejecutable, que se puede ejecutar sin necesidad del lenguaje de programación.
6. **Documentación y mantenimiento**. Estos pasos a veces se dejan a un lado, porque una vez obtenido el programa final, parecería que se ha resuelto el problema por completo. Es conveniente documentar el programa para que diversos líderes de proyecto, analistas de sistemas y programadores puedan entenderlo y utilizarlo. También es conveniente mantener el código fuente

para realizar correcciones o ajustes posteriores, cuya necesidad surge después de algún tiempo de usar el programa.

En general los algoritmos deben contemplar el esquema básico de trabajo de las computadoras.



Programación

Programa es una secuencia de declaraciones e instrucciones que se ejecutan una a una para procesar una serie de datos y obtener un resultado generalmente denominado información. Consta de estructuras de datos que definen el tipo de datos que manejará el programa, operaciones elementales que permiten transformar los datos en información y estructuras de control que posibilitan que el programa se desarrolle en un orden o flujo determinado.

Una de las herramientas más útiles para la informática es la programación, pues todas las operaciones y manejo de información que realiza la computadora solo funcionan bien si el programa o (algoritmo) correspondiente se ha diseñado correctamente, mediante una secuencia lógica de instrucciones bien definidas que permiten resolver paso a paso un problema.

Los cuidados que debe tomar en cuenta todo programador cuando desarrolla un programa son:

1. Que el programa no contenga loops (bucles) o ciclos infinitos de los cuales es muy difícil salir.

2. Que el software diseñado maneje correctamente las bases de datos, tablas y archivos de tal forma que no pierda la información.
3. Que el código del programa no incluya instrucciones que puedan dejar congelada (detener un proceso o una actividad por tiempo indefinido) a la computadora –problema debido generalmente a mal manejo de los bloques o direcciones de memoria.

Lenguajes de programación

Hace apenas unas tres décadas, los desarrolladores tenían que escribir sus programas utilizando solamente el lenguaje máquina o código binario, lo que significaba un trabajo complicado y tedioso. Por tal motivo se evolucionó al lenguaje ensamblador que permite el uso de expresiones mnemotécnicas y las traduce a lenguaje de máquina. Estos son los lenguajes que se conocen como de bajo nivel, por estar limitado su uso a programadores profesionales. Algunos lenguajes de bajo nivel conocidos son Ensamblador, Easycoder, Neat y Macroassembler.

La diferencia de programar en ensamblador o en cualquier otro lenguaje de programación se encuentra en la facilidad para tomar el control absoluto de la computadora, optimizar el uso del hardware y de las unidades de entrada/salida y el tamaño reducido de los programas ejecutables que se realizan con este lenguaje.

Con el vertiginoso avance de la informática, pronto se desarrollaron los lenguajes de programación denominados de alto nivel, que al permitir la inclusión de instrucciones y comandos en lenguaje común (generalmente en inglés) quedaron al alcance de la mayoría de los usuarios. En este caso, el mismo lenguaje sirve de traductor para que las instrucciones puedan ser ejecutadas por la computadora. Estos **intérpretes** necesitan estar siempre presentes en la memoria convencional (RAM) para traducir cada instrucción o comando y ejecutarlo en el orden indicado, por lo que resultan más lentos en su operación.

Para hacer más rápida la ejecución de los programas creados usando lenguajes de alto nivel, se debe emplear un compilador. Este es, en esencia, un programa traductor que interpreta las instrucciones o comandos del lenguaje de alto nivel y los traduce al código binario que usan las computadoras, creando así un programa compilado y ejecutable.

El primer lenguaje de alto nivel fue el FORTRAN, acrónimo de FORmula TRANslator o lenguaje traductor de fórmulas. Y otros son:

ADA. Llamado así en honor de Agusta Ada Byron, reconocida como la primera programadora por sus trabajos con tarjetas perforadas al lado de Charles Babbage.

ALGOL. Acrónimo de ALGOrithmic Language o lenguaje algorítmico para la solución de problemas.

APL. Acrónimo de A Programming Language. Desarrollado en 1962 lenguaje interactivo orientado a problemas matemáticos.

APT. Acrónimo de Automatic Programmed Tools. Lenguaje de alto nivel del grupo de los lenguajes para procesos de control.

BASIC. Acrónimo de Beginner's All-purpose Symbolic Instruction Code es el más sencillo y más fácil de aprender. Aunque siempre resulto muy lento en sus procesos por ser un Interprete, ya existen paquetes como Quick BASIC, Turbo BASIC y Visual Basic, que son compiladores con capacidad de crear programas ejecutables a partir del código fuente, haciéndolos tan rápidos como aquellos que han sido elaborados con Pascal.

C. un lenguaje de programación muy compacto desarrollado por investigadores de los laboratorios Bell, que debe su éxito al sistema operativo UNIX (el cual está totalmente escrito en este lenguaje).

COBOL. Acrónimo de COmmon Business-Oriented Language o lenguaje orientado a usos comerciales. Particularmente adecuado a las operaciones matemáticas necesarias en las áreas de contabilidad y administración.

FORTH. Acrónimo de FOuRTH. Bautizado con ese nombre aludiendo a los lenguajes de cuarta (fourth) generación. Desarrollado por Charles Moore, permite al usuario hacerlo crecer de acuerdo con sus necesidades y sus principales aplicaciones son la robótica, programación de juegos electrónicos y aplicaciones matemáticas.

LISP. Acrónimo de LISt Processor. Lenguaje usado en aplicaciones de inteligencia artificial (Artificial Intelligence, AI), conocido también como Common LISP.

LOGO. Escrito por Seymour Papert, es un lenguaje de alto nivel enfocado a la enseñanza de programación a principiantes y niños.

MODULA-2. Lenguaje estructurado de alto nivel escrito por N. Wirth, que permite desarrollar módulos que trabajan independientes uno de otro.

PASCAL. Escrito en 1971 y nombrado así en honor al matemático y filósofo francés Blaise Pascal. Desarrollado por N. Wirth.

PL/1. Acrónimo de Programming Language one o lenguaje de programación número uno. Tiene uso en aplicaciones científicas, comerciales y administrativas. Fue desarrollado por IBM como una alternativa al FORTRAN, COBOL y ALGOL.

Actualmente, debido a la euforia creada por Windows, una plataforma o interfaces gráficas entre la computadora y el usuario, se ha puesto de moda la Programación Orientada a Objetos, que no es otra cosa que la utilización de rutinas o librerías de código consideradas como objetos independientes, prefabricadas por los desarrolladores de herramientas de programación. Los principales creadores de estos programas son los de Borland: Turbo Pascal, Delphi y C++, y los de Microsoft: los paquetes Visual, C++ y Visual BASIC, o el Visual Studio, que incluye además Visual Fox Pro como programa de desarrollo de base de datos, aunque existe una gran cantidad de empresas y programadores dedicados a crear este tipo de herramientas.

El manejo de grandes cantidades de datos en las empresas propició el desarrollo de lenguajes de programación tipo Xbase como dBASE, Clipper, Oracle, Fox Pro y otros, que permiten crear bases de datos y generar aplicaciones para el manejo de esa información, lo que facilita la creación de información o reportes adecuados a las necesidades de cada empresa en particular.

Fundamentos de programación estructurada

La programación estuvo enfocada durante muchos años a resolver problemas de tipo científico y matemático. Los programas, cada vez más complejos, propiciaron la creación de nuevas estructuras de control y se atendió a la necesidad de subdividir los problemas en pequeños módulos, más sencillos de analizar, de tal manera que permitan entender más claramente tanto el problema como la solución.

Programación estructurada son las técnicas que se utilizan para diseñar y escribir programas considerando plenamente el método científico, utilizando esas nuevas estructuras de control con un diseño descendente (Top-Down) y un lenguaje natural denominado pseudocódigo, de manera disciplinada.

Pseudocódigo y diagrama de flujo

El diseño de un algoritmo puede ser representado en forma de pseudocódigo o gráficamente mediante un diagrama de flujo, por lo tanto se dice que un diagrama de flujo o fluxograma es la representación gráfica de todos los pasos de un algoritmo. Del diseño de un diagrama de flujo, depende la mayoría de las veces la creación de un buen código de programa en cualquier lenguaje de programación.

Pseudocódigo o pseudolenguaje se denomina a una serie de instrucciones en lenguaje natural como español, ingles, alemán, etc., (de preferencia su lenguaje natal) y expresiones u operadores, que representan cada uno de los pasos para resolver un programa.

Para evitar confusiones y sobre todo, para acostumbrarse a utilizar las estructuras de control de la programación estructurada, es conveniente no traducir las estructuras IF-THEN-ELSE por SI-ENTONCES-SI NO, FOR por PARA, o DOWHILE por MIENTRAS.

Estructura de datos

Los datos que son procesados por las computadoras se almacenan en celdas de memoria identificadas con una dirección única. Los tipos de datos que reconocen los lenguajes de programación se dividen en numéricos, texto y lógicos.

- **Datos numéricos**. Compuestos exclusivamente de números y se utilizan para realizar cálculos numéricos. Pueden ser **Enteros**, los que representan solo números enteros (sin decimales), positivos o negativos y **Reales**, los positivos o negativos, que contienen decimales o fracciones.
- **Datos tipo texto**. Se componen de letras y números, aunque no se utilizan para realizar cálculos numéricos. Generalmente se incluyen entre comillas en la mayoría de los lenguajes de programación.

- **Datos lógicos**. Datos booleanos que pueden tomar sólo dos valores: Verdadero (True) o Falso (False):

Constantes y variables

En el desarrollo del procesamiento de datos, los programas necesitan utilizar valores constantes y variables para producir los resultados esperados.

- **Constantes**. Son datos o valores específicos, que no cambian durante la ejecución de un programa. Las constantes pueden ser numéricas (**Reales** o **Enteras**) y tipo texto. Por ejemplo, si se va a utilizar el valor de π (PI), 3.1416 en repetidas ocasiones en un programa, se deberá declarar desde el principio.
- **Variables**. Se utilizan para representar datos que pueden cambiar su valor durante la ejecución de un programa. Cada lenguaje de programación requiere que las variables que se van a utilizar en un programa se definan al principio o en alguna sección determinada, de la manera apropiada.

Programa de aplicación

Un programa informático que interactúa con la base de datos emitiendo las apropiadas solicitudes (normalmente una instrucción SQL) dirigidas al SGBD.

Los usuarios interactúan con las bases de datos mediante una serie de programas de aplicación que se utilizan para crear y mantener la base de datos y para generar información. Estos programas pueden ser programas de procesamiento por lotes convencionales o, lo que resulta más habitual hoy en día, aplicaciones en línea. Los programas de aplicación pueden estar escritos en algún lenguaje de programación o en un lenguaje de cuarta generación o de mayor nivel.

La figura 3.8 ilustra la técnica de bases de datos, utilizando los departamentos de ventas y de contratos, éstos utilizan sus programas de aplicación para acceder a la

base de datos a través del SGBD. Cada conjunto de programas de aplicación departamentales gestiona la introducción de datos, el mantenimiento de los mismos y la generación de informes. Sin embargo, si lo comparamos con la técnica basada en archivos, la estructura física y el almacenamiento de los datos ahora son responsabilidad del SGBD.

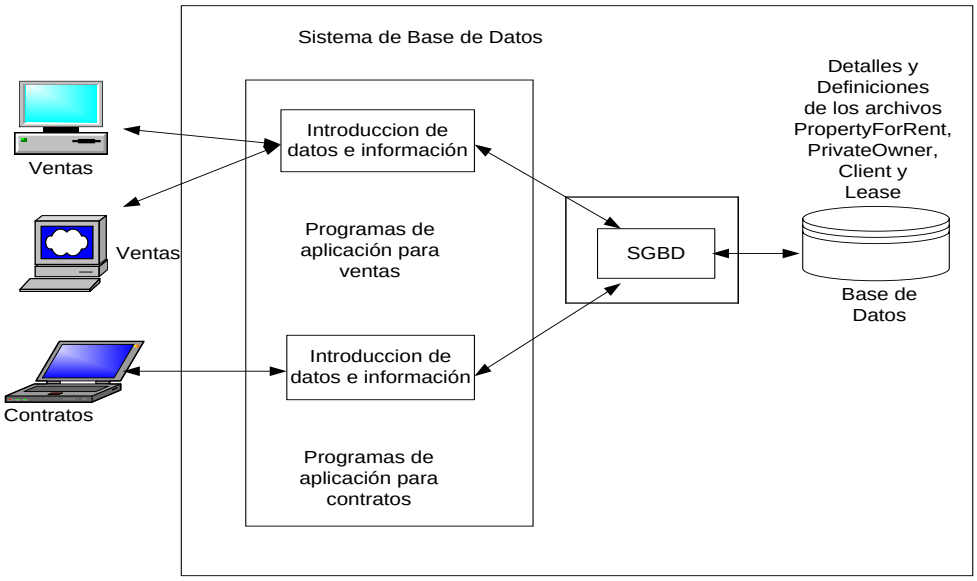


Figura 3.8. Procesamiento con bases de datos

propertyForRent (propertyNo, street, city,postcode, type, rooms, rent, ownerNo)
PrivateOwner (ownerNo,fName, IName, address, telNo)
Client (clienteNO, fName, IName, address, telNo, prefType, maxRent)
Lease (leaseNo, propertyNo, clientNo, paymentMethod, deposit, paid, rentStart. rentFinish)

Vistas

Con esta funcionalidad, el SGBD es una herramienta extremadamente potente y útil. Sin embargo, como a los usuarios finales no les interesa demasiado si una determinada tarea resulta sencilla o compleja para el sistema, podría argumentarse que los SGBD han hecho que las cosas se compliquen, ya que ahora los usuarios

ven más datos de los que requieren o necesitan. Por ejemplo, los detalles que el departamento de contratos quiere ver en lo que respecta a un inmueble en alquiler, como en los archivos tradicionales, uno para ventas y otro para contratos, en la actualidad han cambiado las cosas en la técnica que utiliza una base de datos, como se muestra en la figura 3.8. Ahora las bases de datos también almacenan el tipo de inmueble, el número de habitaciones y los detalles referidos al propietario con imágenes. Para hacer frente a este problema, un SGBD proporciona otra funcionalidad denominada **mecanismo de vista** que permite que cada usuario disponga de su propia vista de la base de datos (una **vista** es, en esencia, un cierto subconjunto de la base de datos). Por ejemplo, podríamos definir una vista que permita que el departamento de contratos solo vea los datos que les interesan referidos a los inmuebles en alquiler.

Además de reducir la complejidad al permitir que los usuarios vean los datos en la forma que desean verlos, las vistas tienen otras diversas ventajas.

- Las vistas proporcionan un cierto nivel de seguridad. Pueden configurarse para excluir aquellos datos que algunos usuarios no deben ver. Por ejemplo, podemos crear una vista que permita que los gerentes de sucursal y el departamento de nómina vean todos los datos referidos al personal, incluyendo los detalles salariales, y una segunda vista que utilizaría el resto del personal y de la que se excluirían esos detalles. salariales.

- Las vistas proporcionan un mecanismo para personalizar la apariencia de la base de datos. Por ejemplo, el departamento de contratos podría denominar al campo de alquiler mensual (rent) utilizando un nombre más conveniente, como por ejemplo Monthly Rent.

- Una vista puede presentar una imagen coherente y estática de la estructura de la base de datos, aún cuando se modifique la base de datos. Esconde

(por ejemplo, se pueden añadir o eliminar campos; se permitiría modificar relaciones o podrían partirse; reestructurarse o renombrarse los archivos). Si se añade o elimina campos de un archivo y estos campos no son requeridos por la vista, ésta no se verá afectada por las modificaciones. Por tanto, las vistas ayudan a conseguir la independencia entre programas y datos de la que hemos hablado.

La explicación anterior es de carácter general y el nivel real de funcionalidad ofrecido por un SGBD difiere entre unos y otros productos. Por ejemplo, un SGBD para una computadora personal puede no permitir el acceso compartido concurrente y proporcionar sólo mecanismos limitados de seguridad, integridad y control de recuperación. Sin embargo, los productos SGBD multiusuario modernos son de gran complejidad ofrecen todas las funciones anteriores y muchas otras. Los sistemas actuales son programas (software) extremadamente complejos compuestos por millones de líneas de código y la documentación está formada por múltiples volúmenes. Esto se debe a la necesidad de proporcionar un programa que realice requisitos de naturaleza muy general. Además, la utilización de un SGBD hoy en día suele requerir un sistema que proporcione una fiabilidad prácticamente total y una disponibilidad 24/7/365 (24 horas al día, 7 días de la semana) incluso en el caso de fallas de hardware o de software. Los SGBD están continuamente evolucionando y expandiéndose para adaptarse a las nuevas necesidades de los usuarios. Por ejemplo, algunas aplicaciones requieren ahora el almacenamiento de imágenes, gráficas, video, sonido y otros tipos de información similar. Para poder satisfacer las necesidades de este mercado, es necesario cambiar los SGBD. Resulta previsible que cada vez se vayan recibiendo nuevas funcionalidades, por lo que las funciones de un SGBD nunca serán estáticas.

Componentes de un entorno SGBD

Podemos identificar cinco componentes principales dentro del entrono SGBD: hardware, software, datos, procedimientos y personas, como se muestra en la Figura 3.9.

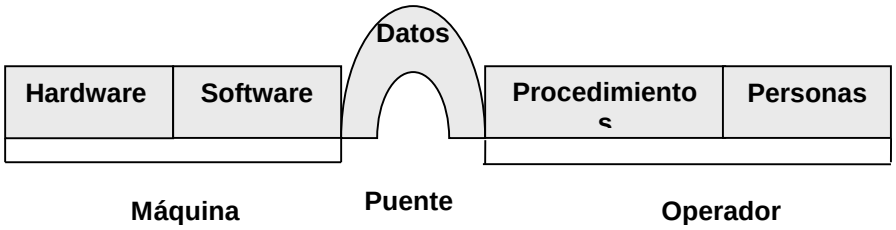


Figura 3.9. Entorno SGBD

Hardware

El SGBD y las aplicaciones requieren una plataforma (hardware) sobre la cual se ejecutará. El hardware puede ir desde una única computadora personal hasta un único *mainframe* o una red de computadoras. El hardware dependerá de las necesidades de la organización y del SGBD utilizado. Algunos SGBD sólo se ejecutan sobre una plataforma o sobre un sistema operativo particular, mientras que otros se ejecutan sobre un rango más amplio de plataformas y sistemas operativos. Todo SGBD requiere una cantidad mínima de memoria principal de espacio en disco para poder ejecutarse, pero esta configuración mínima puede no necesariamente proporcionar un rendimiento aceptable. En la Figura 3.10:

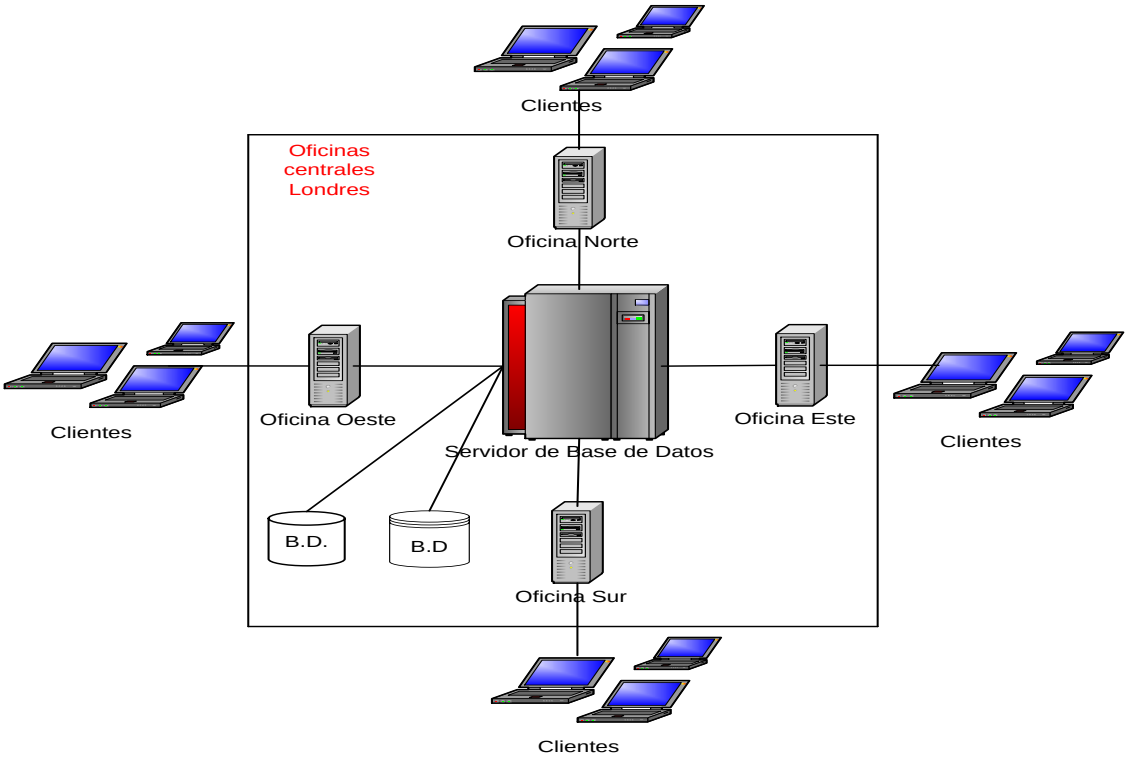


Figura 3.10. Configuración hardware para una Empresa Grande

Se ilustra una configuración (hardware) simplificada para DreamHome. Está compuesta por una red de minicomputadoras. Una computadora central ubicada en Londres y en la que se ejecuta el sistema de servicio (**backend**) del SGBD, es decir, la parte del SGBD que gestiona y controla el acceso a la base de datos. También se muestran varias computadoras en distintas ubicaciones en las que se ejecuta el sistema de interfaz (**frontend**) del SGBD, es decir, la parte del SGBD que implementa la interfaz con el usuario. Este tipo de arquitectura se denomina **cliente-servidor**: el backend es el servidor y el frontend es el cliente.

Software

El software comprende el propio software del SGDB y los programas de aplicación, junto con el sistema operativo, que incluye el software de red si el SGBD se está utilizando en una red. Normalmente los programas de aplicación se escriben en un lenguaje de aplicación de tercera generación (3GL), como C, C++, Java, Visual

Basic, COBOL, Fortran, Ada o Pascal, utilizando un lenguaje de cuarta generación (4GL) como SQL, incrustado dentro de un lenguaje de tercera generación. El SGBD puede disponer de sus propias herramientas de cuarta generación que permitan los desarrollos rápidos de aplicaciones gracias a la existencia de lenguajes de consulta no procedimentales, generadores de informes, formularios, gráficos y aplicaciones. La utilización de herramientas de cuarta generación puede mejorar de forma significativa la productividad y dar como resultado programas que son más fáciles de mantener.

Datos

Quizá el componente más importante de un entorno SGBD, al menos desde el punto de vista de los usuarios finales sean los datos. En la Figura 3.9 podemos observar que los datos actúan como una especie de puente entre los componentes ligados a la máquina y al operador humano. La base de datos contiene tanto los datos operacionales como los metadatos, es decir, los 'datos acerca de los datos'. La estructura de la base de datos se denomina **esquema**.

En la Figura 3.8 el esquema está compuesto por cuatro archivos o **tablas**, que son: PropertyForRent, PrivateOwner, Client y Lease. La tabla PropertyForRent tiene ocho campos o **atributos**, a saber: propertyNo, street, city, postcode, type (el tipo de inmueble), *rooms* (el número de habitaciones), *rent* (el alquiler mensual) y ownerNo. El atributo ownerNo modela la relación entre PropertyFor Rent y PrivateOwner: es decir, el propietario posee (*owns*) un inmueble para el alquiler.

Procedimientos

Los procedimientos son las instrucciones y reglas que gobiernan el diseño y utilización de la base de datos.

Los usuarios del sistema y el personal que gestiona la base de datos requieren una serie de procedimientos documentados que les permitan saber cómo utilizar o

ejecutar el sistema. Estos procedimientos pueden estar compuestos de instrucciones que les digan cómo:

- Iniciar una sesión con el SGBD;
- Utilizar una funcionalidad concreta del SGBD o un programa de aplicación;
- Iniciar y detener el SGBD;
- Realizar respaldos de seguridad de la base de datos en varias copias,
- Gestionar las fallas de hardware o software. Esto puede incluir procedimientos para identificar el componente fallido, para reparar este componente (por ejemplo, telefonear al ingeniero de servicio (hardware) apropiado), después de reparar la falla, recuperar la base de datos;
- Cambiar la estructura de una tabla, reorganizar la base de datos entre múltiples discos, mejorar el rendimiento o respaldar y guardar los datos en un almacenamiento secundario.

Personas

El componente más importante y fundamental es el recurso humano que se relaciona con el sistema. Podemos identificar cuatro tipos distintos de personas que pueden participar en un entorno SGBD; administradores de datos y de la base de datos, diseñadores de bases de datos, desarrolladores de aplicaciones y usuarios finales.

Administradores de datos y de la base de datos

La base de datos y el SGBD son recursos corporativos que deben gestionarse igual que cualquier otro recurso. La administración de datos y de la base de datos son papeles que generalmente se asocian con la gestión y control de un SGBD y de los datos en él almacenados. El **administrador de datos** (DA, Data Administrator) es responsable de gestionar los recursos de datos, lo que incluye la planificación de la base de datos, el desarrollo y mantenimiento de estándares, políticas y procedimientos y el diseño procedimental/lógico de la base de datos.

El **administrador de la base de datos** (DBA, *Database Administrator*) es responsable de la materialización física de la base de datos, incluyendo la implementación y diseño físico de la base de datos, el control de la seguridad y de la integridad, el mantenimiento de la fiabilidad del sistema y la garantía de que las aplicaciones exhiban un rendimiento satisfactorio para los usuarios. El papel de un DBA tiene una orientación más técnica que el de DA, requiriéndose un conocimiento detallado del SGBD de destino y del entorno de sistema en el que está implementado. En algunas organizaciones no hay distinciones entre estos dos papeles, mientras que en otras la importancia de los recursos corporativos se ve reflejada en la asignación de equipos de personas a cada uno de estos dos papeles.

Diseñadores de bases de datos

En los grandes proyectos de diseño de bases de datos, podemos distinguir entre dos tipos de diseñadores: los diseñadores lógicos de la base de datos y los diseñadores físicos de la base de datos. Las responsabilidades del **diseñador lógico de la base de datos** son identificar los datos (es decir, las entidades y atributos), las relaciones entre los datos y las restricciones que hay que aplicar a los datos que se almacenen en la base de datos. El diseñador lógico de la base de datos debe tener una comprensión profunda y completa de los datos de la organización y de las restricciones aplicables (las restricciones se denominan en ocasiones **reglas de negocio**).

Para poder tener éxito, el diseñador lógico de la base de datos debe implicar a todos los potenciales usuarios de la base de datos en el desarrollo del modelo de datos, y esta implicación debe producirse lo más pronto posible dentro del proceso.

- Diseño conceptual de la base de datos, que es independiente de los detalles de implementación, como el SGBD de destino, los programas de aplicación, los lenguajes de programación o cualquier otra consideración física.

- Diseño lógico de la base de datos, dirigido a un modelo de datos específico, como por ejemplo el modelo relacional, el modelo en red, el modelo jerárquico o el modelo orientado a objetos.

El **diseñador físico de la base de datos** decide cómo materializar físicamente el diseño lógico de la base de datos. Esto implica:

- Establecer la correspondencia entre el diseño lógico de la base de datos y un conjunto de tablas y restricciones de integridad.
- Seleccionar estructuras de almacenamiento y métodos de acceso específicos para los datos con el fin de conseguir unas buenas prestaciones.
- Diseñar las medidas de seguridad que los datos requieran.

El diseñador físico de la base de datos debe conocer a la perfección la funcionalidad del SGBD de destino y puede entender las ventajas y desventajas de cada alternativa para cada implementación concreta, además debe ser capaz de seleccionar una estrategia de almacenamiento adecuada que tenga en cuenta el uso de la base de datos. Mientras que el diseño conceptual y lógico de la base de datos estén relacionados con el diseño físico de la base de datos. Se requieren capacidades y conocimientos diferentes, lo que implica en muchas ocasiones utilizar personas distintas.

Desarrolladores de aplicaciones

Una vez implementada la base de datos, es necesario efectuar también los programas de aplicación que proporcionen la funcionalidad requerida por los usuarios finales. Esto es responsabilidad de los **desarrolladores de aplicaciones**. Normalmente, éstos trabajan a partir de una especificación producida por los líderes del proyecto, analistas de sistemas. Cada programa contiene enunciados que exigen al SGBD realizar algún tipo de operación sobre la base de datos. Esto incluye extraer datos, insertarlos, actualizarlos o borrarlos. Los programas pueden estar escritos en un lenguaje de programación de tercera o cuarta generación.

Usuarios finales

Los usuarios finales son los “clientes” de la base de datos, que se diseña, implementa y mantiene precisamente para dar servicio a sus necesidades de información. Los usuarios finales pueden clasificarse de acuerdo con la forma en que utilizan el sistema.

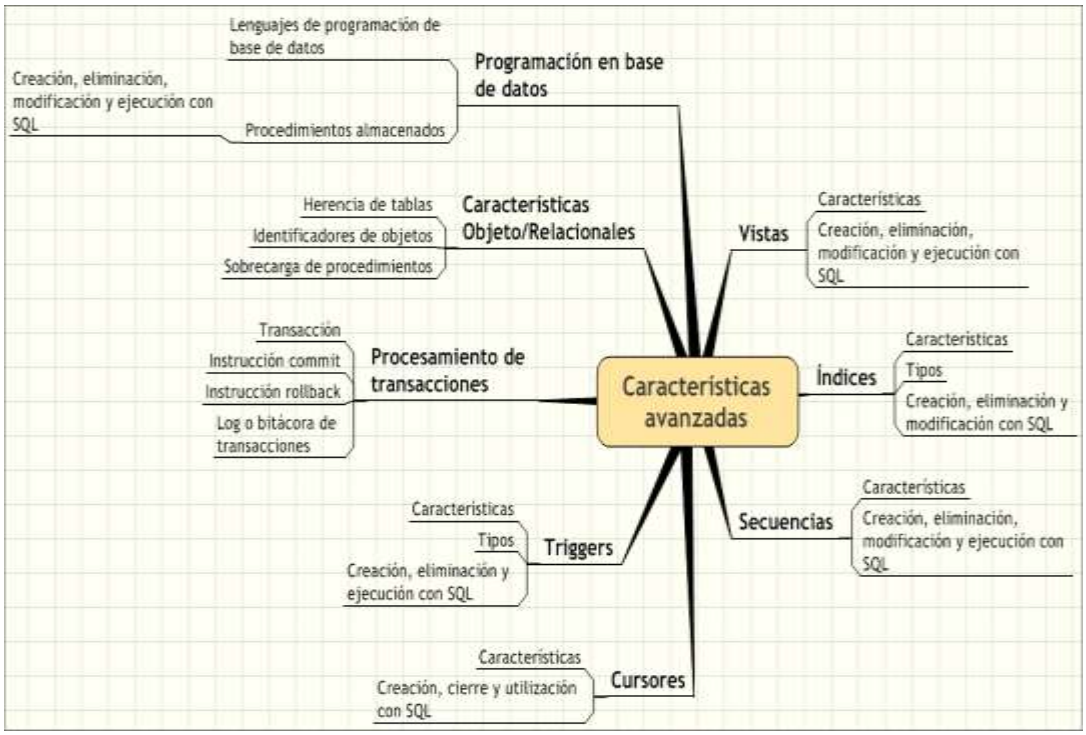
Usuarios inexpertos.

Normalmente no son conscientes de la existencia de un SGBD. Acceden a la base de datos mediante programas de aplicación escritos a propósito y que intentan que las operaciones sean lo más simples. Estos usuarios invocan las operaciones sobre la base de datos introduciendo comandos simples o seleccionando una serie de opciones en un menú. Esto quiere decir que no necesitan conocer ningún detalle ni sobre la base de datos, ni sobre el SGBD. Por ejemplo, el cajero de un supermercado utiliza un lector de código de barras para averiguar el precio de un artículo. Sin embargo, se está ejecutando un programa de aplicación que lee el código de barras, consulta el número de artículo, su precio en la base de datos de la tienda o almacén y muestra el precio en la pantalla.

Usuarios avanzados.

Están familiarizados con la estructura de la base de datos y con las funcionalidades ofrecidas por el SGBD. Estos usuarios pueden utilizar un lenguaje de consulta de alto nivel, como SQL, para llevar a cabo las operaciones requeridas. Este tipo de usuarios pueden incluso escribir sus propios programas de aplicación para su uso personal.

RESUMEN


Características avanzadas
Programación en base de datos
Lenguajes de programación de base de datos
Creación, eliminación, modificación y ejecución con SQL
Procedimientos almacenados
Características Objeto/Relacionales
Herencia de tablas
Identificadores de objetos
Sobrecarga de procedimientos
Procesamiento de transacciones
Transacción
Instrucción commit
Instrucción rollback
Log o bitácora de transacciones
Triggers
Características
Tipos
Creación, eliminación y ejecución con SQL
Cursores
Características
Creación, cierre y utilización con SQL
Vistas
Características
Creación, eliminación, modificación y ejecución con SQL
Índices
Características
Tipos
Creación, eliminación y modificación con SQL
Secuencias
Características
Creación, eliminación, modificación y ejecución con SQL

BIBLIOGRAFÍA

SUGERIDA

Abbey, Michael, y Michael J. Corey, *Oracle guia para el principiante*, México, McGraw Hill, 1996.

----------, Oracle, México, McGraw Hill, 1995.

Charte Ojeda, Francisco, *SQL*, Madrid, Anaya, 2005.

Connolly, Thomas M., y Carolyn E. Begg, *Sistemas de bases de datos*, 4ª ed., México, Addison Wesley, 2005

Dinerstein, Nelson T., *Sistemas de manejo de archivos y bases de datos para microcomputadoras*, México, CECSA, 1989.

Houlette, Forrest, *Fundamentos de SQL, México,* McGraw Hill, 2003.

Koch, George, *Oracle 7 manual de referencia*, México, McGraw Hill, 1995.

Mendelzon / Ale *Introducción a las bases de datos relacionales*, México, Pearson Educación, 2000.

Oracle Education, *Oracle7 RDBMS Backup and Recovery Strategias*, Mexico, 1996

Piattini, Mario G., y Esperanza Marcos, *Tecnología y diseño de bases de datos*, Madrid, Alfaomega, 2007.

http://www.postgresql.com

Unidad 4

Consultas

OBJETIVO PARTICULAR

Al finalizar la unidad el alumno será capaz de:
Identificar los componentes básicos y capacidades operativas del lenguaje SQL y demostrará los conocimientos relativos de alcance y uso de este lenguaje.

TEMARIO DETALLADO

(12-horas)

4. Consultas

- 4.1. Cross Join
- 4.2. Inner Join
- 4.3. Outer Join
  - 4.3.1. Left Outer Join
  - 4.3.2. Right Outer Join
  - 4.3.3. Full Outer Join
- 4.4. Subconsultas (Self Join)
- 4.5. Operadores relacionales
- 4.6. Agrupamiento
- 4.7. Rangos de salida

INTRODUCCIÓN

La mayor parte del tiempo, SQL se utiliza para efectuar consultas sobre una base de datos, extrayendo la información que en ese momento interesa a una determinada persona o aplicación. La sentencia de SQL por excelencia para estas tareas es el SELECT y sus posibilidades son casi interminables.

4.1. Cross join

Es posible efectuar combinaciones cruzadas entre tablas, auto-combinaciones y combinaciones naturales. Una combinación cruzada de tablas, o CROSS JOIN, consiste en cruzar cada una de las filas de la primera tabla con cada una de las filas de la segunda tabla, generando lo que se conoce como un producto cartesiano. En la figura 4.1 puede verse el resultado obtenido al ejecutar la consulta siguiente:

```
SELECT apellidos, titulo
FROM socios
CROSS JOIN libros
```

```
arojas=> select apellido_paterno, apellido_materno,título from socios cross
      join libros;
```

apellido_paterno	apellido_materno	título
Arias	Trera	El misterio del perro secuestrado
Moreno	Pardo	El misterio del perro secuestrado
Charte	Luque	El misterio del perro secuestrado
Charte	Ojeda	El misterio del perro secuestrado
Charte	Luque	El misterio del perro secuestrado
Lopez	Aguilera	El misterio del perro secuestrado
Lopez	Hilera	El misterio del perro secuestrado
Garcia	Aguilera	El misterio del perro secuestrado
Garcia	Hilera	El misterio del perro secuestrado

Perez	Aguilera	El misterio del perro secuestrado
Perez	Leon	El misterio del perro secuestrado
Cid	Luque	El misterio del perro secuestrado
Arias	Trera	El secreto de los piratas
Moreno	Pardo	El secreto de los piratas
Charte	Luque	El secreto de los piratas
Charte	Ojeda	El secreto de los piratas
Charte	Luque	El secreto de los piratas
Lopez	Aguilera	El secreto de los piratas
Lopez	Hilera	El secreto de los piratas
Garcia	Aguilera	El secreto de los piratas
Garcia	Hilera	El secreto de los piratas
Perez	Aguilera	El secreto de los piratas
Perez	Leon	El secreto de los piratas
Cid	Luque	El secreto de los piratas

Figura 4.1. Combinación cruzada de dos tablas.

El total de renglones impresos son (180 filas) para efectos de ejemplificar solo se imprimió una muestra.

4.2. Inner join

El tipo más habitual de combinación es aquél en el que interesa obtener las filas de dos o más tablas que tienen un contenido equivalente en una determinada columna, también se conoce como uniones internas al expresarse en SQL como INNER JOIN.

Supón que quieres obtener el NIF, apellido paterno y fecha de préstamo de todos aquellos socios que tienen un libro en préstamo, es decir, una tabla compuesta de

filas formadas por columnas: nif, apellido paterno y préstamo, siendo cada fila la representación de un socio.

Este tipo de consulta se expresaría de la siguiente manera:

```
SELECT socios.nif, apellido_paterno, prestamo
FROM socios
INNER JOIN prestamos
ON socios.nif=prestamos.nif;
```

```
arojas=> select socios.nif,apellido_paterno,fecha_prestamo from socios
arojas-> inner join préstamos on socios.nif=préstamos.nif;
```

nif	apellido_paterno	fecha_prestamo
23727319S	Arias	2005-07-03
74381725T	Charte	2005-07-05
62877137F	Charte	2005-07-02
23727319S	Arias	2007-06-22

(4 filas)

La segunda tabla[4], sin embargo, se facilita después de INNER JOIN. Esta cláusula indica que solo han de aparecer las filas de la tabla socios que tienen una correspondencia con la tabla préstamos, entendiendo por correspondencia la condición que se comunica en el apartado ON: la coincidencia entre la columna nif de ambas tablas.

[4] Al hablar de tablas se hace referencia a los contenidos de los datos incluidos en éstas, en la base de datos.

Las combinaciones naturales son similares a las INNER JOIN, si bien no es necesario indicar de forma explícita el vínculo existente entre las tablas sino que éste se deduce a partir de los nombres de las columnas. La sintaxis estándar es la siguiente:

```
SELECT columnas FROM tabla1 NATURAL JOIN tabla2;
```

Hay que tener en cuenta que este tipo de combinación compara todas las columnas de nombre común que existan entre las dos tablas, no únicamente la columna que se puede considerar clave primaria en una tabla y externa en la otra.[5]

Las combinaciones naturales pueden anidarse, como todas las demás, y permiten simplificar consultas como las usadas anteriormente al eliminar el apartado ON asociado a cada JOIN. El ejemplo siguiente obtiene datos de las tres tablas, vinculando socios con préstamos y ésta con libros sin necesidad de identificar explícitamente las columnas por usar:

```
SELECT apellido_paterno,apellido_materno,fecha_prestamo
FROM socios
NATURAL JOIN prestamos
    NATURAL JOIN LIBROS;
```

Por default NATURAL JOIN se interpreta como NATURAL INNER JOIN, de tal manera que únicamente se incluyen en el resultado las filas que tienen una correspondencia. Es posible, usar las formas NATURAL LEFT JOIN, NATURAL RIGHT JOIN y NATURAL FULL JOIN, equivalentes a LEFT OUTER JOIN, RIGHT OUTER JOIN y FULL OUTER JOIN pero con la salvedad de que no es preciso el apartado ON.

[5] Las uniones de tipo NATURAL JOIN son una extensión al estándar SQL y los RDBMS las contemplan de manera desigual.

```
arojas=> select apellido_paterno,apellido_materno,fecha_prestamo
from socios
natural join prestamos
natural join libros;
```

apellido_paterno	apellido_materno	fecha_prestamo
Arias	Trera	2005-07-03
Charte	Ojeda	2005-07-05
Charte	Luque	2005-07-02
Arias	Trera	2007-06-22

(4 filas)

4.3. Outer join

A veces puede interesarnos la inclusión de todas las filas de la primera tabla y sólo las concordantes de la segunda, solamente las concordantes de la primera y todas las de la segunda o incluso todas las filas de ambas tablas pero distribuidas de acuerdo con la relación entre columnas que se haya indicado.

En estos casos se utilizan las combinaciones no exclusivas o combinaciones externas, OUTER JOIN, en contraposición a las internas: INNER JOIN. La sintaxis general de este tipo de combinaciones es la siguiente:

```
SELECT columnas
FROM tabla1
LEFT|RIGHT|FULL [OUTER] JOIN tabla2
ON relación
```

En una combinación de este tipo tabla1 se considera la tabla que queda a la izquierda de la combinación, mientras que la tabla2 sería la tabla de la derecha. Esto es importante porque dependiendo de que deseemos incluir todas las filas de la primera o de la segunda, tabla1 o tabla2, tendremos que preceder OUTER con la palabra LEFT o RIGHT, respectivamente.

Las columnas de la segunda tabla para las que no exista una correspondencia en la primera aparecerán como NULL o sencillamente, vacías, dependiendo del RDBMS.

Comencemos con un caso sencillo, queremos obtener los apellidos de los socios junto con la fecha de préstamo que tengan asociada, pero sin excluir aquellos socios que no tengan un libro en préstamo. Haciendo un INNER JOIN obtendríamos únicamente las filas de socios con libros prestados, pero con la consulta siguiente el resultado es el esperado. Ver consulta

```
arojas=> select apellido_paterno,apellido_materno,fecha_prestamo from
socios
arojas-> inner join prestamos
arojas-> on socios.nif = prestamos.nif;

apellido_paterno     apellido_materno    fecha_prestamo

 Arias          Trera              2005-07-03
 Charte              Ojeda              2005-07-05
 Charte              Luque              2005-07-02
 Arias          Trera              2007-06-22
(4 filas)
```

4.3.1. Left Outer Join y 4.3.2 Right Outer Join

Si cambiamos LEFT por RIGHT obtendríamos todas las filas de la tabla préstamos y únicamente las filas de socios que tuviesen una correspondencia, lo cual equivale, en este caso.

Las consultas de tipo OUTER JOIN también se pueden anidar, combinando las filas resultantes de una primera combinación con las de una tercera consulta. Observe el siguiente ejemplo:

```
arojas=> select apellido_paterno,apellido_materno,titulo,fecha_prestamo
from socios
arojas-> left outer join préstamos
arojas-> right outer join libros
arojas-> on libros.codigo = pretamos.codigo
arojas-> on socios.nif = prestamos.nif;
```

apellido_paterno	apellido_materno	titulo	fecha_prestamo
Arias	Trera	Leyendas de las calles de la ciudad de México	2005-07-03
Charte	Ojeda	Ventanas de Nueva York	2005-07-05
Charte	Luque	Mi amigo agapito	2005-07-02
Arias	Trera	El capitan calzoncillos	2007-06-22

(4 filas)

Esta consulta extrae de la tabla libros el título de todos los libros y efectúa un RIGHT OUTER JOIN con la tabla préstamos, lo cual significa que el resultado incluirá los títulos de todos los libros y la fecha de préstamo de aquellos que han sido prestados.

Este conjunto de filas se combina con el resultado de la primera parte de la consulta, las filas que incluyen los apellidos de todos los socios.

Al ser esta una consulta LEFT OUTER JOIN, se conservan todas las filas de la tabla socios, que es la de la izquierda, desechando las que no son coincidentes en la tabla de la derecha, que en este caso contenía el título de todos los libros y la fecha de préstamo.

Si cambiamos el LEFT JOIN por un RIGHT JOIN, incluiríamos entonces en los resultados todas las filas correspondientes a los libros y únicamente las de los socios que tienen un libro en préstamo, tal y como se muestra a continuación:

```
arojas=> select apellido_paterno,título,fecha_prestamo from socios
arojas-> right outer join prestamos
arojas-> right outer join libros
arojas-> on libros.codigo = prestamos.codigo
arojas-> on socios.nif = prestamos.nif;
```

apellido_paterno	titulo	fecha_prestamo
Arias	El capitan calzoncillos	2007-06-22
	El misterio del perro secuestrado	
	El secreto de los pirats	
Charte	Mi amigo agapito	2005-07-02
	Un cesto lleno de lapices	
	Fabulas mitologicas	
Arias	Leyendas de las calles de la ciudad de Mexico	2005-07-03
	Platero y yo	
	En busca del unicornio	
Charte	Ventanas de Nueva York	2005-07-05
	Cuentos libertinos	
	La incognita Newton	
	Excel 2003	
	Curso de Linux	
	Curso de Rdbms	

(15 filas)

4.3.3. Full Outer Join

También existe la posibilidad de recuperar tanto todas las filas de socios como todas las filas de libros, relacionadas a través de la fecha de préstamo, sencillamente sustituyendo el primer RIGHT OUTER JOIN por un FULL OUTER JOIN. De esta manera la primera subconsulta, que tiene las filas de socios, se combina con la segunda, consecuencia de las tablas libros y préstamos, sin eliminar fila alguna. El resultado sería similar al del último ejemplo pero añadiendo una fila más con el nombre del socio que no tiene ningún libro en préstamo.

```
arojas=> select apellido_paterno,titulo,fecha_prestamo from socios
arojas-> full outer join prestamos
arojas-> right outer join libros
arojas-> on libros.codigo = prestamos.codigo
arojas-> on socios.nif = prestamos.nif;
```

apellido_paterno	titulo	fecha_prestamo
Arias	El capitan calzoncillos	2007-06-22
Arias	Leyendas de las calles de la ciudad de Mexico	2005-07-03
Lopez		
Lopez		
Garcia		
Garcia		
Perez		
Perez		
Moreno		
Charte	Mi amigo agapito	2005-07-02
Charte		
Cid		
Charte	Ventanas de Nueva York	2005-07-05
	El misterio del perro secuestrado	
	El secreto de los piratas	
	Un cesto lleno de lapices	
	Fabulas mitologicas	
	Platero y yo	
	En busca del unicornio	
	Cuentos libertinos	
	La incognita Newton	
	Excel 2003	
	Curso de Linux	
	Curso de Rdbms	

(24 filas)

4.4. Subconsultas (self join)

Hay casos en los que los valores que han de utilizarse como referencia en una expresión condicional, para obtener las filas que se ajustan a un criterio, no pueden facilitarse directamente como valores constantes, sino que se encuentran en otra tabla de la base de datos. En este caso lo que se hace es incluir como parte del predicado una subconsulta.

Supón que quieres obtener la signatura y título de los libros que están prestados, pero no consultados la columna disponible proporciona esta información porque un libro podría no estar disponible por otras causas y en realidad, no estar prestado a un socio. Lo que necesita es obtener las filas cuyo código de libro coincida con el código registrado en la tabla préstamos, haciendo algo así:

```
SELECT signatura, tiulo
FROM libros
WHERE codigo IN (codigo1, codigo2…)
```

El problema es que no puedes escribir directamente la lista de códigos tras IN, porque estos cambian a medida que los socios se llevan y devuelven los libros. Sería necesario incluir en los paréntesis todos los códigos que haya en la tabla préstamos, con una lista similar a la que genera la consulta SELECT código FROM préstamos. Pues dicho y hecho, no hay más que incluir esa consulta entre paréntesis:

arojas=> select signatura,título from libros where codigo in (select codigo from préstamos);

signatura	titulo
I DIE, mia	Mi amigo agapito
T CHA, php	Leyendas de las calles de la ciudad de mexico
G MUN, ven	Ventanas de Nueva York
I PIL cap	El capitan calzoncillos

(4 filas)

También pueden introducirse subconsultas en el predicado de una cláusula WHERE mediante los elementos ALL, ANY, SOME y EXISTS.

Adición de columnas calculadas

Además de columnas existentes en las tablas referenciadas en la cláusula FROM, la sentencia SELECT también permite la creación de columnas, normalmente calculadas a partir de las ya existentes y su inclusión en el conjunto de resultados. Esas columnas, además, pueden también ser utilizadas en la cláusula WHERE como parte de una expresión condicional.

En el caso más simple de columna calculada se da cuando sencillamente queremos introducir un valor constante como columna de resultado. Por ejemplo, en la consulta siguiente se obtiene el título de todos los libros que están disponibles y se añade una columna con el texto ‘Disponible’

```
arojas=> select titulo, disponible as disponible from libros where disponible = 'S';

          título                      disponible

 El misterio del perro secuestrado    S
 El secreto de los pirats             S
 Un cesto lleno de lapices            S
 Fabulas mitologicas                  S
 Platero y yo                         S
 En busca del unicornio               S
 La incognita Newton                  S
 Excel 2003                           S
 Curso de Linux                       S
 Curso de Rdbms                       S
 El capitan calzoncillos              S
(11 filas)
```

Igual que una secuencia de caracteres, también puede introducirse un número o una fecha, así como el resultado de funciones y ciertas columnas especiales con las que cuentan los RDBMS. Para insertar en los resultados la fecha actual, por ejemplo, usaríamos la columna CURRENT_DATE como se ve a continuación:

```
arojas=> select nif,fecha_prestamo,current_date from prestamos;
```

nif	fecha_prestamo	Date
23727319S	2005-07-03	2007-07-10
74381725T	2005-07-05	2007-07-10
62877137F	2005-07-02	2007-07-10
23727319S	2007-06-22	2007-07-10
(4 filas)		

Cálculos numéricos

El contenido de una columna puede ser el resultado de una operación aritmética, con independencia que en ella intervengan o no otras columnas de la tabla. Los operadores a utilizar son los habituales: +, -, *, y /, para efectuar una suma, una resta, una multiplicación o una división.

Mediante estos operadores y asumiendo que tuviésemos tablas con datos numéricos, podríamos calcular un porcentaje del precio de un producto, por ejemplo un impuesto o descuento; sumar datos de varias columnas para hallar un total, etc. En nuestras tablas el único dato numérico que tenemos es el código de los libros, sobre el que podemos operar como se aprecia en el siguiente ejemplo:

arojas=> select codigo,codigo + 100 as codigomas100, codigo/2 as codigoentre2 from libros;

codigo	codigo + 100	codigo / 2
2	102	1
3	103	1
4	104	2
5	105	2
6	106	3
7	107	3
8	108	4
9	109	4
10	110	5
11	111	5
12	112	6
13	113	6
14	114	7
15	115	7
1	101	0

(15 filas)

En la expresión aritmética pueden utilizarse múltiples operadores y los niveles de paréntesis que sean necesarios para alcanzar el resultado deseado.

1.5. Operadores relacionales

Existen distintos tipos de condiciones de búsqueda, si bien el más simple, y seguramente de uso más frecuente, es aquel en el que se establece una

determinada relación entre una columna de datos y un valor concreto, algo que se expresaría de la siguiente manera:

WHERE columna operador valor

El operador será uno de los enumerados en la tabla 4.1.permitiendo buscar coincidencias entre el contenido de la columna y el valor, todos aquellos casos en el que el contenido de la columna puede considerarse menor que el valor de referencia, etc.

Operador	Relación
=	Igualdad entre contenido de la columna y valor.
<>	Desigualdad entre contenido de la columna y valor
<=	Contenido de la columna menor o igual al valor
<	Contenido de la columna menor que el valor
>=	Contenido de la columna mayor o igual al valor
>	Contenido de la columna mayor que el valor

Tabla 4.1. Operadores de relación de SQL

Para comprender cómo actúan los operadores <=, <, >= y > es necesario tener en cuenta que todos los tipos de datos: números, secuencias de caracteres, fechas, etc., son susceptibles de ser ordenados de mayor a menor conforme a un determinado criterio que, en ocasiones depende de la configuración del RDBMS. Una condición del tipo WHERE columna < valor sería TRUE, incluyendo la fila en el conjunto de resultados, si en dicho orden el contenido de la columna queda delante del valor de referencia.

A la hora de utilizar una condición es necesario tener en cuenta que la columna cuyo contenido va a evaluarse y el valor de referencia deben de ser, necesariamente, del mismo tipo. Es decir, no podemos comparar una columna que contiene un número

con una secuencia de caracteres alfabéticos sin obtener un error o bien, en su defecto, un conjunto de datos vació, dependiendo de lo estricto que sea el RDBMS. Esto es lo que nos dice PostgreSQL , por poner un ejemplo, al intentar comparar la columna código de libros con la secuencia de caracteres SQL:

```
arojas=> select codigo,título from libros where codigo < 'SQL';
ERROR:  la sintaxis de entrada no es vÃ¡lida para integer: Â«SQLÂ»
```

Suponiendo que quisiéramos obtener una lista de los libros

```
arojas=> select codigo, título from libros where disponible = 'S';

 codigo          titulo

 2          El misterio del perro secuestrado
 3          El secreto de los pirats
 5          Un cesto lleno de lapices
 6          Fabulas mitologicas
 8          Platero y yo
 9          En busca del unicornio
 12         La incognita Newton
 13         Excel 2003
 14         Curso de Linux
 15         Curso de Rdbms
 1          El capitan calzoncillos
(11 filas)
```

Operadores lógicos

En ocasiones puede ser interesante usar más de una condición a la hora de filtrar las filas de una tabla, por ejemplo para comprobar que se dan ciertas condiciones en dos o más columnas. Supón que quieres saber qué títulos están disponibles de un autor determinado, en vez de obtener la lista de todos los títulos disponibles. En este caso tendrías que crear dos expresiones de relación:

WHERE disponible = ‘S’ operador _ logico autor = ‘X’

Las dos expresiones no pueden disponerse, sin más, una detrás de la otra, sino que es necesario unirlas con uno de los operadores lógicos con que cuenta SQL. Según el operador elegido, el resultado final será TRUE si ambas expresiones son ciertas, en el caso de AND, o si lo es al menos una de ellas, al utilizar OR.

Nos interesa obtener los libros de un autor que, además, estén disponibles, usaríamos el operador lógico AND en una expresión como la siguiente:

arojas=> select codigo, titulo from libros where disponible = 'S' and autor = 'Charte, Francisco';

```
 codigo          título

                  Fabulas
 6               mitologicas
 13               Excel 2003
 (2 filas)
```

Si cambiamos AND por OR, la lista de resultados incluiría todos aquellos libros que estén disponibles, sean del autor que sean y todos los del autor indicado, estén o no disponibles.

Es decir, se incluyen todas las filas en las que se cumpla al menos una de las dos condiciones.

El tercer operador lógico de que dispone SQL es NOT, sirviendo no para unir dos expresiones sino para invertir el sentido de la expresión que se facilita a continuación.

Por ejemplo, con la sentencia siguiente obtenemos la lista de los libros disponibles que no son de un cierto autor cuyo nombre indicamos:

```
arojas=> select codigo, titulo from libros where disponible = 'S' and not autor = 'Charte, Francisco';

 codigo                 título

                 El misterio del perro
 2               secuestrado
 3                El secreto de los pirats
 5                Un cesto lleno de lapices
 8                Platero y yo
 9                En busca del unicornio
 12               La incognita Newton
 14               Curso de Linux
 15               Curso de Rdbms
 1                El capitan calzoncillos
 (9 filas)
```

Agrupar expresiones con paréntesis

Al usar los operadores lógicos para componer una condición formada por varias expresiones, puede darse el caso de que el orden de evaluación por default no genere los resultados esperados en un principio. Supón que quieres obtener la lista de todos los libros que están disponibles y cuyo autor es uno de dos posibles.

En un principio podría intentar hacer algo así:

WHERE disponible = ‘S’ AND
autor = ‘autor1’ OR autor = ‘autor2’

El resultado, sin embargo, sería una lista con los libros de autor1 que se encuentren disponibles y todos los libros de autor2, estén o no disponibles. Esto es así porque el operador AND une las dos primeras expresiones, cuando lo que interesa en este caso concreto es unir la segunda, tercera y evaluar ese resultado con la primera. El orden de evaluación puede modificarse fácilmente mediante el uso de paréntesis, como haríamos en cualquier expresión matemática:

WHERE disponible = ‘S’ AND
(autor = ‘autor1’ OR autor = autor2’)

Con esta condición sí generaríamos la tabla de resultados requerida, con los libros que están disponibles y pertenecen a uno u otro autor. De ser necesario, podrían utilizarse varios niveles de paréntesis.

4.6. Agrupamiento

La cláusula GROUP BY para agrupar una serie de filas en subconjuntos, según el contenido de una o más de sus columnas o bien de una expresión obtenida a partir de ellas, emplearemos la cláusula GROUP BY.

Esta aparecerá tras SELECT, FROM y WHERE, es decir, partimos de una consulta que extraería una serie de filas y después usamos GROUP BY para dividirlas en grupos.

Tras GROUP BY, al igual que después de ORDER BY, dispondremos del nombre de la columna o columnas por las que se agrupara, o bien la expresión o cálculo que servirá como base de la agrupación. A diferencia de ORDER BY, sin embargo, GROUP BY no admite que se utilicen los alias de columnas ni tampoco los indicadores de posición relativa.

Una de las reglas fundamentales a la hora de usar GROUP BY es el hecho de que tras SELECT no pueden aparecer más columnas individuales que aquellas que aparezcan también en el criterio de agrupación, con la única excepción de las funciones de agregado. Es decir, no podemos hacer lo siguiente:

```
SELECT titulo, autor
FROM libros
GROUP BY autor
```

Si agrupamos las filas según el contenido de la columna autor, quedando una única fila por cada autor distinto, dicha fila no puede contener también el título, puesto que

podrían existir varios asociados a un mismo autor y no es posible determinar cuál de ellos habría de tomarse.

Desde este punto de vista, una sentencia como esta:

```
SELECT autor
FROM libros
GROUP BY autor
```

Sería equivalente al utilizar:

```
SELECT DISTINCT autor
FROM libros
```

Obteniendo únicamente las filas en las que el nombre del autor no esta repetido.

Funciones de agregación

La cláusula GROUP BY no encuentra su mayor sentido hasta que no conozcamos las funciones de agregación, aquellas que nos permiten operar sobre las columnas de cada grupo de filas para efectuar algún tipo de totalización. En la tabla 4.2 se indica el nombre y finalidad de cada una de estas funciones.

Función	Operación que efectúa
COUNT	Cuenta el número de filas que contienen un valor no nulo en la columna indicada y lo devuelve como resultado.
AVG	Halla el valor medio de los valores existentes en la columna indicada y lo devuelve como resultado.
MIN	Devuelve el mínimo valor existente en la columna indicada.
MAX	Devuelve el máximo valor existente en la columna indicada.

SUM	Calcula la suma de los valores contenidos en la columna indicada y lo devuelve como resultado.

Tabla 4.2. Funciones de agregación de SQL

Todas estas funciones deben ir seguidas de unos paréntesis entre los cuales se indicará el nombre de una columna, sobre la que se llevará a cabo la operación descrita. Por ejemplo:

```
arojas=> select autor,count(titulo) as num_titulos from libros group by autor;
```

autor	num_títulos
Li	1
Ramon, Juan	1
Schroder	1
Balzak, H.	1
Dios, Juan de	1
Munoz Molina, Antonio	1
Armando Rojas Marin	1
Eslava Galan, Juan	1
Diez Barrio, German	1
Pilkey, Dav	1
Masters, M	1
Charte, Francisco	2
Shaw, Catherine	1
Farias, Juan	1

(14 filas)

Nota: La mayoría de los RDBMS cuentan, además de las explicadas aquí, con otras funciones de agregación específicas que facilitan la obtención de

datos a partir de cálculos más complejos. Consulta el manual específico de su base de datos para saber cuales puede usar.

En este caso se han tomado las filas de la tabla libros y se han agrupado por la columna autor, lo cual significa hacer tantos grupos distintos como autores existan, incluyendo en cada uno todas las filas que coincidan en esa columna. A continuación se ha contado el número de filas de cada grupo que contiene un valor en la columna título, obteniendo como resultado el nombre de cada autor y el número de libros que le corresponden.

Mientras que la función COUNT puede aplicarse a cualquier columna, ya que se limita a contar el número de filas que tienen un valor en esa columna, SUM, MIN, MAX y AVG no tienen aplicación sobre ciertos tipos de datos. No es posible, por ejemplo, sumar los valores de una columna que contiene un título, ni tampoco hallar el valor máximo, mínimo o medio.

La consulta siguiente, por ejemplo, emplea las funciones MAX y MIN para saber cuál es el mayor y menor código asociado al grupo de libros disponibles o no disponibles, respectivamente. Ver

```
arojas=> select disponible,count(disponible) as count_disponible,
arojas-> min(codigo) as min_codigo,max(codigo) as max_codigo
arojas-> from libros
arojas-> group by disponible;
 disponible | count_disponible | min_codigo | max_codigo
------------+------------------+------------+------------
 S          |               11 |          1 |         15
 N          |                3 |          4 |         10
            |                1 |         11 |         11
(3 filas)
```

4.7. Rangos de salida y lista de valores

Buscar las columnas cuyo contenido está comprendido en un rango de valores específico, o bien en una lista de valores, es algo que puede efectuarse mediante los operadores de relación y lógicos que ya conocemos. Existen, sin embargo, elementos de la sentencia SELECT especialmente indicados para estos casos: los predicados BETWEEN e IN.

Con BETWEEN se crea un filtro basado en un rango de valores, para lo cual es necesario facilitar el límite inferior y superior conforme a la siguiente sintaxis:

WHERE columna BETWEEN inferior AND superior

Este predicado sería equivalente a una expresión condicional compuesta como ésta:

```
WHERE columna >= inferior
AND       columna <= superior
```

El siguiente ejemplo muestra cómo usar BETWEEN para obtener la lista de aquellos socios que se dieron de alta en el año 2003, para lo cual se usan como límites el primer día y el último día del año. Observa que las fechas van entre comillas pero precedidas del tipo DATE, comunicando así al RDBMS que se trata de una fecha. Esta es la sintaxis estándar para filtrar un rango de fechas. Ver

```
arojas=> select apellido_paterno,apellido_materno,nombre from socios
arojas-> where alta_socios between date '2003-01-01' and date '2003-12-31';

 apellido_paterno  apellido_materno  nombre

 Arias             Trera            Belen

 Moreno            Pardo            Antonio
(2 filas)
```

Nota: El formato de la fecha en SQL depende de la configuración regional y existen diversos métodos de conversión y generación de valores que pueden ser interpretados como fechas. Consulta la documentación específica de tu base de datos si vas a utilizarlo habitualmente

O bien utilizar IN facilitando la lista de valores posibles entre paréntesis, como se hace en la siguiente sentencia de ejemplo:

Ventana emergente (pop-up) a partir de la palabra ***Ejemplo:*** que aparecerá en botón que cambia de color de manera intermitente, el cambio de color no debe ser muy rápido. La ventana tendrá el contedido del cuadro situado después de la palabra.

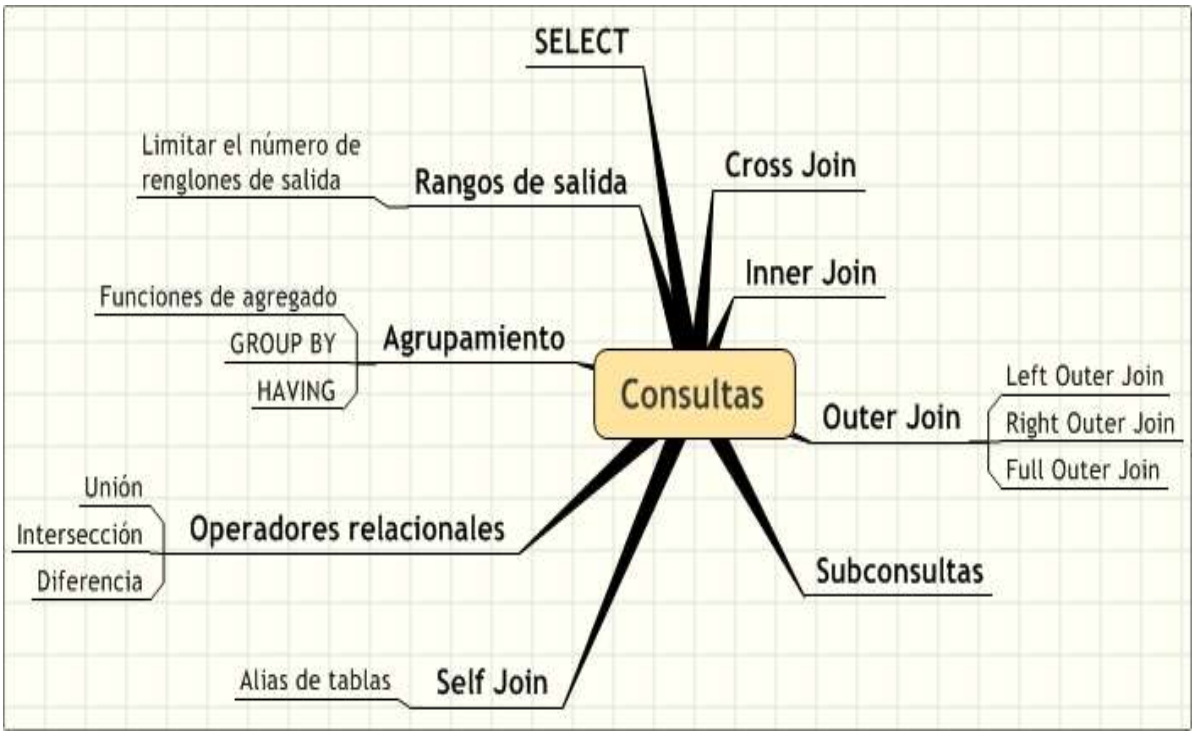
```
arojas=> select apellido_paterno,apellido_materno,nombre,cp from socios
arojas-> where cp in ('23001','23005','23008');

 apellido_paterno  apellido_materno  nombre    cp
 Arias             Trera             Belen     23001
 Moreno            Pardo             Antonio   23008
 Cid               Luque             Manuel    23008
(3 filas)
```

En este caso se han buscado todas las filas de la tabla socios en las que la columna cp contenga el valor 23001, 23005 ó 23008.

Tanto BETWEEN como IN pueden invertir el sentido de la selección de las filas si disponemos delante de ellos el operador NOT, como ocurriría con LIKE.

RESUMEN


SELECT
Limitar el número de renglones de salida
Rangos de salida
Cross Join
Inner Join
Funciones de agregado
GROUP BY
HAVING
Agrupamiento
Consultas
Left Outer Join
Right Outer Join
Full Outer Join
Outer Join
Unión
Intersección
Diferencia
Operadores relacionales
Subconsultas
Alias de tablas
Self Join

BIBLIOGRAFÍA

SUGERIDA

Abbey, Michael, y Michael J. Corey, Oracle guia para el principiante, México, McGraw Hill, 1996.

----------, Oracle, México, McGraw Hill, 1995.

Charte Ojeda, Francisco, SQL, Madrid, Anaya, 2005.

Connolly, Thomas M., y Carolyn E. Begg, Sistemas de bases de datos, 4ª ed., México, Addison Wesley, 2005

Dinerstein, Nelson T., Sistemas de manejo de archivos y bases de datos para microcomputadoras, México, CECSA, 1989.

Houlette, Forrest, Fundamentos de SQL, México, McGraw Hill, 2003.

Koch, George, Oracle 7 manual de referencia, México, McGraw Hill, 1995.

Mendelzon / Ale Introducción a las bases de datos relacionales, México, Pearson Educación, 2000.

Oracle Education, Oracle7 RDBMS Backup and Recovery Strategias, Mexico, 1996

Piattini, Mario G., y Esperanza Marcos, Tecnología y diseño de bases de datos, Madrid, Alfaomega, 2007.

http://www.postgresql.com

WORSLEY C. Y JOSHUA D.DRAKE. 2002. Practical PostgreSQL. Sebastopol, CA: O'Reilly. (Disponible en http://www.faqs.org/docs/ppbook/book1.htm).

Documentación en línea de PostgreSQL versión 8.4, en http://www.postgresql.org/docs/8.4/interactive/index.html.

Unidad 5

Administración

OBJETIVO PARTICULAR

Al finalizar la unidad el alumno será capaz de:

Demostrar las relaciones entre los diferentes manejadores de bases de datos
Identificar la traducción de los datos, el formato de representación interna de las computadoras.

TEMARIO DETALLADO

(10 horas)

5. Administración

5.1. Iniciar y detener el servidor de la base de datos
5.2. Respaldo y recuperación
5.3. Administración de usuarios
5.4. Asignación de privilegios

INTRODUCCIÓN

En este tema se explicarán e ilustrarán los principios de las importantes acciones de respaldar, recuperar, administrar y asignar privilegios a los usuarios. Así como iniciar y detener el servidor de B.D.[6]

El lenguaje SQL tiene una sintaxis prácticamente idéntica en todos los productos que se ajustan. Sin embargo, no significa que no existan ciertas diferencias entre las implementaciones de los distintos fabricantes. Las tres variantes más importantes de SQL, por el número de instalaciones en las que se utilizan, son Oracle PL/SQL, IBM DB2 SQL y Microsoft Transact-SQL, existiendo muchas otras como las usadas en MySQL, PostgreSQL, Informix, etc.

Al utilizar el término base de datos hay que tener en cuenta que puede estar haciéndose referencia al conjunto de información que forma la base de datos, las tablas, filas y columnas, pero también al programa que se encarga de gestionar esa información, lo que se conoce normalmente como RDBMS (*Relational Database Management System*/ Sistema de administración de base de datos relacionales.)

Tipos de RDBMS

Los RDBMS, programas que se encargan de manipular los datos según nuestras indicaciones, se clasifican en dos grandes categorías: locales y servidores. Un RDBMS local se caracteriza por almacenar la información en el propio equipo del usuario que accede a ella. El caso más común es el administrador que utiliza Microsoft Access en su equipo de escritorio. En este caso el RDBMS es Microsoft

[6] En este tema se recomienda consultar la documentación específica de instalación de la base de datos si va a ser el administrador (DBA**).**

Access, un programa que, a demanda del usuario, recupera y manipula datos que están almacenados en el disco de la propia computadora.

En contraposición a éstos, se encuentran los RDBMS que actúan como servidores. Se trata de programas que se ejecutan en un computador central, denominado servidor, al que se conectan los equipos de los usuarios, que se denominan clientes. Estos tienen en sus computadores un software que les permite comunicarse con el servidor, enviando sus sentencias SQL. De esta manera es posible contar con un único almacenamiento de información común a un grupo de usuarios o, incluso, a una empresa entera, facilitando tareas como la seguridad en el acceso a la información, la realización de copias de seguridad y otras labores de mantenimiento. **Que es lo que se verá en este tema.**

Por regla general, no tendrás limitación alguna para operar sobre bases de datos que tengas alojadas en un RDBMS local, en tu propio equipo, lo cual te permitirá ejecutar cualquier tipo de sentencia SQL. Al trabajar con un servidor, sin embargo, la seguridad es un aspecto mucho más importante y se requieren unos permisos para poder efectuar ciertas acciones. Salvo que seas el administrador de bases de datos o bien un usuario con experiencia, normalmente no tendrás acceso directo al intérprete SQL de un RDBMS remoto.

Esto, sin embargo no significa que no puedas probar tus conocimientos del lenguaje SQL en productos como Oracle, SQL Server o MySQL. Lo único que tienes que hacer es instalarlos en tu propio equipo o bien en uno que pueda hacer las funciones de servidor de pruebas.

Durante años, la carrera por conseguir una cuota de mercado en el negocio de software de bases de datos se ha intensificado. Las aplicaciones multimedia y de GUI (interfaz gráfica de usuario) han lanzado a los principales implicados (provedores) hacia empresas millonarias. El trabajo está progresando en una serie

de servidores de medios para satisfacer las demandas de servicios de vídeo, que pretenden llegar a los cuartos de las casas de todo el mundo.

Su servidor proporciona estándares aceptados para un SGBD (sistema de gestión de bases de datos). Un SGBD debe:

- Proveer un archivo para el almacenamiento de datos corporativos.
- Prestar acceso concurrente de usuarios a los datos con propósitos de comunicación, creación de información y actualización.
- Suministrar mecanismos de seguridad eficaces para restringir actividades sobre datos importantes.
- Facilitar mecanismos para asegurar la integridad de los datos.
- Proporcionar un lenguaje de acceso a los datos conforme a los estándares de la industria.
- Permitir que se segmenten las operaciones entre uno o más servidores (computadoras anfitrionas) y muchos clientes (estaciones de trabajo locales y distantes (remotas) sin capacidad de almacenamiento compartido).

Es deseable que la mayoría de las instalaciones proporcionen acceso a una base de datos durante 24 horas, 7 días de la semana y 365 días del año. Los clientes de bases de datos insisten en que las inversiones hechas en los sistemas corporativos están vedadas si deciden moverse a plataformas de hardware distintas y a otros sistemas operativos.

Una puesta eficaz del sistema puede mitigar la necesidad de nuevas compras de hardware y software. Hay muchos productos de terceros disponibles para ayudar a que el DBA (administrador de la base de datos) optimice la configuración, y proporcione un escenario con mejores condiciones para el personal de apoyo de bases de datos –un sistema que sea fiable y no propenso a interrupciones (basado en el hardware, software o usuario). Un SGBD fiable proporciona mecanismos para

copias de seguridad (hacer copias externas de los datos) y recuperación (restaurar desde esa copia externa)

¿Por qué optimizar?

El rendimiento óptimo de una base de datos se traduce a:

- Clientes satisfechos.
- La base de datos responde bien.
- El rendimiento de las transacciones es aceptable para el usuario final.
- Los usuarios pueden obtener información oportuna de su base de datos.
- Uso eficaz de los recursos.
- Se pueden impulsar al límite las UCP (unidad central de proceso) con la máxima utilización de memoria, unidades de disco, controladores y otros componentes de hardware.
- Se pueden minimizar las costosas mejoras de hardware.
- Transferencia de conocimiento.

Instalación

Una instalación eficiente es crucial en el proceso de optimización.

La complejidad al principio es abrumadora. Una vez que se inicie la instalación del software y herramientas de la base de datos, se te pedirá que tomes una serie de decisiones. Las respuestas que des determinarán la estructura de archivos y la situación de los componentes que forman su instalación trata:

- Archivos README.
- Espacio en disco.
- Privilegios (del sistema operativo).
- Registro de instalación.
- Estructura de archivos.
- Aspectos sobre la creación de la base de datos.

- Configuración de espacio en tablas.
- Archivos de control.

Memoria

La memoria de la computadora se mide en gigabytes. Hay muchas estructuras específicas que residen en memoria principal, incluyendo el SGA, PGA, y una serie de procesos de usuario, de segundo plano y de servidor. Estos segmentos de memoria se asignan cuando se inicia una base de datos y permanecen en uso hasta que se cierra la base de datos.

El SGA (área global de sistema) es un segmento de memoria compartida específico de una instancia (una instancia es un SGA acompañado por una serie de procesos necesarios para abrir y trabajar con una base de datos).Además contiene:

- La memoria caché de búfer de base de datos (bloques de la base de datos modificados más recientemente)
- Los búferes del registro de actividad (información destinada para los registros de transacciones de actividad en línea)
- Las áreas SQL compartidas (sentencias SQL analizadas y compiladas que comparten las aplicaciones de usuario y los mecanismos de soporte interno)
- Cursores (segmentos de memoria asociados a una sentencia específica)
- La memoria caché del diccionario de datos.

El PGA (área global del programa) está asociado a un proceso servidor y contiene algunos datos e información de control estadístico. La naturaleza de la información del PGA depende de cómo esté configurada la base de datos.

Los procesos de segundo plano soportan operaciones de la instancia. Por ejemplo, estos procesos se encargan de:

- Recuperación de instancias durante el arranque.
- Limpieza de búferes cuando se aborta un proceso de usuario.

- Archivo de la información del registro de actividad cuando se ejecuta la instancia en modo ARCHIVELOG.
- Escritura de la información de bloque de datos en el disco.
- Escritura de la información del registro de actividad en registros de transacción de actividad en línea.

Entrada salida

A todos los archivos que soportan una instancia (excepto el archivo de inicialización de parámetro leído solamente durante el inicio) se accede por distintas razones durante las actividades de usuario y soporte del sistema. El uso eficiente de las unidades de disco disponibles ayuda en el proceso de afinación que consiste en:

- Segmentos de tabla de índice.
- Partición de tabla e índice.
- Propagación de tabla e índice.
- Segmentos deshacer.
- Segmentos temporales.
- Registros de actividad.
- Controladores de disco.
- Tamaño apropiado de tabla e índice lo que se esta utilizando realmente.

Unidad Central de proceso (UCP)

Dado que la potencia de procesamiento de las computadoras está en relación con el tamaño de la UCP, asegurar un tamaño de UCP adecuado contribuye al proceso de ajuste. Algunos problemas en la ejecución de sistemas se pueden atribuir a que la UCP es demasiada pequeña, para detectar cuando la UCP no es suficientemente grande u óptima analizar:

- Apoyo a la UCP.
- Opción de petición paralela.
- Como está de ocupada la UCP.

- Maximización de la potencia de la UCP.
- Control de sesión.

Puesta a punto de la aplicación

Los desarrolladores son responsables de que el código que escriban se adapte a los estándares de afinación aceptados. El búfer de memoria cache asegura que la información necesaria que da el soporte a la aplicación se lea directamente de la memoria en lugar de leerse solamente del disco. Como se sugiere considerar los siguientes:

- El área SQL compartida.
- Procesamiento de sentencias SQL.
- Utilización de código genérico.
- Optimización basada en el costo.
- Seguimiento SQL.
- Indexación de columnas.
- Bloqueo.

¿Quién optimiza la base de datos?

El **DBA** (administrador de la base de datos) es responsable de la optimización del rendimiento del software. Los **desarrolladores** son socios en este ejercicio de afinación posterior. También, los de **soporte técnico** (expertos de hardware) juegan una parte asegurándose que el hardware funcione de forma eficiente. Estas tres personas clave necesitan trabajar juntas para asegurar un buen rendimiento de la base de datos y de las aplicaciones que soporta.

El administrador de la base de datos

El DBA es responsable de la posterior instalación de software, mejoras, ajuste del rendimiento y transferencia de conocimiento al personal asociado con el DBA y desarrolladores de aplicación. Se esperaría que un DBA fuese el punto central para

todos los asuntos relativos a la base de datos y llevase a cabo deberes como los siguientes:

- Atender a los sucesos técnicos de la base de datos (como los eventos internacionales de usuarios de la base de datos, en América del Norte o el forum de usuarios en Europa) y permanecer al tanto de la tecnología emergente.
- Ser consultado por directores o desarrolladores sobre cuestiones o sugerencias sobre nuevos productos.
- Comparar la cantidad de espacio libre en todas las bases de datos frente a las cifras de periodos anteriores.
- Educar a los desarrolladores sobre el uso correcto de índices, gestión de espacio y asuntos técnicos generales sobre la base de datos.
- Inspeccionar la tasa de errores en la memoria caché de diccionario de datos para todas las bases de datos activas.
- Inspeccionar todos los archivos de alerta de la base de datos en busca de errores anormales en la base de datos.
- Codificación y gestión de rutinas posteriores para copias de seguridad y optimizar la base de datos.
- Registro de peticiones de acción técnica con soporte del proveedor

Esta lista da una muestra sobre lo que implica el trabajo del administrador de base de datos.

Desarrolladores de aplicación

La optimización del código de programa es la responsabilidad principal de este personal. Con el DBA, los desarrolladores deben considerar los programas de rendimiento a lo largo del SDCL (ciclo de vida de desarrollo del sistema). El proveedor de la Base de datos proporcionará al desarrollador una guía para las herramientas disponibles con el fin de optimizar las operaciones de programa de base de datos. Con el personal de gestión de instalación, se anima a los

desarrolladores a que pasen más de 25 ó 30 por ciento de su tiempo examinando asuntos relacionados al código mientras escriben. La experiencia dicta que el tiempo invertido durante el análisis y desarrollo sobre el proceso permita amortizar muchos problemas que puedan surgir en el futuro.

Soporte técnico (expertos en hardware)

El personal responsable del mantenimiento de los componentes de hardware de la instalación también juega un papel importante en el proceso de puesta a punto. El mantenimiento a los periféricos, computadora (servidores empresariales) y mejoras de la UCP se deben hacer junto con algunas progresos en la base de datos. Por ejemplo, cuando se emigra de una versión inferior a otra superior o a diferentes servidores de distinta marca.

Cuando se pasa la base de datos al Servidor Corporativo en UNIX, uno debe prestar atención al número de versión del sistema operativo antes de continuar. El DBA necesita la ayuda de los administradores del sistema cuando instala archivos de base de datos sobre distintos dispositivos. Esto puede ser especialmente importante en sistemas UNIX (cuando se elige entre espacio de sistema de archivo y dispositivos raw), pero tiene impacto en todos los entornos multiusuario.

Optimización de la base de datos

El proceso de optimización es un enfoque metódico donde el DBA, desarrolladores y personal de soporte técnico encuentran las dificultades, afinan las aplicaciones, y proporcionan un entorno de hardware adecuado para optimizar el software de la base de datos.

5.1. Iniciar y detener el servidor de la base de datos

Es posible instalar más de una base de datos en una misma máquina, este término denota, de forma más precisa, cualquier conjunto concreto de programas binarios y bases de datos instaladas.

El superusuario es el usuario que es dueño de los archivos de la base de datos y binarios. Como superusuario de la base de datos, no le es aplicable ninguno de los mecanismos de protección y puede acceder a cualquiera de los datos de forma arbitraria. Además, al superusuario se le permite ejecutar programas de soporte que generalmente no están disponibles para todos los usuarios. Ten en cuenta que el superusuario no es el mismo que el superusuario de Unix (que es conocido como root). El superusuario debería tener un identificador de usuario (UID) distinto de cero por razones de seguridad.

El administrador de la base de datos (database administrator) o DBA es la persona responsable de instalar el software de la base de datos con mecanismos para hacer cumplir una política de seguridad para un *site*. El DBA puede añadir nuevos usuarios.

La cantidad mínima de memoria que se requiere para ejecutar el software de base de datos es de 8 MB mínimo. Sin embargo, se verifica una notable mejora en la velocidad cuando ésta se expande a 96 MB o más. La regla es que, por más memoria que usted instale en su sistema, nunca será suficiente.

Verifique que exista suficiente espacio libre en el disco. Necesitara alrededor de 30 MB para los archivos con el código fuente durante la compilación, y unos 5 MB para el directorio de instalación. Una base de datos vacía ocupa aproximadamente 1 MB. De no estar vacía, la base de datos ocupará unas cinco veces el espacio que ocuparía un archivo de texto que contuviera los mismos datos. Si ejecuta las pruebas de regresión, necesitará alrededor de 20 MB extra como espacio temporal.

5.2. Respaldo y recuperación

Esta función es responsabilidad del DBA en la cual los respaldos se efectuarán diarios, semanales, quincenales y mensuales.

La recuperación la solicitará el analista o el líder de proyecto o el programador de la aplicación en los casos que se haya sufrido algún contratiempo.

5.3. Administración de usuarios

En cualquier sistema es necesaria la seguridad y esta necesidad se acentúa cuando la base de datos es multiusuario. Será necesario como mínimo establecer la autenticación y administración de usuarios, la administración de privilegios y funciones, la administración de contraseñas de usuario y el establecimiento de límites de recursos de la base de datos.

Autenticación y Administración de usuarios

Cualquier usuario que intente conectarse a la base de datos debe hacerlo con un nombre de usuario determinado para que el motor de la base de datos autentifique que dicha persona está autorizada para usar la cuenta. La base de datos utiliza autenticación de contraseña, autenticación de sistema operativo y autenticación global del usuario. La autenticación de contraseña consistente en un nombre de usuario y una contraseña asociada, es típica en entornos distribuidos y sobre todo en sistemas cliente servidor. La autenticación de sistema operativo, típica en sistemas de terminales con conexión directa al servidor, consiste en que la base de datos autentica un nombre de usuario usando el sistema operativo del computador que ejecuta el servidor de base de datos. En el caso de la autenticación global de usuario, el motor autentica un nombre de usuario usando un servicio de red externo. En este caso, el manejador utiliza un servicio de seguridad externo, aunque también dispone de sus propios servicios de seguridad para realizar autenticación global de usuario. Este tipo de autenticación suele utilizarse en redes no seguras y en accesos de los usuarios a varias bases de datos.

Crear una contraseña de usuario

La base de datos permite crear un usuario con autenticación de contraseña mediante la sentencia CREATE USER, cuya sintaxis es la siguiente.

En el ejemplo siguiente se crea un usuario de nombre Pba con autenticación por la contraseña suerte, con una cuota de 10 megabytes de espacio en el tablespace por default demo, con tablespace temporal de nombre temp, con 5 megabytes de espacio en el tablespace SYSTEM, con límite a los recursos de la base de datos determinados por el perfil app_user y con pasword expirada, de modo que el usuario deberá cambiarla después de hacer *login* en la base de datos.

1. SQL> CREATE USER pba
2. IDENTIFIED BY suerte
3. DEFAULT TABLESPACE demo
4. QUOTA 10M ON demo
5. TEMPORARY TABLESPACE temp
6. QUOTA 5M ON system
7. PROFILE app_user
8. PASSWORD EXPIRE;

En el ejemplo siguiente se crea un usuario global de nombre global_user con tablespace por default tbs_1 y con una cuota de espacio en él de 56 megabytes.

SQL> CREATE USER global_user
IDENTIFIED BY GLOBALLY AS ’CN=analyst, OU=division1, 0=**nombre de la base de datos,** c=us’
DEFAULT TABLESPACE tbs_1
QUOTA 5M on tbs_1;

Nombres de usuario y grupos

Para crear un nuevo usuario, ejecuta el programa createuser.

Para añadir un usuario o un grupo de usuarios a un nuevo grupo uno de los usuarios debes crear el grupo y añadir al resto a ese grupo. En Postgres estos pasos no pueden realizarse actualmente mediante el comando create group. Los grupos se definen añadiendo los valores a la tabla pg_group, y usando el comando grant para asignar privilegios al grupo.

Actualmente no hay una forma fácil de crear grupos de usuarios. Hay que añadirlos/actualizarlos uno a uno en la tabla pg_group table. Por ejemplo: jolly=> insert into pg_group (groname, grosysid, grolist) jolly=> values ('posthackers', '1234', '{5443, 8261}'); INSERT 548224 jolly=> grant insert on foo to group posthackers; CHANGE jolly=> Los campos de pg_group son: * groname: El nombre del grupo.

Este campo debe de ser únicamente alfanumérico. No añadas subrayados u otros signos de puntuación. * grosysid: El id del grupo. El tipo del campo es int4 y debe de ser único para cada grupo. * grolist: La lista de id de usuarios que pertenecen al grupo. Este campo es de tipo int4.

Agregar y Eliminar Usuarios

Createuser permite que usuarios específicos accedan a la base de datos dropuser elimina usuarios y previene que éstos accedan a la base de datos.

Estas órdenes sólo afectan a los usuarios con respecto a la base de datos; no tienen efecto en otros privilegios del usuario o en su estado con respecto al sistema operativo.

Copia de seguridad y restauración

Deben realizarse copias de seguridad de las bases de datos regularmente. Dado que los diversos manejadores de base de datos gestionan sus propios archivos en el sistema, no se recomienda confiar en la copia de seguridad del sistema para las copias de respaldo de las bases de datos; no hay garantía de que los archivos estén en un estado seguro que permita su uso después de la restauración.

Las bases de datos proporcionan dos utilerías para realizar las copias de seguridad de su sistema: pg_dump para copias de seguridad de bases de datos individuales y pg_dumpall para realizar copias de seguridad de toda la instalación de una sola vez.

La copia de seguridad de una sola base de datos puede realizarse usando la siguiente orden:

```
% pg_dump nombredb > nombredb.pgdump
```

y puede ser restaurada usando

```
cat nombredb.pgdump | psql nombredb
```

Esta técnica puede usarse para mover bases de datos a una nueva localización y para renombrar bases de datos existentes

Usuario de la base de datos

Los usuarios de la base de datos deben ser "creados para esa base de datos", es decir, creados en los registros internos de la base de datos, antes de poder operar con la base de datos. El usuario de **Unix** debe ser declarado usuario de de la base de datos. Esta operación debe hacerse operando como el usuario de la base de datos, accediendo a este usuario desde root. Es sumamente importante crear un

usuario con privilegios de administrador para la base de datos, para evitar tener que operar como root y luego no personalizar.

5.4. Asignación de privilegios

Los grupos sirven para simplificar la asignación de privilegios. Los privilegios ordinarios deberán ser asignados a un único usuario, uno cada vez. Esto puede resultar tedioso si los usuarios a los que hay que asignar los mismos accesos a una gran variedad de objetos de base de datos, es muy compleja.

Los grupos son creados para evitar este problema. Un grupo requiere un nombre, y puede ser creado vacío (sin usuarios). Una vez creado, los usuarios que se pretende compartan permisos comunes son añadidos todos al grupo, y quedan asociados al grupo por su número de miembro. Los permisos en los objetos de base de datos son entonces asignados al grupo, en vez de a cada uno de los miembros del grupo. Para un sistema con muchos usuarios y bases de datos, los grupos hacen que la asignación de permisos sea más cómoda para el administrador.

Nota: Los usuarios pueden pertenecer a cualquier número de grupos, o a ninguno.

Listado de usuarios

Permite ver los usuarios registrados en el sistema, (recuerda que todos los usuarios pertenecen por defecto a la lista 0)

Revisión de usuarios pendientes de registro

Cuando un usuario solicita el registro, éste queda almacenado en una tabla temporal. El administrador, periódicamente, debe revisar la lista de usuarios pendientes de registro, y previa comprobación de datos, proceder al alta de éstos

Alta manual en el registro de usuarios

Alternativamente, el administrador puede proceder a realizar el alta de un usuario de forma manual, sin intervención de éste. Para ello debe conocer previamente los datos que identifican al usuario.

Edición de datos de usuarios

Utilizaremos esta opción en el caso de desear modificar los datos referidos a un usuario. Normalmente el usuario tiene acceso a un menú propio para proceder por sí mismo a la modificación, por lo que sólo se usará este formulario en los siguientes casos:

Cambio de Nombre y/o Apellidos
Cambio de categoría de usuario (identificador de grupo)
Generación de una nueva contraseña
Si la edición de datos conlleva un cambio en la dirección de correo electrónico, el usuario recibirá la notificación del cambio tanto en la dirección antigua como en la nueva

Eliminación de registro de usuarios

El nombre es auto-explicativo. Al eliminar a un usuario del sistema, toda la información generada por éste, es, dependiendo del caso, bien eliminada a su vez, o bien asignada al usuario con identificador cero (nadie)

Categorías de usuarios

Free-Vote agrupa los usuarios en diversas categorías, en función de los privilegios de acceso que tienen. Una de las tareas del administrador es proporcionar estos privilegios.

El nivel de privilegio de un usuario se almacena en el campo groupid de la tabla usuarios. Se definen los siguientes niveles en orden creciente de privilegios:

- Usuario autorizado a solicitar nuevas consultas
- Usuario con derecho a crear consultas sin autorización ulterior
- Administrador con derecho a añadir usuarios y consultas pendientes de aprobación. Derecho a crear nuevas listas de usuarios
- Derecho a modificar y/o borrar usuarios, listas y consultas
- Derecho de acceso a la consola de SQL

El privilegio más bajo corresponde al groupid 0 (cero). El nivel 6 (acceso a consola) permite acceso total a los recursos de la aplicación, incluida la posibilidad de alterar resultados de las consultas. **Sólo el administrador de la aplicación debería poseer este nivel de acceso.**

El nivel 4 permite crear y borrar usuarios, listas, y consultas. El nivel 5 permite la modificación de éstas una vez creadas. El resto de los niveles corresponden a usuarios sin categoría de administrador (salvo el 3, que permite dar de alta consultas sin necesidad de aprobación previa).

Tareas del administrador

El administrador del sistema tiene las siguientes tareas:

- Instalar y configurar el sistema
- Registrar -en caso necesario- el sistema en los organismos competentes
- Revisar y en su caso dar de alta las solicitudes de registro y alta de consultas
- Crear/Editar/Borrar y la asignación de usuarios a éstas

- Asignar privilegios a los diversos usuarios
- Supervisar funcionamiento y logs del sistema

Alguna de estas tareas puede ser llevada a cabo por usuarios con privilegios especiales desde la interfaz de web. Para la supervisión del funcionamiento y del registro de eventos se debe tener acceso a una cuenta en el servidor

Tareas periódicas.

Las tareas de *backup* (periódicas de respaldo y recuperación), y limpieza y consistencia de la base de datos, se realizan automáticamente.

Las tareas que se realizan son las siguientes:

- Revisión de la base de datos, dando de baja aquellas consultas cuyo plazo de vigencia ha caducado. Esta tarea se realiza todos los días
- Copia de seguridad de la base de datos y de sus contenidos. Para ello, en el proceso de instalación, el administrador habrá definido unos directorios de backup y log
- Rotación de backups y registro de sucesos
- Reindexación y limpieza de la base de datos

Procedimientos de Backup y Recuperación

Si bien el script que se ejecuta bajo estas órdenes realiza automáticamente las tareas de backup, en ocasiones es necesario realizar dicha tarea a mano. Para ello deberemos entrar en la base de datos como administrador de la base de datos, y procederemos como se describe a continuación.

Para realizar un backup ejecutamos:

```
pgdump -h $DBHOST -p $DBPORT -u $DBNAME > $FILE
```

Sustituyendo las variables por su valor correspondiente. El sistema nos preguntará el nombre de usuario y la contraseña correspondientes al administrador (usuario xxxxx "operador")

Para recuperar los contenidos tras una caída del sistema, procederemos como sigue:

Asegurarse que el usuario xxxxx "operador" está definido, y que la base de datos está creada

Ejecutar el comando (substituyendo las variables por su valor correspondiente):
psql -h $DBHOST -p $DBPORT -U $DBUSER -e $DBNAME < $FILE
La base de datos queda así restaurada

Respaldo

Examina cuidadosamente el contenido de la base de datos actualizada. Si encuentras algún problema, entonces necesitarás recuperar sus datos restableciendo. Su respaldo completo pg_dump. Puede eliminar el directorio data.old/ cuando se encuentre satisfecho con los resultados obtenidos.

Recuperación

Algunas bases de datos disponen de distintos mecanismos de protección para situaciones de caída del sistema, fallas en el disco duro, etc. La pérdida de un archivo de base de datos importante debido a un error de operación, la alteración (corrupción) de un archivo o una falla en el disco son problemas serios para los que hay que estar preparados. Por ejemplo, para restaurar el archivo de datos perdido hay que tener una copia de seguridad de la base de datos que contiene el archivo de la base de datos eliminado. Asimismo, para recuperar todo el trabajo efectuado desde que se utilizó la copia de seguridad hay que tener los grupos de **registros de transacciones** necesarios. Los **mecanismos de recuperación y copia de**

seguridad deben recuperar la base de datos de cualquier tipo de pérdida de archivos.

Registro de transacciones

Para mostrar la configuración actual de los registros de transacciones de la base de datos nos conectamos a la misma con privilegios de administrador. Una sesión de administrador privilegiada tiene acceso a los privilegios SYSDBA y SYSOPER.

Con SYSOPER se puede iniciar y cerrar un servidor de base de datos montar, abrir, cerrar, copiar y recuperar una base de datos y administrar su estructura de registro de transacciones. Con SYSDBA se puede llevar a cabo cualquier operación de base de datos y conceder cualquier privilegio de sistema a otros usuarios. Para establecer una sesión de administrador privilegiada SYSDBA para la instancia de inicio con la cuenta SYS.

Privilegios

Una vez creados los usuarios será necesario dotarlos de privilegios para que puedan realizar operaciones específicas en la base de datos. Estos privilegios suelen clasificarse en privilegios del sistema (permiten al usuario realizar algún tipo de operación que afecta a todo el sistema) y privilegios de objeto (permiten al usuario realizar operaciones específicas sobre objetos específicos de la base de datos como tablas, vistas, etc.). **Para conocer la lista y nomenclatura de todos los privilegios es necesario acudir al manual de referencia de la base de datos de la instalación donde trabajas.**

Los privilegios de sistema más comunes son: CREATE SESSION (conexión al servidor de base de datos y establecimiento de sesión), CREATE TABLE (crear una tabla en el esquema propio), CREATE ANY TABLE (crear una tabla en cualquier esquema de la base de datos), SELECT ANY TABLE (hacer peticiones a cualquier tabla de la base de datos), EXECUTE ANY PROCEDURE (ejecutar cualquier

procedimiento almacenado, función o componente de paquete de base de datos) y ALTER DATABASE (modificar la estructura física y la capacidad del sistema de base de datos):

Los privilegios de objeto dependen del tipo de objeto. Para una tabla los privilegios típicos son: SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX y REFERENCES. Para una vista los privilegios más comunes son: SELECT, INSERT, UPDATE y DELETE. Para una secuencia los privilegios más comunes son SELECT y ALTER y para un procedimiento el privilegio más común es EXECUTE.

La sentencia que permite administrar privilegios es GRANT. Su sintaxis es la siguiente:

```
GRANT
{privilegios_sistema
| privilegios_objeto};
```

Ventana emergente (pop-up) a partir de la leyenda ***MODIFICAR USUARIO*** que aparecerá en botón que cambia de color de manera intermitente, el cambio de color no debe ser muy rápido. La ventana tendrá el contedido del cuadro situado después de la leyenda.

MODIFICAR USUARIO

Nombre

MODIFICAR USUARIO — Modificar la información de la cuenta de usuario

Synopsis

MODIFICAR USUARIO nombre de usuario
[WITH PASSWORD 'palabra clave']
[CREATEDB | NOCREATEDB] [CREATEUSER | NOCREATEUSER]
[VALID UNTIL 'abstime']

Instrucciones SQL

Entradas

Nombre de usuario

El nombre del usuario cuyos detalles van a ser modificados

Palabra clave

La nueva palabra clave que va a ser usada en esta cuenta.

CREATEDB

NOCREATEDB

Estas cláusulas definen la capacidad de un usuario para crear bases de datos. Si se especifica CREATEDB, el usuario podrá definir sus propias bases de datos.

Usando NOCREATEDB se rechaza a un usuario la capacidad de crear bases de datos.

CREATEUSER

NOCREATEUSER

Estas cláusulas determinan si un usuario está autorizado a crear nuevos usuarios él mismo. Está opción hace ser además al usuario un superusuario que puede pasar por encima de todas las restricciones de acceso.

Abstime

La fecha (y, opcionalmente, la hora) en la que la palabra clave de este usuario expirará.

Instrucciones SQL

Resultados

MODIFICAR USUARIO

Mensaje recibido si la modificación es correcta.

ERROR: MODIFICAR USUARIO: usuario "nombre de usuario" no existe

Mensaje de error recibido si el usuario especificado no existe en la base de datos.

Descripción, MODIFICAR USUARIO se usa para cambiar los atributos de la cuenta de un usuario de SQL. Sólo un superusuario de una base de datos puede cambiar privilegios y fechas de caducidad de palabras clave con esta orden. Ordinariamente los usuarios sólo pueden cambiar su propia palabra clave.

Usar CREAR USUARIO para crear un nuevo usuario y DROP USER para eliminar un usuario.

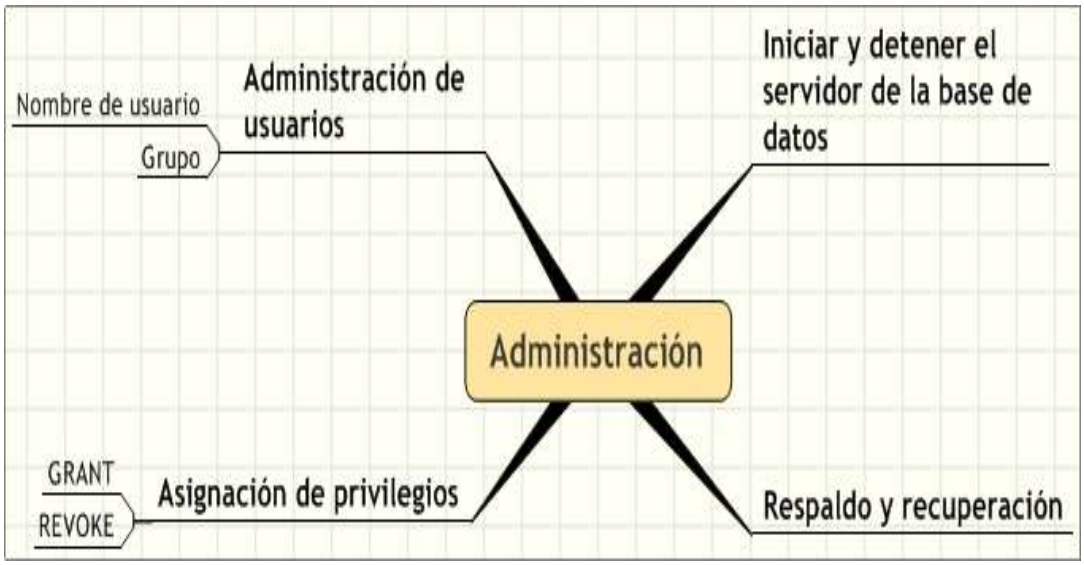
Modo de uso

Cambiar la palabra clave de un usuario:

MODIFICAR USUARIO David CON PALABRA CLAVE ’hu8jmn3’; Cambiar la validez de un usuario hasta la fecha MODIFICAR USUARIO Manuel VALIDO HASTA ’31 En 2030’;

Cambiar la validez de un usuario hasta la fecha, especificando que su autorización expirara al mediodía del 4 de Mayo de 1998 usando la zona horaria que tiene 1 hora

RESUMEN


Nombre de usuario
Grupo
Administración de usuarios
Iniciar y detener el servidor de la base de datos
Administración
GRANT
REVOKE
Asignación de privilegios
Respaldo y recuperación

.

BIBLIOGRAFÍA

SUGERIDA

Abbey, Michael, y Michael J. Corey, Oracle guia para el principiante, México, McGraw Hill, 1996.

----------, Oracle, México, McGraw Hill, 1995.

Charte Ojeda, Francisco, SQL, Madrid, Anaya, 2005.

Connolly, Thomas M., y Carolyn E. Begg, Sistemas de bases de datos, 4ª ed., México, Addison Wesley, 2005

Dinerstein, Nelson T., Sistemas de manejo de archivos y bases de datos para microcomputadoras, México, CECSA, 1989.

Houlette, Forrest, Fundamentos de SQL, México, McGraw Hill, 2003.

Koch, George, Oracle 7 manual de referencia, México, McGraw Hill, 1995.

Mendelzon / Ale Introducción a las bases de datos relacionales, México, Pearson Educación, 2000.

Oracle Education, Oracle7 RDBMS Backup and Recovery Strategias, Mexico, 1996

Piattini, Mario G., y Esperanza Marcos, Tecnología y diseño de bases de datos, Madrid, Alfaomega, 2007.

http://www.postgresql.com

Unidad 6

Construcción de la aplicación

OBJETIVO PARTICULAR

El alumno identificará los conceptos y objetivos de conexión, actualización y consultas de la base de datos, además de los aspectos de actualización y consultas.

TEMARIO DETALLADO

(10 horas)

6. Construcción de la aplicación

6.1. Conexión a la base de datos
6.2. Actualización
6.3. Consultas

INTRODUCCIÓN

Puesta a punto de la aplicación

Como en todos los lenguajes de programación de computadoras, la puesta a punto juega un papel importante. Debido a las mejoras en la administración de la memoria incluida, la puesta a punto del programa y de la aplicación se vuelve cada vez más decisiva. Es necesario practicar para optimizar las sentencias SQL, hacer uso de los bloques centrales de código (disparadores de la base de datos, procedimientos, funciones y paquetes), investigar la integridad declarativa y utilizar las herramientas de supervisión del rendimiento de que se dispone. La incorporación de la puesta a punto del programa, el mantenimiento y el desarrollo continuo del mismo, permite codificar de modo que se aprovechen las rutinas de optimización de velocidad y rendimiento que continuamente se añade al software. El DBA y el personal de apoyo de hardware (soporte técnico) pueden poner a punto la base de datos para soportar el hardware, si no se dejan a punto las aplicaciones, se podría echar a perder este esfuerzo.

Configurar índices para acelerar la recuperación de los datos.

El área SQL compartida

Una de las mejoras del rendimiento que se proporciona es la creación de un área en el SGA llamada área compartida. Este segmento de memoria contiene sentencias SQL analizadas y ejecutadas así como la memoria cache del diccionario.

Regla 1 puesta a punto de la aplicación

Debe haber una correspondencia carácter a carácter entre la sentencia que se está examinando y una que ya esté en el área compartida.

TABLA	COLUMNA	TIPO DE DATO
PERSONA	ID	NUMERIC(6)
	POS_ID	NUMERIC(2)
	APELLIDO	VARCHAR (20)
	NOMBRE	VARCHAR (20)
PLANT_DETA	PLANT_ID	NUMERIC (2)
	CIUDAD_ID	NUMERIC (2)
	SITUACION	VARCHAR (20)
SAL_LIMITE	POS_ID	NUMERIC (2)
	SAL_CAP	NUMERIC (8.2)
	TIEMPO_EXTRA	VARCHAR (1)
CIUDAD_TRAB	CIUDAD_ID	NUMERIC (2)
	SDESC	VARCHAR (20)
	LDESC	VARCHAR (60)

Tabla 6.1. Ejemplo de tablas de base de datos

NOTA: Antes de que se lleve a cabo esta comparación, se aplica un algoritmo interno utilizando la nueva sentencia. Después se comprueba los resultados comparando los valores de las sentencias que ya están en el área. Si el nuevo valor coincide con uno que ya esté allí, se lleva a cabo la comparación de cadena definida en la Regla 1.

Regla 2 puesta a punto de aplicación

Los objetos a los que se hace referencia en la nueva sentencia son exactamente los mismos que los objetos de una sentencia que han pasado la comparación de la Regla 1.

Para este ejemplo, los usuarios tienen acceso a los objetos de la Tabla 6.2.

NOTA: si se modifica un objeto al que se hace referencia en la sentencia de SQL del área compartida, a la sentencia se le coloca un indicador que la inválida. La siguiente vez que se le pase una sentencia que sea igual a la sentencia invalidada, se sustituirá la antigua sentencia por la nueva, ya que se ha modificado el objeto oculto.

USUARIO	NOMBRE DE OBJETO	VIA DE ACCESO
mcleodmg	sal_limite	sinonimo privado
	ciudad_trab	sinonimo publico
	plant_deta	sinonimo publico
jproudf	sal_limite	sinonimo privado
	ciudad_trab	sinonimo publico
	plant_deta	propietario de la tabla

Tabla 6.2. Ejemplo de objetos de base de datos para los usuarios

Considera las sentencias de la Tabla 6.3 y por qué pueden o no ser compartidas entre los dos usuarios listados en la Tabla 6.2.

La Tabla 6.3 muestra que los objetos son diferentes. Aunque ambos usuarios tengan un sinónimo privado sal_limite para consultar a la misma tabla de la base de datos, estos sinónimos privados individuales son de hecho objetos de base de datos en sí mismos.

SENTENCIA SQL	CONCORDANCIA DE OBJETOS	POR QUE
select max(sal_cap) from sal_limite	NO	Cada usuario tiene un sinónimo privado **sal_limite** –estos son objetos diferentes.
select count(*) from ciudad_trab where sdesc like 'NEW%';	SI	Ambos usuarios consultan **ciudad_trab** por medio del mismo sinónimo publico –el mismo objeto
select a.sdesc,b. location from ciudad_trab a,plant_deta b where a.ciudad_id =b.ciudad_id;	NO	El usuario mcleodmg referencia **plant_deta** mediante un sinónimo público mientras que el usuario Jproudf es el propietario de la tabla -estos son objetos diferentes.
select * from sal_limite where tiempo_extra is not null;	NO	Cada usuario tiene un sinónimo privado **sal_limite** –son objetos diferentes

Tabla 6.3. Resolución de objetos de sentencias de SQL

REGLA 3 DE PUESTA A PUNTO DE LA APLICACIÓN

Si se consultan variables con enlace, deben tener el mismo nombre tanto en la sentencia existente como en la nueva.

Arquitectura cliente- servidor en tres niveles

La necesidad de mejorar la escalabilidad de los sistemas empresariales hizo que se pusiera en cuestión este modelo tradicional cliente-servidor en dos niveles. A mediados de la década de los 90, a medida que las aplicaciones fueron creciendo en complejidad, y debían tener la capacidad de implantarse en centenares o miles de clientes, el lado del cliente comenzó a mostrar síntomas de dos problemas que impedían conseguir una escalabilidad adecuada:

- Se utilizaban clientes ‘complejos’, lo que requería unos recursos considerables en la computadora del cliente para que éste pudiera ejecutarse de forma adecuada. Estos recursos incluían espacio en disco, memoria RAM y potencia de procesamiento de CPU.
- Las tareas de administración en el lado del cliente eran bastante significativas.

En 1995 apareció una nueva variación del modelo tradicional cliente-servidor en dos niveles, para intentar resolver los problemas de escalabilidad en las empresas. Esta nueva arquitectura proponía tres niveles, cada uno de los cuales puede ejecutarse en una plataforma distinta:

(1) El nivel de interfaz de usuario, que se ejecuta en la computadora del usuario final (el cliente).
(2) El nivel de lógica de negocio y procesamiento de datos. Este nivel intermedio se ejecuta en un servidor que a menudo se denomina servidor de aplicaciones.
(3) Un SGBD que almacena los datos requeridos por el nivel intermedio. Este nivel puede ejecutarse en un servidor independiente, denominado servidor de base de datos.

Como se ilustra la Figura 6.1, el cliente sólo es ahora responsable de la interfaz de usuario de la aplicación y quizás, de realizar algún tipo de procesamiento lógico simple, como por ejemplo la validación de los datos de entrada, por lo que con esta arquitectura se dispone de lo que se denomina clientes ‘simples’. La lógica principal de la aplicación reside ahora en su propio nivel, que se conecta físicamente al cliente y al servidor de base de datos a través de una red de área local (LAN) o de una red de área extensa (WAN, *wide area network*). Un mismo servidor de aplicaciones puede dar servicio a múltiples clientes.

El diseño en tres niveles tiene muchas ventajas con respecto a los diseños tradicionales de dos niveles o de un nivel. Entre esas ventajas podemos citar:

- El hardware necesario es menos costoso, ya que los clientes son ‘simples’.
- El mantenimiento de las aplicaciones está centralizado, al transferirse la lógica de negocio desde las plataformas de los usuarios finales a un único servidor de aplicaciones. Esto elimina los problemas de distribución de software que preocupa en el modelo tradicional cliente-servidor en dos niveles.
- Al ser mayor la modularidad, resulta más sencillo modificar o sustituir uno de los niveles sin que los otros se vean afectados.
- Resulta más fácil equilibrar la carga de procesamiento al separar la lógica principal de negocio de las funciones de base de datos.

Otra ventaja adicional es que la arquitectura en tres niveles se adapta de forma bastante natural a los entornos web, en los que un explorador web actúa como cliente ‘simple’ y un servidor web actúa como servidor de aplicaciones. La arquitectura en tres niveles puede ampliarse a n niveles, añadiéndose los niveles adicionales para aumentar la flexibilidad y la escalabilidad. Por ejemplo, el nivel intermedio de la arquitectura en tres niveles puede partirse en dos, disponiéndose de un nivel para el servidor web y de otro para el servidor de aplicaciones.

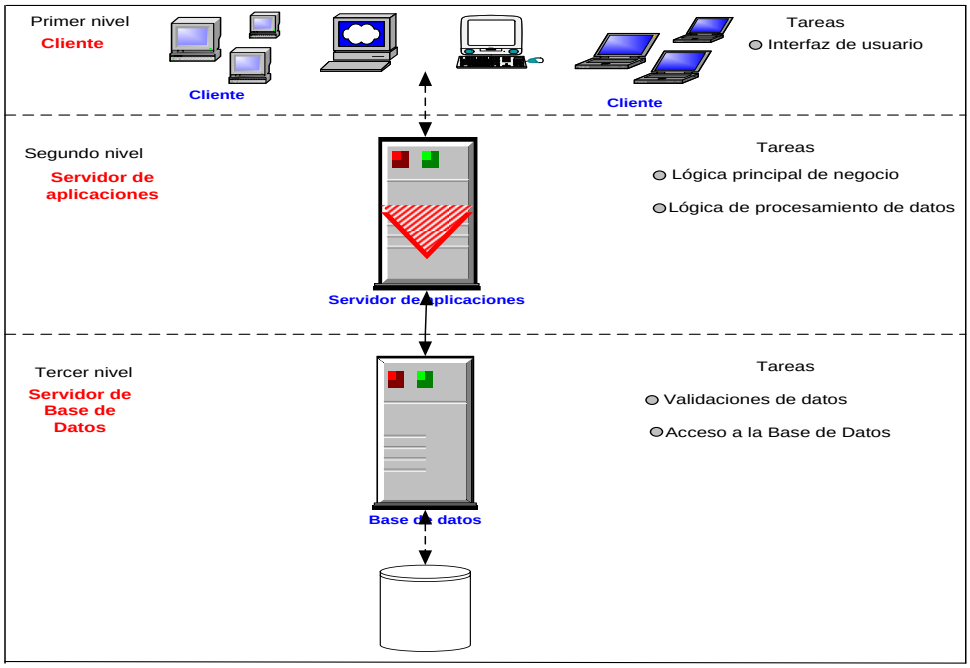


Figura 6.1. Arquitectura en tres niveles

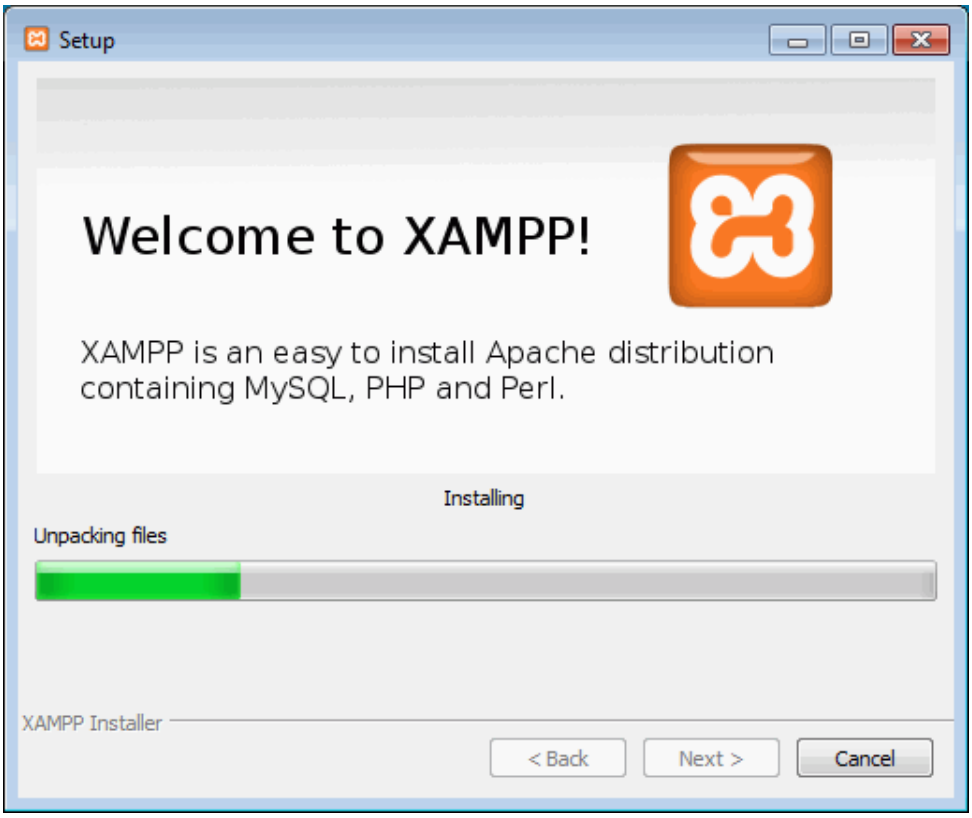
Esta arquitectura en tres niveles ha demostrado ser más adecuada para algunos entornos, como la Internet y las Intranets corporativas, en las que puede utilizarse un explorador web como cliente. También es una arquitectura de gran importancia para los **monitores de procesamiento de transacciones**.

PROCEDIMIENTO PARA INSTALAR EL MANEJADOR DE BASES DE DATOS MYSQL, ASI COMO LA INTERFAZ PHPMYADMIN PARA GESTIONARLA

Bajar el conjunto de aplicaciones XAMPP que se puede conseguir en la siguiente liga, y seguir las instrucciones que nos indique, el programa de instalación.

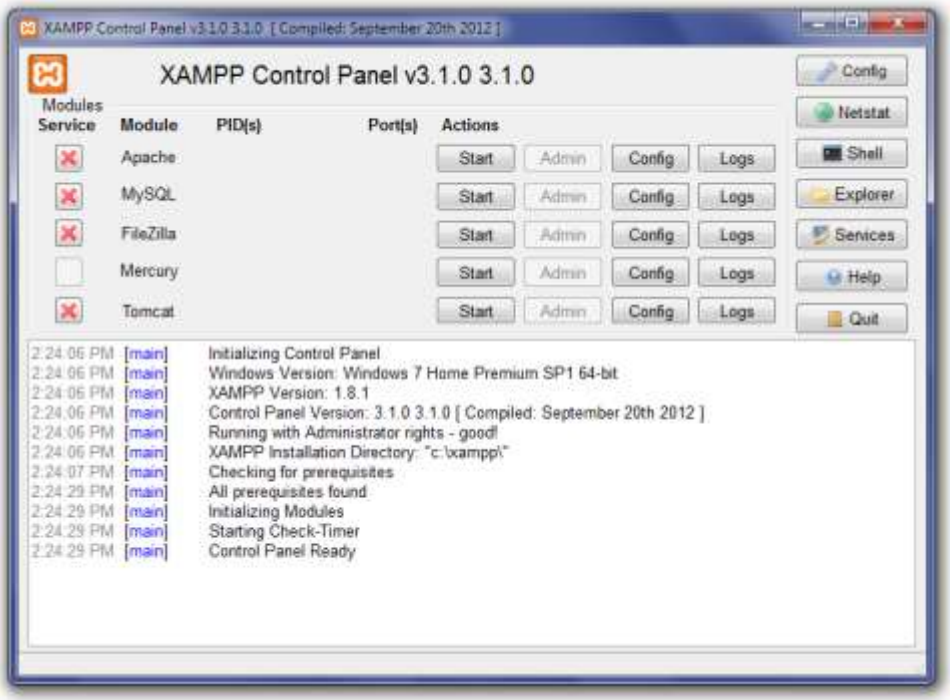
https://www.apachefriends.org/es/index.html

Aparecera un pantalla que indicara el proceso de instalación.



Fuente:https://www.google.com.mx/search?q=instalaci%C3%B3n+de+xampp&tbm=isch&imgil=Sc72lm6IgNxQDM%253A%253BtUaUXNrDConwUM%253Bhttp%25253A%25252F%25252Fwww.mclibre.org%25252Fconsultar%25252Fphp%25252Fotros%25252Fin_php_instalacion.html&source=iu&pf=m&fir=Sc72lm6IgNxQDM%253A%252CtUaUXNrDConwUM%252C_&usg=__hicBPSwu3KM_9dLoI8qgEAJ9fT4%3D&biw=1024&bih=651&ved=0ahUKEwj45-KFoczOAhVL5mMKHYoDARsQyjcIMw&ei=gVe2V7ijEsvMjwOKh4TYAQ#imgrc=Sc72lm6IgNxQDM%3A

Una vez instalado el software debes abrir el panel de control de XAMPP y verificar que los servicios de MySQL y Apache están corriendo.



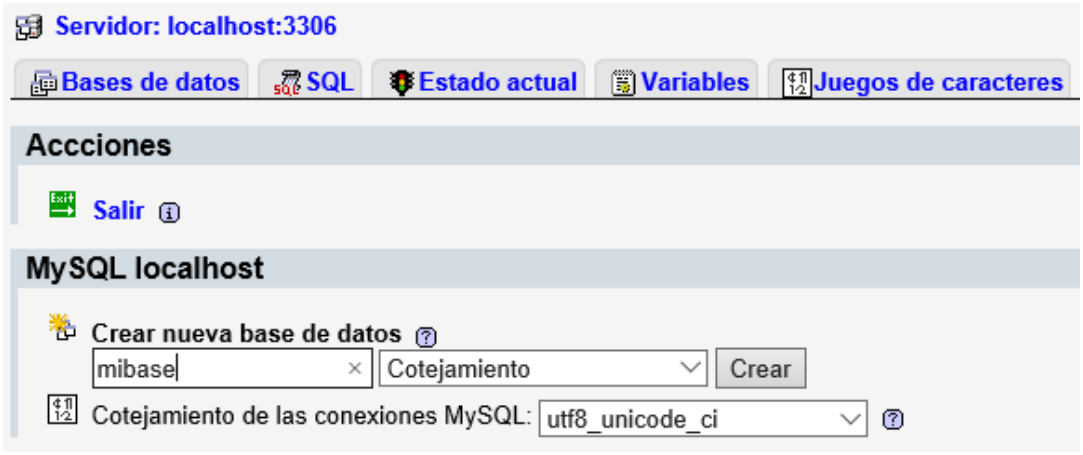
Fuente:https://www.google.com.mx/search?q=xampp+panel+de+control&biw=1024&bih=651&source=lnms&tbm=isch&sa=X&sqi=2&ved=0ahUKEwj4uMaIoszOAhVImx4KHUagAScQ_AUIBigB#imgrc=ACf084frR2M1XM%3A

Posterior a esto, debes teclear en el navegador las siglas:

http://localhost/phpmyadmin

Te aparecerá la interfaz de phpmyadmin, el cual es un programa que te permite gestionar la base de datos de manera gráfica.

Entonces creas una base de datos, en este caso la base se llama mibase.

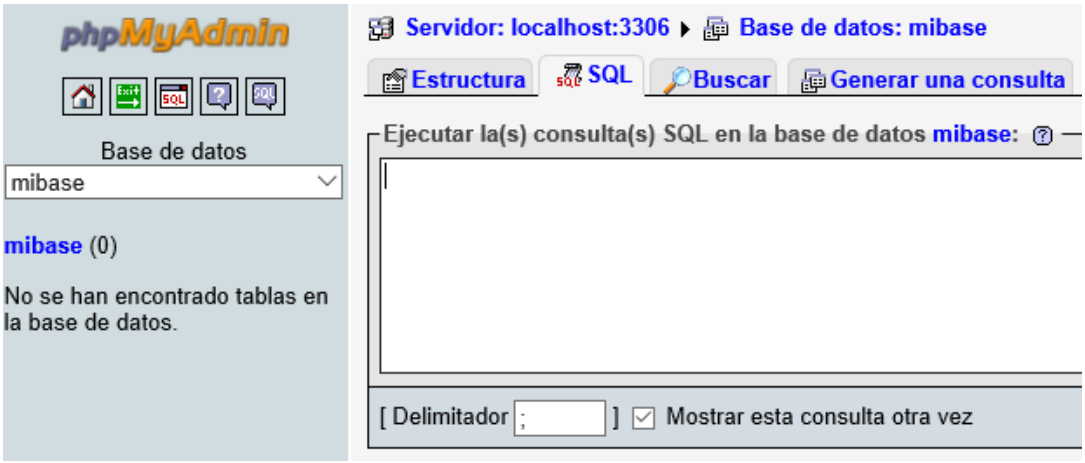

Servidor: localhost:3306
Bases de datos
SQL
Estado actual
Variables
Juegos de caracteres
Accciones
Salir
MySQL localhost
Crear nueva base de datos
mibase
Cotejamiento
Crear
Cotejamiento de las conexiones MySQL:
utf8_unicode_ci

6.1. Conexión a la base de datos

Para poder realizar la conexion a la base de datos, solo tienes que volver a teclear la dirección de:

http://localhost/phpmyadmin

Posterior a esto seleccionas la base de datos creada. Para poder utiizar sentencias de SQL, solo tienes que seleccionar la pestaña que dice SQL y podras ejecutar las sentencias.



6.2. Actualización

Inserción, actualización, eliminación

Los temas anteriores se han centrado en técnicas básicas de consulta, todas orientadas en torno a la tarea de obtener datos de una base de datos. Este tema da un giro y se centra en los tres temas siguientes:

- Inserción de nuevos registros en la base de datos.
- Actualización de los registros existentes.
- Eliminación de registros que ya no son necesarios.

La inserción suele ser una tarea sencilla. Comienza con el simple problema de insertar una única fila. Sin embargo, muchas veces es más eficaz utilizar un método basado en conjuntos para crear nuevas filas. Con este fin, también encontrarás técnicas para insertar muchas filas al mismo tiempo.

Del mismo modo, la actualización y la eliminación empiezan como tareas sencillas. Puedes utilizar un registro y puedes eliminar un registro. Pero también puedes actualizar conjuntos enteros de registros de una vez y de maneras muy eficaces. Y existen muchas formas prácticas de eliminar registros; por ejemplo, puedes eliminar filas de una tabla dependiendo de si existen o no en otra tabla.
SQL incluso cuenta con una forma, una novedad relativa del estándar, por la que se puede insertar, actualizar y eliminar simultáneamente. Tal vez no parezca muy útil ahora, pero la instrucción MERGE representa una manera muy eficaz de sincronizar la tabla de una base de datos con una base de datos externa (como un suministro de archivos sin formato de un sistema remoto).

Aunque la mayor parte de las operaciones que se llevan a cabo sobre una gran parte de las bases de datos empresariales son de consulta, a fin de elaborar

informes, gráficos y cualquier otro tipo de resultado, también es habitual que a partir de aplicaciones de proceso de transacciones en línea se efectúen operaciones de inserción, modificación y borrado, enfocadas todas ellas a mantener actualizada la información que reside en la base de datos

La sentencia INSERT

Para agregar filas a una tabla, o en las columnas obtenidas mediante una vista si ésta es actualizable, nos aprovecharemos de la sentencia INSERT que, al igual que SELECT, forma parte del subconjunto de SQL conocido como DML o de manipulación de datos.

La sintaxis general de esta sentencia es la siguiente:

```
INSERT INTO tabla [(col1, col2 …)]
VALUES (val1, val2…)
```

Tras INSERT dispondremos la palabra INTO y a continuación, el nombre de la tabla a la que van a añadirse datos. Esta irá seguida opcionalmente, entre paréntesis, de los nombres de las columnas donde quieren insertarse los valores enumerados tras la palabra clave VALUES. Habrá tantos valores como columnas se indiquen, o existan en la tabla, agregándose una nueva fila conteniendo esos valores. Si alguno de ellos incumple una restricción, por ejemplo duplicando un valor que no puede estar repetido o bien dejando como nula una columna que no puede ser NULL, la operación completa fallará, obtendremos un mensaje de error y no se producirá ningún cambio en la tabla.

Inserción de valores por posición

Si omitimos la indicación de los nombres de columnas donde van a insertarse los valores, la sentencia INSERT tomará el formato mostrado a continuación:

INSERT INTO tabla
VALUES (val1, val2 …)

Los valores proporcionados tras VALUES han de aparecer necesariamente en el mismo orden en que se definieron las columnas de la tabla, existiendo tantos valores como columnas existen en la tabla.

Esos valores, además, deben coincidir en tipo y no incumplir ninguna de las restricciones que pudieran haberse establecido.

Suponiendo que quisiésemos agregar un nuevo libro a la tabla libros, deberíamos suministrar un valor para la columna código, otro para signatura, título, autor y disponible, en ese mismo orden. Por ejemplo:

INSERT INTO libros
VALUES (12,’G SHA inc’, ’La incognita Newton’, ‘Shaw, Catherine’, ’S’);

De entregar menos valores de los necesarios, el RDBMS nos informará del error con un mensaje similar al obtenido en la siguiente sentencia:

arojas=> insert into libros values (16,'Principios de Contabilidad','Gonzalez Torres, Patricia');
ERROR: el valor es demasiado largo para el tipo character varying(10)

arojas=> insert into socios (nif,nombre,apellido_paterno,apellido_materno,alta_socios)
arojas-> values ('63273827G','David','Charte','Luque',current_date);

ERROR: el valor null para la columna Â«direccionÂ» viola la restricciÃ³n not null

Algo similar ocurrirá al incumplir una restricción, por ejemplo si intentamos introducir en la columna código un valor que ya exista. Esta columna es la clave principal de la tabla, como tal, no admite valores duplicados. El RDBMS nos lo hace saber con un mensaje de error:

arojas=> insert into libros values (16,'G SHA inc','La incognita Newton','Shaw, Catherine');

ERROR: llave duplicada viola restricciÃ³n unique Â«libros_signatura_keyÂ»
En caso que necesitemos agregar varias filas a una misma tabla, como por ejemplo varios libros nuevos, escribiríamos una serie de sentencias INSERT, si bien algunos RDBMS permiten agregar varias filas en un único paso.

Inserción de valores por nombre de columna

Si no conocemos la posición de las columnas en la tabla, para proporcionar los valores en el orden correcto, o bien vamos a suministrar menos valores que columnas existen en la tabla, será necesario que tras el nombre de ésta, y entre paréntesis, indiquemos los nombres de las columnas donde van a introducirse los valores.

Utilizando esta técnica, la sentencia siguiente añade una nueva fila a la tabla socios agregando valores solamente para parte de las columnas que la componen, puesto que no se adiciona ni la dirección ni el código postal. El RDBMS no genera error alguno porque esas dos columnas pueden quedar nulas, al no haberse establecido la restricción NOT NULL en la definición.

INSERT INTO socios (nif, nombre, apellido_paterno, apellido_materno, alta_socios) VALUES (‘63273827I’, ‘David’, ‘Charte’, ‘Luque’, CURRENT_DATE);

```
arojas=> insert into socios
(nif,nombre,apellido_paterno,apellido_materno,alta_socios)
arojas-> values ('63273827J','David','Charte','Luque',current_date);

ERROR:  el valor null para la columna Â«direccionÂ» viola la restricciÃ³n not null
```

Nota: La definición de la estructura conocerá la restricción NOT NULL y otras restricciones que pueden aplicarse a las columnas de una tabla.

```
arojas=> insert into socios
(nif,nombre,apellido_paterno,apellido_materno,direccion,cp,alta_socios)
arojas-> values ('63273827I','David','Charte','Luque','Pitagoras
      1139',03100,current_date);
INSERT 149764 1
```

Observa cómo se utiliza la función CURRENT_DATE para introducir en la columna alta_socios la fecha actual, la que indique el propio RDBMS.

Indicar los nombres de las columnas en las que deben introducirse los valores es usual cuando, como en este ejemplo, van a proporcionarse menos valores que columnas existentes en la tabla. De no ser así, lo habitual es omitir los nombres de las columnas y sencillamente, enumerar los valores en el orden adecuado.

Si no sabemos en principio cuál es el orden de las columnas, o qué columnas existen en la tabla, podemos obtener dicha información de diversas formas. La más sencilla es ejecutar una sentencia del tipo SELECT * FROM tabla y observar los nombres de las columnas en la cabecera del resultado. Además una consulta de este tipo nos permite saber el nombre y orden de las columnas, pero no el tipo de información que pueden contener, si permiten o no valores nulos, etc.

Las operaciones sobre una base de datos nos permiten llevar a cabo las instrucciones SELECT e INSERT, únicas que conocemos hasta el momento, no son en ningún caso destructivas, es decir, difícilmente causaremos con ellas la pérdida de información ya alojada en la base de datos. Las que trataremos, por el contrario, sí que tienen esa capacidad, de ahí que las abordemos por separado, teniendo especial cuidado en cómo deben utilizarse. Aunque la inserción de datos podría considerarse también una actualización de la información contenida en una base de datos, vamos a tratar las dos operaciones que más se identifican con el término actualización: la modificación de los datos que existen en las filas y su eliminación. Con este fin aprenderemos a utilizar dos nuevas sentencias del lenguaje SQL: UPDATE y DELETE.

Modificación de datos UPDATE

En las tablas de una base de datos, como puede ser la utilizada en los ejemplos, de estos apuntes, se almacenan dos categorías de información: aquella que puede ser considerada invariable y la susceptible de sufrir cambios a lo largo del tiempo. En la primera categoría estarían datos como el nombre y apellidos de una persona, el título de un libro o bien el nombre de su autor, mientras que la segunda podrían tener cabida la dirección donde vive el socio, la disponibilidad de un libro o la fecha en que fue prestado por última vez.

Incluso los datos que son considerados invariables, como el nombre de una persona o el título de un libro, pueden necesitar una modificación en caso de que se hubiesen introducido inicialmente con algún tipo de error. En todos estos casos deberemos recurrir a la sentencia UPDATE, parte del lenguaje de manipulación de datos de SQL. Su sintaxis general es la siguiente:

```
UPDATE [ONLY] tabla
SET col1=valor1, col2=valor2 …
WHERE condición
```

Observa que la cláusula WHERE no se ha introducido entre corchetes, aunque en realidad se trata de una cláusula opcional de la sentencia UPDATE según el estándar. El riesgo de no incluirla es muy grande, ya que la modificación que indicásemos se aplicaría a todas las filas de la tabla.

La finalidad de la cláusula WHERE, en este caso, es seleccionar las filas a cuyas columnas se asignarán los valores indicados en el SET.

Nota: La cláusula ONLY no está contemplada por todos los RDBMS, siendo su finalidad limitar la aplicación de cambios solamente a la tabla indicada, no a tablas cuyos valores pudiesen actuar como claves externas haciendo referencia a la indicada.

Cambiar una columna de una fila

Uno de los casos de uso más habituales de la sentencia UPDATE se da cuando necesitamos cambiar el dato contenido en una columna de una fila determinada, por ejemplo, aplicando el caso a nuestra base de datos, actualizar la columna disponible de un libro que acaba de ser prestado o devuelto. En esta situación suele utilizarse una sentencia del tipo:

```
UPDATE tabla
SET columna = dato
WHERE claveprimaria = valor
```

La selección de la fila a través de la columna que actúa como clave primaria, que en nuestro caso sería el código del libro, garantiza que el almacenamiento del dato en la columna afecte únicamente a esa fila, dado que la clave primaria es irrepetible.

Ejecutamos una primera consulta para comprobar cuál es el estado, disponible o no, de los libros que tenemos en la tabla libros.

```
SELECT codigo, titulo,disponible
FROM libros;
```

```
arojas=> select codigo,titulo,disponible from libros;
```

codigo	titulo	disponible
2	El misterio del perro secuestrado	S
3	El secreto de los pirats	S
4	Mi amigo agapito	N
5	Un cesto lleno de lapices	S
6	Fabulas mitologicas	S
7	Leyendas de las calles de la ciudad de Mexico	N
8	Platero y yo	S
9	En busca del unicornio	S
10	Ventanas de Nueva York	N
11	Cuentos libertinos	
12	La incognita Newton	S
13	Excel 2003	S
14	Curso de Linux	S
15	Curso de Rdbms	S
1	El capitan calzoncillos	S
16	Principios de Contabilidad	
17	Valles Faudoa,Miryam	

(17 filas)

Supongamos que el socio que tenía en préstamo el título Ventanas de Nueva York acaba de devolverlo y por tanto, la columna disponible debe ahora cambiar su contenido actual de N a S, para lo cual ejecutamos la sentencia siguiente:

```
UPDATE libros
SET disponible = ‘S’
WHERE codigo = 10;
```

```
arojas=> update libros set disponible = 'S' where codigo = 10;
UPDATE 1
arojas=> select codigo,titulo,disponible from libros;
```

codigo	titulo	disponible
2	El misterio del perro secuestrado	S
3	El secreto de los pirats	S
4	Mi amigo agapito	N
5	Un cesto lleno de lapices	S
6	Fabulas mitologicas	S
7	Leyendas de las calles de la ciudad de Mexico	N
8	Platero y yo	S
9	En busca del unicornio	S
11	Cuentos libertinos	
12	La incognita Newton	S
13	Excel 2003	S
14	Curso de Linux	S
15	Curso de Rdbms	S
1	El capitan calzoncillos	S
16	Principios de Contabilidad	
17	Valles Faudoa,Miryam	

```
10          Ventanas de Nueva York          S
(17 filas)
```

El RDBMS se limita a indicarnos el número de filas que se han visto afectadas, una información valiosa porque nos permite saber, ya que la actualización ha afectado solamente a una fila, como debía ser, y no a varias. La repetición de la consulta anterior, obteniendo código, título y disponibilidad de todos los libros, será la confirmación definitiva de que todo ha ido bien.

Cambiar varias columnas de una fila

El procedimiento para modificar el contenido de varias columnas pertenecientes a una misma fila, por ejemplo la dirección y el código postal de un socio, es similar al que acaba de describirse en el punto anterior. La diferencia estriba en que tras la cláusula SET, separadas por comas, irán facilitándose las parejas de nombres de columna y nuevo valor a asignar:

SET col1 = valor1, col2 = valor2 ..

Obviamente, cada uno de los valores debe ser del tipo adecuado según la columna a la que vaya a asignarse. La respuesta del RDBMS, nuevamente, se limitará a indicarnos el número de filas que se han visto afectadas. Normalmente recurrimos de nuevo a utilizar la clave primaria de la tabla en la cláusula WHERE, asegurándonos así de no modificar nada más que la fila que corresponde.

La sentencia mostrada a continuación modifica las columnas dirección y código postal de una fila concreta de la tabla socios, fila que identificaremos a partir de la columna nif que es la clave primaria. En la figura 6.2.puede verse una consulta previa a la modificación y otra posterior, apreciándose así el cambio en las columnas mencionadas.

```
UPDATE socios
SET direccion = ‘las Flores, 12’, cp = ‘23021’
WHERE nif = ‘63273827G’;
```

```
arojas=> select * from socios where apellido_paterno like 'Charte%';
```

nif	nombre	apellido_paterno	apellido_materno	direccion	cp	alta_socios
62877137F	Alejandro	Charte	Luque	Las Flores # 12	23021	2002-04-10
74381725T	Francisco	Charte	Ojeda	Las Flores # 12	23021	2005-06-09
63273827G	David	Charte	Luque	Las Flores # 12	23021	2005-06-29
63273827I	David	Charte	Luque	Pitagoras 1139	3100	2007-07-25

(4 filas)

```
arojas=> update socios
arojas-> set direccion = 'Las Flores 12', cp = '23021' where nif = '63273827G';
UPDATE 1
arojas=> select * from socios where apellido_paterno like 'Charte%';
```

nif	nombre	apellido_paterno	apellido_materno	direccion	cp	alta_socios
62877137F	Alejandro	Charte	Luque	Las Flores # 12	23021	2002-04-10
74381725T	Francisco	Charte	Ojeda	Las Flores # 12	23021	2005-06-09
63273827I	David	Charte	Luque	Pitagoras 1139	3100	2007-07-25
63273827G	David	Charte	Luque	Las Flores 12		2005-06-29

(4 filas)

Figura 6.2. Modificamos dos columnas de una misma fila

Nota: Según el estándar SQL, la sentencia UPDATE puede utilizarse tanto para actualizar una tabla como los datos generados por una vista. En la práctica, sin embargo, no todos los RDBMS contemplan la posibilidad de actualizar las vistas.

Modificación de datos en varias filas

Cuando necesitamos asignar el mismo valor a las mismas columnas de varias filas, no es preciso que usemos una sentencia UPDATE individual para cada fila utilizando la clave principal como referencia de selección. En su lugar, sustituiremos el filtro de búsqueda de la cláusula WHERE por uno adecuado que nos permita actuar sobre las filas que nos interesan. Este es un aspecto en el que debe ponerse la mayor atención, ya que un error en dicho filtro podría causar la pérdida de información en filas que no deberían haberse visto afectadas. Lo mejor que podemos hacer, a fin de asegurarnos anticipadamente de que no vamos a manipular las filas inadecuadas, es ejecutar una consulta con el criterio de selección que pretendemos utilizar en la actualización. Las filas obtenidas como resultado serán las mismas que se actualicen con UPDATE.

Veamos un ejemplo partiendo de la consulta siguiente, en la que obtenemos los apellidos y dirección y código postal de todos los socios y en la que se aprecia fácilmente que muchos de ellos no tienen código postal.

```
SELECT apellido_paterno, apellido_materno, direccion,cp
FROM socios;
```

```
arojas=> select apellido_paterno,apellido_materno,direccion,cp from socios;
```

apellido_paterno	apellido_materno	direccion	cp
Arias	Trera	Betania # 7	23001
Moreno	Pardo	Juan Rincon # 2	23008
Charte	Luque	Las Flores # 12	23021
Charte	Ojeda	Las Flores # 12	23021
Lopez	Aguilera	Desconocida	
Lopez	Hilera	Desconocida	
Garcia	Aguilera	Desconocida	

Garcia	Hilera	Desconocida	
Perez	Aguilera	Desconocida	
Perez	Leon	Desconocida	
Cid	Luque	Juan Carlos I # 23	23008
Charte	Luque	Pitagoras 1139	3100
Charte	Luque	Las Flores 12	

(13 filas)

Lo que pretendemos hacer es almacenar el valor 99999 en la columna código postal de todas aquellas filas en las que no tiene valor. Necesitamos, por lo tanto, definir una condición en la cláusula WHERE que nos permita elegir esas filas y no otras. Con este fin podemos utilizar una consulta como la mostrada a continuación, en la que se emplea el operador ‘ ’:

arojas=> select apellido_paterno,apellido_materno,direccion,cp from socios where cp = ' ';

apellido_paterno	apellido_materno	direccion	cp
Lopez	Aguilera	Desconocida	
Lopez	Hilera	Desconocida	
Garcia	Aguilera	Desconocida	
Garcia	Hilera	Desconocida	
Perez	Aguilera	Desconocida	
Perez	Leon	Desconocida	
Charte	Luque	Las Flores 12	

(7 filas)

Puesto que la consulta nos devuelve precisamente las filas que queremos, podemos sustituir la sentencia SELECT por una sentencia UPDATE en la que se mantenga la cláusula WHERE:

```
arojas=> update socios set cp = '99999'where cp = ' ';
UPDATE 7
```

El resultado de la ejecución de esta sentencia es la notificación de que se han actualizado 7 filas, aquellas que no tenían un CP, algo que podemos comprobar volviendo a usar la misma consulta original:

```
arojas=> select apellido_paterno,apellido_materno,direccion,cp from socios;
```

apellido_paterno	apellido_materno	direccion	cp
Arias	Trera	Betania # 7	23001
Moreno	Pardo	Juan Rincon # 2	23008
Charte	Luque	Las Flores # 12	23021
Charte	Ojeda	Las Flores # 12	23021
Cid	Luque	Juan Carlos I # 23	23008
Charte	Luque	Pitagoras 1139	3100
Lopez	Aguilera	Desconocida	99999
Lopez	Hilera	Desconocida	99999
Garcia	Aguilera	Desconocida	99999
Garcia	Hilera	Desconocida	99999
Perez	Aguilera	Desconocida	99999
Perez	Leon	Desconocida	99999
Charte	Luque	Las Flores 12	99999

(13 filas)

Lógicamente, esta versión de la sentencia UPDATE, en la que se seleccionan varias filas por modificar, puede combinarse con la asignación de valores a más de una columna.

Valores nulos y por default

Además de valores constantes y expresiones, en la cláusula SET de una sentencia UPDATE también puedes recurrir a las palabras clave NULL y DEFAULT para introducir en una columna el valor nulo y su valor por default, respectivamente.

Recuerda que el valor NULL representa la ausencia de contenido, lo cual no es equivalente a una cadena de caracteres vacía, el valor 0 o cualquier otro valor. También debe tenerse en cuenta que el intento de asignar un valor nulo a una determinada columna podría provocar un error, en caso de que en la definición de la tabla se indicase que esa columna no puede quedar vacía.
Es lo que ocurriría si, por ejemplo, intentamos cambiar la signatura de un libro asignándole el valor NULL.

arojas=> update libros set signatura = null where codigo = 12;

ERROR: el valor null para la columna Â«signaturaÂ» viola la restricciÃ³n not null

El uso de DEFAULT tendrá sentido únicamente en caso de que, durante la definición de la estructura de la tabla, se hubiese establecido un valor por default para las columnas. No es el caso de nuestra base de datos de ejemplo.

Eliminacion de filas (Delete)

Las sentencias SELECT y UPDATE son capaces de recuperar y modificar columnas individuales de una fila, en contraposición a sentencias como INSERT y DELETE que actúan sobre las filas completas, añadiéndolas o eliminándolas, respectivamente.

Para eliminar una fila, por tanto, no es necesario facilitar el nombre de columna alguna, como se aprecia en la sintaxis de la sentencia DELETE:

```
DELETE FROM [ONLY] tabla
WHERE condicion
```

Nuevamente y a pesar de que el estándar indica que es opcional, la cláusula WHERE resulta imprescindible a la hora de ejecutar una operación de eliminación de filas ya que, de lo contrario, estaríamos borrando todo el contenido actual de la tabla.

La tabla en sí, su estructura, permanecería, de tal forma que sería posible añadir nuevas filas, pero todas las que hubiese en el momento de ejecutar una sentencia como DELETE FROM socios se perderían.

Los dos procedimientos básicos de eliminación de filas son los siguientes:

- Borrado de una fila concreta, utilizando en la cláusula WHERE el nombre de la columna que actúa como clave primaria y el valor que seleccionará esa fila de manera única.
- Eliminación de varias filas, usando para ello un filtro de selección adecuado.

Al utilizar la sentencia UPDATE para modificar varias filas es recomendable probar antes el criterio de selección, con una consulta que permita ver dicho conjunto de filas, en el momento de eliminar información esa comprobación es indispensable, puesto que nos permitirá subsanar cualquier error antes de que sea demasiado tarde. Primero emite una sentencia SELECT con la condición adecuada en la cláusula WHERE y solamente si las filas obtenidas son las que quieres borrar, a continuación usa la misma condición con la sentencia DELETE.

```
SELECT nombre, apellido_paterno, apellido_materno, fecha_alta
FROM socios
WHERE EXTRACT (DAY FROM fecha_alta) = 30;
```

arojas=> select nombre,apellido_paterno,apellido_materno,alta_socios from socios where extract (day from alta_socios) = 30;

nombre	apellido_paterno	apellido_materno	alta_socios
Juan	Lopez	Aguilera	2005-06-30
Juan	Lopez	Hilera	2005-06-30
Luis	Garcia	Aguilera	2005-06-30
Luis	Garcia	Hilera	2005-06-30
Rosa	Perez	Aguilera	2005-06-30
Rosa	Perez	Leon	2005-06-30

(6 filas)

El resultado es el que nos interesa. Recuerda que la función EXTRACT, empleada aquí, tiene versiones diferentes según el RDBMS sobre el que se trabaje.

Procederemos a borrar las filas usando la misma cláusula WHERE de la consulta previa:

arojas=> delete from socios where extract (day from alta_socios) = 30;
DELETE 6

Al igual que UPDATE, la sentencia DELETE se limita a efectuar su trabajo sin generar resultado alguno aparte de indicar el número de filas afectadas por la operación. Será procedente, por lo tanto consultar de nuevo la tabla para comprobar si su contenido es el que debe ser:

arojas=> select nombre,apellido_paterno,apellido_materno,alta_socios from socios;

nombre	apellido_paterno	apellido_materno	alta_socios
Belen	Arias	Trera	2003-02-10
Antonio	Moreno	Pardo	2003-10-28
Alejandro	Charte	Luque	2002-04-10
Francisco	Charte	Ojeda	2005-06-09
Manuel	Cid	Luque	2007-06-09
David	Charte	Luque	2007-07-25
David	Charte	Luque	2005-06-29

(7 filas)

Actualización de datos y transacciones

Al igual que la inserción de datos, la modificación y eliminación son operaciones que se verán afectadas por las transacciones en curso que pudieran existir en el RDBMS. Previamente se apuntó, por ejemplo, que al trabajar cualquier cambio que efectuemos se perderá si no terminamos ejecutando la sentencia COMMIT.

Ante operaciones potencialmente peligrosas, como lo son la sustitución de unos datos por otros o la eliminación de datos, su inclusión en el contexto de una transacción puede ahorrarnos un serio disgusto. Aunque es éste un tema de SQL, puesto que ya conocemos las sentencias COMMIT y ROLLBACK únicamente necesitamos saber cómo iniciar una transacción a petición para proteger de fallas las actualizaciones y borrado.

Tras iniciar una transacción, cualquier operación que efectuemos no será realmente confirmada hasta que no usemos la sentencia COMMIT. Esto significa que si de forma accidental, por un error, modificamos o eliminamos las filas inadecuadas, siempre podemos usar ROLLBACK para cancelar las operaciones que forman parte de la transacción, volviendo a dejar la base de datos en el estado original.

DELETE FROM prestamos;
Introduce el punto y coma al final accidentalmente, antes de haber escrito la cláusula WHERE y la condición de selección, provocando que todas las filas de la tabla préstamos se pierdan. Nada habría ocurrido si, con anterioridad, hubiese iniciado una transacción. Observe la secuencia de sentencias ejecutadas a continuación y los resultados que generan:

COMMIT;

VALIDACION TERMINADA.

DELETE FROM prestamos;

3 filas suprimidas.

SELECT * FROM prestamos;

Ninguna fila seleccionada.

ROLLBACK;

Rollback terminado.

SELECT * FROM prestamos;

Nota: Dependiendo de la versión del RDBMS así como del tipo de tablas que estés utilizando, las modificaciones y borrado pueden aplicarse directamente a las tablas no teniendo ningún efecto sentencias tales como START TRANSACTION y ROLLBACK. **Consulte la documentación específica de su producto antes de**

probar a eliminar todas las filas de una tabla, a fin de evitar sorpresas como en la que el contenido de la tabla libros se ha perdido irremediablemente.

```
arojas=> begin;
BEGIN
arojas=> delete from libros where codigo = 17;
DELETE 1
arojas=> rollback;
ROLLBACK
arojas=> select * from libros;
```

codigo	signatura	titulo	autor	disponible
2	I MAS mis	El misterio del perro secuestrado	Masters, M	S
3	I LI, sec	El secreto de los pirats	Li	S
4	I DIE, mia	Mi amigo agapito	Diez Barrio, German	N
5	I FAR, unt	Un cesto lleno de lapices	Farias, Juan	S
6	T CHA, sse	Fabulas mitologicas	Charte, Francisco	S
7	T CHA, php	Leyendas de las calles de la ciudad de mexico	Dios, Juan de	N
8	T CHA, htm	Platero y yo	Ramon, Juan	S
9	G ESL, uni	En busca del unicornio	Eslava Galan, Juan	S
11	G BAL, cli	Cuentos libertinos	Balzak, H.	
12	G SHA inc	La incognita Newton	Shaw, Catherine	S
13	T CHA exc	Excel 2003	Charte, Francisco	S
14	T CLI sch	Curso de Linux	Schroder	S
15	A INF fca	Curso de Rdbms	Armando Rojas Marin	S
1	I PIL cap	El capitan calzoncillos	Pilkey, Dav	S
16	T CON inc	Principios de Contabilidad	Gonzalez Torres, Patricia	
10	G MUN, ven	Ventanas de Nueva York	Munoz Molina, Antonio	S
17	T CON fca	Contabilidad IV	Valles Faudoa,Myriam	S

(17 filas)

6.3. Consultas

El propósito de la instrucción SELECT consiste en extraer y visualizar datos de una o más tablas de la base de datos. Se trata de un comando extremadamente poderoso, capaz de realizar el equivalente de las operaciones de Selección, Proyección y Combinación del álgebra relacional en una única instrucción.

SELECT es el comando SQL más frecuentemente utilizado y tiene el siguiente formato general:

SELECT [DISTINCT | ALL] {* | [expresiónColumna [AS nuevonombre]] [,…]}

FROM NombreTabla [alias] [,…]

[WHERE condición**]**

[GROUP BY listaColumnas**] HAVING** condicion]

[ORDER BY listaColumnas**]**

La expresión Columna representa un nombre de columna o una expresión, NombreTabla es el nombre de una tabla o vista de base de datos ya existente y a la que se tenga acceso y alias es una abreviatura opcional para NombreTabla. La secuencia de procesamiento en una instrucción SELECT es:

FROM especifica la tabla o tablas que hay que usar

WHERE filtra las filas de acuerdo con alguna condición

GROUP BY forma grupos de filas que tengan el mismo valor de columna

HAVING filtra los grupos de acuerdo con alguna condición

SELECT especifica qué columnas deben aparecer en la salida

ORDER BY especifica el orden de la salida

El orden de las cláusulas en la instrucción SELECT no puede cambiarse. Las dos únicas cláusulas obligatorias son las dos primeras: SELECT y FROM; las restantes son opcionales.

Selección de filas (cláusula WHERE)

Los ejemplos anteriores muestran el uso de la instrucción SELECT para extraer todas las filas de una tabla.

Sin embargo, a menudo necesitamos restringir las filas que hay que extraer. Esto puede hacerse mediante la cláusula WHERE que está compuesta de la palabra clave WHERE seguida de una condición de búsqueda que especifica las filas que hay que extraer. Las cinco condiciones básicas de búsqueda (o predicados) en terminología ISO son las siguientes:

- Comparación. Compara el valor de una expresión con el valor de otra.
- Rango. Comprueba si el valor de una expresión cae dentro de un rango especificado de valores.
- Pertenencia a conjunto. Comprueba si el valor de una expresión coincide con uno de los valores de un cierto conjunto.
- Correspondencia de patrones. Comprueba si una cadena de caracteres se ajusta a un patrón especificado.
- Nulo. Comprueba si una columna tiene un valor nulo.

La cláusula WHERE es equivalente a la operación de selección de álgebra relacional.

Pueden generarse predicados más complejos utilizando los operadores lógicos AND, OR, NOT, con paréntesis (si se necesitan o desean) para mostrar el orden de evaluación. Las reglas para evaluar una expresión condicional son:

- las expresiones se evalúan de izquierda a derecha;
- primero se evalúan las subexpresiones contenidas entre corchetes;
- el operador NOT se evalúa antes que los operadores AND y OR:
- el operador AND se evalúa antes que el operador OR;

Se recomienda utilizar siempre paréntesis para eliminar cualquier posible ambigüedad.

Ordenar las filas

Al ejecutar una consulta sobre una tabla, en principio las filas obtenidas aparecen normalmente en el orden que establece su clave primaria o, en su defecto, el orden físico que ocupan y que suele coincidir con el que fueron añadidas a sus respectivas tablas. Podemos verlo claramente si recuperamos todas las columnas y filas de la tabla libros, tal y como se ha hecho en la figura 6.3. Observa la columna código y te darás cuenta de que está en orden creciente, es decir, las filas aparecen ordenadas según el código secuencial que actúa como clave primaria de la tabla.

arojas=> SELECT * from libros;

codigo	signatura	titulo	autor	disponible
1	I PIL cap	El capitan calzoncillos	Pilkey, Dav	S
2	I MAS mis	El misterio del perro secuestrado	Masters, M	S
3	I LI, sec	El secreto de los pirats	Li	S
4	I DIE, mia	Mi amigo agapito	Diez Barrio, German	N
5	I FAR, unt	Un cesto lleno de lapices	Farias, Juan	S
6	T CHA, sse	Fabulas mitologicas	Charte, Francisco	S
7	T CHA, php	Leyendas de las calles de la ciudad de Mexico	Dios, Juan de	N
8	T CHA, htm	Platero y yo	Ramon, Juan	S
9	G ESL, uni	En busca del unicornio	Eslava Galan, Juan	S
10	G MUN, ven	Ventanas de Nueva York	Munoz Molina, Antonio	S
11	G BAL, cli	Cuentos libertinos	Balzak, H.	
12	G SHA inc	La incognita Newton	Shaw, Catherine	S
13	T CHA exc	Excel 2003	Charte, Francisco	S
14	T CLI sch	Curso de Linux	Schroder	S
15	A INF fca	Curso de Rdbms	Armando Rojas Marin	S
16	T CON inc	Principios de Contabilidad	Gonzalez Torres, Patricia	
17	T CON fca	Contabilidad IV	Valles Faudoa,Myriam	S

(17 filas)

Figura 6.3. Orden por default de las filas e la tabla libros

En una consulta así sería muy normal que prefiriésemos tener las filas ordenadas por el título del libro, de tal manera que podamos localizar fácilmente por orden alfabético, o quizá por el nombre del autor, o también por ambas columnas.

La cláusula ORDER BY

ORDER BY es otra más de las cláusulas que pueden incluirse en una sentencia SELECT, debiendo aparecer siempre al final. La sintaxis para su uso es la siguiente:

SELECT columnas
FROM tablas
ORDER BY col1 [ASC | DESC] [, col2 …]

En caso de que existan cláusulas como WHERE, JOIN o GROUP BY, la cláusula ORDER BY la dejaríamos siempre para el final. Como puede apreciarse, hay que indicar el nombre de una o más columnas por las que queremos ordenar y opcionalmente, especificar si ese orden debe ser ascendente (ASC) o descendente (DESC).

Bastaría, por tanto, con añadir la cláusula ORDER BY título a la consulta del ejemplo anterior para obtener (vea la figura 6.4.) la lista de libros ordenada por título. En este caso, al ser título una columna que contiene una secuencia de caracteres, el orden es alfabético. De elegir una columna con contenido numérico, como es código, se utilizaría un orden numérico. Análogamente, en una columna que admitiera fechas el orden sería cronológico.

arojas=> SELECT * from libros order by titulo;

codigo	signatura	titulo	autor	disponible
17	T CON fca	Contabilidad IV	Valles Faudoa,Myriam	S
11	G BAL, cli	Cuentos libertinos	Balzak, H.	
14	T CLI sch	Curso de Linux	Schroder	S
15	A INF fca	Curso de Rdbms	Armando Rojas Marin	S
1	I PIL cap	El capitan calzoncillos	Pilkey, Dav	S
2	I MAS mis	El misterio del perro secuestrado	Masters, M	S
3	I LI, sec	El secreto de los pirats	Li	S
9	G ESL, uni	En busca del unicornio	Eslava Galan, Juan	S
13	T CHA exc	Excel 2003	Charte, Francisco	S
6	T CHA, sse	Fabulas mitologicas	Charte, Francisco	S
12	G SHA inc	La incognita Newton	Shaw, Catherine	S
7	T CHA, php	Leyendas de las calles de la ciudad de mexico	Dios, Juan de	N
4	I DIE, mia	Mi amigo agapito	Diez Barrio, German	N
8	T CHA, htm	Platero y yo	Ramon, Juan	S
16	T CON inc	Principios de Contabilidad	Gonzalez Torres, Patricia	
5	I FAR, unt	Un cesto lleno de lapices	Farias, Juan	S
10	G MUN, ven	Ventanas de Nueva York	Munoz Molina, Antonio	S

(17 filas)

Figura 6.4. La lista de libros ordenada alfabéticamente por título

La ordenación de columnas que contienen secuencias de caracteres vendrá determinada por la configuración del propio RDBMS, que podría distinguir o no entre mayúsculas y minúsculas y establecer la posición de caracteres como las letras acentuadas o la eñe dependiendo del conjunto de caracteres que se elija como activo.

Recurre a la documentación específica del producto que estés utilizando para saber cómo puedes efectuar esa configuración.

Nota: La cláusula ORDER BY puede ir seguida de la palabra COLLATE, cuya finalidad es elegir un cierto esquema de ordenación contemplado por el RDBMS en lugar de usar la configuración por default.

Orden ascendente y descendente

Por default, salvo que se indique lo contrario, el orden de las filas siempre será ascendente, es decir, de menor a mayor. Tras cada una de las columnas que aparezcan en la cláusula ORDER BY es posible indicar explícitamente el tipo de ordenación que se desea: ASC o DESC. La primera es la usada por default, mientras que la segunda efectuaría la ordenación en sentido inverso, de mayor a menor.

En la figura 6.5 se puede ver la misma consulta de la figura 6.4, añadiendo DESC tras la columna título, se puede observar en el resultado que los títulos aparecen en orden alfabético descendente.

arojas=> SELECT * from libros order by titulo desc;

codigo	signatura	título	autor	disponible
10	G MUN, ven	Ventanas de Nueva York	Munoz Molina, Antonio	S
5	I FAR, unt	Un cesto lleno de lapices	Farias, Juan	S
16	T CON inc	Principios de Contabilidad	Gonzalez Torres, Patricia	
8	T CHA, htm	Platero y yo	Ramon, Juan	S
4	I DIE, mia	Mi amigo agapito	Diez Barrio, German	N
7	T CHA, php	Leyendas de las calles de la ciudad de mexico	Dios, Juan de	N
12	G SHA inc	La incognita Newton	Shaw, Catherine	S
6	T CHA, sse	Fabulas mitologicas	Charte, Francisco	S
13	T CHA exc	Excel 2003	Charte, Francisco	S
9	G ESL, uni	En busca del unicornio	Eslava Galan, Juan	S
3	I LI, sec	El secreto de los pirats	Li	S
2	I MAS mis	El misterio del perro secuestrado	Masters, M	S
1	I PIL cap	El capitan calzoncillos	Pilkey, Dav	S
15	A INF fca	Curso de Rdbms	Armando Rojas Marin	S
14	T CLI sch	Curso de Linux	Schroder	S
11	G BAL, cli	Cuentos libertinos	Balzak, H.	
17	T CON fca	Contabilidad IV	Valles Faudoa,Myriam	S

(17 filas)

Figura 6.5. La lista de libros en orden alfabético inverso por la columna título

Si tras ORDER BY disponemos el nombre de varias columnas separadas por comas, cada una de ellas puede llevar asociado uno de los modificadores ASC | DESC. De no aparecer, se entenderá siempre que el orden es ascendente.

Ordenar por varias columnas

En caso de que varias de las filas obtenidas en el resultado de la consulta contengan el mismo valor en la columna elegida para ordenarlas, algo que ocurriría si, por ejemplo, ordenamos los datos de la figura 6.5 por la columna autor, el orden de esas filas en concreto no está determinado. Es en estas situaciones donde tiene sentido de disponer dos o más columnas tras la cláusula ORDER BY, como se hace en la siguiente consulta:

```
SELECT *
FROM libros
ORDER BY autor, titulo;
```

Sabemos que hay varias filas que contienen el mismo nombre en la columna autor, que es la que determinará el orden principal. Esas filas en concreto, que pertenecen al mismo autor pero contienen distintos títulos, se ordenarán ascendentemente por la columna título.

Como se ha indicado en el punto anterior, podemos modificar el orden por default de cada columna disponiendo tras ellas los modificadores ASC y DESC, como se aprecia en la figura 6.6. En la parte superior tenemos la consulta ordenada por las dos columnas en el sentido por default, mientras que en la parte inferior se opta por ordenar ascendentemente por nombre de autor y a la inversa por título de libro.

Si no existen duplicados en los valores de la primera columna de ordenación, las demás no tienen utilidad alguna.

```
arojas=> SELECT * from libros order by autor,titulo;
```

codigo	signatura	titulo	autor	disponible
15	A INF fca	Curso de Rdbms	Armando Rojas Marin	S
11	G BAL, cli	Cuentos libertinos	Balzak, H.	
13	T CHA exc	Excel 2003	Charte, Francisco	S
6	T CHA, sse	Fabulas mitologicas	Charte, Francisco	S
4	I DIE, mia	Mi amigo agapito	Diez Barrio, German	N
7	T CHA, php	Leyendas de las calles de la ciudad de mexico	Dios, Juan de	N
9	G ESL, uni	En busca del unicornio	Eslava Galan, Juan	S
5	I FAR, unt	Un cesto lleno de lapices	Farias, Juan	S
16	T CON inc	Principios de Contabilidad	Gonzalez Torres, Patricia	
3	I LI, sec	El secreto de los pirats	Li	S
2	I MAS mis	El misterio del perro secuestrado	Masters, M	S
10	G MUN, ven	Ventanas de Nueva York	Munoz Molina, Antonio	S
1	I PIL cap	El capitan calzoncillos	Pilkey, Dav	S
8	T CHA, htm	Platero y yo	Ramon, Juan	S
14	T CLI sch	Curso de Linux	Schroder	S
12	G SHA inc	La incognita Newton	Shaw, Catherine	S
17	T CON fca	Contabilidad IV	Valles Faudoa,Myriam	S

(17 filas)

```
arojas=> SELECT * from libros order by autor asc, titulo desc;
```

codigo	signatura	titulo	autor	disponible
15	A INF fca	Curso de Rdbms	Armando Rojas Marin	S
11	G BAL, cli	Cuentos libertinos	Balzak, H.	
6	T CHA, sse	Fabulas mitologicas	Charte, Francisco	S
13	T CHA exc	Excel 2003	Charte, Francisco	S
4	I DIE, mia	Mi amigo agapito	Diez Barrio, German	N
7	T CHA, php	Leyendas de las calles de la ciudad de mexico	Dios, Juan de	N
9	G ESL, uni	En busca del unicornio	Eslava Galan, Juan	S
5	I FAR, unt	Un cesto lleno de lapices	Farias, Juan	S
16	T CON inc	Principios de Contabilidad	Gonzalez Torres, Patricia	
3	I LI, sec	El secreto de los pirats	Li	S
2	I MAS mis	El misterio del perro secuestrado	Masters, M	S
10	G MUN, ven	Ventanas de Nueva York	Munoz Molina, Antonio	S
1	I PIL cap	El capitan calzoncillos	Pilkey, Dav	S
8	T CHA, htm	Platero y yo	Ramon, Juan	S
14	T CLI sch	Curso de Linux	Schroder	S
12	G SHA inc	La incognita Newton	Shaw, Catherine	S
17	T CON fca	Contabilidad IV	Valles Faudoa,Myriam	S

(17 filas)

Figura 6.6. Filas ordenadas por dos columnas en diferentes sentidos

La cláusula GROUP BY

Para agrupar una serie de filas en subconjuntos, según el contenido de una o más de sus columnas o bien de una expresión obtenida a partir de ellas, emplearemos la cláusula GROUP BY.

Esta aparecerá tras SELECT, FROM y WHERE, es decir, partimos de una consulta que extraería una serie de filas y después usamos GROUP BY para dividirlas en grupos.

Tras GROUP BY, al igual que después de ORDER BY, dispondremos el nombre de la columna o columnas por las que se agrupará, o bien la expresión o cálculo que servirá como base de la agrupación. A diferencia de ORDER BY, sin embargo, GROUP BY no admite que se utilicen los alias de columnas ni tampoco los indicadores de posición relativa.

Una de las reglas fundamentales a la hora de usar GROUP BY es el hecho de que tras SELECT no pueden aparecer más columnas individuales que aquellas que aparezcan también en el criterio de agrupación, con la única excepción de las funciones de agregado. Es decir, no podemos hacer lo siguiente:

```
SELECT titulo, autor
FROM libros
GROUP BY autor
```

Si agrupamos las filas según su contenido de la columna autor, quedando una única fila por cada autor distinto, dicha fila no puede contener también el título, puesto que podrían existir varios asociados a un mismo autor y no es posible determinar cuál de ellos habría de tomarse. Desde este punto de vista, una sentencia como ésta:

```
SELECT autor
FROM libros
GROUP BY autor
```

Sería equivalente a utilizar:

```
SELECT DISTINCT autor
FROM libros
```

Obteniendo únicamente las filas en las que el nombre del autor no está repetido.

Funciones de agregación

La cláusula GROUP BY no encuentra su mayor sentido hasta que no conozcamos las funciones de agregación, aquellas que nos permiten operar sobre las columnas de cada grupo de filas para efectuar algún tipo de totalización. En la tabla 6.4 se indica el nombre y finalidad de cada una de estas funciones.

Función	Operación que efectúa
COUNT	Cuenta el número de filas que contienen un valor no nulo en la columna indicada y lo devuelve como resultado.
AVG	Halla el valor medio de los valores existentes en la columna indicada y lo devuelve como resultado.
MIN	Devuelve el valor mínimo existente en la columna indicada.
MAX	Devuelve el valor máximo existente en la columna indicada.
SUM	Calcula la suma de los valores contenidos en la columna indicada y lo devuelve como resultado.

Tabla 6.4. Funciones de agregación de SQL

Nota: La mayoría de los RDBMS cuentan, además de las explicadas aquí, con otras funciones de agregación específicas que facilitan la obtención de datos a partir de cálculos más complejos. Consulte la documentación específica de su base de datos para saber cuáles puede usar.

Todas estas funciones deben ir seguidas de unos paréntesis entre los cuales se indicará el nombre de una columna, sobre la que se llevará a cabo la operación descrita. Por ejemplo:

```
SELECT autor, count(titulo) as NumTitulos
```

```
FROM libros
GROUP BY autor;

arojas=> select autor,count(título) as NumTítulos from libros group by autor;

          autor          numtitulos
 Li                      1
 Gonzalez Torres, Patricia 1
 Ramon, Juan             1
 Schroder                1
 Valles Faudoa,Myriam    1
 Balzak, H.              1
 Dios, Juan de           1
 Munoz Molina, Antonio   1
 Armando Rojas Marin     1
 Eslava Galan, Juan      1
 Diez Barrio, German     1
 Pilkey, Dav             1
 Masters, M              1
 Charte, Francisco       2
 Shaw, Catherine         1
 Farias, Juan            1
(16 filas)
```

En este caso se han tomado las filas de la tabla libros y se han agrupado por la columna autor, lo cual significa hacer tantos grupos distintos como autores existan, incluyendo en cada uno todas las filas que coincidan en esa columna. A continuación se ha contado el número de filas de cada grupo que contiene un valor en la columna título, obteniendo como resultado el nombre de cada autor y el número de libros que le corresponden.

Mientras que la función COUNT puede aplicarse a cualquier columna, ya que se limita a contar el número de filas que tienen un valor en esa columna, SUM, MIN, MAX y AVG no tienen aplicación sobre ciertos tipos de datos. No es posible, por ejemplo, sumar los valores de una columna que contiene un título, ni tampoco hallar el valor máximo, mínimo o medio.

La consulta siguiente, por ejemplo, emplea las funciones MAX y MIN para saber cuál es el mayor y menor código asociado al grupo de libros disponibles o no disponibles, respectivamente.

arojas=> select disponible,count(disponible) as count_disponible, min(codigo) as min_codigo, max(codigo)as max_codigo from libros group by disponible;

disponible	count_disponible	min_codigo	max_codigo
	0	16	16
S	13	1	17
N	2	4	7
	1	11	11

(4 filas)

Observa la primera fila, correspondiente al libro que tenía el valor NULL en su columna disponible.

Existe una fila, correspondiente al libro de código 16, que no tiene valor en disponible y por ello, la expresión COUNT(disponible) devuelve el valor 0, dado que COUNT cuenta únicamente las filas en las que hay un valor.

Para obtener el valor correcto no tienes más que cambiar COUNT (disponible) por COUNT (*), indicando así que quieres contar las filas que hay en cada grupo, sin

examinar contenido de columna alguna. Esto es aplicable solamente al operador COUNT y no al resto de funciones de agregado.

arojas=> select disponible,count(*) as count_disponible, min(codigo) as min_codigo, max(codigo)as max_codigo from libros group by disponible;

disponible	count_disponible	min_codigo	max_codigo
	1	16	16
S	13	1	17
N	2	4	7
	1	11	11

(4 filas)

Uso de las funciones de agregado sin GROUP BY

La cláusula GROUP BY siempre va asociada con el uso de alguna función de agregado, pero esto no es siempre cierto a la inversa, es decir, las funciones de agregado enumeradas en la tabla 6.4 pueden emplearse en otros contextos, por ejemplo, para obtener un resumen de todas las filas obtenidas a partir de una consulta sin grupos, ya sea parcial, si existe una cláusula WHERE, o total en ausencia de ésta.

Como ejemplo, imagina que quieres saber cuántos socios hay en la biblioteca, cuál es la fecha del más antiguo y el más reciente. En este caso no hay grupos, solamente tres datos concretos.

Para obtenerlos utilizaremos entonces la siguiente sentencia:

```
arojas=> select count(*) as NumSocios,
arojas-> min(alta_socios) as "Mas Antiguo",
arojas-> max(alta_socios) as "Mas Reciente"
```

```
arojas-> from socios;

 numsocios   Mas Antiguo   Mas Reciente
          7  2002-04-10    2007-07-25
(1 fila)
```

Lo que obtenemos es una fila con resultados elaborados a partir del contenido de la tabla completa, en lugar de la información de una fila y columna concretas, lo cual demuestra la gran flexibilidad del lenguaje SQL.

En una consulta como la anterior, que emplea distintas funciones de resumen en la cláusula SELECT, no pueden aparecer columnas individuales por la razón que ya se explicó anteriormente al tratar los grupos. Sí podemos, sin embargo, utilizar la sentencia en la que se resumen los datos como una subconsulta de otra.

Es lo que se hace en el ejemplo siguiente para obtener los apellidos y el nombre del socio más antiguo de la biblioteca:

```
arojas=> select apellido_paterno, apellido_materno, nombre
arojas-> from socios
arojas-> where alta_socios = (select min(alta_socios) from socios);

 apellido_paterno   apellido_materno    nombre
  Charte             Luque              Alejandro
(1 fila)
```

De manera similar podrían obtenerse datos mucho más elaborados, especialmente si las tablas de nuestra base de datos contienen columnas con cantidades numéricas. En una hipotética base de datos con información sobre vendedores de una gran superficie y su nivel de ventas, por ejemplo, podría utilizarse la función

AVG en una subconsulta para obtener la media de ventas y usar este dato como filtro de una consulta principal en la que aparecen aquellos que están por encima o por debajo de esa media.

Filtrado de las filas agrupadas

En los ejemplos anteriores hemos estado agrupando y resumiendo los datos de todas las filas de una tabla, obteniendo como resultado un conjunto de filas con datos de cada grupo. Mediante la cláusula WHERE, que hemos utilizado anteriormente, podemos filtrar las filas de la consulta original antes de que éstas lleguen al proceso de agrupación o resumen. Esto significa que la cláusula WHERE actúa antes que GROUP BY y las funciones de agregado.

Recuperando uno de los ejemplos anteriores, en el que obteníamos el número total de títulos por categoría, podríamos añadir una cláusula WHERE (destacada en negrita a continuación) para saber cuántos títulos hay disponibles en cada categoría, en lugar del número total.

```
arojas=> select case substr(signatura,1,1)
arojas-> when 'G' then 'General'
arojas-> when 'I' then 'Infantil'
arojas-> when 'T' then 'Tecnico'
arojas-> when 'A' then 'Fca'
arojas-> end as categoria,
arojas-> count(substr(signatura,1,1)) as "Num Titulos" from libros where disponible
      = 'S'
arojas-> group by substr(signatura,1,1);
```

categoria	Num Titulos
Infantil	4
General	3
Fca	1
Tecnico	5

(4 filas)

A veces puede resultar interesante filtrar, no las filas de origen, aquellas que van a intervenir en la agrupación o resumen, sino las resultantes de la cláusula GROUP BY.

Supón que en la consulta anterior quieres obtener únicamente la categoría y número de títulos de aquellas categorías que tengan más de un cierto número disponible.

No puede agregarse una nueva cláusula WHERE detrás de GRIOUP BY:

```
FROM libros
WHERE disponible = ‘S’
GROUP BY SUBSTR (signature,1,1)
WHERE COUNT (SUBSTR(signature,1,1)) > 1
```

Esto provocaría un error, pero en realidad no tenemos más que sustituir el último WHERE por la palabra HAVING, como se indica a continuación, para obtener el resultado que perseguimos:

```
arojas=> select case substr(signatura,1,1)
arojas-> when 'G' then 'General'
arojas-> when 'I' then 'Infantil'
arojas-> when 'T' then 'Tecnico'
```

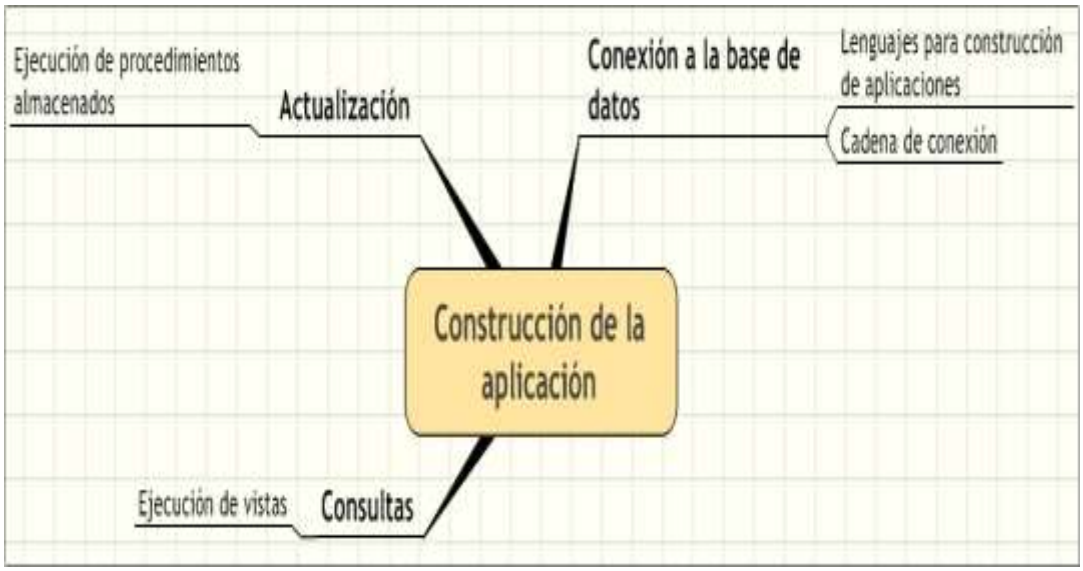
```
arojas-> when 'A' then 'Fca'
arojas-> end as categoria,
arojas-> count(substr(signatura,1,1)) as "Num Titulos" from libros where disponible
      = 'S'
arojas-> group by substr(signatura,1,1)
arojas-> having count(substr(signatura,1,1))>1;

 categoria        Num Titulos
  Infantil        4
  General         3
  Tecnico         5
 (3 filas
```

La cláusula HAVING es a GROUP BY lo que WHERE a SELECT, es decir, filtra las filas de acuerdo con un determinado criterio, pero no las filas de origen sino las resultantes de la agrupación.

Nota: Detrás de todas las cláusulas utilizadas en el último de los ejemplos, quizá el más complejo propuesto hasta el momento, puede añadirse ORDER BY en caso de que interese ordenar los resultados según un cierto criterio, por ejemplo según el número de libros disponibles.

RESUMEN


Ejecución de procedimientos almacenados
Actualización
Conexión a la base de datos
Lenguajes para construcción de aplicaciones
Cadena de conexión
Construcción de la aplicación
Ejecución de vistas
Consultas

.

BIBLIOGRAFÍA

SUGERIDA

DATE, C. J. 2001. *Sistemas de Bases de Datos*. 7ª, México: Pearson.

ELMASRI, RAMEZ. 2002. *Fundamentos de sistemas de bases de datos*. México: Pearson Educación, Addison-Wesley.

WORSLEY C. Y JOSHUA D.DRAKE. 2002. *Practical PostgreSQL*. Sebastopol, CA: O'Reilly. (Disponible en http://www.faqs.org/docs/ppbook/book1.htm).

SILBERSCHATZ, A., H. KORTH Y S. SUDARSHAN. 2006. *Fundamentos de bases de datos*. 5ª, Madrid, España: McGraw-Hill.

POSTGRESQL DOCUMENTATION, en http://www.postgresql.org/docs/.

GESCHWINDE, EWALD Y SCHÖNIG, HANS-JÜRGEN. 2002. *PHP and PostgreSQL. Advanced Web Programming*. Indianapolis, Ind.: Sams.

POSTGRESQL JDBC DRIVER, en http://jdbc.postgresql.org/.

Facultad de Contaduría y Administración
Sistema Universidad Abierta y Educación a Distancia