Unidad Didáctica IV.- Aplicaciones con conexión a B.D. Repaso SQL

*CASO. En **SI Informática** están ya trabajando a pleno rendimiento con la aplicación de **Gestión del Taller Informático** que le ha encargado un cliente. El taller se llama **El PC Feliz** y fiel a su nombre necesita una completa aplicación de gestión que permita dar un buen servicio a sus clientes y sus ordenadores, por ello han confiado en la experiencia y buen hacer de nuestros amigos de **SI Andalucía** para que lleven a cabo la aplicación. Los informáticos de SI Andalucía ya conocen las ventajas de utilizar una arquitectura tan potente y flexible como **Oracle ADF** y han visto en **JDeveloper** la herramienta ideal para aprovechar sus posibilidades. **Carmen** y **María** han estado trabajando en el diseño y planificación del desarrollo de la aplicación, ahora saben que la aplicación necesitará almacenar los datos del Taller en una base de datos. Como gestor de Base de Datos han propuesto Oracle ya que es muy potente y conocido, y además se integra perfectamente con Oracle ADF y JDeveloper. Por tanto comienzan a repartir el trabajo, **José** está trabajando en otro proyecto y de momento no podrá centrarse en éste aunque más adelante podrá reintegrarse en el grupo. **Carmen** y **María** se ofrecen para diseñar la base de datos y diseñar las capas del modelo de datos, pero **Víctor**, que ya estuvo trabajando en el diseño de la interfaz de la aplicación dice que quiere unirse al grupo, aunque no tenga profundos conocimientos del tema. **Carmen** y **María** le tranquilizan y le dicen que ellas le explicarán todo lo que necesite saber, pero a cambio él deberá participar en el desarrollo de la aplicación con todo lo que aprenda. **Víctor** ilusionado da las gracias y se ponen manos a la obra. Lo primero será diseñar la base de datos. Como **Carmen** y **María** tienen experiencia en el tema se encargan de hacerlo, y explican a **Víctor** la importancia del tema, ya que un mal diseño de la BD implicará una mala aplicación o la necesidad de hacer cambios después que llevarán un trabajo extra innecesario.*

Aplicaciones con conexión a base de datos

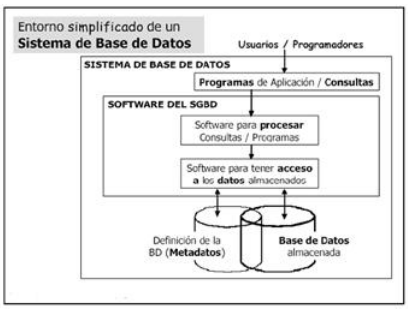
Sistemas Gestores de bases de datos

¿Tienes "frescos" los conceptos de bases de datos? En el Módulo Profesional de "***Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE***" se han descrito los fundamentos de los sistemas gestores de bases de datos por lo que te recomendamos que revises sus contenidos ya que te serán de mucha utilidad para esta unidad. De todas formas, por si no has cursado todavía ese módulo, seguro que te viene bien hacer un repaso a los conceptos fundamentales de los sistemas gestores de bases de datos y eso es lo que vamos a hacer aquí, en los tres primeros apartados de la unidad.

Si ya cursaste el módulo de Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas Case, o si estas familiarizado con los conceptos de bases de datos, podrás revisar rápidamente esos 3 primeros apartados, sólo para refrescar conceptos, y centrarte en el estudio de la parte final de la unidad, que comienza con el apartado dedicado a "CONEXIÓN CON LA BD DESDE JDEVELOPER", que sí trata conceptos nuevos.

El mundo actual está caracterizado por la Sociedad de la Información en la que todos necesitamos acceder y manejar grandes cantidades de datos de forma sencilla y rápida. ¿Sería esto posible sin la informática? ¿Qué ofrece el software para manejar toda esta información? Para que esto sea posible la Informática ha proporcionado como herramienta de ayuda las bases de datos. Podemos definir una **Base de Datos** como un conjunto de datos relacionados entre sí, almacenados en soporte informático, junto a un conjunto de herramientas que permiten su acceso, manipulación y organización de forma eficaz y segura.

A partir de esta definición podemos ver que las bases de datos hacen referencia tanto al conjunto de datos organizados como a las herramientas que los manejan, para lo cual, ¿qué es necesario utilizar? Efectivamente, como ya sabes necesitaremos un **Sistema de Gestión de Bases de Datos** (S.G.B.D. o en inglés, DBMS-Data Base Management System).



Los SGBD aportan grandes ventajas, como permitir al diseñador de aplicaciones tener una visión abstracta de los datos, la visión que tiene de la información el usuario y la manipulación de los datos almacenados en la base de datos es independiente de cómo estén almacenados físicamente, disminuyen las redundancias e inconsistencia de datos, aseguran la integridad de los datos, facilitan el acceso a los datos, aumentan la seguridad y privacidad de los datos, mejoran la eficiencia, permiten compartir datos y accesos concurrentes, facilitan el intercambio entre distintos sistemas, etc.

PARA SABER MÁS

En esta unidad sólo vamos a repasar algunos conceptos básicos, si quieres profundizar en los sistemas de gestión de bases de datos puedes consultar esta página Web de la Universidad Jaume I de Castellón:

Apuntes sobre las Bases de Datos [Versión en caché]

En la wikipedia también tenemos información importante relacionada con los SGBD tal y como puedes leer en estos dos enlaces de Internet:

Las Bases de datos en la Wikipedia [Versión en caché]

Los SGBD en la Wikipedia [Versión en caché]

Autoevaluación

¿Qué es una base de datos?

a) Una base de datos es una colección de archivos relacionados que almacenan tanto los datos de la información del problema como la relación entre ellos y sus restricciones.
b) Una base de datos es un conjunto de datos relacionados entre sí almacenados en soporte informático y que permite acceso directo a los mismos, junto a un conjunto de programas que manipulan esos datos.
c) Base de datos es un conjunto exhaustivo no redundante de datos, organizados independientemente de su utilización y su implementación en máquina, accesibles de forma eficiente y sencilla por diferentes usuarios concurrentes con una visión distinta de la información.
d) Todas las respuestas anteriores son correctas.

Comprobar

Características de un Sistema de Gestión de Base de Datos

Ya hemos recordado qué es un SGBD, ¿recordamos ahora cómo es su estructura? Por si no lo recuerdas vamos a explicar algunos conceptos básicos.

Los SGBD están construidos a partir de una **arquitectura** que proporciona diferentes niveles de abstracción (nivel interno o físico, nivel lógico o conceptual, nivel externo o vista de usuario) para hacer más sencilla la labor al usuario de la base de datos. La idea fundamental es separar los programas de aplicación de la base de datos física mediante la definición de distintos modelos de datos. Esta estructura permite que el modelo lógico de los datos sea independiente del modelo físico. La base de datos no sólo contiene los datos, también almacena su descripción (metadatos) en el **diccionario de datos**, lo cual permite que exista independencia de datos lógica-física. El diccionario de datos es un catálogo en el que se almacenan el nombre, tipo y tamaño de los datos, las relaciones entre los datos, las restricciones de integridad sobre los datos, la información sobre los usuarios autorizados a acceder a la base de datos y sus niveles de acceso, estadísticas de utilización, etc.

Para aprovechar las diferentes herramientas y servicios, los SGBD ofrecen lenguajes e interfaces apropiadas para cada tipo de usuario: administradores de la base de datos, diseñadores, programadores de aplicaciones y usuarios finales. ¿Sabes cómo se denominan esos lenguajes?:

- **Lenguaje de definición de datos** (DDL: Data Definition Language): Se utiliza para especificar el esquema de la base de datos y su modelo de datos.
- **Lenguaje de manejo de datos** (DML: Data Manipulation Language): Permite realizar sobre los datos las operaciones de consulta, actualización, inserción y borrado.
- **Lenguaje de control de datos** (DCL: Data Control Language): Proporciona acceso a diferentes funciones de administración de la base de datos como datos estadísticos, control del almacenamiento, copias de seguridad, protección de accesos, etc.

Nosotros utilizaremos el **Lenguaje Estructurado de Consultas** (SQL: Structured Query Language) que proporciona sentencias para realizar operaciones de DDL, DML y DCL. SQL fue publicado por el ANSI en 1986 (American National Standard Institute) y ha ido evolucionando a lo largo del tiempo. Además, los SGBD suelen proporcionar otras herramientas que complementan a estos lenguajes como generadores de formularios, informes, interfaces gráficas, generadores de aplicaciones, etc. Para estas tareas utilizaremos **Oracle** y **JDeveloper**.

PARA SABER MÁS

Existen muchos SGBD en el mercado informático, tanto comerciales como de software libre. A continuación te comentamos las páginas Web de algunos de ellos para que los visites, nosotros utilizaremos Oracle en este módulo:

SGBD INTERBASE
Microsoft SQL Server
Oracle (Versión en Español)
MySQL
PostgreSQL

Hemos dicho que vamos a usar SQL como lenguaje DDL. Puedes conocer más sobre él en este enlace donde existe un buen tutorial:

Manual de SQL

Autoevaluación

El lenguaje SQL fue propuesto inicialmente por...

a) El Instituto Americano de Estandarización.
b) Por el Organismo Internacional de Estandarización.
c) Por el International Business Machine.
d) Ninguno de los anteriores.

Comprobar

Modelo Relacional

Hay diferentes tipos de SGBD en función del modelo de datos utilizado (jerárquico, red, relacional, orientado a objetos, etc.), nosotros nos vamos a centrar en el más utilizado para nuestros propósitos: el modelo relacional. ¿Recuerdas en qué consistía? Repasemos

algunos conceptos básicos.

El modelo relacional se basa en un concepto matemático denominado "**relación**", que gráficamente se puede representar como una tabla formada por filas y columnas. Esta percepción es sólo a nivel lógico, ya que a nivel físico puede estar implementada mediante distintas estructuras de almacenamiento. Cada tabla representa una **entidad** o concepto del problema (archivo) que incluye filas para almacenar los **registros** de la entidad, y columnas que serán los **campos** o atributos de dicho registro.

Hemos dicho que la base de datos recoge un **modelo de datos**, este modelo debe dar respuesta al problema que queremos abordar. Por ejemplo, si queremos crear una **base de datos para una biblioteca**, el modelo de datos deberá recoger las necesidades de la biblioteca, es decir recogerá los datos de los libros (título, autor, ISBN, etc.), las personas que utilizan la biblioteca (nombre, DNI, teléfono, etc.), los espacios, etc. Toda esta información quedará organizada en tablas según el modelo relacional, por ejemplo:

- habrá una tabla de libros (entidad),
- cada fila contendrá los datos de un libro (registro), y
- las columnas serán el título del libro, el autor, ISBN, etc. (campos).

De forma equivalente otra tabla de usuarios que incluirá los datos de las personas que usan la biblioteca, y será necesario recoger el dato de que una persona tiene prestado un determinado libro, para ello se establecen las relaciones entre la entidad libro y la entidad usuario de biblioteca. Para crear estas relaciones es necesario que existan campos que identifiquen unívocamente a los registros, estos campos se llaman **clave primaria**, por ejemplo el DNI identifica a los usuarios, y el ISBN a los libros.

PARA SABER MÁS

Podemos profundizar en el modelo relacional visitando este enlace donde se justifica el uso y ventajas del modelo relacional:

Introducción al modelo relacional [Versión en caché]

En esta otra web de la Universidad de Santiago de Compostela se hace una introducción teórica y práctica a estos conceptos de bases de datos relacionales:

Teoría de Bases de Datos relacionales

Teoría de Bases de Datos relacionales

Autoevaluación

¿A qué se le denomina modelo de bases de datos?

a) Dada una situación real, a la modelización del problema que se pretende tratar.
b) Es la estructura de la base de datos.
c) Al conjunto formado por la base de datos y los datos.
d) Todas las anteriores son correctas.

Comprobar

Diseño de la Base de Datos

¿Cómo llegamos desde el análisis del problema hasta el diseño del modelo de datos con relaciones y tablas? La Ingeniería del Software nos ofrece distintas herramientas tanto a nivel de teorías como de aplicaciones software de ayuda. Así, disponemos del **modelo Entidad-Relación** que permite modelizar un problema y obtener entidades (libro), relaciones (préstamo: libro-persona) y atributos (autor).

El objetivo de este modelo es simplificar el **diseño de bases de datos** partiendo de la descripción y análisis de la realidad, que establecen los requerimientos del sistema, es decir, se obtiene una representación de la realidad que responda a las propiedades de la misma, siendo una imagen fiel del comportamiento del mundo real.

A partir del **modelo Entidad-Relación**, aplicamos un nuevo proceso de modelización para obtener el modelo relacional que representará la realidad mediante **tablas** y **relaciones** entre las tablas.

¿Ya tenemos el diseño de la base de datos?

Recuerda que nos quedan otras cosas por hacer, como es el proceso de **normalización**, que consiste en analizar las dependencias entre los campos y aplicar una serie de reglas para asegurarnos la eliminación de redundancias e inconsistencias en las tablas que constituyen el diseño de nuestra base de datos. En ocasiones esta Normalización se traduce en la separación de los datos en diferentes tablas manteniendo la información original del problema. Las reglas en las que se basa la **Teoría de la Normalización** son conocidas con el nombre de Formas Normales. De esta manera transformamos, si es necesario, las tablas obtenidas en el **Modelo Relacional** en un conjunto de tablas más simples y fáciles de mantener, tanto desde el punto de vista lógico, como físico.

Habíamos indicado que la Informática nos proporcionaba marcos teóricos como el modelo Entidad-Relación y herramientas software de ayuda.

¿De qué estamos hablando?

Efectivamente nos referimos a las **aplicaciones CASE (Computer Aided Software Engineering: Ingeniería Software Asistida por Ordenador)** que nos facilitan la labor automatizando parte del proceso de modelado y diseño, aumentando la productividad y mejorando la calidad del trabajo. Hay múltiples herramientas de este tipo como *Xcase, CASE Studio - TOAD Data Modeler, DBDesigner, DMS Software Reengineering Toolkit, Eclipse, Oracle Designer, GeneXus, etc.*

Por tanto el proceso de desarrollar nuestra base de datos se realizará con la ayuda de herramientas CASE. Primero **modelizamos el problema aplicando el modelo Entidad-Relación** para obtener una representación de la realidad, a partir del cual obtenemos el modelo relacional de la base de datos, después aplicaremos un proceso de Normalización para obtener el diseño de tablas cumpliendo con las **Formas Normales**. La herramienta CASE generará las sentencias necesarias en el DDL (usaremos SQL) que respondan a las tablas y relaciones desarrolladas para introducirlas en el SGBD (Oracle en nuestro caso) y crear la base de datos que responda a las especificaciones iniciales con eficiencia, seguridad y calidad. Las sentencias SQL quedan agrupadas en un Script SQL

generado por la herramienta CASE y que debe ser ejecutado en el SGBD para tener nuestra base de datos lista para ser usada.

Este proceso, paso desde el mundo real al modelo en SQL, se ha descrito en el Módulo Profesional de "*Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE*", por lo que no vamos a repetirlo aquí, sino que te vamos a proporcionar directamente la descripción de la base de datos en el lenguaje DDL (SQL en nuestro caso) para que podamos crear la base de datos sobre la que desarrollaremos una aplicación que utilice la información almacenada.

Para el ejemplo anterior de la **biblioteca** podemos ver el proceso en la siguiente animación, donde se muestra cómo, tras el análisis de una situación dada del mundo real, ésta se va modelizando a partir de un Modelo Entidad-Relación, después obtenemos un esquema relacional basado en tablas, que serán normalizadas, y por último conseguimos un conjunto de sentencias SQL que interactúan con el sistema gestor de bases de datos para crear la base de datos que responde al problema del mundo real planteado inicialmente.

En la siguiente animación puedes ver esto explicado de forma gráfica sobre un ejemplo de Diseño de una base de datos para una biblioteca.

Diseño de una base de datos para una biblioteca

En cualquier caso, debes tener presente que esto es sólo una sencilla aproximación al tema para que recuerdes algunos conceptos básicos, pero es necesario profundizar en estos contenidos, pues el diseño de la base de datos es un tema fundamental y complicado. Te volvemos a insistir en la necesidad de que conozcas los contenidos del Módulo Profesional de "Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE".

Para poder beneficiarnos de las ventajas de las Bases de Datos necesitaremos un SGBD y será necesario tener la formación necesaria para saber manejarlo. A lo largo de nuestra unidad veremos los fundamentos para desarrollar aplicaciones informáticas que manejen información a través de bases de datos. Para ello en esta unidad manejaremos el sistema de gestión de base de datos **Oracle Express**, y el entorno de programación **JDeveloper** basado en **Java**.

PARA SABER MÁS

Hemos hecho una pequeña introducción al Modelo Entidad-Relación pero puedes profundizar más visitando este enlace:

Desarrollo del Modelo Entidad-Relación [Versión en caché]

Se ha descrito la Teoría de la Normalización de tablas. En esta página puedes conocer cómo se obtienen las diferentes Formas Normales aplicando diferentes reglas:

Formas normales [Versión en caché]

El tema de la ingeniería del software y las herramientas CASE es muy amplio y complejo, constituyendo una parte importante de la informática profesional de hoy día. Aquí tienes un enlace donde puedes leer más sobre las herramientas CASE:

Herramientas CASE [Versión en caché]

Autoevaluación

La normalización se realiza para...

a) Minimizar la redundancia.
b) Prevenir anomalías de inserción, borrado y actualización de datos en las Bases de Datos.
c) Intenta mejorar el diseño de una base de datos.
d) Todas las respuestas son correctas.

Comprobar

*CASO. **María** sabe que el diseño de la base de datos no depende del SGBD que se utilice, por lo que se centran en el diseño sin detenerse aún en el SGBD. Gracias a sus estudios en el Módulo Profesional de "**Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE**" conocen todo el proceso necesario para obtener un buen diseño de la base de datos. **Carmen** explica a **Víctor** la necesidad de utilizar una herramienta CASE de ayuda al diseño, ya que facilita el trabajo y garantiza la calidad del mismo. Una vez que tienen el diseño de la base de datos, con la herramienta CASE han obtenido un script SQL que se disponen a ejecutar en el SGBD Oracle Express para crear la base de datos. También le explica a **Víctor** que JDeveloper dispone de una herramienta CASE. **Víctor** no ha instalado ni usado nunca este SGBD por lo que está algo preocupado, pero **Carmen** que sabe lo sencillo que es descargar, instalar y ejecutar el script SQL le dice que en una tarde le explica todo el proceso y crean la BD. **Víctor** incrédulo apuesta un refresco a que tardarán más. **Carmen** empieza a explicar...*

Descarga e instalación de Oracle Express

Hasta aquí hemos repasado algunos conceptos teóricos sobre las bases de datos, pero seguro que tienes ganas de practicar así que vamos a empezar con el uso práctico de la base de datos.

Oracle es uno de los SGBD relacional más extendido y utilizado en la actualidad, es un producto maduro y muy potente, cuenta con numerosas utilidades y herramientas que van desde el diseño CASE hasta aplicaciones 4GL, pasando por entornos de desarrollo o utilidades de administración.

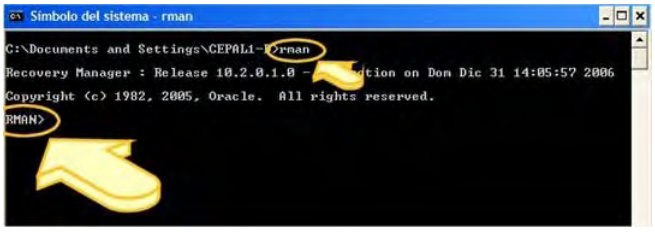
Por tanto, es ideal para nuestros propósitos, por ello vamos a aprender a instalarlo y usarlo a nivel de administración básica. Para nuestras prácticas utilizaremos una versión del SGBD Oracle de descarga y utilización gratuita. Esta

versión se denomina **Oracle Express**, y la versión actual es la 10g. Esta versión limitada de Oracle sólo puede ejecutarse en servidores con un sólo procesador y con hasta 1 GB de RAM, y puede manejar un tamaño máximo de 4 GB de almacenamiento en el disco. Lo qué es más que suficiente para nuestros propósitos. Para ver cómo se descarga e instala Oracle Express, podemos consultar esta animación:

Descarga e instalación de Oracle Express 10g

No olvides que al instalar el SGBD de Oracle has instalado un cliente y un servidor de base de datos. Cuando accedemos a nuestra base de datos desde el cliente, lo que hacemos es acceder al servidor de Oracle. Por ello, es necesario arrancar el servidor de Oracle antes de poder utilizar el cliente. Si has realizado la instalación para Windows puedes ver que se ha creado un nuevo grupo de programas "**Base de Datos Oracle 10g Express Edition**". Veamos los accesos directos creados:

- **Obtener ayuda.**
- **Ejecutar Línea de Comandos SQL,** tenemos una posibilidad interesante de manejar el SGBD desde esta ventana de texto para comandos SQL, para conectar a la BD escribiremos **CONNECT**
- **Iniciar Base de Datos (BD),** pone el marcha el motor de la BD, es decir carga en memoria los procesos del servidor necesarios para usar Oracle. El sistema se encargará de iniciarlo automáticamente, por lo que no será necesario usar este enlace a menos que hayamos detenido el servidor.
- **Introducción,** es un tutorial de Oracle que te recomendamos consultar.
- **Ir a la Página Inicial de Base de Datos,** abre un navegador Web y lanza la página inicial de la interfaz Web de Oracle Express.
- **Parar Base de Datos,** detiene el motor de base de datos.
- **Realizar Copia de Seguridad de la Base de Datos**
- **Restaurar Base de Datos:** Como puedes ver Oracle Express proporciona un método sencillo y eficaz de gestionar las copias de seguridad. Te recomendamos que lo pruebes. En realidad estos accesos directos utilizan una herramienta más potente y completa llamada **RMAN** que también podrás usar desde la consola de comandos SQL.



Seguro que tienes ganas de entrar en Oracle. ¿Probamos este SGBD?

Bien, basta con seleccionar el acceso directo anterior "*Ir a la Página Inicial de Base de Datos*", vemos que se inicia el navegador con la dirección http://127.0.0.1:8080/apex

¿Recuerdas su significado? **127.0.0.1** es la **dirección IP local** de nuestro propio equipo, esto es debido a que Oracle instala un **servidor Web** para gestionar las páginas de administración y gestión de la base datos (aunque podremos administrar la BD de otras formas, por ejemplo con comandos SQL tecleados en el cliente ejecutado en el acceso anterior "*Ejecutar Línea de Comandos SQL*)

¿Qué hay que hacer ahora?

Efectivamente, debemos identificarnos ante el sistema. De momento podemos entrar como administrador del sistema con el usuario **SYSTEM** que creamos durante la instalación de Oracle. Si quieres hacer un recorrido por el SGBD Oracle Express, puedes ver la siguiente animación. No olvides que debes salir del sistema pulsando en **desconectar**.

Un paseo por Oracle Express 10g

PARA SABER MÁS

La Compañía Oracle dispone de una amplia variedad de productos de gran calidad, como puedes ver en esta Web:

Índice de Aplicaciones Oracle

¿Quieres saber quién es el fundador de Oracle? Su nombre es Larry Ellison y para conocer su bibliografía puedes visitar esta página Web:

Bibliografía de Larry Ellison **[Versión en caché]**

Aplicaciones con conexión a base de datos

Características de Oracle Express

Hemos descargado, instalado y probado Oracle Express, pero para aprovechar sus posibilidades deberíamos conocer algunas cuestiones de su funcionamiento interno. Por ejemplo, hemos usado el usuario SYSTEM, pero ¿cómo gestiona la seguridad?, o ¿cómo es la arquitectura de Oracle? Para responder a estas preguntas vamos a repasar algunas de las características de este SGBD.

Cuando trabajamos con el SGBD Oracle se define una **base de datos** que contiene al conjunto de datos y a las estructuras que permiten su acceso y manipulación. Además, a cada base de datos, el SGBD asocia una **instancia de base de datos**, que está conformada por varios procesos ejecutados en segundo plano y un conjunto de estructuras de memoria compartida que son necesarios para gestionar sus estructuras de archivos y acceder a la información contenida en la base de datos. Una base de datos Oracle está almacenada físicamente en ficheros del sistema operativo, y la correspondencia entre los ficheros y las tablas es posible gracias a estas estructuras internas de la BD.

Así, la **arquitectura general de Oracle** incluye para las instancias de bases de datos, varios procesos ejecutándose, más los espacios de memoria dedicados a ejecutar los procesos específicos o al almacenaje de información de cada proceso. Una instancia de Oracle se asigna siempre a una única base de datos, pero existen ocasiones en las que una base de datos Oracle puede tener varias instancias. La instancia se compone del Área Global del Sistema (SGA) y el Área Global de Programas (PGA).

¿Y cómo se almacena la base de datos? Para entenderlo debemos saber que **una base de datos de Oracle tiene una capa lógica y otra física**, veamos cómo son:

1. **Capa física**: Una base de datos Oracle se compone de varios archivos físicos que residen en el disco:
   a. **Datafiles**: almacenan toda la información de la base de datos (tablas, índices, etc.) que puede tener muchos datafiles.
   b. **Redo log**: contienen los registros que permiten efectuar operaciones de reconstrucción de la base de datos para deshacer los cambios efectuados, sirven para gestionar las transacciones.
   c. **Archivos de Control**: incluyen la información necesaria para asociar la base de datos a la instancia como el nombre de la BD, la localización de los *datafiles* y los *redolog*, etc.
2. **Capa lógica**: debemos distinguir entre dos elementos importantes:
   a. **Una base de datos se encuentra dividida en una o más piezas lógicas llamadas tablespaces**, que son utilizados para organizar y agrupar la información y así, simplificar la administración de los datos. Un **tablespace** es una unidad lógica de almacenamiento que está constituido por uno o más datafiles. Como cada BD debe tener su *tablespace* por lo que existe el *tablespace* principal, de nombre SYSTEM con su correspondiente *datafile* físico. Para desarrollar una nueva base de datos deberemos tener un nuevo *tablespace* y deberemos crear nuevos usuarios que asociaremos a los *tablespaces*.

   b. Oracle organiza la base de datos para los distintos usuarios a través del desarrollo de un **esquema de la base de datos** (***schema***) **o conjunto de objetos de un usuario**, que consiste en una colección de objetos que conforman el modelo lógico relacional que se desea implementar y que pertenecen al usuario del esquema. El esquema contendrá **objetos** como: **tablas, índices, vistas, procedimientos PL/SQL, Triggers, secuencias o sinónimos**.

¿Y en cuanto a la seguridad? ¿Qué posibilidades ofrece Oracle?

Veamos, en Oracle existen varios niveles de seguridad que son almacenados en tablas del diccionario de datos (propiedad de SYSTEM):

1. **Seguridad de cuentas:** Para acceder a los datos de una base de datos Oracle debo tener una cuenta de usuario.
2. **Seguridad en el acceso a los objetos de la base de datos**: se realiza según un sistema de privilegios que establece que determinados comandos sean permitidos o no en función del usuario que los lanza. Los **privilegios** (acceso, selección, modificación, borrado, etc.) se pueden agrupar formando **roles** para simplificar su administración cuando tenemos muchos usuarios (**SYSTEM** tiene asignado el rol **DBA**).
3. **Seguridad a nivel de sistema**: permite la gestión de privilegios globales.

Vamos a ver todo esto que hemos comentado en el propio Oracle, veremos algunas posibilidades del administrador del sistema, comprobaremos la arquitectura del SGBD y la seguridad con las cuentas de usuario. Para ello accede a esta animaciónsobre Administración de Oracle Express.

Administración y monitorización de la arquitectura de Oracle Express 10g

Autoevaluación

Señala la respuesta correcta:

a) Un **tablespace** pertenece a la capa lógica.
b) Un **datafile** pertenece a la capa lógica
c) Un **esquema** pertenece a la capa física
d) Un fichero **redo log** pertenece a la capa lógica

Comprobar

PARA SABER MÁS

Oracle pone a nuestra disposición múltiple documentación sobre este SGBD en su web, te recomendamos que lo visites y consultes frecuentemente durante el estudio de esta unidad:

Documentación sobre Oracle Express

En el siguiente enlace tienes un tutorial sobre Oracle Express que aunque está escrito en inglés se sigue fácilmente por las múltiples animaciones que tiene, además, lo puedes descargar a tu ordenador.

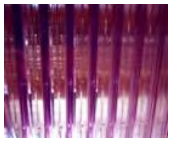
Tutorial sobre Oracle Express

Aquí tienes un tutorial sobre la administración de Oracle en castellano perteneciente a la web de la red científica.

Tutorial de administración de Oracle

Modelo Relacional para un ejemplo de BD

Hemos visto pequeños ejemplos prácticos de uso de la base de datos, pero te preguntarás que cuándo vamos a enfrentarnos a problemas reales, bien pues ha llegado el momento de desarrollar una base de datos que de respuesta a una situación real.



En el caso práctico hemos planteado una situación de la realidad que queremos informatizar:

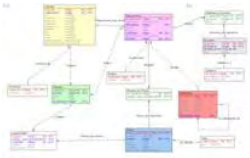
La Gestión de un Taller Informático.

En este apartado vamos a describir con un poco de más detalle el problema y mostraremos la solución que se ha plateado, para lo cual ha sido necesario crear un modelo lógico que respondiese a las necesidades.

Necesitamos informatizar el servicio de mantenimiento de una empresa de informática. Analizando el problema vemos que tenemos que tener en cuenta estas cuestiones:

- La empresa va a tener diferentes **empleados** y debemos tener en cuenta que habrá jefes. De cada empleado deberemos guardar sus datos personales y profesionales.
- La empresa va a tener un **modelo de negocio** basado en vender material informático y en realizar el mantenimiento y reparación de equipos informáticos, por lo que deberá facturar tanto los materiales como las reparaciones que incluirán la mano de obra y las posibles piezas sustituidas.
- Es necesario guardar todos los datos de los **clientes** de la empresa.
- La **facturación** exige que almacenemos todos los datos necesarios para una factura legal y para realizar el control dentro de la empresa. Por tanto se necesitan los datos generales de la factura y el desglose detallado de cada una de las piezas o reparaciones realizadas.
- Hay que especificar el estado en el que se encuentra la **reparación** para informar al cliente y los plazos de entrega.
- Hay diferentes tipos de **acciones** incluidas en las posibles reparaciones que estarán codificadas y sistematizadas.
- Las materiales estarán organizados por los tipos de **piezas** a utilizar.

En la imagen tienes un logo para la aplicación de **Gestión del Taller Informático** que vamos a llamar **El PC Feliz**. Recuerda el proceso que debemos seguir, primero analizaremos el problema y después realizaremos el diseño de la aplicación. Dentro de proceso deberemos diseñar la base de datos en la que almacenaremos toda la información necesaria. Como vimos en el primer apartado este diseño lo realizaremos mediante un diagrama Entidad-Relación. A partir de dicho diagrama se ha elaborado el modelo relacional, a continuación incluimos una imagen del resultado. Haz clic sobre ella para verla a tamaño de pantalla completa.



Vamos a fijarnos en algunos aspectos del modelo para entenderlo mejor:

- La **facturación** recoge la posibilidad de que en una **factura** podamos vender varias **piezas**. Esta actividad comprendería las tablas de: **Facturas**, **Piezas**, la tabla **LineasDetalle** necesaria para crear la relación de muchos-a-muchos y la tabla **Clientes**.
- Las **reparaciones** se llevan a cabo para un **empleado** y pueden acarrear el uso de varias **piezas**. Cada **reparación** puede requerir una serie de **acciones** (por ejemplo: formatear un disco, sustituir una pieza, etc). La reparación está a cargo de un **empleado**.
- Las **acciones** están categorizadas con diferentes **precios de hora** de trabajo, eso se recoge en la tabla **categoría**
- El precio de las **piezas** puede variar de una **factura** a otra, por ello se almacena en la tabla un **recargo**.
- En la tabla **Empleados** tenemos una relación reflexiva para guardar el jefe de un empleado, que a su vez será otro empleado de la empresa.
- En la **Facturación** de reparaciones el modelo comtempla que una reparación puede generar una factura, en cuyo caso, la factura contendrá las piezas que haya necesitado esa reparación. En la tabla **LineasDetalle** incluiremos los elementos facturados, que en las reparaciones serán las **piezas** y las **acciones** realizadas que podrán contener un importe (mano de obra de esa reparación, por ejemplo), siendo en ese caso el campo **numero-serie** nulo.
- En el diagrama se recoge la cardinalidad de las relaciones. Así por ejemplo, un **cliente** puede generar muchas **facturas**, o muchos **clientes** podrán pertenecer a la misma **provincia**.

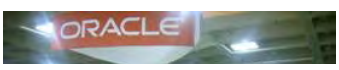
A partir del diagrama Entidad-Relación hemos obtenido un modelo relacional, con el cual hemos desarrollado las tablas que deberán estar normalizadas al menos como **tercera forma normal (3FN)**. Este proceso lo hemos realizado con la ayuda de una herramienta CASE como **TOAD Data Modeler** (Descrita y utilizada en el Módulo Profesional de "***Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE***"). Con esta utilidad hemos generado el siguiente documento con la descripción del modelo lógico de datos que reproduciremos en la base de datos (BD). Aquí se describen las relaciones, las tablas, las claves primarias, los atributos, etc. **JDeveloper** también incluye una herramienta CASE que permite realizar este proceso, incluye una herramienta para realizar el diseño de nuestra aplicación con diagramas UML y el diseño de nuestra Base de Datos con un Diagrama Relacional. A continuación tienes la descripción de las tablas correspondientes al modelo relacional.

Modelo lógico para el proyecto

> ***PARA SABER MÁS***
>
> ***Si tienes dificultades para comprender el modelo de datos anterior debes repasar los contenidos del Módulo Profesional de "Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE". Además, te incluimos aquí un enlace donde puedes profundizar en estos diagramas:***
>
> ***Modelado de datos con diagramas Entidad-Relación*** **[Versión en caché]**

Creación de las tablas de la BD de ejemplo

Ya tenemos un SGBD instalado en nuestro ordenador, y tenemos el modelo lógico de datos que responde a las necesidades de nuestro problema... ¿Qué debemos hacer ahora?



En efecto, ahora vamos a crear ese modelo lógico de tablas en Oracle Express. ¿Y cómo lo hacemos?

En este apartado vamos a ver cómo hacerlo utilizando los menús disponibles a través de la interfaz Web, que como verás es muy intuitiva.

¿Y por dónde empezamos? De lo que conocemos de la arquitectura de Oracle sabemos que debemos crear una nueva base de datos con un nuevo esquema asociado a un nuevo usuario. No debemos usar **SYSTEM** porque no es seguro recurrir a un usuario con demasiados privilegios en el sistema. Como ya vimos, existen diferentes **roles creados por el sistema**:

- **CONNECT**: necesario para acceder a la BD.
- **RESOURCE**: permite crear objetos.
- **DBA**: da privilegios de administrador (el usuario SYSTEM es DBA).

Por tanto, empezaremos creando un usuario con estos roles y luego crearemos las tablas. Es bueno crear un usuario administrador del sistema distinto de **SYSTEM** para poder realizar operaciones de administración (crear usuarios, por ejemplo) sin riesgos de cometer errores graves, ya que **SYSTEM** tiene acceso a objetos de Oracle que conviene no modificar sin control. Podemos ver el proceso en la siguiente animación sobre el desarrollo de una Base de Datos para el Taller Informático, en la que vamos a crear dos tablas con sus claves primarias, las relacionaremos entre sí con claves ajenas, añadiremos algunas filas, comprobaremos cómo funcionan las restricciones de clave ajena (integridad de referencia), haremos un ejemplo de restricción semántica, y alguna cosa más:

Creando una base de datos en Oracle Express 10g

Hemos visto cómo crear las tablas y los usuarios con la interfaz gráfica de Oracle Express, pero como sabes podemos utilizar comandos SQL directamente. ¿Recuerdas los comandos de creación de tablas y usuarios?

Veamos, **la definición de bases de datos y la creación de tablas corresponde al DDL**, y en concreto en SQL los comandos usados para **crear, modificar y borrar bases de datos** son respectivamente:

```
CREATE DATABASE <nombre_bd> ...

ALTER DATABASE <nombre_bd> ...

DROP DATABASE <nombre_bd>
```

Para agilizar el acceso a las tablas los SGBD utilizan índices, en Oracle al crear las tablas con las claves primarias se crean automáticamente estos índices, pero podemos **crear, modificar o borrar índices** con las sentencias:

```
CREATE INDEX <nombre_índice> ON <tabla>(<columna1>,<columna2>,...)

ALTER INDEX <nombre_índice> ...

DROP INDEX <nombre_índice>
```

Como recordarás, a veces podía interesar crear una **vista** como complemento a las tablas. No olvides que las vistas no son tablas reales, pero las vemos como tales. Para crear tablas debemos tener privilegios, al crear la cuenta podemos marcar la casilla: **CREATE VIEW**. La sintaxis en SQL para crear una vista es:

```
CREATE VIEW <nombre_vista> (<columnas>)  AS <sentencia_select>
```

Con respecto a la creación de usuarios, debemos recurrir al **DCL**. Así, en SQL podemos utilizar estas sentencias para **crear, modificar y eliminar usuarios** respectivamente:

```
CREATE USER <nombre_user> IDENTIFIED BY <contraseña>

ALTER USER <nombre_user> ...

DROP USER <nombre_user> [CASCADE]
```

*(Con **cascade** obligamos a eliminar todos los objetos del esquema antes de borrar al usuario).*

Durante esta unidad recordaremos la sintaxis de algunos comandos de SQL, pero en cualquier caso, te insistimos en que si tienes dudas sobre ellos debes recuperar los enlaces propuestos y los contenidos del Módulo Profesional de "***Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE***".

Con este **ejemplo del Taller Informático** hemos visto parte del proceso que explicamos en el primer apartado de esta unidad, hemos aplicado los conceptos fundamentales de las bases de datos en general y del SGBD Oracle Express en particular. De esta manera, ha cobrado más sentido nuestra aproximación a los objetos de las bases de datos, sus relaciones, restricciones, claves, integridad, modelo E/R, SQL, diccionario de datos, niveles físico y lógico, etc.

Autoevaluación

¿Cómo se accede al sistema de gestión y administración de Oracle Express?

a) Mediante un servidor Web.
b) Mediante un cliente de línea de comandos.
c) Mediante un navegador Web.
d) Las respuestas b) y c) son correctas.

Comprobar

PARA SABER MÁS

Hemos explicado que todo el proceso de modelización del problema lo hemos realizado con la herramienta CASE Toad Data Modeler. Si quieres aprender más sobre la herramienta puedes ver su página Web:

Página de CASE Toad Data Model

Es muy importante que conozcas bien SQL, además debes conocer la versión particular que utiliza Oracle, por ello te recomendamos que consultes su manual de referencia oficial:

Manual de referencia de SQL para Bases de Datos Oracle

Un Script SQL para crear la BD del ejemplo

Después del apartado anterior podrás pensar que diseñar el resto del modelo del Taller Informático va a ser complicado y laborioso por el número de tablas, columnas, relaciones y restricciones que tiene. Por otro lado, si recuerdas el proceso explicado en el primer apartado de esta unidad, falta por hacer un paso.

¿Cuál?

Exacto, la herramienta CASE no sólo nos ha proporcionado el modelo relacional y las tablas, sino que también nos ha generado un archivo **SCRIPT con todos los comandos SQL** necesarios para crear nuestra base de datos. ¿Vemos cómo usar este script SQL en Oracle Express?

A continuación incluimos el archivo que contiene el script de comandos SQL para generar el modelo de datos para la **Gestión del Taller Informático**.

Podemos decir que tiene tres partes:

- **Creación de tablas**: fíjate en cómo se crean las tablas con todos sus atributos, se definen las claves primarias y claves ajenas.
- **Creación de las secuencias**: se usan para los campos numéricos que se incrementan automáticamente
- **Definición de los disparadores** (**triggers**): con ellos se establecen restricciones y controles sobre las tablas.

Además, al comienzo hay una cabecera con información sobre su creación, observa que los **comentarios** se indican con **/* */** para varias líneas, o con **--** para una sola.

En la primera parte del script, dedicado al Desarrollo de una Base de Datos para el Taller Informático, se utilizan órdenes **DDL** para crear la base de datos del Taller Informático. Como bien sabes, en este caso hemos utilizado comandos DDL de SQL. A lo largo de la unidad estamos comentando la sintaxis de algunos de estos comandos. Recordemos de lo explicado sobre la arquitectura en Oracle, que al crear el usuario se definen las estructuras lógicas y físicas de la base de datos, dejando preparada la base de datos sobre la que vamos a trabajar. De esta manera, para el usuario creado los diferentes objetos de la base de datos (tablas, índices, etc) se agrupan en un **esquema de base de datos** que queda asociado al usuario. Por tanto, el script puede comenzar creando las tablas. Te incluimos aquí el script para que lo veas, de todas formas, en el próximo apartado lo explicaremos con más detalle.

Descarga el script SQL para crear las tablas

Autoevaluación

En el script hemos incluido comandos SQL para crear tablas, estos comandos pertenecen al...

a) DDL
b) DCL
c) DML
d) Ninguna respuesta es correcta

Comprobar

Ejecución de un Script SQL en Oracle Express

¿Has visto el script? ¿Lo has comprendido bien? Es básico que entiendas su funcionamiento, para facilitarte la tarea a continuación indicaremos la finalidad de las sentencias SQL incluidas en el mismo.

Si revisas el archivo del script, comprobarás que comenzó con la creación de las tablas y después hizo la definición de las **secuencias,** tras lo cual, vemos cómo fueron creados los **disparadores.** Un **trigger o disparador** es un bloque PL/SQL asociado a una tabla, vista, esquema o base de datos. Los triggers se utilizan para asegurar la integridad de datos, revisando datos de forma consistente. Un trigger se ejecuta implícitamente siempre que un evento específico tiene lugar. La **sintaxis** para crear un trigger en PL/SQL es la siguiente:

```
CREATE [OR REPLACE] TRIGGER trigger_name
BEFORE / AFTER
[INSERT / UPDATE / DELETE OR ...]
ON object_name
[ [REFERENCING OLD AS old/NEW AS new]
[FOR EACH ROW]
[WHEN (condition)]]
trigger_body
```

Como vemos, se crea el **trigger** dándole un **nombre (trigger_name),** se ejecuta antes o después **(before/after)** de que se produzca una acción de **inserción, actualización o borrado (insert, update o delete)**, sobre el **objeto (object)** indicado **(tabla, vista, etc)**. Si queremos, opcionalmente podemos elegir que la acción definida en el **cuerpo del trigger (trigger_body)** se puede ejecutar una vez o para **todas las filas de la tabla (for each row)**, o añadir una condición **condición (condition)** para que se ejecute sólo si se cumple. Cuando hacemos una operación de actualización tenemos un valor "antiguo" del registro (**old**) y el nuevo valor que va a tomar (**new**), podemos usar un alias para old y new con "**referencing**".

El **cuerpo del trigger** se define utilizando PL/SQL,que es una extensión o ampliación de SQL, para ofrecer un entorno de programación. Los bloques de **PL/SQL** se almacenan en la base de datos y tienen esta estructura:

```
[DECLARE
```

```
Declaración de variables, constantes, excepciones...]
BEGIN
<órdenes SQL>
<órdenes PL/SQL>
[EXCEPTION
acciones a realizar al producirse una excepción o error]
END;
/
```

El resultado de una consulta podemos almacenarlo en una varable si es un dato simple, si queremos almacenar una tabla obtenida con un **SELECT** podemos usar los Cursores **de PL/SQL,** todos se declaran en la zona **DECLARE**

¿Has observado el símbolo final "/"? ¿Recuerdas para qué se pone?

Es necesario finalizar el código con el **carácter "/"** para que se ejecute. También, si te has fijado, en los scripts SQL las instrucciones y comandos pueden ocupar más de una línea terminando con el **carácter ";"** que además indica un punto de ejecución.

A continuación te incluimos una nueva animación, sobre la Ejecución del Script SQL en Oracle Express, en la que te vamos a explicar más detalladamente el script, y te guiaremos para que puedas cargarlo y ejecutarlo**.** Para ello, crearemos un nuevo usuario que utilizaremos a lo largo de la unidad para el ejemplo del Taller Informático.

Script de comandos SQL para crear una base de datos para la gestión de un taller de informática

Bueno, hemos visto todo el proceso: desde el diseño de la base de datos hasta la creación de las tablas en Oracle. Antes comentamos algunas posibilidades de **JDeveloper** para diseñar bases de datos; ¿podríamos haber hecho todo este proceso con JDeveloper?

Lo cierto es que sí, JDeveloper nos permite diseñar la base de datos y obtener a partir del diagrama relacional un script SQL que podemos ejecutar en Oracle Express para crear las tablas. En la siguiente animación sobre el Diseño de una Base de Datos con Jdeveloper, puedes ver una demostración sobre cómo hacerlo:

Diseño de bases de datos con JDeveloper

PARA SABER MÁS

Se ha estado utilizando PL/SQL, has visto la gran potencia que tiene, es importante conocerlo bien para aprovechar las posibilidades que nos ofrece el SGBD y facilitar el trabajo posterior de programación de la aplicación que utilice la base de datos. En este enlace tienes una introducción a PL/SQL:

Introducción a PL/SQL [Versión en caché]

PL/SQL es un completo lenguaje, puedes aprender más sobre él en este completo tutorial que te recomendamos que consultes:

Tutorial de PL/SQL

Aplicaciones con conexión a base de datos

Manipulación de datos con SQL

Gracias al proceso de diseño seguido y con la ayuda de la herramienta CASE que generó el script anterior, ya tenemos nuestra base de datos en Oracle Express. Pero no contiene ningún dato, lo que deberemos hacer ahora es introducir nuevos datos. Ya vimos en el apartado "**Creación de las tablas de la BD de ejemplo**" de esta unidad, cómo se pueden introducir los datos desde la interfaz Web de Oracle Express, también podemos hacerlo con sentencias **DML** SQL. ¿Lo recuerdas? Démosles un pequeño repaso.

Inserción de filas en una tabla

¿Cómo añadimos datos a las tablas que hemos creado? Para ello usamos el comando INSERT, cuya sintaxis es la siguiente:

```
INSERT [INTO] <tabla> [(columnas...)] VALUES (lista valores,...)
```

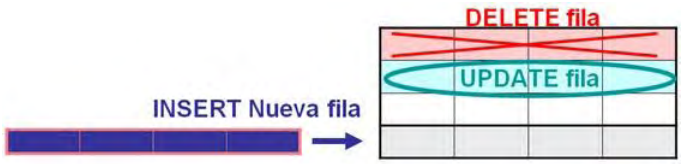
La orden **inserta una fila en la tabla** indicada <tabla> para las columnas indicadas con los valores incluidos en la lista. El número de columnas y de valores debe ser igual y escrito en el mismo orden. Cuando nos saltamos alguna columna el sistema asigna el valor **NULL** a ese campo o el valor por defecto si estaba definido. Si vamos a insertar una fila completa podemos omitir los nombres de las columnas. La lista de valores puede ser sustituida por expresiones SQL que extraigan las filas de otra tabla. Algunos ejemplos pueden ser:

Inserta fila en la tabla **CATEGORIA:**

```
INSERT INTO CATEGORIA (IDCATEGORIA, NBCATEGORIA, PRECIOHORA) VALUES ('', 'Analista', '50');
```

Inserta fila en la tabla **PIEZAS** (no hace falta indicar el nombre de las columnas):

```
INSERT INTO PIEZAS VALUES ('HD100', 'Lacie 250 GB', '2', '99', '20', '3', 'Disco duro multimedia');
```



Actualización de las filas de una tabla

A veces ocurre que debemos realizar cambios en los datos almacenados porque se han actualizado o porque eran erróneos, en este caso debemos usar el comando UPDATE.

¿Y su sintaxis? Aquí está:

```
UPDATE <tabla>
  SET <columna_1>=<nuevo valor o expresión_1>...
 [WHERE <condición>]
```

Actualizaremos las columnas de la tabla indicada con los valores incluidos en las expresiones, y esta modificación la haremos para aquellas filas que cumplan la condición. Si no especificamos la condición, el cambio afectará a todas las filas de la tabla. En los nuevos valores podemos incluir datos o expresiones. Veamos un par de ejemplos:

Actualiza la tabla **piezas**, aumentando el **recargo** aplicado a las **piezas** cuyo **codtipo** es 4:

```
UPDATE piezas SET recargo=recargo*1.05 WHERE codtipo='4'
```

Cambia la **provincia** a Albacete en la tabla **provincias** para la fila cuyo valor de **codprovincia** es 23:

```
UPDATE provincias SET provincia='Albacete' WHERE codprovincia='23'
```

Borrado de filas de una tabla

Al igual que tenemos la posibilidad de añadir y modificar los datos de una tabla, lógicamente también será necesario **eliminar los datos** que ya no interesen. Para ello usaremos el comando DELETE con la sintaxis siguiente:

```
DELETE FROM <tabla> [WHERE <condición>]
```

Con **DELETE** se borran las filas de la tabla que cumplen la condición. Si se omite esta condición se borrarían todas las filas de la tabla. Pongamos un ejemplo para borrar de la tabla **categoria** las filas que tengan valor **21** en el campo **idcategoria**:

```
DELETE FROM categoria WHERE idcategoria='21'
```

¿Podemos hacer cambios en las filas sin preocuparnos de nada? Debemos tener en cuenta que los cambios en las filas deben **cumplir las reglas de integridad** marcadas en la definición de las tablas, además pueden provocar que se ejecute algún trigger como los creados en nuestro script.

Ahora deberíamos cargar nuestra base de datos con suficientes datos como para poder trabajar con las tablas, pero estarás pensando que va a ser un trabajo largo y tedioso,... Tranquilo, tenemos la solución. Hemos creado un script SQL con los comandos DML necesarios para introducir la información en la base de datos. Ya sabes usarlo, ¿no? Simplemente, debes entrar en Oracle Express con el usuario **usuario_taller** que creamos anteriormente, cargar el **archivo script SQL** para introducir datos que incluimos a continuación y ejecutarlo tal y como explicamos en el apartado anterior.

Descarga el script

Por si tienes dudas, en esta animación puedes ver una explicación del script SQL con sentencias DML, su uso y resultado de ejecución:

Script de comandos MDL SQL para introducir datos en la BD del taller informático

Autoevaluación

¿Cuáles de los siguientes comandos SQL son DML?

a) UPDATE.
b) DELETE.
c) INSERT.
d) Todas las respuestas anteriores son correctas .

Comprobar

*CASO. Evidentemente **Víctor** perdió la apuesta e invitó a **Carmen**. Así, nuestros informáticos ya han creado la base de datos y también han incorporado los datos de prueba que usarán durante el desarrollo de la aplicación mediante otro script SQL. **Víctor** ha estado trabajando duro para recordar sus conocimientos de SQL, es consciente de que sin ellos no podrá llevar a cabo la programación de la aplicación. **Víctor** sabe que SQL proporciona las sentencias necesarias para realizar operaciones DDL, DML y DCL y comienza a realizar prácticas sobre la base de datos del taller informático. **Carmen** le explica que no debe olvidar el tema de la seguridad y control de usuarios así como del control de transacciones. En la aplicación de gestión del taller informático podrán trabajar distintos empleados y el sistema debe impedir que haya problemas de acceso*

*concurrente a los mismos datos. **Carmen** explica a **Víctor** que Oracle Express proporciona técnicas y mecanismos muy potentes para la gestión de usuarios y la gestión de concurrencias.*

Recuperación de información de la base de datos con SQL

Esto va marchando, ¿verdad? Ya tenemos nuestra base de datos creada y cargada de datos, hemos visto como ver los datos almacenados desde el interface web de Oracle Express, pero no olvidemos que la finalidad de esta unidad es ver cómo podemos acceder a la base de datos desde un entorno de programación como JDeveloper, para lo cual tendremos que manejar comandos SQL. Seguro que recuerdas como se manejaban, pero vamos a realizar aquí un pequeño repaso porque es básico que manejes bien todos estos comandos.

Las **consultas en SQL** se realizan con la sentencia SELECT cuya sintaxis básica es:

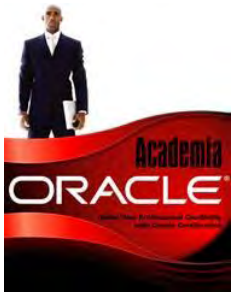
```
SELECT [DISTINCT] <lista_de_expresiones>

FROM <tabla>

WHERE / ORDER BY (ASC/DESC) <criterios>]
```

En **lista de expresiones** especificaremos las condiciones que deben cumplir las filas que buscamos de la tabla indicada. Si queremos obtener todas las columnas de la tabla podemos utilizar el carácter **"*"** en la <**lista_de_expresiones**>. También podemos usar **operaciones de cálculo** en las expresiones**. DISTINCT** permite eliminar las filas resultantes iguales. **ORDER BY** permite ordenar las filas de forma ascendente (**ASC**) o descendente (**DESC**). Las cláusulas finales de la sentencia **SELECT** permiten imponer criterios y restricciones para que se muestren sólo las filas que cumplen estas **condiciones** o respetando determinados criterios como el orden. Esto hace a SELECT más potente y flexible. Para comprender su funcionamiento vamos a poner algunos **ejemplos** que puedes probar entrando en Oracle Express con el usuario creado y haciendo uso de su **Editor de Comandos SQL** tal y como se ha explicado anteriormente:

- Mostrar todas las filas de la tabla Clientes con todos los campos:
  SELECT * FROM clientes
- Mostrar todas las filas de la tabla Clientes con los campos codprovincia y nombre solamente:
  SELECT CODPROVINCIA, NOMBRE FROM CLIENTES
- Mostrar el nombre, descripción y coste real (coste más el porcentaje de recargo) de la tabla piezas:
  SELECT NOMBRE, DESCRIPCION, COSTO+(COSTO* RECARGO/100) FROM PIEZAS
- Mostrar todas las columnas de la tabla Clientes de los clientes de la provincia 22:
  SELECT * FROM clientes where codprovincia= 22
- Mostrar todas las poblaciones distintas de donde provienen nuestros clientes:
  SELECT DISTINCT poblacion FROM clientes
- Mostrar los datos de todas las piezas ordenadas descendentemente por recargo y ascendente el costo:
  SELECT * FROM PIEZAS ORDER BY RECARGO DESC, COSTO ASC

SQL ofrece mucha flexibilidad para establecer las condiciones, permitiendo usar diferentes **operadores** (relaciones, lógicos, etc). Las **expresiones** pueden agruparse con paréntesis para indicar el orden de evaluación o aclarar las expresiones complicadas. Veamos algunas de estas posibilidades mediante ejemplos:

- **Operadores de comparación: <, <=, =, >=, >, <>.**
  SELECT NOMBRE, DESCRIPCION, COSTO FROM PIEZAS WHERE COSTO >=100
- **Operadores lógicos AND, OR, NOT** (y, o, negación). Buscamos las piezas con coste superior a 200 y que quedan 2 ó menos unidades:
  SELECT nombre, descripcion FROM PIEZAS WHERE COSTO > 200 AND UNIDADES <= 2
  Seleccionamos los clientes que no son de Almería capital:
  SELECT * FROM clientes where not poblacion= 'Almería'
- **Operador de pertenencia a un conjunto IN**: Seleccionamos las filas que tienen valores dentro del conjunto para la expresión indicada. Por ejemplo, clientes de "Aguadulce" o "Almería":
  SELECT * FROM clientes where poblacion IN ('Almería', 'Aguadulce')
- **Operador de rango BETWEEN... AND**: Establece una condición entre un rango de valores DESDE...HASTA... Por ejemplo, busquemos las acciones llevadas a cabo en el mes de enero y febrero de 2007 en las distintas reparaciones:
  SELECT * FROM reparacion_acciones WHERE fecha BETWEEN '01/01/2007' AND '28/02/2007'
- **Operador de correspondencia con patrón LIKE**: La selección se hace sobre una condición construida con un patrón donde puedo usar el comodín **"%"** para sustituir a cualquier cadena de caracteres, o el comodín **"_"** para sustituir un carácter en determinada posición. Por ejemplo, seleccionemos los clientes con un código postal de Almería (04 al comienzo):
  SELECT * FROM clientes where codpostal LIKE '04%'
- **Condición de valor nulo IS NULL / IS NOT NULL**: Por ejemplo, empleados que no tienen jefe asignado:
  SELECT * FROM empleados WHERE idjefe IS NULL



Podemos recuperar información de nuestra propia base de datos accediendo al diccionario de datos, veamos algunos ejemplos:

- Mostrar las tablas creadas:

```
SELECT * FROM TAB
```

- Mostrar las columnas de la tabla clientes:

```
SELECT * FROM COL WHERE TNAME = 'CLIENTES'
```

Autoevaluación

¿Cómo consulto todos los datos de las piezas cuyo precio con recargo es mayor de 100?

a) SELECT COSTO FROM PIEZAS WHERE COSTO+(COSTO* RECARGO/100)>100
b) SELECT * FROM PIEZAS WHERE COSTO+RECARGO>100
c) SELECT * FROM PIEZAS WHERE COSTO+(COSTO* RECARGO)>100
d) SELECT * FROM PIEZAS WHERE COSTO+(COSTO* RECARGO/100)>100

Comprobar

Agrupaciones y subconsultas con SQL

Hemos visto un uso básico de la sentencia SELECT pero podemos pensar en consultas que aún no hemos practicado. ¿Imaginas alguna? Por ejemplo obtener **resúmenes o totales** a partir de los datos almacenados, o hacer **subconsultas** dentro de otras consultas, ¿recuerdas como se hacían?

Vamos a recordar en este apartado las agrupaciones, que en SQL se resuelven utilizando **GROUP BY / HAVING.** ¿Y si queremos saber el precio medio de las piezas? Con GROUP podemos hacer agrupaciones o resúmenes de filas, pudiendo aplicar funciones de cálculo. En esta tabla las vemos con ejemplos:

Función	Explicación	Ejemplo	SQL
SUM	Calcula la suma	Total piezas en almacén	`SUM SELECT SUM(unidades) FROM piezas`
SUM AVG	Calcula la media	Precio medio de piezas	`SELECT AVG(costo) FROM piezas`
SUM MIN	El mínimo	Pieza más barata	`SELECT MIN(costo) FROM piezas`
SUM MAX	El máximo	Pieza más cara	`SUM SELECT MAX(costo) FROM piezas`
SUM COUNT	Cuenta las filas	Número de empleados	`SUM SELECT COUNT (*) FROM empleados`
SUM COUNT	Cuenta valores no nulos	Número de empleados del jefe 1	`SELECT COUNT (idjefe) FROM empleados WHERE idjefe='1'`
COUNT	Cuenta valores distintos	Distintas provincias de los clientes	`SELECT COUNT(DISTINCT codprovincia)FROM clientes`

- Veamos cómo agrupar estas funciones con **GROUP BY** de manera que podamos calcular subtotales. Por ejemplo, quiero saber el número de clientes agrupados por población ordenados de más a menos:

```
SELECT POBLACION, COUNT(IDCLIENTE) FROM CLIENTES GROUP BY POBLACION ORDER BY COUNT(IDCLIENTE) DESC
```

- Al igual que utilizamos la cláusula WHERE para restringir los resultados del FROM, podemos hacerlo lo mismo para **GROUP BY usando HAVING**. ¿Cómo limitaría la información anterior para obtener datos sólo de las poblaciones con 2 clientes como mínimo que tienen de codprovincia 22 (Almería)?

```
SELECT POBLACION, COUNT(IDCLIENTE) FROM CLIENTES WHERE CODPROVINCIA='22'
GROUP BY POBLACION HAVING COUNT(IDCLIENTE)>='2' ORDER BY COUNT(IDCLIENTE) DESC
```

Hay ocasiones en que nos interesa seleccionar filas del resultado anterior de una consulta previa. Esto es posible hacerlo en SQL por medio de las **subconsultas.** Podemos definir una subconsulta como una consulta que aparece dentro de la cláusula WHERE o **HAVING** de otra sentencia SQL. Veamos algunos ejemplos:

- Queremos saber qué piezas tienen un recargo superior a la media. Para ello primero calcularemos la media y luego haremos la consulta de los más recargados:

```
SELECT * FROM PIEZAS WHERE RECARGO > (SELECT AVG(RECARGO) FROM PIEZAS)
```

- En caso de que necesitemos conocer los tipos de piezas que tienen un stock (número de unidades almacenadas) mayor de la media sin tener en cuenta las piezas software (codtipo 6), deberíamos recurrir a las subconsultas, que pueden usarse para limitar las filas sobre las que operará la cláusula HAVING, veamos la sentencia SQL:

```
SELECT CODTIPO, SUM(UNIDADES) FROM PIEZAS WHERE CODTIPO<>'6'
GROUP BY CODTIPO HAVING SUM (UNIDADES) > (SELECT AVG(UNIDADES) FROM PIEZAS WHERE CODTIPO
```

Como has visto en estos ejemplos, hemos usado operadores relacionales (<, <=, =, >=, >, <>), para combinar las subconsultas **(>)**, pero pueden usarse otros operadores que hemos visto (como IN). Veamos ejemplos:

- Podríamos usar cualquier operador relacional si nos interesa conocer los clientes que están en la misma provincia que la empresa INFOSUR:

```
SELECT * FROM CLIENTES WHERE CODPROVINCIA = (SELECT CODPROVINCIA FROM CLIENTES WHERE NOMBRE='INFOSUR')
```

- Para saber las facturas (codfactura) que han incluido piezas software:

```
SELECT codfactura from lineasdetalle where numero_serie IN (select numero_serie from piezas where codtipo ='6')
```

Autoevaluación

¿Cómo consulto el número medio de unidades de piezas almacenadas en el almacén (stock)?

a) SELECT AVG(unidades) FROM piezas.
b) SELECT AVG(unidades) FROM piezas WHERE unidades=AVG(unidades).
c) SELECT MEDIA(unidades) FROM piezas.
d) Ninguna respuesta anterior es correcta.

Comprobar

Consultas a varias tablas con SQL

Hasta ahora hemos visto **sentencias y cláusulas** de SQL para obtener información de una sola tabla, pero en nuestro ejemplo del Taller tenemos muchas tablas y para recuperar cierta información debemos recurrir a realizar consultas sobre varias tablas a la vez. Por ejemplo, si quiero saber datos sobre los clientes que han trabajado para un empleado concreto deberé consultar las tablas de empleados y clientes. ¿Cómo se puede hacer esto? Para ello hay que realizar **consultas multitabla.** Veamos sus tipos y posibilidades a continuación.

Cuando enlazamos dos tablas distintas hablamos de **Composición de tablas o JOIN**, que podemos describir como la combinación de las filas de una tabla con las de otras, concatenándose para generar una nueva tabla. Esta operación corresponde con el concepto matemático de producto cartesiano. Aunque lo habitual es utilizar un JOIN entre dos tablas, podemos hacerlo sobre más. **La combinación entre una tabla y otra se realza con la clave ajena de una que es clave primaria de la otra**. Pero si tienen el mismo nombre ¿cómo hacemos referencia a una u otra columna? Para ello hay que indicar el **nombre de la tabla seguido de un punto y el nombre de la columna** ("*clientes.nombre*") también podemos usar **alias** para evitar ambigüedades (pensemos en un **JOIN** entre filas de la misma tabla) o aumentar la comprensión de la consulta.

- Veamos un **ejemplo**, queremos listar los ***nombres*** de las empresas y su ***provincia*** de procedencia. Si sólo consultamos la tabla ***clientes*** podremos ver solamente el ***codprovincia*** pero eso no es suficiente, queremos saber el nombre de las provincias, para ello es necesario consultar también la tabla ***provincias***:

NOMBRE	PROVINCIA
REDESMAR	Almería
Logística Gádor	Almería
Gestión y Control	Almería
TECNORED	Almería
INFOSUR	Almería
AgroPoni	Almería
INFOGRANA	Granada
MENOGA	Sevilla
Cons.Educación	Sevilla

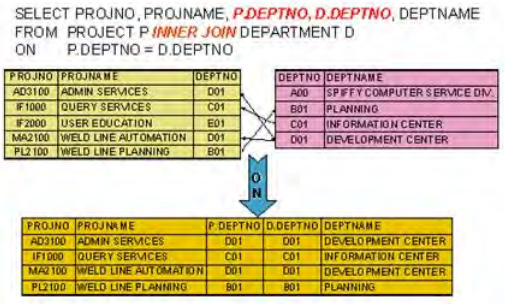
```
SELECT clientes.nombre, provincias.provincia FROM clientes, provincias
```

Con ello obtenemos el **producto cartesiano**, es decir todas las posibles combinaciones de filas entre ***clientes*** y ***provincias*** *(11 provincias * 9 clientes = 99 filas).* Pero lo que nos interesa es enlazar una tabla con otra para que se vea sólo una fila por cliente con su provincia. Para ello debemos utilizar SELECT **comparando la clave primaria** de provincias (codprovincia) **con la clave ajena** de clientes *(9 filas):*

```
SELECT clientes.nombre, provincias.provincia FROM clientes, provincias
WHERE clientes.codprovincia=provincias.codprovincia
```

- A esta última consulta se le llama **Composición interna o producto interno**, y puede realizarse también mediante la cláusula **INNER JOIN...ON** que tiene la ventaja de usar índices para realizar más rápido el proceso. La consulta anterior podíamos escribirla así:

```
SELECT clientes.nombre, provincias.provincia FROM clientes
INNER JOIN provincias ON clientes.codprovincia=provincias.codprovincia
```



- La composición interna muestra sólo las filas en las que coincide el valor de la clave externa pero pensemos en la tabla ***clientes*** y ***facturas.*** No se les ha facturado a todos los ***clientes***, por tanto en un **JOIN interno** no aparecerían los clientes sin factura. Si tenemos interés en que aparezcan podemos usar los **JOIN externos o productos externos**, hay dos posibilidades **LEFT JOIN** que muestra todas las filas, incluidos valores nulos o no coincidentes en las claves ajenas, de la primera tabla (tabla de la izquierda), y **RIGHT JOIN** que lo hace de la segunda tabla (tabla de la derecha). Por tanto, si hacemos este JOIN obtenemos *7 filas*:

```
SELECT clientes.*, facturas.fecha FROM clientes INNER JOIN facturas ON clientes.idcliente=facturas.idcliente
```

Observa el uso del ***** para ver todas las columnas de la tabla ***clientes***. Si utilizamos el **LEFT JOIN** recuperaremos las filas de todas las facturas y todos los clientes aunque no tengan factura *(9 clientes más un cliente repetido por tener 2 facturas: 10 filas):*

```
SELECT c.*, f.fecha FROM clientes c LEFT JOIN facturas f ON c.idcliente=f.idcliente
```

Observa el uso de los alias (c para clientes y f para facturas) no son obligatorias pero facilitan la escritura de los

comandos.

- También es posible utilizar la **Unión de tablas**, que se realiza con el comando **UNION**, y sirve para unir las filas de dos tablas distintas en una sola tabla. Evidentemente, las filas de ambas tablas deben tener el mismo número de columnas y ser del mismo tipo. Las tablas también pueden ser el resultado de consultas **select**. Por ejemplo, si queremos saber los datos de los ***clientes*** que tienen alguna ***factura*** o alguna ***reparación*** con nuestro taller (aunque no se haya facturado aún) tenemos que hacer dos consultas **JOIN** y unirlas de esta manera:

```
SELECT c.* FROM clientes c INNER JOIN reparaciones r ON c.idcliente=r.idcliente)
UNION (SELECT c.* FROM clientes c INNER JOIN facturas f ON c.idcliente=f.idcliente)
```

Con la consulta anterior se eliminan las filas repetidas, si queremos verlas usaremos **UNION ALL**:

```
(SELECT c.* FROM clientes c INNER JOIN reparaciones r ON c.idcliente=r.idcliente)
UNION ALL (SELECT c.* FROM clientes c INNER JOIN facturas f ON c.idcliente=f.idcliente)
```

Te recomendamos que practiques con los comandos vistos. Además, a lo largo de los siguientes apartados usaremos los conceptos vistos aquí y completaremos el repaso con más ejemplos.

Autoevaluación

¿Con qué composición de dos tablas obtendría en general más filas?

a) Producto Cartesiano de las dos tablas
b) Producto Interno INNER JOIN
c) Producto Externo LEFT JOIN
d) Producto Externo RIGHT JOIN

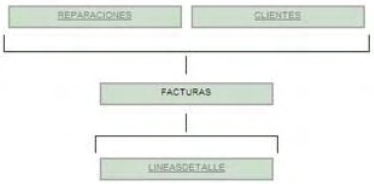
Comprobar

PARA SABER MÁS

Aquí te presentamos un tutorial de SQL donde puedes encontrar muchos más ejemplos explicados de la manipulación y definición de objetos de base de datos, te recomendamos su visita:

Tutorial de SQL

Uso de funciones y operadores SQL



Seguro que recuerdas de tu anterior estudio de los DML, que SQL disponía de más **funciones y operadores** de los que hemos visto ahora, que podían ser utilizados en las expresiones y daban más potencia a las consultas. Vamos a continuar el repaso a través de distintos ejemplos donde usamos algunos de estos recursos:

Hay **funciones de matemáticas, para manejo de cadenas de caracteres, de fechas, de conversión de tipos, etc.** En esta tabla las usamos en distintos **ejemplos**:

- **Valores nulos:** permite poner una condición sobre un campo o expresión (condición), si se cumple se muestra "cierto", si no se muestra "falso", **NVL2 (condición, cierto, falso).** Por ejemplo, mostrar los nombres y CIF de los clientes con la palabra NULO si no los tenemos almacenados:

```
SELECT nombre,NVL2(cif,cif,'NULO') cif FROM clientes
```

También podemos consultar si un valor es nulo con **ISNULL**, veamos los CIF nulos de la tabla clientes:

```
SELECT nombre FROM clientes WHERE cif IS NULL
```

- **Operaciones sobre cadenas de caracteres:** hay diferentes funciones como **LENGTH** para ver la longitud de una cadena, **CONCAT** ó || para concatenar cadenas, **INSTR** para buscar caracteres en una cadena, **SUBSTR** para substraer subcadenas, **RTRIM** para eliminar blancos, **LOWER** para pasar a minúsculas y **UPPER** a mayúsculas, etc. Algunos ejemplos:

¿Nombre y apellidos de los empleados?

```
select nombre||' '|| apellido1|| ' ' || apellido2 nombre_completo from empleados
```

¿Cuántos caracteres tiene el nombre completo del empleado 5?

```
SELECT nombre||apellido1||apellido2 nombre_completo, length (rtrim(to_char(nombre)) ||
rtrim(to_char(apellido1)) || rtrim(to_char(apellido2))) longitud from empleados where idempleado='5'
```

NOMBRE_COMPLETO	LONGITUD
Ana Moreno Sierra	15

El nombre completo lo obtenemos concatenando con || los campos *nombre, apellido1 y apellido2* y le ponemos el alias *nombre_completo*, debemos pasarlos a char (**to_char**) y eliminar los caracteres en blanco (**rtrim**) para calcular la longitud (**length**) también con **alias**.

¿Qué nos devuelve si escribimos esto?:

```
SELECT INSTR('Hola', 'o') FROM DUAL
```

En efecto devuelve 2, el lugar de la letra o.

¿Y si queremos saber la letra del NIF de nuestros clientes?

```
SELECT SUBSTR(CIF, 9, 1) FROM clientes
```

ORACLE DATABASE EXPRESS EDITION

¿Para ver el nombre de los clientes en mayúsculas?

```
SELECT UPPER(nombre) FROM clientes
```

- **Funciones numéricas,** veamos algunas mediante ejemplos:

SQL	Resultado
`SELECT ABS(-15) "Absolute" FROM DUAL;`	15
`SELECT CEIL(15.7) "Ceiling" FROM DUAL;`	16
`SELECT MOD(11,4) "Modulus" FROM DUAL;`	3
`SELECT TRUNC(15.79,1) "Truncate" FROM DUAL;`	15,7

- **Funciones de fecha y hora,** unos ejemplos más autoexplicativos:

SQL	Resultado
`SELECT SYSDATE FROM DUAL;`	Fecha actual
`SELECT TO_CHAR(SYSDATE,'MM-DD-YYYY HH24:MI:SS') Ahora FROM DUAL;`	Fecha y hora actual
`SELECT ADD_MONTHS(SYSDATE, 3) FROM DUAL;`	Añade 3 meses a fecha actual
`SELECT ROUND (TO_DATE ('27-OCT-07'),'YYYY') "Año nuevo" FROM DUAL;`	Se redondea a 1-1-2008

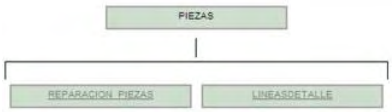
Otras útiles **funciones de conversión** son: **`TO_CHAR, TO_DATE, TO_NUMBER`**

- **Número de filas del resultado**: por ejemplo queremos conocer las 3 piezas más caras:

```
SELECT * FROM (SELECT * FROM PIEZAS ORDER BY COSTO DESC) WHERE ROWNUM <=3
```

Ahora vamos consultar la suma del costo de las piezas agrupadas por categoría, y queremos saber qué 3 categorías tienen la suma de costo mayor:

```
SELECT * FROM (SELECT CODTIPO, SUM (COSTO) costo_total FROM PIEZAS
GROUP BY CODTIPO ORDER BY costo_total DESC) WHERE ROWNUM <=3
```



- **Operadores de comparación,** veamos algunos operadores para combinar consultas a través de unos ejemplos:

Imaginemos que los empleados de nuestra empresa van a pertenecer a diferentes departamentos, uno de ellos es el de administración (adm), y supongamos que almacenamos la fecha en la que ingresaron a la empresa. Si queremos obtener la lista de los empleados que tienen más antigüedad que alguno del departamento de administración "adm" escribiríamos (usando **SOME**):

```
SELECT nombre,fecha_ingreso,idEmpleado
 FROM empleados
 WHERE fecha_ingreso < SOME(SELECT fecha_ingreso FROM empleados WHERE idDep='adm')
```

Si ahora queremos la lista de empleados de los que tienen más antigüedad que cualquiera del departamento de administración escribiremos (usando **ALL**):

```
SELECT nombre,fecha_ingreso,idEmpleado
 FROM empleados
 WHERE fecha_ingreso < ALL(SELECT fecha_ingreso FROM empleados WHERE idDep='adm')
```

Volviendo a nuestro ejemplo del taller de informática... Para mostrar todas las piezas hardware que tienen un precio superior a cualquier producto software (usando **ANY,** es lo mismo que **SOME**):

```
SELECT * FROM piezas WHERE (codtipo<>6) and (costo> ANY (SELECT costo FROM piezas WHERE codtip
```

Consultamos los clientes que tienen facturas posteriores a marzo (usando **EXISTS**):

```
SELECT idcliente,nombre FROM clientes c WHERE
EXISTS (SELECT * FROM facturas f WHERE fecha > '31/03/2007' AND c.idcliente=f.idcliente)
```

> ***PARA SABER MÁS***
> ***Si quieres profundizar en el uso de las funciones SQL en Oracle debes visitar el manual de referencia que nos proporciona desde su web:***
> ***Funciones SQL en Oracle*** [Versión en caché]

Control de usuarios con SQL

En Oracle los usuarios pueden realizar las operaciones para las que tienen permiso, el **administrador** del SGBD (cuenta **SYSTEM**) puede dar **permisos y privilegios** a los distintos objetos de la BD según sea necesario, además, cualquier usuario puede dar permisos a otro sobre sus propios objetos. Para llevar a cabo estas acciones podemos usar las sentencias SQL:

- **GRANT** para conceder permisos, y
- **REVOKE** para quitarlos.

La **sintaxis** de **`GRANT`** es la siguiente:

```
GRANT <tipo_privilegio de SISTEMA/OBJETO> TO <usuario> [IDENTIFIED BY '<password>'] [WITH ADMIN OPTION |
WITH GRANT OPTION]
```

Identified sirve para indicar la contraseña del usuario, **with admin option** permite al usuario concederlos **permisos de sistema** concedidos, y **with grant option** permite al usuario conceder los **permisos de objeto** concedidos. Podemos usar **ALL** para conceder todos los privilegios sobre un objeto.

Algunos permisos que se pueden conceder sobre el sistema son:

- **CREATE USER**
- **CREATE TABLE**
- **CREATE VIEW**

Algunos de los privilegios que pueden concederse o revocarse sobre los objetos son:

1. **SELECT**: permite la consultar a un objeto.
2. **INSERT**: permite insertar filas en una tabla o vista.
3. **UPDATE**: permite actualizar filas en una tabla o vista.
4. **DELETE**: permite borrar filas dentro de la tabla o vista.
5. **ALTER**: permite alterar la tabla.
6. **INDEX**: permite crear índices de una tabla.
7. **REFERENCES**: permite crear claves ajenas que referencien a esta tabla.
8. **ALL**: concede todos los permisos

Vamos a practicar como siempre con algunos **ejemplos SQL**:

- Primero vamos a **crear un usuario *rosa*** con contraseña ***1234***:
  ```
  CREATE USER rosa IDENTIFIED BY "1234"
  ```
- Podemos cambiar la contraseña con:
  ```
  ALTER USER rosa IDENTIFIED BY "PASO"
  ```
- También podemos obligarle a cambiar la contraseña:
  ```
  ALTER USER rosa PASSWORD EXPIRE
  ```
- Podemos bloquear/desbloquear cuentas con LOCK/UNLOCK:
  ```
  ALTER USER rosa ACCOUNT LOCK
  ```
- Le concedemos los **roles** de conexión a base de datos y creación de objetos:
  ```
  GRANT CONNECT, RESOURCE TO rosa
  ```
- Le concedemos permisos para consular la tabla de clientes:
  ```
  GRANT SELECT ON usuario_taller.clientes TO rosa
  ```
- Debes tener en cuenta que si entras al sistema como *rosa* y deseas acceder a la tabla *clientes* debes escribir:
  ```
  Select * from usuario_taller.clientes
  ```
- Concedemos todos los **permisos** sobre la tabla de empleados y puede conceder permisos sobre ella:
  ```
  GRANT ALL ON usuario_taller.clientes TO rosa WITH GRANT OPTION
  ```
- Es posible ver los **nombres de usuario y sus contraseñas** (encriptadas) si somos administradores:
  ```
  SELECT username, password FROM dba_users
  ```
- Podemos crear un Nuevo rol
  ```
  CREATE ROLE taller
  ```
- Le damos **privilegios** y lo asignamos el nuevo rol a *rosa*
  ```
  GRANT ALL ON usuario_taller.empleados TO taller
  GRANT taller TO rosa
  ```
- Podemos consultar el **Diccionario de datos** para ver los roles y privilegios asignados a *rosa*:
  ```
  Select * from dba_role_privs where grantee='ROSA'
  Select * from dba_tab_privs where grantee='ROSA'
  ```

```
Select * from dba_tab_privs  where grantee='ROSA'
```

Resultados | Explicar | Describir | SQL Guardado | Historial

GRANTEE	OWNER	TABLE_NAME	GRANTOR	PRIVILEGE	GRANTABLE	HIERARCHY
ROSA	USUARIO_TALLER	CLIENTES	USUARIO_TALLER	FLASHBACK	YES	NO
ROSA	USUARIO_TALLER	CLIENTES	USUARIO_TALLER	DEBUG	YES	NO
ROSA	USUARIO_TALLER	CLIENTES	USUARIO_TALLER	QUERY REWRITE	YES	NO
ROSA	USUARIO_TALLER	CLIENTES	USUARIO_TALLER	ON COMMIT REFRESH	YES	NO
ROSA	USUARIO_TALLER	CLIENTES	USUARIO_TALLER	REFERENCES	YES	NO
ROSA	USUARIO_TALLER	CLIENTES	USUARIO_TALLER	UPDATE	YES	NO
ROSA	USUARIO_TALLER	CLIENTES	USUARIO_TALLER	INSERT	YES	NO
ROSA	USUARIO_TALLER	CLIENTES	USUARIO_TALLER	INDEX	YES	NO
ROSA	USUARIO_TALLER	CLIENTES	USUARIO_TALLER	DELETE	YES	NO
ROSA	USUARIO_TALLER	CLIENTES	USUARIO_TALLER	ALTER	YES	NO
ROSA	USUARIO_TALLER	CLIENTES	USUARIO_TALLER	SELECT	YES	NO

11 filas devueltas en 0.00 segundos Exportación de CSV

- Podemos **modificar el rol** con ALTER ROLE, o eliminarlo con:
  ```
  DROP ROLE taller
  ```
- Para **eliminar un usuario** con todos sus objetos (**CASCADE**):
  ```
  DROP USER rosa CASCADE
  ```

Autoevaluación

¿Qué comando SQL usaremos para asignar el privilegio insertar filas en la tabla piezas al usuario rosa?

a) GRANT rosa TO INSERT ON piezas.
b) GRANT INSERT ON piezas TO rosa.
c) GRANT piezas TO INSERT ON rosa.
d) GRANT INSERT TO piezas ON rosa.

Comprobar

Concurrencia (I): Transacciones

Imagina que vas al banco y pides que traspasen dinero de tu cuenta a otra cuenta distinta. Cuando el cajero realiza la operación con el ordenador, el SGBD debe restar la cantidad de tu cuenta y sumarla a la otra cuenta.

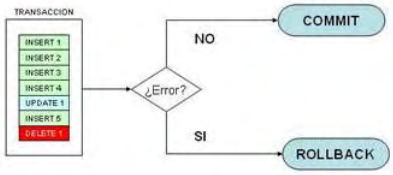
¿Y si realiza la primera parte y el ordenador falla antes de realizar la segunda?, ¿pierdes el dinero?

También puede ocurrir un problema si como consecuencia del uso concurrente de nuestra base de datos, varios usuarios están modificando los mismos datos de la misma tabla. ¿Qué puede hacer el SGBD para solucionar estos problemas?

Como vimos al principio de la unidad, todos los SGBD tienen una gran responsabilidad en la **integridad y seguridad de la información** que almacenan y procesan, ya que en muchas ocasiones (como en el banco) almacenan datos sensibles con importante repercusión en las personas. Por tanto, cuando diseñamos y manejamos la BD debemos garantizar la **seguridad de la información** (usuarios, roles, etc) y garantizar la **integridad de los datos** (valor correcto en las cuentas de los clientes del banco), para lo que los SGBD tienen distintos mecanismos. En este apartado vamos a ver que Oracle nos ofrece el control de **transacciones y los bloqueos** como mecanismos de garantizar la consistencia de la BD. Pero, ¿qué es una transacción?

Podemos **definir una transacción como un conjunto de sentencias SQL que se ejecutan como una única operación, confirmándose o deshaciéndose en grupo**. Las operaciones que se incluyen en las transacciones son las propias de la manipulación de datos: **SELECT, INSERT y DELETE**.

Para confirmar una transacción se utiliza la **sentencia COMMIT**, de manera que al ejecutar COMMIT los cambios se escriben realmente en la base de datos. Podemos, sin embargo deshacer una transacción utilizando la **sentencia ROLLBACK**, quedando la base de datos en el mismo estado que antes de iniciarse la transacción.

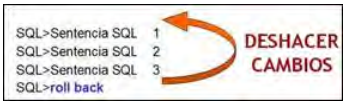


ORACLE es completamente transaccional, es decir, siempre debemos especificar si queremos deshacer o confirmar la transacción. Por tanto, una transacción comienza con la primera sentencia SQL ejecutable, y finaliza explícitamente con un **COMMIT** o un **ROLLBACK**, o bien implícitamente con una sentencia DDL, o cuando termina la sesión del usuario, o cuando comienza una nueva transacción. Si alguna de las tablas afectadas por la transacción tiene **triggers**, las operaciones que realiza el trigger están dentro del ámbito de la transacción, y son confirmadas o deshechas conjuntamente con la transacción. Si durante la ejecución de cualquier sentencia SQL se produce un **error** por sintaxis, se deshacen todos los efectos de la sentencia con un **ROLLBACK** automático.

¿Cómo puede hacer esto Oracle? Como vimos en el apartado 2.2 (características de Oracle Express), existen los archivos **REDOLOG FILE** donde se almacenan las transacciones realizadas, por lo que pueden recuperarse.

Veamos con un **ejemplo** el uso de **COMMIT** y **ROLLBACK**. Si queremos añadir una factura a nuestra BD del Taller porque un cliente (idcliente 1) ha comprado un disco duro (numero_serie HD100) y un sistema operativo (numero_serie MS400), hay que tener en cuenta que debo añadir primero la factura (codfactura 10) en la tabla FACTURAS, y después las líneas_detalle que desglosan la factura, esto incluye varios comandos insert, pero deben ejecutarse todos juntos como una sola instrucción, porque si no, la BD tendría información incompleta de la factura. Esto sería un ejemplo de transacción. Si para llevar acabo esta factura escribimos:

```
INSERT INTO FACTURAS (CODFACTURA, IDCLIENTE, FECHA, IVA, ID_REPARACION)
VALUES ('10', '1', TO_DATE('2007-04-5', 'YYYY-MM-DD HH24:MI:SS'), '16', '8');

INSERT INTO LINEASDETALLE (CODLINEA, NUMERO_SERIE, CODFACTURA, CANTIDAD, PRECIOUNITARIO)
VALUES ('10', 'HD100', '10', '1', '120');

INSERT INTO LINEASDETALLE (CODLINEA, NUMERO_SERIE, CODFACTURA, CANTIDAD, PRECIOUNITARIO)
VALUES ('10', 'MS400', '10', '1', '484');

ROLLBACK;
```



La transacción no se habrá realizado, ya que hemos usado ROLLBACK, pero si volvemos a escribir todos los comandos cambiando ROLLBACK por COMMIT, veremos que sí se realiza la operación. Podemos **probarlo con el script SQL** para probar transacciones que incluimos a continuación:

Descarga el script

Para comprobar el resultado basta con usar el explorador de objetos y ver si se añaden las filas o no. El script debemos modificarlo para cambiar **ROLLBACK** por **COMMIT** desde el editor de Oracle Express, tal y como se explica en la próxima animación.

Podemos fijar **puntos de restauración intermedios** dentro de un bloque de transacción, de manera que podemos elegir que el **ROLLBACK** vuelva a uno de estos puntos intermedios fijados. Para ello, los puntos se marcan con **SAVEPOINT** y el **ROLLBACK** puede hacerse al punto indicado con SAVEPOINT. Un ejemplo simple sería el siguiente:

```
UPDATE PROVINCIAS SET PROVINCIA='ALBACETE' WHERE CODPROVINCIA ='21';

SAVEPOINT p1;

UPDATE PROVINCIAS SET PROVINCIA='MADRID' WHERE CODPROVINCIA ='23';

SAVEPOINT p2;

UPDATE PROVINCIAS SET PROVINCIA='VALENCIA' WHERE CODPROVINCIA ='25';

ROLLBACK TO SAVEPOINT p1;
```

Tras su ejecución solo habría tenido efecto el primer UPDATE. Para probar todo esto vamos a incluir a continuación una animación sobre control de transacciones, que va a usar el acceso directo del grupo de programas de Oracle Express "**Ejecutar Línea de Comandos SQL**". Para conectarse a la base de datos desde la línea de comandos hay que usar el comando **CONNECT**, para iniciar un script SQL hay que usar el comando **START scriptsql** indicando la ruta del sistema operativo del archivo. Desde este entorno **SQL*Plus** de línea de comandos los cambios no son definitivos en la base de datos hasta que hacemos **COMMIT** o salimos de la sesión con **EXIT**. No olvidemos que los comandos acaban con el carácter punto y coma (;)

Transacciones con Oracle Express

Concurrencia (II): Bloqueos en SQL

Hemos visto cómo gestionar las transacciones y volver a puntos anteriores del estado de la base de datos, pero Oracle proporciona otro mecanismo para el control de la concurrencia llamado **bloqueo**. Permite **bloquear el acceso a un determinado objeto como una tabla o una vista**. El objetivo es garantizar que durante el tiempo que se trabaje con la tabla, no sea modificada por el usuario y que otros usuarios puedan acceder a la tabla. Los bloqueos se crean con el comando **LOCK TABLE**, y se quitan con COMMIT y ROLLBACK. La sintaxis es la siguiente:

```
LOCK TABLE [usuario.[<tabla>|<vista>]] IN <modo bloqueo> MODE [NOWAIT]
```

NOWAIT indica que el sistema no espera hasta que finalice el bloqueo. Si no se pone esta opción y se intenta entrar una tabla bloqueada, el SGBD se queda esperando hasta que termine el bloqueo. El **<modo bloqueo>** indica las operaciones permitidas mientras dura el bloqueo, y son de menor a mayor protección:

- **ROW SHARE:** Es el nivel mínimo de bloqueo y se usa para modificar algunas filas. El resto de usuarios puede consultar o modificar otras filas, pero no pueden bloquearla con EXCLUSIVE
- **ROW EXCLUSIVE:** Similar al anterior pero impide que otros usuarios bloqueen la tabla para uso compartido.
- **SHARE:** Permite que otros usuarios vean la tabla, pero las actualizaciones sólo se pueden realizar sobre filas específicas. Además, impide que el resto bloquee la tabla de forma más restringida.
- **SHARE ROW EXCLUSIVE:** Permite ver los registros de la tabla, pero no permite otros bloqueos sobre la tabla.
- **EXCLUSIVE:** Es el nivel máximo de bloqueo, de manera que el usuario que realiza el bloqueo es el único que puede hacer operaciones sobre la tabla, el resto no puede hacer nada.

Los bloqueos tienen sentido dentro de un módulo de programa, por lo que su uso práctico se verá cuando desarrollemos aplicaciones a lo largo de este módulo. Aquí vamos a ver un par de **ejemplos** que deberían ser ejecutados dentro de una transacción de un módulo de código, como por ejemplo en un procedimiento PL/SQL:

```
LOCK TABLE usuario_taller.facturas IN SHARE MODE NOWAIT

LOCK TABLE usuario_taller.reparaciones IN ROW SHARE MODE
```

Como hemos visto, Oracle posee sentencias SQL que garantizan un control adecuado de las transacciones para que no haya problemas en el uso concurrente de la base de datos.

Autoevaluación

Observa el ejemplo anterior:

```
UPDATE PROVINCIAS SET PROVINCIA='ALBACETE' WHERE CODPROVINCIA ='21';

SAVEPOINT p1;

UPDATE PROVINCIAS SET PROVINCIA='MADRID' WHERE CODPROVINCIA ='23';

SAVEPOINT p2;

UPDATE PROVINCIAS SET PROVINCIA='VALENCIA' WHERE CODPROVINCIA ='25';

ROLLBACK TO SAVEPOINT p1;
```

Hemos hecho: **`ROLLBACK TO SAVEPOINT p1`** ¿Qué habría pasado si hubiésemos escrito **`ROLLBACK TO SAVEPOINT p2`**?

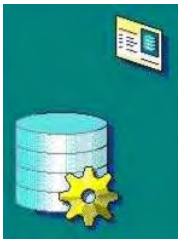
a) El "**codprovincia"** 21 cambiaría a 'ALBACETE', el 23 a 'MADRID' y el 25 a 'VALENCIA'
b) El "**codprovincia"** 21 cambiaría a 'ALBACETE', el resto sin cambios
c) El "**codprovincia"** 21 cambiaría a 'ALBACETE', el 23 a 'MADRID' y el 25 sin cambios.
d) No se efectuaría ningún cambio debido al uso de **`ROLLBACK`**

Comprobar

*CASO. **María** comenta que para realizar la conexión con la base de datos deberán programar la API JDBC con Java. **Víctor** ha comenzado a recopilar documentación de JDBC y ha visto que es muy extensa y no entiende por qué, **María** le explica que el objetivo de JDBC es que un programador de Java pueda implementar el acceso a cualquier base de datos sin preocuparse de las características específicas de cada SGBD, lo cual implica que el API JDBC sea tan grande y complejo. Sin embargo es imprescindible conocerlo bien. Por ello, **Víctor** comienza a estudiar y practicar con JDBC. Pasados unos días en los que **Víctor** ha estado entregado al estudio*

*de JDBC, **Carmen** tranquiliza a Víctor y le dice que JDeveloper ofrece potentes mecanismos que permiten realizar una conexión con la base de datos con unos cuantos clic de ratón, **Víctor** no lo tiene claro, así que **Carmen** le apuesta una cerveza a que le enseña a conectar con la BD de Oracle Express desde JDeveloper en 5 minutos...*

Hemos estado viendo el proceso para diseñar, crear y manejar bases de datos, todo ello lo podemos hacer con la ayuda del SGBD Oracle Express. Para el diseño nos ayudamos de herramientas CASE como la proporcionada por JDeveloper. Sin embargo, no perdamos de vista nuestro objetivo, que es diseñar una aplicación informática que almacene la información en una base de datos, como por ejemplo la gestión necesaria para un taller informático.



La aplicación informática la vamos a desarrollar con Java utilizando JDeveloper, pero necesitamos que esta aplicación esté conectada a la base de datos que hemos realizado en Oracle Express para el taller informático. En la unidad anterior hemos visto la arquitectura que deseamos utilizar para nuestra aplicación. Según vimos, utilizaremos la arquitectura ADF proporcionada por Oracle basada en la estructura Modelo-Vista-Controlador. En esta arquitectura tenemos la capa del modelo de datos que tratará con la información almacenada en la base de datos. Para poder llevar a cabo esto es necesario establecer una conexión desde la capa inferior con la base de datos.

¿Cómo realizaremos esta conexión?

En este apartado contestaremos a esta pregunta, para lo cual será necesario conocer el Java DataBase Connectivity (JDBC). Nos centraremos en los aspectos técnicos de la programación con JDBC, por tanto para que los conceptos queden claros, no nos vamos a preocupar demasiado de la interfaz de usuario ni de la arquitectura MVC. Una vez que conozcamos bien la programación JDBC, en el próximo apartado pondremos diferentes ejemplos de uso del acceso JDBC a bases de datos pensando en una perspectiva de desarrollo de aplicaciones.

Aplicaciones con conexión a base de datos

Descripción de JDBC

Cada SGBD tiene sus propias particularidades que hacen que su acceso desde una aplicación sea diferente. Por ejemplo, en el **Módulo Profesional** de "***Desarrollo de Aplicaciones en Entornos de Cuarta Generación y con Herramientas CASE***" viste el manejo del SGBD MySQL, ahora has visto Oracle Express y has podido comprobar múltiples diferencias, por ejemplo, en la sintaxis de SQL a pesar de que ambos siguen los estándares ANSI.

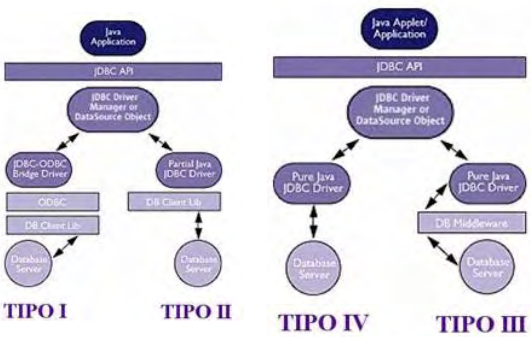
Cuando estamos programando el acceso a una base de datos desde un lenguaje de programación, en principio debemos tener en cuenta todas sus particularidades para que todo funcione correctamente. Esto exigiría una programación específica para cada SGBD, lo que comprenderás que es poco deseable, porque haría nuestras aplicaciones poco portables, o excesivamente dependientes del SGBD elegido, si lo prefieres. Para evitar esto la industria ha proporcionado distintas soluciones.

- Microsoft desarrolló el **Open DataBase Connectivity (ODBC) para Windows**:
  Es un estándar de acceso a Bases de Datos que hace posible el acceso a la base de datos desde cualquier aplicación, sin importar qué Sistema Gestor de Bases de Datos almacene los datos. Para ello, se inserta una capa intermedia (driver) entre la aplicación y el DBMS que traduce las instrucciones de la aplicación a comandos que el DBMS entienda. Sin embargo ODBC es difícil de programar, está desarrollado en C, lo que limita la portabilidad Java, y no es software libre, por lo que se desarrolló el **Java DataBase Connectivity (JDBC)** pensado para utilizarse desde aplicaciones Java.
- **JDBC es un API (Application Programming Interface) desarrollado por Sun Microsystems** que permite escribir aplicaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute, del SGBD o de la versión de SQL. Cada fabricante de SGBD debe proporcionar su propio controlador de conexiones a base de datos JDBC (driver), de manera que actualmente todos los SGBD, como por ejemplo Oracle Express, proporcionan su driver JDBC.

JDBC tiene dos interfaces diferenciadas:

- Una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa Java acceder a sistemas de bases de datos. Es utilizada por el programador de aplicaciones.
- Una interfaz del manejador de conexión a la base de datos (driver) que establece la comunicación entre el API JDBC y la base de datos real traduciendo las operaciones al protocolo nativo, por lo que dependerá de cada SGBD.

Sun divide a los drivers JDBC que se instalan en el cliente en 4 categorías:



- **Tipo I JDBC-ODBC bridge**: establece un puente JDBC-ODBC por lo que es necesario convertir todas las llamadas JDBC a llamadas ODBC y tener en la máquina cliente un controlador ODBC que acceda a la base de datos, todo ello ralentiza el proceso por lo que no suele utilizarse comercialmente.

- **Tipo II Native-API Driver**: Este driver parcialmente escrito en Java, es más rápido porque no requiere ODBC al tratar directamente con la librería nativa del SGBD, pero exige la existencia de un código binario no java (se debe cargar la librería con el driver de acceso al SGBD) en la máquina del cliente, por lo que no es adecuado para aplicaciones Web. Un ejemplo de este **JDBC son los drivers OCI proporcionados por Oracle** y que se conocen como **thick driver**.
- **Tipo III Pure Java Driver for Database Middleware**: Es un driver realizado completamente en Java que se comunica con el SGBD utilizando el protocolo de red nativo del servidor. La ventaja de este tipo de driver es que es una solución independiente de la máquina en la que se va a ejecutar el programa al no requerir código binario dependiente del SGBD en la máquina cliente. Requiere un servidor intermedio (**middleware**) entre la máquina que ejecuta la aplicación Java y la máquina que ejecuta el SGBD, de manera que el nivel intermedio mantiene el control de las operaciones proporcionando seguridad y optimización de los accesos (balanceo de carga...). Además, se libera a la máquina cliente de la instalación del controlador JDBC. El middleware proporciona acceso a datos que residan en distintos SGBD, por lo que este modelo es adecuado para soluciones de grandes proyectos.
- **Tipo IV Direct to Database Pure Java Driver**: es la opción más flexible, se trata de un driver totalmente Java e independiente del protocolo del SGBD que convierte las peticiones JDBC en protocolos de red para cada base de datos. El driver del lado del cliente está incluido en la máquina java del cliente. Al no requerir capas intermedias no perjudica el rendimiento y es transparente al cliente por lo que es apto para desarrollar applets. Su uso está muy extendido, un ejemplo es el **driver JDBC de Oracle llamado thin.**

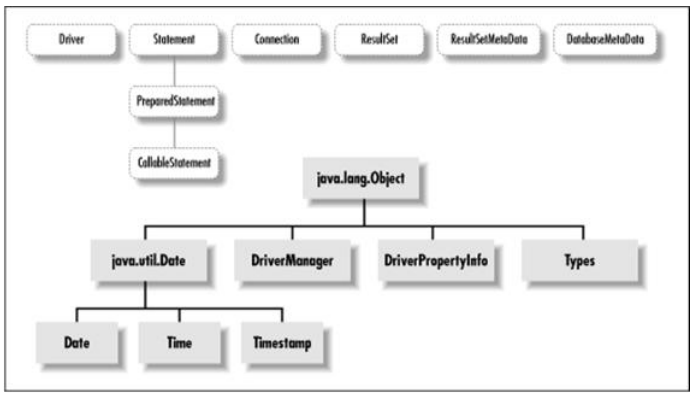
> ***PARA SABER MÁS***
>
> ***Puedes profundizar en el estudio de JDBC visitando la página oficial de Sun, en ella encontrarás una descripción de JDBC, y la documentación para las diferentes versiones J2SE 1.4.2 \| J2SE 5.0 \| Java SE 6:***
>
> ***Web de Sun: JDBC***

El API JDBC

Hemos visto la estructura y tipos de drivers JDBC, hemos indicado que JDBC ofrece una API con clases Java,... ¿Cómo son estas clases?

En la penúltima unidad del Módulo Profesional **Programación en Lenguajes Estructurados** se explica su uso para el SGBD MySQL. En este caso vamos a verlo para Oracle, aunque como comprobarás su funcionamiento es básicamente el mismo.

Las clases e interfaces de JDBC están incluidas en el paquete java.sql e incluyen clases, manejo de excepciones e interfaces tal y como podemos ver en la figura.



Además, Oracle nos ofrece otras librerías que podemos utilizar desde JDeveloper:

- **oracle.jdbc:** este interfaz define las **extensiones de Oracle** al interfaz estándar incluido en java.sql, permitiendo utilizar las mejoras Java al rendimiento y facilidad de uso del acceso a bases de datos (por ejemplo posibilita el uso directo de cursores PL/SQL) y proporciona acceso al formato de datos Oracle SQL.
- **oracle.sql:** da soporte al acceso directo a datos en formato SQL por medio de un sistema **de** mapeo **de tipos SQL** y unas clases de apoyo, los EJB utilizan este sistema.
- **oracle.jdbc.pool:** permite **realizar** pooling **en las conexiones** a la base de datos.

¿Y cómo funcionan estas clases? Veamos, la clase que se encarga de cargar inicialmente el **driver JDBC** es **DriverManager**. Una aplicación puede utilizar **DriverManager** para obtener un objeto de tipo conexión, **Connection**, que se liga con una base de datos concreta. La conexión se indica siguiendo una sintaxis basada en la especificación URL que incluye el controlador de BD, el tipo de driver, el servidor donde está el SGBD, el puerto de acceso, y el identificador de base de datos en el caso de Oracle (System Identifier SID). Además hay que identificarse en la BD con un usuario y contraseña válidos. Podemos usar esta especificación estándar para conectarnos a Oracle Express:

"jdbc:oracle:thin:@localhost:1521:XE". Indica driver JDBC con protocolo y subprotocolo Oracle Thin, en el servidor local (localhost), con el puerto TCP/IP 1521, y el identificador de la base de datos Oracle Express (cambiar **XE** por **ORCL** en el caso de una base de datos Oracle que no sea Express), este último sólo es necesario indicarlo cuando usamos el driver thin.

Una vez que se tiene un objeto de tipo **Connection**, se pueden crear sentencias **Statement**, ejecutables basadas en el lenguaje SQL. Estas sentencias pueden devolver resultados almacenados como objetos de tipo **ResultSet**. En la tabla podemos ver una pequeña descripción de las principales clases:

CLASES INTERFACES	DESCRIPCIÓN
Driver	**Permite conectarse a una BD, cada SGBD requiere su driver**
DriverManager	Carga y configura el driver instalado en el sistema
DriverPropertyInfo	**Proporciona información acerca de un driver**
Connection	Realiza la conexión y autenticación con una BD, se puede tener más de una conexión con más de

	una BD
DatabaseMetadata	**Proporciona información acerca de la BD, como las tablas que contiene, etc.**
Statement	Permite ejecutar sentencias SQL sin parámetros en la BD.
PreparedStatement	**Permite ejecutar sentencias SQL con parámetros de entrada indicados con ?**
CallableStatement	Permite ejecutar procedimientos PL/SQL en la BD
ResultSet	**Contiene las filas de resultado tras un acceso a la BD con un Statement**
ResultSetMetadata	Nos da información sobre un **ResultSet**, como el número de columnas, sus nombres, etc.

Ya conocemos qué necesitamos para conectarnos a Oracle desde JDeveloper, y la estructura de JDBC.

¿Lo probamos?

Puedes ver el proceso en la siguiente animación que establece una conexión con la base de datos del Taller de Informática y explica cómo manipular la base de datos desde JDeveloper. En la anterior animación con JDeveloper vimos cómo diseñar bases de datos y crear un script SQL, ahora se trata de conectar con una BD que ya existe. Por eso, antes nos indicaba que estábamos con una base de datos "offline", sin conexión.

Insertar aquí PEG04_Recurso15_JDeveloper2.ppt

Autoevaluación

Para conectarnos con una Base de Datos en Oracle Express debemos usar una conexión con esta descripción:

a) jdbc:oracle:thin:@localhost:1521:XE
b) jdbc:oracle:@localhost:1521:XE
c) jdbc:thin:@localhost:1521:XE
d) jdbc:oracle:thin:@localhost:1521:ORCL

Comprobar

Aplicaciones con conexión a base de datos

Programando una conexión JDBC a la BD (I)

Hemos visto cómo es el API JDBC y hemos accedido a nuestra base de datos desde JDeveloper, pero no hemos realizado un programa JAVA que acceda a la base de datos a través de JDBC.

¿Cómo se hace? En el módulo profesional de Programación en Lenguajes Estructurados se ve cómo hacer que Java acceda a una BD almacenada en MySQL, pero ahora vamos a verlo con Oracle Express. ¿Comenzamos?

En el apartado anterior vimos las clases que debíamos utilizar. El proceso a seguir tiene una **serie de pasos** que vamos a comentar a continuación:

- **Inclusión de la librería java.sql.***
  También podemos incluir otras librerías que vayamos a utilizar con la BD como las extensiones antes comentadas de Oracle (**oracle.jdbc.***).
- **Manejo de Excepciones**
  Debemos manejar las excepciones que pueden producir el driver JDBC o el SGBD accedido, para ello usaremos sentencias **try..catch** en la ejecución de los siguientes pasos.
- **Registro y carga del driver JDBC**
  El registro del driver permite al sistema cargar el driver. En nuestro caso utilizamos JDeveloper con Oracle y el driver ya lo tenemos disponible. Pero si no utilizamos JDeveloper y nuestra base de datos es otra, por ejemplo MySQL, es necesario conseguir el driver JDBC de la Web del fabricante de MySQL e instalarlo en nuestra máquina. Así, para MySQL el driver JDBC podría ser un archivo **jar** del tipo:
  `mysql-connector-java-3.1.12-bin.jar`

La posición de este archivo debe estar en la variable de entorno **CLASSPATH** de nuestra Máquina Virtual Java (JVM). Como el archivo del diver JDBC es una librería externa al JDK (ext) podemos copiar el archivo jar que contiene al driver JDBC en el entorno de ejecución (Java Runtime Environment JRE), por ejemplo en: `C:\Archivos de programa\Java\jdk1.5.0_04\jre\lib\ext` Con ello nos aseguramos que estará disponible, pues el JRE está incluido en el CLASSPATH.

En nuestro caso el archivo que contiene la **librería Oracle JDBC** con los driver thin y OCI se llama `ojdbc14.jar` y podemos encontrarlo en `C:\jdevstudio10132\jdbc\lib` (También podemos usar el archivo **ojdbc14dms.jar** que es igual pero permite usar Oracle Dynamic Monitoring Service). Esta carpeta debe estar en el CLASSPATH de nuestro proyecto. Por suerte, todo este proceso de instalación se encargará de hacerlo JDeveloper por nosotros al añadir la librería JDBC a nuestro proyecto. En la próxima animación lo veremos.

Para registrar el driver en nuestro programa Java debemos conocer la **ruta de la clase OracleDriver**. Si abrimos el archivo `ojdbc14.jar` podremos observar que la ruta de la clase es `oracle.jdbc.driver.OracleDriver()`. Así el registro se realiza usando el método estático registerDriver() de la clase DriverManager del API JDBC de esta manera:


```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

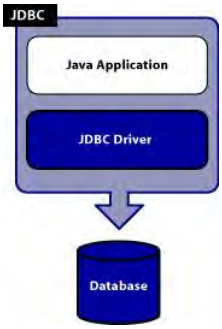
El driver se registra una sóla vez para cada base de datos aunque se realicen múltiples conexiones al mismo servidor de BD. Alternativamente, podemos usar el método `forName()` de la clase `java.lang.Class` de esta manera:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Sin embargo preferimos usar `registerDriver()` a `Class.forName`, porque crea explícitamente una instancia del Driver, lo cual implica

menos restricciones a la JVM.

Programando una conexión JDBC a la BD (II)



Continuamos explicándote los pasos a seguir para programar la conexión JDBC con la BD. Hasta ahora hemos visto que es necesaria la **Inclusión de la librería java.sql.* y que pueden incluirse** otras librerías que vayamos a utilizar con la BD como (**oracle.jdbc.***), es necesario, hacer el **Manejo de Excepciones** y es necesario hacer el **Registro y carga del driver JDBC.**

¿Cuál es el siguiente paso para completar la programación del la conexión?

Seguramente lo habrás pensado: Una vez que todo está listo, el siguiente paso es la conexión a la BD... pero quedan algunos más que te enumeramos en la siguiente lista:

Conexión con la base de datos

Ya tenemos el driver registrado y cargado, por tanto podemos establecer la conexión con la base de datos, lo cual se realiza con el método **`getConnection()`** de la clase **`DriverManager`**. Una llamada a este método crea una instancia del objeto de la clase **`java.sql.Connection class`**. El método tiene 3 parámetros de entrada tipo String:

- el URL de conexión (**driver**, **hostname**, **puerto, SID Oracle**),
- el nombre de usuario y
- su contraseña.

Aunque el método está sobrecargado y podemos usarlo con un único parámetro URL que incluya al usuario y contraseña. En nuestro caso, para acceder a nuestra base de datos del Taller en Oracle Express de forma local usando el driver tipo IV thin con usuario_taller y contraseña 123456, crearemos una instancia del objeto **Connection** llamada "**conn"** de esta forma:

```
Connection conn = DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:XE", "usuario_taller", "123456");
```

Evidentemente, esto lo haremos en la fase de desarrollo, cuando el sistema esté en explotación el usuario y/o la contraseña deberemos pedirla al usuario para mejorar la seguridad.

Definición del acceso a los datos de la BD con SQL

Para hacer una consulta a la BD debemos primero crear un objeto **Statement** que contiene la consulta SQL que después ejecutaremos sobre al BD. Para ello usaremos el método **createStatement**() del objeto **Connection** que creamos antes. La llamada a este método devuelve un objeto (smtp) de la clase **Statement** como se ve a continuación:

```
Statement stmt = conn.createStatement();
```

Ejecución de la sentencia SQL

El siguiente paso es ejecutar la consulta SQL usando el método **executeQuery**() del objeto **Statement** antes definido. Como parámetro de este método usaremos un String que contiene la sentencia SQL a procesar en la BD, el método devuelve un objeto **ResultSet**. Un ejemplo podría ser esta línea de código que ejecuta una consulta sobre la tabla empleados almacenando las filas del resultado en una instancia del objeto **ResultSet** llamada **rset**:

```
ResultSet rset = stmt.executeQuery ("SELECT NOMBRE, TELEFONO FROM EMPLEADOS WHERE IDJEFE=1");
```

Podemos recorrer las filas del resultado con el método rset.next () y recuperar los campos con diferentes métodos dependiendo del tipo de los datos, por ejemplo para recuperar un String usaremos getString(), que requiere como parámetro el nombre de la columna o un número que indica el orden en que aparecía la columna en la consulta SQL. Un ejemplo de esto podría ser:

```
while (rset.next ()) {
System.out.println("Nombre: " + rset.getString( "nombre" ) +  "Teléfono: " + rset.getString("telefono") );
}
```

Liberar los recursos

Es necesario eliminar los objetos creados y **cerrar la conexión con la base de datos**, para ello usaremos el método **close**() de cada uno de los objetos que hemos creado (ResultSet, Statement, Connection), por ejemplo:


```
rset.close();

stmt.close();

conn.close();
```

Aquí tienes un ejemplo completo de un programa Java que accede a la BD del Taller Informático, usando JDBC:

Descarga el programa Java

En la siguiente animación se explica el proceso para probar el programa anterior, usando JDBC, desde JDeveloper:

Uso de JDBC en programa Java para acceder a la BD del taller informático

> ***PARA SABER MÁS***
> ***Es importante conocer bien el funcionamiento de JDBC para que podamos sacarle todo el partido, si tienes dudas en su***

manejo puedes consultar el foro oficial de JDeveloper:
Foros de JDeveloper

Manejo de excepciones al usar JDBC

Como habrás visto en el programa Java del ejemplo anterior, hemos realizado el tratamiento de las excepciones al realizar operaciones de acceso a la BD.

¿Era obligatorio hacerlo? Lo cierto es que debe realizarse un tratamiento explícito de las excepciones al conectarse con una BD, para lo cual debemos usar secciones `try...catch`.

Las excepciones pueden producirse en el driver JDBC o pueden ser provocadas por el propio SGBD Oracle. Cuando se produce un error se lanza una excepción del tipo `java.sql.SQLException` o una subclase de la misma, que podemos tratar mediante los métodos proporcionados con la clase **SQLException**, veámoslos:

- `getErrorCode()`: devuelve el código de error ocurrido al manejar el driver JDBC
- `getMessage()`: devuelve el mensaje de error
- `getSQLState()`: devuelve el código que indica la sentencia SQL que provocó el error en el acceso a la BD
- `printStackTrace()`: utilizado para imprimir la traza del error.

Los objetos **SQLWarning** son una subclase de **SQLException,** trata los avisos que se producen al acceder a una BD. Estos avisos no detienen la ejecución del programa como una excepción, pero alertan de que se ha producido alguna situación comprometida (por ejemplo un error durante una petición de desconexión). Los objetos **Connection**, **Statement** (incluyendo objetos **PreparedStatement** y **CallableStatement**) y **ResultSet** proporcionan un método **getWarnings** que nos devuelve el primer aviso provocado en la llamada al objeto. Si **getWarnings** devuelve un aviso, podemos llamar al método **getNextWarning** de **SQLWarning** para ver avisos adicionales.

A lo largo de las diferentes secciones de este apartado dedicado a JDBC veremos distintos ejemplos de manejo de las excepciones, además a continuación, vemos un **ejemplo de su uso** (debemos realizar `import java.io.*` para manejar `IOException`):

```
try { <JDBC code> }

catch (SQLException e) {

int ret_code = e.getErrorCode();

System.err.println("Oracle Error: "+ ret_code + e.getMessage());}

catch (IOException e) {

System.out.println("Java Error: "+  e.getMessage()); }
```

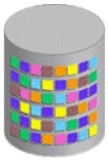
Autoevaluación

getErrorCode() es un método de un objeto...

a) Connection
b) Statement
c) SQLException
d) Ninguna de las respuestas anteriores es correcta

Comprobar

Preparación de consultas SQL



Hemos visto cómo construir una consulta SQL y ejecutarla en la base de datos para obtener los resultados. Sin embargo, cuando queremos realizar la misma consulta muchas veces en diferentes partes de nuestro programa, quizás con parámetros distintos (por ejemplo pasar el idjefe de la consulta anterior como parámetro) el método visto parece engorroso. ¿No hay una solución mejor?

El **API JDBC** nos ofrece otra opción que consiste en preparar la consulta antes de ejecutarla mejorando la eficiencia (ya que es precompilada) y facilitando la tarea (ya que podemos parametrizarla y llamarla varias veces sin necesidad de volver a escribirla). Para ello debemos hacer una llamada al método **prepareStatement** de un objeto **Connection**, que devuelve un objeto **PreparedStatement** que tiene una relación de herencia con el objeto **Statement**, añadiéndole la funcionalidad de poder utilizar parámetros de entrada para la sentencia SQL a ejecutar. Los parámetros se indican con el símbolo de interrogación (**?**)

Un ejemplo que crea el objeto **pstmt** de la clase **PreparedStatement** podría ser el siguiente (se prepara una consulta SQL usando un parámetro para la columna **codprovincia**):

```
PreparedStatement pstmt = conn.prepareStatement ("SELECT nombre FROM clientes WHERE codprovincia = ?" );
```

Bien, hemos preparado la consulta SQL pero ¿cómo le pasamos el parámetro con el **codprovincia**?

Antes vimos el uso del método **getString**(), de forma equivalente existe **setInt**() para asignar los valores a los parámetros de la consulta, debe haber un método **setxxx()** por cada parámetro del **PreparedStatement**. Por ejemplo podríamos escribir la siguiente sentencia que asigna el entero 22 a **codprovincia**, el 1 indica que es el primer parámetro por si apareciesen varios parámetros (símbolo ?):

```
pstmt.setInt(1, 22);
```

Hemos utilizado **métodos getxxx()** y **setxxx()** para recuperar consultas o asignar parámetros respectivamente. ¿Has pensado por qué son necesarios?

Esto está relacionado con la necesidad de convertir los tipos de datos SQL de la BD, ya que son distintos de los tipos de datos de Java, por ello JDBC proporciona este sistema de **mapeo**. Existen múltiples métodos **getxxx**() y **setxxx**() dependiendo del tipo de dato a mapear. Además de **getString**() están disponibles **getBoolean(), getByte(), getDouble(), getFloat(), getInt(), getLong(), getNumeric(), getObject(), getShort(), getDate(), getTime()** o **getUnicodeStream(),** cada uno de los cuales devuelve la columna en el formato correspondiente, si es posible. De forma equivalente podemos usar métodos **setxxx**(). Para los tipos de datos muy grandes se recupera la información con **streams**, por ejemplo para datos almacenados en binario se usan los métodos **setBinaryStream**() y **getBinaryStream**() que no realizan conversión de tipos.

Para cada tipo de dato SQL, se ofrece un mapeo a los tipos correspondientes a tipos estándar de Java, tipos nuevos proporcionados por JDBC (definidos en la clase **java.sql.Types** o en la clase de Oracle **oracle.jdbc.driver.OracleTypes),** y tipos Java proporcionados por las extensiones de Oracle (definidas en **oracle.sql.***), estas cuatro categorías de tipos pueden ser mapeadas entre sí no sólo como tipos básicos sino también por medio de nuevas clases. En esta animación, dedicada a las tablas de mapeo SQL, puedes ver las correspondencias entre estas categorías:

Tablas de mapeo SQL

Como vimos en el ejemplo, a los métodos **getxxx**() le pasamos como parámetro el nombre de la columna o un número que indica el número de columna, empezando por 1 y siguiendo el orden en que aparecen en la sentencia SQL. Se puede combinar la doble indicación de parámetros (nombre columna o número). De igual manera, a los métodos **setxxx**() le pasamos 2 parámetros, el primero es el número de la columna y el segundo el dato a enviar.

Recordemos que cuando obtenemos un **ResultSet** que contiene varias filas, un indicador se posiciona en el primer elemento. Mediante el método **next()** el indicador se situará en la siguiente fila del resultado, o bien sobre el primero si todavía no se ha utilizado. La función **next()** devuelve **true** o **false** dependiendo de si el elemento existe, de manera que puede iterarse en un bucle.

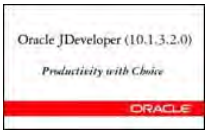
Por otro lado, si queremos llamar a procedimientos PL/SQL almacenados en la BD podemos hacer una llamada al método **prepareCall** de la clase **Connection** que devuelve un objeto **CallableStatement** que tiene una relación de herencia con el objeto PreparedStatement. La ventaja de utilizar funciones o procedimientos PL/SQL es que al estar implementados directamente sobre el SGBD se ejecutan de forma más eficiente.

A continuación te incluimos un **programa Java** que utiliza consultas preparadas JDBC (**PreparedStatement)**. Incluye comentarios que explican su funcionamiento. Para probarlo en JDeveloper crea un nuevo proyecto o añade el archivo (**DBTaller1.java**) como nueva clase Java al proyecto anterior y ejecútalo (recuerda cambiar la contraseña a la que pusiste al crear el usuario):

Descarga el script

> ***PARA SABER MÁS***
>
> ***Aquí tienes un tutorial sobre JDBC donde se describen sus posibilidades desde un punto vista teórico y práctico, con múltiples ejemplos, te aconsejamos una lectura del mismo:***
>
> ***Tutorial JDBC***

Modificación de datos y control de transacciones con JDBC



Hasta ahora hemos visto la ejecución de comandos SQL de consulta de la base de datos, pero desde nuestro programa Java necesitaremos también hacer otras operaciones DML, como actualizaciones o borrado de los datos almacenados.

¿Vemos cómo se hace?

En este apartado vamos a explicar cómo ejecutar comandos INSERT, UPDATE y DELETE. Para ello, utilizaremos los objetos y clases que ya conocemos, es decir necesitamos objetos **Statement** y **PreparedStatement**. La diferencia va a estar en el método que realiza la ejecución sobre la base de datos de manera que si antes usábamos **executeQuery**(), ahora vamos a utilizar **executeUpdate**() que devuelve un entero con el número de filas modificadas (cero si no hay cambios). Veamos un ejemplo de actualización de la tabla **Clientes** suponiendo que **stmt** es un objeto **Statement**:

```
stmt.executeUpdate ("UPDATE CLIENTES SET NOMBRE = 'AGROTEC' WHERE NOMBRE = 'AgroPoni'");
```

Otro ejemplo de actualización con **PreparedStatement** parametrizado usando el objeto pstmt podría ser:

```
pstmt = conn.prepareStatement ("UPDATE CLIENTES SET NOMBRE = ?, TELEFONO= ? WHERE NOMBRE = 'Gestión y Control'");

pstmt.setString (1, "GECO");

pstmt.setString (2, "950555555");

pstmt.executeUpdate ();
```

Si queremos ejecutar otro tipo de sentencias SQL, como por ejemplo sentencias DDL usaremos igualmente **Statement**() pero el método de ejecución es **execute**() al que mandaremos como parámetro un String con el comando SQL que podría ser CREATE TABLE por ejemplo.

A veces ocurre que para reflejar una nueva situación debemos modificar el contenido de varias tablas. Pensemos en nuestro taller, si añadimos una nueva factura debemos actualizar las tablas de **Reparaciones, Facturas, LineasDetalle**, etc, y no tiene sentido que sólo se modifique una tabla porque la factura estaría incompleta. Cuando explicamos el uso de Oracle Express ya planteamos esta cuestión y dijimos que debía resolverse con el control de transacciones. Recordemos que **una transacción es un conjunto de sentencias**

DML que se ejecutan como si fuesen una sola operación. Ahora debemos tener también cuidado con este tema.

¿Cómo se controlan las transacciones desde JDBC?

Cuando trabajamos con JDBC todas las órdenes SQL que enviamos a la BD se ejecutan una a una, es decir, es como si se realizase un **commit**() después de cada sentencia. Pero esto lo podemos modificar para una situación como la descrita antes con las facturas de los clientes. Para ello debemos utilizar el método **setAutoCommit(boolean nuevovalor)** de un objeto **Connection**. En el caso de que se establezca **AutoCommit** a **false**, será necesario llamar de forma explícita al método **commit**() para guardar los cambios realizados o al método **rollback**() para deshacerlo, ambos son métodos también del objeto **Connection**. De esta manera cuando no se ejecuta una de las sentencias de la transacción, todas ellas deben deshacerse, tanto commit como **rollback** afectan a toda la transacción. Cuando cerramos una conexión o realizamos una sentencia DDL se realiza automáticamente un **commit** aunque hayamos anulado el **modo auto-commit**. Unos ejemplos para el objeto **conn** creado anteriormente serían:

- Anulación del modo **auto-commit** (esto mejora también la eficiencia pues nos ahorramos el tiempo de cada uno de los commit):
  `conn.setAutoCommit(false);`
- Forzamos el commit para la transacción:
  `conn.commit();`
- Deshacemos la transacción:
  `conn.rollback();`

Además de manejar las transacciones, el objeto **Connection también proporciona otros métodos que permiten especificar algunas características de la conexión a la BD**. Por ejemplo, el método permite saber si una conexión a una BD es de sólo lectura, **isReadOnly**() y el método **setReadOnly**() sirve para hacerla de sólo lectura. El método **isClosed**() se utiliza para averiguar si una conexión está cerrada o no, y el método **nativeSQL**() permite conocer exactamente la cadena SQL que el driver JDBC envía a la BD al ejecutar el comando SQL especificado, esto puede ser útil en la fase de pruebas.

Para comprobar todo lo que hemos visto con un programa Java, te incluimos a continuación este **archivo DBTaller2.java,** que ejemplifica el uso de DML y transacciones con JDBC que puedes añadir al proyecto anterior o crear uno nuevo, probarlo y ejecutarlo. La intención de este ejemplo es mostrar las posibilidades de JDBC sin preocuparse demasiado por otras cuestiones técnicas o de estilo (recuerda cambiar la contraseña para que coincida con la que pusiste al crear el usuario).

Descarga el programa Java

> ***PARA SABER MÁS***
> ***En Internet hay mucha información sobre la programación Java del driver JDBC, aquí te recomendamos otro tutorial con múltiples ejemplos en español:***
> ***Tutorial práctico de JDBC en español***

Recorrer y modificar los resultados de consultas SQL (I)

Si te has fijado, en las consultas que hemos realizado en los ejemplos anteriores hemos obtenido como resultado un conjunto de filas (**cursor**). Estas filas las hemos recorrido secuencialmente para mostrar los resultados, pero ¿podría interesarnos recorrer estas filas de otra forma? Por ejemplo, podemos mostrar una tabla de resultados y permitir al usuario interactuar gráficamente con la tabla al estilo Windows, o puede ocurrir que nos interese presentar al usuario el resultado organizado por páginas, o pudiera interesarnos ir directamente a la última fila sin tener que pasar por todas las intermedias, en estos casos el recorrido secuencial se muestra insuficiente. ¿Cómo podemos recorrer libremente el cursor de resultados obtenido tras una consulta SQL sobre la BD?

Para ello, debemos utilizar los cursores o conjuntos de resultados con desplazamiento.

¿Cómo maneja JDBC los cursores? Cuando obtenemos como resultado varias filas en un **ResultSet,** el cursor activo será la posición que apunta a la fila actual del **ResultSet**. Inicialmente estamos en una posición anterior a la primera fila (**pseudofila**), por lo que comenzamos usando **next(). Para recorrer el conjunto de filas tenemos una serie de métodos de la clase ResultSet, que te recomendamos que consultes seleccionando el siguiente enlace, aunque también podrías consultarlos en la documentación que sobre la clase ResultSet incluye la propia API de Java.**

Métodos de la clase ResultSet para recorrer el conjunto de filas

¿Podremos recorrer libremente todos los ResultSet?

En realidad debemos elegir cómo debe ser abierto el ResultSet e indicar las posibilidades para recorrerlo.

Para ello se seleccionan distintas posibilidades. Se puede determinar si el acceso a los ResultSet es secuencial o aleatorio. Además, podemos elegir si los datos del ResultSet son sensibles a las modificaciones realizadas en la base de datos por otras transacciones o se mantienen fijos hasta que se cierre el ResultSet. Para ello**, podemos utilizar las siguientes constantes static de la clase ResultSet:**

- `ResultSet.TYPE_FORWARD_ONLY`: el acceso es sólo secuencial hacia delante (sólo podemos usar **next()**). El ResultSet no es afectado por los cambios realizados por otras transacciones en la base de datos.
- `ResultSet.TYPE_SCROLL_INSENSITIVE`: el acceso al ResultSet es aleatorio (podemos usar todos los métodos de recorrido del cursor antes vistos), y el ResultSet tampoco se ve afectado por los cambios realizados por otras transacciones en la base de datos.

- **ResultSet.TYPE_SCROLL_SENSITIVE**: el acceso al ResultSet es aleatorio (podemos usar todos los métodos de recorrido del cursor antes vistos), y el ResultSet refleja la situación real de la base de datos ya que se reflejan los cambios realizados por otras transacciones.

¿Podríamos modificar los datos del ResultSet y trasladar estos cambios a la base de datos? Al igual que ocurre con el acceso directo**, podemos indicar si el ResultSet puede o no modificar la base de datos**. Para ello **utilizaremos estas otras constantes de la clase ResultSet** que pueden combinarse con las 3 anteriores:

- **ResultSet.CONCUR_READ_ONLY**: El ResultSet no puede modificar la base de datos.
- **ResultSet.CONCUR_UPDATABLE**: El ResultSet puede modificar la base de datos.

¿Cómo se utilizan estos métodos y constantes?

En este **ejemplo** vemos cómo preparar el **ResultSet** para que pueda tener accesos directos, sea sensible a las modificaciones en la base de datos y no permita realizar cambios en el ResultSet:

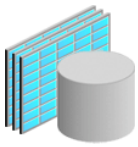
```
conn.createStatement (ResultSet.TYPE_SCROLL_SENSITIVE,   ResultSet.CONCUR_READ_ONLY);
```

En este otro ejemplo utilizamos el acceso directo para hacer un desplazamiento relativo en el **ResultSet** de nombre **rs**, avanzando 3 filas:

```
rs.relative(3);
```


Recorrer y modificar los resultados de consultas SQL (II)

Hemos dicho que **podemos realizar actualizaciones, inserciones y borrados en los ResultSet** preparados con **CONCUR_UPDATABLE**, ¿pero cómo se realiza?



Para ello debemos utilizar los siguientes **métodos de la clase ResultSet:**

- **Actualizaciones**: para asignar los valores a las columnas a modificar usaremos métodos del tipo **updateXXX(columna, valor)** dependiendo del tipo de datos, así usaremos por ejemplo **updateInt(), updateString(), updateFloat(), ...** Como parámetro indicaremos la columna afectada y el nuevo valor asignado. Además usaremos **updateRow()** para que el cambio se refleje en la base de datos.
- **Inserciones**: los valores se asignan con los mismos métodos **updateXXX()**, para insertar la fila en la base de datos usamos **insertRow()** Para asignar una fila debemos posicionar el cursor actual en la fila de inserción con el método **moveToInsertRow()**, para volver a la posición anterior del cursor utilizaremos el método **moveToCurrentRow()**.
- **Borrados**: una vez colocado el cursos en la fila a borrar utilizamos el método **deleteRow()**

Podemos saber qué posibilidades permite nuestro ResultSet utilizando estos **dos métodos de la clase ResultSet:**

- **public int getConcurrency()**: nos devuelve la constante que indica si podemos realizar actualizaciones en el ResultSet o no.
- **public int getType()**: nos devuelve la constante que indica si podemos realizar desplazamientos por el ResultSet y si es sensible a los cambios realizados en la BD.

Hemos visto dos formas de trabajar con la base de datos, o directamente con comandos SQL o a través de los ResultSet, ¿qué ventajas aporta cada método? En general, **el uso directo de SQL es más eficiente, pero el trabajo con ResultSet puede ser más flexible** y a veces imprescindible. Pensemos por ejemplo en un objeto tabla (JTable) en la que el usuario pueda moverse libremente por los datos y modificarlos de forma inmediata, esto obliga a trabajar con ResultSet.

Como en las secciones anteriores, para comprobar lo visto, te incluimos a continuación este **archivo DBTaller3.java,** como ejemplo de trabajo con Resulset**,** que puedes añadir al proyecto anterior o crear uno nuevo, probarlo y ejecutarlo (recuerda cambiar la contraseña para que coincida con la que pusiste al crear el usuario). El programa realiza recorridos por un ResultSet hacia delante y hacia atrás y hace una actualización del ResultSet:

Descarga el programa Java

Autoevaluación

Para posicionarnos en la primera fila con datos de un ResultSet debo utilizar el método:

a) beforeFirst().
b) first().
c) next().
d) isFirst()

Comprobar

Uso de Metadatos

Hasta ahora hemos estado utilizando el acceso a la base de datos suponiendo que siempre conocíamos perfectamente la estructura de la BD, es decir, conocíamos el nombre de las tablas, sus columnas, sus restricciones, etc. Pero esto no siempre es así.

¿Cómo podemos manejar una base de datos de la que no conocemos la estructura de sus tablas?
Para ello, debemos consultar información sobre la propia base de datos, no sobre su contenido. A esta información la llamamos Metadatos (metadata).

¿Y cómo recuperamos los metadatos?

A través de JDBC podemos recoger **metadatos referentes a un ResultSet** (nombre de las columnas, sus tipos, etc.) y **metadatos referentes a una base de datos** (nombre de las tablas, restricciones, etc). A continuación vamos a ver ambos:

Resultset metadata

Los metadatos de un ResultSet se obtienen a través del uso de métodos de la clase **ResultSetMetaData,** como los siguientes:

1. **`getMetaData()`**: Este método opera sobre un objeto **ResultSet** y devuelve un objeto **ResultSetMetaData** tal y como vemos en este ejemplo:

```
ResultSet rset;

ResultSetMetaData rsmd;

rset = pstmt.executeQuery();

rsmd = rset.getMetaData();
```

2. **`getColumnName(int)`**: Una vez obtenido el objeto **ResultSetMetaData** podemos usar este método para obtener el nombre de la columna especificada por el parámetro **int** en el **ResultSet**, un ejemplo puede ser:

```
String col_nombre = rsmd.getColumnName(i);
```

3. **`getColumnType(int)`**: Este método al igual que el anterior opera sobre un objeto **ResultSetMetaData** y devuelve el tipo de dato de la columna del **ResultSet** especificada con el parámetro int. El valor devuelto es un entero de acuerdo con las variables definidas en la clase **java.sql.Types** que ya vimos. Un ejemplo puede ser:

```
int col_tipodato = rsmd.getColumnType(i);

if (col_tipodato = java.sql.Types.VARCHAR)

{...}
```

Database metadata

Podemos consultar las características de una BD mediante los metadatos de la BD (drivers JDBC soportados, nombre de las tablas, restricciones definidas sobre las tablas, etc). Para ello, usaremos los métodos de la clase **DatabaseSetMetaData**, como por ejemplo estos:

1. **`getMetaData()`**: Este método opera sobre un objeto **Connection** y devuelve un objeto **DatabaseMetaData** . Lo podemos ver en este ejemplo:

```
Connection con; DatabaseMetaData dbmd; dbmd = con.getMetaData();
```

<
2. Existen múltiples métodos para obtener metadatos sobre cualquier característica de nuestro SGBD y la BD. Los más comunes pueden ser **`getColumns()`** que devuelve las columnas de una tabla, **`getPrimaryKeys()`** que devuelve la lista de columnas que forman la clave primaria, o el método **`getTables()`** que devuelve la lista de todas las tablas en la base de datos.

Como siempre en este apartado, incluimos un archivo java **DBTaller4.java,** que puedes añadir al proyecto anterior o crear uno nuevo, probarlo y ejecutarlo para comprobar cómo extraer metadatos de nuestra BD (recuerda cambiar la contraseña para que coincida con la que pusiste al crear el usuario).

Descarga el programa Java

PARA SABER MÁS
Disponemos de otros métodos para obtener metadatos, además de los vistos en este apartado. Podemos obtener una completa relación de los mismos en la Web de Sun sobre JDBC que antes se había recomendado:
Web de Sun: JDBC
Oracle ofrece algunas posibilidades añadidas al estándar JDBC. En este interesante artículo en inglés tienes información completa y actualizada sobre el uso del driver JDBC de Oracle, hay múltiples ejemplos que puedes probar para practicar:
Usando JDBC con Oracle

*CASO. En **SI Andalucía** ya tienen la base de datos creada, han realizado la conexión con la base de datos desde JDeveloper y se disponen a aprovechar las tecnologías ofrecidas por Oracle ADF para diseñar una aplicación siguiendo el patrón MVC. **Víctor** ya conoce bien las ventajas de esta arquitectura, pues pudo comprobar la eficacia de los JavaBean para reutilizar clases, también conoce las ventajas de utilizar una arquitectura por capas como MVC, proporcionando seguridad y uso de patrones. Sin embargo ahora se trata de implementar la capa de acceso a la base de datos, es decir el modelo de datos, y de nuevo le entran ciertas dudas. Es **Carmen** otra vez quien intenta tranquilizar a **Víctor** y le comenta que para aproximarse al tema y perder el miedo puede enseñarle en poco tiempo cómo hacer una aplicación Web que acceda a la base de datos con Jdeveloper. **Víctor,** que sabe de la complejidad de la arquitectura Oracle ADF, se muestra completamente extrañado, así que a pesar de haber pagado la coca-cola y la cerveza vuelve a aceptar la apuesta de **Carmen** que en*

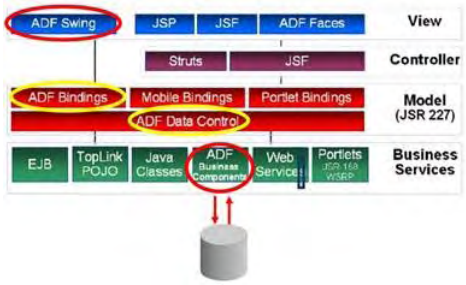
este caso es una cena... ¿quién crees que invitará a cenar?

En el apartado anterior hemos visto cómo podemos programar el acceso a nuestra BD con Java y hemos incluido distintos ejemplos donde se han mostrado las posibilidades del driver JDBC. En los distintos programas Java incluidos nos hemos centrado en la programación del driver, dejando a un lado los aspectos de diseño de una aplicación comercial para no aumentar la complejidad y tamaño de los archivos Java.

Sin embargo, una vez que conocemos el funcionamiento de JDBC, ya estamos en disposición de aprovecharnos de las herramientas y tecnologías que nos ofrece JDeveloper para desarrollar aplicaciones. Por ello, queremos proporcionarte ejemplos en los que sacarás partido a JDeveloper para realizar aplicaciones que utilizan diferentes tecnologías con acceso a BD. Como hemos hecho a lo largo de toda la unidad nos centraremos en dicho acceso a la BD sin detenernos en otras cuestiones, ya que en las próximas unidades completarás la visión del tema y desarrollarás aplicaciones más completas.

Uso de Swing con Oracle ADF

Después de estar trabajando con el acceso a la base de datos del Taller Informática desde Java y estar mostrando los resultados con simples textos, seguro que tienes ganas de hacer un ejemplo con objetos gráficos como paneles y tablas. A continuación vamos a mostrar un ejemplo de una aplicación local que necesita respuesta inmediata, ya que vamos a realizar directamente cambios en la base de datos.



Ya vimos en la unidad anterior la necesidad de utilizar patrones y las ventajas que ofrece la **arquitectura ADF (Application Development Framework) de Oracle basada en J2EE (Java 2 Platform Enterprise Edition),** por ello realizaremos nuestra aplicación siguiendo esta arquitectura con sus diferentes capas. La aplicación la vamos a utilizar de forma local usando el JRE incrustado en JDeveloper por lo que obviaremos la capa ligada al acceso por Internet. ADF permite utilizar distintas tecnologías, en la figura vemos las que vamos a usar:

- Comencemos con la **Vista, vamos a utilizar ADF Swing para diseñar el interface gráfico** de la capa de presentación. ADF Swing conecta los componentes estándar Swing con la capa ADF que desarrolla el modelo de datos.
- En el apartado anterior implementamos la capa de servicios de negocio con una clase Java. Pero ahora, **para la capa de servicios de negocio usaremos los componentes ADF BC (Business Components),** que se encargarán de lograr la persistencia con la base de datos y alojar la lógica de negocio que requiera la aplicación. ADF BC es la actualización de Business Components for Java (BC4J), y permite crear objetos que implementen los servicios de negocio de forma declarativa y sencilla. ADF BC ofrece múltiples ventajas, como que se encarga de la gestión de transacciones, del pooling, los bloqueos o las reglas de validación, permite el mapeo de objetos relacionales, posibilita la escalabilidad, puede usarse con cualquier SGBD o Servidor de Aplicaciones, etc. Con ADF BC podemos definir los Servicios de Negocio una vez y ser utilizados por múltiples aplicaciones distintas sin necesidad de volver a definirlos, y pudiendo modificarlos fácilmente si es necesario.
- Siguiendo la arquitectura ADF separaremos la interfaz de usuario de los servicios de negocio mediante una **capa intermedia (modelo ADF)** que proporciona abstracción con respecto a la capa de servicios de negocio y permitiendo a la vista y al controlador trabajar consistentemente con diferentes implementaciones de la capa de negocio. El modelo ADF utiliza dos tipos de **componentes**: **control de datos (Data Control) y ligadura de datos (Data Binding)** con la GUI, que permiten realizar el mapeo de los componentes de negocio. Estos componentes utilizan archivos de metadatos XML para definir su interfaz, lo que facilita la reutilización y configuración de los componentes de la aplicación.

Cuando utilizamos **ADF BC debemos definir tres tipos de componentes**:

- **Entidades (Entity object):** representa a los datos (filas) almacenados en una tabla de la BD y simplifica la modificación de datos, ya que se encarga de todas las operaciones DML. Además, encapsula la lógica de negocio para asegurar que se cumplen las reglas fijadas. Podemos asociar unas entidades con otras para reflejar las relaciones entre las tablas de la BD que podremos reutilizar en múltiples aplicaciones.
- **Vistas (View object):** representa una consulta SQL simplificando el manejo de los resultados. Podemos utilizar toda la potencia de SQL (join, filtros, órdenes, etc.) para definir estas vistas. Podemos enlazar las vistas con otros objetos para representar las necesidades de la aplicación. Cuando modificamos la vista desde un UI (User Interface), las entidades reflejan los cambios para mantener la consistencia.
- **Módulo de aplicación (Application module):** es un contenedor lógico que proporciona el contexto necesario para definir y ejecutar transacciones, es utilizado por los UI para navegar y modificar los datos. Para ello, en base a las vistas creadas, define un modelo de datos actualizable y los procedimientos y funciones asociados.

Los componentes ADF se definen con un asistente mediante el que podemos definir sus atributos, relaciones y reglas de negocio, pero al igual que el resto de la arquitectura Oracle ADF, **la tecnología ADF BC está implementada en Java (oracle.jbo.*), y utiliza archivos XML para mantener los metadatos de cada componente**. Esto quiere decir, que para cada componente el asistente genera un archivo de metadatos XML (incluye información descriptiva sobre la aplicación) y uno o varios archivos Java (incluyen el código que implementa el comportamiento de la aplicación). Estas clases e interfaces Java se organizan en paquetes, y podemos modificarlas directamente como cualquier archivo Java para adaptarlo a nuestras necesidades. El uso de XML permite mantener la información de forma estructurada fácil de entender y adaptar. Esta información proporciona el contexto adecuado en el que podremos ejecutar nuestras aplicaciones, de esta manera los enlaces (bindings) se realizarán en tiempo de ejecución según la configuración establecida en estos XML. En el siguiente apartado tienes el ejemplo desarrollado.

PARA SABER MÁS

El uso de ADF BC ofrece muchas ventajas como hemos visto, por lo que es muy importante conocer bien su manejo, puedes profundizar en su conocimiento visitando el manual Oracle(r) Application Development Framework Developer's Guide For Forms/4GL Developers 10g Release 3 (10.1.3.0) incluido en la web de Oracle:

Guía ADF BC Oracle

Un ejemplo con Swing ADF

La **aplicación que proponemos de ejemplo trabajará sobre la BD del Taller Informático** y mostrará los datos de la tabla Provincias. Para cada provincia mostrará una subtabla con los datos de los clientes de esa provincia, será posible editar los datos de los clientes y guardarlos en la base de datos.

Los **pasos necesarios para realizar la aplicación** serán los siguientes:

- Crear una **conexión con la BD**, utilizaremos la que hicimos para la BD del Taller Informático.
- Creamos una **nueva aplicación** con dos nuevos proyectos, uno para los **componentes ADF BC,** y otro para la interfaz **ADF Swing**.

A continuación tienes una animación de Ejemplo sobre Swing ADF, que te muestra todo el proceso para llevar a cabo esta aplicación con JDeveloper, (no olvides que aunque aquí manejaremos algunos componentes, en la siguiente unidad profundizarás en el desarrollo con Swing):

Uso de Swing con Oracle ADF para la aplicación del taller informático

Aquí hemos hecho sólo una introducción a esta tecnología. En la próxima unidad "Desarrollo de una aplicación de escritorio con JDeveloper", profundizarás en el tema.

PARA SABER MÁS

En los diferentes ejemplos de uso de ADF van a ir apareciendo archivos XML de metadatos que son generados por JDeveloper, es importante conocer su contenido y funcionamiento, pues a veces puede ser necesario modificarlos para aprovechar todo su potencial, en este enlace tienes una completa descripción de los mismos:

Descripción de los archivos de metadatos ADF **[Versión en caché]**

Uso de JSP con Oracle ADF

En los ejemplos anteriores hemos accedido a la base de datos en modo local, pero podríamos querer acceder a través de Internet. ¿Cómo podemos hacerlo?

En este apartado veremos algunas opciones disponibles y en el siguiente pondremos un ejemplo de acceso a la BD del *Taller Informático* utilizando la tecnología **Java Server Pages (JSP).**

Imaginamos que ya sabes qué son las JSP, pero antes de plantear el ejemplo, quizás merezca la pena recordar brevemente sus antecedentes y posibilidades. Como sabes, las páginas Web realizadas con HTML son estáticas. Cuando se planteó la necesidad de introducir programación en dichas páginas, por ejemplo, para acceder a BD, surgieron los **CGI (Common Gateway Interface)** como solución. Cuando un usuario (cliente) solicita información a través de **formularios HTML**, en el servidor web se ejecuta un **programa CGI escrito en Perl, C++ u otro lenguaje**, que accede a la BD y **genera una página HTML** con los datos solicitados que se muestra en el navegador del usuario. Sin embargo esta solución presenta problemas, porque es complicado hacer programas CGI, no es código portable y satura al servidor porque para cada acceso de un usuario el CGI genera un proceso. Para solucionar estos problemas se propusieron los **servlets Java que aportan portabilidad de plataforma, son más fáciles de usar y reducen la carga del servidor.**

¿Sabes qué es un servlet?

Es un programa Java ejecutado en el servidor (a diferencia de los applets que se ejecutan en el cliente), que recibe peticiones (desde un formulario HTML, un applet, otro servlet, etc.) y manda resultados en HTTP, con el formato necesario (una página HTML, un archivo XML, un objeto tipo MIME, etc.) generando páginas web dinámicas.

Sin embargo, como puedes ver en la imagen, la forma en que el servlet genera la página HTML para el cliente complica el diseño de web vistosas. Cuando en la página HTML generada por el servidor predomina el contenido estático HTML y es importante realizar un buen diseño,

los servlets son insuficientes.

Como respuesta Sun crea las **Java Server Pages (JSP)** como extensión de los servlets.

> **Las páginas JSP son una combinación de página HTML (para definir la estructura estática de la página Web) y programas Java (para desarrollar la lógica de programación del contenido dinámico).**

En realidad, los servlets son objetos Java que para poder utilizarse requieren no sólo un servidor web HTTP (como **Apache**), sino que además, deben correr dentro del contexto de un **contenedor de servlets** proporcionado por un **servidor de aplicaciones** (como **Tomcat**). **Oracle nos proporciona el servidor de aplicaciones Oracle Containers for J2EE (OC4J) junto a JDeveloper**, por lo que será el que utilicemos para nuestros ejemplos.

Con el código Java incluido en la página JSP, el contenedor genera un servlet que se ejecuta para obtener los datos dinámicos (por ejemplo de una BD) y crear la página Web HTML para el cliente. La estructura de JSP permite una arquitectura en capas separando la lógica de presentación de la lógica de negocio.

En la unidad 2 vimos cómo desplegar una aplicación Java con JDeveloper. De igual manera, antes de poder utilizar JSP y servlets, éstos deben ser desplegados (deployment) en el contenedor, lo cual exige crear en el servidor una estructura de directorios predeterminada. Así, se incluye el directorio **WEB-INF** donde se aloja el archivo **web.xml**. Este archivo XML es el **descriptor de despliegue de la aplicación** que contiene la información necesaria para el contenedor de los servlets incluidos en la aplicación web.

Las **páginas JSP,** además de HTML incluyen otro contenido que se forma con **etiquetas**:

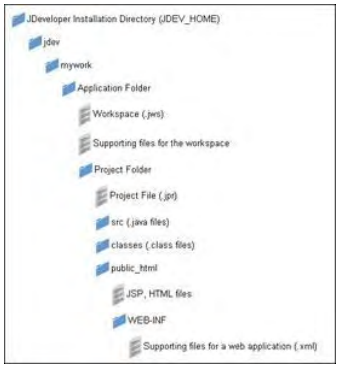
- **Scriptlet**: es el código Java incluido en páginas JSP, se indica encerrado por los caracteres **<%** y **%>**
- **Directivas**: transmiten al contenedor la configuración adecuada, se indican entre **<%@** y **%>**
- **Taglib**: además de las etiquetas básicas podemos usar otras de las librerías de etiquetas (**taglib**)
- **Acciones**: permiten realizar una tarea en el momento en que se solicita la página, indicadas con **&ltjs**

> ***PARA SABER MÁS***
>
> ***En esta página tienes diferentes tutoriales que te recomendamos para que los consultes durante la lectura del resto de esta unidad, incluye información en castellano sobre servlets, JSP, JDBC, JSF, EJB (en los próximos apartados explicaremos estos conceptos), también contiene múltiples ejemplos:***
>
> ***Tutoriales de servlet, JSP, JSF, EJB, etc***

Aplicaciones con conexión a base de datos

Ejemplos de acceso a BD usando JSP

Después de la introducción anterior ya tendrás ganas de probar las páginas JSP, empezaremos por poner un pequeño ejemplo para realizar una sencilla página JSP "*a mano*" que acceda a la BD del Taller informático y después generaremos otra página JSP con ayuda de JDeveloper.



Vamos a crear una pequeña aplicación que muestre en una página web una tabla con los datos de los clientes de la BD del Taller Informático. Para **ello vamos a crear una página JSP con JDeveloper**, esta página **incluirá código Java para realizar el acceso a la BD con JDBC**, el código Java será igual que el utilizado en los apartados anteriores. A continuación tienes el código de la página JSP donde puedes ver el uso de las etiquetas antes explicado:

Descarga el código JSP

Para ayudarte a crear este archivo JSP en JDeveloper te incluimos esta animación, JDeveloper se encargará del **despliegue de nuestro JSP en el servidor de aplicaciones OC4J** que tiene embebido:

Ejemplo de página JSP para acceder a la BD del taller informático

Hemos utilizado una página JSP con el driver JDBC para acceder a la BD, pero **podríamos usar los componentes ADF BC que desarrollamos en un ejemplo anterior para crear una página JSP**, en la siguiente animación se te explica cómo hacerlo:

Uso de JSP y Oracle ADF para una aplicación web del taller informático

JSP ofrece una tecnología potente y sencilla para realizar páginas web dinámicas con un diseño flexible y atractivo, por ello, aunque aquí sólo hemos hecho una pequeña introducción, profundizarás más en su conocimiento en las unidades de "**Generación de aplicaciones para la web**".

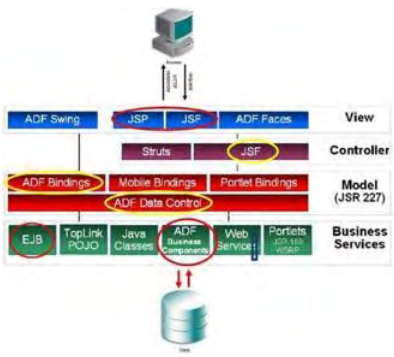
Autoevaluación

En la arquitectura ADF Oracle utilizamos JSP en la capa de...

a) Servicios de negocio.
b) Vista.
c) Controlador.
d) Modelo.

Comprobar

Las tecnologías JSF y EJB de ADF Oracle



Hemos visto diferentes tecnologías que se utilizan en el desarrollo de programas que manejan BD, tanto en aplicaciones de escritorio como en aplicaciones web. En la unidad anterior estudiamos las posibilidades del uso de patrones como MVC, así como de JSF dentro del Oracle ADF (Application Development Framework). Como ya sabes**, JSF nos ofrece un *framework* con arquitectura MVC** que facilita el desarrollo de aplicaciones con componentes reutilizables, mapeo de componentes, fácil desarrollo de la lógica de navegación entre páginas, etc. Estas tecnologías están pensadas para enfrentarse a los problemas asociados a las aplicaciones de gran tamaño.

También en esta unidad hemos visto el uso de **ADF BC que nos aporta una solución para la capa de servicios de negocio**, pero cuando hablamos de grandes proyectos para grandes corporaciones donde debemos acceder al mismo tiempo a varias bases de datos que pueden estar distribuidas en diferentes servidores, así como a sistemas de información para empresas como ERP, nos enfrentamos a nuevos problemas que deben ser resueltos con tecnologías potentes. ¿Qué nuevas tecnologías nos permite utilizar Oracle ADF?

Los **Enterprise JavaBean (EJB)** surgen como una eficaz alternativa, ya que están especialmente pensados para encapsular la lógica de negocio de **sistemas distribuidos**, de tal forma que el desarrollador no tenga que preocuparse por la programación a nivel de sistema (como **control de transacciones**, **concurrencia, seguridad, persistencia, acceso remoto de objetos,** etc.), sino que se centre en la representación de entidades y reglas de negocio. Además los EJB son reutilizables por lo que en nuestros desarrollos podremos usar EJB producidos por terceros (como suministradores de sistemas ERP). No hay que confundir los Enterprise JavaBeans con los JavaBeans que vimos en la unidad 2. Los JavaBeans también son un modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones, pero no pueden utilizarse en entornos de objetos distribuidos al no soportar nativamente la invocación remota RMI (*Java Remote Method Invocation*).

Los EJB son una API (**Application Programming Interface** - Interfaz de Programación de Aplicaciones) que forman parte del estándar **J2EE de Sun Microsystems**, y se definen como clases Java que mapean el modelo de persistencia sobre la BD utilizando **anotaciones** para los metadatos, de esta forma el modelo de datos queda integrado en clases Java. Para ejecutarse los EJB, deben ser desplegados en un **contenedor de EJB.** Los contenedores son el contexto donde se ejecutan los EJB, y el gestor de los contenedores es el Servidor de aplicaciones (como OC4J o **JBoss**).

A lo largo de este módulo iremos ampliando los contenidos sobre ADF y los servidores de aplicaciones que te permitirán entender mejor el contexto en que se ejecutan los EJB, por ello la última unidad de este módulo (**Proyecto Globalizador**) será el momento de volver a los EJB y profundizar en su conocimiento.

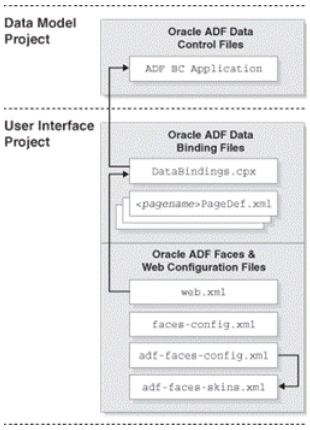
Autoevaluación

En la arquitectura ADF Oracle utilizamos EJB en la capa de...

a) Servicios de negocio.
b) Vista.
c) Controlador.
d) Modelo.

Comprobar

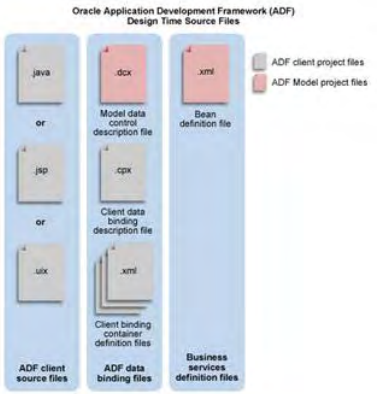
Un ejemplo de acceso a la BD con ADF BC y JSF



Vamos a sacar partido a la tecnología ADF que vimos en la unidad anterior, a partir del proyecto desarrollado con ADF BC para acceder a la BD del Taller Informático, vamos a presentar un ejemplo con una pequeña **aplicación Web con JSP y JSF** que nos permitirá acceder a la tabla de clientes y poder modificarla.

Durante el desarrollo del ejemplo veremos cómo se crean diferentes **archivos XML de metadatos**, en las imágenes podemos ver la relación jerárquica entre estos archivos cuando se usan ADF BC en la arquitectura ADF Oracle. Aunque ya conocemos la mayoría de estos archivos vamos a hacer una pequeña descripción de los mismos:

1. **Archivos de Control de datos (data control):** pertenecen a un proyecto de Modelo de datos, el archivo **DataControls.dcx** contiene información necesaria para realizar el control de datos en tiempo de ejecución al utilizar servicios como EJB, ADF BC, etc.
2. **Archivos de Enlace de datos (data binding):** aparecen en el proyecto de interface de usuario:
   a. El archivo **DataBindings.cpx** proporciona la información necesaria para crear los enlaces de datos de la BD con componentes de la interface de usuario.
   b. Además se crean para cada página web archivos **<pagename>PageDef.xml** que permiten realizar los enlaces de datos para cada componente UI de la página web.
3. **Archivos de configuración para ADF Faces:** aparecen en el proyecto de interface de usuario y se componen de:
4. **web.xml:** se incluye en cualquier aplicación J2EE para definir la información necesaria para realizar el despliegue de la aplicación en el servidor.
5. **faces-config.xml:** es un archivo de configuración ADF que permite registrar los recursos JSF como validadores, conversores, bean manejados o reglas de navegación.
6. **adf-faces-config.xml:** es otro archivo de configuración ADF que permite indicar características del UI como niveles de accesibilidad, zona horaria, la apariencia (skin), etc.



A continuación te mostramos una animación de ejemplo de uso de JSF con ADF BC, donde puedes ver el proceso para utilizar los componentes ADF BC que desarrollamos en un ejemplo anterior junto a la tecnología ADF Faces para crear una aplicación Web que permite mostrar los datos de los clientes y editarlos:

Uso de JSP y Oracle para una aplicación web del taller informático

Con este ejemplo **concluimos la visión general que hemos querido dar a la realización de accesos a Bases de Datos desde las aplicaciones,** para ello hemos utilizado diferentes tecnologías profundizando en el manejo y programación del driver JDBC. En la unidad hemos puesto distintos ejemplos en los que nos hemos centrado en la conexión con la BD, a lo largo del módulo utilizaremos todo lo visto en esta unidad y realizaremos más ejemplos en los que desarrollaremos otros aspectos de la aplicación.

Autoevaluación

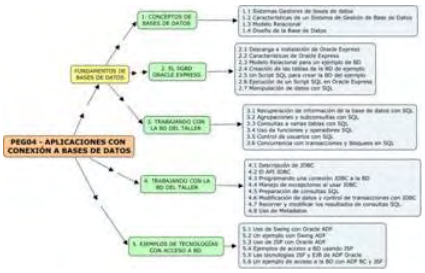
En la arquitectura ADF Oracle utilizamos ADF BC en la capa...

a) Servicios de negocio.
b) Vista.
c) Controlador.
d) Modelo.

Comprobar

A continuación te representamos de forma gráfica, en una sola imagen, un resumen de los conceptos importantes tratados en la unidad.

Haz clic sobre la siguiente imagen para verla a tamaño de pantalla completa.



Aplicaciones con conexión a base de datos