ORGANIZACION

El módulo consta de una serie de unidades en las que se especifican los objetivos, contenidos y actividades.

Las actividades no son de obligada realización, su resolución servirá al alumnado para la adquisición de los conocimientos necesarios que le permitan superar el módulo, además de constituir un instrumento de evaluación.

La materia se distribuye en dos trimestres:

En el primero se verá lo relativo a bases de datos, en este caso se trabajará con el S.G.B.D. de Oracle. En esta parte se desarrollan los contenidos sobre fundamentos de las bases de datos en general, arquitectura de Oracle, lenguajes SQL y PL/SQL

1. Introducción a las Bases de Datos. Modelo de Datos
2. Organización de un SGBD relacional.
3. SQL y el modelo relacional. LMD
4. SQL Lenguajes LDD y LCD.
5. PL/SQL fundamentos.

En el segundo trimestre se desarrollan los contenidos necesarios para poder realizar una aplicación Web. Se verá la última parte de PL/SQL, XHTML y J2EE

6. PL/SQL programación avanzada
7. Programación WEB - XHTML
8. Programación Web – J2EE fundamentos
9. Programación Web – JSP aplicaciones
10. Programación Web – Acceso a Datos

LA EVALUACIÓN.

Criterios de evaluación específicos.

- Describir las características de las bases de datos.
- Diferenciar las estructuras de ficheros tradicionales y las estructuras basadas en tecnología de bases de datos.
- Conocer los aspectos fundamentales de la arquitectura cliente-servidor.
- Describir las características de los modelos de bases de datos en red, jerárquico y relacional.
- Describir las características estáticas del modelo relacional.
- Identificar, en el modelo relacional, los problemas asociados a la representación de información desconocida o no aplicable.
- Describir los lenguajes relacionales.
- Utilizar correctamente operaciones del álgebra relacional.
- Identificar las características que definen un SGBD como relacional.
- Describir los elementos que componen la estructuras físicas y lógicas de una base de datos.
- Relacionar la estructura de una base de datos con los tres niveles de estructuración ANSI.
- Identificar los mecanismos que optimizan los procesos de acceso a los datos.
- Identificar los principales procesos y estructuras de memoria de un SGBD Oracle.
- Explicar la estructura del diccionario de datos de Oracle.
- Recuperar información conforme a las especificaciones recibidas, incluyendo: agrupamientos, subconsultas y diferentes funciones, contemplándose al menos las de conversión.
- Crear tablas y modificar su estructura, utilizando restricciones, y estableciendo relaciones.
- Crear vistas de acuerdo al diseño que se establezca, realizando actualizaciones de las tablas a través de las vistas, en aquellos casos en los que sea posible.
- Modificar algún privilegio sobre algunos objetos, realizar las pruebas necesarias y presentar las conclusiones.

- Crear bloques anónimos que realicen procesos de mantenimiento sobre la base de datos, controlando las posibles excepciones que se puedan producir.
- Crear subprogramas almacenados que realicen funciones de rutinas estándar.
- Realizar operaciones de definición de datos y de obtención de resultados utilizando los paquetes estándar de Oracle.
- Diseñar disparadores que cumplan unas determinadas especificaciones.
- Construir páginas estáticas sencillas utilizando los diferentes elementos del lenguaje XHTML
- Diseño y utilización de formularios
- Identificar los distintos elementos que constituyen el lenguaje JSP
- Conocer los métodos de conexión y acceso a una base de datos
- Saber crear clases para la manipulación de los datos
- Conocer las distintas formas de acceso y recuperación de la información
- Conocer el proceso de las transacciones
- Saber realizar procedimientos almacenados

Cuándo se hace.

1. Durante el curso se realizará una prueba objetiva coincidiendo con el final de cada trimestre, no es obligatoria pero si se supera es eliminatoria. La prueba consistirá en una batería de supuestos teórico/prácticos con un grado de dificultad adaptado a los contenidos y actividades propuestas en cada una de las unidades.

2. A finales del segundo trimestre se realizará la prueba final. Es obligatoria para todos los alumnos/as que tengan algún trimestre pendiente o que no se hayan presentado. De esta prueba se obtendrá la calificación final del módulo.

Instrumentos.

1- Durante el curso se hará un seguimiento del proceso de aprendizaje del alumnado a través de las tutorías y de las actividades.

2- Resultado de las pruebas objetivas.

Criterios de calificación

La calificación se realizará en base al seguimiento realizado durante el curso y a las pruebas objetivas trimestrales.

METODOLOGÍA.

Las unidades se publicarán en Educastur Campus**.** Para aquellos alumnos/as que tengan dificultades de acceso, se les proporcionará la información en CDs.

En cada unidad se especifican:

- Los objetivos que se pretenden conseguir.
- Los contenidos desarrollados, en algunos casos con indicaciones o enlaces donde se puede ampliar la información.
- Las tareas o actividades, propuestas.
- Cuestionario para la autoevaluación en aquellas unidades que lo requieran

El alumnado podrá realizar el seguimiento de sus actividades a través de:

Tutorías presenciales:

Colectivas: Dirigidas a todos los alumnos/as del grupo, sin que su asistencia sea obligatoria. Se realizará una al mes, y en ella se tratarán aquellos contenidos que, a juicio del profesor, ofrezcan una especial dificultad, o necesiten un tratamiento en aula.
Individuales: A demanda del alumno/a, se atenderán, dentro del horario semanal establecido para ello.

Tutorías telefónicas:

A través de la línea telefónica, el alumno podrá realizar, en las horas establecidas para ello, cuantas consultas considere necesarias, para poder avanzar en el conocimiento de cada uno de los módulos.

E_mail:

A cada alumno/a del curso se les ha proporcionado una cuenta de correo electrónico, mediante la cual podrá realizar aquellas consultas que considere oportunas y dirigidas a los correos de cada módulo profesional. Los profesores contestarán en un periodo razonable de tiempo, siempre a través de las cuentas de correo puestas a disposición por la Consejería de Educación y Ciencia.

MINIMOS EXIGIBLES

Se establecen teniendo en cuenta que se seleccionan los básicos o prioritarios, y garantizan que el alumnado está en posesión de las capacidades que han de saberse ejercitar en relación con ellos. Es decir, estos mínimos deben alcanzarse para superar el módulo.

- Saber cuales son las ventajas de las bases de datos respecto a los ficheros clásicos.
- Definir las claves primarias y ajenas
- Saber realizar ejercicios de álgebra relacional
- Saber crear tablas y vistas, modificar su estructura utilizando restricciones y estableciendo relaciones
- Saber seleccionar información de una tabla según las condiciones que se establezcan
- Realizar consultas sobre varias tablas.
- Realizar actualizaciones de tablas.
- Crear bloques anónimos que realicen procesos de mantenimiento sobre la base de datos,controlando las posibles excepciones que se puedan producir.

- Crear subprogramas almacenados que realicen funciones de rutinas estándar.
- Realizar operaciones de definición de datos y de obtención de resultados utilizando los paquetes estándar de Oracle.
- Diseñar disparadores que cumplan unas determinadas especificaciones.
- Diseñar y crear páginas estáticas que incluyan formularios y enlaces
- Programar páginas dinámicas sin acceso a bases de datos
- Crear páginas dinámicas con acceso a bases de datos

MATERIALES Y RECURSOS

Para poder seguir el módulo se debe disponer de ordenador con acceso a Internet con banda ancha y descargar e instalar el software de libre distribución JDK, TOMCAT, ECLIPSE y ORACLE 10G.

BIBLIOGRAFÍA Y OTROS RECURSOS

No se van recomendar libros concretos dado que hoy en día los contenidos del módulo se encuentran en multitud de libros y manuales así como en Internet. Las principales editoriales tienen libros específicos del módulo. También se pueden consultar la bibliografía de C.J. Date, Adoración de Miguel y Mario Piatini para las unidades una y dos. La de Juan Manuel y Rodrigo Fernández para las seis primeras unidades, junto con los manuales de oracle para SQL y PL/SQ. Para las unidades de programación web consultar las páginas de Sun Microsystems, Eclipse, Apache y cualquier libro reciente de Java que incluya algún capitulo de JSP.

Introducción

En esta unidad se presenta una visión general de los sistemas de bases de datos y se desarrollan los conceptos fundamentales del modelo relacional.

Se empieza describiendo las características de los sistemas de base de datos y analizando sus ventajas con respecto a los sistemas tradicionales basados en ficheros. Se estudian las funciones de un Sistema Gestor de Bases de Datos (SGBD), teniendo en cuenta tanto la arquitectura a tres niveles, propuesta por ANSI/X3/SPARC, en que se estructuran los SGBD modernos, como las necesidades de los distintos tipos de usuarios del sistema. También, se presenta la arquitectura cliente/servidor y sus distintas formas de implementación en los sistemas actuales. Se trata el concepto de modelo de datos y se describen las características de los modelos convencionales en que se apoyan los sistemas de bases de datos dominantes en el mercado en las últimas décadas, es decir, los modelos CODASYL, jerárquico y relacional. Finalmente, se desarrolla con cierto detalle la estructura y restricciones del modelo relacional, así como los elementos fundamentales del álgebra relacional.

Objetivos

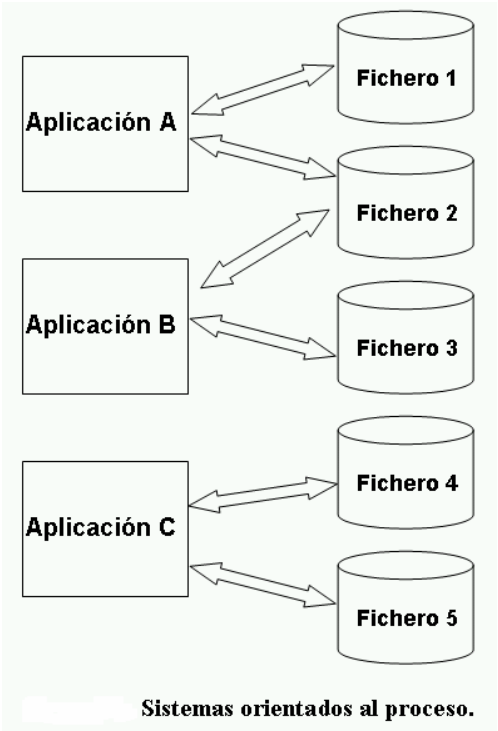
- Describir las características de las bases de datos.
- Diferenciar las estructuras de ficheros tradicionales y las estructuras de almacenamiento basadas en tecnologías de bases de datos.
- Conocer los aspectos fundamentales de la arquitectura cliente-servidor.
- Identificar las características de los modelos de bases de datos en red, jerárquico y relacional.
- Describir los elementos que componen la estructura del modelo relacional.
- Comprender el fundamento de las restricciones inherentes y de usuario.
- Utilizar el álgebra relacional para escribir expresiones que representen operaciones de recuperación o actualización de una base de datos relacional.

Contenidos Generales

1. CONCEPTO DE BASE DE DATOS. 3
2. EL SISTEMA GESTOR DE LA BASE DE DATOS. 4
3. DESCRIPCIÓN, MANIPULACIÓN Y TRANSFORMACIÓN DE LOS DATOS. 6
4. PROTECCIÓN DE DATOS. 7
5. EL DICCIONARIO DE LA BASE DE DATOS 13
6. ARQUITECTURAS DE APLICACIÓN. 14
7. TIPOS DE USUARIOS DE UN SISTEMA DE BASES DE DATOS. 17
8. MODELOS DE DATOS. 19
9. TIPOS DE MODELOS DE DATOS 21
10. MODELOS CONVENCIONALES. 22
11. CARACTERÍSTICAS GENERALES DEL MODELO DE DATOS RELACIONAL 25
12. ESTRUCTURA DEL MODELO. 26
13. RESTRICCIONES DEL MODELO. 29
14. EL TRATAMIENTO DE NULOS. 33
15. EL MODELO RELACIONAL EN LA ARQUITECTURA ANSI. 34
16. EL ÁLGEBRA RELACIONAL 35
17. OPERADORES TRADICIONALES DE CONJUNTOS. 36
18. OPERACIONES RELACIONALES ESPECIALES. 39
19. EXPRESIONES ALGEBRAICAS. 42
20. OPERADORES ADICIONALES 43

1. Concepto de base de datos.

Una base de datos es **"una colección no redundante de datos compartibles entre diferentes sistemas de aplicación"**. Esta definición que aparece en la obra "*Data Análisis for Database Design*" de David R. Howe permite un primer acercamiento al concepto de base de datos.



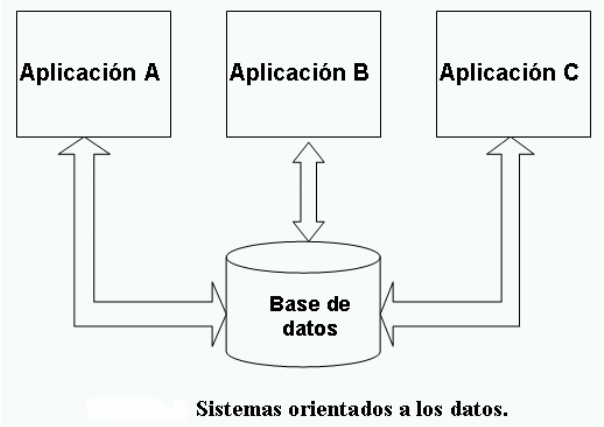
Sistemas orientados al proceso.

El almacenamiento de un conjunto de datos con el objetivo de que sirvan a distintas aplicaciones, evita su redundancia y todos los problemas asociados a la misma. Si fuera necesaria la repetición de datos por razones de eficiencia, esta redundancia será controlada por el sistema. Así, en caso de que una aplicación actualice un dato redundante, el sistema se encargará de propagar dicha actualización a todas sus ocurrencias. De esta forma, se consigue que los resultados obtenidos por las distintas aplicaciones sean coherentes, ya que no cabe la posibilidad de que utilicen distintos valores del mismo dato.

Los sistemas bases de datos se desarrollan en los años sesenta del pasado siglo como un enfoque alternativo a los sistemas basados en ficheros utilizados hasta entonces. En los sistemas informáticos de aquella época los datos se almacenan en ficheros diseñados específicamente para cada aplicación. Aunque varias aplicaciones manejen datos comunes, los ficheros en que se encuentran no suelen ser compartidos entre ellas, por lo que el mismo dato puede aparecer repetido en dos o más ficheros. Cuando una aplicación actualiza uno de estos datos redundante, lo hace sólo en su fichero, con lo que durante cierto tiempo pueden obtenerse distintos valores para el mismo dato dependiendo del fichero que se consulte. Problemas adicionales de la redundancia de información son la ocupación extra de memoria secundaria y el aumento del tiempo de proceso al tener que repetirse los mismos procedimientos de validación, actualización y almacenamiento de los datos en los distintos ficheros.

Los sistemas de ficheros son **sistemas orientados al proceso**, ya que en ellos los datos están subordinados a los procesos, cada aplicación dispone de sus propios ficheros de datos.

La descripción de estos datos y sus interrelaciones aparecen en los programas que hacen uso de ellos. Los cambios en la estructura de los datos o en su almacenamiento llevan a un rediseño de las aplicaciones que los utilizan.



Sistemas orientados a los datos.

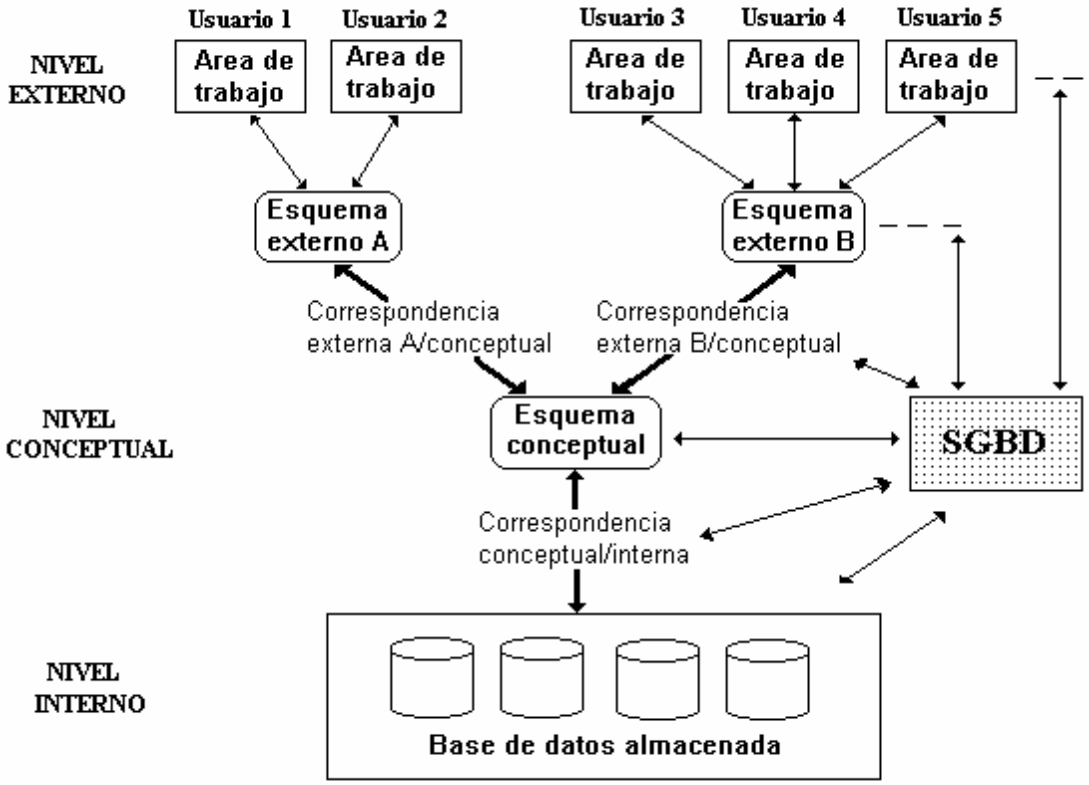
Las bases de datos son **sistemas orientados a los datos.** Éstos se almacenan independientemente de las aplicaciones. En la base de datos también se guardan la descripción de los datos y las interrelaciones que puedan existir entre ellos. Los datos se almacenan interrelacionados y estructurados de acuerdo con un determinado modelo de datos que permita recoger el máximo significado asociado a ellos. Los cambios en los datos, en su forma de almacenamiento o en los caminos de acceso a ellos no tienen por que llevar a una reprogramación de las aplicaciones.

2. El sistema gestor de la base de datos.

En un sistema de base de datos, los programas no acceden directamente a los datos, sino que es el software del sistema el encargado de proporcionar a las aplicaciones los datos que éstas demanden. Este software, conocido genéricamente como **sistema gestor de la base de datos (SGBD)**, no sólo tendrá a su cargo las funciones de recuperación y manipulación de los datos, sino que también deberá permitir su definición, e igualmente deberá implementar otras funciones encaminadas a mantener la integridad, seguridad y confidencialidad de los datos.

El sistema gestor de la base de datos puede ser definido como **un conjunto de programas que permiten que los usuarios describan, recuperen y manipulen eficazmente los datos almacenados en la base de datos, protegiendo dichos datos contra todas aquellas acciones, intencionadas o no, que los puedan corromper**.

Las arquitecturas de bases de datos han evolucionado mucho desde sus comienzos, aunque la considerada estándar hoy en día es la propuesta en 1975 por el comité ANSI/X3/SPARC (Standard Planning and Requirements Committee of the American National Standards Institute on Computers and Information Processing). Este comité propuso una arquitectura general para DBMSs (Database Management Systems) basada en tres niveles o esquemas: el nivel físico, o de máquina, el nivel externo, o de usuario, y el nivel conceptual. Así



Arquitectura de un SGBD a tres niveles

mismo describió las interacciones entre estos tres niveles y todos los elementos que conforman cada uno de ellos.

El **nivel externo** o **esquema externo** es la visión que tiene de la base de datos un usuario en particular y en él estarán solo reflejados aquellos datos e interrelaciones que necesite ese usuario. También se incluirán las restricciones de uso de esos datos y, en algunas implementaciones, los caminos de acceso a esos datos. Existirán distintos esquemas externos de la misma base de datos, ya que distintos usuarios y aplicaciones tienen distintas percepciones de ella. Un mismo esquema externo puede ser compartido por varios usuarios.

El **nivel conceptual** o **esquema conceptual** corresponde a la visión global de todos los datos de la base. En este nivel se incluirá la descripción de todos los datos e interrelaciones, así como las restricciones de integridad de los datos y las restricciones de acceso a ellos. El esquema conceptual representa la visión integrada de todos los usuarios.

El **nivel interno** o **esquema interno** corresponde a la estructura de almacenamiento que sostiene los datos. Es muy dependiente de la implementación concreta del SGBD. En el esquema interno se especifican aspectos relativos al almacenamiento de los datos en memoria secundaria y acceso a los mismos.

El SGBD es el encargado de posibilitar la definición y manipulación de la base de datos a estos tres niveles y de gestionar las interfaces entre niveles y con el usuario.

3. Descripción, manipulación y transformación de los datos.

El SGBD debe permitir la descripción de los distintos esquemas de la base de datos, así como que los usuarios de la base recuperen, inserten, modifiquen o eliminen datos de ella. También, se debe encargar de realizar la correspondencia o transformación (**mapping**, *en inglés*) entre los formatos de los datos correspondientes a los distintos esquemas vistos en el punto anterior.

El administrador de la base de datos es el encargado de especificar los elementos de datos que la forman, su estructura, interrelaciones, restricciones, etc. Es su responsabilidad la definición tanto del esquema conceptual como del interno como de los distintos esquemas externos, aunque a veces, puede dejar que los programadores realicen la descripción de los esquemas externos implicados en las aplicaciones que implementan.

Al hablar de administrador de la base de datos hay que entenderlo más como una función que como una persona concreta. La función de administración puede ser realizada por una sola persona o por un grupo de ellas.

El administrador especificará los distintos esquemas mediante un **lenguaje de definición de datos** (LDD, o DDL en inglés). Cada tipo de SGBD dispondrá de uno o más lenguajes LDD específicos. En unos casos existe un único lenguaje, con sentencias que permiten describir los elementos de los distintos esquemas, y en otros hay un lenguaje distinto para cada nivel de abstracción.

Las operaciones a realizar sobre la información contenida en la base se especifican mediante un **lenguaje de manipulación de datos** (LMD, o DML en inglés). Este lenguaje de

manipulación puede estar formado por un conjunto de sentencias (**lenguaje huésped**) que son admitidas dentro de un programa escrito en otro lenguaje (**lenguaje anfitrión**) de propósito general. En los SGBD actuales suele existir también un LMD **autocontenido,** que no necesita ir embebido en otro para especificar cualquier tarea de recuperación o actualización de información.

El SGBD llevará a cabo las operaciones de manipulación de datos cumpliendo siempre las restricciones de seguridad especificadas por el administrador.

En un SGBD que siga la propuesta ANSI/X3/SPARC, existirán dos niveles de transformación:

- la **transformación conceptual-interna** que permite hacer pasar los datos desde su representación conceptual a su representación interna. y viceversa, y
- **la transformación externa-conceptual** que permite hacer pasar los datos desde su representación en el nivel conceptual a la correspondiente a los distintos esquemas externos, y viceversa.

Para poder efectuar automáticamente la transformación de datos de un nivel a otro, el SGBD debe conocer las correspondencias existentes entre niveles que estarán almacenadas junto con los esquemas en el diccionario de la base de datos.

4. Protección de datos.

El SGBD debe disponer de mecanismos que eviten que la base de datos quede en un estado inconsistente a causa de fallos que den lugar a la alteración y corrupción de los datos. Estos fallos pueden ser fallos físicos (de memoria principal, de memoria secundaria, etc.), fallos lógicos (de programa, del sistema operativo, del SGBD, etc.) y fallos humanos, intencionados o no. También, el SGBD dispondrá de facilidades para que el administrador proteja los datos contra accesos no autorizados, manteniendo su privacidad.

Al tratar la protección de los datos de una base, hay que referirse a los conceptos de **seguridad**, **integridad** y **confidencialidad**.

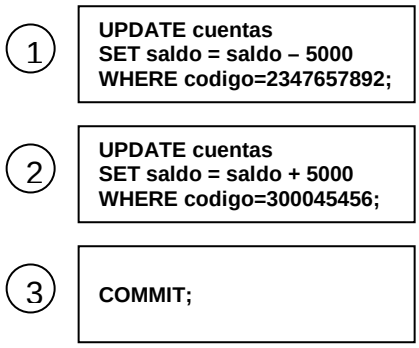
❑ Seguridad

La seguridad (o **recuperación**) de la base de datos se refiere a su protección contra fallos lógicos o físicos que destruyan los datos total o parcialmente.

El origen de estos fallos puede ser múltiple, y además de los instrumentos de protección facilitados por el SGBD, se emplearán otros ajenos a él como: sistemas contraincendios, sistemas tolerantes a fallos, sistemas de control de acceso físico a los equipos, etc.

El objetivo de la seguridad, cuando se produce cualquiera de estos fallos, es mantener la consistencia de la base de datos.

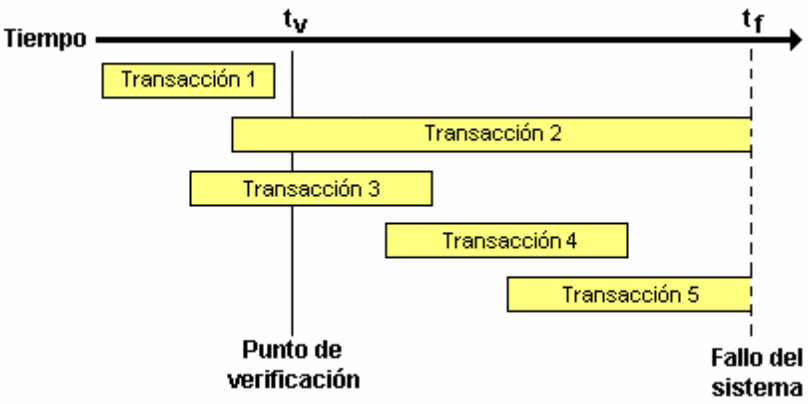
Veamos un ejemplo en que al producirse un fallo la base de datos queda en un estado inconsistente. Se supone un proceso en que se está realizando una transferencia de fondos entre dos cuentas bancarias, concretamente, se transfieren 5.000 euros de la cuenta **2347657892** a la cuenta **300045456**. En la primera operación se restan los 5.000 euros del saldo de la primera cuenta, y en la segunda operación se suma la misma cantidad al saldo de la segunda cuenta. Si se produjese un fallo a consecuencia del cual sólo se realizase la primera de las dos operaciones, la base de datos quedaría en un estado inconsistente: habría 5.000 euros perdidos, que no está ni en una cuenta ni en otra.



Transferencia bancaria.

La implementación por parte del SGBD del concepto de transacción hace que este tipo de problemas nunca se produzcan. Una **transacción es una secuencia de operaciones que han de ejecutarse de forma atómica,** es decir, o se realizan todas las operaciones que componen la transacción o ninguna. De esta forma, en el ejemplo, la transacción sería la operación de transferencia, que estaría compuesta de las dos operaciones de modificación de saldo. Si durante la segunda operación de modificación se produjera un fallo que impidiera su terminación, el SGBD automáticamente desharía la primera y así la base de datos quedaría en un estado consistente. La orden **COMMIT** de la figura, comunicaría al SGBD el fin de la transacción.

Un fallo que afecta a la seguridad de la base de datos es aquél que provoca la pérdida de memoria volátil (por ejemplo, fallo en el suministro eléctrico). Al producirse esta situación, pueden quedar transacciones sin finalizar y otras, que habiendo concluido satisfactoriamente, se encontraban en áreas de memoria intermedia y aun no se habían grabado en disco. La recuperación de estos fallos, conocida como **recuperación en caliente**, implica deshacer las operaciones correspondientes a transacciones inconclusas y repetir las transacciones finalizadas y no grabadas. Esta recuperación la realizará automáticamente el SGBD, y para ello el método más habitual es apoyarse en uno o más ficheros diarios.



Situación de varias transacciones en el instante de producirse un fallo de sistema.

Un fichero **diario** o **log** es un fichero en el que el SGBD va grabando toda la información necesaria para deshacer o rehacer una transacción. El SGBD ejecuta periódicamente la operación conocida como **punto de verificación** o **punto de recuperación** (**checkpoint**), consistente en grabar en memoria secundaria el contenido de las áreas de memoria intermedia y registrar tanto en el fichero diario como en un fichero de rearranque dicho punto de verificación. Cuando el SGBD debe realizar una recuperación de la base de datos después de una caída del sistema, o sea, cuando lleva a cabo una recuperación en caliente, obtiene la dirección del registro de recuperación más reciente del fichero de rearranque y recorre el fichero ***log*** desde ese punto hasta el final para obtener la información de qué transacciones debe deshacer o rehacer.

En la figura se ven varias situaciones de transacciones ante un supuesto de fallo que causa la pérdida de memoria volátil. La transacción 1 no es afectada por el fallo ya que finalizó con anterioridad al último punto de verificación y, por tanto, cuando se produce la pérdida de memoria estaba grabada en disco. Las transacciones 3 y 4 finalizan antes del fallo pero no hay seguridad de que estuviesen grabadas, pues finalizaron después del ***checkpoint***, por lo que deben ser rehechas. Finalmente, las transacciones 2 y 5 no habían concluido cuando se produjo el fallo y deben ser deshechas.

Para recuperarse de un fallo de memoria secundaria que afecte a la base de datos (**recuperación en frío**) debe utilizarse una copia de seguridad(***backup***) de la base de datos. La restauración del último ***backup*** realizado deja la base de datos recuperada a la fecha en que se hizo dicha copia de seguridad. Para poner la base de datos al día habrá que utilizar todos los ficheros diarios que se han generado desde esa fecha hasta la actualidad en orden cronológico.

- **Integridad**

La base de datos puede entrar también en un estado inconsistente a consecuencia de la realización de operaciones incorrectas. El SGBD debe detectar y corregir estas operaciones,

asegurando en todo momento la integridad de la base de datos, es decir, que los datos contenidos en la base sean correctos.

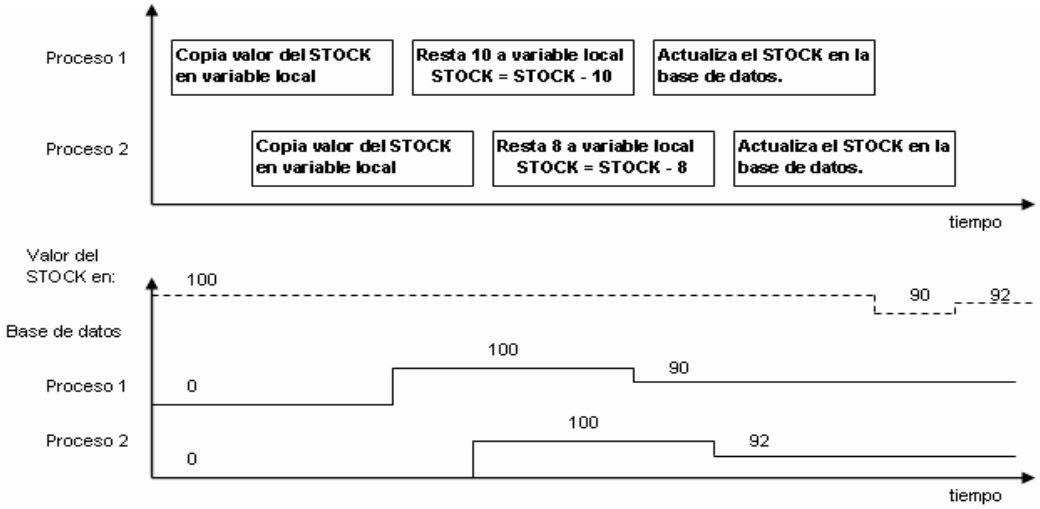
La integridad de los datos pude verse afectada por operaciones que atentan contra las restricciones de integridad y por las interferencias producidas entre accesos concurrentes.

Las restricciones de integridad son definidas en el momento del diseño de la base de datos y son una forma más de captar y almacenar el significado asociado a los datos. Por ejemplo, en la base de datos de una empresa de distribución una restricción de integridad puede ser que en la entidad LINEAS_PEDIDO nunca pueda existir una referencia a un artículo inexistente en la entidad ALMACEN. O dicho de otra forma, sólo pueden incluirse en un pedido artículos registrados en almacén.

El SGBD debe velar por que se cumplan las restricciones de integridad y, siguiendo el ejemplo, la operación de insertar una línea en LINEAS_PEDIDO con una referencia de artículo que no esté en ALMACEN será detectada como un intento de violar dicha restricción y, en consecuencia, será rechazada.

En general, una restricción de integridad irá acompañada de la especificación de la acción a realizar en caso de intento de violación de la misma (rechazo de la operación, en el ejemplo) y de la indicación del momento en que debe realizarse dicha acción.

En sistemas multiusuario el SGBD debe disponer de algún mecanismo de control de concurrencia para evitar las inconsistencias que se pueden producir en la base de datos por acceso concurrente. Un caso típico producido por accesos concurrentes es el conocido como **problema de la operación perdida,** que se ejemplifica en la figura.



El problema de la operación perdida

Una aplicación de gestión de pedidos está siendo ejecutada simultáneamente por dos usuarios, y en un instante dado ambos procesos intentan actualizar el stock en almacén del mismo artículo. Los distintos pasos de los dos procesos se desarrollan paralelamente en el tiempo como se indica en la figura. Al final, la base de datos queda en un estado inconsistente puesto que el stock en almacén del artículo en cuestión es de 92 unidades cuando en realidad debería ser de sólo 82 unidades. El problema reside en que la actualización del stock que realiza el proceso 2 hace que se pierda la operación de actualización previamente realizada por el proceso 1.

Existen distintas técnicas para evitar los problemas derivados de los accesos concurrentes. Una de estas técnicas consiste en establecer un bloqueo sobre los datos que se están modificando, de forma que, hasta que no finalice dicha modificación, ningún otro proceso podrá acceder a esos datos. Los bloqueos serán gestionados automáticamente por el SGBD, aunque es posible que se permita también a los usuarios establecer bloqueos explícitos.

En el ejemplo anterior, se establecería un bloqueo al iniciarse la transacción del proceso 1 y se levantaría al terminar ésta, de forma que la transacción del proceso 2 actuaría sobre un stock de 90 unidades y no de 100, como ocurría sin bloqueo.

- **Confidencialidad**

Este concepto alude a la protección de los datos contra el acceso de usuarios no autorizados. El SGBD debe proteger los datos de los intentos de acceso de los usuarios que no cuenten con la autorización adecuada, pero en el sistema informático existirán también otros mecanismos ajenos al propio SGBD con esta misma función, como controles del sistema operativo, controles de tipo físico de acceso a las instalaciones, etc.

La implementación del esquema de confidencialidad es responsabilidad del administrador de la base de datos, pero en su diseño intervendrán otras instancias de la empresa u organismo. La decisión sobre qué información es pública y a qué niveles, puede depender de aspectos legales, sociales y éticos, cuestiones de política de la empresa u organismo, etc.

Por lo que respecta exclusivamente al SGBD, existentes dos tipos de gestión de la confidencialidad conocidos como **control de acceso discrecional** y **control de acceso obligatorio**. En ambos casos, la unidad de datos a proteger varía desde una base de datos entera hasta un ítem de datos específico. La diferencia entre ambos tipos estriba en lo siguiente:

- En el caso del **control discrecional**, cada usuario tiene diferentes derechos de acceso (también conocidos por **privilegios** o **autorizaciones**) sobre diferentes

objetos. Normalmente, usuarios diferentes tendrán diferentes privilegios en el mismo objeto. Los esquemas discrecionales son muy flexibles.

- En el caso de **control obligatorio**, cada objeto es etiquetado con un cierto **nivel de clasificación** y a cada usuario se le da un **nivel de acreditación**. Un objeto sólo puede ser accedido por aquellos usuarios con un nivel de acreditación apropiado. Los esquemas obligatorios son comparativamente rígidos.

Las decisiones sobre a qué usuarios se les debe permitir realizar qué operaciones sobre qué objetos son decisiones políticas, no técnicas, por lo que quedan fuera de la jurisdicción del SGBD en sí; lo único que puede hacer el SGBD es obligar al cumplimiento de estas decisiones una vez tomadas. Para que el SGBD pueda llevar a cabo esas funciones adecuadamente ha de cumplirse que:

- El resultado de esas decisiones políticas se dé a conocer al sistema por medio de sentencias en un lenguaje de definición apropiado, y sean recordadas por el sistema registrándolas en forma de reglas o restricciones de **autorización**.
- Exista una forma de verificar una solicitud de acceso dada contra las restricciones de autorización aplicables. (Por "solicitud de acceso" entendemos aquí la combinación de operación solicitada más objeto solicitado más usuario solicitante.)

Para poder decidir qué restricciones son aplicables a una solicitud dada, el sistema debe ser capaz de reconocer el origen de dicha solicitud, es decir, debe ser capaz de reconocer al usuario específico del cual proviene una solicitud. Por esa razón, cuando los usuarios obtienen acceso al sistema casi siempre se les pide proporcionar no sólo su identificador de usuario, sino también una contraseña.

Además de estas técnicas, en algunos SGBD se utiliza la criptografía como medio de protección de la confidencialidad. Esta técnica permite transformar el contenido de la base, haciendo que sea necesario conocer la correspondiente clave de descifrado para hacer uso de la información almacenada.

Complementando la protección de la confidencialidad, los SGBD suelen proporcionar facilidades de auditoria que permiten registrar las operaciones realizadas por los usuarios, pudiendo así detectar accesos no permitidos y mejorar el sistema de protección de la confidencialidad.

5. El diccionario de la base de datos.

La estructura de un SGBD está articulada alrededor del diccionario de datos. En él estará almacenada la definición de los distintos esquemas de la base de datos y de las correspondencias entre ellos.

La información almacenada en el diccionario de datos es imprescindible para realizar las funciones de manipulación y transformación. Igualmente, en el diccionario de datos se registrarán las informaciones necesarias para implementar la función de protección de datos. Un diccionario completo incluirá también información del uso que las distintas aplicaciones y usuarios están haciendo de la base de datos, información del estilo de qué usuarios están conectados a la base de datos, qué aplicaciones hacen uso de qué datos, etc.

Al referirse al diccionario de datos se utilizan con frecuencia otros términos como **directorio de datos** o **catálogo.** Existe una cierta confusión en la terminología utilizada, por lo que con ánimo de aclararla vamos a describir distintos tipos de almacenes de datos.

Un **diccionario de datos** contiene información sobre los datos almacenados en la base de datos desde el punto de vista del usuario. En el diccionario estará almacenado lo que los usuarios necesitan para comprender el significado de los datos y poder manejarlos adecuadamente, como descripciones, significado, estructuras, consideraciones de seguridad, uso por las aplicaciones, etc.

Un **directorio de datos** contiene, en un formato legible para la máquina, las especificaciones necesarias para pasar de la representación externa de los datos a su representación interna. El directorio es un instrumento del SGBD, y esta orientado a facilitarle la información que necesita para su funcionamiento.

A veces, se habla de **diccionario/directorio de datos**, en referencia a paquetes de soporte lógico que incluyen ambas funciones.

Cuando el diccionario de datos está implementado sin ninguna interfaz directa con el directorio, se habla de **diccionario pasivo**. Por contra, cuando el diccionario y el directorio están integrados en un diccionario/directorio que sirve tanto a las necesidades de los usuarios como del SGBD, se habla de **diccionario activo**.

Una **enciclopedia o repositorio** es un diccionario que almacena la información manejada por una herramienta CASE (***C****omputer* ***A****ided* ***S****oftware* ***E****ngineering*). En el repositorio se almacena información textual y gráfica relativa a las distintas fases del ciclo de vida de un sistema. Las enciclopedias no suelen ser activas, o sea, no están integradas con el directorio de

datos, pero suelen proporcionar mecanismos que facilitan la tarea de generar las descripciones de los datos propias del directorio a partir de las descripciones lógicas de los datos, obtenidas en la fase de diseño.

Los SGBD relacionales almacenan información sobre los datos contenidos en la base de datos para ser utilizada tanto por el propio sistema como por el usuario. Este diccionario/directorio de datos que implementan los sistemas relacionales se conoce como **catálogo.** No obstante, la descripción lógica de los datos almacenada en un catálogo es pobre, incorpora poco significado. Un aspecto importante del catálogo es que su sistema de gestión no es, como en otros tipos de almacenes, un sistema de gestión propio e independiente del SGBD, sino que es el propio SGBD. O dicho de otra forma, al catálogo se accede con las mismas herramientas que al resto de la base de datos. Se dice que el catálogo es una **metabase**, es decir, una base de datos con datos sobre los datos (sobre su estructura) de la base de datos. A los datos del catálogo, también se les denomina **metadatos**.

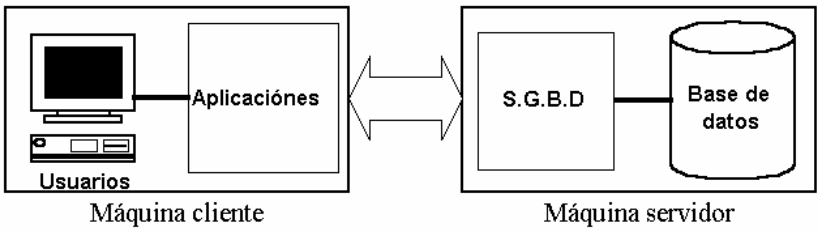
6. Arquitecturas de aplicación.

El objetivo último de un sistema de bases de datos es facilitar el desarrollo y ejecución de aplicaciones. Por tanto, desde un punto de vista amplio, un sistema de bases de datos posee una estructura compuesta de dos partes: un **servidor**, también denominado **backend**, y un conjunto de **clientes** o **frontends**.

Arquitectura cliente/servidor

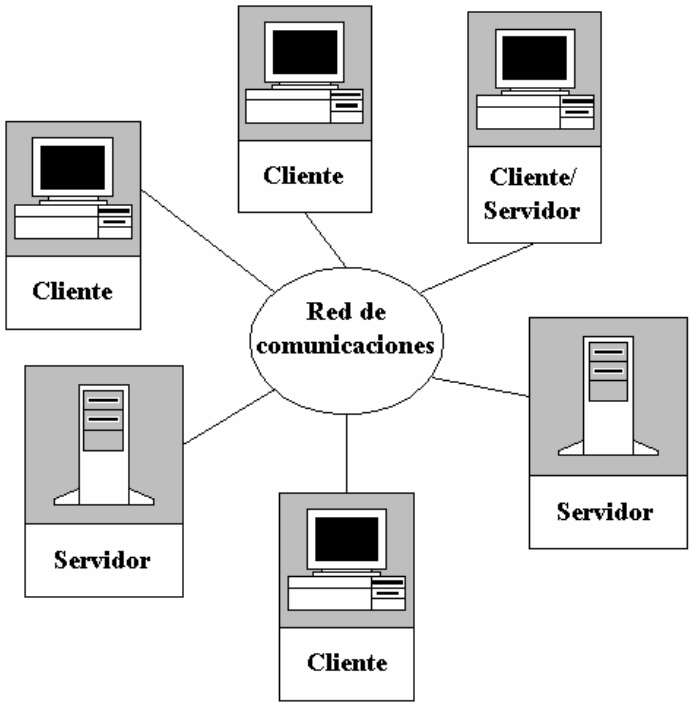
El servidor permite llevar a cabo todas las funciones propias del SGBD indicadas en los apartados anteriores. Realmente, el servidor es el SGBD mismo.

Los clientes son las distintas aplicaciones ejecutadas entorno al SGBD, tanto las aplicaciones de usuario como las distintas utilidades proporcionadas por el proveedor del SGBD o por otros proveedores. Todas ellas se comunican con el servidor a través de los correspondientes esquemas externos.



Sistema de B.D. con varias máquinas.

Estas dos partes en que se divide el sistema pueden ejecutarse en la misma máquina o en máquinas distintas interconectadas a través de algún sistema de comunicación. Esta segunda posibilidad, el servidor ejecutándose en una máquina y el cliente en otra, es a la que se suele aplicar el término cliente/servidor, hablándose de máquina cliente y máquina servidor o servidor de base de datos.



Sistema de B.D. con varias máquinas servidores y clientes

La arquitectura cliente/servidor nos lleva a lo que se conoce como **proceso distribuido**, esto es, al procesamiento de los datos distribuido sobre varias máquinas conectadas entre sí a través de una red de comunicaciones. El proceso distribuido exige la presencia de algún software que se encargue de gestionar las comunicaciones entre las distintas máquinas que participan en el proceso. Estos programas de comunicaciones harán de interfaz entre el software propio del sistema de bases de datos y el de gestión de la red.

También estaríamos ante un caso de proceso distribuido cuando una máquina con más de un procesador separa la ejecución de las aplicaciones y del S.G.B.D. en distintos procesadores. No obstante, esto no es lo más frecuente.

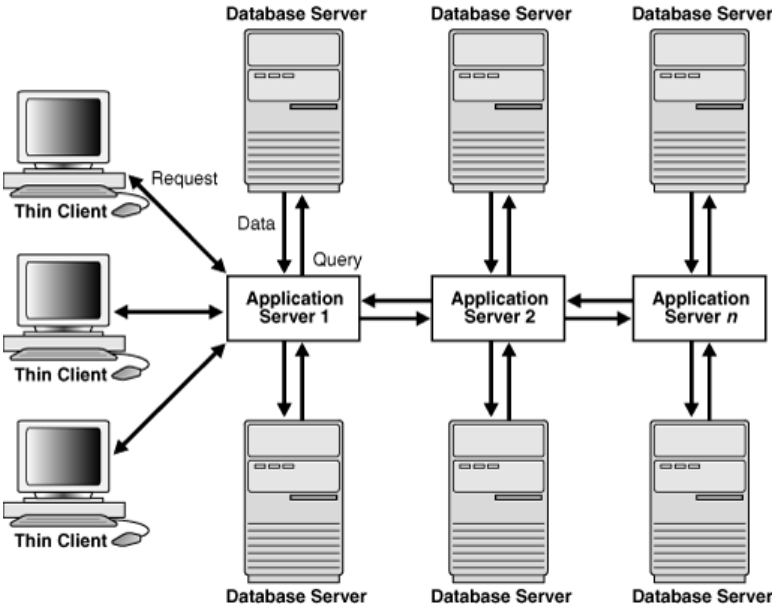
Existen distintas posibilidades de proceso distribuido según se repartan las partes cliente y servidor sobre las distintas máquinas. En la figura podemos ver un ejemplo en el que se conjugan las distintas posibilidades. Unidos por una red de comunicaciones, que en realidad podrían ser varias redes interconectadas, tenemos varios clientes y servidores. A cada servidor pueden tener acceso simultáneo varias máquinas clientes y también es posible que una máquina cliente pueda acceder a más de un servidor. Igualmente hemos representado un caso de máquina que actúa como cliente de un SGBD local, es decir, que ejecuta tanto la parte cliente como la parte servidor de un sistema de base de datos local a ella. Sería posible, que

esta misma máquina actuase como cliente de otros servidores o que gestionase el acceso a sus datos por parte de otros clientes.

El caso de una máquina cliente con acceso a más de un servidor puede presentarse en dos variantes:

- El cliente sólo puede conectarse a un servidor a un tiempo. No es posible en una sola consulta obtener datos de más de un servidor.
- El cliente puede acceder simultáneamente a varios servidores. En una sola consulta se pueden combinar datos disponibles en distintos servidores. Desde el punto de vista del cliente, desde una perspectiva lógica, es como si sólo existiera un único servidor. Este caso se conoce como **sistema de base de datos distribuida**.

En un sistema de base de datos distribuida, una aplicación debe ser capaz de acceder a los datos sin tener que saber que éstos pueden estar distribuidos en varias bases de datos, soportadas por diferentes sistemas gestores, que se ejecutan en diferentes máquinas, con diversos sistemas operativos y conectadas por diferentes redes. Se dice, que esta distribución de los datos es **transparente** para las aplicaciones.



*Arquitectura **multitier**.*

Otra arquitectura ampliamente utilizada en la actualidad es la conocida como **multinivel** o **multicapa (***Multitier Architecture*), que se utiliza en aplicaciones WEB, tanto en intranets como en Internet. En esta arquitectura la parte servidor se sigue ejecutando en el servidor de base de datos, pero la parte cliente se reparte en dos o más niveles: uno o varios servidores de aplicaciones y las máquinas clientes. La mayor parte del proceso de la aplicación se realiza de una forma centralizada en el servidor de aplicación, quedando para las máquinas clientes la parte de la aplicación que actúa de interfaz con el usuario.

7. Tipos de usuarios de un sistema de bases de datos.

Veremos a continuación los distintos tipos de de usuarios que interactúan con un sistema de bases de datos, relacionándolos con las facilidades que el sistema les proporciona para cubrir sus necesidades.

Usuarios finales.- Se trata de usuarios, con poco o ningún conocimiento informático, que acceden a la base de datos a través de:

- Aplicaciones diseñadas específicamente para atender sus necesidades. Por ejemplo, un cliente de una empresa de distribución tiene acceso a una aplicación Web que le permite realizar pedidos, imprimir facturas, ver el estado de sus pedidos, etc. Todas estas operaciones dan lugar a la inserción y manipulación de información en la base de datos de una forma totalmente transparente al usuario.
- Un lenguaje de manipulación de datos autocontenido, que permite al usuario realizar operaciones de recuperación y actualización de información de la base de datos. Por ejemplo, si el sistema de base de datos de la empresa de distribución está montado sobre un servidor Oracle, el personal de administración de la empresa podrá saber el número de pedidos pendientes de proceso escribiendo una consulta en lenguaje SQL desde el entorno SQL*Plus. El uso de lenguajes de manipulación exige el conocimiento de dicho lenguaje y del esquema externo al que se accede, pero facilita el acceso inmediato a la base de datos sin necesidad de desarrollar una aplicación específica.
- Utilidades que permiten realizar consultas y modificaciones a la base de datos sin necesidad de conocer un lenguaje de manipulación. Estas utilidades, basadas en menús y formularios, permiten que los usuarios satisfagan necesidades de información no previstas de antemano y para las que no existen aplicaciones desarrolladas.

Los lenguajes de manipulación son facilidades suministradas por el propio SGBD. Las utilidades pueden formar parte del SGBD o ser aplicaciones externas a él.

Programadores de aplicaciones.- Los programadores que desarrollan aplicaciones en entornos de base de datos pueden llevar a cabo su trabajo de distintas formas:

- Escribiendo sus programas en un lenguaje de tercera generación en el que se intercalan sentencias en un lenguaje de manipulación de datos. Las sentencias del lenguaje LMD, denominado **lenguaje huésped**, se encargarán de las labores de

acceso a la base de datos y las sentencias del lenguaje de tercera generación, o **lenguaje anfitrión**, se ocupan del resto de las tareas encomendadas al programa. La aplicación Web puesta a disposición de los usuarios del punto anterior podría estar desarrollada en lenguaje JAVA (lenguaje anfitrión) con sentencias en lenguaje SQL (lenguaje huésped) para acceder a la base de datos.

- Desarrollando sus aplicaciones por medio de generadores de pantallas, generadores de informes y lenguajes de cuarta generación. Los generadores de pantallas e informes permiten generar el código de la aplicación de una forma rápida a partir del diseño realizado por el programador utilizando las facilidades gráficas que la herramienta pone a su disposición. También pueden escribir código utilizando un lenguaje de cuarta generación (L4G). Los lenguajes L4G incluyen entre sus sentencias aquellas que manejan el acceso a los datos de la base. Por ejemplo, nuestra empresa de distribución podría disponer de una aplicación para llevar a cabo todo el proceso de gestión, cuyos módulos se hayan desarrollado utilizando el generador de formas y menús de la firma Oracle (*Oracle Forms*) y el generador de informes de informes y gráficos de la misma empresa (*Oracle Reports*). También se habrá escrito código utilizando el lenguaje Oracle PL/SQL.
- Empleando para escribir sus programas el lenguaje LMD autocontenido, sin apoyo de un lenguaje anfitrión. Esta es una posibilidad real pero poco usada por los programadores, dado que se suele obtener una mayor eficiencia utilizando el lenguaje LMD como huésped de otro lenguaje o utilizando un lenguaje L4G.

El administrador de la base de datos (DBA o DataBase Administrador).- Es el usuario, o usuarios, responsable del diseño, creación y mantenimiento de la base de datos.

Para facilitarle la tarea de diseño, el sistema puede poner a su disposición herramientas de ayuda, como las herramientas CASE (*Computer Aided Software Engineering*).

El SGBD pondrá a disposición del DBA varios lenguajes de definición de datos (lenguajes LDD) mediante los que especificará los distintos esquemas (externos, conceptual e interno) de la base de datos para su compilación y almacenamiento en el directorio de datos. Es normal que exista un lenguaje para cada nivel de abstracción, pero también es posible que se utilice el mismo para más de un nivel. Mediante estos lenguajes también se especificarán las restricciones de integridad y de confidencialidad.

Dentro del mantenimiento de la base de datos está incluida la modificación de su estructura, a cualquier nivel, cuando sea necesario. Cuando haya que añadir nuevos objetos o modificar los existentes, se utilizarán las sentencias adecuadas de los lenguajes LDD.

También, los SGBD suelen disponer de utilidades o programas de servicio diseñados para ayudar al DBA en la explotación del sistema. Estos programas realizan funciones de modificación del tamaño de los ficheros, obtención de estadísticas de utilización, carga de ficheros, obtención de distintos tipos de copias de seguridad de la base de datos, recuperación tras una caída del sistema, etc. Algunas de estas utilidades, aunque suministradas por el mismo proveedor del SGBD, puede que trabajen en el nivel externo del sistema, por lo que son en realidad programas de aplicación especial. Otras trabajan directamente en el nivel interno, formando parte realmente del SGBD.

El administrador de la base de datos de la empresa de distribución que venimos presentando como ejemplo, dispondrá de todas las sentencias SQL implementadas en el SGBD Oracle para definición y modificación de todo tipo de objetos, tanto lógicos como físicos. Utilizando las sentencias de este sistema, el DBA podrá definir o modificar objetos del nivel lógico de la base de datos, como tablas y vistas, establecer restricciones de integridad o controlar el acceso a los objetos de la base. También, podrá definir y modificar objetos del nivel físico, como *tablespaces* o *rollback segments*. Si dispone de la utilidad *Oracle Enterprise Manager*, podrá realizar muchas de las tareas de administración sin necesidad de escribir líneas SQL. Para facilitar la carga en la base de datos contenidos en ficheros tendrá a su disposición la utilidad *SQL*Loader*. Las operaciones de exportación e importación de información entre bases de datos Oracle se pueden llevar a cabo con las utilidades *Export* e *Import.*

8. Modelos de datos.

En una base de datos se representa una parte del mundo real con el objetivo de que dicha representación pueda ser utilizada por distintas aplicaciones actuales y futuras. Esta representación de la parcela del mundo real objeto de interés se hará de acuerdo con unas reglas, que varían de unas bases de datos a otras y que constituyen el modelo de datos.

El **universo del discurso** es la parte o aspecto del mundo real que el diseñador toma en consideración y que tras el adecuado diseño se plasmará en una base de datos. Por ejemplo, en una empresa pueden tomarse en consideración distintos universos del discurso como el de gestión comercial, el de gestión de personal o el de gestión de suministros. Todos tienen el

mismo mundo real de referencia, la empresa, pero cada uno contempla distintos aspecto del mismo.

El estudio y descripción del universo del discurso se lleva a cabo con el apoyo de un conjunto de conceptos, reglas y convenciones que reciben el nombre de **modelo de datos**.

Existen distintos modelos de datos, no todos implementados en sistemas de bases de datos. Todos ellos son una herramienta que facilita la interpretación y representación del universo del discurso. La representación del universo del discurso obtenida por aplicación de un modelo de datos será una estructura de datos denominada **esquema**.

Al aplicar una cierta sintaxis al modelo se obtiene un **lenguaje de datos**. Distintas sintaxis aplicadas al mismo modelo dan lugar a distintos lenguajes. Por ejemplo, el álgebra relacional y el cálculo relacional son dos lenguajes del mismo modelo de datos, el relacional, totalmente equivalentes, que sólo difieren en su sintaxis. El esquema estará descrito mediante un lenguaje de datos propio del modelo aplicado al universo del discurso.

En el universo del discurso es preciso distinguir entre las estructuras y los datos o valores que se almacenan en esas estructuras. Las primeras son propiedades estáticas o que varían poco con el tiempo, las segundas son propiedades que varían con el tiempo o dinámicas. Todo modelo de datos tendrá una parte estática que permitirá representar las estructuras y una parte dinámica que permitirá representar las propiedades dinámicas del universo del discurso. Asociado a la parte estática del modelo habrá un **lenguaje de definición de datos (LDD)** y asociado a la dinámica existirá un **lenguaje de manipulación de datos (LMD)**.

La estática del modelo constará de una serie de objetos con los que representar las estructuras del mundo real. Estos objetos varían de unos modelos a otros, varían sus nombres y varía su representación. Asociados a los objetos pueden existir restricciones que hagan que no sea posible representar, en ese modelo, ciertos objetos o asociaciones entre ellos. Se suele distinguir entre restricciones inherentes, que son las que vienen impuestas por la definición del modelo, y restricciones de usuario, que son las introducidas por el diseñador en el esquema como representación fiel del universo del discurso.

La representación de las propiedades estáticas del universo del discurso, obtenida aplicando la parte estática de un modelo de datos, constituye el esquema de dicho universo. El conjunto de los valores que toman los distintos objetos del esquema en un instante dado se denomina **ocurrencia del esquema** o base de datos. La ocurrencia del esquema varía en el tiempo, ya que, como consecuencia de operaciones de actualización sobre la base de datos, los objetos del esquema van tomando distintos valores a lo largo del tiempo.

La parte dinámica de un modelo de datos estará constituida por un conjunto de operaciones, definidas sobre los objetos de su parte estática, que permiten la transformación entre ocurrencias del esquema. Habrá operaciones que permiten localizar una ocurrencia o conjunto de ocurrencias de un objeto del esquema y otras que permitirán realizar una determinada acción (recuperación, modificación, inserción o borrado) sobre la ocurrencia u ocurrencias previamente localizadas. Las primeras se conocen genéricamente como **operaciones de selección** y las segundas como **operaciones de acción**.

9. Tipos de modelos de datos

De acuerdo con los niveles de abstracción de la arquitectura ANSI, se distinguen tres tipos de modelos:

- **Modelos externos**, que sirven para construir esquemas externos o de usuario.
- **Modelos conceptuales**, que permitirán la construcción de esquemas lógicos de todo el universo del discurso.
- **Modelos internos**, que se utilizarán para definir el esquema interno o físico de la base de datos.

Los dos primeros tipos de modelos se pueden considerar como **modelos lógicos**, ya que se ocupan de representar aspectos lógicos del universo del discurso omitiendo cualquier aspecto físico.

Algunos autores dividen los modelos lógicos en **conceptuales** y **convencionales**. No está clara la separación entre estos dos tipos de modelos. Los modelos convencionales son los implementados actualmente en sistemas gestores de bases de datos, como el modelo jerárquico, el modelo en red o el modelo relacional. Los modelos conceptuales poseen un mayor nivel de abstracción que los convencionales y como consecuencia una mayor flexibilidad y una mayor capacidad semántica. Ejemplos de modelos conceptuales serían el modelo de entidad/interrelación de Chen o el modelo Relacional/Tasmania (RM/T).

Los modelos convencionales sirven de conexión entre los esquemas externos y el interno, actuando como interfaz entre el informático y el sistema.

Dado el mayor nivel de abstracción de los modelos conceptuales, éstos sirven como interfaz entre el informático y el usuario final en las primeras fases del proceso de diseño de bases de datos. No obstante, dado que algunos modelos conceptuales, como el modelo E/R,

están implementados en herramientas CASE y empiezan a formar parte de algunos SGBD, también pueden actuar como conexión entre los esquemas externos e interno.

10. Modelos convencionales.

Son los que tienen implementación en SGBD reales, por lo que también podríamos referirnos a ellos como modelos de bases de datos. A continuación, veremos las características generales del modelo de datos jerárquico y del modelo de datos en red, que son el fundamento de distintos sistemas de bases de datos comerciales que dominaron el mercado en los años setenta y ochenta del siglo pasado. El modelo de datos relacional, en el que se apoyan los productos utilizados en la actualidad, será objeto de un amplio estudio en los puntos que siguen hasta el final de la unidad.

El modelo de datos en red representa las entidades del universo del discurso como nodos de un grafo y las interrelaciones entre ellas como los arcos que unen los nodos. No existe ninguna restricción ni en el tipo ni en la cantidad de los arcos y, por tanto, es posible representar estructuras de datos tan complejas como sea preciso. En este modelo se pueden representar directamente interrelaciones de cualquier cardinalidad y de cualquier grado: reflexivas, binarias, ternarias, etc.

La flexibilidad del modelo en red trae como consecuencia que su implementación sea difícil y poco eficiente, por lo que es preciso aplicarle ciertas restricciones a la hora de llevarlo a la práctica. Los modelos CODASYL y jerárquico pueden considerarse como derivados del modelo en red al establecer unas restricciones concretas.

❑ El modelo CODASYL.

El grupo DBTG (***Data Base Task Group***) del comité CODASYL (***COnference on DAta SYstem Languages***) presenta entre los años 1969 y 1981 distintos informes en los que se especifica un modelo de datos en red que seguirán con mayor o menor fidelidad distintos productos.

En el modelo CODASYL, los nodos son tipos de **registro** (*record*, en su nomenclatura) que están formados por **campos** (*data item*). Dentro de un registro pueden existir **agregados de datos** (*data aggregate*), que pueden ser grupos repetitivos o vectores. Los arcos o interrelaciones entre entidades reciben el nombre de **conjuntos** (*SET*).

El conjunto o SET de este modelo sólo permite representar interrelaciones **1:N** o **1:1**. Uno de los nodos del conjunto, el que participa con la cardinalidad uno, se denomina **propietario** (*owner*) y los demás son **miembros** (*members*). En un conjunto siempre existe un propietario y cero o más miembros. Los registros miembros de un conjunto pueden ser todos del mismo tipo o de tipos distintos. Un registro puede participar en más de un conjunto, y lo puede hacer con distinta categoría en cada uno, es decir, puede ser miembro en un conjunto y propietario en otro.

Como consecuencia de las restricciones inherentes, el modelo CODASYL no permite representar directamente interrelaciones reflexivas, ni interrelaciones N:1, ni tampoco de tipo N:M. Estas restricciones se salvan por el uso de ciertos mecanismos, como los registros de enlace o *links*, en los que no vamos a entrar. La limitación en la representación de las interrelaciones reflexivas fue eliminada en las especificaciones CODASYL de 1978.

El modelo presenta, además de los elementos lógicos ya citados, elementos de naturaleza física que el usuario debe definir y manejar.

Las especificaciones CODASYL plantean inicialmente una arquitectura de sistema gestor de base de datos a dos niveles (esquema y subesquemas), pasando posteriormente a tres niveles para mejorar la independencia físico/lógica.

Dispone de un **lenguaje de definición de datos del esquema** (*Schema DDL*) autocontenido que permite la definición de todos los elementos que forman parte de la base de datos.

De igual forma, CODASYL especifica un **lenguaje de definición de datos del subesquema** (*Subschema DDL*) huésped, inicialmente de COBOL y luego de otros lenguajes, que permite la descripción de subconjuntos del esquema para distintas aplicaciones que sólo necesitan trabajar sobre una parte de la base de datos.

En el informe de 1978 se introduce un nuevo nivel en la arquitectura del SGBD, **el esquema de almacenamiento**, con lo que se sacan del esquema muchos de los aspectos físicos de la base de datos. Para definición de este nuevo nivel aparece un nuevo lenguaje, el **lenguaje de definición del esquema de almacenamiento** (DSDL), que se va a ocupar de la definición de todos los aspectos físicos del almacenamiento de los datos.

El **lenguaje de manipulación de datos** (DML) definido por CODASYL implementa la parte dinámica del modelo. Este lenguaje, que es de tipo navegacional, puede ir embebido en distintos lenguajes de tercera generación (COBOL, C, FORTRAN, PL/1), y permite que el

usuario pueda navegar por la base de datos seleccionando el registro sobre el que se hará una determinada operación de borrado, modificación, recuperación, etc.

Dos implementaciones populares del modelo CODASYL lo constituyen los sistemas de bases de datos Total e IDMS.

El modelo jerárquico.

En este modelo de datos se apoyaron algunos de los primeros sistemas gestores de bases de datos comerciales. El modelo carece de estandarización, como el caso del CODASYL para el modelo en red, pero el sistema comercial jerárquico más importante, el IMS (***Information Management System***) de IBM, constituye su estándar de facto.

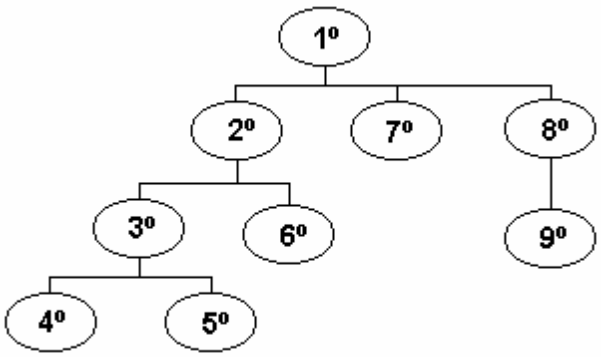
El esquema en el modelo jerárquico es una estructura en árbol formada por nodos, que representan entidades, y arcos, que representan las interrelaciones entre las entidades. Cada nodo será un registro con sus correspondientes campos, como en el modelo CODASYL.

Este modelo puede ser visto como un caso particular del modelo en red, al que se somete a una serie de restricciones inherentes para conseguir las características de árbol ordenado.

El modelo jerárquico presenta muchos problemas a la hora de ser utilizado para modelar el universo del discurso. Estos problemas vienen derivados fundamentalmente de la falta de capacidad de la estructura jerárquica para representar ciertas estructuras muy corrientes en la realidad, como las interrelaciones N:M, las interrelaciones reflexivas, las redes, etcétera. Esta falta de flexibilidad del modelo obliga a la introducción de redundancias cuando se quieren introducir en el esquema situaciones como las citadas.

La parte dinámica del modelo estará implementada en los sistemas de bases de datos jerárquicas por medio de un lenguaje de manipulación de datos huésped de un lenguaje de tercera generación. De estos lenguajes de manipulación, el más conocido es el DL/I del sistema IMS.

La manipulación de datos en el modelo jerárquico, igual que en todo modelo, necesita la selección previa de los datos sobre los que se realizará a continuación la acción de recuperación o actualización.



Orden de recorrido de un árbol en preorden

La operación de selección en el modelo jerárquico es de tipo navegacional, semejante a la del modelo CODASYL. El funcionamiento de algunos formatos de la operación de selección va unida al concepto de recorrido del árbol en un cierto orden, normalmente **preorden**. Recorrer un árbol o subárbol en preorden significa comenzar por la raíz, luego el subárbol izquierdo, y finalmente el derecho; cada subárbol se recorrerá también en preorden.

La recuperación en DL/I, que va asociada a la selección, consiste, como en cualquier otro lenguaje de datos, en llevar el registro marcado como activo a la correspondiente plantilla de entrada/salida. Cuando se inserta un nuevo registro en una estructura jerárquica, excepto para la raíz, queda conectado a un registro padre seleccionado mediante una sentencia de selección. Cuando se borra un registro de un árbol, se borran también todos sus hijos.

11. Características generales del modelo de datos relacional.

El modelo relacional es propuesto por Edgar Frank Codd en 1970 a través de un artículo (*"A Relational Model of Data for Large Shared Data Banks"*) publicado en el boletín de la ACM (*Communications of the Association for Computer Machinery*). Codd propone un modelo de datos basado en la teoría matemática de las relaciones, en el que los datos se estructuran en el ámbito lógico en forma de relaciones. En el artículo se concede gran importancia al hecho de que la estructura lógica de los datos sea independiente de su almacenamiento interno, cosa que los modelos de la época no recogían y que constituye el objetivo fundamental del modelo relacional.

El modelo relacional es un modelo lógico que adopta como estructura fundamental la **relación**. Una relación puede ser vista y tratada como una tabla de datos siempre que se tomen ciertas precauciones que más adelante mencionaremos. Todos los datos de una base de datos relacional se representan en forma de relaciones cuyo contenido varía con el tiempo.

El modelo relacional, como todo modelo de datos, tiene una parte estática constituida por los objetos permitidos y las restricciones, y una parte dinámica o conjunto de operaciones definidas sobre la estructura, que permite la transición entre estados de la base de datos. La parte dinámica del modelo está constituida por el **álgebra relacional** y el **cálculo relacional**.

Se apoya en un sólido fundamento matemático, lo que facilita su estudio y desarrollo y, además, resulta fácil de comprender y usar debido a que permite presentar la información en forma de tablas.

A partir de su definición original, el modelo ha ido evolucionando por aportación de muchos autores. Unas partes del modelo son más sólidas y se mantienen prácticamente sin cambios, mientras que otras son aún motivo de gran controversia entre distintos autores.

12. Estructura del modelo.

El elemento fundamental del modelo relacional, la **relación**, puede ser representada como una tabla de datos, de forma que cada fila de la tabla sería una **tupla** de la relación y cada columna representaría un **atributo** de la relación. El número de filas de la tabla constituye la **cardinalidad** de la relación y el número de columnas es el **grado** de la relación. Se puede decir que en una base de datos relacional el usuario percibe los datos estructurados en forma de tablas.

El valor que adopta en cada tupla un atributo es tomado de un conjunto de valores, conocido como **dominio,** relacionado con él. Varios atributos pueden tomar sus valores de un mismo dominio, pero cada atributo toma valores de un único dominio.

Tabla	Relación
Fila	Tupla
Columna	Atributo

Tablas y relaciones

Para que una tabla represente con fidelidad a la correspondiente relación se deben tener en cuenta ciertas características que diferencian a una relación de una tabla:

- Una relación no tiene tuplas duplicadas; por tanto, en una tabla nunca puede haber filas duplicadas.
- Las tuplas de una relación no tienen orden, luego el orden de las filas de una tabla no es significativo.
- Cada atributo de una tupla toma un solo valor del dominio correspondiente, por lo que en una tabla no puede haber más de un valor en cada celda.

Dominios.

Un **dominio** es un conjunto finito de valores homogéneos y atómicos caracterizados por un nombre.

Los atributos de una relación toman sus valores de dominios asociados a ellos. Por ejemplo, en la base de datos de nuestra empresa de distribución existirá una relación con los datos de sus clientes. En la figura mostramos las columnas que podría tener la tabla junto con sus dominios correspondientes.

Columnas/ Atributos	Dominios subyacentes
CODIGO	Cualquier combinación de cuatro dígitos.
CIF	Conjunto de todos los NIF's y CIF's posibles.
NOMBRE	Conjunto de todas las combinaciones de letras que den lugar a nombres de clientes.
DIRECCION	Conjunto de todas las combinaciones de letras que formen direcciones postales.
LOCALIDAD	Conjunto de todas los nombres de localidades españolas.
PROVINCIA	Conjunto de todos los nombres de provincias.
COD_POSTAL	Conjunto de los códigos postales.

Los valores de los dominios deben ser **homogéneos,** es decir, todos los valores de un dominio tienen que ser del mismo tipo. El dominio correspondiente a PROVINCIA no podría contener el valor "MIERES", o el valor "33600".

Los valores de los dominios también deben ser **atómicos,** lo que significa que serán indivisibles desde el punto de vista semántico, es decir, que si se descompusieran perderían el significado asociado a ellos en el modelo. Por ejemplo, en el dominio LOCALIDAD cada una de las tres palabras que componen el valor "Santiago de Compostela" tienen un significado distinto al conjunto.

Los dominios participan en las relaciones a través de los atributos. Se puede definir **atributo** diciendo que es el papel que un dominio toma en una relación. Todo atributo debe tomar sus valores de un único dominio, que se denomina **dominio subyacente**.

Las implementaciones comerciales del modelo relacional no incorporan, o lo hacen de una forma deficiente, el concepto de dominio, pese a ser un elemento muy importante del modelo.

Algunos autores hablan de **dominios compuestos** en contraposición al concepto de **dominio simple** definido aquí. Un dominio compuesto es una combinación de dominios simples que tiene un nombre y, quizá, ciertas restricciones de integridad asociadas. El ejemplo típico es un dominio FECHA, compuesto por tres dominios simples: DIA, MES y AÑO, que lleva

asociadas las restricciones adecuadas para que, de las posibles combinaciones de valores de los tres dominios simples, no se admitan como validas aquellas que dan lugar a fechas inexistentes.

Otra forma de ver un dominio es considerarlo como un **tipo de datos definido por el usuario** de un lenguaje de programación. Desde este punto de vista, podemos decir que los sistemas gestores de bases de datos relacionales implementan los dominios de una forma primitiva, puesto que suelen soportar tipos de datos primarios como INTEGER, CHAR, etc. Estos tipos de datos son **tipos definidos por el sistema** y el concepto de dominio exigiría que el sistema permitiera al usuario definir sus propios tipos de datos, tales como "nombre de localidad", "código de cliente", etc.

Para algunos autores, como C.J. Date, el concepto de dominio compuesto está inmerso en otro más amplio. Según Date, no hay nada en el modelo relacional que exija que los valores de un dominio se limiten a tipos sencillos como números, cadenas, etc.; pueden ser tan complejos como se quiera, la única limitación es que su estructura sea invisible al SGBD (valores encapsulados). Incluso, podríamos tener un dominio compuesto de relaciones. Al definir uno de estos dominios es preciso definir los operadores, o funciones, que manejen los valores del dominio. Estos operadores son los únicos que pueden ver la estructura interna de los valores del dominio, pero el SGBD no la ve. En la definición de un dominio pueden entrar otros dominios definidos previamente.

- **Relaciones.**

Dados los conjuntos D_1, D_2, ..., D_n, no necesariamente distintos, R es una relación sobre esos n conjuntos si es un conjunto de n-tuplas, cada una de las cuales obtiene su primer elemento de D_1, su segundo elemento de D_2, y así sucesivamente. Se dice que D_j es el j-ésimo dominio de R. Se puede expresar este mismo concepto diciendo que R es un subconjunto del producto cartesiano $D_1 \mathbf{X} D_2 \mathbf{X} .... \mathbf{X} D_n$ de sus dominios.

La definición matemática anterior, que es la definición original dada por Codd, presenta algunos aspectos criticables desde el punto de vista de las bases de datos, como es el que no se distinga claramente entre dominios y atributos, y que exija un orden en los elementos componentes de cada tupla. Una definición más adecuada de relación es la siguiente:

Una relación R sobre un conjunto de dominios D_1, D_2, ..., D_n, no necesariamente distintos, se compone de dos partes, una cabecera y un cuerpo.

- La **cabecera**, denominada también **intensión o esquema de la relación**, es un conjunto de n pares atributo-dominio subyacente, siendo n el **grado** de la relación.
- El cuerpo, extensión, instancia u ocurrencia de la relación es un conjunto de *m* n-tuplas (o, abreviadamente, tuplas) donde cada n-tupla es un conjunto de *n* pares atributo-valor. El valor asociado a cada atributo es tomado de su dominio subyacente. El número de n-tuplas, *m*, es la **cardinalidad** de la relación.

La cabecera es la parte estática de la relación; define a la relación, y un cambio en ella significa que tenemos otra relación distinta.

El cuerpo esta formado por el conjunto de tuplas que en un instante determinado satisfacen el correspondiente esquema de relación. La extensión de una relación, al contrario que su intensión, varía con el tiempo.

De la definición de relación se deducen las siguientes propiedades:

- En una relación no existen tuplas repetidas.
- Las tuplas de una relación no están ordenadas.
- Los atributos de una relación no tienen orden.
- Todos los valores de los atributos son atómicos.

13. Restricciones del modelo.

El modelo relacional, como todo modelo de datos, tiene unas restricciones, es decir, unas estructuras u ocurrencias no permitidas. Estas restricciones pueden ser inherentes y de usuario.

Las cuatro propiedades derivadas de la definición de relación, junto con la **regla de integridad de entidad** constituyen las **restricciones inherentes** del modelo.

Una **restricción de usuario** es un predicado definido sobre un conjunto de dominios, atributos o tuplas, que debe ser verificado por los correspondientes objetos para que constituyan una ocurrencia válida del esquema. Entre las restricciones de usuario se incluye la **regla de integridad referencial**.

Los datos almacenados en la base han de adaptarse a las estructuras impuestas por el modelo (restricciones inherentes), y han de cumplir las restricciones de usuario para ser ocurrencias válidas.

- **Claves.**

Algunas de las restricciones están definidas en torno a los conceptos de clave candidata, clave primaria o clave ajena.

Clave candidata de una relación es un subconjunto no vacío de atributos de esa relación, digamos K, que en todo instante de tiempo satisface las dos propiedades siguientes:

- Unicidad: **en cualquier momento no existen dos tuplas de la relación con el mismo valor de K.**
- Irreductibilidad: **No existe ningún subconjunto de atributos de K que cumpla la propiedad de unicidad.**

CLIENTES

CODIGO	CIF	NOMBRE	DIRECCION	LOCALIDAD	PROVINCIA	COD_POSTAL
1111	111111A	SISTEMAS INFORMATICOS S.L.	PEZ 13	MADRID	MADRID	28005
2222	2222222B	SISINFO S.A.	TRIBULETE, 14	MADRID	MADRID	28006
3333	3333333C	BYTES C.B.	MAYOR, 41	ALCOBENDAS	MADRID	28123
...	.....	.....	.....	.....	.....	.....

Esta relación tendría como claves candidatas: CODIGO y CIF.

El atributo NOMBRE podría ser clave candidata si aceptamos que no es posible tener dos clientes con el mismo nombre.

Probablemente, por simplicidad, escogeríamos CODIGO o CIF como clave primaria. Las claves candidatas no elegidas pasan a ser claves alternativas.

Claves candidatas, primaria y alternativas

Toda relación tiene por lo menos una clave candidata, ya que de la propiedad de que no pueden existir dos tuplas iguales se deriva que el conjunto de todos los atributos de la relación cumplirá siempre la propiedad de unicidad y dicho conjunto, o algún subconjunto de él, será clave candidata.

Clave primaria es una de las claves candidatas que se elige como tal según criterios ajenos al propio modelo. En principio es obligatorio elegir la clave primaria, aunque hay autores que consideran que el papel de la clave primaria debería ser desempeñado por las claves candidatas.

La clave primaria, o las claves candidatas siguiendo el criterio de los autores discrepantes, constituye el mecanismo básico de direccionamiento a nivel de tuplas en un sistema relacional.

Una vez elegida clave primaria, el resto de claves candidatas reciben el nombre de **claves alternativas.**

Clave ajena de una relación R_2 es un conjunto no vacío de atributos cuyos valores han de coincidir con los valores de la clave primaria de una relación R_1 (R_1 y R_2 pueden ser la misma relación). R_1 es la relación referenciada y R_2 la relación que referencia.

La clave ajena y la clave primaria a la que apunta deben estar definidas en el mismo dominio.

Si seguimos el criterio de que no es obligatoria la existencia de clave primaria, la clave ajena de una relación debería apuntar a una clave candidata.

ALMACEN

REFERENCIA	DENOMINACION	P_TARIFA	
0001	MONITOR 15" SONY	250,5	....
0002	MONITOR 17" SONY	280,25	....
0003	MONITOR 19" SONY	301,2	....
0004	MONITOR 21" SONY	332,2	....
0005	TECLADO GENIUS INT	57	....
...	.....	...	....

Clave primaria: REFERENCIA

PEDIDOS

NUM_PEDIDO	FECHA	CODIGO_CLIENTE
000010	14/11/2008	5555
000011	14/11/2008	7777
000012	16/11/2008	2222

Clave primaria: NUM_PEDIDO.

L_PEDIDOS

PEDIDO	REFERENCIA	CANTIDAD
000010	0005	5
000010	0003	10
000011	0006	10
000011	0005	2
000012	0005	4

Clave primaria: concatenación de PEDIDO y REFERENCIA.

Claves ajenas: PEDIDO, que apunta a NUM_PEDIDO de PEDIDOS y REFERENCIA que apunta a REFERENCIA de ALMACEN.

Claves primarias y ajenas

- **Regla de integridad de entidad.**

La **regla de integridad de entidad** dice que "ningún componente de la clave primaria de una relación puede ser nulo". Esta regla debería aplicarse también a las claves candidatas, pero el modelo no lo exige.

Surge aquí el concepto de **nulo**, muy controvertido en el modelo relacional. Un nulo no es un valor, es más bien una marca que se aplica a un atributo indicando que éste no tiene valor. El que un atributo no tenga valor puede obedecer a causas diversas, por ejemplo, a que no conocemos su valor, o a que en esa tupla concreta el atributo no es aplicable.

❑ Regla de integridad referencial.

La segunda regla general de integridad del modelo relacional, la de **integridad referencial**, dice que "la base de datos no debe contener valores de clave ajena sin concordancia". Es decir, que una clave ajena debe coincidir con un valor de clave primaria de la relación a la que apunta o tener valor nulo.

Se puede observar que los conceptos de clave ajena e integridad referencial se han definido uno en términos de otro. No es posible hablar de clave ajena sin mencionar el concepto de integridad referencial, ni definir integridad referencial sin hablar de claves ajenas.

Cualquier estado de la base de datos que no satisfaga la regla de integridad referencial, así como cualquier otra restricción, será incorrecto por definición. Ahora bien, la regla no dice nada de cómo evitar estos estados incorrectos. Una solución sería rechazar cualquier operación que intente llevar a la base de datos a un estado que viole la regla de integridad referencial. Otra opción menos drástica sería que el sistema aceptara la operación pero llevara a cabo, si fuese necesario, operaciones adicionales que dejasen la base de datos en un estado válido. Por tanto, al diseñar una base de datos, además de definir las claves ajenas, es preciso especificar qué debe hacer el sistema en caso de que se presenten operaciones de borrado o modificación que atenten contra la integridad referencial.

❑ Otras restricciones de usuario.

Algunos autores distinguen una tercera regla denominada **regla de integridad de atributo**; que dice que "todo atributo debe satisfacer la restricción de tomar los valores de su dominio subyacente". Aunque no enunciada hasta ahora, esta regla iba implícita en la definición de la relación.

Una restricción de usuario en general está constituida por un predicado definido sobre un conjunto de dominios, atributos o tuplas, que debe ser verificado por los correspondientes objetos para que constituyan una ocurrencia válida del esquema.

En las distintas implementaciones del modelo relacional, las restricciones pueden tener un soporte declarativo (se declaran en un cierto lenguaje, se almacenan en el catálogo y el sistema las verifica) o ser soportadas por procedimientos almacenados o disparadores.

Una restricción, en general, constará de un nombre, el predicado que debe ser satisfecho y una respuesta a la operación que intenta violar la restricción. También puede ser necesario

indicar el momento en que se produce la verificación de la restricción dentro de la transacción, si se hace al finalizar cada sentencia o al final de la transacción.

14. El tratamiento de nulos.

AND	C	Q	F
C	C	Q	F
Q	Q	Q	F
F	F	F	F

NOT	
C	F
Q	Q
F	C

OR	C	Q	F
C	C	C	C
Q	C	Q	Q
F	C	Q	F

IS NULL	
C	F
Q	C
F	F

Operadores lógicos en lógica trivaluada y tabla de verdad del operador IS NULL.

Cuando intentamos modelar y representar en un esquema relacional un determinado universo del discurso, sucede a veces que ciertas propiedades de una determinada entidad carecen de valor. Esta falta de valor puede ser debida a que no se conoce, caso del número de teléfono en una ocurrencia de la entidad CLIENTE, o a que esa propiedad no es aplicable a cierta ocurrencia de la entidad, como sucede con el NIF del cónyuge de un empleado soltero en la entidad EMPLEADOS.

Para resolver el problema, se introdujo en el modelo relacional el concepto de nulo (*null*). Un nulo no es realmente un valor, se trata de una marca que indica que el atributo correspondiente no tiene asignado valor.

El uso de nulos exige definir operaciones y funciones específicas para nulos, como las operaciones aritméticas, las de comparación, los operadores lógicos, operaciones del álgebra relacional o funciones estadísticas.

Sin valores nulos solo existen dos valores lógicos: **cierto** y **falso**. Con la aparición de los nulos surge un tercer valor el **quizás** (NULL). Surge así la lógica trivaluada, distinta de la lógica habitual.

En la figura se muestran las tablas de verdad de las operaciones AND, OR y NOT en lógica trivaluada.

La lógica trivaluada hace necesarios nuevos operadores como el operador unario IS NULL, que se evalúa como cierto sólo en el caso de que el operando sea nulo.

Las operaciones aritméticas en que aparece algún operador nulo dan como resultado nulo; lo mismo ocurre en las operaciones de comparación. Más adelante veremos como afectan los nulos al álgebra relacional.

La introducción de valores nulos en el modelo relacional da lugar a una serie de problemas y anomalías, por lo que algunos autores argumentan que la solución dada al problema de la información faltante no es la adecuada, y que, mientras se estudia mejor el problema, se deberían utilizar valores por defecto en lugar de nulos.

La implementación de los nulos en los productos relacionales no es uniforme, sino que varía de unos a otros, por lo que, antes de usar nulos, debemos analizar perfectamente cómo son tratados por el producto concreto que vayamos a utilizar.

Terminaremos este apartado indicando que algunos autores propugnan utilizar dos tipos de nulos, que diferencien entre los valores **inaplicables** y los valores **desconocidos aplicables**. El uso de dos tipos de nulos lleva a la aparición de la lógica cuatrivaluada, es decir, en lugar de tres valores lógicos cuatro.

15. El modelo relacional en la arquitectura ANSI.

Todos los elementos del modelo vistos hasta ahora (dominios, relaciones, claves y restricciones) constituyen el esquema conceptual de la arquitectura ANSI.

Hasta ahora hemos hablado de relaciones sin más, pero existen distintos tipos. Las **relaciones base o relaciones reales** son relaciones autónomas con nombre y tienen representación directa en el almacenamiento interno. Estas relaciones base son las que forman parte del esquema conceptual de la base de datos.

Existen otras relaciones denominadas **vistas o relaciones virtuales**. Una vista es una relación derivada con nombre, que se representa dentro del sistema exclusivamente por su definición en términos de otras relaciones con nombre; no poseen datos almacenados propios, separados y distinguibles, como las relaciones base. Las vistas, que son como ventanas sobre relaciones reales, equivalen al esquema externo de la arquitectura ANSI, aunque dicha equivalencia presenta la salvedad de que no todas las vistas relacionales son actualizables.

En la arquitectura ANSI el usuario solo tiene acceso a su esquema externo, pero en el modelo relacional tiene acceso, si dispone de las correspondientes autorizaciones, tanto a las relaciones base como a las vistas.

El modelo relacional no dice nada del esquema interno ya que se trata de un modelo lógico. Cada producto relacional implementa el esquema interno de la forma que considera más oportuna para conseguir un mejor rendimiento del sistema.

16. El álgebra relacional.

El álgebra relacional fue introducida por E.F.Codd como una colección de operaciones formales que actúan sobre las relaciones produciendo como resultado otras relaciones. Por medio de estos operadores se pueden construir expresiones que indiquen las transformaciones a realizar en una base de datos para pasar de un estado origen a otro estado objetivo. El álgebra, como todos los lenguajes relacionales, es un lenguaje de especificación que opera sobre conjuntos de tuplas.

El álgebra relacional, que veremos más adelante, constituye la parte dinámica del modelo relacional.

El álgebra original presentada por Codd estaba formada por ocho operadores, de los cuales cuatro son adaptaciones al mundo relacional de los correspondientes de teoría de conjuntos y otros cuatro son nuevos. El álgebra, como otras partes del modelo relacional, ha sufrido una evolución considerable, surgiendo nuevos operadores y modificaciones de algunos de los originales a partir de propuestas de distintos autores.

Operadores de conjuntos	Operadores especiales
Unión	Restricción
Intersección	Proyección
Diferencia	División
Producto cartesiano	Reunión natural

Origen de los operadores originales de álgebra relacional.

Los ocho operadores originales del álgebra son**: unión, producto cartesiano generalizado, diferencia, restricción, proyección, intersección, combinación natural (*natural join*) y división**. De estos ocho, los cinco primeros se suelen considerar como **operadores primitivos**, siendo la intersección, división y reunión natural **operadores derivados**, ya que se pueden deducir de los primeros.

El hecho de que el resultado de cualquier operación del álgebra relacional sea una relación, lo que se conoce como **propiedad de cierre,** hace posible escribir expresiones algebraicas anidadas, es decir, que una expresión siempre puede ser un operando de cualquier otra expresión.

17. Operadores tradicionales de conjuntos.

Son operadores definidos en la teoría de conjuntos a los que se les aplicaron ciertas restricciones para adaptarlos al hecho de que una relación es un conjunto en que sus elementos son tuplas.

Así, para que la propiedad de cierre siga siendo cierta, la unión incluida en el álgebra relacional es una versión limitada de la unión matemática, en la que se obliga a que las dos relaciones que actúan de operandos tengan cabeceras idénticas o, expresado de otra forma, a que sean **compatibles respecto de la unión**. El que dos relaciones tengan cabeceras idénticas significa que:

a) Las dos tienen el mismo conjunto de nombres de atributo; y
b) los atributos correspondientes, es decir, los que tienen el mismo nombre en ambas relaciones, se definen sobre el mismo dominio.

Hay autores (C. J. Date) que hablan de relaciones de **tipos compatibles** en lugar de compatibles respecto a la unión. Los dos conceptos son equivalentes, salvo en que en el caso de relaciones de tipos compatibles la condición ***b*** anterior diría: "los atributos correspondientes deben ser del mismo tipo (definidos en el mismo dominio) o existir alguna función de conversión de tipo (dominio) conocida por el sistema, de forma que puedan ser convertidos al mismo tipo".

La exigencia de que los nombres de atributos de las dos relaciones compatibles sean iguales, es juzgada por algunos autores como un aspecto sintáctico poco relevante y demasiado restrictivo. No obstante, aunque se mantenga el requisito, siempre es posible disponer de un operador unario, que aplicado a una relación dé como resultado otra con los nombres de atributo renombrados.

Artículos suministrados por el proveedor A

ARTPROVA

REFERENCIA	DENOMINACION
0001	MONITOR 15" SONY
0002	MONITOR 17" SONY
0003	MONITOR 19" SONY
0004	MONITOR 21" SONY

Artículos suministrados por el proveedor B

ARTPROVB

REFERENCIA	DENOMINACION
0002	MONITOR 17" SONY
0003	MONITOR 19" SONY
0005	TECLADO GENIUS INT
0010	TECLADO LOGITECH A23

Relaciones ARTPROVA y ARTPROVB.

Al igual que la unión, la diferencia y la intersección exigen relaciones de tipos compatibles. El producto cartesiano, en cambio, no tiene esta exigencia, aunque sí tiene otra que veremos más adelante.

En la figura tenemos dos relaciones correspondientes al nuestro ejemplo de la empresa de distribución. En cada una de ellas están los artículos suministrados por dos proveedores, que llamamos A y B, a la empresa.

La **unión** de dos relaciones A y B compatibles respecto de la unión es otra relación, cuya cabecera es idéntica a la de A o B y cuyo cuerpo esta formado por todas las tuplas pertenecientes a A o a B o a ambas.

En el resultado de una unión, aquellas tuplas que existen en las dos relaciones, lógicamente, aparecen una sola vez, es decir, que se eliminan las tuplas repetidas.

Artículos suministra por cualquiera de los proveedores

ARTPROVA ∪ ARTPROVB

REFERENCIA	DENOMINACION
0001	MONITOR 15" SONY
0002	MONITOR 17" SONY
0003	MONITOR 19" SONY
0004	MONITOR 21" SONY
0005	TECLADO GENIUS INT
0010	TECLADO LOGITECH A23

Unión **de ARTPROVB y ARTPROVA.**

La **intersección** de dos relaciones, compatibles respecto de la unión, A y B es otra relación cuya cabecera es idéntica a la de A o B, y cuyo cuerpo está formado por las tuplas *t* pertenecientes tanto a A como a B.

Artículos suministrados por ambos proveedores

ARTPROVA ∩ ARTPROVB

REFERENCIA	DENOMINACION
0002	MONITOR 17" SONY
0003	MONITOR 19" SONY

ARTPROVA intersección ARTPROVB

Intersección de **ARTPROVA y ARTPROVB.**

Artículos suministrados sólo por el proveedor B

ARTPROVB - ARTPROVA

REFERENCIA	DENOMINACION
0005	TECLADO GENIUS INT
0010	TECLADO LOGITECH A23

ARTPROVB menos ARTPROVA

Diferencia de **ARTPROVB** menos **ARTPROVA.**

La **diferencia** entre dos relaciones, compatibles respecto de la unión, A y B es otra relación cuya cabecera es idéntica a la de A o B, y cuyo cuerpo está formado por todas las tuplas *t* pertenecientes a A pero no a B. (la imagen de la derecha las referencias debe ser 1 y 4)

Artículos suministrados sólo por el proveedor A

ARTPROVA - ARTPROVB

REFERENCIA	DENOMINACION
0002	MONITOR 15" SONY
0003	MONITOR 21" SONY

ARTPROVA menos ARTPROVB

Diferencia de **ARTPROVA** menos **ARTPROVB.**

El **producto cartesiano** de dos relaciones, A y B, compatibles respecto al producto, es otra relación cuya cabecera es la combinación de las cabeceras de A y B y cuyo cuerpo esta formado por el conjunto de todas las tuplas *t,* tales que *t* es la combinación de una tupla *a* perteneciente a A y una tupla *b* perteneciente a B.

El producto cartesiano generalizado o ampliado, definido en el párrafo anterior, parte del concepto de producto cartesiano de dos conjuntos adaptándolo para tener en cuenta que los conjuntos operando son relaciones.

En matemáticas, el producto cartesiano de dos conjuntos es el conjunto de todos los pares ordenados de elementos tales que el primer elemento de cada par pertenece al primer conjunto y el segundo elemento de cada par pertenece al segundo conjunto. Si se aplica esta operación a dos relaciones obtendríamos un conjunto de pares ordenados de tuplas, es decir, que no obtendríamos una relación con lo que se estaría atentando contra la propiedad de cierre del algebra relacional. El producto cartesiano de relaciones, sustituye el par ordenado de tuplas del producto cartesiano matemático por una tupla, que contiene todos los pares atributo-valor de dicho par de tuplas.

Puesto que la cabecera de un producto cartesiano generalizado estará formado por la combinación de las cabeceras de las dos relaciones iniciales, éstas no pueden tener nombres de atributos en común. El que dos relaciones tengan algún nombre de atributo en común, sólo obligaría a renombrar uno de ellos con el operador adecuado.

PEDIDOS

NUM_PEDIDO	FECHA	CODIGO_CLIENTE
000010	14/11/2008	5555
000011	14/11/2008	7777
...	...	...

L_PEDIDOS

PEDIDO	REFERENCIA	CANTIDAD
000010	0005	5
000010	0003	10
000011	0006	10
...	...	...

PEDIDOS X L_PEDIDOS

NUM_PEDIDO	FECHA	CODIGO_CLIENTE	PEDIDO	REFERENCIA	CANTIDAD
000010	14/11/2008	5555	000010	0003	10
000010	14/11/2008	5555	000010	0005	5
000010	14/11/2008	5555	000011	0006	10
000011	14/11/2008	7777	000010	0003	10
000011	14/11/2008	7777	000010	0005	5
000011	14/11/2008	7777	000011	0006	10
...	...	...	...	...	...

Producto cartesiano generalizado.

Diremos que dos relaciones son **compatibles con respecto al producto** si y, sólo si, sus cabeceras son disjuntas, es decir no tienen nombres de atributo en común.

La operación de producto cartesiano generalizado se incluye en el álgebra por razones conceptuales y no por su importancia práctica.

Los operadores unión, intersección y producto tienen las propiedades conmutativa y asociativa, mientras que la diferencia no las posee.

18. Operaciones relacionales especiales.

La restricción es un operador unario que permite seleccionar las tuplas de una relación que satisfacen una determinada condición o predicado de selección. Una definición formal de la **restricción**, también conocida como **selección**, es la siguiente:

Sea *theta* cualquier operador de comparación escalar simple (por ejemplo =, <>, >, >=, etc); la restricción *theta* de la relación A, según los atributos X e Y**,** denotada $\sigma_{X\Theta Y}$ **(A)**, es una relación con la misma cabecera de A y con un cuerpo formado por el conjunto de todas las tuplas *t* de A para las que la condición "X *theta* Y" se evalúa afirmativamente. Los atributos X e Y deben estar definidos sobre el mismo dominio y la operación **theta** debe ser aplicable a ese dominio.

Se puede especificar un literal en lugar de cualquiera de los dos atributos de la condición. También se pueden utilizar condiciones compuestas formadas por condiciones simples unidas por los operadores AND, OR y NOT.

$\sigma_{PEDIDO = NUM_PEDIDO}$ (PEDIDOS X L_PEDIDOS)

NUM_PEDIDO	FECHA	CODIGO_ CLIENTE	PEDIDO	REFERENCIA	CANTIDAD
000010	14/11/2008	5555	000010	0003	10
000010	14/11/2008	5555	000010	0005	5
000011	14/11/2008	7777	000011	0006	10
...	...	...	...	...	...

Se ha restringido la relación resultado del producto cartesiano a aquellas tuplas en que los valores de PEDIDO y NUM_PEDIDO son iguales:

*Ejemplo de **RESTRICCIÓN.***

La **proyección** de la relación A, según los atributos X, Y, ..., Z, denotada $\Pi_{X, Y, ..., Z}$ **(A)**, es una relación con (X, Y, ..., Z) como cabecera y cuyo cuerpo está formado por el conjunto de

todas las tuplas de A definidas sobre los atributos (X, Y, ...,Z), eliminando las que resulten duplicadas.

$\Pi_{\text{CODIGO_CLIENTE, PEDIDO, REFERENCIA, CANTIDAD}}$ **(**$\sigma_{\text{PEDIDO = NUM_PEDIDO}}$ (PEDIDOS X L_PEDIDOS)**)**

CODIGO_CLIENTE	PEDIDO	REFERENCIA	CANTIDAD
5555	000010	0003	10
5555	000010	0005	5
7777	000011	0006	10
...	...	...	...

Se ha proyectado la relación resultado de la restricción sobre los atributos CODIGO_CLIENTE, PEDIDO, REFERENCIA y CANTIDAD.

*Ejemplo de **PROYECCIÓN.***

Podríamos decir que la proyección de una relación sobre ciertos atributos se obtiene cortando verticalmente las columnas correspondientes a los atributos y pegándolas para formar la tabla resultado (eliminando aquellas filas duplicadas). De igual forma, la restricción de una relación se construiría cortando horizontalmente aquellas filas que cumplen el predicado de selección y pegándolas.

Existen distintos tipos de reunión, siendo el más general la **reunión *theta***. En ella participan dos relaciones compatibles con respecto al producto, es decir, que no tienen atributos en común, y el resultado consta de las tuplas que se obtienen concatenando cada tupla de una relación con todas las tuplas de la segunda y eliminando aquellas que no cumplan una cierta condición.

Una definición sería la siguiente:

Sean las relaciones A y B compatibles con respecto al producto y sea *theta* un operador como el definido en la operación de restricción, la **reunión *theta*** de la relación A, según el atributo X, con la relación B, según el atributo Y**,** denotada $\mathbf{(A * B)_{X \Theta Y}}$, se define como el resultado de evaluar la expresión $\mathbf{\sigma_{X \Theta Y} (A \times B)}$**.**

Dicho de otra forma, el resultado es una relación con la misma cabecera que el producto cartesiano de A y B y con un cuerpo formado por el conjunto de todas las tuplas *t*, tales que *t* pertenece a ese producto cartesiano y la evaluación de la condición "X *theta* Y" resulta cierta para esa tupla t. Los atributos X e Y deberán estar definidos sobre el mismo dominio.

(PEDIDOS * L_PEDIDOS)$_{NUM_PEDIDO = PEDIDO}$

NUM_PEDIDO	FECHA	CODIGO_CLIENTE	REFERENCIA	CANTIDAD
000010	14/11/2008	5555	0003	10
000010	14/11/2008	5555	0005	5
000011	14/11/2008	7777	0006	10
...	...	...	...	...

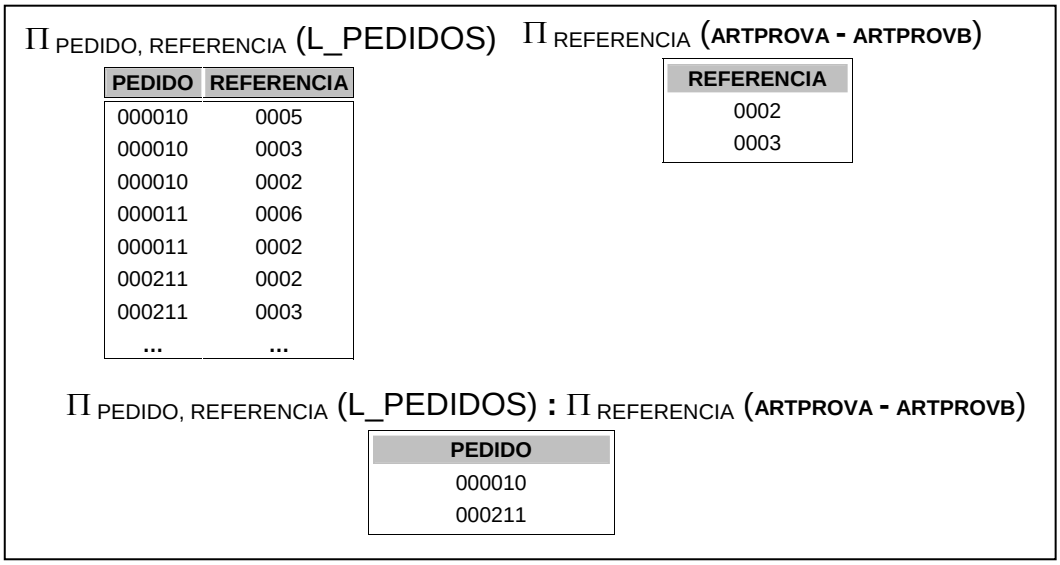
La reunión natural de PEDIDOS con L_PEDIDOS da lugar a una relación en que se ha eliminado una de las columnas donde aparecía el número de pedido.

*Ejemplo de **REUNIÓN NATURAL.***

Si el operador theta es "igual a", la reunión theta se llama **equirreunión**. Una equirreunión en que se elimina uno de los atributos que participa en la condición de igualdad se denomina **reunión natural**, y es el tipo de reunión más usado en la práctica. Una reunión natural es el resultado de un producto cartesiano seguido de una restricción por igualdad y finalmente una proyección que elimina uno de los atributos X o Y.

La **división** de dos relaciones es otra relación constituida por las tuplas que, al completarse con las tuplas (con todas) de la segunda relación, permiten obtener tuplas de la primera.

Si la relación A, relación dividendo, tiene su cabecera formada por los atributos X_1, X_2, ..., X_m, Y_1, Y_2, ..., Y_n y la relación B, relación divisor, tiene la cabecera formada por los atributos Y_1, Y_2, ...,



*Ejemplo de **DIVISIÓN.***

Y_n, la división de A entre B, denotada **A : B**, es una relación con la cabecera formada por los atributos X_1, X_2, ..., X_m, y el cuerpo formado por el conjunto de todas las tuplas tales que concatenadas con cada una de las tuplas de B dan lugar a tuplas incluidas en A.

Todos los atributos del divisor coinciden con atributos del dividendo y están definidos en los mismos dominios.

19. Expresiones algebraicas.

La propiedad de cierre del álgebra relacional permite escribir expresiones algebraicas. Estas expresiones sirven a distintos propósitos, entre los que están los siguientes:

- Definir el alcance de una operación de recuperación; es decir, definir qué datos se van a extraer de una base de datos en una consulta.
- Definir el alcance de una actualización; es decir, definir los datos que se van a insertar, modificar o eliminar en una actualización.
- Definir relaciones virtuales o vistas.
- Definir derechos de acceso; es decir, definir los datos incluidos en algún tipo de autorización.
- Definir restricciones de integridad; es decir, definir alguna restricción de usuario que deba satisfacer la base de datos.

Mediante una expresión algebraica el usuario especificará al sistema qué quiere hacer. Antes de ejecutar la expresión, el sistema la transformará en otra más eficiente desde el punto de vista del rendimiento. Este proceso de optimización se lleva a cabo utilizando reglas de transformación del álgebra relacional. Por tanto, el álgebra constituye la base del proceso de optimización necesario en todo sistema relacional.

20. Operadores adicionales .

Con el fin de mejorar el poder expresivo de álgebra relacional, distintos autores han propuestos operadores adicionales. Algunos de esos operadores, que se especifican a continuación, son los de: renombrado, ampliación, resumen y reunión externa.

El operador de **renombrado** permite cambiar el nombre de los atributos de una relación. El cambio de nombre de un atributo puede ser necesario, como ya se indicó, para realizar algunas operaciones algebraicas. Este operador toma una relación especificada y devuelve una copia de dicha relación con algunos de los atributos originales cambiados de nombre.

L_PEDIDOS **RENOMBRAR** REFERENCIA **COMO** CODART, PEDIDO **COMO** NUM_PEDIDO

En este ejemplo, se cambia el nombre de los atributos REFERENCIA y PEDIDOS de la relación L_PEDIDOS a CODART y NUM_PEDIDO respectivamente.

El operador de **ampliación** dota al álgebra relacional de capacidad de cálculo escalar.

El operador de **ampliación** toma como operando una relación y devuelve otra semejante a la original pero con atributos añadidos. Estos atributos nuevos se obtienen de la evaluación de expresiones de cálculo escalar especificadas. Por ejemplo,

ALMACEN **AMPLIAR** (P_TARIFA * EXISTENCIAS) **COMO** VALOR_STOCK

La relación resultado contiene las tuplas de ALMACEN ampliadas en el producto precio de tarifa por existencias, es decir, que aparecen los artículos en almacén con sus existencias valoradas a precio de tarifa.

El operador **resumen**, o **agrupación**, permite agrupar en subconjuntos tuplas con valores comunes de ciertos atributos y aplicar a estos subconjuntos funciones de grupo como SUMA, CUENTA, PROMEDIO, MÁXIMO, MÍNIMO, etc. Por ejemplo, en la siguiente expresión

L_PEDIDOS **AGRUPAR POR** PEDIDO, **CUENTA** (REFERENCIA) **COMO** LINEAS_PEDIDO

el operador de agrupación (**AGRUPAR POR)** se aplica a la relación L_PEDIDOS, formándose grupos de tuplas con igual valor en el atributo PEDIDO, y se cuentan el número de valores del atributo REFERENCIA; cada tupla de la relación resultado tendrá un código de pedido y el número de líneas que contiene ese pedido

- **Reunión externa**

La reunión externa es una versión ampliada de la operación normal o interna, en cuyo resultado aparecen las tuplas de una relación que no tienen contraparte en la otra, con nulos en las posiciones de los demás atributos. No es una operación primitiva pero se utiliza bastante en la práctica.

La reunión natural de la relación

$\Pi_{\text{REFERENCIA, DENOMINACION}}$ **(ALMACEN)**

con la relación

$\Pi_{\text{PEDIDO, REFERENCIA}}$ **(L_PEDIDOS)**

no incorpora las tuplas de la primera que no tienen contrapartida en la segunda, es decir, las tuplas de referencias 0001, 0002 y 0004. En cambio, cuando se hace la reunión externa estas tuplas se incorporan al resultado.

$\Pi_{\text{PEDIDO, REFERENCIA}}$ **(L_PEDIDOS)**

PEDIDO	REFERENCIA
000010	0005
000010	0003
000011	0006
000011	0005
000012	0005

$\Pi_{\text{REFERENCIA, DENOMINACION}}$ **(ALMACEN)**

REFERENCIA	DENOMINACION
0001	MONITOR 15" SONY
0002	MONITOR 17" SONY
0003	MONITOR 19" SONY
0004	MONITOR 21" SONY
0005	TECLADO GENIUS INT
...	.....

Reunión natural

REFERENCIA	DENOMINACION	PEDIDO
0003	MONITOR 19" SONY	000010
0005	TECLADO GENIUS INT	000010
0005	TECLADO GENIUS INT	000011
0005	TECLADO GENIUS INT	000012
0006	RATON OPT. GENIUS G-34	000011

Reunión natural externa

REFERENCIA	DENOMINACION	PEDIDO
0001	MONITOR 15" SONY	null
0002	MONITOR 17" SONY	null
0003	MONITOR 19" SONY	000010
0004	MONITOR 21" SONY	null
0005	TECLADO GENIUS INT	000010
0005	TECLADO GENIUS INT	000011
0005	TECLADO GENIUS INT	000012
0006	RATON OPT. GENIUS G-34	000011
...	....	...

LA REUNIÓN EXTERNA.

Se distingue entre reunión externa izquierda (left outer join) y reunión externa derecha (right outer join) según esté como primer operando o segundo operando la relación que tiene tuplas sin contrapartida. También se habla de reunión externa simétrica cuando hay tuplas sin contrapartida en las dos relaciones.

Introducción

Visto el modelo relacional en la unidad de trabajo anterior, nos ocuparemos ahora del estudio de las características y organización de los Sistemas Gestores de Bases de Datos que lo implementan.

La unidad comienza analizando qué características debe poseer un SGBD para que pueda ser calificado como relacional, pasando a continuación al estudio de la organización de estos sistemas.

La organización de un SGBD relacional se estudia viendo las características del SGBD Oracle. Se trata, en primer lugar, la estructura de la base de datos, analizando cómo se organizan y gestionan las estructuras de almacenamiento para dar soporte a los objetos relacionales, y para lograr una protección adecuada de los datos y un rendimiento óptimo en el acceso a los mismos. Después, se describen las estructuras de memoria y los procesos que desarrollan la funcionalidad del SGBD. Finalmente, nos ocupamos del catálogo de la base de datos.

Por último se comenta como se descarga, se instala y se crean usuarios en Oracle 10G

Objetivos

- Identificar las características que definen un SGBD cómo relacional.
- Describir los elementos que componen la estructuras físicas y lógicas de una base de datos.
- Relacionar la estructura de una base de datos con los tres niveles de estructuración ANSI.
- Comprender los mecanismos de gestión del espacio de almacenamiento.
- Identificar los mecanismos que optimizan los procesos de acceso a los datos.
- Conocer los principales procesos y estructuras de memoria de un SGBD Oracle.
- Explicar la estructura del diccionario de datos de Oracle.

Contenidos Generales

1. CARACTERIZACIÓN DE UN SGBD RELACIONAL 3
2. EL SGBD ORACLE (ORACLE SERVER) 4
3. ARQUITECTURA DE LA BASE DE DATOS. 4
4. ESTRUCTURA FÍSICA DE LA BASE DE DATOS 5
5. TABLESPACES, BLOQUES DE DATOS, EXTENTS Y SEGMENTOS. 7
6. OBJETOS DE ESQUEMA. 10
7. ESTRUCTURAS DE MEMORIA Y PROCESOS. 16
8. EL DICCIONARIO DE DATOS. 19
9. DESCARGAR ORACLE 10G EXPRESS EDITION 23
10. INSTALAR ORACLE 10G EXPRESS EDITION 26
11. TRABAJANDO CON ORACLE 10G 30
11.1. Crear un nuevo usuario 30
11.2. Trabajando con usuarios 32

1. Caracterización de un SGBD Relacional

La caracterización de un sistema gestor de bases de datos como relacional es una cuestión que tiene cierta complejidad. En principio, para que un sistema sea relacional debería implementar con total fidelidad el modelo relacional. La realidad es que, aunque existen en el mercado multitud de productos con la etiqueta de relacional, ninguno cumple la condición anterior. Entonces, habría que tener en cuenta que no todas las características del modelo tienen la misma importancia, con lo que bastaría que un SGBD incorporase en su diseño aquéllas consideradas como fundamentales para poder ser llamado relacional.

En 1982 E. F. Codd estableció que para que un SGBD pueda considerarse relacional debe poseer las dos características siguientes:

El usuario debe percibir las bases de datos como tablas, y nada más que como tablas.

El SGBD debe manejar las operaciones de restricción, proyección y reunión natural, sin requerir definiciones previas de rutas de acceso físico.

La primera característica, que es fundamental en todo sistema relacional al facilitar la comprensión y el uso de la base de datos por parte del usuario, no basta para definir un sistema como relacional. Son necesarias también unas operaciones que, aunque no sean todas las del álgebra relacional, permiten resolver la mayoría de los problemas prácticos.

Un SGBD que disponga de las tres operaciones anteriores pero que necesite de la definición previa por el usuario de rutas de acceso físico para manejarlas no puede considerarse como relacional, ya que ello significaría atentar contra uno de los principios del modelo relacional: la independencia entre la estructura lógica de los datos y su almacenamiento interno. Así, por ejemplo, no puede considerarse relacional un sistema que implementa la operación de restricción de forma que en la condición solo puedan intervenir columnas sobre las que existe definido un índice.

La existencia de índices, o cualquier otro mecanismo de rutas de acceso físico a los datos, mejora el rendimiento del sistema cuando procesa operaciones de manipulación de la base de datos. Aunque el uso de estos mecanismos no puede imponerse a los usuarios, el propio SGBD relacional los utiliza. Ante cualquier operación planteada por el usuario, la parte del sistema conocida como el **optimizador** se encargará de establecer las rutas más eficientes de acceso a los datos.

Según C. J. Date, un SGBD que sólo cumpla los dos criterios anteriores es un sistema **mínimamente relacional**. Si, además, maneja todas las operaciones del álgebra relacional se puede decir que es **relacionalmente completo**. Y si, también, posee otras características del

modelo relacional, incluyendo los dominios y las dos reglas de integridad, se denomina **totalmente relacional**.

En el resto del capítulo, analizaremos las características de un sistema relacional real: el SGBD Oracle. A través de este estudio veremos cómo se implementan en la práctica muchas de las características del modelo relacional.

2. El SGBD Oracle (Oracle Server).

En el año 1979 Relational Software Inc. presenta el primer SGBD relacional: Oracle V2. Esta versión no soportaba transacciones, pero sí toda la funcionalidad SQL de queries y joins.

En 1983 RSI cambia su nombre definitivo a Oracle Corporation, y lanza Oracle V3, agregando el manejo de transacciones a través de las instrucciones COMMIT y ROLLBACK.

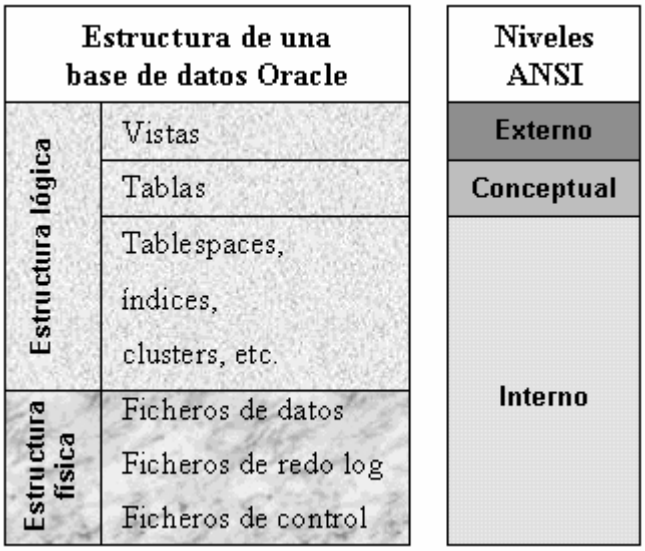
En años posteriores, van surgiendo sucesivas versiones que amplían y potencian las capacidades de este sistema. Paralelamente al desarrollo de estas versiones, aparecen herramientas de análisis, desarrollo y usuario final que hacen uso y complementan la funcionalidad del SGBD Oracle. En la actualidad el SGBD Oracle domina totalmente el mercado de las bases de datos, y las versiones del producto están disponible para una gran variedad de plataformas hardware/sotfware.

Un SGBD Oracle (*Oracle server* en la documentación del producto) consta de: **una base de datos Oracle** y **una instancia Oracle**.

Una **base de datos Oracle** es un conjunto de información tratado como una unidad, y una **instancia Oracle** está formada por un conjunto de procesos y estructuras de memoria compartidas que permiten definir, almacenar y manipular la base de datos, así como controlar el acceso, concurrencia y uso de la información.

3. Arquitectura de la base de datos.

En la documentación del producto, se dice que una base de datos Oracle tiene una **estructura física** y una **estructura lógica**. La **estructura física** está constituida por los ficheros del sistema operativo que dan soporte a los datos. Los *tablespaces* y los



Los niveles ANSI y la estructura de la base de datos Oracle.

objetos de esquema son los elementos que configuran la **estructura lógica**.

Cuando Oracle habla de "estructura lógica", está incluyendo en el término elementos que corresponden a los niveles externo y conceptual de la arquitectura ANSI, lo que en el capítulo anterior considerábamos estructuras lógicas, pero también elementos que corresponden al nivel interno ANSI. En la figura se muestran los niveles ANSI de la base de datos Oracle, observamos que algunos elementos de la estructura lógica Oracle (tablespaces, índices y *clusters*) están en el nivel interno ANSI.

Los SGBD se apoyan, para gestionar el acceso a los datos, en el sistema operativo sobre el que se ejecutan. En qué grado tiene lugar este apoyo, es decir, qué estructuras son gestionadas con el apoyo del sistema operativo y cuáles directamente por el SGBD, varía de unos sistemas a otros. En el caso de Oracle, la estructura física es gestionada a través de las rutinas del sistema operativo y la estructura lógica es gestionada directamente por el software de Oracle.

El SGBD de Oracle soporta el lenguaje SQL, tanto autocontenido como huésped. El usuario puede plantear operaciones de definición o manipulación de objetos de los niveles externo y conceptual en SQL estándar. Para definir de objetos del nivel interno el SGBD también soporta SQL, aunque, como es lógico, se trata de una extensión del lenguaje propia de Oracle.

4. Estructura física de la base de datos.

La estructura física de una base de datos ORACLE está compuesta por tres tipos de ficheros: **ficheros de datos**, **ficheros *redo log*** y **ficheros de control**.

Una base de datos Oracle está asociada a uno o más **ficheros de datos**, que contendrán todos los datos de la base Los datos de las estructuras lógicas, como tablas e índices, se almacenan físicamente en ficheros reservados para la base de datos.

Los **ficheros de datos** tienen las siguientes características:

- Un fichero sólo puede estar asociado a una base de datos.
- El fichero, aunque en el momento su creación para asociarlo a la base de datos tiene un tamaño fijo, puede definirse con características de "*auto-extend*" para hacer que incremente su tamaño en una cierta cantidad cuando se agote el espacio de almacenamiento de la base de datos.
- Uno o más ficheros de una base de datos forman una unidad lógica de almacenamiento denominada ***tablespace***.

Durante el funcionamiento normal de la base de datos, la información contenida en un fichero de datos se lee, cuando se necesita, y se almacena en memoria caché del servidor Oracle. Por ejemplo, si un usuario quiere acceder a ciertos datos de una tabla sólo se lee del fichero correspondiente si la información solicitada no está en la caché.

La información nueva, o la modificada, no se graba necesariamente de inmediato en un fichero de datos. Para reducir el número de accesos a disco y, en consecuencia, mejorar el rendimiento, los datos se almacenan en memoria y son grabados todos de una vez, en determinado instante, por el proceso de la instancia Oracle denominado "*Database Writer*".

Los ficheros de ***redo log*** desempeñan un papel fundamental en la seguridad de la base de datos. El concepto de fichero de *redo log* corresponde exactamente al de fichero **diario** visto en la unidad anterior, y, por tanto, se trata de un fichero en el que el SGBD graba la información necesaria para la recuperación de la base de datos después de un fallo en el sistema o en los dispositivos de almacenamiento.

Cada uno de los ficheros de *redo log* tiene un tamaño que se determina en el momento de su creación y que no es modificable posteriormente. Una base de datos tiene dos o más ficheros de este tipo siendo utilizados de forma cíclica, es decir, el SGBD empieza a grabar en uno de ellos y cuando lo llena pasa al siguiente, luego al siguiente, y así sucesivamente hasta llenarlos todos, momento en que empieza a grabar nuevamente en el primero.

Una base de datos Oracle puede funcionar en modo *NOARCHIVELOG*, modo por defecto, o en *ARCHIVELOG*. Cuando trabaja en el segundo modo un proceso del SGBD se encarga de salvar el contenido de los ficheros de *redo log* que se han llenado en otro soporte. De esta forma, en una base de datos trabajando en modo *ARCHIVELOG* se habla de ficheros *archived redo log,* u *offline redo log*, refiriéndose a aquellos copiados en un soporte fuera de línea, y ficheros *online redo log*, aquellos en disco sobre los que graba directamente el SGBD. Cuando se trata de recuperar la base de datos, después de un fallo que haya destruido ficheros de la base, son necesarios todos los ficheros *archived redo log* generados desde el último *backup*, además, por supuesto, de dicho *backup.*

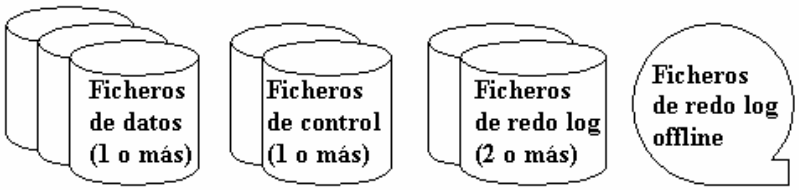
Puesto que la información de *redo log* es crítica en la seguridad de la base de datos, para protegerse contra fallos en los propios ficheros de *redo log* Oracle da la posibilidad de disponer de copias de estos ficheros en diferentes discos.

Cada base de datos Oracle tiene un **fichero de control**. En este fichero se registra la estructura física de la base de datos, y contiene, entre otros, los siguientes datos:

- Nombre de la base de datos.

- Fecha y hora de creación de la base de datos.
- Nombres y localizaciones de los ficheros de datos y de redo log
- Información sobre puntos de verificación.

El **fichero de control** es imprescindible en el momento de arranque de la base para localizar los ficheros de datos y de *redo log* a utilizar. También, es imprescindible en el proceso de recuperación en caliente de la base de datos después de un fallo, puesto que, según vimos



Ficheros de una base de datos Oracle.

en la unidad 1, la información sobre el último punto de verificación ejecutado es fundamental para saber qué transacciones deben ser rehechas y cuáles deshechas.

Este fichero, que en la unidad 1 llamábamos fichero de rearranque, debe estar disponible en todo momento durante el funcionamiento del SGBD, dado que en él se va a grabar información cada vez que se produzca un *checkpoint* o se modifique la estructura física de la base de datos.

Al igual que en el caso de los ficheros de *redo log*, existe la posibilidad de gestionar varias copias del fichero de control en diferentes dispositivos, con lo que se protege la información vital que contiene contra posibles fallos de disco.

5. Tablespaces, bloques de datos, extents y segmentos.

En una base de datos Oracle el espacio de almacenamiento físico, constituido por los ficheros de datos, está estructurado en una o varias unidades lógicas denominadas *tablespaces.* El SGBD gestiona estas unidades lógicas asignando espacio en ellas a los objetos de esquema como tablas, índices, etc.; para almacenar sus datos.

Toda base de datos Oracle dispone al menos de un *tablespace* de nombre *SYSTEM* en el que están las tablas que soportan el catálogo. Como vimos en el capítulo anterior, el catálogo es el diccionario activo gestionado por el propio SGBD que debe existir en todo sistema relacional.

Un *tablespace* puede estar *online* u *offline*. Normalmente todos los *tablespaces* están *online*, con lo que todos los objetos con espacio asignado en ellos están accesibles. En

momentos determinados, el administrador de la base de datos puede poner *offline* algún *tablespace* para realizar en él tareas de mantenimiento como *backups* o actualizaciones. Mientras un *tablespace* esté *offline* los objetos almacenados en él permanecen inaccesibles para los usuarios y aplicaciones. El *tablespace SYSTEM* siempre debe estar *online*, ya que en caso contrario el SGBD no podría funcionar ante la imposibilidad de acceder al catálogo.

Los *tablespaces* facilitan las tareas de administración de la base de datos. Así, por ejemplo, los *tablespaces* permiten al administrador:

- Controlar la asignación de espacio en disco a los distintos objetos de la base, decidiendo en qué *tablespace* se almacenarán los datos de cada objeto y estableciendo un límite de ocupación de espacio para cada objeto.
- Controlar el uso del espacio de almacenamiento por parte de los usuarios asignándoles límites específicos en cada *tablespace*.
- Controlar la disponibilidad de los datos poniendo determinados *tablespaces online* u *offline.*
- Realizar operaciones parciales de *backup* o *restore* de la base de datos.

El espacio total de almacenamiento de la base de datos será la suma de los espacios de cada *tablespace* y si en algún momento dicho espacio se agota, el administrador debe ampliarlo. El espacio de una base de datos puede ampliarse creando un nuevo *tablespace* o haciendo más grande un *tablespace* ya existente. Un *tablespace* se puede ampliar añadiéndole un nuevo fichero o permitiendo que sus ficheros crezcan dinámicamente cuando se precisa más espacio.

Para el SGBD un *tablespace* es un conjunto de **bloques de datos** o **páginas** en los que va almacenando los datos de los distintos objetos. Un bloque se corresponde con un número determinado de bytes en disco. Todos los bloques de un *tablespace* son del mismo tamaño. El tamaño del bloque de datos del *tablespace* SYSTEM se establece en el momento de la creación de la base y debe ser un múltiplo del tamaño del bloque físico manejado por el sistema operativo; su valor puede estar comprendido entre 2KB y 32KB

Cada bloque contiene datos de un solo objeto, y un objeto tendrá, normalmente, repartidos sus datos por varios bloques del mismo *tablespace*. Cuando es preciso almacenar más datos en un objeto y en los bloques que tiene asignados no hay espacio, el SGBD le asigna más bloques. La unidad de asignación de bloques a un objeto es el *extent*.

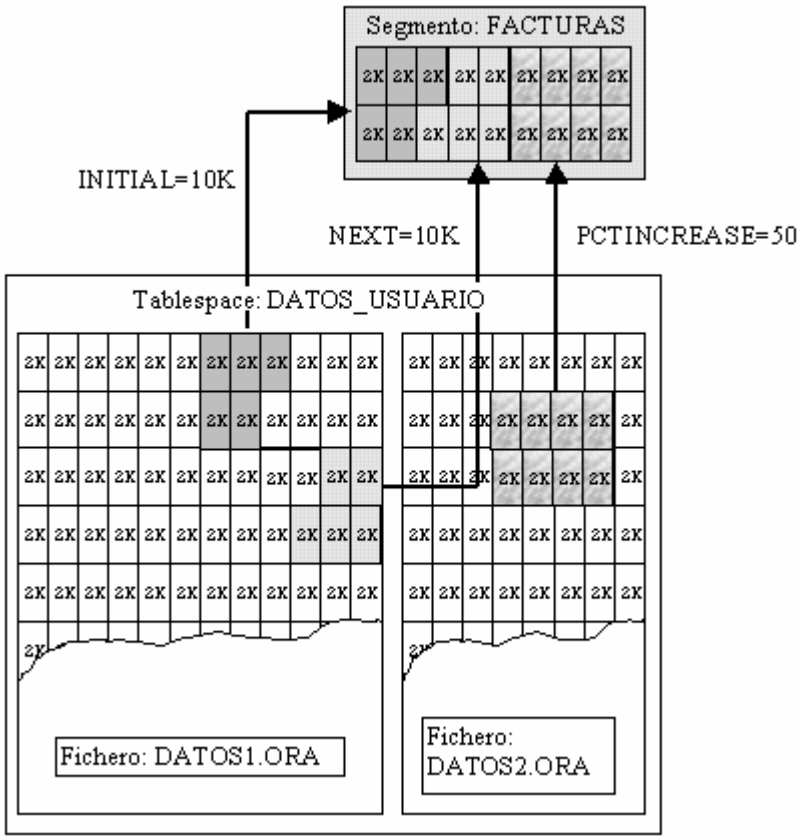
Un *extent* es un número determinado de bloques contiguos que se asignan en una sola operación a un objeto. El tamaño del *extent* varía tanto de un objeto a otro como de una

asignación a otra en el mismo objeto. Este tamaño viene controlado por los parámetros *INITIAL, NEXT* y *PCTINCREASE*.

Los valores de los parámetros *INITIAL* y *NEXT* marcan, respectivamente, los tamaños del primer y segundo *extent* asignados a un objeto. El tamaño de cualquier otro *extent* será igual al tamaño del último *extent* asignado al objeto incrementado en un porcentaje igual al valor del parámetro *PCTINCREASE*.

Los valores de los parámetros anteriores se pueden dar al crear cada objeto, aunque, si no se hace así, se aplicarán al objeto los valores dados a los parámetros al crear el *tablespace* en que se va a almacenar.

El conjunto de *extents* asignados a un objeto recibe el nombre de **segmento**. Todos los *extents* de un segmento pertenecen al mismo *tablespace*, aunque pueden pertenecer a distintos ficheros de dicho *tablespace*. Cuando se crea un objeto, por ejemplo una tabla, el SGBD le asigna un segmento con uno o más *extents.* El número de *extents* iniciales de un segmento viene dado por el parámetro *MINEXTENS*, cuyo valor por defecto es 2 para los segmentos de *rollback* y 1 para el resto.



Segmentos, extents y bloques.

En la figura vemos un ejemplo correspondiente al segmento de la tabla FACTURAS. En un determinado instante tiene 18 bloques, asignados mediante 3 *extents* cuyos tamaños fueron marcados por los parámetros INITIAL, NEXT y PCTINCREASE. De acuerdo con el valor de PCTINCREASE, el tercer *extent* debería ser un 50% mayor que el segundo, es decir, de 15K; pero dado que el mínimo espacio que se puede asignar es un bloque, el SGBD redondea siempre los valores por arriba a un número entero de bloques, y por tanto asignó 16K (8 bloques).

Existen cuatro tipos de segmentos: de datos, de índice, temporales y de *rollback*.

Los **segmentos de datos** y los **segmentos de índice** se utilizan, respectivamente, para almacenar los datos de las tablas e índices de la base. A cada índice definido en la base el SGBD le asigna un segmento de índice y a cada tabla o *cluster* le asigna un segmento de datos. Un *cluster* es una estructura que permite almacenar juntas una o más tablas.

Los **segmentos temporales** son empleados por el SGBD para almacenar la información producida en las etapas intermedias del proceso de ciertas sentencias SQL. Por ejemplo, cuando se ejecuta una sentencia SQL de consulta que exige un resultado con las filas ordenadas, el SGBD intentará obtener el resultado mediante el uso de índices, si no es posible, ordenará el resultado de la consulta en memoria, pero si no hay suficiente memoria disponible, utilizará un segmento temporal para realizar la operación.

Los **segmentos de *rollback*** son usados por el SGBD durante la ejecución de las transacciones. En la unidad de trabajo anterior, vimos que una transacción es una secuencia de operaciones que han de ejecutarse de forma atómica, es decir, o se realizan todas o ninguna. Por cada operación de la transacción que se ejecute, el SGBD guarda en un segmento de *rollback* la información necesaria que permita deshacerla más adelante si fuera necesario. La acción de deshacer todas las operaciones ya ejecutadas de una transacción se denomina ***rollback***.

La información almacenada en los segmentos de *rollback* se utiliza también para facilitar lecturas consistentes de los datos involucrados en una transacción. Cuando un usuario o una aplicación consultan datos que han sido modificados por una transacción aun sin concluir, el SGBD suministra los valores previos a la modificación utilizando la información contenida en el correspondiente segmento de *rollback,* con lo que el resultado de la consulta es el mismo que si se hubiese realizado justo antes del comienzo de la transacción. De esta forma, los efectos de una transacción no son visibles hasta su conclusión.

Los segmentos de *rollback* también son vitales en la recuperación la base de datos después de un fallo del sistema.

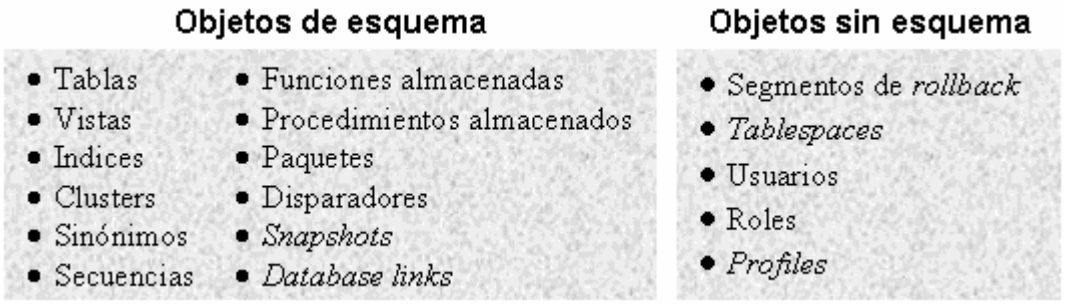
6. Objetos de esquema.

En una base de datos Oracle se denomina **esquema** a una colección de objetos. Asociado a cada usuario definido en la base de datos existe un esquema con su mismo nombre. Todos los objetos de un esquema son propiedad del usuario asociado a él.

Los **objetos de esquema** son estructuras lógicas que aluden directamente a los datos de la base y que aparecen siempre formando parte de un esquema. En la figura aparecen tipos de objetos de esquema y otros que no pueden formar parte de un esquema. Estudiaremos a continuación las características de algunos de estos objetos.

No existe relación entre los *tablespaces* y los esquemas; objetos en el mismo esquema pueden estar almacenados en distintos *tablespaces,* y un *tablespace* puede contener objetos de diferentes esquemas.

Los datos de un objeto de esquema están almacenados en un solo *tablespace*, pero pueden estar repartidos por los ficheros de dicho *tablespace*.



Objetos de esquema

- **Tablas.**

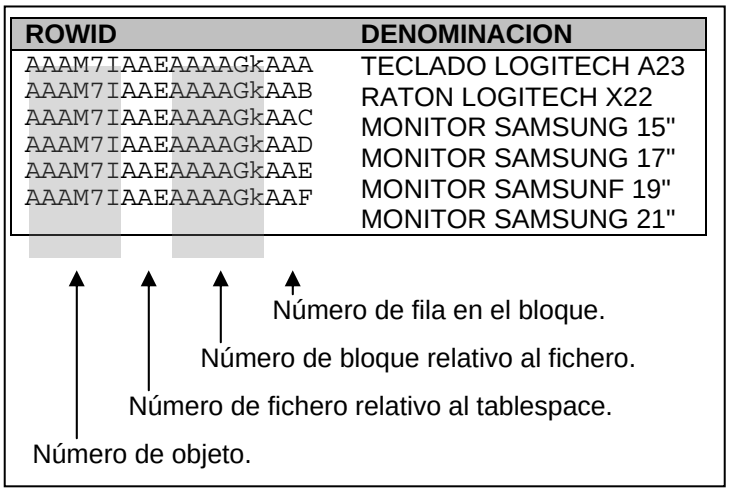
Sabemos que una tabla es la unidad básica de almacenamiento de un sistema relacional.

Cuando se crea una tabla que no forma parte de un *cluster*, el SGBD Oracle le asigna un segmento, en el que posteriormente se almacenarán sus datos. El segmento asignado pertenecerá al *tablespace* indicado en la sentencia de creación de la tabla, aunque, si no se indica nada, pertenecerá al *tablespace* por defecto del usuario que da la orden de creación. Todo usuario definido en una base de datos tiene asignado un *tablespace* por defecto en el que se almacenarán los objetos creados por él cuando no se explicita otro en las correspondientes sentencias de creación.

Como ya se dijo, el segmento asignado a una tabla tendrá inicialmente un número de *extents* igual al valor dado al parámetro *MINEXTENTS.* Este parámetro tiene un valor en cada *tablespace,* que se asigna al crearlo, pero también puede asignársele un valor específico para una tabla en el momento de crearla. El valor de tabla prevalece sobre el de *tablespace*.

El SGBD Oracle almacena las filas de cada tabla en los bloques del segmento asignado; en cada bloque se almacena una o más filas, dependiendo de su tamaño. Cada fila queda identificada por un número, único dentro de la base de datos, que se denomina *ROWID* y que corresponde a su dirección física.

El formato de *ROWID* cambió a partir de la versión 8 de Oracle; en la documentación se utiliza el término ***ROWID* extendido**(***extended rowid***) para referirse al formato actual, y ***ROWID* restringido**(***restricted rowid***) cuando se habla del utilizado en la versión 7 y anteriores. En lo que sigue nos referiremos siempre al ***ROWID* extendido**.



Formato del ROWID de una fila.

EL *ROWID* de cada fila no se almacena físicamente como las columnas de la tabla, pero su valor debe almacenarse como un dato más en ciertos casos como en el de los índices o el de una fila almacenada en varios trozos. Un *ROWID* extendido almacenado ocupa 10 bytes, y se guarda codificado en base 64.

Un *ROWID* está formado por 18 caracteres con el siguiente significado:

- El **número de objeto**, los seis primeros caracteres, que identifica el segmento dentro de todos los de la base de datos.
- El **número de fichero de datos**, los tres caracteres que siguen, que identifica al fichero en el que se encuentra el bloque de datos que contiene la fila. Este número es relativo al *tablespace*.
- El **número de bloque**, los seis caracteres siguientes, que identifica al bloque de datos que contienen la fila dentro de todos los bloques del fichero.
- El **número de fila**, los últimos tres caracteres, que identifican la fila dentro del bloque.

Oracle permite consultar el *ROWID* de cualquier fila mediante lo que se denomina la **pseudocolumna *ROWID.*** Si en una consulta a cualquier tabla se incluye esta pseudocolumna, se obtiene el *ROWID* de cada fila recuperada. Si sobre la tabla ALMACEN realizamos la consulta SQL siguiente.

SELECT ROWID, DENOMINACION FROM ALMACEN

obtendríamos el resultado mostrado en la figura.

El *ROWID* de una fila se mantiene invariable durante el tiempo de existencia de la misma.

Al almacenar una fila, el SGBD graba los valores de cada columna en el mismo orden en que se definieron las columnas al crear la tabla, excepto en el caso de columnas de tipo LONG que van siempre en último lugar. El valor de cada columna va precedido de un campo donde se indica la longitud de la columna en bytes. Oracle soporta columnas de longitud fija y de longitud variable. Las columnas de longitud variable tienen siempre el tamaño justo para almacenar el dato que contengan, por lo que dicho tamaño puede variar cuando se actualiza el dato. Los nulos no se almacenan, de forma que las columnas que contienen nulos solo llevan el campo de longitud con un valor de cero. Si una o más columnas con nulos ocupan los últimos lugares de la fila ni siquiera se almacena la longitud. Cada fila va precedida de un campo de cabecera con información sobre el formato de la fila.

Cuando el tamaño de una fila es tal que no cabe en un bloque el SGBD la parte en dos o más trozos que almacena en distintos bloques. Cada trozo se identifica por su *ROWID* y los distintos trozos quedan lógicamente unidos al almacenar en la cabecera de cada uno el *ROWID* del siguiente.

Al modificar una fila que contenga columnas de longitud variable o con nulos, su tamaño puede aumentar. Si el espacio libre en el bloque no es suficiente para acomodar su nuevo tamaño, la fila se traslada, aunque, para que su *ROWID* no cambie, Oracle dejará en el bloque original un trozo de la fila con un enlace a la nueva dirección física.

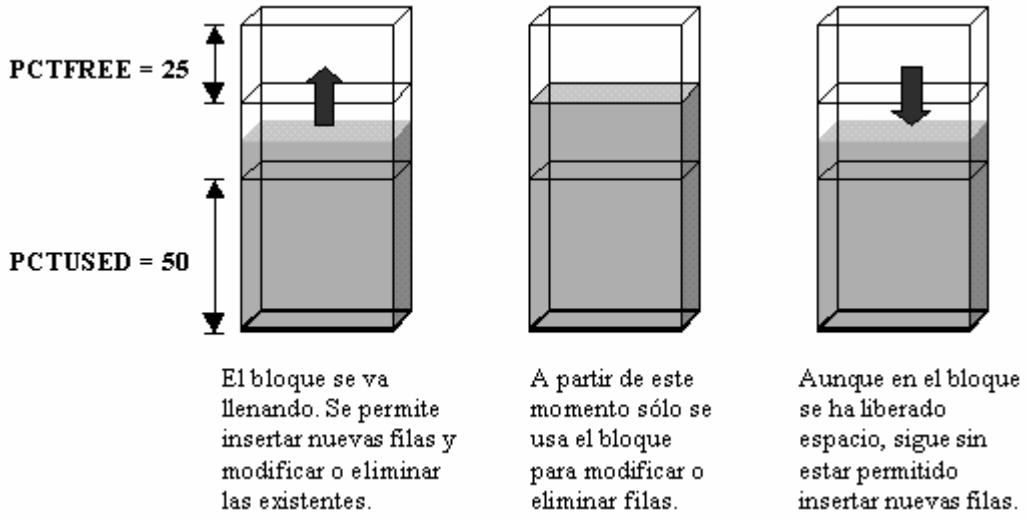
Cuando una fila fue trasladada a otro bloque, el rendimiento del sistema en los accesos a esa fila decrece ya que para localizarla es preciso leer dos bloques; el original, para saber la nueva dirección de la fila, y el nuevo. Y esta doble lectura puede, en algunos casos, necesitar dos accesos a disco. Para evitar en lo posible este problema, ORACLE permite configurar en cada bloque una reserva de espacio utilizable sólo para operaciones de actualización.

En una base de datos Oracle existen dos parámetros, asociados a toda tabla, índice o *cluster*, denominados *PCTFREE* y *PCTUSED*. Estos parámetros tienen unos valores específicos para cada tabla, índice o cluster, que se asignan en el momento de creación del correspondiente objeto, y que, también, pueden ser modificados posteriormente.

El valor de *PCTFREE* marca el porcentaje de espacio del bloque que se reservará para permitir realizar modificaciones en las filas del bloque. Mientras el porcentaje de bloque libre sea superior al valor de este parámetro el SGBD utilizará el espacio libre del bloque tanto para insertar nuevos registros como para modificar los ya existentes. En el momento en el que el porcentaje de espacio libre en el bloque sea igual o inferior al valor de *PCTFREE* ya no se insertarán nuevas filas en él, utilizándose sólo para acomodar los posibles aumentos de tamaño de las filas que se modifiquen. Posteriormente, como consecuencia de borrado de filas o modificaciones que acorten la longitud de la fila, puede aumentar el espacio libre del bloque, pero aunque éste vuelva a ser superior a *PCTFREE*, no se permitirán inserciones hasta que el porcentaje de bloque ocupado sea igual o inferior al valor de *PCTUSED*. En la figura tenemos un ejemplo de uso del espacio de un bloque de acuerdo con los valores de estos parámetros.

- **Índices.**

Los índices son estructuras opcionales asociadas a tablas o *clusters*, que permiten acelerar el acceso a los datos de estos objetos. La existencia o inexistencia de un índice es totalmente transparente a los usuarios y a las aplicaciones que hacen uso de los datos, pero el



Significado de los parámetros PCTFREE y PCTUSED.

SGBD utiliza siempre los índices existentes.

Los índices son lógica y físicamente independientes de los datos de la tabla a la que se asocian. Se puede crear o eliminar en cualquier momento un índice sin afectar a la tabla base ni a otros índices. Los índices, como estructuras independientes que son, necesitan espacio de almacenamiento.

Un índice puede estar definido sobre una o más columnas de una tabla. Una tabla puede tener más de un índice. Por cada fila de la tabla el índice almacenará el valor de la columna o columnas indexadas junto al *ROWID* de la fila. De esta forma, el SGBD leerá el índice para, a

partir de valor de la columna o columnas indexadas, encontrar la dirección física de la fila a la que se desea acceder.

Así como el rendimiento de las consultas a una tabla puede aumentar por la presencia de índices, el rendimiento de las operaciones de modificación, eliminación e inserción de filas puede disminuir debido a que el SGBD debe actualizar los índices después de ejecutar cualquiera de ellas. Por tanto, se debe limitar la definición de índices a aquellos casos en que realmente se consiga una mejora en el rendimiento de las operaciones de consulta. Así, es adecuado definir índices en las columnas cuyos valores se utilizarán para buscar filas específicas de la tabla o para acceder en orden a dichas filas. El SGBD Oracle crea automáticamente un índice asociado a cada columna definida como clave primaria o alternativa.

El SGBD asigna un segmento a cada índice para almacenar sus datos. El *tablespace* del segmento, así como la asignación de extents, se rige por los mismos mecanismos vistos al tratar las tablas.

- **Clusters.**

Un *cluster* es un método alternativo de almacenar tablas con columnas comunes y que se usan juntas con frecuencia. Cuando se crea un *cluster* el SGBD le asigna un segmento y las tablas del *cluster* comparten los bloques de este segmento para almacenar sus datos.

Las columnas comunes de las tablas del *cluster* constituyen la **clave del *cluster*** (*cluster key*, en inglés). El SGBD almacena físicamente juntas, en el mismo bloque, las filas de las tablas con el mismo valor de clave del *cluster*. Con esto se consigue mejorar el tiempo de acceso a los datos cuando se realizan reuniones naturales de las tablas del *cluster* por las columnas que constituyen la clave, ya que las filas relacionadas están almacenadas juntas. También, cada valor de clave se almacena una sola vez, independientemente del número de filas que contengan dicho valor, lo que lleva a un ahorro de espacio de almacenamiento.

Como ocurría con los índices, la existencia de *clusters* es transparente a los usuarios y a las aplicaciones.

Cuando se crea un *cluster* el SGBD le asigna un segmento del *tablespace* por defecto asociado al usuario que lo crea o del *tablespace* indicado en la sentencia de creación. La asignación de *extents* y el uso de los bloques funciona exactamente igual a lo visto para las tablas.

Para poder hacer uso de un *cluster* el SGBD Oracle obliga a definir un índice sobre su clave.

7. Estructuras de memoria y procesos.

Al arrancar un SGBD Oracle, u *Oracle Server,* en cualquier máquina se asigna un área de memoria denominada *System Global Area* (SGA) y se inician varios procesos conocidos como procesos *background* (*background processes*). El conjunto de los procesos *background* y la SGA constituye una **instancia Oracle.** La memoria y los procesos de una instancia gestionan los datos de la base y facilitan el acceso de los usuarios a los mismos.

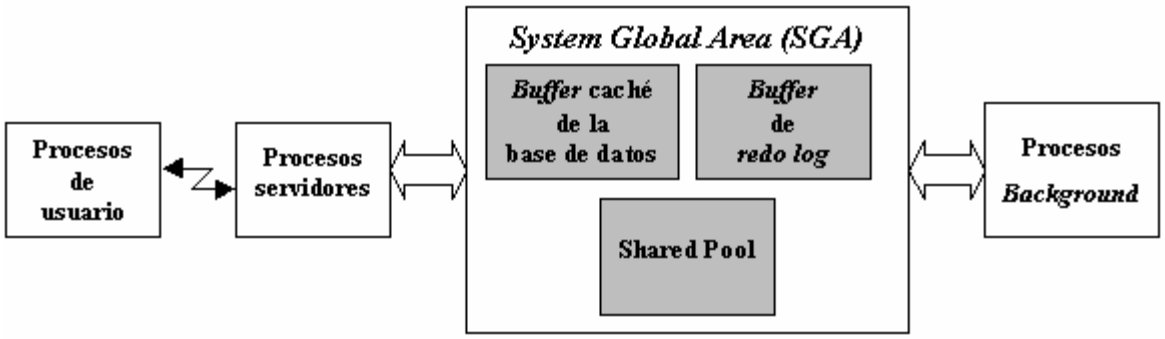
Es posible la existencia de varias instancias ejecutándose en la misma máquina. En este caso, cada instancia accede a su propia base de datos física, es decir, cada una utiliza un conjunto distinto de ficheros de datos, control y *redo log*. También, en sistemas que trabajan en paralelo, es posible la existencia de múltiples instancias accediendo a una sola base de datos.

Además de los procesos *background*, existen otro tipo de procesos Oracle denominados **procesos servidores** (*server processes*). Oracle crea procesos servidores para atender las peticiones de los procesos de usuario, es decir, de los procesos que ejecutan el código de los programas de aplicación. Cualquier operación que desee realizar un proceso de usuario sobre los datos de la base será gestionada por un proceso servidor. Cuando el SGBD arranca un proceso servidor le asigna un buffer en memoria, denominado *Program Global Area* (PGA), para contener los datos y la información de control.

Cuando el proceso de usuario y el proceso servidor que lo atiende residen en distintas máquinas de una red es necesaria la ejecución de SQL*Net tanto en la máquina servidor como en la cliente. SQL*Net es el *software* de Oracle que actúa de interfaz con el protocolo de comunicación usado por las red.

❑ System Global Area (SGA).

La SGA es un área compartida de la memoria del ordenador en el que se ejecuta el SGBD, que contiene datos e información de control de una instancia. Cada instancia tiene su



Estructuras de memoria y procesos.

propia SGA.

Los usuarios conectados a un *Oracle server* comparten la información contenida en las distintas estructuras que componen la SGA. En la figura tenemos representadas las tres subdivisiones de la SGA: el *buffer* caché de la base de datos, el *buffer* de *redo log* y el *shared pool*.

Para mejorar la velocidad de acceso a los datos de la base el SGBD gestiona un sistema de caché, manteniendo en memoria, en el ***buffer* del caché de la base de datos**, aquellos bloques de datos más recientemente usados. Al estar estos bloques en memoria disminuye la necesidad de accesos a disco, ya que los bloques sobre los que se ha realizado recientemente alguna operación son los que con mayor probabilidad intentarán acceder las próximas operaciones. Algunos de estos bloques pueden contener datos modificados que aun no han sido grabados en disco.

El *buffer* de *redo log* mantiene información sobre los cambios realizados a la base de datos. Los procesos servidores se encargan de grabar la información de *redo log* en este *buffer* y un proceso *background* grabará el contenido del *buffer* en un fichero de *redo log*. Como sabemos, esta información es fundamental en la gestión de las transacciones y en la recuperación de la base de datos ante fallos del sistema.

El ***shared Pool*** es la parte de la SGA que mantiene otras estructuras como las **áreas de SQL compartidas** y el **caché del diccionario**.

Para ejecutar cada sentencia SQL de una aplicación de usuario el SGBD necesita un **área SQL compartida.** Esta área contendrá la sentencia compilada y su plan de ejecución. Una misma área SQL compartida es utilizada por múltiples aplicaciones que ejecutan la misma sentencia, con lo que además de ahorrar memoria se mejora el tiempo de ejecución de las sentencias, ya que no es preciso realizar la compilación y la optimización de cada una de ellas. Por cada ejecución de una de estas sentencias que comparten la misma área SQL se mantiene un **área SQL privada** que contiene la información propia de la ejecución concreta. El área SQL privada se mantiene en el *Program Global Area* (PGA), aunque, en ciertos casos, puede estar la SGA. Las aplicaciones pueden manejar las áreas SQL privadas mediante la definición de **cursores.** Como estudiaremos en una unidad posterior, un **cursor** permite controlar la ejecución de una sentencia SQL desde un programa de aplicación.

El **caché del diccionario** es la estructura del *shared pool* que contiene los bloques de datos del catálogo de la base más recientemente accedidos. Como sabemos, el catálogo o diccionario activo de la base de datos es accedido continuamente por el SGBD para ejecutar las sentencias SQL, por lo que la existencia de un caché mejora el rendimiento del sistema.

- **Los procesos background.**

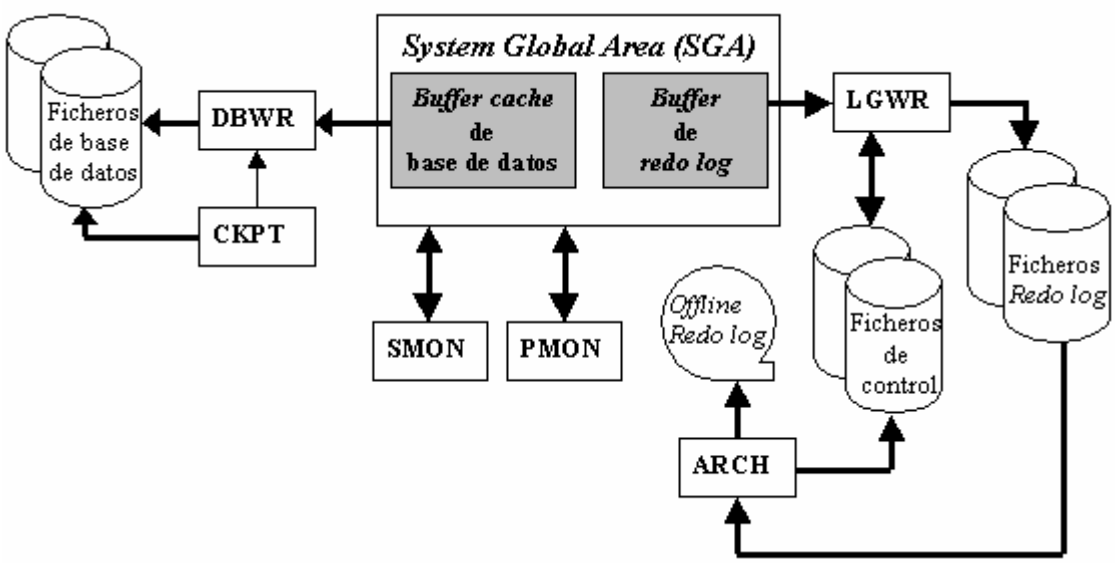
Nada más arrancar una instancia, Oracle crea un conjunto de procesos *background*. Estos procesos se ejecutan en segundo plano (en *background*) mientras existe la instancia, realizando tareas de entrada/salida de datos y de monitorización de otros procesos para conseguir una mejora en el rendimiento y fiabilidad del sistema. Los principales procesos *background* son los siguientes:

El **Database Writer (DBWR)** escribe en los ficheros de datos los bloques modificados del *buffer* caché de la base de datos. El DBWR está optimizado para minimizar las operaciones de escritura en disco. En general, escribe en el disco cuando se precisa introducir nuevos nuevos bloques en el caché y no hay espacio. Los bloques del *buffer* caché que hace más tiempo que no fueron accedidos son los primeros en ser grabados en disco.

El **log Writer (LGWR)** graba la información de *redo log* contenida en *buffer* de *redo log* en el fichero de *redo log* activo. Las escrituras en disco tienen lugar cuando termina una transacción o cuando se llena el *buffer*.

El **Checkpoint (CKPT)** genera, en instantes determinados, un punto de verificación (*checkpoint*) haciendo que el DBWR grabe en los ficheros de datos todos los bloques modificados y no grabados. También, registra en los ficheros de datos y de control el *checkpoint*. CKPT es opcional; si no existe, su labor es realizada por LGWR.

El **Archiver (ARCH)** se ocupa de copiar los ficheros de *redo log* en un dispositivo de archivo. La copia se produce cuando un fichero se ha llenado y deja de estar activo. Este proceso sólo existe en aquellas bases de datos funcionando en modo *ARCHIVELOG*.



Los procesos background.

El **System Monitor (SMON)** realiza la recuperación de la base de datos al arrancar la instancia. Si es necesario, al comenzar la ejecución de la instancia, SMON realiza la recuperación de la base de datos a partir de la información sobre el último punto de verificación registrado en el fichero de control y de la información de *redo log*. Este proceso se encarga también de desasignar segmentos temporales que ya no están en uso, así como de reunir *extents* no usados para conseguir que el espacio libre dentro de los *tablespaces* sea contiguo y más fácil de asignar.

El **Process Monitor (PMON)** se ocupa de la recuperación de procesos cuando se produce el fallo de un proceso de usuario. PMON es responsable de limpiar el caché y de liberar los recursos que el proceso estuviera usando.

8. El diccionario de datos.

Todo SGBD está construido alrededor de un diccionario de datos. En el caso de los sistemas relacionales este diccionario es en realidad un diccionario/directorio gestionado por el propio SGBD, que recibe el nombre de catálogo. El catálogo no sólo sirve al SGBD sino también a los usuarios, que lo pueden consultar utilizando el mismo lenguaje con el que consultan el resto de la base de datos.

Oracle implementa el catálogo mediante un conjunto de tablas y vistas. Las tablas almacenan la información sobre la base de datos y en ellas sólo escribe el SGBD. Los usuarios

no suelen acceder directamente a ellas porque la mayoría de sus datos están almacenados en un formato críptico. Las vistas permiten a los usuarios consultar la información en un formato más adecuado a sus necesidades. Los usuarios de una base de datos Oracle consultan el catálogo utilizando el lenguaje SQL.

Los datos del catálogo son necesarios para el funcionamiento del SGBD. Cuando se ejecuta una operación de manipulación de datos el SGBD accederá a las tablas del catálogo para, por ejemplo, comprobar que el objeto existe y que el usuario tiene privilegios suficientes para realizar la operación. Durante la ejecución de una operación de definición de datos el SGBD actualizará las tablas del catálogo para reflejar la existencia de un nuevo objeto o las modificaciones realizadas a un objeto ya existente.

Todas las tablas y vistas están definidas en el esquema del usuario **SYS**. Existen sinónimos públicos definidos sobre muchas de las vistas para facilitar el acceso a los usuarios. Cuando existe un sinónimo público sobre un objeto de esquema, éste puede ser referenciado por cualquier usuario a través de dicho sinónimo sin necesidad de hacer referencia al esquema en que está definido. El acceso a estas vistas, como a todos los objetos de la base de datos, está controlado por el sistema de privilegios o autorizaciones, por lo que los usuarios sólo podrán consultar aquellas vistas sobre las que tengan los privilegios adecuados.

La mayoría de las vistas del catálogo aparecen en conjuntos de tres que contienen similar información y que se diferencian entre si por el prefijo con que se nombran. Por ejemplo, las vistas USER_TABLES, ALL_TABLES y DBA_TABLES contienen información sobre tablas definidas en la base de datos. La diferencia básica entre ellas está en que si un usuario consulta la primera obtiene información sobre las tablas definidas en su esquema, si consulta la segunda la información obtenida correspondería a todas las tablas accesibles por él, es decir, a todas aquellas sobre las que tiene privilegios, y al consultar la tercera obtendría descripciones de todas las tablas de la base de datos.

En general, la diferencia entre los tres tipos de vistas es la que se deduce del ejemplo anterior, es decir, las vistas con prefijo USER devuelven información referente al esquema del usuario, las de prefijo ALL se refieren a los objetos a los que el usuario puede acceder, y las DBA tratan de todos los objetos de la base. Además, las columnas de los tres tipos de vistas no suelen ser exactamente las mismas. Así, las de prefijo USER suelen omitir la columna OWNER (propietario), dado que se da por supuesto que el propietario es el usuario que realiza la consulta. También, algunas vistas con prefijo DBA suelen contener columnas adicionales con información útil para el administrador.

A continuación damos una lista de las vistas del catálogo que consideramos más interesantes. Si se desea obtener más información sobre estas vistas, o sobre todas las vistas del catálogo, se debe consultar el manual *Oracle Database Reference.*

Vistas	Descripción
DICTIONARY	Descripción de las tablas y vistas que componen el catálogo.
DICT	Sinónimo de DICTIONARY.
DBA_CATALOG, ALL_CATALOG	Información sobre tablas, vistas, sinónimos y secuencias de la base.
DBA, ALL, USER_OBJECTS	Información sobre objetos de la base de datos.
DBA, ALL, USER_TABLES	Descripción de las tablas de la base de datos.
DBA, ALL, USER_VIEWS	Definiciones de vistas.
DBA, ALL, USER_TAB_COLUMNS	Descripción de las columnas de tablas, vistas y *clusters*.
DBA, ALL, USER_CONSTRAINTS	Definición de restricciones definidas sobre tablas.
DBA, ALL, USER_CONS_COLUMNS	Columnas que forman parte de definiciones de restricciones.
TABLE_PRIVILEGES, ALL_TAB_PRIVS, USER_TAB_PRIVS, DBA_TAB_PRIVS	Información de privilegios sobre objetos de la base de datos.
COLUMN_PRIVILEGES, ALL_COL_PRIVS, USER_COL_PRIVS, DBA_COL_PRIVS	Información de privilegios sobre columnas.
USER_SYS_PRIVS, DBA_SYS_PRIVS	Privilegios del sistema concedidos a usuarios y roles.
DBA, ALL, USER_USERS	Información sobre usuarios definidos en la base de datos.
DBA, ALL, USER_INDEXES	Descripción de índices definidos sobre tablas y *clusters*.
DBA, ALL, USER_IND_COLUMNS	Columnas de los índices definidos.
DBA, ALL, USER_SEQUENCES	Información sobre secuencias.

DBA, ALL, USER_SYNONYMS	Información sobre sinónimos existentes en la base.
DBA_CLUSTERS, USER_CLUSTERS	Descripción de los *clusters* definidos.
DBA_CLU_COLUMNS, USER_CLU_COLUMNS	Correspondencias entre columnas de la clave del *cluster* y columnas de las tablas agrupadas en el *cluster*.
DBA_DATA_FILES	Información sobre los ficheros de datos.
DBA, USER_TABLESPACES	Descripción de los *tablespaces*.
DBA, USER_SEGMENTS	Información sobre segmentos.
DBA_ROLLBACK_SEGS	Descripción de los segmentos de *rollback*.
DBA, USER_EXTENTS	*Extents* de los segmentos.
DBA, USER_FREE_SPACE	*Extents* no asignados de los *tablespaces*.

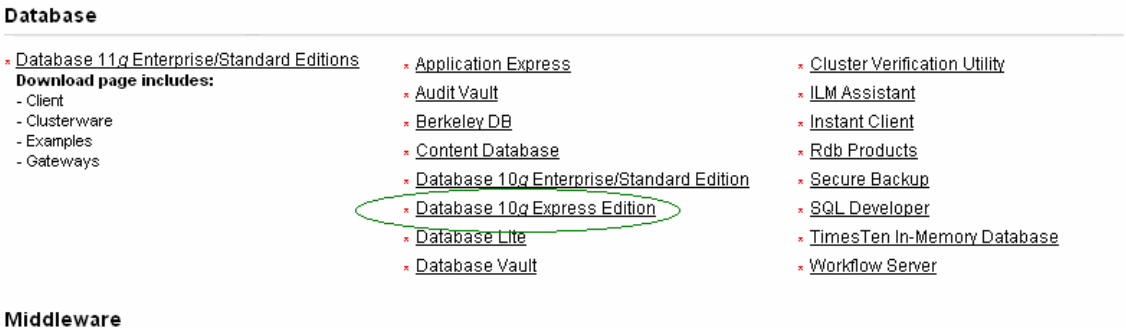
El catálogo contiene también otro tipo de tablas conocido como **tablas de rendimiento dinámico** (*Dynamic Performance Tables*). Estas tablas son continuamente actualizadas por SGBD Oracle durante su funcionamiento, almacenando en ellas información relativa al rendimiento del sistema.

Realmente, las tablas de rendimiento dinámico no son auténticas tablas aunque pueden ser consultadas como si lo fueran. Sobre ellas hay definidas vistas cuyo nombre empieza siempre por el prefijo **V_$**. Los usuarios acceden a estas vistas a través de sinónimos públicos que comienzan siempre con **V$**.

9. Descargar Oracle 10G Express Edition

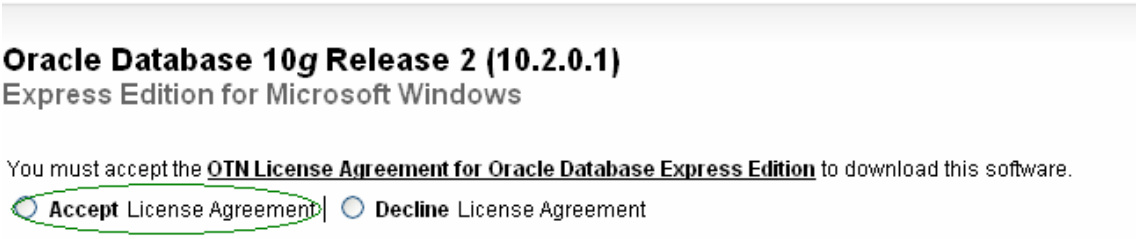
1º Oracle Express es un SGBDR que se puede descargarse de forma gratuita, accedemos a la página oficial de Oracle para España**,** **www.oracle.es**. y seleccionamos **TODAS LAS DESCARGAS.**

2º Seleccionar **Database 10g Express Edition**

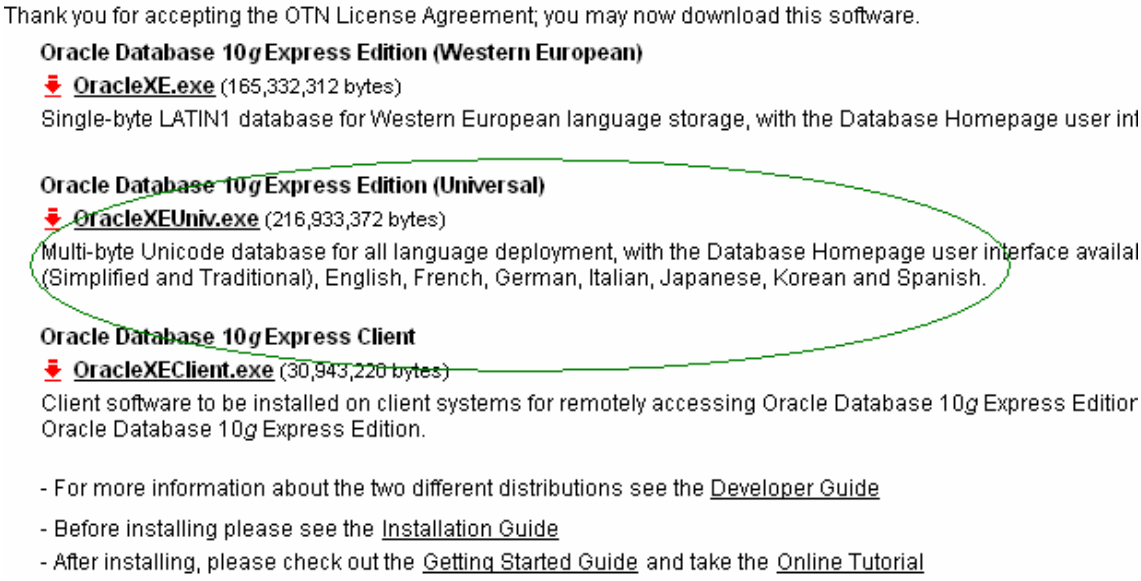


3º seleccionar la instalación para Windows

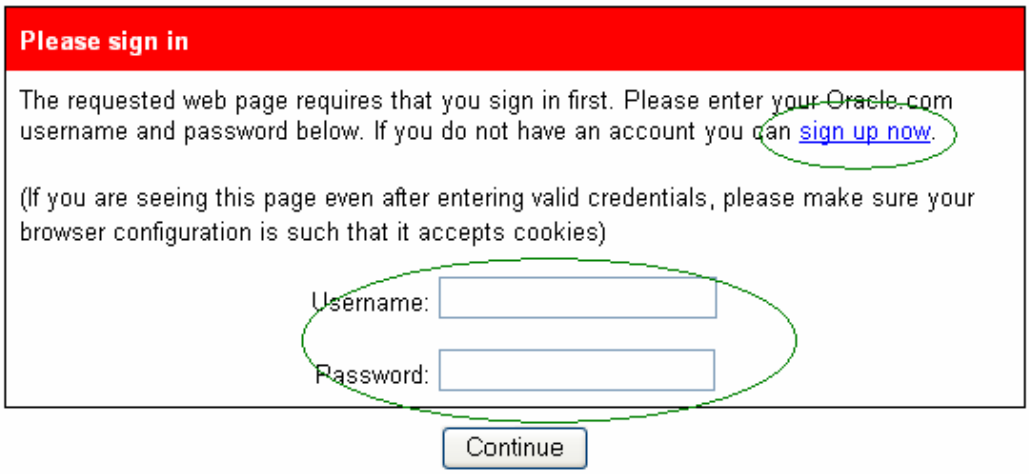
4º Aceptar el acuerdo de licencia



5º Escoger la versión que incluye el español



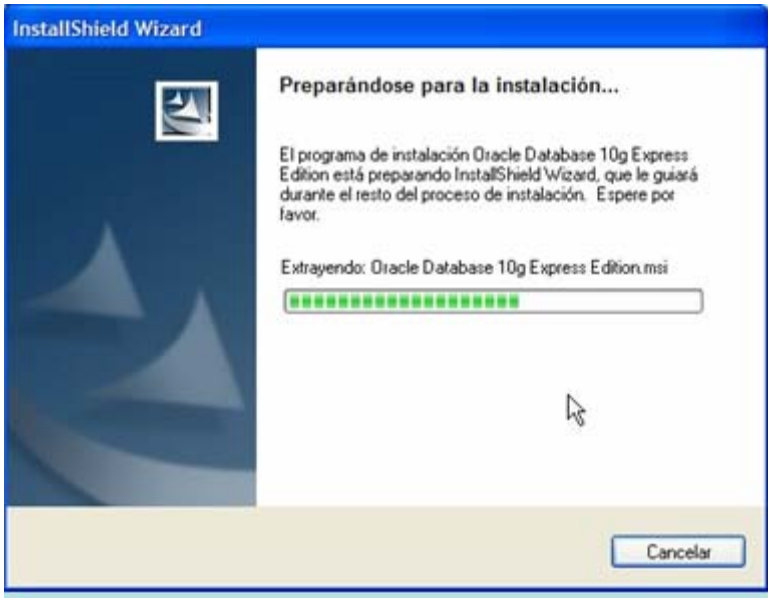
6º Para descargar cualquier software hay que registrarse ***sign up now***. Si ya te registraste en una descarga anterior teclea el nombre tu usuario y la contraseña.



7º Guardar el fichero que se descarga

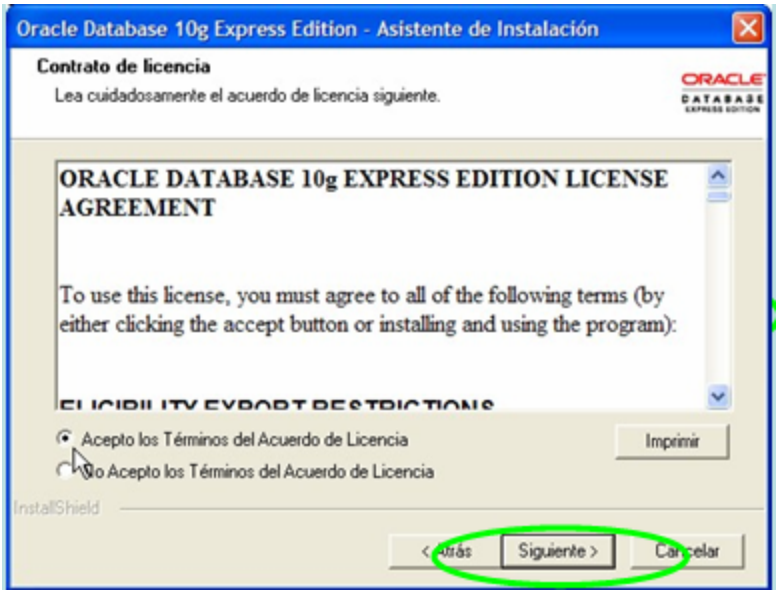
10. Instalar Oracle 10G Express Edition

1º ejecutamos el fichero de instalación que se ha descargado

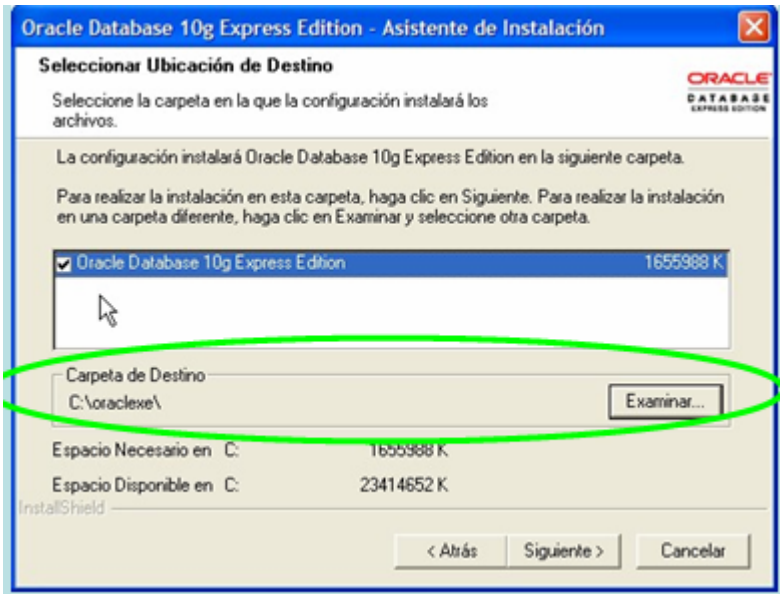


2º En la pantalla que indica que se va iniciar la instalación se pincha en *siguiente*.

3º en la siguiente pantalla aceptamos los términos de acuerdo de licencia y *siguiente*

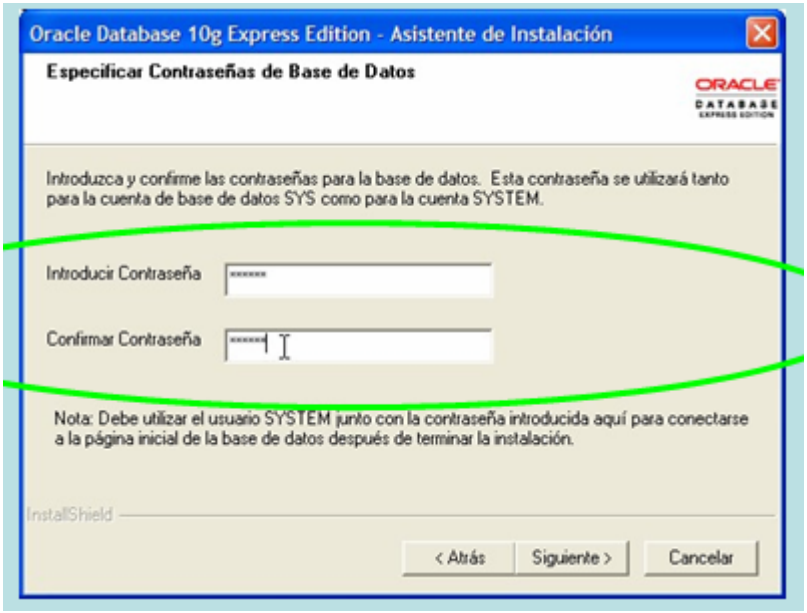


4º Por defecto nos proporciona una carpeta para la instalación. Se puede cambiar

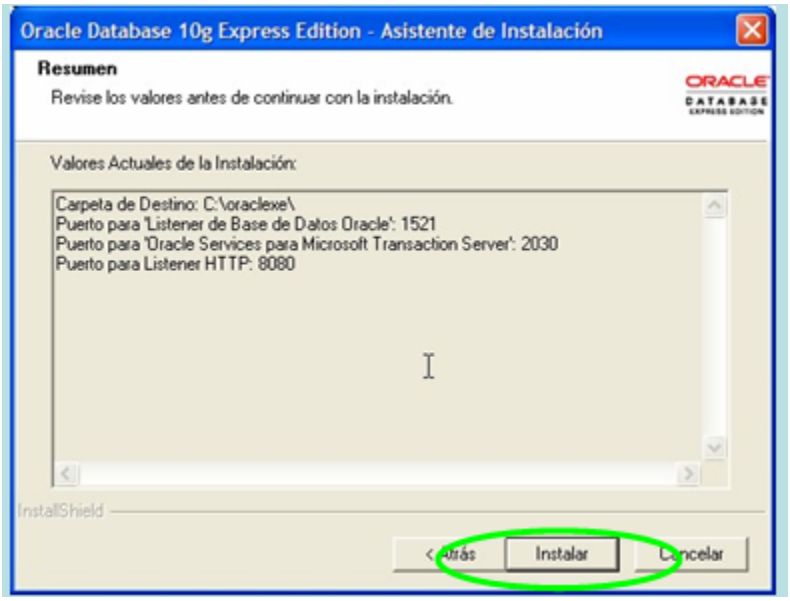


5º Si pide que se introduzca el puerto de listener, dejad el que da por defecto, si el 1521 está ocupado propondrá otro, no lo cambiéis. Lo mismo para los puertos MTS (2030) y HTTP (8080), dejad los que propone.

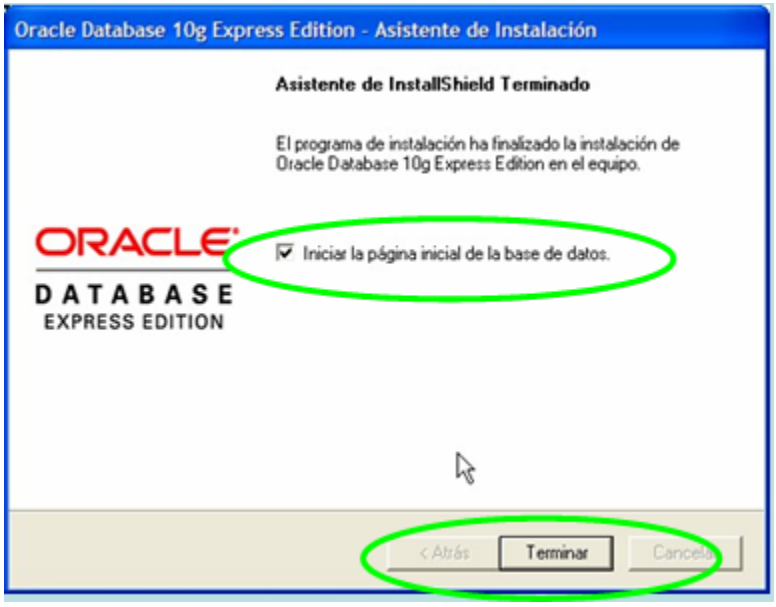
6º Pide la contraseña para el usuario SYSTEM. Es el administrador así que acordaros de la que ponéis



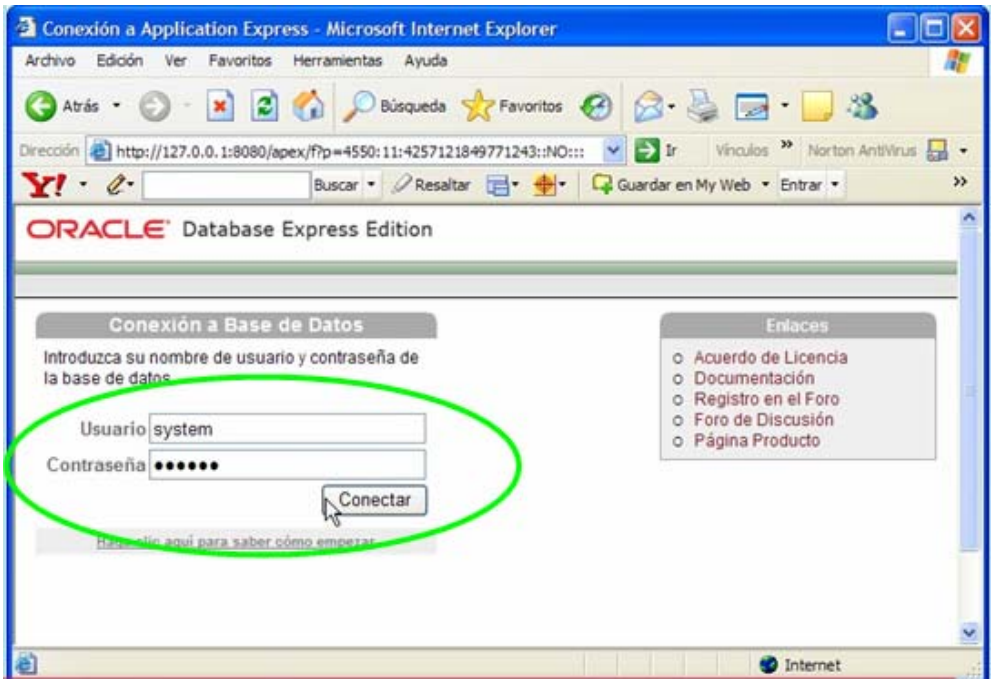
7º Si todo está bien, pinchaís en instalar



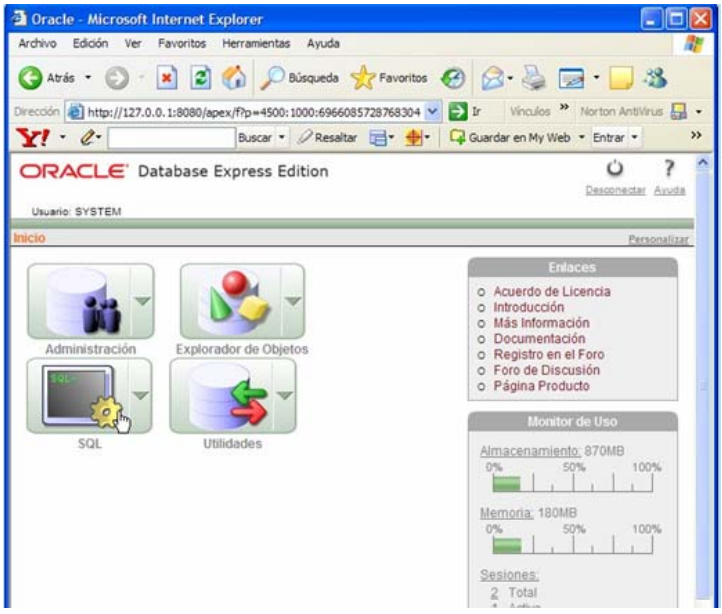
8º cuando termina la instalación, por defecto nos invita a ir a la página de inicio de la base de datos. Se pulsa terminar y nos lleva a la pagina de inicio.



9º Se accede utilizando el usuario SYSTEM y la palabra de paso que le hayamos dado.



10º La página que se muestra a continuación es la inicial de la base de datos. El interface con Oracle se hace a través de páginas web.

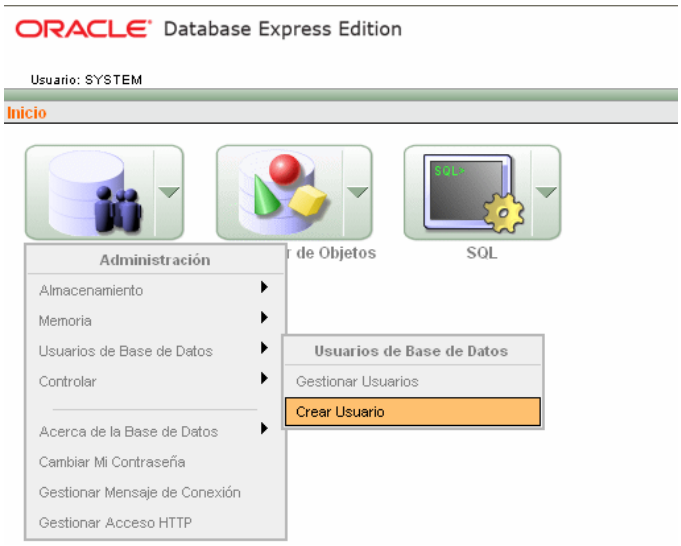


11. Trabajando con Oracle 10G

Inicialmente crearemos una nueva cuenta de usuario en cuyo esquema almacenaremos los objetos para dicho usuario.

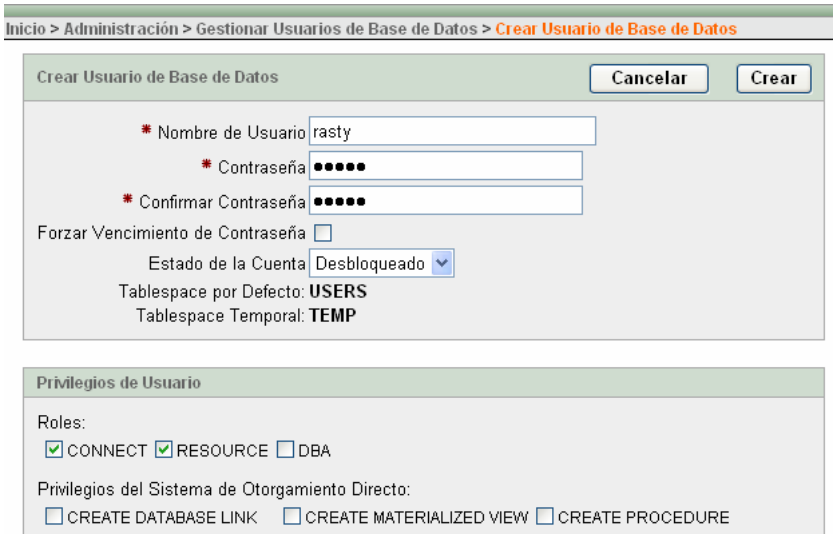
11.1. Crear un nuevo usuario

Para crear un usuario hay que ser administrador. Si no estáis conectados como SYSTEM, lo primero que debéis hacer es conectaros. Desde la herramienta administración crear el usuario, lo llamaremos ***rasty***

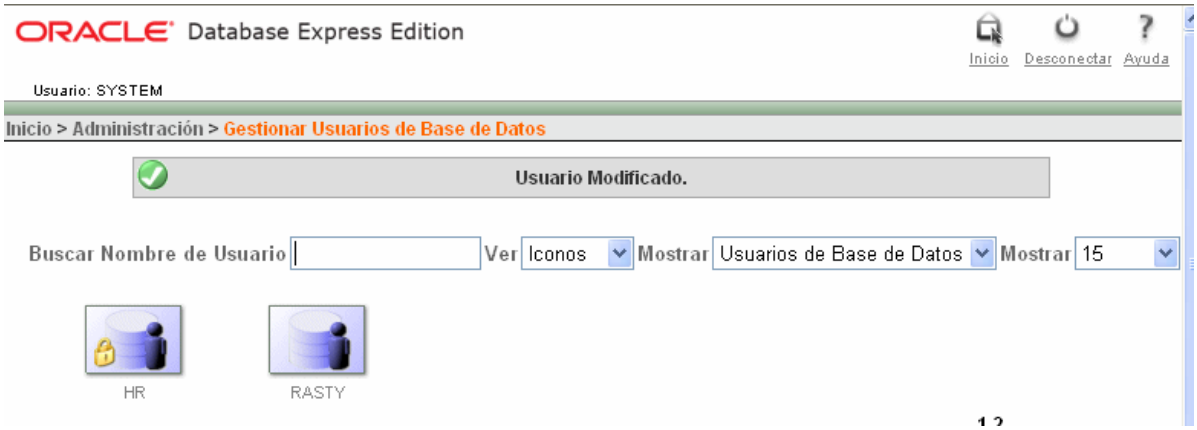


Le damos el nombre rasty y la contraseña, el resto de los parámetros los dejamos por defecto. Fijaros que los roles son CONNECT y RESOURCE, permiten crear todo tipo de objetos en nuestro esquema salvo las vistas, dar derechos sobre nuestros objetos a otros usuarios, etc No podemos administra (crear usuarios, cambiar parámetros de almacenamiento, gestionar usuario, etc.) Para poder crear vistas marcar CREATE VIEW.

Si lo preferís, como trabajáis en local podéis seleccionar DBA y así tenéis todos los derechos.



La página que se muestra después de crear es la siguiente

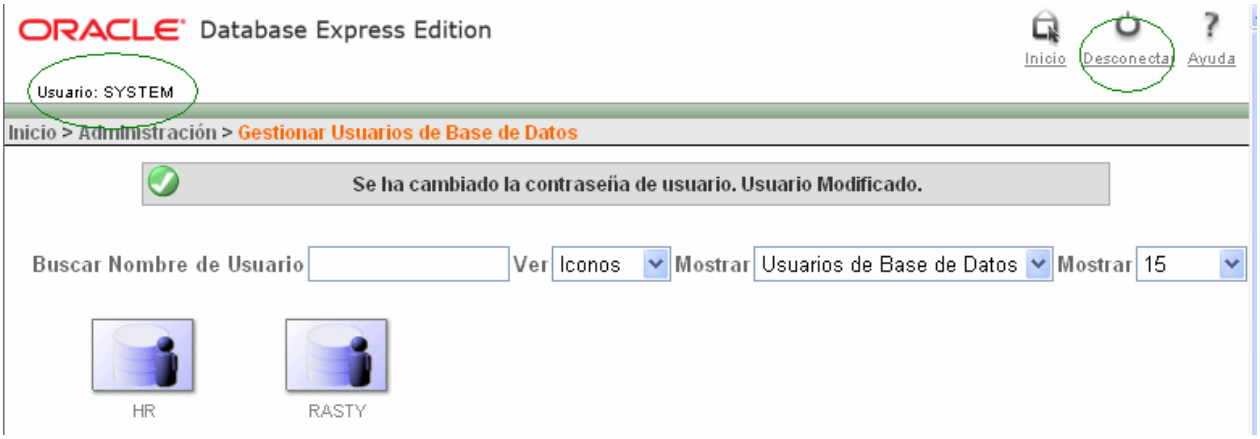


Veis que además de RASTY hay un usuario que se llama HR, este usuario se creó durante la instalación, tiene un candado, quiere decir que está bloqueado, hay que desbloquearlo, se pincha en él, la contraseña es ***hr***, se quita el bloqueo y seleccionar modificar.

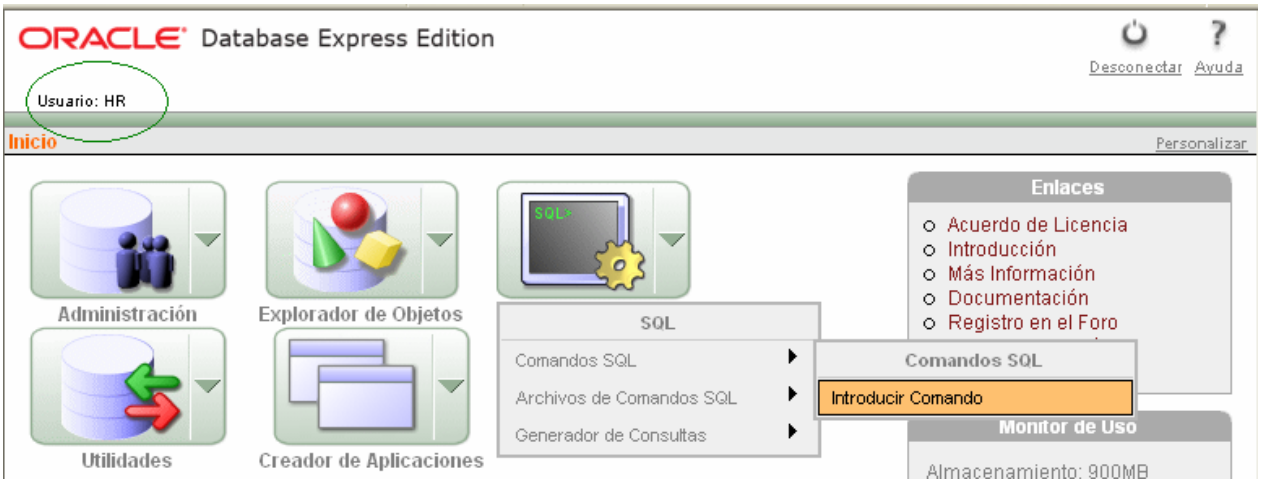
En el esquema de HR existen una serie de objetos (tablas) con sus correspondientes filas. El de RASTY está vacio.

11.2. Trabajando con usuarios

Desconectarse de SYSTEM y conectarse como HR.



Una vez conectados como HR



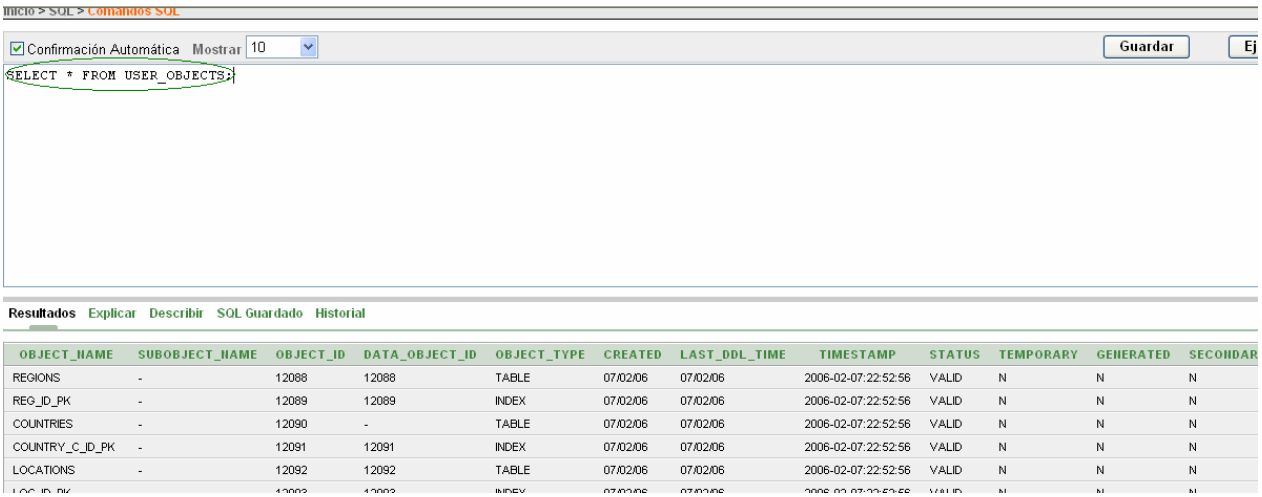
Entramos en la página que nos permite escribir sentencias SQL. Seguimos la secuencia que se ve en la imagen *(sql → comandos sql → introducir comando)*

La página tiene dos ventanas, una para escribir las sentencias SQL y en la otra se muestran los resultados.

Vamos visualizar los objetos que hay en el esquema de HR. Escribimos la siguiente sentencia y pulsamos **ejecutar**:

```
SELECT * FROM USER_OBJECTS;
```

Donde **USER_OBJECTS** es la vista del catálogo vista en el epígrafe *EL DICCIONARIO DE DATOS* que muestra los objetos del esquema del usuario.



OBJECT_NAME	SUBOBJECT_NAME	OBJECT_ID	DATA_OBJECT_ID	OBJECT_TYPE	CREATED	LAST_DDL_TIME	TIMESTAMP	STATUS	TEMPORARY	GENERATED	SECONDAR
REGIONS	-	12088	12088	TABLE	07/02/06	07/02/06	2006-02-07:22:52:56	VALID	N	N	N
REG_ID_PK	-	12089	12089	INDEX	07/02/06	07/02/06	2006-02-07:22:52:56	VALID	N	N	N
COUNTRIES	-	12090	-	TABLE	07/02/06	07/02/06	2006-02-07:22:52:56	VALID	N	N	N
COUNTRY_C_ID_PK	-	12091	12091	INDEX	07/02/06	07/02/06	2006-02-07:22:52:56	VALID	N	N	N
LOCATIONS	-	12092	12092	TABLE	07/02/06	07/02/06	2006-02-07:22:52:56	VALID	N	N	N

El resultado de la consulta nos muestra los nombres de los objetos y sus características.

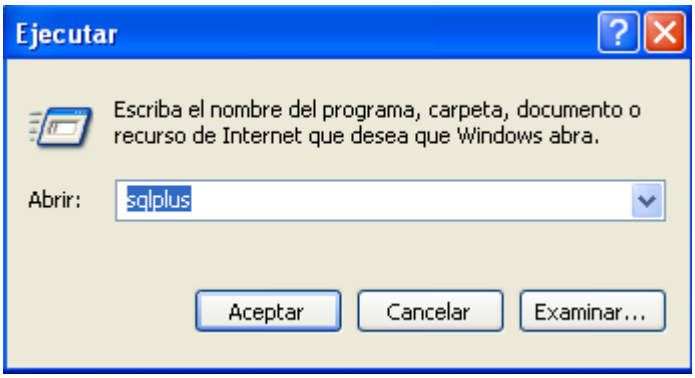
Podéis practicar con otras vistas del catálogo comentadas en el epígrafe *EL DICCIONARIO DE DATOS.*

11.3. Cargando tablas

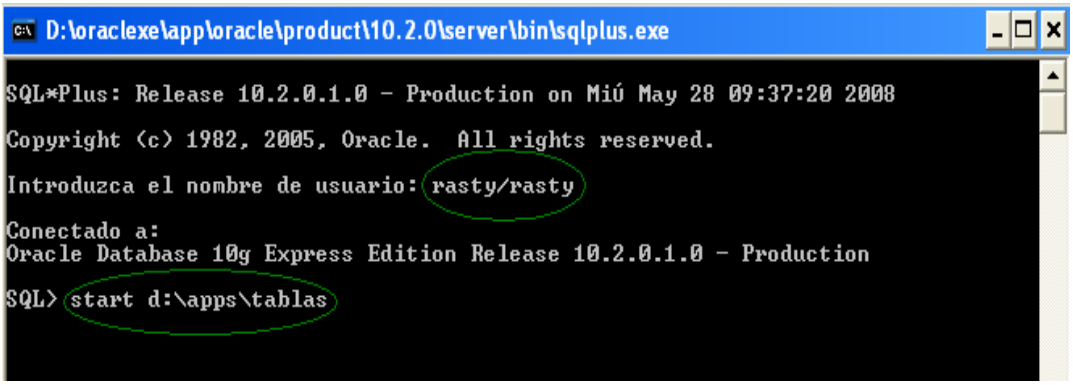
Vamos cargar en el usuario RASTY las tablas con contenido que se detallan en el fichero TABLAS.SQL Estas tablas se utilizarán en la siguiente unidad de trabajo.

Hay diferentes soluciones que permiten introducir las tablas en el esquema de RASTY, emplearemos la más simple. Consiste en abrir una ventana **SqlPlus** y ejecutar el comando ***Start***. Las siguientes capturas muestran el proceso.

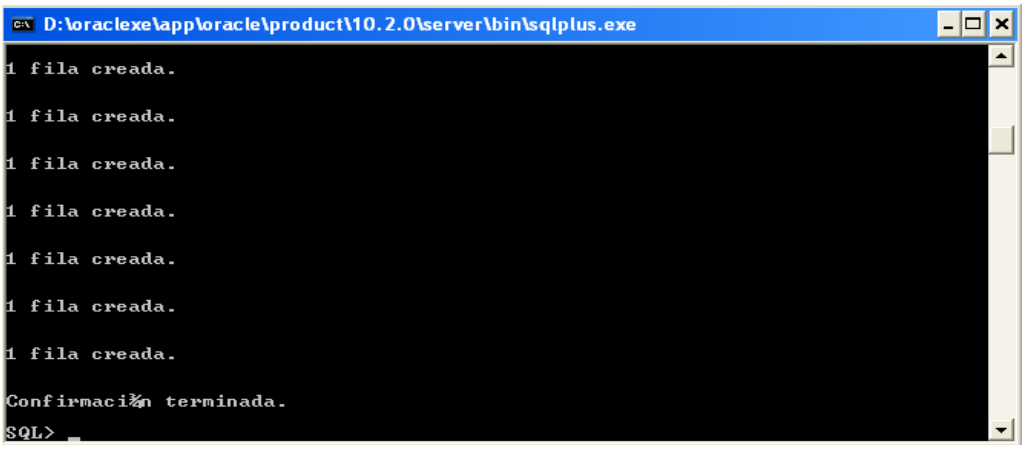
1º abrir la ventana SQLPLUS: *Inicio→ejecutar__>sqplus*



2º Conectarse como *rasty/rasty.* Lanzar el fichero TABLAS.SQL con el comando START. Si el fichero no tiene extensión *sql* hay que llamarlo con la extensión. Cambiar el directorio *d:\apps*\ por el vuestro



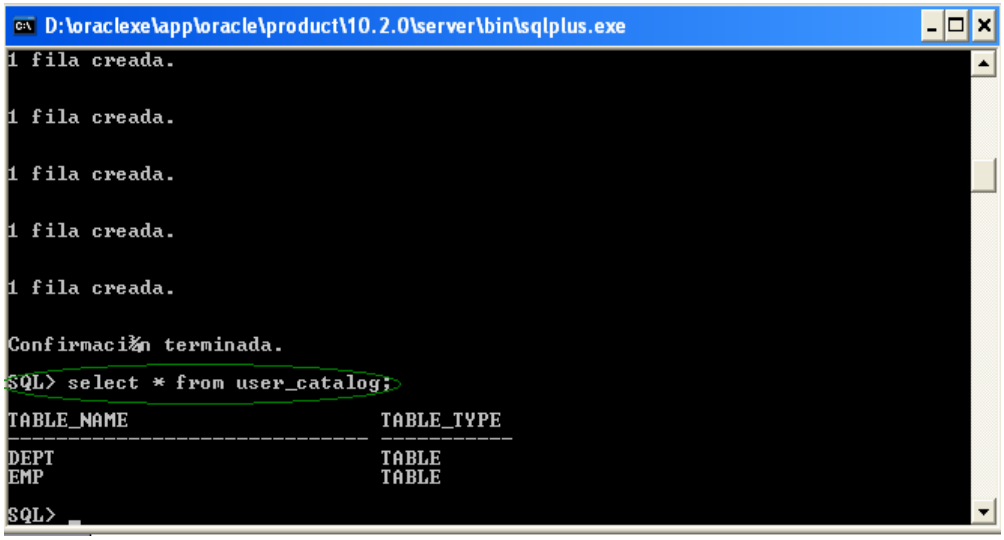
3º Si no visualizó errores y aparece la siguiente pantalla es que las creó correctamente



También podéis comprobar, desde aquí o desde la página web, que existen las tablas ejecutando la siguiente sentencia

```
SELECT * FROM USER_CATALOG;
```

Donde USER_CATALOG es una vista del catálogo



Introducción

En las unidades didácticas anteriores hemos representado la información mediante el **modelo relacional**. ¿De qué manera podemos almacenar y tratar la información esquematizada en el modelo relacional por medio de un Sistema Gestor de Bases de Datos? La respuesta a esta pregunta se llama **SQL**.

En esta unidad se explican los fundamentos del SQL estándar, cómo utilizarlo de forma interactiva, y se sientan las bases para su utilización en PL/SQL. Es importante tener una cierta soltura en el manejo de SQL, pues nos permite obtener de forma rápida cualquier información que tengamos en las tablas de la base de datos, no solo en las tablas del usuario, sino también en el diccionario de datos.

El lenguaje SQL lo puede utilizar el usuario final como un recurso eficaz para obtener datos de diferentes tablas, el programador para confeccionar los módulos PL/SQL y el administrador para acceder a las tablas del diccionario y así poder tomar decisiones que optimicen la explotación de la base de datos.

Objetivos

- Conocer en qué consiste el estándar SQL como lenguaje para acceder a los datos de las tablas.
- Comprender los distintos objetos que constituyen el lenguaje SQL.
- Utilizar las sentencias, operadores y funciones que constituyen la base de SQL.
- Trabajar con múltiples tablas y vistas.

Contenidos Generales

1. HISTORIA Y EVOLUCIÓN 3
2. CARACTERÍSTICAS DEL LENGUAJE SQL 4
2.1. Sentencias SQL *5*
2.2. Tipos de datos. *6*
2.3. Operadores, expresiones y funciones. *8*
3. SQL Y EL MODELO RELACIONAL. 13
4. LENGUAJE DE MANIPULACIÓN DE DATOS (LMD) 16
4.1. Recuperación de información. Sentencia SELECT *16*
4.2. Operadores tradicionales de conjuntos (UNIÓN, INTERSECCIÓN, DIFERENCIA) *25*
4.3. Resumen o agrupación (GROUP BY) *26*
4.4. Consultas en múltiples tablas (REUNIÓN O COMBINACIÓN) *29*
4.5. Reuniones externas *30*
4.6. Los ALIAS (variables de filas explícitas) *31*
4.7. Producto cartesiano generalizado *32*
4.8. SELECT anidadas (subconsultas) *32*
4.9. Añadiendo filas. INSERT *34*
4.10. Actualizar columnas. UPDATE *35*
4.11. Borrar filas DELETE *36*

1. Historia y evolución

Las primeras implementaciones en SQL se realizaron en 1975, con el nombre de SEQUEL. En 1979 ORACLE implementa el primer sistema comercial de gestión de bases de datos relacionales basado en SQL , posteriormente aparece otros sistemas también basados en SQL como DB2, SYBASE, INTERBASE, INFORMIX, etc, y otros productos que no lo poseían como lenguaje ofrecen un interface con el mismo, por ejemplo, INGRES, DATACOM, ADABAS, etc.

El comité de bases de datos X3H2 de ANSI, aprueba en 1986 una normalización del lenguaje pasando a denominarse SQL/ANS que también es aprobado al año siguiente como norma ISO. En un principio esta normalización es criticada por algunos autores, aduciendo que fallan numerosas características que los usuarios necesitan y que se contemplan en el modelo relacional.

En 1989 se publica una nueva versión del estándar ISO que añade ciertas reglas de claves ajenas que permiten definir la opción de modificación y borrado restringido. Por otra parte ANSI define ese mismo año un estándar para el lenguaje SQL embebido. En 1992 se aprueba una nueva versión de SQL conocida como SQL2 o SQL-92 , en la que se incrementa la capacidad semántica del esquema relacional y se añaden nuevos operadores, también se incluyen normas para el SQL embebido.

Aunque SQL-92 ha mejorado considerablemente la capacidad semántica (por ejemplo con la definición de dominios, como se describe en la Unidad 2) aún ha de evolucionar para adaptarse a las nuevas tendencias, como es la orientación a objetos. En este sentido, los comités ANSI/ISO están trabajando en la definición de una nueva versión del estándar SQL denominada SQL3 que amplia la capacidad semántica (por ejemplo, con la incorporación de disparadores definidos con la sentencia CREATE TRIGGER) e incluye bastantes de las características de los lenguajes orientados a objetos como son los tipos abstractos de datos, encapsulamiento, herencia, polimorfismo, métodos. Está previsto que SQL3 se normalice en 1999, y ya se está hablando de SQL4 que incluiría aquellas características del lenguaje que necesiten una mayor profundización.

SQL (Structured Query Language) es un lenguaje que permite realizar una serie de operaciones en bases de datos relacionales. Se compone de tres partes o sublenguajes, el LDD

o lenguaje de definición de datos, LMD o lenguaje de manipulación de datos, LCD o lenguaje de control de datos.

Como lenguaje de manipulación relacional es un lenguaje de especificación que permite seleccionar un conjunto de filas sobre las que se va a realizar una acción recuperación o actualización).

Su principal característica es la de ser un estándar en la manipulación y definición de las tablas de los sistemas gestores de bases de datos relacionales. No es un lenguaje en si mismo, en el sentido que se pueda adquirir independientemente, forma parte del núcleo del sistema gestor de la base de datos. Es utilizado por usuarios finales, interactivamente o bien a través de herramientas que lo incorporan, por programadores también de forma interactiva o embebido en lenguajes de tercera o cuarta generación, y por los administradores para crear los objetos de la base (tablas, usuarios, etc) conceder permisos, interrogar el diccionario de datos, etc.

Otra característica es su portabilidad a través de diferentes sistemas, incluso se puede utilizar como pasarela cuando en una red se dispone de diferentes productos de bases de datos.

En resumen el SQL estándar no es un lenguaje completo como el COBOL, PASCAL…., no dispone de sentencias propias de los lenguajes mencionados como IF, PERFORM, FOR …, pero si se le puede incorporar como lenguaje embebido a otro lenguaje denominado anfitrión para permitir el acceso a través de dichos lenguajes a las bases de datos. En algunos casos se le añaden las estructuras necesarias para convertirlo en un lenguaje específico de una herramienta como es el caso del PL/SQL.

Resumiendo, con SQL podemos:

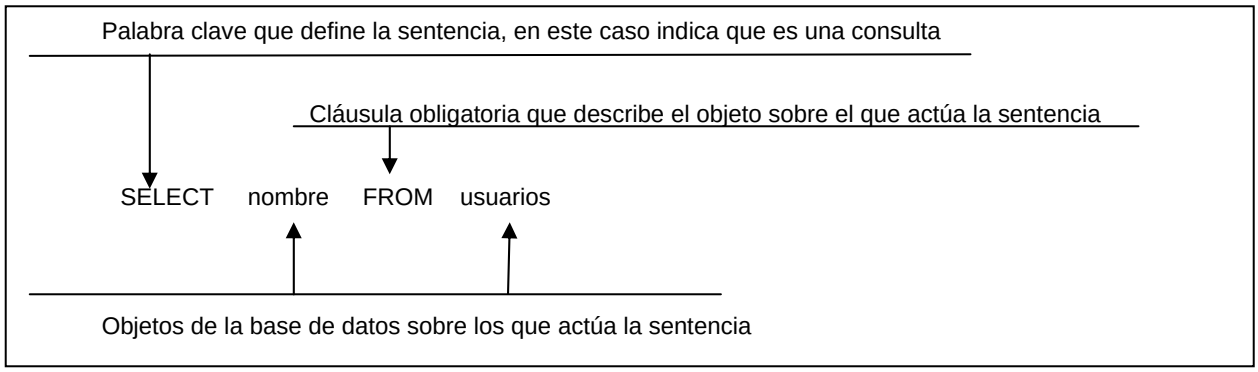
- Definir y eliminar objetos
- Conceder y denegar autorizaciones
- Consultar y actualizar datos.

2. Características del lenguaje SQL

En lo que sigue vamos a describir los distintos elementos de SQL según el estándar ANSI/ISO, pero debemos advertir que los productos comerciales que implementan el estándar suelen incorporar algunas extensiones y variaciones con respecto a él, y que nosotros nos referiremos siempre al SQL implementado por Oracle.

2.1. Sentencias SQL

Las sentencias SQL se componen de una palabra clave, que también podemos llamar palabra reservada, que las definen y de una o varias cláusulas (algunas obligatorias otras opcionales) que túan sobre los objetos o elementos de la base de datos. Como en cualquier otro lenguaje se debe evitar utilizar estas palabras reservadas para nombrar otros objetos de la base de datos



Teniendo en cuenta las tres partes o sublenguajes de SQL podemos hablar de:

Sentencias de manipulación de datos.

Permiten manipular y consultar la información contenida en las tablas. Las consultas son las operaciones más realizadas a través del lenguaje SQL, muchos usuarios utilizan SQL únicamente como herramienta de consulta. (usuarios finales, programadores y administradores). Las siguientes sentencias pertenecen a este grupo:

SELECT	recupera datos
INSERT	añade nuevas filas
DELETE	elimina filas
UPDATE	modifica el contenido de las filas

Sentencias de definición de datos.

Permiten crear y borrar tablas y vistas así como modificar la estructura de las tablas. Estas sentencias, básicamente, las utilizan los administradores y los programadores. Son sentencias de este grupo:

CREATE	crea tablas y vistas
DROP	borra tablas, vistas, índices…
ALTER	modifica la estructura de una tabla

Sentencias de control de datos.

Se utilizan para gestionar la confidencialidad, por ejemplo las sentencias GRANT y REVOKE, y para gestionar las transacciones, como COMMIT y ROLLBACK.

Algunos autores incluyen las sentencias de control junto con las de definición de datos. No tiene mayor importancia que se incluyan en el grupo de definición o que constituyan un grupo aparte, es una mera clasificación.

2.2. Tipos de datos.

El estándar ANSI/ISO define una serie de tipos de datos que suelen ser un subconjunto de los que incorporan la mayoría de las bases de datos, en nuestro caso, enumeraremos los tipos de datos del estándar y luego trabajaremos con los tipos que incorpora Oracle.

Los tipos de datos los podemos agrupar en: numéricos y alfanuméricos.

Numéricos: Pueden contener números con o sin punto decimal y signo.

- Números enteros	INTEGER	
- Números enteros pequeños	SMALLINT	
- Números decimales	DECIMAL(p,s)	p=precisión s=escala
	NUMERIC(p,s)	
- Números en coma flotante	FLOAT(p)	

- De baja precisión REAL
- De alta precisión DOUBLE PRECISION

Alfanuméricos: Almacenan cadenas de caracteres.

- Cadena de longitud fija CHAR(n) n=longitud
  CHARACTER(n)
- Cadena de longitud variable CHARACTER VARYING(n)

Tipos de datos en Oracle

La mayor parte de las bases de datos tienen un conjunto de datos más amplio que los especificados por ANSI/ISO. En el siguiente cuadro definimos los tipos de datos de Oracle que vamos a utilizar en esta Unidad.

Tipo	Descripción
VARCHAR2(n)	Cadena de longitud variable.
CHAR(n)	Cadena de longitud fija, *n* es opcional si se omite el asume 1 byte. La máxima longitud es de 255 bytes
LONG	Cadena de longitud variable, 2 GB como máximo
RAW(n)	Información binaria, la longitud máxima de *n* es de 255 bytes
LONG RAW	Información binaria, hasta dos gigabytes, no se pueden indexar las columnas de este tipo. Se utiliza para almacenar información gráfica, sonido, etc
NUMBER(p,s)	Datos numéricos, donde *p* es la precisión con un rango desde 1 hasta 38 y *s* es la escala con rango desde –84 hasta 127
DATE	Almacena datos tipo fecha y hora
CLOB	Cadena de longitud variable, a partir de 4 GB como máximo
BLOB	Datos binarios no estructurados, 4 GB como máximo
BFILE	Datos binarios almacenados en archivos externos a la base, 4GB como máximo

2.3. Operadores, expresiones y funciones.

A continuación se describen los operadores aritméticos, de comparación, lógicos , de cadena y algunas funciones que iremos utilizando en el desarrollo de los siguientes puntos al tiempo que describimos las sentencias.

Si el lector está utilizando una base de datos que no sea Oracle, puede que algún operador o función de las que se describe no se ejecute correctamente; en este caso debe de consultar el manual del SQL para su base de datos.

Los operadores se usan para manipular datos y devolver un resultado. La combinación de operadores aritméticos, valores numéricos y funciones numéricas dan lugar a expresiones aritméticas. Si la combinación es de operadores de cadena, funciones de cadena y constantes o variables alfanuméricas, se obtienen expresiones de cadena o alfanuméricas. Por último si intervienen operadores de relación y/o lógicos, las expresiones son condicionales y el resultado es un único valor TRUE o FALSE.

Operadores aritméticos por orden de precedencia	**Descripción**	**Ejemplos**
+ -	Proporciona carácter positivo o negativo a una expresión	- 1
* /	Multiplicación y división	PTS * HORAS
+ -	Sumar y restar	SUELDO + COMISION

Operadores comparación	Descripción	Ejemplos
=	Comprobación de igualdad	SUELDO = 10000
!= <>	Distinto de…..	SUELDO <> 5000
>	Mayor que….	SUELDO > COMISION
<	Menor que…	COMISION < SUELDO
>=	Mayor o igual	SUELDO >= PRIMAS
<=	Menor o igual	SUELDO <= 400000
IN	Igual a cualquier elemento de la lista, si lo encuentra evalúa TRUE	**IN** ('ANALISTA','OPERADOR')
NOT IN	Evalúa FALSE si está en la lista	**NOT IN** (lista de valores)
ANY, SOME	Se compara con cada uno de los valores de una lista. Deben de ir precedidos de alguno de los siguientes operadores =, !=, >, <, <=, >= El operador IN equivale =ANY Evalúa TRUE cuando cumple la condición con cualquiera de los elementos de la lista.	SUELDO = ANY (100, 200) ó SUELDO = SOME(100, 200) La lista se puede sustituir por los valores devueltos por una consulta
ALL	Compara un valor con cada uno de la lista o los devueltos por una consulta. Debe ir precedido por uno de los siguiente operadores =, !=, >, <, <=, >=	SUELDO >= **ALL** (1000,2000)

	Evalúa TRUE cuando cumple la condición para todos y cada uno de los elementos de la lista.	
[NOT] BETWEEN ***A*** AND ***B***	Evalúa TRUE si el valor está comprendido entre A y B ambos inclusive	SUELDO **BETWEEN** 2000 **AND** 3000
EXISTS	Evalúa TRUE si devuelve alguna fila	Pendiente de ver las consultas **SELECT**
A [NOT] LIKE ***B***	Evalúa TRUE si A coincide con B, siendo B una constante que puede incorporar los caracteres comodin % que sustituye cualquier cadena y _ que sustituye un carácter	NOMBRE **LIKE** 'MAR%' Devuelve **TRUE** si NOMBRE contiene un valor que comience por MAR (MARIA, MARIANA, …)
IS [NOT] NULL	Testea si un item es o no nulo	DIRECCION **IS NULL**

Operadores lógicos	**Descripción**	**Ejemplos**
NOT	Evalúa TRUE si la condición es falsa	NOT (SUELDO > 10000)
AND	Evalúa TRUE si ambas condiciones son verdaderas	(SUELDO > 1000) **AND** (COMISION < 2000)
OR	Evalúa TRUE si al menos una de las condiciones es verdadera	(SUELDO > 1000) **AND** (COMISION < 2000)

Operador de cadenas	**Descripción**	**Ejemplos**
||	Concatena dos cadenas	'sema'||'na' → 'semana'

Funciones numéricas	**Descripción**	**Ejemplos**
ABS(n)	Valor absoluto de *n*	**ABS**(-15) → 15
ACOS, ASIN, ATAN, COS, LN, LOG, SIN, TAN	Funciones trigonométricas	
CEIL(n)	Devuelve el entero mas pequeño mayor o igual que *n*	**CEIL**(12.4) → 13
FLOOR	Devuelve el entero más grande menor o igual que *n*	**FLOOR**(12.4) → 12
MOD(m,n)	Devuelve el resto de dividir *m* por *n*	**MOD**(7,2) → 1
POWER(m,n)	Eleva *m* a *n*	**POWER**(4,2) → 16
ROUND (n[,m])	Redondea *n.* Si *m* no es cero redondea a *m* decimales	**ROUND**(12.19,1)→ 12.2
SIGN(n)	Devuelve –1 si *n*<0, 0 si *n*=0, 1 si *n*>0	**SIGN**(10) → 1
SQRT(n)	Devuelve la raíz cuadrada de *n*	**SQRT**(16) → 4
TRUNC(n[,m])	Trunca *n*. Mismo tratamiento que la función ROUNDED, salvo que no redondea	**TRUNC**(12.19,1) → 12.1
Funciones alfanuméricas	**Descripción**	**Ejemplos**
CHR(n)	Devuelve el carácter del código ASCII	**CHR**(65) → A
CONCAT(A, B)	Concatena las cadenas o variables A y B. Equivale a ||	
INITCAP(A)	Convierte a mayúsculas la	**INITCAP**('la casa') → La Casa

	inicial de cada palabra contenida en argumento	
LOWER(A)	Convierte en minúsculas el argumento	**LOWER**(‘ CASA’) → casa
LPAD(A,n [,B])	Devuelve A ajustada a la derecha con relleno por la izquierda con B hasta completar *n*	**LPAD**(‘casa’,7,’*’) → ***casa
LTRIM(A, B)	Elimina empezando por la izquierda del argumento A los caracteres que coincidan con el argumento B hasta que encuentre uno diferente.	**LTRIM**(‘ababcasa’,’ab’)→ casa
REPLACE(A,B[,C])	Reemplaza en la cadena A la cadena B por la cadena C	**REPLACE**(‘roma re’,’r’,’T’) → Toma Te
RPAD(A,n[,B])	Mismo tratamiento que LPAD pero por la derecha	**RPAD**(‘casa’,7,’*’) → casa***
RTRIM(A, B)	Mismo tratamiento que LTRIM por la derecha	
SUBSTR(A,m[,n])	Devuelve una subcadena de A de longitud *n* comenzando en la posición *m*, si *m* es negativo comienza por la derecha	**SUBSTR**(‘abcdef’,2,3) → bcd
UPPER(A)	Convierte a mayúsculas el argumento	**UPPER**(‘casa’) → CASA
ASCII(A)	Devuelve el código ascii	**ASCII**(‘A’) → 65
INSTR(A, B, [,n[,m]])	Busca en el argumento A la cadena B devolviendo la posición de comienzo. *m* y *n* por defecto valen 1,	**INSTR**(‘piso raso’,’so’,2,2) → 8

	de lo contrario comienza a buscar a partir de *n* y cuenta a partir de la ocurrencia *m*	
LENGTH(A)	Devuelve la longitud de una cadena	**LENGTH**('la casuca') → 9

3. SQL y el modelo relacional.

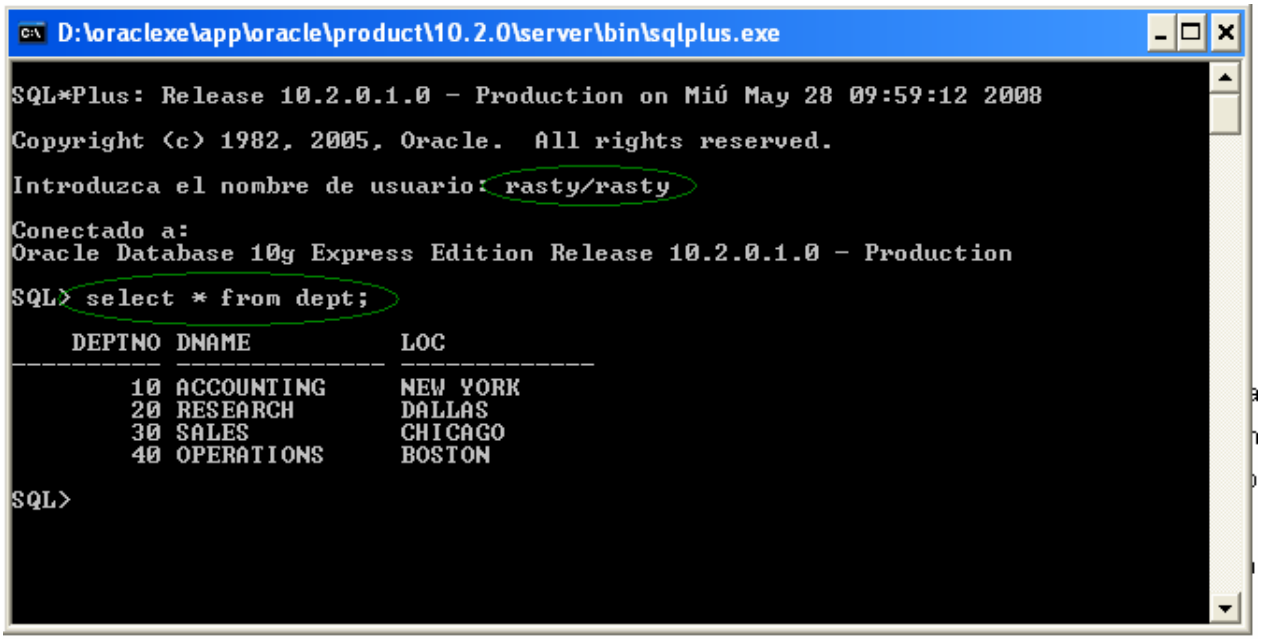
El lenguaje SQL es un lenguaje relacional, y aunque sea muy discutible la fidelidad con la que implementa los distintos elementos del modelo, es el lenguaje que incorporan la mayoría de los productos relacionales disponibles en el mercado.

En lo que sigue veremos las distintas sentencias SQL, relacionándolas con los operadores relacionales tratados en las unidades anteriores.

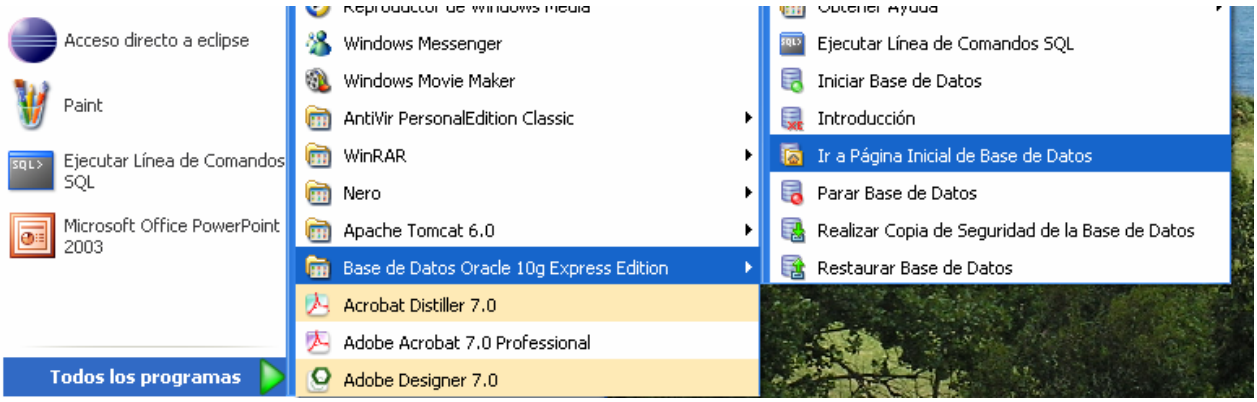
Comenzaremos con la sentencia SELECT, siguiendo siempre el método de ir descubriendo todas las posibilidades de cada sentencia, desde sus opciones más simples a las más complejas, con el apoyo continuo de ejemplos prácticos. A la vez que se explican las sentencias, en los ejemplos se irá introduciendo el uso de las funciones y operadores descritos anteriormente.

Para poder realizar comprobaciones y resolver los ejercicios que se proponen hay que conectarse como usuario RASTY.

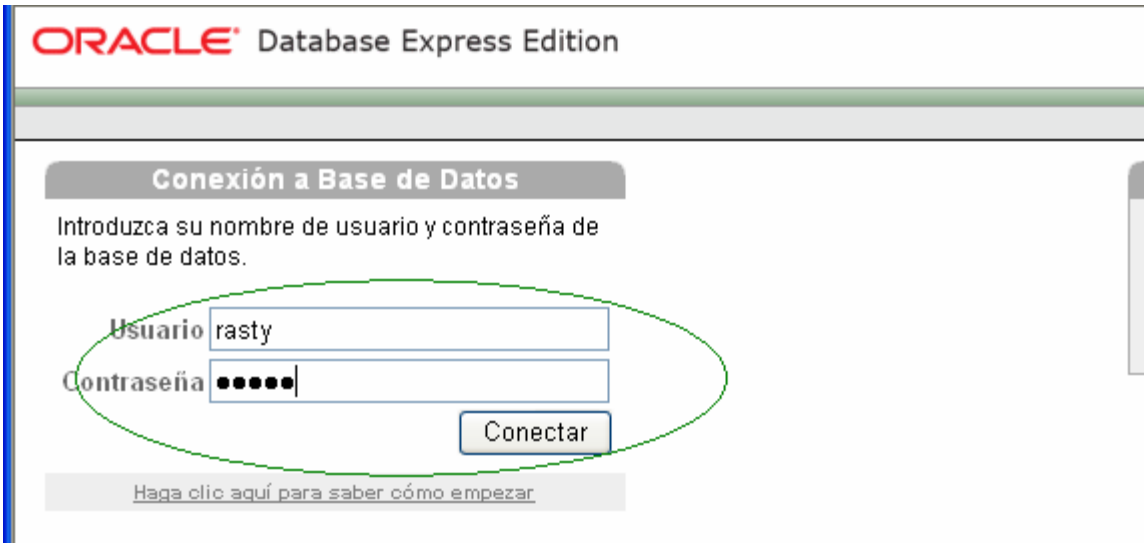
Podéis trabajar desde el entorno SQLPLUS. ***Inicio→ejecutar→sqlplus*** y conectarse como *rasty/rasty* . En la siguiente pantalla se muestra un ejemplo



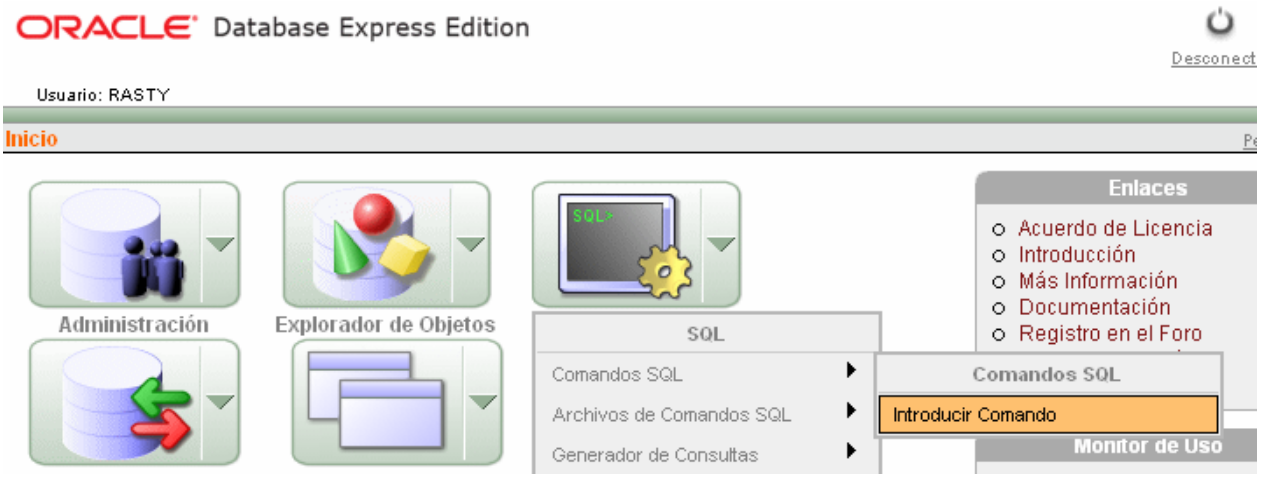
También se puede trabajar desde el asistente, la siguiente captura muestra como se inicia



Conectarse como *rasty*, contraseña *rasty*



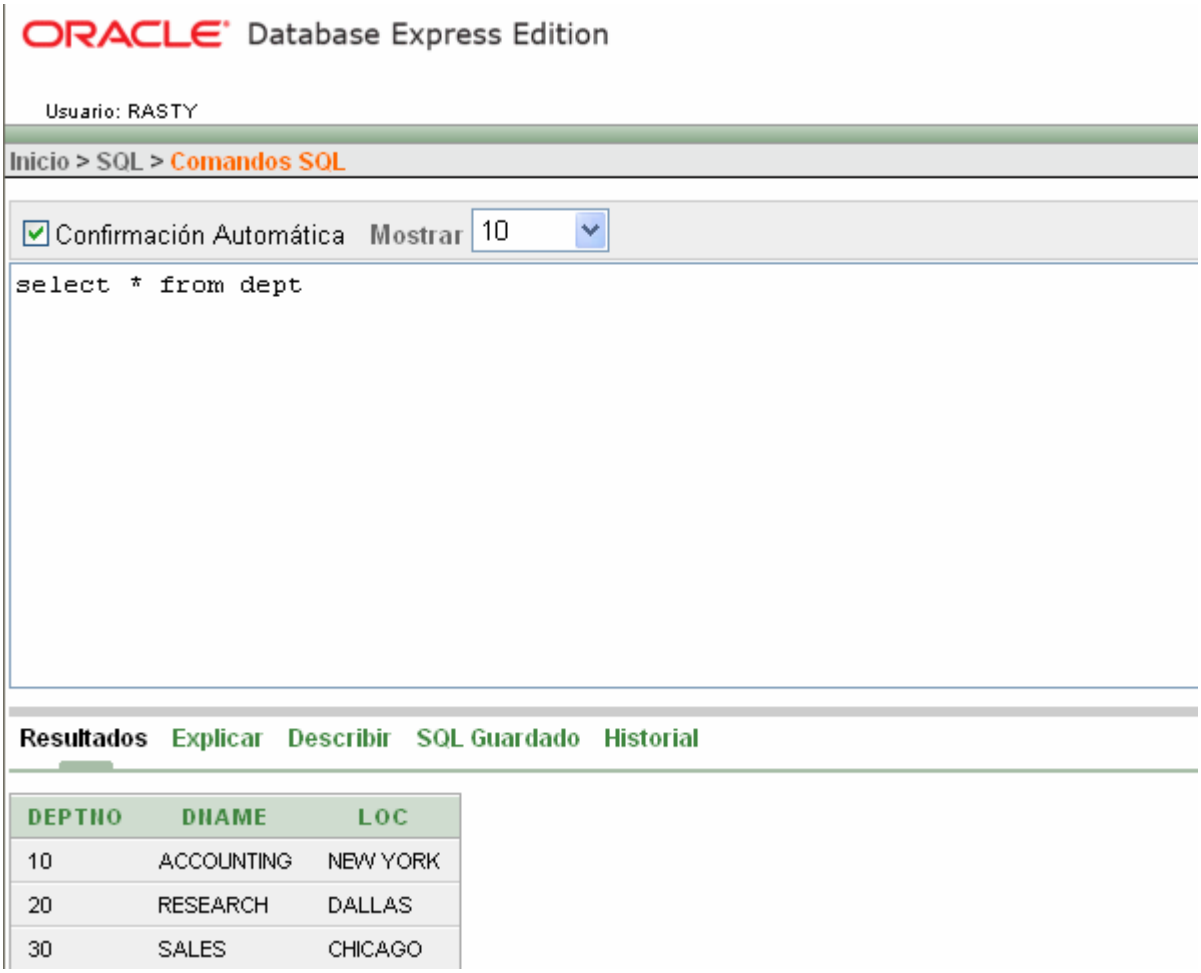
Seleccionar ***sql→comandos sql→ introducir comando***



Ejecutamos la siguiente sentencia

```
SELECT * FROM DEPT
```

Observad que si la ejecutáis desde *sqlplus* termina en punto y coma. Desde el asistente no hace falta que termine en punto y coma.



El resultado es el mismo, solo cambia el entorno.

4. Lenguaje de manipulación de datos (LMD)

4.1. Recuperación de información. Sentencia SELECT

La sentencia SELECT permite recuperar información contenida en una o más tablas de la base de datos. Esta sentencia nos permite escribir una expresión relacional equivalente a las que escribíamos en la Unidad 2 usando el álgebra o el cálculo relacionales. Esta expresión define la información a recuperar.

El resultado de evaluar una expresión relacional es siempre una relación, luego el resultado de evaluar una sentencia SELECT es siempre una serie de datos en forma de tabla.

- **Selección de determinadas columnas (operación de proyección)**

La primera operación que vamos a realizar con la SELECT será la de consultar determinadas columnas de una tabla sin eliminar los duplicados.

Sintaxis simple de la sentencia SELECT

SELECT expresión FROM tabla;

Por el momento *expresión* describe las columnas que se desean recuperar de la tabla. Para seleccionar algunas columnas se las nombra separadas por comas

SELECT ename, sal FROM emp;

```
ENAME             SAL
---------- ----------
SMITH              800
ALLEN             1600
WARD              1250
JONES             2975
MARTIN            1250
BLAKE             2850
CLARK             2450
SCOTT             3000
KING              5000
TURNER            1500
ADAMS             1100
ENAME              SAL
JAMES              950
FORD              3000
MILLER            1300

14 filas seleccionadas.
```

Para seleccionar todas las columnas de una tabla se utiliza el asterisco (*).

SELECT * FROM tabla;

Este formato no se puede utilizar si la tabla tienen tipos de datos LONG RAW

- **Clausula WHERE (operación de restricción).**

A la SELECT le siguen las columnas que se quieren consultar, a FROM la tabla a consultar, y si se quiere establecer una condición de búsqueda (restricción sobre las filas, seleccionando las que satisfagan una determinada condición) se utiliza la cláusula WHERE .

Sintaxis con la cláusula WHERE

SELECT expresión FROM tabla WHERE condición;

Como se puede ver, el resultado de una consulta es una tabla, si la hacemos interactivamente la salida se visualiza de forma tabular siempre que quepa en la línea de la pantalla. También se puede enviar a impresora o a un fichero. Si la consulta se hace desde un programa la tabla se vuelca en el programa.

Vamos a hacer una restricción según una condición. Utilizamos la cláusula WHERE para seleccionar determinadas filas de la tabla consultada. La cláusula consta de la palabra WHERE seguida de una condición simple o compuesta. Si al evaluar la condición da como resultado *true* la fila se recupera, si *false* o *desconocido* no se recupera.

SELECT ename, sal FROM emp WHERE sal > 1500;

```
ENAME             SAL
---------- ----------
ALLEN            1600
JONES            2975
BLAKE            2850
CLARK            2450
SCOTT            3000
KING             5000
FORD             3000
ALLEN            1600
```

La operación anterior puede ser vista como una operación de cálculo relacional.

Las filas que se recuperan no están ordenadas, conforme a la teoría relacional, si se quieren ordenar, lo que va en contra del modelo relacional, pero que desde el punto de vista de explotación (facilidad para el usuario final) se puede considerar una operación necesaria, se utiliza la cláusula ORDER BY seguida por las columnas por las que se quiere que se ordenen. Si se quiere que se ordenen por más de una columna se deben separar por comas.

Sintaxis con ORDER BY

SELECT expresión FROM tabla
WHERE condición
ORER BY {expresión | posición}[{ASC | DESC}];

Ejemplo.

SELECT ename, sal FROM emp WHERE sal > 1500 ORDER BY ename;

```
ENAME             SAL
---------- ----------
ALLEN            1600
BLAKE            2850
CLARK            2450
FORD             3000
JONES            2975
KING             5000
SCOTT            3000
```

La ordenación puede ser ascendente (ASC) o descendente (DESC), por defecto es ASC.

También se puede ordenar por posición de la columna, por ejemplo, en el caso anterior podemos utilizar posición y descendente.

SELECT ename, sal FROM emp WHERE sal > 1500 ORDER BY 2 DESC;

```
ENAME             SAL
---------- ----------
KING             5000
SCOTT            3000
FORD             3000
JONES            2975
BLAKE            2850
CLARK            2450
ALLEN            1600
```

Este es un ejemplo más de la falta de fidelidad de SQL al modelo relacional, los atributos de una relación no tienen orden, luego no se puede hacer referencia a su posición en la cabecera.

- **Función *SUBSTR***

Con esta función se puede obtener parte del contenido de una columna.

SELECT SUBSTR(dname, 1, 2) FROM dept;

```
SU
--
AC
RE
SA
OP
```

Las cabeceras que visualizan las consultas son las expresiones que se colocan en la SELECT ajustadas al ancho del contenido que se recupera. Si queremos cambiarlas, colocamos las nuevas a continuación de cada expresión de la SELECT . Si la cabecera contiene espacio u otro carácter con significado especial, se debe colocar entre comillas, sino es opcional.

Realmente lo que estamos haciendo es nombrar los atributos de la relación resultado. Recuérdese el operador RENOMBRAR del álgebra relacional.

SELECT dname "Nombre del Departamento" FROM dept;

```
Nombre del Dep
--------------
ACCOUNTING
RESEARCH
SALES
OPERATIONS
```

- **Tabla *DUAL***

Lo habitual es trabajar con la SELECT accediendo a las tablas de la base de datos, pero si queremos realizar cualquier tipo de operación en la que no tengan que intervenir tablas del usuario podemos utilizar la tabla del sistema DUAL cuya estructura es la siguiente:

DESC DUAL

```
Nombre                          ¿Nulo?    Tipo
 ------------------------------ -------- -----
 DUMMY                                   VARCHAR2(1)
```

Como vemos es una tabla de una sola columna de un byte de longitud, y si ejecutamos

SELECT * FROM DUAL;

Veremos que tiene una sola fila.

DESCRIBE (abreviado **DESC)** es una sentencia de SQL*PLUS que nos permite ver la estructura de una tabla (se verá en la próxima Unidad de Trabajo).

Ejemplos de utilización de la tabla DUAL.

Empleando la función **ABS** para obtener el valor absoluto

SELECT ABS(-30) FROM DUAL;

```
     VALOR
----------
        30
```

Evaluando una expresión aritmética

SELECT 4 * 5 / 10 Resultado FROM DUAL;

```
RESULTADO
---------
        2
```

En términos relacionales, en todos estos ejemplos tenemos una proyección nula de la tabla DUAL seguida de una ampliación.

- **Cláusula *DISTINCT***

Realmente, todos los ejemplos de la SELECT vistos hasta ahora no implementan el operador relacional de proyección con fidelidad, puesto que permiten que la tabla resultado tenga filas duplicadas.

Como vimos en la Unidad 2, por definición, una relación no puede tener tuplas repetidas. Para que la SELECT genere un resultado sin filas repetidas es necesario incorporarle la cláusula DISTINCT

SELECT DISTINCT sal FROM EMP;

```
  SAL
------
   800
   950
  1100
  1250
  1300
  1500
  1600
  2450
  2850
  2975
  3000
  5000
```

- **Expresiones en la *SELECT***

Como ya vimos en los ejemplos anteriores en los que se utilizó la tabla DUAL, podemos utilizar expresiones que implican la aparición de columnas nuevas como consecuencia de la evaluación de la expresión. Por ejemplo, si se quiere obtener el nombre de los empleados con el 50% de su salario siempre que éste sea mayor de 1000 y que el departamento sea 10, se codificaría la siguiente sentencia:

SELECT ename, 0.5*sal Salario FROM emp WHERE sal > 1000 AND deptno = 10;

```
ENAME        SALARIO
---------- ----------
CLARK            1225
KING             2500
MILLER            650
```

- **DECODE**

Podemos utilizar DECODE, no existe en el SQL estándar, para obtener un dato de una expresión multievaluada. Tiene la siguiente sintaxis:

DECODE (exp0, exp1, res1, exp2, res2, ..., def)

DECODE compara el valor de exp0 con el valor de exp1 si coincide devuelve res1, si no coincide compara con exp2 si es igual devuelve res2, así sucesivamente hasta que encuentre uno que sea igual. Si no hay ninguno que coincida asume el valor def.

Ejemplo.

SELECT ename, DECODE(deptno, 10, 'contabilidad',30,'ventas', 'no hay') DPTO FROM emp;

```
ENAME      DPTO
---------- ------------
SMITH      no hay
ALLEN      ventas
WARD       ventas
JONES      no hay
MARTIN     ventas
BLAKE      ventas
CLARK      contabilidad
SCOTT      no hay
KING       contabilidad
TURNER     ventas
ADAMS      no hay
JAMES      ventas
FORD       no hay
MILLER     contabilidad
```

DECODE también se puede utilizar en las cláusulas WHERE y ORDER BY.

- ***El operador IS NULL***

Cuando se quiera controlar el contenido nulo o no nulo de una columna utilizaremos el operador IS [NOT] NULL. Por ejemplo, para consultar todos los clientes con nulos en la columna de *observaciones*, se escribiría la siguiente sentencia:

SELECT ename, sal FROM emp WHERE comm IS NULL;

El resultado siempre será *true* o *false* nunca podrá ser, como puede suceder con otras condiciones, *nulo.*

- ***El operador BETWEEN.***

Es un operador que testea rangos, es decir comprueba si un valor se encuentra entre otros dos valores especificados. Se puede sustituir por otros operadores de relación y lógicos, Por ejemplo la expresión *A BETWEEN B AND C* se puede sustituir por *(A >= B) AND (A =< C)*

Para obtener la relación de empleados cuyos saldos estén comprendidos entre 1000 y 2000 codificaríamos la siguiente sentencia:

SELECT ename, sal FROM emp WHERE sal BETWEEN 1000 AND 2000;

- ***El operador LIKE***

Con este operador seleccionamos aquellas filas en las que el contenido de una columna se corresponde con un patrón en el que interviene el comodín *%* que sustituye una subcadena o el carácter subrayado (_) que sustituye un carácter. En el patrón pueden intervenir más de un comodín.

En la tabla de *emp* existen los nombres *ALLEN* y *ADAMS,* si se quieren obtener dichos nombre deberíamos comparar la columna *nombre* con el patrón *A%,.*

SELECT ename FROM emp WHERE ename LIKE 'A%';

```
ENAME
-------
ALLEN
ADAMS
```

Para sustituir un único carácter usaremos el carácter subrayado (_)

SELECT ename FROM emp WHERE ename LIKE '_MIT_';

```
ENAME
----------
SMITH
```

- ***La función LPAD***

Se puede hacer que en una consulta el resultado ajuste a izquierda (por defecto) o derecha y que además rellene con una determinada cadena (uno o varios caracteres).

En el siguiente ejemplo hacemos que el nombre ajuste a la derecha y que rellene por la izquierda con * sobre un ancho de campo de 20 caracteres en total.

SELECT LPAD(ename,20,'*') "NOMBRE" FROM emp;

```
NOMBRE
--------------------
***************SMITH
***************ALLEN
***************WARD
***************JONES
**************MARTIN
**************BLAKE
**************CLARK
***************SCOTT
****************KING
**************TURNER
***************ADAMS
***************JAMES
****************FORD
**************MILLER
```

Si se desea completar por la derecha se empleará RPAD.

4.2. Operadores tradicionales de conjuntos (UNIÓN, INTERSECCIÓN, DIFERENCIA)

Hasta aquí hemos planteado casos de recuperación de información con la sentencia SELECT. Se pueden plantear recuperaciones en las que intervengan más de una SELECT utilizando los operadores UNION, INTERSECCION y DIFERENCIA. Se habla así de consultas compuestas.

Cada SELECT daría lugar a una tabla resultado, combinándose estas tablas, de acuerdo a los operadores anteriores presentes en la expresión, para dar lugar a una única tabla resultado. En una consulta compuesta, las expresiones que aparecen en las sentencias SELECT deben ser iguales en número y tipo.

Operador	Descripción
UNION	Selecciona las filas de ambas tabla, se eliminan las duplicadas.
UNION ALL	No elimina las duplicadas
INTERSECT	Selecciona únicamente filas que pertenezcan a ambas tablas
MINUS	Selecciona los las filas que pertenezcan a la primera pero no a la segunda.

4.3. Resumen o agrupación (GROUP BY)

La cláusula GROUP BY permite agrupar las filas recuperadas en una sentencia SELECT según un determinado criterio, devolviendo una fila por cada uno de los grupos formados. O sea, permite obtener información resumen de cada grupo formado por las filas que recuperaría la SELECT.

Formato con la cláusula GROUP BY

SELECT expresión FROM tabla GROUP BY expresión;

En los resúmenes se emplean las funciones de columna o funciones de grupo, funciones que actúan sobre un conjunto de filas y afectan a los valores de una columna devolviendo un único resultado.

Función	Descripción	Ejemplos
AVG([DISTINCT | ALL] n)	Calcula el promedio de *n*	SELECT AVG(saldo) FROM clientes
SUM([DISTINCT | ALL] n)	Calcula la suma total de *n*	SELECT SUM(saldo) FROM clientes
MIN([DISTINCT | ALL] n)	Devuelve el mínimo valor de *n*	SELECT MAX(saldo) FROM clientes
MAX([DISTINCT | ALL] n)	Devuelve el máximo valor de *n*	SELECT MIN(saldo) FROM clientes
COUNT([DISTINCT | ALL] n)	Cuenta el número de filas donde *n* no es nulo	SELECT COUNT(saldo) FROM clientes
COUNT(*)	Cuenta el número de filas consultadas	SELECT COUNT(*) FROM clientes

Con la opción DISTINCT la función tiene en cuenta únicamente los valores no repetidos. Con la opción ALL la función tiene en cuenta todos los valores, incluidos los duplicados. Por defecto asume ALL.

Todas las funciones de grupo excepto COUNT(*) ignoran los valores nulos.

Las funciones no sólo actúan sobre filas agrupadas, sino que pueden hacerlo sobre toda la tabla como vemos en los ejemplos del cuadro anterior

A continuación veremos ejemplos de agrupaciones o resúmenes:

En la tabla *emp* podemos sumar los salarios para un determinado departamento.

SELECT SUM(sal) FROM emp WHERE deptno= '10';

```
SUM(SAL)
---------
     8750
```

Pero si se quiere consultar subtotales para cada departamento tenemos que utilizar la cláusula

GROUP BY

SELECT deptno, SUM(sal) FROM emp GROUP BY deptno;

```
  DEPTNO   SUM(SAL)
-------- ----------
      10       8750
      20      10875
      30       9400
```

No se permiten incluir columnas de detalle en las agrupaciones, por ejemplo, en este caso no se podría incluir el nombre del empleado. En general en la selección siempre se incluirá una columna de agrupación y una función de columna. Por otra parte las columnas de totalización NULL se realizan dentro del mismo grupo.

- ***Cláusula HAVING***

Se pueden excluir filas agrupadas, para lo cual utilizaremos la cláusula HAVING cuyo comportamiento es análogo al de la WHERE, con la salvedad que la cláusula HAVING actúa sobre las filas que previamente han sido agrupadas con la cláusula GROUP.

Sintaxis de la cláusula HAVING

SELECT expresión FROM tabla GROUP BY expression HAVING condición;

Ejemplo.

Excluir aquellas filas cuya suma de salarios por departamento no superen los 2000

SELECT deptno, SUM(sal) FROM emp GROUP BY deptno HAVING SUM(sal) <= 9500;

```
DEPTNO   SUM(SAL)
------- ----------
     10       8750
     30       9400
```

La cláusula WHERE también se puede utilizar en agrupamientos, pero hay que tener en cuenta que actúa sobre filas individuales, en cambio la cláusula HAVING actúa sobre filas agrupadas, por lo tanto las expresiones que contenga actuaran sobre grupos de filas, siempre incluirá una función de columna. Dicho de otra manera WHERE interviene en el proceso de selección, en cambio HAVING interviene una vez que las filas están agrupadas. En el siguiente caso vamos obtener las salarios por departamento que superen los 5000 solamente de los empleados que tengan comisión.

SELECT deptno, SUM(sal) FROM emp WHERE comm IS NOT NULL
GROUP BY deptno HAVING SUM(sal) > 5000;

```
DEPTNO   SUM(SAL)
------ ----------
    30       5600
```

Si se quiere realizar varios niveles de totalización, por ejemplo totalizar por departamento y dentro de estos por categoria, incluiríamos ambas columnas de agrupamiento en la selección (no obligatorio) y en la GROUP BY.

SELECT deptno, job, SUM(sal) FROM emp GROUP by deptno, job;

```
DEPTNO JOB         SUM(SAL)
------ --------- ----------
    10 CLERK           1300
    10 MANAGER         2450
    10 PRESIDENT       5000
    20 CLERK           1900
    20 ANALYST         6000
    20 MANAGER         2975
    30 CLERK            950
    30 MANAGER         2850
    30 SALESMAN        5600
```

La cláusula GROUP BY ordena los datos en ascendente y del nivel más externo al más interno.

4.4. Consultas en múltiples tablas (REUNIÓN O COMBINACIÓN)

La combinación consiste en concatenar las filas de una tabla con las de la otra.

En general se pueden recuperar datos de varias tablas obteniendo una única tabla compuesta por datos de las tablas accedidas.

Las consultas en más de una tabla se suelen hacer cuando nos encontramos con tablas relacionadas, tablas que tienen alguna columna en común, de forma que queremos obtener información de ambas tablas o de una de ellas, a través de la relación. En las tablas que manejamos hay una muy sencilla que es la relación *empleados/departamentos,* supongamos que queremos hacer una consulta para visualizar los nombres de los departamento con sus correspondientes empleados.

SELECT dname, ename FROM emp, dept WHERE emp.deptno=dept.deptno;

```
DNAME          ENAME
-------------- ----------
RESEARCH       SMITH
SALES          ALLEN
SALES          WARD
RESEARCH       JONES
SALES          MARTIN
SALES          BLAKE
ACCOUNTING     CLARK
RESEARCH       SCOTT
ACCOUNTING     KING
SALES          TURNER
RESEARCH       ADAMS
SALES          JAMES
RESEARCH       FORD
ACCOUNTING     MILLER
```

Cuando en la cláusula FROM hay más de una tabla, cualquier referencia a una columna debería ir calificada con el nombre de la tabla, por ejemplo, *emp.deptno*, para referirse a la columna código del departamento de la tabla empleados. La calificación sólo es obligatoria en caso de ambigüedad, o sea, cuando nos referimos a un nombre de columna que se repita en más de una tabla. En la consulta anterior las columnas que relaciona ambas tablas (deptno) se llaman igual, para distinguirlas hay que anteponer el nombre de la tabla a la que pertenecen, empleados o departamentos (emp.deptno, dep.deptno).

4.5. Reuniones externas

Cuando se incluyen en una selección atributos de una tabla que no tienen contrapartida en la relacionada, se dice que es una reunión externa.

La reunión externa se identifica con el signo **+** asociado al atributo que interviene en la relación, puede ser reunión externa izquierda o bien reunión externa derecha. En el siguiente ejemplo la relación se establece entre los códigos de departamento de las tablas de empleados *(emp)* y departamentos
(dep)

SELECT dname, ename FROM emp, dept WHERE emp.deptno(+) = dept.deptno;

```
DNAME          ENAME
-------------- ----------
ACCOUNTING     CLARK
ACCOUNTING     KING
ACCOUNTING     MILLER
RESEARCH       SMITH
RESEARCH       ADAMS
RESEARCH       FORD
RESEARCH       SCOTT
RESEARCH       JONES
SALES          ALLEN
SALES          BLAKE
SALES          MARTIN
SALES          JAMES
SALES          TURNER
SALES          WARD
OPERATIONS
```

Como se puede observar el departamento OPERATIONS no tiene empleados, es decir se obtiene una relación con todos los departamentos que tienen empleados, incluyen los departamento que no tienen correspondencia en la tabla *emp.*

4.6. Los ALIAS (variables de filas explícitas)

Como vimos en las unidades anteriores, en las expresiones de cálculo relacional se utilizan variables de tupla. El mecanismo de calificación de columnas utilizado anteriormenete no es ni más ni menos que el uso que de este concepto hace la sentencia SELECT. En los ejemplos vistos se usó como variable de fila el nombre de la tabla. Se dice que estas son variables de fila implícitas.

En la cláusula FROM se pueden definir alias de los nombres de tabla y estos alias pueden usarse como variables de fila. Los alias son variables de fila explícitas. Con el objeto de clarificar el código en una consulta donde intervengan muchas tablas podemos utilizar variables de filas explícitas, o sea alias, acortándose el nombre de las tablas. En el siguiente ejemplo se pone el alias ***e*** a la tabla *emp* y a la tabla *dep* se pone ***d***

SELECT dname, ename FROM emp e, dept d WHERE e.deptno = d.deptno;

Una vez establecidas las bases de acceso a varias tablas, se pueden restringir las búsquedas tanto como se quiera, por ejemplo, podemos realizar una consulta para visualizar los nombres de los empleados con los correspondientes departamentos, pero sólo de aquellos que tengan comisión.

SELECT dname, ename FROM emp e, dept d WHERE e.deptno = d.deptno
AND comm IS NULL;

```
DNAME          ENAME
-------------- ----------
RESEARCH       SMITH
RESEARCH       JONES
SALES          BLAKE
ACCOUNTING     CLARK
RESEARCH       SCOTT
ACCOUNTING     KING
RESEARCH       ADAMS
SALES          JAMES
RESEARCH       FORD
ACCOUNTING     MILLER
```

Como norma aconsejamos que si se manejan varias tablas se utilicen los alias tanto si existe ambigüedad como si no, es decir se aplicará el siguiente formato:

```
SELECT a.columna, b.columna….
FROM tabla1 a, tabla2 b, ….
WHERE condición;
```

4.7. Producto cartesiano generalizado

Como se explicó en la Unidad 2, el producto cartesiano generalizado de dos tablas da lugar a otra cuyas filas resultan de la concatenación de cada fila de una tabla con todas las filas de la otra. La operación es semejante a una combinación en la que se omite la cláusula WHERE.

```
SELECT ename, dname  FROM   emp,  dept ;
```

También se puede combinar la tabla consigo misma

```
SELECT a.ename, b.ename     FROM  emp  a,  emp  b;
```

La estructura anterior puede parecer que carece de sentido, sin embargo es importante recordarla, en su momento se propondrá un ejercicio en el que es imprescindible aplicarla.

4.8. SELECT anidadas (subconsultas)

La capacidad de utilizar SELECT anidadas nos permite utilizar el resultado de una consulta como entrada para otra consulta. Hay algunas restricciones al utilizar las SELECT anidadas, por ejemplo no se pueden seleccionar una lista de valores si no vamos a utilizar cuantificadores (***some, all, any, in***). En general, puesto que el resultado de una subconsulta va a formar parte de la condición de búsqueda de otra consulta debe concordar con la misma. Se utilizarán en las cláusulas WHERE y HAVING con los operadores relacionales descritos anteriormente y también se puede utilizar en la cláusula FROM en lugar de una tabla.

En el siguiente ejemplo no hace falta utilizar cuantificadores ya que la subconsulta devuelve un único valor.

Obtener el nombre de los empleados cuyo salario sea mayor que el del empleado Jones

```
SELECT ename FROM emp WHERE sal >
        (SELECT sal FROM emp WHERE ename='JONES');
```

```
ENAME
------
SCOTT
KING
FORD
```

Con este otro ejemplo la subconsultas devuelve una lista de valores, luego hay que utilizar un cuantificador.

Obtener el nombre de los empleados que tienen un salario igual a cualquiera de los salarios mínimos de cada uno los departamentos.

```
SELECT ename FROM emp WHERE sal =
        ANY (SELECT min(sal) FROM emp GROUP BY deptno);
```

```
ENAME
------
SMITH
JAMES
MILLER
```

- ***Subconsulta en la FROM***

Al utilizar la SELECT en la FROM las consultas las hacemos contra la vista que se obtiene y no contra la tabla.

La siguiente sentencia selecciona el nombre de la tabla *emp* de aquellas filas que cumplan la condición que se establece.

SELECT ename FROM emp WHERE comm IS NULL;

La siguiente consulta es equivalente a la anterior.

SELECT ename FROM (SELECT * FROM emp) WHERE comm IS NULL;

En el ejemplo anterior la consulta se realiza sobre una vista, resultado de utilizar una subconsulta en la FROM.

4.9. Añadiendo filas. INSERT

Para añadir filas a las tablas se utiliza la sentencia INSERT. Se pueden añadir las filas completando todas las columnas de la tabla con algún valor o con nulos (NULL). También se pueden añadir filas completando solo algunas columnas de la tabla, en este caso hay que nombrarlas.

Sintaxis simple de la sentencia INSERT

INSERT INTO nombre_tabla [(columnas…)] VALUES (valores…)

Vamos añadir una fila a la tabla de *dept* completando todas las columnas de la tabla con algún valor:

INSERT INTO dept VALUES (50,'CONTABILIDAD','AVILES');

Se puede prescindir de algún valor sustituyéndolo por NULL.

INSERT INTO dept VALUES (60,'PERSONAL',NULL);

En el caso de que se quiera prescindir de varias columnas se puede determinar en cuáles vamos introducir información en vez de sustituir los valores por NULL.

En la tabla *emp* vamos añadir una fila dando valores a determinadas columnas, en este caso debemos especificar las columnas en las que se quiere introducir datos.

INSERT INTO emp (empno, ename, sal) VALUES(9999,'JOSE MARIA',3000);

La fila se insertará con valores nulos en las columnas no especificadas. Si alguna de estas columnas fue definida con un valor por defecto (DEFAULT), en ella se insertará dicho valor en lugar de NULL. Si no se definió con valor por defecto y tiene la restricción NOT NULL, la fila será rechazada por el SGBD, y no se insertará ante el intento de violación de la restricción. En un próximo apartado trataremos las restricciones.

Otro formato de la INSERT es el que permite recuperar datos de una tabla para añadirlos a otra, sustituyendo la cláusula VALUES por una subconsulta. En este caso las columnas de la subconsulta deben de coincidir en número y tipo con las especificadas en la INSERT.

```
INSERT  INTO dept2 SELECT * FROM dept;
```

4.10. Actualizar columnas. UPDATE

Con la sentencia UPDATE se modifican los valores, contenidos, de las columnas de las filas seleccionadas de una sola tabla.

Sintaxis simple de la UPDATE

```
UPDATE   Nombre_de_la _tabla
        SET columna= expresión [, columna = expresión].....
              [WHERE condición]
```

La WHERE selecciona las filas a actualizar. Sigue las mismas reglas que las empleadas en las sentencia SELECT, pudiendo incluso utilizar SELECT subordinadas.

En la cláusula SET se incluyen expresiones que generen un valor que esté en concordancia con el tipo de dato definido para la columna. No se pueden incluir funciones de columna ni subconsultas. Si se pueden incluir datos de la fila que está seleccionada.

Un ejemplo sencillo para actualizar una tabla puede consistir en incrementar el salario de todos los empleados en un 10% siempre que su comisión sea nula

```
UPDATE  emp  SET  sal = sal * 1.10   WHERE  comm  IS  NULL;
```

Para actualizar más de una columna en la misma UPDATE separarlas con una coma.

En el caso de que actualicemos una columna C1 con el valor de otra columna C2 (siempre el de la fila seleccionada) que a su vez se actualiza con el valor de una expresión, C1 toma el valor que tenia C2 antes de actualizarse.

Se pueden utilizar subconsultas en la cláusula WHERE, lo cual es interesante cuando hay que actualizar columnas a partir de valores contenidos en otras tablas. El tratamiento descrito en el apartado de SELECT ANIDADAS es aplicable en este punto.

4.11. Borrar filas DELETE

Con esta sentencia podemos borrar una serie de filas de la tabla accedida. Las filas que se borran se seleccionan con la cláusula WHERE cuyas condiciones de búsqueda son las mismas que las vistas en la sentencia SELECT.

Sintaxis simple de la sentencia DELETE

DELETE FROM nombre_de_la_tabla [WHERE condición]

Si quisiéramos borrar todos los empleados del departamento 10, codificaríamos la siguiente sentencia.

DELETE FROM emp WHERE deptno = 10;

Si no se utiliza la cláusula WHERE se borran todas las filas de la tabla, quedando la tabla vacía. No se borra la estructura.

DELETE FROM emp;

Con la sentencia anterior borraríamos todas las filas de la tabla *emp.*

No vamos a entrar en este momento a comentar que pasaría si la tablas estuvieran definidas con sus correspondientes ***constraints***, estamos suponiendo que no se han declarado los controles de integridad, este caso lo veremos en otro apartado.

En la cláusula WHERE podemos incluir subconsulta para poder borrar filas en base a datos de otras tablas. Al igual que se describió en el apartado de actualizaciones, la subconsulta permite referencias externas y diferentes niveles de anidamiento.

Ejemplo

Borrar los empleados del departamento ubicado en DALLAS

```
DELETE FROM emp WHERE deptno IN
  (SELECT deptno FROM dept WHERE loc LIKE '%DALLAS%');
```

Si queremos eliminar la estructura de una tabla, es decir eliminar realmente la tabla, no sólo el contenido, se utilizará la sentencia DROP TABLE. Se verá en el siguiente epígrafe.

Introducción

SQL es un lenguaje que nos **permite interactuar con los SGBD Relacionales** para especificar las operaciones que deseamos realizar sobre los datos y su estructura. En la unidad anterior hemos utilizado SQL para manipular la información contenida en las tablas.

En esta unidad se explican los fundamentos del SQL estándar para definir las estructuras, modificarlas y establecer los sistemas de seguridad. En definitiva vamos trabajar con los lenguajes de definición y control. Por último veremos funciones avanzadas del lenguaje.

Objetivos

- Conocer las sentencias de definición (LDD)
- Conocer las sentencias de control (LCD)
- Modificar las estructuras de los objetos creados
- Crear y manipular vistas, sinónimos e índices
- Trabajar con múltiples tablas.

Contenidos Generales

1. CREACIÓN Y ELIMINACIÓN DE TABLAS 3
1.1. Creación de tablas *3*
1.2. Modificar la definición de las tablas (ALTER) *11*
1.3. Indices *13*
1.4. Sinónimos *13*
1.5. Eliminar tablas, índices, sinónimos. *14*
2. TRANSACCIONES. 15
3. VISTAS. 16
3.1. Vistas simples. *16*
3.2. Vistas agrupadas. *17*
3.3.Vistas compuestas *18*
3.4. Acceso a las vistas *18*
3.5. Actualización de vistas. *19*
3.6.Clausulas de CREATE VIEW *21*
3.7. Eliminar vistas. *24*
3.8. Ventajas e inconvenientes de las vistas. *25*
4. OTRAS FUNCIONES 25
4.1. Funciones de conversión y tratamiento de fechas y números *25*
4.2. Resto de Funciones. *33*
5. CONFIDENCIALIDAD DE LOS DATOS. 37
5.1 Privilegios y roles. *38*
5.2. Autentificación de usuarios. *42*
6. DESCRIPCIÓN DE LAS TABLAS UTILIZADAS EN ESTAS UNIDADES. 44

1. Creación y eliminación de tablas

Las sentencias descritas hasta este apartado formaban parte del lenguaje de manipulación de datos. La creación de tablas, modificación de su estructura, eliminación, etc., se llevan a cabo con un conjunto de sentencias que constituyen lo que se llama el **lenguaje de definición de datos (LDD)**.

1.1. Creación de tablas

La creación de tablas no es responsabilidad de los desarrolladores, ya que generalmente son los administradores quienes crean los objetos de la base de datos (tablas, vistas, ….), sin embargo consideramos que los programadores deben conocer los diferentes tipos de sentencias del lenguaje de definición de datos: crear tablas, vistas, índices, etc. En este caso los usuarios crearán sus propias tablas, serán tablas privadas que cargarán y actualizarán y sobre las que concederán derechos a los usuarios que consideren oportuno, para que puedan acceder a ellas.

Para que un usuario cree tablas en su propio esquema, o en otro tiene que tener los privilegios necesarios.

Para crear las tablas se utiliza la sentencia CREATE que en su formato más simple, sin tener en cuenta las reglas de integridad ni los parámetros de almacenamiento (asume los de por defecto) y siendo el usuario que la crea el propietario, es el siguiente:

```
CREATE TABLE nombre_de_la _tabla
(
columna                    tipo
[,columna                  tipo]
[,columna                  tipo]
……..
);
```

Los tipos de datos son los descritos al principio de la unidad, las columnas se separan con comas y van encerradas entre paréntesis. Si se utilizan nombres compuestos se unen por el carácter subrayado _

La siguiente definición crea la tabla de *cursos*

```
CREATE TABLE CURSOS
(
 CODIGO                    VARCHAR2(10),
 NOMBRE                     VARCHAR2(100),
 FECHA_INICIO               DATE,
 FECHA_FIN                  DATE,
 VACANTES                   NUMBER(2),
 MODALIDAD                 VARCHAR2(100),
 OBSERVACIONES             VARCHAR2(400)
);
```

la tabla es propiedad del usuario que la creó, para manipularla sólo tenemos que hacer referencia al nombre de la tabla.

Hasta ahora todas las manipulaciones de tablas las hemos hecho suponiendo que éramos propietarios de las mismas, es decir, que están en nuestro esquema, de no ser así, lo mismo que calificábamos una columna para romper la ambigüedad tendríamos que calificar la tabla para asociarla al esquema al que pertenece.

En lo sucesivo suponemos que somos propietarios de nuestras tablas, por lo tanto no las calificaremos.

Para más información sobre **esquemas** y **usuarios** consultar el epígrafe: Confidencialidad

En las tablas creadas hasta ahora no se controlaba la duplicidad de filas, la relación entre tablas, ni los chequeos de validez, en definitiva, no existían restricciones.

Con las reglas de integridad, tanto de integridad de entidad, como referencial, que se estudiaron en las unidades anteriores, se dota a las tablas de las restricciones necesarias para que los datos almacenados sean consistentes.

A continuación, utilizando la tabla de *alumnos* como modelo, vamos a definir las reglas de integridad.

Como primer paso se trata de encontrar la clave primaria (PRIMARY KEY), que puede ser simple o compuesta y en la que ninguno de sus componentes puede ser nulo, (integridad de entidad). En nuestro caso va ser el *nif*

A continuación se pueden definir claves alternativas (UNIQUE), que son las claves candidatas excepto la elegida como primaria.

Si se considera oportuno, se pueden asignar valores por defecto (DEFAULT) a determinadas columnas. En las altas (inserciones) las columnas que no se les asigna un valor asumen el valor por defecto especificado con DEFAULT. En el caso de que no se les asigne ningún valor y tampoco se les de un valor por defecto asumen nulos.

Podemos establecer chequeos de validez (CHECK). La restricción CHECK permite controlar el valor que se asigna a una columna, comparando el valor asignado con una serie de valores posibles especificados en la CHECK, en caso de no coincidir con ninguno no permite que se grabe la fila. Es aconsejable que el chequeo se realice en la aplicación y no en la base de datos.

También se puede establecer que columnas no pueden ser nulas (NOT NULL).

Por último definiremos la integridad referencial definiendo claves ajenas.

Las definiciones de restricciones de integridad de entidad y de integridad referencial las haremos con la cláusula CONSTRAINT. En Oracle una clave ajena puede hacer referencia a cualquier clave candidata, sea primaría o alternativa.

Tanto si se hace referencia a claves primarias o alternativas, las columnas de referencia deben coincidir en número, tipo y tamaño con las columnas referidas.

Veamos un ejemplo con las definiciones de constraints:

```
CREATE TABLE CURSOS
(
 CODIGO                      VARCHAR2(10),
 NOMBRE                       VARCHAR2(100),
 FECHA_INICIO                 DATE,
 FECHA_FIN                    DATE,
 VACANTES                     NUMBER(2),
 MODALIDAD                   VARCHAR2(100),
 OBSERVACIONES               VARCHAR2(400),
       CONSTRAINT PK_CURSOS PRIMARY KEY(CODIGO)
);
```

Los nombres de las **constraints** (PK_CURSOS) son los nombres con los que se almacenan en el diccionario de datos. Estas **constrains** se denominan de tabla por estar definidas a nivel de tabla (después de definir todas las columnas), se pueden definir a nivel de columna siempre que no intervenga más de una columna en su composición.

```
CREATE TABLE CURSOS
(
 CODIGO                      VARCHAR2(10)  NOT  NULL,
 NOMBRE                      VARCHAR2(100),
 FECHA_INICIO                DATE  DEFAULT SYSDATE,
 FECHA_FIN                   DATE,
 VACANTES                    NUMBER(2),
 MODALIDAD                   VARCHAR2(100),
 OBSERVACIONES               VARCHAR2(400),
       CONSTRAINT PK_CURSOS PRIMARY KEY(CODIGO)
);

CREATE TABLE ALUMNOS
(
 MATRICULA                  VARCHAR2(10)    NOT NULL,
 NOMBRE                     VARCHAR2(15),
 APELLIDOS                  VARCHAR2(30),
 NIF                        VARCHAR2(10),
 TELEFONO                   NUMBER(10),
 E_MAIL                     VARCHAR2(100),
```

```
CURSO                          VARCHAR2(10),
  CONSTRAINT UQ_CLAVE                UNIQUE (NIF),
   CONSTRAINT PK_ALUMNOS PRIMARY KEY(MATRICULA),
   CONSTRAINT FK_CURSOS FOREIGN KEY (CURSO)
       REFERENCES CURSOS(CODIGO) ON DELETE CASCADE
);
```

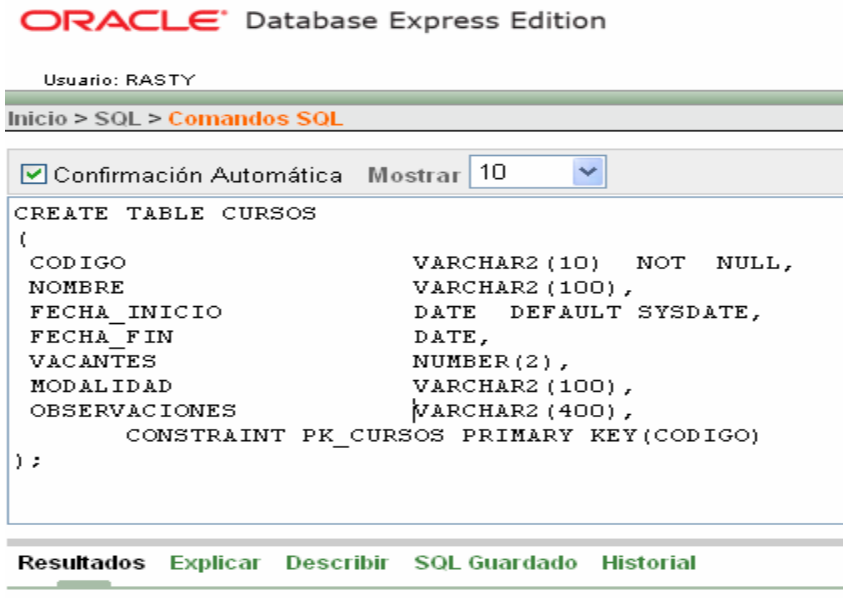
Como norma, utilizaremos CONSTRAINT de tabla para declarar claves ajenas y candidatas

Oracle permite realizar la operación de borrado en cascada.

Con el borrado en cascada, ON DELETE CASCADE, al eliminar una fila de la tabla maestra se borran todas las filas de la tabla relacionada cuyo valor de clave ajena coincida con el valor de la clave primaria borrada.

Con la definición de tablas anteriores si se elimina un curso se borran todos los alumnos que pertenecen a ese curso.

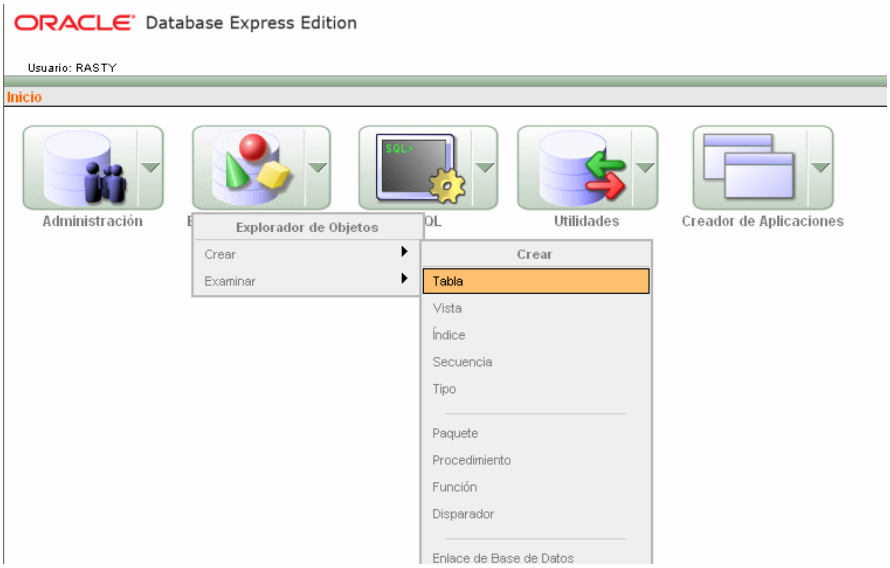
Para crear las tablas os conectáis como *rasty/rasty* y desde ***comandos sql*** las podéis crear, tal cual se muestra en la siguiente figura



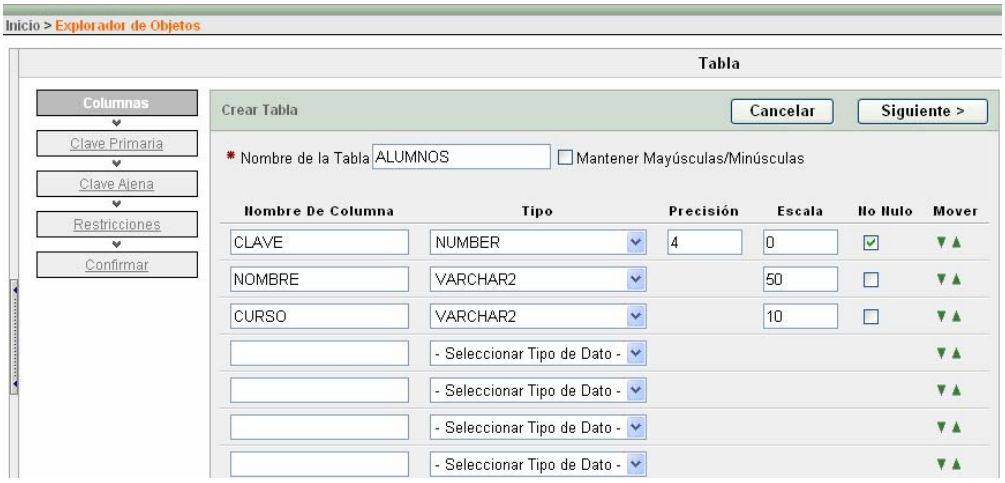

```
Tabla creada.
```

También se pueden crear las tablas utilizando específicamente el asistente de Oracle para esta labor. En las secuencias siguientes se muestra como se crea la tabla ALUMNOS relacionada con la de curso por medio de la clave ajena ***curso***

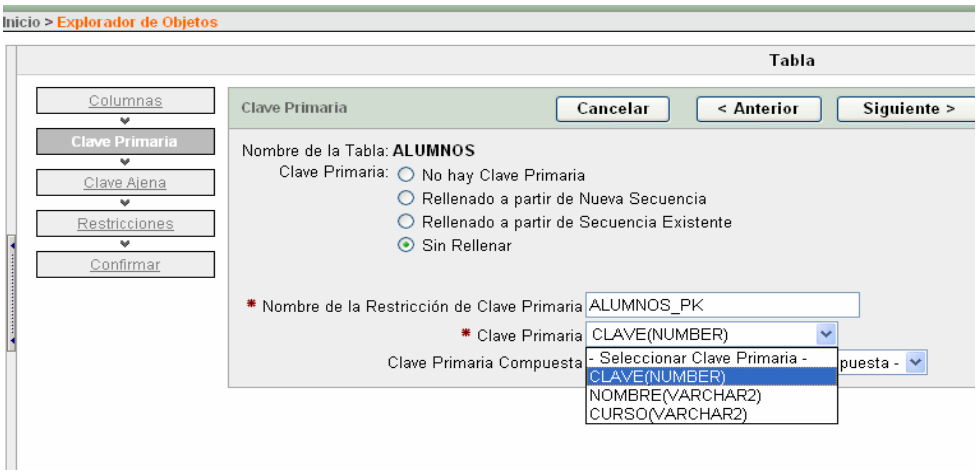
Explorador de objetos → crear → tabla



Se van creando las columnas, dándoles nombre, naturaleza y longitud

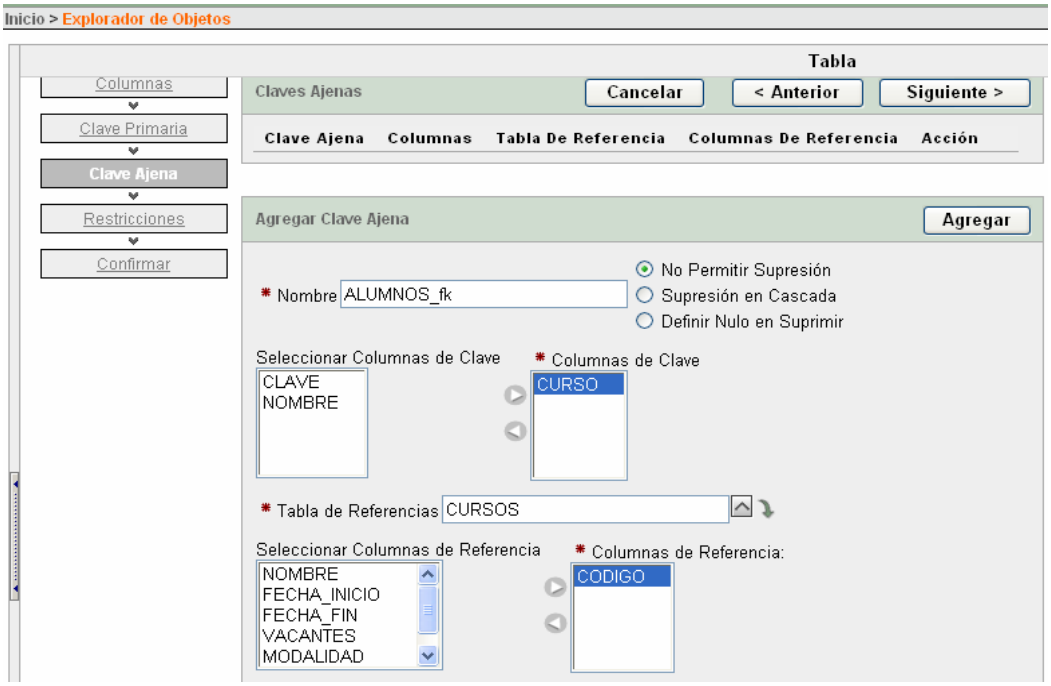


Se define la clave primaria, elegimos la columna CLAVE como primaria. El nombre que propone Oracle para guardar en el catálogo (ALUMNOS_PK) no hace falta cambiarlo. Todos los objetos, incluidas las claves tienen un nombre en el diccionario de datos.



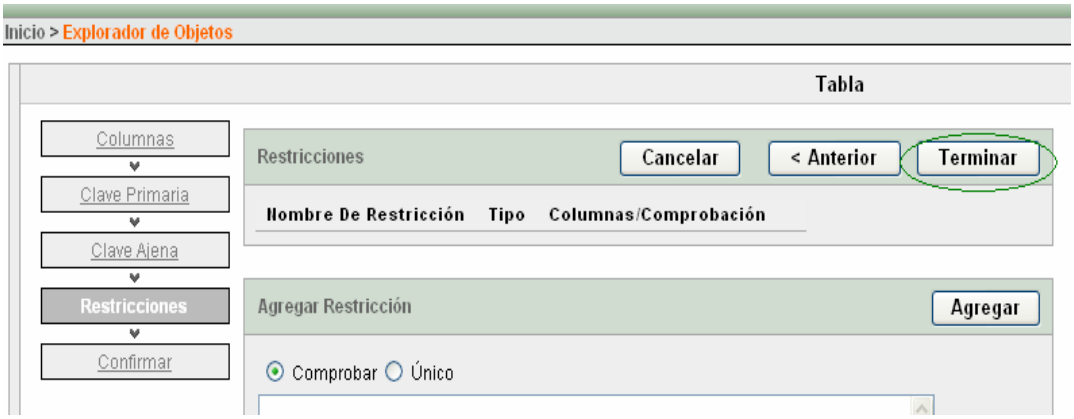
En la siguiente captura se define la clave ajena, en este caso la columna CURSO, que es la columna que va enlazar con la tabla cursos. En tabla de referencias se elige CURSOS y en columnas de referencia se selecciona CODIGO que es la clave primaria de la tabla CURSOS.

Por defecto no se permite supresión, lo que significa que no se pueden eliminar cursos si tiene alumnos relacionados. Si se elige supresión en cascada, permite eliminar el curso y al mismo tiempo elimina los alumnos que relacionados con el curso. Por último la opción de puesta a nulos, permite borrar el curso y poner nulos en la clave ajena CURSO de los alumnos relacionados con el curso que se elimina.

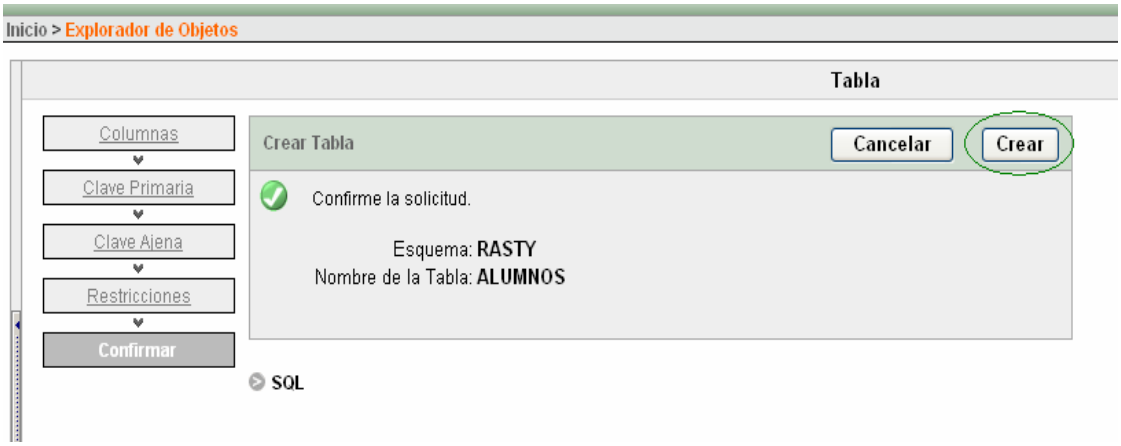


En restricciones no añadimos ninguna. Sería para poner otras restricciones aparte de las claves. Por ejemplo, no permitir nulos, que en una columna se almacene una determinada información (en el caso de que hubiera un PAGA_BANCO que permita almacenar S o N).

Seleccionar la opción TERMINAR



Por último seleccionar CREAR



La siguiente captura muestra la tabla de ALUMNOS

1.2. Modificar la definición de las tablas (ALTER)

Con la sentencia ALTER podemos añadir columnas, nuevas restricciones, redefinir los tipos de datos, tamaños y valores por defecto. También se pueden modificar las características de almacenamiento y desactivar y activar los constraints.

Sintaxis simple de la sentencia ALTER

ALTER TABLE tabla {ADD | MODIFY} {columnas | constraint};

Con ADD añadimos columnas o constraints. Con MODIFY modificamos la definición de las columnas o las constraints.

Ejemplos.

Si queremos añadir a la tabla *dept* la columna *observaciones* emplearemos el siguiente formato de la sentencia ALTER

ALTER TABLE dept ADD (observaciones VARCHAR2(50));

El contenido de la nueva columna es NULL, si queremos definirla como NOT NULL la tabla tiene que estar vacía.

Para modificar el tamaño o el tipo de un dato empleamos la opción MODIFY. En el siguiente ejemplo se modifica el tamaño de la columna *observaciones*

ALTER TABLE dept MODIFY (observaciones VARCHAR2(80));

Sólo se puede cambiar el tipo o decrecer el tamaño de las columnas, si tienen nulos en todas las filas. Por el contrario, se puede incrementar el tamaño con datos en las columnas.

Se pueden definir nuevos valores por defecto, pero la columnas toman esos valores para las nuevas filas que se inserten, las que ya existen permanecen con el valor que tienen.

En cuanto a los controles de integridad, con la cláusula MODIFY sólo se puede añadir NOT NULL a una columna existente, pero con la clausula ADD se pueden añadir todas las constraints (PRIMARY, FOREING, UNIQUE,…..)

Suponiendo que en la tabla ALUMNOS no se hubiese definido la alternativa del NIF, se podría hacer con ALTER

ALTER TABLE alumnos ADD CONSTRAINT clave_alternativa_nif UNIQUE (nif);

Para desactivar las constraints utilizaremos la cláusula DISABLE

ALTER TABLE alumnos DISABLE CONSTRAINT clave_alternativa_nif

Para activar las constraints utilizaremos la cláusula ENABLE

ALTER TABLE alumnos ENABLE CONSTRAINT clave_alternativa_nif

1.3. Indices

Podemos indexar la tabla para aquellas columnas que se necesiten accesos rápidos. Se puede indexar una columna simple o una concatenación de varias. Además de los índices que nosotros creemos, el propio S.G.B.D. crea un índice automáticamente cuando se establecen CONSTRAINS de tipo PRIMARY KEY o UNIQUE

Sintaxis para crear índices

```
CREATE INDEX nombre_del_índice
        ON nombre_de_la_tabla (columnas [ASC | DDESC]);
```

CREATE INDEX i_ape_nom ON alumnos (apellidos, nombre);

El nombre del índice crea una entrada en el diccionario de datos. El número de columnas que se puede indexar por tabla dependerá de la base de datos con la que estemos trabajando. Los nulos no son indexados.

Los índices aceleran las búsquedas pero hacen más lentas las actualizaciones, ya que ellos mismos deben actualizarse.

1.4. Sinónimos .

Los sinónimos son nombres alternativos que se pueden emplear para gestionar las tablas, vistas, secuencias, procedimientos, funciones, paquetes, snapshot, y otros sinónimos.

Formato para crear sinónimos

CREATE [PUBLIC] SYNONYM nombre_del_sinónimo
FOR {tabla | vista | función | procedimiento |secuencia | paquete |snapshot |sinónimo};

Los sinónimos pueden ser utilizados en las sentencias de manipulación INSERT, UPDATE, DELETE, SELECT ya comentadas y por las de control GRANT y REVOKE.

Los sinónimos proporcionan independencia para tratar los objetos (tablas, vistas, procedimientos, etc) ya que podemos acceder a objetos de otros propietarios o bases de datos remotas sin necesidad de especificar el propietario ya que va implícito en el nombre del sinónimo.

Suponiendo que el propietario de la tabla *emp* sea *ana*, para hacer una consulta sobre dicha tabla desde otro usuario, que tenga los correspondientes derechos, utilizaríamos la siguiente SELECT

SELECT * FROM ana.emp;

Si creamos el siguiente sinónimo

CREATE SYNONYM con_emp FOR ana.emp;

Podemos realizar la consulta de la siguiente forma

SELECT * FROM con_emp;

1.5. Eliminar tablas, índices, sinónimos.

Para eliminar objetos tales como: tablas, índices, sinónimos. Y otros que se verán en próximas unidades como: procedimientos, funciones, paquetes, disparadores, etc. Se utiliza la sentencia DROP. Cuando se utiliza esta sentencia, la definición de los objetos se pierde junto con su contenido, y no hay forma de recuperarlo. En el caso de las tablas para crear de nuevo la estructura tenemos que volver a utilizar la sentencia CREATE.

DROP TABLE alumnos;

Con la sentencia anterior eliminamos la tabla de *alumnos.*

Para eliminar una tabla relacionada (con constraints de relación), se empleará el siguiente formato

DROP TABLE alumnos CASCADE CONSTRAINTS;

Para borrar el índice que hemos creado en el ejemplo anterior, empleamos la sentencia:

DROP INDEX i_ape_nom;

Y para borra el sinónimo *con_emp;*

DROP SYNONYM con_emp;

2. Transacciones.

Una transacción es una secuencia de instrucciones SQL que el S.G.B.D gestiona como una unidad de tratamiento. (Más información en la unidad de trabajo PL/SQL)

Una transacción comienza en la primera sentencia SQL, después de una sentencia COMMIT, sentencia ROLLBACK o conexión a la base de datos.

Una transacción termina con una sentencia COMMIT o ROLLBACK o tras una desconexión, intencionada o no, de la base de datos. El S.G.B.D. realiza un COMMIT implícito antes de ejecutar cualquier sentencia DDL (*create, alter,..)* o al realizar una desconexión que no haya sido precedida de un error.

Los cambios realizados a la base de datos en el transcurso de una transacción solo son visibles para el usuario que los ejecuta. Al ejecutar COMMIT los cambios realizados a la base de datos pasan a ser permanentes y por lo tanto visibles para todos los usuarios.

Si una transacción termina con ROLLBACK, se deshacen los cambios realizados a la base de datos por las sentencias previas, quedando en el estado previo al inicio de la transacción.

Durante la ejecución de una transacción, el S.G.B.D. guarda temporalmente información que permita, si se precisa, hacer ROLLBACK y que haga posible realizar lecturas consistentes de la información involucrada en esa transacción. En el caso de Oracle, esta información se almacena en los denominados ***segmentos de rollback***

Las sentencias COMMIT y ROLLBACK serán tratadas con más profundidad en la unidad de trabajo PL/SQL

3. Vistas.

Las vistas definen una tabla virtual basada en una o más tablas o vistas. Esta tabla virtual se almacena permanentemente en la base de datos, generando, al igual que las tablas, una entrada en el diccionario de datos. Las vistas permiten organizar los datos de diferentes formas, de modo que los usuarios los ven desde diferentes perspectivas. De igual modo, se restringe el acceso a los datos, permitiendo que determinados usuarios a través de las vistas sólo puedan acceder a determinadas filas y columnas de las tablas.

Desde el punto de vista del usuario, la vista es como una tabla real, con estructura de filas y columnas; pero a diferencia de éstas sus datos no se almacenan físicamente en la base de datos. Las filas y columnas visibles para el usuario son consecuencia de la consulta que se realiza en las tablas reales sobre las que se definió la vista.

La consulta que define la vista permite los operadores relacionales y los de conjunto que se han estudiado en los apartados anteriores. Así podemos aplicar a las vistas los operadores de restricción, proyección, reunión, división. También podemos utilizar los operadores tradicionales de conjuntos: unión, intersección y diferencia.

3.1. Vistas simples.

Para crear una vista utilizamos la sentencia CREATE VIEW, que en su formato más simple tiene la siguiente estructura:

CREATE VIEW nombre_vista AS consulta

Se puede hacer una clasificación de las vistas atendiendo al tipo de consulta. Si la vista se crea a partir de determinadas filas de la tabla fuente pero con todas sus columnas, obtenemos una vista por restricción.

CREATE VIEW sin_comm AS SELECT * FROM emp WHERE comm IS NULL;

Con la sentencia anterior se crea una vista con todos los empleados que no tienen comisión

Si construimos la vista con todas o parte de las filas pero con algunas de las columnas de la tabla fuente tendríamos una vista por proyección.

CREATE VIEW sin_comm AS SELECT empno, ename FROM emp WHERE comm IS NULL;

3.2. Vistas agrupadas.

Otra construcción posible es la que procede de agrupaciones de filas de la tabla original GROUP BY, se denomina vista agrupada, ya que sus filas están construidas por agrupaciones de la tabla fuente. En este caso hay que especificar las columnas que se tienen que corresponder con las de la consulta.

Sintaxis para las vistas agrupadas

CREATE VIEW nombre_vista (columnas…) AS consulta con GROUP BY

Vamos a crear una vista con los salarios de los empleados agrupados por nombre

Ejemplo

CREATE VIEW suma_sal (nombre, salario) AS SELECT ename, sum(sal) FROM emp WHERE comm IS NOT NULL GROUP BY ename;

La vista anterior estará constituida por los nombres y los salarios de los empleados que tienen comisión.

Al crear una vista podemos dejar que herede los nombres de las columnas de las expresiones que aparecen detrás de la SELECT o darles nosotros nombre, como en este caso que los llamamos *nombre y salario*

Para consultar una vista empleamos la sentencia select con el mismo formato que si fuera una tabla. La consulta de la vista que se acaba de crear sería.

SELECT * FROM suma_sal;

```
NOMBRE       SALARIO
---------- ---------
ALLEN           1600
MARTIN          1250
TURNER          1500
WARD            1250
```

3.3.Vistas compuestas.

Las vistas no tienen porque definirse sobre una sola tabla, pueden estarlo sobre varias tablas. En el siguiente ejemplo la vista *emp_dept* tiene dos columnas obtenidas de las tablas *emp y dept*

CREATE VIEW emp_dept AS SELECT ename, dname FROM emp, dept
WHERE emp.deptno=dept.deptno AND sal > 2000;

En este caso la vista se construye con los nombres de los empleados y departamentos, excluyendo aquellos empleados cuyo salario es inferior a 2000.

3.4. Acceso a las vistas

El usuario accede a los datos de una vista exactamente igual que si estuviese haciéndolo a una tabla. De hecho, en muchos casos el usuario no sabrá siquiera que está consultando una vista. En general, el uso de las vistas simplifica las consultas a la base de datos.

La siguiente consulta se hace sobre la vista creada en el epígrafe anterior.

SELECT * FROM emp_dept;

```
ENAME      DNAME
---------- --------------
JONES      RESEARCH
BLAKE      SALES
CLARK      ACCOUNTING
SCOTT      RESEARCH
KING       ACCOUNTING
FORD       RESEARCH
```

Las consultas actúan sobre las vistas igual que sobre las tablas reales, por lo tanto podemos trabajar con las mismas cláusulas de la SELECT que se describieron en las consultas sobre tablas.

3.5. Actualización de vistas.

Para que una vista se pueda actualizar debe de existir una relación directa entre las filas y las columnas de la vista y las de la tabla fuente. Por ejemplo, en el caso siguiente sí se pueden realizar inserciones, borrados y actualizaciones.

CREATE VIEW sin_comm AS SELECT * FROM emp WHERE comm IS NULL;

La vista **sin_comm** tiene la misma estructura que la tabla fuente, luego las actualizaciones que se realicen sobre ella se realizan sobre la tabla fuente.

No se pueden actualizar las vistas que para su construcción se hayan incluido en las consultas las cláusulas:

- GROUP BY o HAVING
- En la WHERE subconsultas
- En las columnas que tienen expresiones o funciones
- En la FROM varias tablas
- La cláusula DISTINCT

Estas reglas no son aplicables a todos los Sistemas Gestores de Bases de Datos. Se consideran reglas generales reguladas por ANSI/ISO, pero cada SGBD tiene sus particularidades y por lo tanto se debe estudiar en cada caso.

Insertamos una fila en la vista *sin_comm*

INSERT INTO sin_comm (empno,ename,sal) VALUES (9999,'BALDOMERO', 9000);

Si ahora seleccionamos las filas de la tabla fuente

SELECT empno, ename, sal FROM emp;

```
 EMPNO ENAME              SAL
------ ---------- ----------
  7369 SMITH             800
  7499 ALLEN            1600
  7521 WARD             1250
  7566 JONES            2975
  7654 MARTIN           1250
  7698 BLAKE            2850
  7782 CLARK            2450
  7788 SCOTT            3000
  7839 KING             5000
  7844 TURNER           1500
  7876 ADAMS            1100
  7900 JAMES             950
  7902 FORD             3000
  7934 MILLER           1300
  9999 BALDOMERO        9000
```

Vemos que insertó la fila a través de la vista.

Si consultamos la vista

```
EMPNO ENAME              SAL
------ ---------- ----------
  7369 SMITH             800
  7566 JONES            2975
  7698 BLAKE            2850
  7782 CLARK            2450
  7788 SCOTT            3000
  7839 KING             5000
  7876 ADAMS            1100
  7900 JAMES             950
  7902 FORD             3000
  7934 MILLER           1300
  9999 BALDOMERO        9000
```

Se observa que la fila está en la vista

Vamos intentar insertar una fila en la vista *emp_dept*

INSERT INTO emp_dept VALUES ('baldomero', 'contabilidad');

El sistema visualiza el siguiente mensaje:

ERROR en línea 1:
ORA-01776: no se puede modificar más de una tabla base a través de una vista de unión

El error se debe a la imposibilidad de actualizar las tablas fuente cuando la vista se construye con una selección multitabla.

Cuando hacemos una inserción en una vista definida sobre una sola tabla pero que no coge todas sus columnas, sucede que:

- Las columnas de la tabla que no están en la vista y tienen la restricción DEFAULT asumen el valor por defecto.
- Las columnas de la tabla que no están en la vista y no tienen la restricción DEFAULT, asumen nulos.
- Si las columnas de la tabla que no están en la vista tienen la restricción NOT NULL y no la DEFAULT no se inserta la fila .

3.6.Clausulas de CREATE VIEW

A continuación exponemos algunas de las opciones que se pueden utilizar cuando se crea una vista .

- **Cláusula *CHECK OPTION.***

Acabamos de ver que las vistas son actualizables cuando existe una relación directa entre las filas y columnas de la vista con las de la tabla fuente, sin embargo podemos observar que si insertamos filas con una condición, la tabla fuente se actualiza con dichas filas, pero si la consulta que define la vista no satisface dicha condición para las filas insertadas, éstas no son

visibles a través de la vista. Dicho de otra forma, cuando se define una vista simple por restricción, a través de la vista solo son visibles aquellas filas de la tabla base que cumplen la restricción.

Veamos el contenido de la tabla *emp:*

SELECT empno, ename, sal FROM emp;

Creamos una vista con los empleados cuyo salario exceda de 2000

CREATE VIEW vista_emp AS SELECT empno, ename, sal FROM emp WHERE sal > 2000;

A través de ella se visualiza

SELECT * FROM vista_emp;

```
     EMPNO ENAME             SAL
---------- ---------- ----------
      7566 JONES            2975
      7698 BLAKE            2850
      7782 CLARK            2450
      7788 SCOTT            3000
      7839 KING             5000
      7902 FORD             3000
```

Tal como fue definida esta vista, es posible insertar filas que luego no sean visibles a través de ella misma.

Insertamos una fila con salario inferior a 2000

INSERT INTO vista_emp VALUES (9999,'LAURA', 1000);

Si ahora realizamos una consulta de la vista comprobamos que la fila insertada no es visible

SELECT * FROM vista_emp;

```
  EMPNO ENAME             SAL
------- ---------- ----------
   7566 JONES            2975
   7698 BLAKE            2850
   7782 CLARK            2450
   7788 SCOTT            3000
   7839 KING             5000
   7902 FORD             3000
```

Sin embargo la fila si está en la tabla fuente.

SELECT empno, ename, sal FROM emp;

```
  EMPNO ENAME             SAL
------- ---------- ----------
   7369 SMITH             800
   7499 ALLEN            1600
   7521 WARD             1250
   7566 JONES            2975
   7654 MARTIN           1250
   7698 BLAKE            2850
   7782 CLARK            2450
   7788 SCOTT            3000
   7839 KING             5000
   7844 TURNER           1500
   7876 ADAMS            1100
   7900 JAMES             950
   7902 FORD             3000
   7934 MILLER           1300
   9999 LAURA            1000
```

Con la cláusula CHECK OPTION se asegura que las operaciones de inserción y actualización sobre la vista satisfagan el criterio de búsqueda en la WHERE de la definición de la vista. De esta forma se asegura que las filas actualizadas o insertadas van a ser visibles a través de la vista.

CREATE VIEW vista_emp AS SELECT empno, ename, sal FROM emp WHERE sal > 2000

WITH CHECK OPTION;

Si intentamos insertar una fila con un salario menor que 2000.

INSERT INTO vista_emp VALUES (9999,'LAURA', 1000);

Produce el siguiente error.

```
          *
ERROR en línea 1:
ORA-01402: violación de la cláusula WHERE en la vista WITH CHECK OPTION
Cláusula OR REPLACE
```

Cuando se necesitan modificar la definición de una vista, la única solución es crear una nueva vista con el mismo nombre, ya que no existe una sentencia parecida a la ALTER TABLE que permitía modificar la definición de una tabla.

- **Cláusula *OR REPLACE***

CREATE OR REPLACE VIEW vista_emp AS SELECT empno, ename, sal FROM emp;

La cláusula OR REPLACE de la sentencia CREATE VIEW permite sustituir la definición de una vista por una nueva definición

- **Cláusula *WITH READ ONLY***

No se permiten borrados, inserciones o actualizaciones a través de la vista.

3.7. Eliminar vistas.

Las vistas se eliminan con la sentencia DROP, la cual tiene el mismo formato que el utilizado para eliminar tablas, índices,..

DROP VIEW nombre_de_la_vista

Si la vista está en otro esquema y tenemos derechos, podemos borrarla anteponiendole el nombre del usuario

DROP VIEW nombre_usuario.nombre_vista

3.8. Ventajas e inconvenientes de las vistas.

Las vistas proporcionan una serie de beneficios como:

- La simplicidad de las consultas, haciendo que consultas con selecciones complejas se simplifiquen a través de las vistas.
- La personalización de la base de datos para determinados usuarios, de forma que presenta los datos con una estructura lógica para dichos usuarios.
- Como consecuencia del apartado anterior se controla el acceso a la base de datos a través de las vistas, dejando que el usuario pueda ver y manejar determinada información

Como principales inconvenientes, podemos citar:

- Las restricciones referidas a actualizaciones.
- Caída del rendimiento cuando se construyen vistas con selecciones complejas.

4. Otras funciones

Además de las funciones de cadena y aritméticas descritas, existen otras funciones tales como funciones de conversión, tratamiento de fechas y otras de proposito general. En este apartado vamos describir algunas de estas funciones.

4.1. Funciones de conversión y tratamiento de fechas y números

Este tipo de funciones convierten un tipo de datos en otro. En este apartado describiremos las más utilizadas, en cualquier caso cada implementación de SQL incorpora, además de las que aquí se describen funciones propias de conversión.

Las funciones de conversión y tratamiento de fechas varían dependiendo del S.G.B.D que se este utilizando, en nuestro caso nos vamos ajustar las funciones definidas por ORACLE.

- ***Función TO_CHAR***

Función que convierte datos de tipo numérico o fecha a tipo VARCHAR2.

Sintaxis Formato de TO_CHAR para conversión de fechas

TO_CHAR(dato [, mascara [,'nlsparams']])

La función TO_CHAR para conversión de fechas, convierte el dato de tipo DATE al tipo VARCHAR2 según el formato especificado por la máscara. Si se omite la máscara el dato es convertido a VARCHAR2 según el formato por defecto.

El parámetro *nlsparams* especifica el lenguaje en el que aparecerán escritos los nombres y abreviaturas de meses y días de la semana. El formato de este parámetro es **NLS_DATE_LANGUAJE = lenguaje.** Si se omite *nlsparams,* esta función utiliza el lenguaje por defecto de la sesión.

Elementos de la mascara para fechas

Separadores	- / . : ; "texto"
MM	número del mes
MON	abreviatura del mes
MONTH	nombre del mes relleno con espacios hasta una longitud de 9 caracteres
RM	mes en números romanos
DDD	número del día del año
DD	número del día del mes
D	número del día de la semana
DY	abreviatura del día de la semana
DAY	nombre del día de la semana completo
YYYY	año con cuatro dígitos
YY	año con dos dígitos
YEAR	año en letra
RR	dos últimos dígitos del año, permitiendo almacenar fechas del siglo 21 en el 20 y viceversa
Q	número del trimestre
WW	semana del año (1-53) donde la semana 1 comienza en el día primero del año y continua hasta el día séptimo
W	semana del mes (1-5) donde la semana 1 comienza el día 1 del mes y termina el día 7
HH	hora del día
HH12	hora del día (1-12)
HH24	hora del día (1-24)
MI	minuto de la hora
SS	segundo del minuto
AM	muestra AM o PM dependiendo de la hora del día

Hemos descrito los elementos que consideramos más significativos. Consultar el manual o ayuda en línea para ver el resto de elementos que se pueden utilizarse en la mascara.

Con TO_CHAR podemos cambiar el formato en edición.

Si editamos la fecha del sistema

SELECT SYSDATE FROM DUAL;

Por defecto produce la siguiente salida (toma la mascara que tiene definida por defecto para la

sesión)

```
SYSDATE
--------
01/11/08
```

Si queremos editarla con otra mascara:

```
SELECT  TO_CHAR(SYSDATE,'DD-MM-YYYY') FROM DUAL;

TO_CHAR(SY
----------
11-02-2008
```

Para conocer el día del año:

```
SELECT TO_CHAR(SYSDATE,'YYDDD') FROM DUAL;

TO_CH
-----
08042
```

El trimestre

```
SELECT TO_CHAR(SYSDATE,'Q') FROM DUAL;

T
-
1
```

- ***Función TO_CHAR para la conversión de números.***

La función TO_CHAR para la conversión de números tiene el mismo formato que la TO_CHAR
para la conversión de fechas, en este caso el dato es de tipo NUMBER. Si se omite la mascara convierte el número a un tipo VARCHAR2 de la longitud necesaria para contener todos sus dígitos significativos.

El parámetro ***nlsparams*** especifica los caracteres que serán utilizados para representar:
Separador de decimales
Separador de millares
Símbolo monetario local
Símbolo monetario internacional
Este argumento puede tener esta forma:
NLS_NUMERIC_CHARACTERS = 'dg'
NLS_CURRENCY = "text"
NLS_ISO_CURRENCY = 'territory'

Los caracteres ***d*** y ***g*** representan el separador de decimales y de millares respectivamente. Deben ser caracteres diferentes. Se pueden usar hasta diez caracteres para el símbolo monetario.

Si se omite ***nlsparams*** o cualquiera de sus componentes, esta función usa los valores por defecto de los parámetros para la sesión.

Elementos de la mascara para edición de números

Elemento	Ejemplo	Descripción
9	999	Devuelve el valor numérico eliminando ceros a la izquierda
0	00999	Con ceros por la izquierda
$	$999	Devuelve el valor con el $
MI	999MI	Devuelve el signo menos por la derecha si el valor es negativo
S	S999	Devuelve - si es negativo y + si es positivo
G	99G999	Separador de grupo de enteros

D	99D99	Separador de grupo de decimales
,	99,999	Puntuación de millar
.	99.99	Puntuación decimal
EEEE	9.9EEEE	Devuelve el valor en notación científica

Ejemplos

Consulta del salario de JONES sin máscara de edición

```
SELECT  ename, sal  FROM  emp WHERE ename = 'JONES';

NAME             SAL
--------- ----------
JONES            2975
```

Consulta del salario de JONES con máscara de edición

```
SELECT  ename, TO_CHAR(sal,'9G990D00')   FROM   emp  WHERE ename = 'JONES';

ENAME      TO_CHAR(S
---------- ---------
JONES       2.975,00
```

- ***Función TO_DATE***

Convierte el dato de tipo CHAR o VARCHAR2 al tipo DATE. La mascara es un formato de fecha que especifica en que forma esta representado el dato. Si se omite la mascara, el dato debe de estar en el formato de fecha por defecto.

Sintaxis de TO_DATE

TO_DATE(dato [, mascara ['nlsparams']]);

Con esta función podemos ajustar la fecha a una determinada mascara.

- **Modelos de formatos de fechas**

Se pueden usar modelos de formatos de fechas en los siguientes lugares:

- En la función TO_CHAR para convertir un valor de tipo DATE que esté en un formato distinto del formato por defecto
- En la función TO_DATE para convertir un valor de tipo carácter que esté en un formato distinto del formato por defecto

El formato de fechas por defecto se especifica explícitamente con el parámetro de inicialización NLS_DATE_FORMAT o implícitamente con el parámetro de inicialización NLS_TERRITORY.

Aquellos elementos de la mascara que devuelven valores en letra (MONTH, DAY, ..) lo harán en el idioma especificado explícitamente con el parámetro de inicialización NLS_DATE_LANGUAJE o implícitamente con NLS_LANGUAJE.

- **El elemento *RR* de la mascara.**

Este elemento es similar al YY, pero proporciona una flexibilidad adicional para almacenar valores de fechas en otros siglos. Esta máscara permite almacenar fechas del siglo 21 en el 20 especificando solo los dos últimos dígitos del año. También se puede hacer el proceso inverso, almacenar fechas del 20 en el 21.

Si se usa la función TO_DATE con la máscara YY, el valor devuelto pertenece siempre al siglo actual. Si se usa la máscara de formato RR en su lugar, el siglo devuelto varia con los dos últimos dígitos del año especificado y con los dos últimos del año actual. En la siguiente tabla se especifica el funcionamiento de este elemento.

Si los dígitos del año especificado son Si los dos últimos Dígitos del año actual Son	0 - 49	50 – 99
0 – 49	La fecha devuelta está en el siglo actual	La fecha devuelta está en el siglo anterior al actual
50 – 99	La fecha devuelta está en el siglo siguiente al actual	La fecha devuelta está en el siglo actual

Ejemplos:

Las consultas se hacen con la fecha del ordenador en el 2008

Consulta con máscara RR

```
SELECT TO_CHAR(TO_DATE ('27-OCT-95','DD-MON-RR'), 'YYYY') "Año"   FROM DUAL;

Año
----
1995
```

Consulta con mascara YY

```
SELECT TO_CHAR(TO_DATE ('27-OCT-95','DD-MON-YY'), 'YYYY') "Año"    FROM DUAL;
Año
----
2095
```

- ***Función TO_NUMBER***

Convierte un dato tipo CHAR o VARCHAR2 con contenido numérico, en un valor de tipo NUMBER según el formato especificado por la máscara.

El parámetro *nlsparams* tiene el mismo propósito que en la función TO_CHAR para conversión de números.

TO_NUMBER(dato [, mascara [,'nlsparams']])

Ejemplo con la función TO_NUMBER:

SELECT TO_NUMBER('-008000','S999999') FROM DUAL;

```
TO_NUMBER('-008000','S999999')
------------------------------
                         -8000
```

- **Función *ADD_MONTHS***

Suma meses a una fecha

ADD_MONTHS(fecha, cantidad_meses)

Ejemplo.

Obtenemos la fecha del sistema con formato DD-MM-YYYY

SELECT TO_CHAR(SYSDATE,'DD-MM-YYYY') "Fecha" FROM DUAL;

```
Fecha
----------
12-02-2008
```

Sumamos cuatro al mes

SELECT TO_CHAR(ADD_MONTHS(SYSDATE,4),'DD-MM-YYYY') "Nueva fecha" FROM DUAL;

```
Nueva fecha
----------
12-06-2008
```

- ***Función MONTHS_BETWEEN***

Devuelve el número de meses entre dos fechas

MONTHS BETWEEN(fecha1. fecha2)

SELECT MONTHS_BETWEEN('01-05-98','01-01-97') FROM DUAL;

```
MONTHS_BETWEEN('01-05-98','01-01-97')
-------------------------------------
                                   16
```

- ***Función LAST_DAY***

Devuelve la fecha correspondiente al último día del mes de la fecha que se consulta.

LAST_DAY(fecha)

SELECT SYSDATE, LAST_DAY(SYSDATE) "Ultimo " ,
LAST_DAY(SYSDATE) - SYSDATE "Diferencia" FROM DUAL;

```
SYSDATE  Ultimo   Diferencia
-------- -------- ----------
12/02/08 29/02/08         17
```

4.2. Resto de Funciones.

A continuación describimos algunas funciones que podríamos denominar de propósito general por englobarlas de alguna manera en un único epígrafe. En cualquier caso no se contemplan todas las funciones disponibles. Invitamos al lector que consulte el manual de SQL .

- ***Función LEAST***

Selecciona el menor valor de una lista. Los valores pueden ser columnas, literales, expresiones,...

LEAST(VALOR1, VALOR2, ...)

En el caso de fechas, selecciona la menor fecha de la lista. Si la fecha es un literal hay que convertirla con TO_DATE

Primero vemos cual es la fecha del sistema

SELECT TO_CHAR(SYSDATE,'DD-MM-YYYY') "Fecha" FROM DUAL;

```
Fecha
----------
12-02-2008
```

Luego seleccionamos la más pequeña, entre ella y el 01-01-08

SELECT TO_CHAR(LEAST(SYSDATE,'01-01-08'),'DD-MM-YYYY') "LA MENOR" FROM DUAL;

```
LA MENOR
----------
01-01-2008
```

- ***Función GREATEST***

Selecciona el mayor valor de una lista. Los valores pueden ser columnas, literales, expresiones,...

GREALEST (VALOR1, VALOR2,...)

En el caso de fechas, selecciona la mayor fecha de la lista.

Se puede establecer una comparación entre valor fecha con un literal que contenga una fecha con formato por defecto sin necesidad de convertirlo con TO_DATE.

No se pueden comparar literales aunque sean fechas, sin antes convertirlos con TO_DATE

- ***Función NVL.***

Se utiliza para sustituir nulos por un valor real.

NVL(expresion1, expresion2)

Si ***expresion1*** *y* ***expresion2*** son tipos diferentes, primero hay que convertir *expresion2* al tipo *expresion1* y luego se compara.

En el ejemplo siguiente se cambian los nulos obtenidos en la consulta por 0. (Las comisiones nulas por cero)

SELECT NVL(COMM,0) FROM EMP;

```
NVL(COMM,0)
-----------
          0
        300
        500
          0
       1400
          0
          0
          0
          0
          0
          0
          0
          0
          0
```

- **Función *UID***

Devuelve el identificativo del usuario. Es de tipo numérico.

SELECT UID FROM DUAL;

```
  UID
-------
   41
```

- ***Función USER***

Devuelve el nombre del usuario. Es de tipo carácter.

SELECT USER FROM DUAL;

```
USER
-----
SCOTT
```

- **Función *USERENV***

Devuelve información de tipo carácter acerca de la sesión que está activa.

USERENV (opciones)

Ejemplo.

Lenguaje definido para la sesión activa.

SELECT USERENV('LANGUAGE') FROM DUAL;

```
USERENV('LANGUAGE')
----------------------------
SPANISH_SPAIN.WE8MSWIN1252
```

- ***Función VSIZE***

Devuelve el número de bytes que ocupan los datos seleccionados.

SELECT dname, VSIZE(dname) *"BYTES"* FROM dept;

```
DNAME               BYTES
-------------- ---------
ACCOUNTING             10
RESEARCH                8
SALES                   5
OPERATIONS             10
```

5. Confidencialidad de los datos.

En este apartado veremos los elementos del lenguaje SQL que facilitan la protección de la base de datos contra accesos no autorizados.

El lenguaje SQL estándar soporta el sistema de gestión de confidencialidad conocido como **control de acceso discrecional**. Este sistema se basa en que cada usuario tendrá diferentes derechos de acceso, también conocidos como **privilegios**, sobre distintos objetos de la base de datos. Normalmente, usuarios diferentes tendrán privilegios diferentes sobre el mismo objeto. El mecanismo de vistas soportado por SQL facilita también la gestión de la confidencialidad, ocultando información a usuarios no autorizados.

Oracle utiliza el concepto de **esquema** o colección con nombre de objetos, como tablas, vistas, procedimientos y packages. Un usuario es un nombre definido en la base de datos con el que se puede conectar y acceder a los objetos en los esquemas de la base de datos.

Cuando se crea un usuario en la base de datos, se crea también un esquema con el mismo nombre para el usuario. Por defecto, cuando el usuario se conecta con la base de datos, tiene acceso a todos los objetos del esquema correspondiente. Un usuario está asociado sólo con el esquema de su mismo nombre.

Disponiendo de los privilegios correspondientes, un usuario puede trabajar en cualquier esquema, no sólo en el suyo. El nombre de cualquier objeto pude calificarse con el nombre del esquema en que se encuentra

[nombre del esquema.] nombre del objeto

Si se omite el nombre del esquema se utiliza el esquema del usuario.

5.1 Privilegios y roles.

Oracle, para facilitar la administración de privilegios, utiliza el concepto de **rol** o "grupo con nombre de privilegios que pueden ser concedidos o revocados a usuarios o a otros roles". Soporta dos tipos de privilegios: **privilegios de objetos** y **privilegios del sistema.**

- ***Privilegios de objeto***

Un **privilegio de objeto** es un derecho para realizar una acción concreta en una tabla, vista, secuencia, procedimiento, función o package específicos. Por ejemplo, el privilegio para borrar filas en la tabla *usuarios* es un privilegio de objeto. Dependiendo del tipo de objeto, hay diferentes tipos de privilegios.

Las sentencias GRANT y REVOKE permiten que un usuario conceda o retire privilegios a otros usuarios sobre los objetos de su propiedad.

El creador de un objeto, tiene automáticamente concedidos todos los privilegios que son aplicables a dicho objeto. Por ejemplo, el creador de la tabla *emp* tiene automáticamente los privilegios SELECT, INSERT,UPDATE, DELETE y REFERENCES en *emp*. Además, tiene concedidos estos privilegios con WITH GRANT OPTION, es decir con autoridad para concederlos a otros usuarios.

Sintaxis de la sentencia GRANT

```
GRANT {lista de privilegios | ALL [PRIVILEGES]} [(columna1, columna2,...)] ON  objeto
  TO  {lista de usuarios | roles | PUBLIC}  [ WITH GRANT OPTION ]
```

Donde:

La **lista de privilegios** consiste en uno o más nombres de privilegios separados por comas. (ALTER, DELETE, EXECUTE, INDEX, INSERT, REFERENCES, SELECT, UPDATE)

ALL PRIVILEGES concede todos los privilegios al objeto especificado.

Columna1, columna2,.. especifica las columnas de la tabla o vista sobre las que se conceden los privilegios (solo para INSERT, REFERENCES y UPDATE)

Objeto se refiere al nombre del objeto sobre el que se conceden privilegios. (tablas, vistas, secuencias, procedimientos, funciones, paquetes, sinonimos)

La **lista de usuarios** serán los identificativos de uno o más usuarios separados por comas. Se puede usar PUBLIC, con el significado de todos los usuarios definidos en el sistema.

Si se especifica **WITH GRANT OPTION,** los usuarios a los que se conceden los privilegios podrán a su vez concederlos a otros usuarios.

Un usuario que concede ciertos privilegios a otro, puede posteriormente revocar dicha concesión. La eliminación de privilegios sólo puede ser realizada por aquél que los concedió y para llevar a cabo la revocación de la concesión de uno o mas privilegios se utiliza la sentencia **REVOKE**.

Sintaxis de la sentencia REVOKE

REVOKE {lista de privilegios | ALL [PRIVILEGES] } ON objeto
FROM {lista de usuarios | roles | PUBLIC} [CASCADE CONSTRAINTS]

- **Lista de privilegios, objeto, lista de usuarios** tienen el mismo significado que para GRANT
- **CASCADE CONSTRAINTS**, cuyo significado se explica a continuación.

Supongamos que el usuario **A** concede el privilegio **p** sobre cierto objeto al usuario **B**, quien a su vez lo concede al usuario **C**. ¿Que ocurriría si ahora **A** revoca **p** de **B**?. Si el privilegio no fue concedido con la opción CASCADE, la correspondiente sentencia REVOKE no ejecutaría con éxito. Si el privilegio sí fue concedido con la opción CASCADE, la REVOKE se ejecutaría con éxito y además se revocaría el privilegio al usuario **C**.

Si se elimina un objeto, se revocan automáticamente todos los privilegios concedidos sobre el objeto.

Ejemplos de concesión y eliminación de privilegios de objeto

Conceder al usuario *usu1* el privilegio de actualizar la tabla *emp*

GRANT UPDATE ON *emp* TO *usu1;*

Conceder al usuario *usu2* el derecho de modificar solamente la columna *sal* de la tabla *emp*

GRANT UPDATE *(sal)* ON *emp* TO *usu2;*

Conceder todos los derechos a *usu3* sobre la tabla *emp*

GRANT ALL ON *emp* TO *usu3;*

Quitar a *usu1* el derecho de actualizar la tabla *emp*

REVOKE UPDATE ON *emp* FROM *usu1;*

Quitar a *usu3* los derechos de consulta y actualización sobre la tabla *emp*

REVOKE SELECT, UPDATE ON *emp* FROM *usu3;*

Un usuario puede conceder cualquier privilegio de objeto sobre cualquier objeto de su propiedad a cualquier usuario o rol. Si la concesión incluye la cláusula WITH GRANT OPTION, el usuario que recibe el privilegio puede a su vez concedérselo a otros usuarios. Si no incluye la cláusula, dicho usuario sólo puede usar el privilegio, pero no concedérselo a otros.

- ***Privilegios del sistema***

Un **privilegio del sistema** es el derecho para realizar una acción concreta en un tipo concreto de objeto. Por ejemplo, los privilegios para crear tablespaces (CREATE TABLESPACE) o para eliminar filas en cualquier tabla de la base de datos (DELETE ANY TABLE), son ejemplos de privilegios del sistema. Hay más de 60 privilegios del sistema.

Los privilegios del sistema son concedidos o revocados a usuarios y roles utilizando las sentencias GRANT y REVOKE.

Sintaxis de la sentencia GRANT

GRANT {privilegios del sistema |roles }
TO {lista de usuarios | roles | PUBLIC} [WITH ADMIN OPTION]

Solo los usuarios que tienen concedido el privilegio GRANT ANY PRIVILEGE pueden conceder o revocar privilegios del sistema a otros usuarios. Cuando un usuario tiene concedido un privilegio del sistema con la opción WITH ADMIN OPTION, puede conceder o revocar este privilegio a otros usuarios.

Sintaxis de la sentencia REVOKE

REVOKE {privilegios del sistema | roles} FROM {lista de usuarios | roles | PUBLIC}

Ejemplos de concesión y eliminación de privilegios del sistema

Con la siguiente sentencia se permite al usuario *profesor* crear sinónimos en cualquier esquema. También, puede conceder este privilegio a cualquier otro usuario.

GRANT CREATE ANY SYNONYM TO *profesor* WITH ADMIN OPTION;

El usuario *usu1* puede eliminar tablas en cualquier esquema

GRANT DROP ANY TABLE TO *usu1;*

Quitar el derecho de eliminar tablas a los usuarios *usu1* y *usu2* en todos los esquemas que no sean los suyos, dicho de otra forma, para que los usuarios *usu1* y *usu2* no puedan eliminar tablas que no les pertenezcan, se utiliza la siguiente sentencia REVOKE

REVOKE DROP ANY TABLE FROM *usu1, usu2;*

- ***Funcionalidad de los Roles***

- A un rol se le pueden conceder privilegios del sistema o de objeto.
- A un rol se le pueden conceder otros roles. No obstante, un rol no puede concederse a si mismo ni de forma circular (por ejemplo, el rol A no se puede conceder al rol B si el rol B ha sido previamente concedido al rol A).
- Cualquier rol puede ser concedido a cualquier usuario.

- Cada rol concedido a un usuario puede, en un momento dado, estar habilitado o deshabilitado. Oracle permite a los usuarios habilitar y deshabilitar roles para lograr distintas disponibilidades de privilegios.
- Un rol concedido indirectamente (un rol concedido a otro rol) puede ser explícitamente habilitado o deshabilitado por el usuario. Sin embargo, al habilitar un rol que contiene otros roles, implícitamente se habilitan todos los roles concedidos indirectamente por el rol directamente concedido.
- Cualquier usuario con el privilegio del sistema GRANT ANY ROLE puede conceder o revocar cualquier rol a otros usuarios o roles. Para conceder o revocar roles se usan las sentencias GRANT y REVOKE.
- La sentencia CREATE ROL permite crear un rol con un cierto nombre. El rol se crea vacío, los privilegios se le asignan posteriormente con la sentencia GRANT.
- La sentencia SET ROL permite habilitar y deshabilitar roles.
- A través de la sentencia ALTER USER se pueden establecer los roles por defecto, o sea, aquellos roles concedidos al usuario que se habilitan automáticamente al iniciar la sesión.
- Al crear una base de datos Oracle, se definen automáticamente los roles CONNECT, RESOURCE, DBA, EXP_FULL_DATABASE y IMP_FULL_DATABASE. Estos roles existen por compatibilidad con versiones anteriores de Oracle y se pueden manejar igual que cualquier otro rol.

5.2. Autentificación de usuarios.

Cada base de datos ORACLE tiene una lista de usuarios válidos. Para acceder a la base de datos, un usuario debe de ejecutar una aplicación (SQL*Plus, FORMS, etc.) y conectarse a una instancia de la base de datos usando un nombre de usuario y una contraseña validos.

La validación del nombre y la contraseña dada en el proceso de conexión, lo que se conoce como autentificación del usuario, puede ser realizada por el S.G.B.D. o por el sistema operativo.

En el caso de autentificación por el sistema operativo el S.G.B.D. se limita a comprobar que el identificativo del usuario que intenta acceder a la base de datos está definido en esa base. El sistema operativo se había ocupado, en su momento, de verificar el nombre y la contraseña del usuario.

Las sentencias CREATE USER y ALTER USER permiten definir el método de autentificación que utilizará cada usuario.

Por simplicidad, normalmente se usará uno de los dos métodos para todos los usuarios, aunque es posible usar métodos distintos según los usuarios.

Los nombres de usuario, y en su caso las contraseñas, son creados y modificados con las sentencias CREATE USER, ALTER USER y DROP USER.

6. Descripción de las tablas utilizadas en estas Unidades.

```
CREATE TABLE DEPT
(DEPTNO NUMBER(2) CONSTRAINT PK_DEPT PRIMARY KEY,
DNAME VARCHAR2(14) ,
LOC   VARCHAR2(13) ) ;

CREATE TABLE EMP
(EMPNO     NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
ENAME      VARCHAR2(10),
JOB        VARCHAR2(9),
MGR        NUMBER(4),
HIREDATE   DATE,
SAL        NUMBER(7,2),
COMM       NUMBER(7,2),
DEPTNO     NUMBER(2) CONSTRAINT FK_DEPTNO REFERENCES DEPT);


INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK');
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
INSERT INTO DEPT VALUES  (30,'SALES','CHICAGO');
INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON');
INSERT INTO EMP VALUES
(7369,'SMITH','CLERK',7902,to_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);
INSERT INTO EMP VALUES
(7499,'ALLEN','SALESMAN',7698,to_date('20-2-1981','dd-mm-yyyy'),1600,300,30);
INSERT INTO EMP VALUES
(7521,'WARD','SALESMAN',7698,to_date('22-2-1981','dd-mm-yyyy'),1250,500,30);
INSERT INTO EMP VALUES
(7566,'JONES','MANAGER',7839,to_date('2-4-1981','dd-mm-yyyy'),2975,NULL,20);
INSERT INTO EMP VALUES
(7654,'MARTIN','SALESMAN',7698,to_date('28-9-1981','dd-mm-yyyy'),1250,1400,30);
INSERT INTO EMP VALUES
(7698,'BLAKE','MANAGER',7839,to_date('1-5-1981','dd-mm-yyyy'),2850,NULL,30);
INSERT INTO EMP VALUES
(7782,'CLARK','MANAGER',7839,to_date('9-6-1981','dd-mm-yyyy'),2450,NULL,10);
INSERT INTO EMP VALUES
(7788,'SCOTT','ANALYST',7566,to_date('13-JUL-87')-85,3000,NULL,20);
```

```
INSERT INTO EMP VALUES
(7839,'KING','PRESIDENT',NULL,to_date('17-11-1981','dd-mm-yyyy'),5000,NULL,10);
INSERT INTO EMP VALUES
(7844,'TURNER','SALESMAN',7698,to_date('8-9-1981','dd-mm-yyyy'),1500,0,30);
INSERT INTO EMP VALUES
(7876,'ADAMS','CLERK',7788,to_date('13-JUL-87')-51,1100,NULL,20);
INSERT INTO EMP VALUES
(7900,'JAMES','CLERK',7698,to_date('3-12-1981','dd-mm-yyyy'),950,NULL,30);
INSERT INTO EMP VALUES
(7902,'FORD','ANALYST',7566,to_date('3-12-1981','dd-mm-yyyy'),3000,NULL,20);
INSERT INTO EMP VALUES
(7934,'MILLER','CLERK',7782,to_date('23-1-1982','dd-mm-yyyy'),1300,NULL,10);
COMMIT;
```

Introducción

En las unidades anteriores trabajamos con SQL como lenguaje autocontenido, ejecutando sentencias SQL de forma interactiva desde el entorno SQL*Plus. Ahora vamos a ocuparnos de la utilización de SQL dentro de un programa de aplicación escrito en otro lenguaje de programación.

En un programa de aplicación escrito en un lenguaje podemos tener sentencias de SQL intercaladas que se ocupen del acceso a la base de datos. Se dice en este caso que SQL está **embebido** o **inmerso** en el lenguaje de programación, o que el lenguaje SQL es **huésped** del lenguaje de programación que actúa como lenguaje **anfitrión**.

También existen lenguajes de programación que llevan incorporadas las sentencias SQL. No se trata en este caso de que SQL actúe como lenguaje embebido, sino que forma parte del propio lenguaje de programación. Este es el caso del lenguaje PL/SQL desarrollado por Oracle Corporation.

Con el uso de SQL embebido se aprovechan las ventajas del SQL para acceso a los datos de una base y la potencia y flexibilidad de los lenguajes procedimentales.

En principio, cualquier sentencia SQL que pueda funcionar de forma interactiva puede formar parte de un programa de aplicación, aunque es posible que exista alguna diferencia entre el formato de la sentencia interactiva y el de la del programa de aplicación. Igualmente, hay una serie de sentencias que sirven para implementar las particularidades del uso de SQL como lenguaje de programación y no pueden ser utilizadas de forma interactiva.

Objetivos

- Conocer las características del SQL embebido.
- Describir los elementos que intervienen en la ejecución de SQL como lenguaje huésped.
- Codificar bloques anónimos en PL/SQL.
- Utilizar cursores

Contenidos Generales

1. CARACTERÍSTICAS DEL LENGUAJE PL/SQL. 3
2. ESTRUCTURA DE LOS BLOQUES PL/SQL. 4
3. VARIABLES Y CONSTANTES. 6
4. TIPOS DE DATOS Y EXPRESIONES 8
5. ESTRUCTURAS DE CONTROL 10
5.1. Sentencia IF 10
5.2. Sentencia CASE 12
5.3. Sentencia NULL 13
5.4. Bucles. 13
6. CURSORES 22
6.1. Atributos de cursor 24
6.2. Ejemplos 28
6.3. La sentencia FOR para manejo de cursores 31
6.4. Modificación y eliminación de filas de una tabla usando cursores. 33
7. ALCANCE Y VISIBILIDAD DE IDENTIFICADORES 37

1. Características del lenguaje PL/SQL.

PL/SQL es la extensión procedimental del lenguaje SQL implementada por Oracle Corporation, que combina la sencillez y flexibilidad de SQL para acceso a la base de datos con las características procedimentales de un lenguaje de programación estructurado como IF ... THEN, WHILE o LOOP. Con PL/SQL se pueden usar sentencias SQL para acceder a bases de datos Oracle y sentencias de control de flujo para procesar los datos. Además, se pueden declarar variables y constantes, definir procedimientos y funciones y capturar y tratar errores en tiempo de ejecución. Así, PL/SQL combina la potencia de SQL para manipular datos con la potencia de proceso de los lenguajes procedimentales.

En un programa escrito en PL/SQL se pueden utilizar sentencias SQL de tipo LMD (Lenguaje de Manipulación de Datos) directamente y sentencias de tipo LDD (Lenguaje de Definición de Datos) a través de procedimientos suministrados por Oracle.

Los precompiladores de Oracle y la interfaz de programación OCI reconocen y aceptan bloques de código PL/SQL huéspedes de un lenguaje de tercera generación.

El código PL/SQL, como las sentencias SQL, es ejecutado por el S.G.B.D. Oracle, aunque existen algunas herramientas, como Oracle Forms, que disponen de un motor PL/SQL y lo ejecutan localmente.

Los programas PL/SQL están estructurados en bloques, cualquier programa estará formado al menos por un bloque lógico. Cada bloque lógico puede tener anidados cualquier número de sub-bloques.

Existen distintos tipos de programas PL/SQL:

- Bloques anónimos.
- Disparadores.
- Procedimientos y funciones.
- Paquetes, que contienen procedimientos y funciones.

El código PL/SQL puede estar almacenado en la base de datos (procedimientos, funciones, disparadores y paquetes), junto con un programa de aplicación o en ficheros. La ejecución de los bloques PL/SQL puede realizarse interactivamente desde herramientas como SQL*Plus o Procedure Builder, junto con el programa de aplicación del que forme parte o cuando el S.G.B.D. detecte determinados eventos (disparadores).

2. Estructura de los bloques PL/SQL.

PL/SQL no interpreta un comando cada vez, sino un conjunto de comandos contenidos en un bloque lógico PL/SQL cuya estructura es la siguiente:

```
[DECLARE
declaración de variables, constantes,  excepciones y cursores
BEGIN  [nombre del bloque]
sentencias de SQL,   estructuras …….
[EXCEPTION
    gestión de errores]
END [nombre del bloque];
```

El bloque consta de tres secciones:

DECLARE en ella se declaran las variables, constantes, cursores,subprogramas,….

BEGIN es la sección que contiene las sentencias que se van a ejecutar

EXCEPTION en esta sección se tratan los errores.

Las secciones DECLARE y EXCEPTION son opcionales.

Los bloques pueden contener sub-bloques, es decir, podemos tener bloques anidados. Los anidamientos se pueden realizar en la parte ejecutable y en la de manejo de excepciones, pero no en la declarativa.

```
DECLARE
  Declaraciones;
BEGIN
    DECLARE
      Declaraciones ;
    BEGIN
      Sentencias;
    EXCEPTION
      Excepciones;
    END;
EXCEPTION
  Excepciones;
END;
```

A continuación tenemos un ejemplo de bloque PL/SQL anónimo.

```
DECLARE
                    anos        NUMBER(2);
                    f_alta      DATE;
                    incremento      NUMBER(4,3);
BEGIN
                    SELECT fecha_alta  INTO  f_alta
                           FROM usuarios
                           WHERE num_socio = 122;
                     anos := TRUNC(MONTHS_BETWEEN(SYSDATE, f_alta) / 12);
                     IF anos  >  6 THEN
                            incremento := 1,02;
                           ELSIF  anos  >  4  THEN
                              incremento  :=  1,025;
                                    ELSE
                                    incremento  :=  1,03;
                      END IF;
                      UPDATE usuarios
                        SET cuota_socio  =  cuota_socio  *  incremento
                                  WHERE  num_socio  =  122;
                      COMMIT;
END;
```

Con este programa actualizamos la cuota del socio 122. El incremento de la cuota se realiza analizando la antigüedad del socio.

El programa tiene parte declarativa, en la que se definen dos variables, y parte ejecutable.

Podemos observar que hay sentencias **SQL**, como SELECT ... INTO y UPDATE, sentencias de **control** IF ... THEN y sentencias de **asignación**. El símbolo **:=** representa el operador de asignación.

La ejecución del bloque comienza siempre en la primera sentencia detrás de BEGIN y termina al alcanzar la palabra END o al ejecutar la sentencia RETURN, lo que ocurra antes.

3. Variables y constantes.

PL/SQL permite declarar variables y constantes y utilizarlas en cualquier parte de una sentencia SQL o procedimental en que pueda usarse una expresión. Las variables pueden ser de cualquiera de los tipos vistos en SQL o de otros tipos específicos que veremos. Al declarar una variable se le puede asignar valor. PL/SQL permite definir tipos complejos basados en estructuras de tablas y registros.

Ejemplos.

```
importe   NUMBER (9,2);
nombre    VARCHAR2(20);
iva       NUMBER(4,2):=16.00;
casado    BOOLEAN  := FALSE;
```

La palabra reservada **DEFAULT** puede usarse para inicializar una variable a la vez que se define, en lugar del operador de asignación.

```
casado   BOOLEAN  DEFAULT TRUE;
```

La asignación de valores a una variable en la parte ejecutable de un bloque se puede realizar a través de del operador **:=** , pero también con la cláusula INTO de las sentencias SELECT y con FETCH.

Ejemplos.

```
total_factura := base + iva  - descuento

SELECT base INTO importe FROM  facturas WHERE  numero = 1;
```

Se puede utilizar cualquier expresión PL/SQL valida para asignar valor a una variable.

Podemos utilizar el atributo %TYPE para dar a una variable el tipo de dato de otra variable o de una columna de la base de datos.

```
saldo    importe%TYPE;
```

Con la definición anterior, ***saldo*** es del mismo tipo que ***importe***.

En este otro ejemplo, declaramos la variable *total_factura* con el mismo tipo de dato definido para la columna *base_imponible* de la tabla *facturas.*

```
total_factura    facturas.base_imponible%TYPE;
```

Si una columna tiene la restricción NOT NULL la variable definida de ese tipo no asume dicha restricción, pudiendo por lo tanto ponerla a NULL.

Se puede declarar una variable inicializada como NOT NULL

```
comision  NUMBER(5,2)  NOT  NULL  := 2;
```

Esta variable no aceptaría valores nulos

Se pueden declarar constantes utilizando la palabra reservada CONSTANT. Se deben inicialiazar en el momento de la declaración.

El valor de una constante, una vez determinada, puede utilizarse en cualquier lugar del bloque pero no modificarse. La declaración de constantes facilita la modificación de programas que contienen datos constantes.

```
descuento  CONSTANT  NUMBER(2)  := 10;
```

En el ejemplo anterior si no declaramos la constante *descuento* un cambio en el porcentaje de descuento obligaría a cambiarlo en todas las expresiones donde apareciese.

La declaración de una constante es igual al de una variable salvo que hay que utilizar CONSTANT e inicializarla.

Cualquier variable o constante debe ser declarada antes de ser referenciada en otra sentencia, incluso en otra sentencia declarativa.

```
            saldo    importe%TYPE;                    -- Error
            importe  NUMBER(9,2);
```

El nombre de una variable debe estar formado por una letra seguida, opcionalmente, de más letras, números, signos de dólar, subrayados y signos #. No hay distinción entre mayúsculas y minúsculas

Caracteres permitidos.

PL/SQL utiliza el siguiente conjunto de caracteres para la confección de programas:

- Los caracteres en mayúscula y minúscula de la 'A' a la 'Z'
- Los números del 0 al 9
- Tabulados, espacio y retorno de carro.
- Y los símbolos: () + - * / <> = ¡ ~ ; : . ' @ % , " # $ ^ & _ | { } ¿ [ ]

4. Tipos de datos y expresiones

BINARY_INTEGER Subtipos NATURAL POSITIVE	Almacena enteros con signo en un rango de : –2147483647 a 214783647 Podemos usar los subtipos cuando deseamos restringir la variable a positiva 0 .. 2147483647 1 .. 2147483647
NUMBER [precisión, escala] Subtipos DEC DECIMAL DOUBLE PRECISION FLOAT INTEGER INT NUMERIC REAL SMALLINT	Almacena números (enteros y fraccionarios) en coma fija o flotante en un rango de 1.0E-129 a 9.99. Los subtipos tienen las mismas características que NUMBER y existen por compatibilidad con otras bases de datos y normas ANSI/ISO.
CHAR [máxima longitud] Subtipos CHARACTER	Almacena cadenas de longitud fija. Una columna de tipo CHAR admite 255 caracteres, sin embargo una variable del tipo CHAR puede tener una longitud de hasta 32767 bytes. Los subtipos existen por compatibilidad con otros sistemas y normas ANSI/ISO.

VARCHAR2[máxima longitud] Subtipos VARCHAR STRING	Almacena cadenas de longitud variable, de una longitud de 1 a 32767 bytes. Las columnas definidas con este tipo pueden almacenar hasta 2000 caracteres en la versión que estamos manejando (V7.3); en versiones posteriores tanto los rangos como las capacidades de almacenamiento pueden variar, por lo tanto invitamos a los lectores que estas cifras las tomen como referencia y usen los manuales de la versión que manejen para establecer los valores reales. Los subtipos solo existen por compatibilidad.

BFILE	Almacena la referencia a un archivo del sistema que contiene datos físicos.
CLOB	Almacena cadenas de caracteres de longitud variable, hasta 4GB
BLOB	Almacena un objeto binario, hasta 4 GB (Se puede utilizar para guardar imágenes, gráficos, etc)
RAW[máxima longitud]	Mismas características que CHAR pero para almacenar información en binario. En este caso el juego de caracteres no tiene importancia.
LONG	Cadena de caracteres de longitud variable máximo 3 GB
LONG RAW	Mismas características que LONG almacenando la información en binario..
BOOLEAN	Sólo admite valores TRUE, FALSE y NULL.
DATE	Permite almacenar fechas.
ROWID	Se utiliza para definir variables que contendrán valores correspondientes a la pseudocolumna ROWID, o sea, los valores binarios que identifican físicamente cada fila de datos de una base.
TABLE y RECORD	Tipos compuestos que definen estructuras de datos complejas. Los veremos mas adelante

Expresiones

Una expresión está formada por un conjunto de operandos unidos por operadores. Un operando puede ser una variable, constante, literal o llamada a función que contribuye a la expresión con un valor. Los valores de los distintos operandos se combinan de acuerdo con los operadores dando lugar a un valor único. Las operaciones en una expresión se hacen según el orden de precedencia de los operadores.

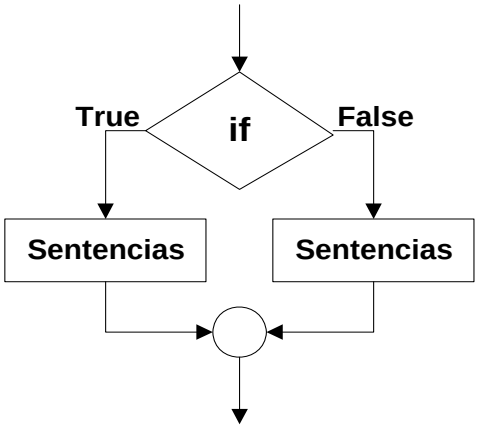
Orden	Operador	Operación
1º	**, NOT	Potencia, negación lógica
2º	+, -	Identidad, negación
3º	*, /	Multiplicación, división
4º	+, -, ||	Suma, resta, concatenación
5º	=, !=, <>, ~=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	Comparación
6º	AND	Conjunción
7º	OR	Inclusión

Como en todos los lenguajes, primero se evalúan las subexpresiones encerradas en paréntesis.

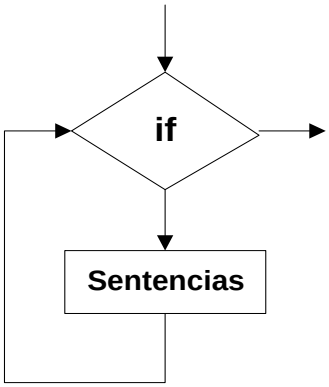
Cuando en una expresión hay operadores con la misma precedencia, por ejemplo, * y **/** , PL/SQL los aplica en un orden indeterminado. Por tanto, se deben usar paréntesis para evitar ambigüedades.

5. Estructuras de control

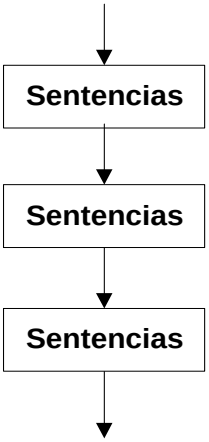
PL/SQL dispone de sentencias que permiten construir las tres estructuras de control de la programación estructurada: ***alternativa, repetitiva*** y ***secuencial.***



Estructura alternativa



Estructura repetitiva



Estructura secuencial

5.1. Sentencia IF

La sentencia alternativa IF permite que se ejecuten unas sentencias u otras según sea el resultado de evaluar una condición.

Sintaxis

```
IF condición THEN
                    sentencias;
END IF;
```

Si la condición se evalúa como TRUE se ejecuta el grupo de sentencias. Si se evalúa como FALSE o NULL no se ejecutaría. Los operadores utilizados en la condición son los mismos que en SQL.

Ejemplo.

```
IF cuota < 50000 THEN
                    cuota := cuota * 1.10;
END IF;
```

Sintaxis con la opción ELSE.

```
IF   condición            THEN
                        Sentencias-1;
             ELSE
                        Sentencias-2;
END IF;
```

Si la condición es verdadera ejecuta el grupo de **Sentencias-1**, si es falsa o NULL el grupo de **Sentencias-2**.

Ejemplo.

```
IF cuota < 50000 THEN
                        cuota := cuota * 1.10;
   ELSE
                        cuota := cuota * 1.05;
END IF;
```

Sintaxis con ELSIF

Se puede realizar un anidamiento de IF a través de la opción ELSIF

```
IF   condición-1 THEN
                        Sentencias-1;
  ELSIF condición-2  THEN
                        Sentencias-2;
  ELSE
                        Sentencias-3;
END IF;
```

Si la primera condición se cumple ejecuta el grupo de **Sentencias-1**, si no se cumple pregunta por la segunda condición, si se cumple ejecuta el grupo de **Sentencias-2** y si no se cumple ejecuta el grupo de **Sentencias-3**.

```
IF cuota > 50000 THEN
                                    cuota := cuota * 1.05;
   ELSIF  cuota  > 30000 THEN
                                    cuota := cuota * 1.10;
       ELSE
                                    cuota := cuota * 1.15;
END IF;
```

5.2. Sentencia CASE

La sentencia CASE permite realizar una ejecución condicional, la diferencia con la IF es que se adapta mejor a las condiciones que implican varias opciones diferentes.

Sintaxis

```
CASE  elemento
       WHEN  valor1       THEN  sentencias;
       WHEN  valor2       THEN  sentencias;
       ……..
       [ELSE  sentencias]
END CASE;
```

Ejemplo

```
DECLARE
       localidad  number := 10;
       nombre varchar2(30);
BEGIN
       CASE localidad
            WHEN 10 THEN nombre := 'Avilés';
            WHEN 20 THEN nombre := 'Oviedo';
            WHEN 30 THEN nombre := 'Santoña';
            WHEN 40 THEN nombre := 'Solares';
            ELSE nombre := 'desconocida';
       END CASE;
END;
```

En el siguiente ejemplo se evalúa la condición en cada cláusula WHEN

```
DECLARE
       localidad  number := 10;
       comunidad varchar2(30);
BEGIN
       CASE
            WHEN localidad IN (10,20) THEN comunidad := 'Asturias';
            WHEN localidad IN (30,40) THEN comunidad := 'Cantabria';
       END CASE;
END;
```

5.3. Sentencia NULL

NULL es una sentencia ejecutable que no realiza ninguna acción. Se utiliza cuando es necesario que exista una sentencia ejecutable en alguna cláusula que así lo exija, por ejemplo en la IF-THEN-ELSE

```
IF condición  THEN       NULL;
              ELSE       Sentencias;
END IF;
```

En el ejemplo que sigue la última sentencia tiene por objeto que la ejecución del bloque no termine con una excepción no tratada. Cuando se produzcan excepciones distintas a las tratadas por las tres primeras sentencias WHEN, serán tratadas por la última, aunque dicho tratamiento consiste en no hacer nada especial.

```
EXCEPTION
                    WHEN condición THEN …….
                    WHEN condición THEN …….
                    WHEN condición THEN …….
                    WHEN  OTHERS  THEN  NULL;
END;
```

En algunos casos, la sentencia NULL mejora la legibilidad del programa.

5.4. Bucles.

Los bucles permiten ejecutar de forma iterativa un grupo de sentencias. En los siguientes apartados estudiaremos cada una de las sentencias que podemos utilizar para construir un bucle.

- ***Sentencia LOOP***

El bucle más simple se corresponde con la sintaxis de la sentencia LOOP que a continuación se describe.

```
LOOP
        Sentencias;
END  LOOP;
```

El formato anterior constituye un bucle infinito, dado que no hay ninguna sentencia vinculada a LOOP que provoque una salida.

Sintaxis con EXIT

```
LOOP
     Sentencias-1;
  EXIT;
     Sentencias-2;
END LOOP;
```

EXIT provoca una salida incondicional de un bucle. Se podría colocar dentro de alguna sentencia condicional, por ejemplo en una IF, para hacer que la salida del bucle dependa de una condición.

```
LOOP
  Sentencias-1;
  IF condición  THEN
     Sentencias-2;
     EXIT;
  END IF;
  Sentencias-3;
END LOOP;
```

Ejemplo.

```
contador := 5;
LOOP
  IF contador > 10  THEN   EXIT;
  END IF;
  contador := contador + 1;
END LOOP;
```

Sintaxis con EXIT WHEN

Con el siguiente formato sale del bucle cuando la condición es verdadera.

```
LOOP
    Sentencias-1;
    EXIT WHEN condicion;
    Sentencias-2;
END LOOP;
```

```
contador := 5;
LOOP
   EXIT WHEN contador > 10;
  contador := contador + 1;
END LOOP;
```

Formato con etiquetas

Los bucles pueden identificarse con una etiqueta, o sea con un identificador encerrado entre ángulos inmediatamente antes del bucle.

```
<<proceso-1>>
LOOP
                                        .....
                                        .....
END LOOP  proceso-1;
```

La etiqueta **proceso_1** es opcional

La sentencia EXIT admite un nombre de bucle.

```
<<proceso-1>>
LOOP
.............
<<proceso-2>>
   LOOP
      .............
     EXIT proceso-1  WHEN  .....;
  END LOOP;
END LOOP;
```

La sentencia EXIT de este ejemplo daría lugar a la finalización tanto del bucle **proceso-2**, el interno, como **proceso-1**, el bucle externo.

Los nombres de las etiquetas siguen las mismas reglas de formación que los de las variables.

- ***Sentencia WHILE***

WHILE constituye un bucle cuyas sentencias se ejecutan mientras la condición asociada a la sentencia sea cierta. Se sale del bucle cuando el resultado de la evaluación sea falso o nulo. (en todas las condiciones el valor nulo se traduce por falso)

Formato de la sentencia WHILE

```
WHILE condición LOOP
   Sentencias;
END LOOP;
```

Alguna de las sentencias tiene que afectar a la condición para que en algún momento pase a falsa, de lo contrario el bucle es infinito.

```
contador := 0
WHILE contador  < 10   LOOP
     ........
     contador :=  contador + 1;
END LOOP;
```

- ***Sentencia FOR***

Esta sentencia constituye un bucle controlado por un índice que progresa en un rango establecido. Una de las diferencias con respecto a la sentencia WHILE es que se conoce el numero de iteraciones. Las instrucciones que están dentro del bucle se ejecutan tantas veces como cambia el índice de valor.

Sintaxis de la sentencia FOR

```
FOR indice IN [REVERSE]  valor_inicial     ..   valor_final  LOOP
        Sentencias;
END LOOP;
```

El índice no tiene que declararse explícitamente, es de tipo NUMBER y toma valores enteros. No se puede modificar su valor pero si se puede consultar, por lo que puede aparecer en expresiones. Fuera del bucle no se puede referenciar. Por defecto el índice se incrementa desde el límite inferior hasta llegar al superior. Con la opción REVERSE por cada iteración el índice decrece desde el rango superior hasta el inferior.

```
FOR  i  IN 1..4  LOOP
Sentencias;  -- las ejecuta cuatro veces, tomando i los valores 1,2,3,4
END LOOP;
```

En el siguiente ejemplo el índice es referenciado en una expresión.

```
FOR  i  IN  1 .. 50  LOOP
     IF  MOD (i, 2)  THEN  sentencias;
     END IF;
END LOOP;
```

Con la opción REVERSE

```
FOR  i IN REVERSE 1..4 LOOP
 Sentencias;  -- las ejecuta cuatro veces, tomando i los valores 4,3,2,1
END LOOP;
```

El rango puede ser dinámico, estableciéndose los límites en tiempo de ejecución.

```
SELECT COUNT(*)  INTO  tope  FROM  actividades;
FOR  I  IN  3 ..  TOPE  LOOP
     .........
END LOOP;
```

Si el límite superior es menor que el inferior, el bucle no se ejecuta. En el caso anterior si tope es menor de 3 no iteraría ninguna vez. Si es 3, se ejecutaría una vez.

Con la sentencia **EXIT - WHEN** se puede salir del bucle antes de llegar al límite del rango.

```
          FOR  i  IN 1..4  LOOP
                   Sentencias-1;
         EXIT WHEN  condición   -- si se cumple la condición salta a ejecutar
                    Sentencias-2;              -- Sentencias-3
         END LOOP;
         Sentencias-3;
```

En un programa PL/SQL los comentarios irán precedidos de dos símbolos guión (--) o bien encerrados entre los símbolos ***/*** y *****/***

- ***Sentencia GOTO***

PL/SQL dispone de una sentencia GOTO, aunque cualquier programa puede ser escrito utilizando solo las sentencias de control anteriores. Su uso no es aconsejable en general por oscurecer la lógica del programa, pero en ciertos casos muy concretos resulta ser una solución adecuada por simplificar el código.

La sentencia GOTO provoca una bifurcación incondicional. Hace que el control pase al punto del programa marcado por una etiqueta. Esta etiqueta tiene que preceder a una sentencia ejecutable o a un bloque PL/SQL.

```
BEGIN
   sentencias-1;
   GOTO  etiqueta-1
   sentencias-2;
   <<etiqueta-1>>
   sentencias-3;
END;
```

```
BEGIN
 SELECT COUNT(*)  INTO  tope  FROM  actividades;
 FOR  i  IN  3 ..  tope  LOOP;
    sentencias;
     IF condición  THEN
           GOTO   salir ;
     END IF;
<<salir>>    -- es ilegal por no seguirle una sentencia
                    --ejecutable o NULL;
  END LOOP;
END;
```

El programa del ejemplo anterior se puede arreglar colocando una sentencia NULL detrás de la etiqueta.

```
BEGIN
     SELECT COUNT(*)  INTO  tope  FROM  actividades;
     FOR  i  IN  3 ..  tope  LOOP
                     sentencias;
                     IF condición  THEN
                           GOTO   salir ;
                     END IF;
      <<salir>>
        NULL;
        END LOOP;
END;
```

Se puede bifurcar a etiquetas que estén en un bloque mas externo, es decir, etiquetas que estén en un bloque que encierra al bloque en el que se encuentra la sentencia GOTO.

```
BEGIN
......
<<etiqueta-1>>;
......
  BEGIN

  Sentencias;
  ......
  GOTO   etiqueta-1 ;
  ......
  Sentencias;

  END;
END;
```

Por último, la sentencia GOTO tienen una serie de restricciones, que en general coinciden con las que tiene esta misma sentencia en los lenguajes de tercera generación

- Un GOTO no puede bifurcar al interior de una IF, un bucle LOOP ni a un sub-bloque
- Dentro de una IF, no puede bifurcar de una cláusula THEN a una ELSE o viceversa.
- No puede bifurcar de un bloque externo a uno interno.
- No puede bifurcar fuera de un subprograma.
- No puede bifurcar desde la sección de excepciones a la sección de ejecución

ACTIVIDADES

Actualizar la comisión de los empleados que no tengan jefe, teniendo en cuenta el departamento al que pertenecen.

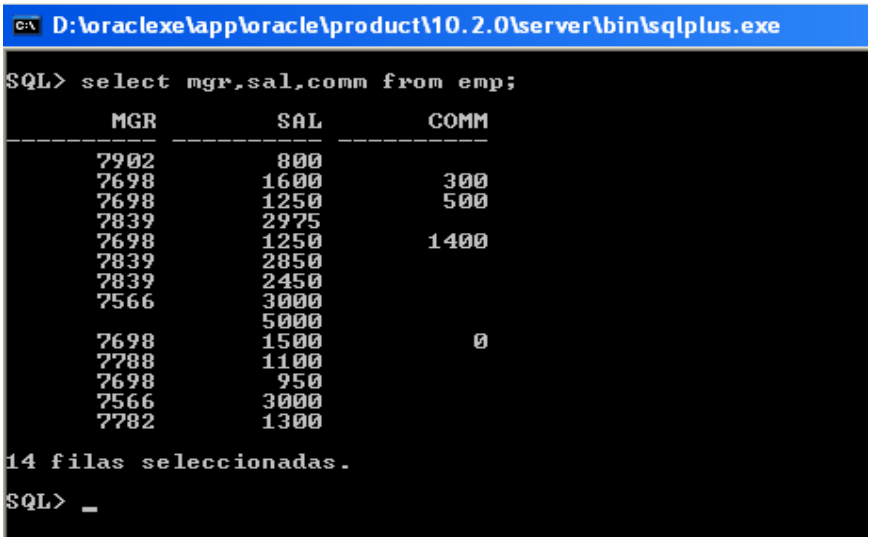
- Los que pertenecen al departamento 10 la comisión será de un 5% del salario.
- Los que pertenecen al departamento 20 la comisión será de un 7% del salario.
- Los que pertenecen al departamento 30 la comisión será de un 9% del salario.
- Los que no pertenecen a ninguno de los anteriores la comisión será de un 2%.

SOLUCION

```
DECLARE
departamento emp.deptno%TYPE;
salario emp.sal%TYPE;
incremento number(3,2);

BEGIN
SELECT deptno INTO departamento FROM emp WHERE mgr IS NULL;

IF    departamento=10 THEN  incremento:=0.05;

ELSIF departamento=20 THEN  incremento:=0.07;

ELSIF departamento=30 THEN  incremento:=0.09;

ELSE   incremento:=0.02;

END IF;

UPDATE emp SET comm = sal * incremento      WHERE mgr IS  NULL;

-- COMMIT;
END;
```

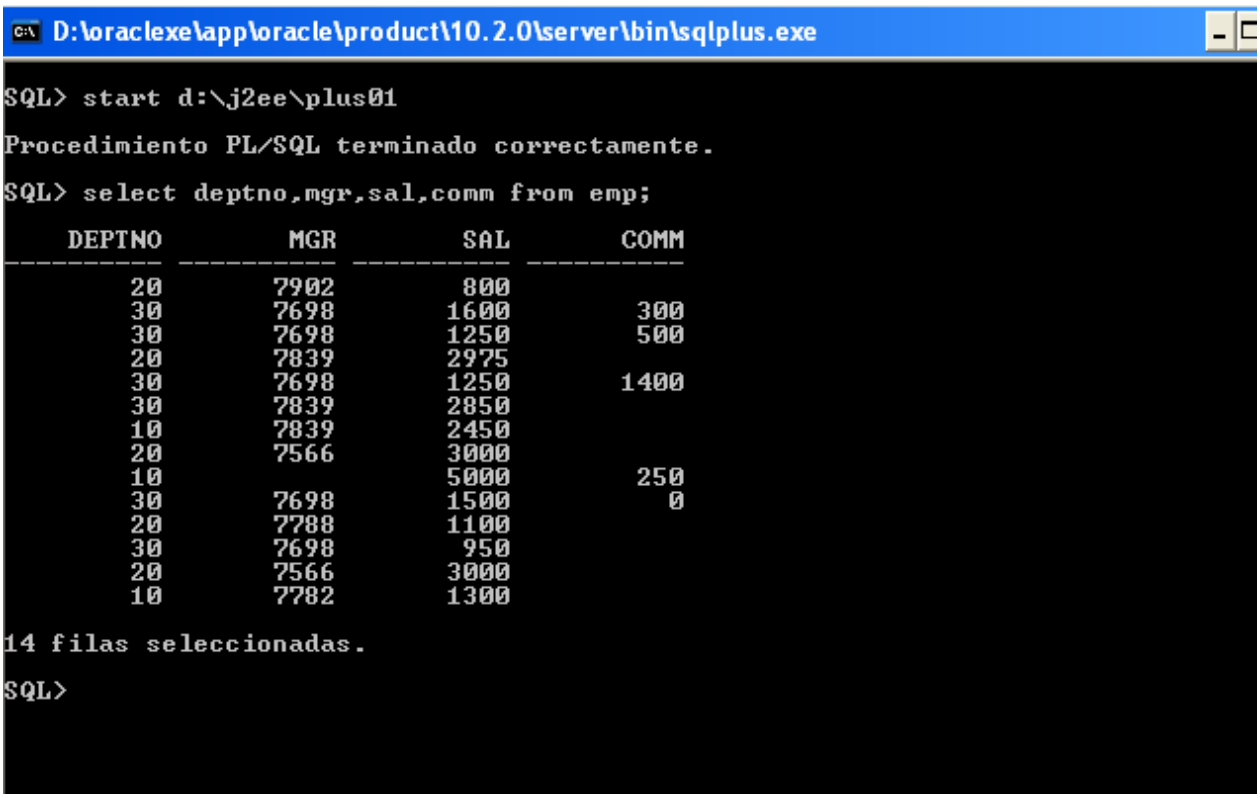
La transacción no se valida, el COMMIT está comentado.

Si queréis comprobar este ejemplo, lo copiáis en un fichero de texto lo guardáis con extensión SQL y lo ejecutáis desde el entorno SqlPlus, como se muestra en las siguientes ventanas. La barra (/) es para que lo ejecute.



1º. Consultamos *emp* para ver cuál es el salario, la comisión y el contenido de mgr

2º. Se ejecuta el bloque PL/SQL con start. A continuación se repite la consulta incluyendo el departamento para comprobar que actualizó correctamente.



Se ve que el empleado del departamento 10 que no tiene jefe (mgr) pasa a tener una comisión de 250 que es el 5% de su salario.

6. Cursores

PL/SQL maneja un área de trabajo, también llamada área de contexto, que utiliza para ejecutar las sentencias SQL y almacenar la información que proporciona una consulta. Con los cursores podemos acceder a la información contenida en esta área de contexto.

Existen dos tipos de cursores: los explícitos, que son los definidos por el programador para acceder a las filas de una consulta, y los implícitos que son los generados por Oracle cuando se ejecuta una sentencia SQL no asociada a un cursor explícito.

Cuando se realiza una consulta el número de filas devueltas por la SELECT puede ser cero, una o más de una. Si el número de filas devueltas es más de una y necesitamos procesarlas para obtener una información, necesitamos utilizar un cursor explícito, de lo contrario se produce el error TOO_MANY_ROWS. En definitiva, utilizar cursores es como trabajar sobre un fichero virtual en memoria, en el que las filas son los registros (utilizando conceptos de los lenguajes de tercera generación) y vamos tratando uno a uno. Es aconsejable trabajar con cursores explícitos.

Sintaxis del Cursor

CURSOR nombre_cursor [(variable [IN] tipo [{:= | DEFAULT} value] ,…….)]
IS SELECT ……

La parte opcional de la declaración del cursor son los parámetros.

Para trabajar con cursores hay que realizar los siguientes pasos: declarar el cursor, abrirlo, acceder a cada una de las filas, procesarlas y cerrar el cursor.

Declarar el Cursor

```
DECLARE
CURSOR  c1  IS  SELECT descripcion, cuota FROM actividades  WHERE cuota  >  5000;
BEGIN
     ........
END;
```

El nombre del cursor **c1** es un identificador, no es una variable, se utiliza para identificar la consulta, no se puede utilizar en expresiones. Las reglas para nombrar los cursores son las mismas que las descritas para nombrar las variables

En la declaración de un cursor se pueden incorporar parámetros, que pueden ser útiles para restringir la selección de filas. Un parámetro puede aparecer en cualquier parte de la sentencia SELECT donde pueda ir una constante.

Ejemplo

```
CURSOR c2  (cuota_minima  NUMBER ) IS
SELECT descripcion, cuota FROM actividades WHERE cuota > cuota_minima;
```

Los valores a los parámetros se pasan cuando se abre el cursor.

.

Abrir el Cursor

Al abrir el cursor se ejecuta la consulta asociada a la declaración del mismo y el conjunto de filas cargadas en el cursor, en adelante las denominaremos registros para distinguirlas de las filas de la tabla, quedan disponibles para un posterior acceso registro a registro. Un puntero señala el primer registro.

OPEN c1;

Con el siguiente OPEN pasaríamos el valor 5000 al parámetro *cuota_minima* al abrir el cursor declarado en el ejemplo anterior.

OPEN c2 (5000);

Acceder a las filas

La sentencia **FETCH** lee el registro marcado por el puntero poniendo el valor de sus columnas en las variables que siguen a INTO. La lectura de un registro hace que el puntero

pase a señalar el registro siguiente, con lo que el acceso se realiza secuencialmente en el orden en que la consulta recuperó las filas.

Sintaxis de la sentencia FETCH

FETCH cursor INTO variable [, variable, …….]

Ejemplo.

```
FETCH  c1  INTO  denominacion, importe;
```

Debe de existir una correspondencia entre columnas de la consulta y variables en la FETCH en cuanto a número y tipo.

Las variables *denominacion* e *importe* de la FETCH anterior se deben declarar y son del mismo tipo que *descripcion* y *cuota*.

Cerrar el Cursor.

Cuando se cierra el cursor los registros dejan de estar accesibles y se liberan los recursos asociados.

CLOSE nombre del cursor;

Para el ejemplo que estamos planteando sería.

```
CLOSE   c1;
```

6.1. Atributos de cursor

Cada cursor tiene asociados cuatro atributos predefinidos. Cuando se abre un cursor estos atributos devuelven información sobre la ejecución de la sentencia SQL asociada al cursor.

%NOTFOUND

Devuelve TRUE si el último FETCH ejecutado no encontró más registros. En caso contrario devuelve FALSE. Después de abrir el cursor y antes del primer FETCH devuelve NULL.

Sintaxis

nombre_cursor%NOTFOUND

Ejemplo.

```
DECLARE
CURSOR  c1  IS  SELECT descripcion FROM actividades  WHERE cuota  >  5000;
nombre actividades.descripcion%TYPE;
BEGIN
     LOOP
       FETCH  c1 INTO nombre;
       EXIT WHEN C1 %NOTFOUND;
        …… mientras haya registros en el cursor se ejecutan estas sentencias
     END LOOP;
     CLOSE c1;
END;
```

Si se hace referencia a un cursor no abierto se produce la excepción predefinida INVALID_CURSOR

%FOUND

Es el atributo cuya función es justamente la contraria a la anterior, devuelve TRUE si el FETCH ha encontrado el registro y FALSE si no lo encontró.

Sintaxis:

nombre_cursor%FOUND

Ejemplo.

```
     ……
     LOOP
       FETCH  c1 INTO nombre;
       IF C1%FOUND THEN
           Si existe el registro ejecuta estas  sentencias;
         ELSE
           EXIT;  --cuando no encuentre más registros sale del bucle
       END IF;
       ……
     END LOOP;
```

%ROWCOUNT.

Devuelve el número de registros accedidos por FETCH hasta el momento.

Sintaxis:

Nombre_cursor%ROWCOUNT

Ejemplo.

```
LOOP
    FETCH  c1 INTO nombre;
    IF c1%ROWCOUNT  > 29  THEN
        Ejecuta las sentencias que están dentro del if cuando haya
        accedido 30 registros del cursor;
    END IF;
    ……
END LOOP;
```

Si el cursor no está abierto se produce la excepción INVALID_CURSOR al hacer referencia a este atributo.

%ISOPEN

Devuelve TRUE si el cursor está abierto y FALSE si está cerrado.

Sintaxis:

Nombre_cursor%ISOPEN

Ejemplo.

```
IF c1%ISOPEN  THEN
      FETCH  c1 INTO nombre;
   ELSE
       OPEN c1;
END IF;
```

Estos atributos permiten también obtener información sobre la última sentencia SQL ejecutada que no tuviese cursor explícito asociado.

Si una sentencia no tiene explícitamente asociado un cursor Oracle le asocia uno. Este cursor no es manipulable desde el programa pero si permite el uso de sus atributos.

Las sentencias SQL de las que podemos tener información a través de estos atributos son: SELECT ... INTO, INSERT, UPDATE, DELETE. Los atributos son los descritos para los cursores explícitos anteponiéndoles el prefijo SQL. Veamos su descripción:

SQL%NOTFOUND

Devuelve TRUE cuando la última sentencia SELECT no recuperó ninguna fila, o cuando INSERT, DELETE, UPDATE no afectó a ninguna fila.

```
INSERT INTO actividades VALUES ('111','BALOMMANO',4000);
IF SQL%NOTFOUND  THEN
     Sentencias    --no insertó. Estas sentencias controlarían el error.
END IF;
```

Si una sentencia SELECT ... INTO no devuelve ninguna fila PL/SQL activa la excepción predefinida NO_DATA_FOUND.

```
SELECT descripcion  INTO  act  FROM actividades WHERE  codigo_actividad  = 999;
```

/* si no existe la actividad para el código 999, se activaría NO_DATA_FOUND pasándose control a la parte de excepciones del bloque. */

```
IF  SQL%NOTFOUND  THEN
        sentencias     -- esta acción nunca se realizaría.
END IF
```

SQL%FOUND

Devuelve TRUE en el caso contrario al anterior, SELECT devuelve alguna fila o INSERT, DELETE, UPDATE afecta a alguna fila

```
INSERT INTO actividades VALUES ('111','BALOMMANO',4000);
IF SQL%FOUND  THEN  sentencias   --sí insertó
END IF;
```

SQL%ROWCOUNT

Devuelve el número de filas afectadas por INSERT, DELETE, UPDATE, y las filas devueltas por una SELECT. Si no hay filas afectadas o devueltas, su valor es cero.

```
DELETE FROM actividades  WHERE cuota  > 4000;
IF SQL%ROWCOUNT  >  5   THEN
    sentencias         --sí se borraron más de 5
END IF;
```

SQL%ISOPEN

Siempre devuelve FALSE, ya que Oracle cierra automáticamente el cursor cuando termina de ejecutar una sentencia SQL.

6.2. Ejemplos

Para comprobar los siguientes ejemplos conectarse como *rasty*. Algunos se hacen desde el entorno SQL y otros desde la ventana del sistema.

```
DECLARE
 numemp NUMBER(4);
 comi   NUMBER(7,2);
 CURSOR c1 IS SELECT empno, comm FROM emp;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO numemp, comi;
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Número EMPLEADO: ' || numemp || '  COMISION: '|| comi);
  END LOOP;
  CLOSE c1;
END;
```

Resultados Explicar Describir SQL Guardado Historial

```
Número EMPLEADO: 7369  COMISION:
Número EMPLEADO: 7499  COMISION: 300
Número EMPLEADO: 7521  COMISION: 500
Número EMPLEADO: 7566  COMISION:
Número EMPLEADO: 7654  COMISION: 1400
Número EMPLEADO: 7698  COMISION:
Número EMPLEADO: 7782  COMISION:
Número EMPLEADO: 7788  COMISION:
Número EMPLEADO: 7839  COMISION:
Número EMPLEADO: 7844  COMISION: 0
Número EMPLEADO: 7876  COMISION:
Número EMPLEADO: 7900  COMISION:
Número EMPLEADO: 7902  COMISION:
Número EMPLEADO: 7934  COMISION:
```

Nota: dbms_output.put_line es un paquete que permite la salida por pantalla, se verá mas adelante

```
DECLARE
 numemp NUMBER(4);
 comi   NUMBER(7,2);
 CURSOR c1 IS SELECT empno, comm FROM emp;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO numemp, comi;
    IF   c1%FOUND THEN
         DBMS_OUTPUT.PUT_LINE('Número EMPLEADO: ' || numemp || '  COMISION: '|| comi);
    ELSE EXIT;
    END IF;
  END LOOP;
  CLOSE c1;
END;
```

Resultados Explicar Describir SQL Guardado Historial

```
Número EMPLEADO: 7369  COMISION:
Número EMPLEADO: 7499  COMISION: 300
Número EMPLEADO: 7521  COMISION: 500
Número EMPLEADO: 7566  COMISION:
Número EMPLEADO: 7654  COMISION: 1400
Número EMPLEADO: 7698  COMISION:
Número EMPLEADO: 7782  COMISION:
Número EMPLEADO: 7788  COMISION:
Número EMPLEADO: 7839  COMISION:
Número EMPLEADO: 7844  COMISION: 0
Número EMPLEADO: 7876  COMISION:
Número EMPLEADO: 7900  COMISION:
Número EMPLEADO: 7902  COMISION:
Número EMPLEADO: 7934  COMISION:
```

```
DECLARE
 numemp NUMBER(4);
 comi   NUMBER(7,2);
 CURSOR c1 IS SELECT empno, comm FROM emp;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO numemp, comi;
    IF   c1%ROWCOUNT < 5 THEN
         DBMS_OUTPUT.PUT_LINE('Número EMPLEADO: ' || numemp || '  COMISION: '|| comi);
    ELSE EXIT;
    END IF;
  END LOOP;
  CLOSE c1;
END;
```

```
Número EMPLEADO: 7369  COMISION:
Número EMPLEADO: 7499  COMISION: 300
Número EMPLEADO: 7521  COMISION: 500
Número EMPLEADO: 7566  COMISION:
```

6.3. La sentencia FOR para manejo de cursores

El manejo de cursores explícitos se simplifica en la mayoría de los casos mediante el uso de un formato de la sentencia FOR disponible para esta función.

FOR nombre_de_registro IN nombre_de_cursor

La sentencia abre el cursor y recupera el primer registro en su primera ejecución. En sucesivas ejecuciones va recuperando cada uno de los registros del cursor. Cuando ya no hay más registros cierra el cursor y se termina la ejecución de la sentencia.

Los valores de los registros son accesibles a través de los campos del registro (nombres de las columnas declaradas en el cursor). El nombre del registro no hay que declararlo y es del tipo **nombre_cursor%ROWTYPE**. El registro no puede ser referenciado fuera del bucle for.

Ejemplo.

```
DECLARE
........
CURSOR  c1  IS  SELECT  codigo_actividad, descripción FROM actividades;
BEGIN
FOR  reg  IN  c1  LOOP   -- abre el cursor y accede a cada una  de los registros
     Sentencias;        -- Por cada registro accedido ejecuta estas  sentencias
     Para hacer referencia a codigo_actividad y descripcion se  utiliza la
     notación reg.codigo_actividad y reg.descripcion
     ......
END LOOP;
       -- cuando termina con todos los registros cierra el cursor
END;
```

Este formato de la sentencia FOR admite el paso de valores a parámetros del cursor, igual que ocurría con la sentencia OPEN.

```
DECLARE
CURSOR  c1  (tope  NUMBER) IS  SELECT  codigo_actividad, descripcion
                 FROM actividades WHERE cuota > tope;
BEGIN
  FOR  reg  IN  c1(5000)  LOOP  -- selecciona las filas con cuota superior a 5000
     ….   Sentencias;
   END LOOP;
END;
```

Cuando existen columnas del cursor resultado de evaluar expresiones, estas se deben renombrar con un alias para permitir su referencia.

```
CURSOR c1 IS
 SELECT denominacion, pvp_tarifa - pvp_coste  ganancia,  existencias
                          FROM    almacen;
```

En el ejemplo anterior para hacer referencia a la expresión (*pvp_tarifa - pvp_coste)* utilizaremos el alias *ganancia*

```
                    IF reg.ganancia > 20000  THEN ……
                     END IF
```

6.4. Modificación y eliminación de filas de una tabla usando cursores.

Como se vio en la unidad anterior, las sentencias UPDATE y DELETE permiten modificar y eliminar, respectivamente las filas de una tabla que verifican la condición especificada en su cláusula WHERE.

Cuando se trabaja en PL/SQL con cursores, estas sentencias admiten también el siguiente formato de la cláusula WHERE:

WHERE CURRENT OF nombre_cursor

```
DELETE FROM actividades   WHERE CURRENT OF  c1;
UPDATE actividades SET  ........   WHERE CURRENT OF c1;
```

Al poner esta cláusula, la fila modificada o eliminada se corresponde con la accedida del cursor especificado.

Para poder usar CURRENT OF es imprescindible haber utilizado la cláusula FOR UPDATE en la sentencia SELECT que define el CURSOR. La cláusula FOR UPDATE hace que al ejecutar la SELECT, con OPEN CURSOR, las filas de la tabla leídas queden bloqueadas. Estos bloqueos son necesarios para evitar los problemas de concurrencia citados en unidades anteriores.

CURSOR c1 IS SELECT cuota FROM actividades FOR UPDATE;

Veamos varios ejemplos:

```
a)
DECLARE
      CURSOR c1 IS SELECT cuota FROM actividades FOR UPDATE;
      nueva_cuota  NUMBER(7);
BEGIN
      OPEN c1;
      LOOP
       FETCH  c1 INTO nueva_cuota;
       EXIT WHEN c1%NOTFOUND;
       UPDATE actividades SET cuota = nueva_cuota * 1.20 WHERE CURRENT OF c1;
      END LOOP;
      --COMMIT; lo comento para que no valide la transacción
END;
```

```
b)

DECLARE
      CURSOR c1 IS SELECT * FROM actividades FOR UPDATE;
BEGIN
  FOR reg IN c1 LOOP
      UPDATE actividades SET cuota = reg.cuota * 1.2  WHERE CURRENT OF c1;
  END LOOP;
     COMMIT; en este caso validaría la transacción
END;
```

```
     c)

     DECLARE
            CURSOR c1(nombre CHAR) IS  SELECT  *  FROM actividades
            WHERE codigo_actividad = nombre FOR UPDATE;
      registro c1%ROWTYPE;
      BEGIN
      OPEN c1('act1');
      LOOP
            FETCH c1 INTO registro;
            EXIT WHEN c1%NOTFOUND;
            DELETE FROM actividades WHERE CURRENT OF c1;
       END LOOP;
       CLOSE c1;
       END;
```

Ejemplo

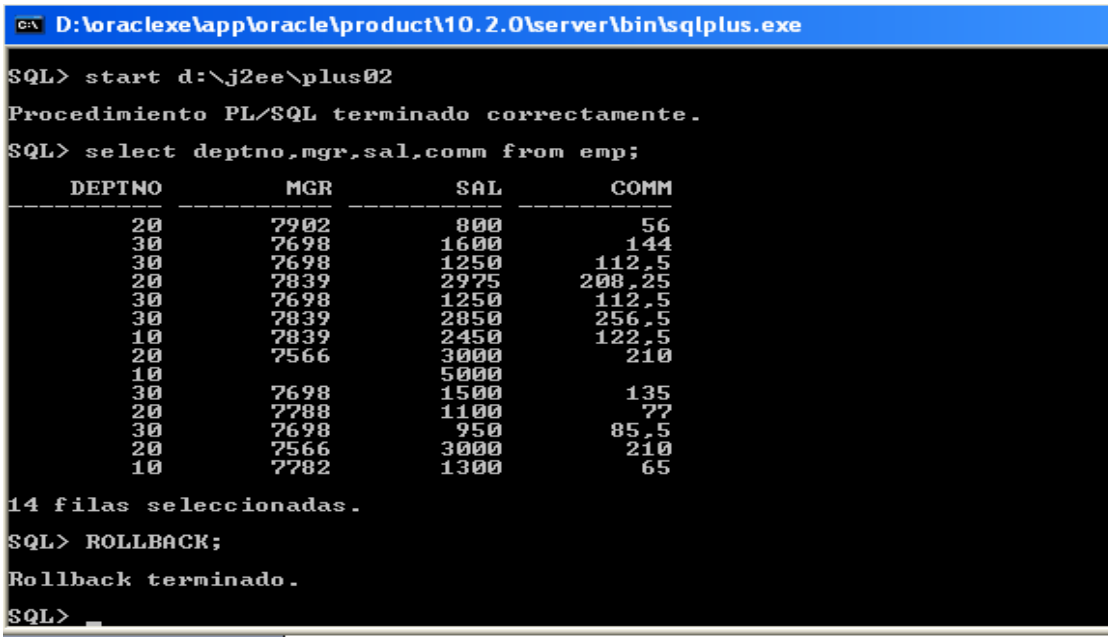
Actualizar la comisión de los empleados que tengan jefe, teniendo en cuenta el departamento al que pertenecen.

- Los que pertenecen al departamento 10 la comisión será de un 5% del salario.
- Los que pertenecen al departamento 20 la comisión será de un 7% del salario.
- Los que pertenecen al departamento 30 la comisión será de un 9% del salario.
- Los que no pertenecen a ninguno de los anteriores la comisión será de un 2%.

En este caso hay que utilizar un cursor dado que hay muchos empleados que tienen jefe.

Solución con FETCH

```
DECLARE
departamento emp.deptno%TYPE;
incremento number(3,2);
CURSOR c1 IS SELECT deptno FROM emp WHERE mgr IS NOT NULL FOR
                                                        UPDATE;
BEGIN
OPEN c1;
LOOP
FETCH c1 INTO departamento;
                      EXIT WHEN c1%NOTFOUND;

                    IF departamento=10    THEN       incremento:=0.05;

                     ELSIF departamento=20 THEN       incremento:=0.07;

                     ELSIF departamento=30 THEN       incremento:=0.09;

                   ELSE   incremento:=0.02;

                  END IF;
UPDATE emp SET comm = sal * incremento           WHERE CURRENT OF C1;
END LOOP;
CLOSE C1;
END;
/
```



En la captura se puede ver el resultado de la ejecución del bloque. Al final se hace *rollback* para deshacer la transacción y que las tablas queden como estaban.

Solución con FOR

```
DECLARE
CURSOR c1 IS SELECT deptno FROM emp WHERE mgr IS NOT NULL FOR
                                                        UPDATE;
incremento number(3,2);
BEGIN
FOR reg IN c1 LOOP
                IF     reg.deptno=10   THEN   incremento:=0.05;

                  ELSIF reg.deptno=20  THEN   incremento:=0.07;

                  ELSIF reg.deptno =30 THEN   incremento:=0.09;

                   ELSE incremento:=0.02;

                END IF;

UPDATE emp SET comm = sal * incremento     WHERE CURRENT OF C1;
END LOOP;
END;
-------------------------------------------------
```

En este caso no hace falta declarar la variable departamento, la notación *reg.deptno* hace referencia al departamento del registro donde está el puntero del cursor que se corresponderá con la fila de la tabla accedida.

Otro ejemplo

Actualizar a la columna **CATEGORÍA** de la tabla **EMP** con una cadena de asteriscos teniendo en cuenta que se escribe un asterisco por cada mil unidades de sueldo. Si la tabla no tiene la columna, modificar la estructura para añadirla.

Solución

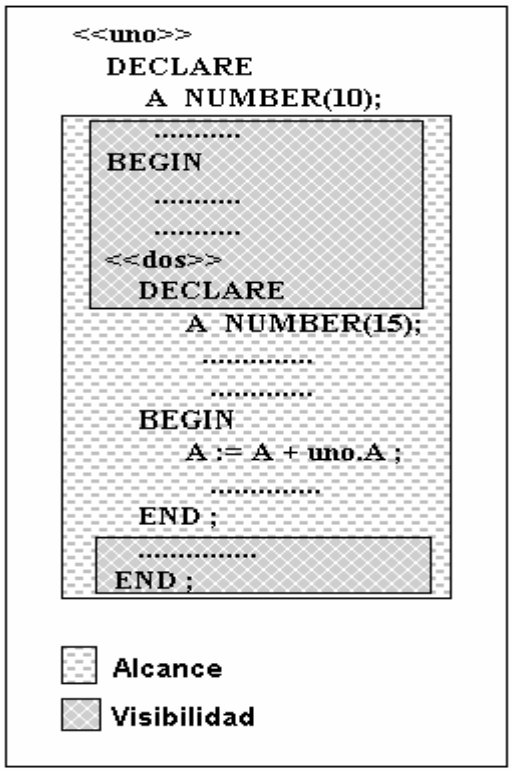
```
DECLARE
 num NUMBER:=0;
 CURSOR C1 IS SELECT sal FROM emp FOR UPDATE;
BEGIN
 FOR REG IN C1 LOOP
       num:=trunc(REG.sal/1000);
       IF num>=1 THEN
             UPDATE emp
                   SET CATEGORIA=lpad('*',num,'*')
                   WHERE CURRENT OF C1;
       END IF;
 END LOOP;

END;
```

7. Alcance y visibilidad de identificadores

Cuando hablamos de identificadores nos estamos refiriendo a los nombres de los distintos objetos de un programa, es decir, a los nombres de las variables, constantes, excepciones, cursores, subprograms y paquetes. También tratamos de las etiquetas que pueden identificar a los bloques PL/SQL o designar determinados puntos de un programa.

Las referencias a un identificador se resuelven según su alcance y visibilidad. El **alcance** de un identificador es la región del programa desde la que se puede referenciar. Un identificador es **visible** en aquella parte del programa en que se puede referenciar con un nombre sin calificar.

Los identificadores declarados en un bloque son locales a ese bloque y globales a todos sus sub-bloques. Si un identificador global es redeclarado en un sub-bloque, ambos identificadores mantienen su alcance, pero dentro del sub-bloque, a partir del punto en que se declara, solo es visible el local.



Alcance y visibilidad de la variable A **del bloque** *uno*

En el ejemplo, se ve el alcance y la visibilidad de la variable **A** declarada en el bloque externo. Para referirse a esta variable en el bloque ***dos***, hay que calificarla (**uno.A**). El alcance y la visibilidad de la **A** del bloque ***dos*** coincidirían con la parte del bloque que va desde el punto donde se declara hasta el final.

Desde un bloque no se pueden referenciar identificadores declarados en sus sub-bloques, ni en otros bloques que no lo contengan.

A continuación tenemos un ejemplo de alcance y visibilidad de identificadores.

```
DECLARE
  a  CHAR;
  b  REAL;
BEGIN
  -- Aquí se puede utilizar: a (CHAR) y b
  DECLARE
        a  INTEGER;
        c  REAL;
  BEGIN
        /* Aquí se pueden referenciar: a (INTEGER), b y c.
         * a (CHAR) tiene alcance, pero no se puede referenciar
         * porque no hay forma de calificarlo.
         */
  END;
  DECLARE
        d  REAL;
  BEGIN
        -- Identificadores referenciables: a  (CHAR), b y d.
  END;
        -- Identificadores referenciables: a (CHAR) y b.
END;
```