



SQL-PL-SQL Y ADMINISTRACIÓN BÁSICA DE ORACLE

Akademo Consulting, S.L.

C/ Ginés de la Neta, 1

28100 FLESA (MADRID)

SQL-PL-SQL Y ADMINISTRACIÓN BÁSICA DE ORACLE

ÍNDICE DE CONTENIDO

Introducción a las bases de datos	Zielstelle nicht gefunden!
Definición del término base de datos.....	Zielstelle nicht gefunden!
Tipos de bases de datos	Zielstelle nicht gefunden!
Ley Orgánica de Protección de Datos de Carácter Personal de España.	Zielstelle nicht gefunden!
Formas de obtención de los datos, tipos de datos y cancelación.....	Zielstelle nicht gefunden!
Introducción a S.Q.L.	Zielstelle nicht gefunden!
Características de S.Q.L.	Zielstelle nicht gefunden!
Lenguaje de definición de datos (DDL).....	Zielstelle nicht gefunden!
Lenguaje de manipulación de datos (DML)	Zielstelle nicht gefunden!
Base de datos Oracle.....	Zielstelle nicht gefunden!
Historia	Zielstelle nicht gefunden!
Arquitectura de Oracle.....	Zielstelle nicht gefunden!
Lenguaje SQL.....	Zielstelle nicht gefunden!
SqlPlus	Zielstelle nicht gefunden!
Versión gráfica de SQL*Plus	Zielstelle nicht gefunden!
iSQL*Plus.....	Zielstelle nicht gefunden!
Otras Aplicaciones de desarrollo SQL.....	Zielstelle nicht gefunden!
Esquemas de ejemplo de Oracle	Zielstelle nicht gefunden!
Esquema SCOTT	Zielstelle nicht gefunden!
Esquema HR	Zielstelle nicht gefunden!
Sentencias DDL(Lenguaje de definición de datos)	Zielstelle nicht gefunden!
Sentencias DCL (Lenguaje de control de datos)	Zielstelle nicht gefunden!
Sentencias DML(Lenguaje de manipulación de datos) TCL (Control de transacciones)	Zielstelle nicht gefunden!
Funciones estandar	Zielstelle nicht gefunden!
Insert	Zielstelle nicht gefunden!
Update	Zielstelle nicht gefunden!
Delete.....	Zielstelle nicht gefunden!
Commit.....	Zielstelle nicht gefunden!

Rollback.....	Zielstelle nicht gefunden!
Savepoint	Zielstelle nicht gefunden!
Transacciones.....	Zielstelle nicht gefunden!
Set transaction	Zielstelle nicht gefunden!
Administración básica de Oracle	Zielstelle nicht gefunden!
Instalacion de Oracle 10G Standard/Enterprise Edition	Zielstelle nicht gefunden!
Instalar Oracle Database 10g XE Express Edition en Windows XP	Zielstelle nicht gefunden!
Cómo instalar oracle client en windows (cliente de oracle)	Zielstelle nicht gefunden!
Tablas del diccionario de datos:	Zielstelle nicht gefunden!
Estructura de ficheros de Oracle	Zielstelle nicht gefunden!
La instancia oracle	Zielstelle nicht gefunden!
Estructura de la memoria de oracle	Zielstelle nicht gefunden!
Procesos de Una Instancia de Oracle	Zielstelle nicht gefunden!
Listener	Zielstelle nicht gefunden!
Actualización de los parámetros del sistema	Zielstelle nicht gefunden!
Control de sesiones (kill session).....	Zielstelle nicht gefunden!
Creación y administración de tablespaces	Zielstelle nicht gefunden!
Copia de seguridad de una base de datos Oracle (rman)	Zielstelle nicht gefunden!
Export / import.....	Zielstelle nicht gefunden!
PL-SQL	Zielstelle nicht gefunden!
Identificadores	Zielstelle nicht gefunden!
Operadores	Zielstelle nicht gefunden!
Variables.....	Zielstelle nicht gefunden!
Constantes	Zielstelle nicht gefunden!
Funciones integradas de PL/SQL	Zielstelle nicht gefunden!
Bloque PL/SQL.....	Zielstelle nicht gefunden!
Excepciones en PL/SQL.....	Zielstelle nicht gefunden!
Cursores en PL/SQL	Zielstelle nicht gefunden!
Cursores Implícitos en PL/SQL.....	Zielstelle nicht gefunden!
Cursores Explícitos en PL/SQL	Zielstelle nicht gefunden!
Cursores de actualización en PL/SQL.....	Zielstelle nicht gefunden!
Bloques Anonimos (sin nombre)	Zielstelle nicht gefunden!
Subprogramas	Zielstelle nicht gefunden!
Registros PL/SQL	Zielstelle nicht gefunden!

Tablas PL/SQL	Zielstelle nicht gefunden!
Varrays	Zielstelle nicht gefunden!
Bulk collect	Zielstelle nicht gefunden!
SQL Dinamico	Zielstelle nicht gefunden!
PL/SQL y Java.....	Zielstelle nicht gefunden!
Buenas prácticas trabajando con PL-SQL	Zielstelle nicht gefunden!
Optimización de SQL-PL-SQL Trazas – tkprof	Zielstelle nicht gefunden!
Sql loader	Zielstelle nicht gefunden!

Introducción a las bases de datos

Definición del término base de datos

Definición de Bases de Datos.- Un conjunto de información almacenada en memoria auxiliar que permite acceso directo y un conjunto de programas que manipulan esos datos

Base de Datos es un conjunto exhaustivo no redundante de datos estructurados organizados independientemente de su utilización y su implementación en máquina accesibles en tiempo real y compatibles con usuarios concurrentes con necesidad de información diferente y no predicable en tiempo.

Tipos de bases de datos

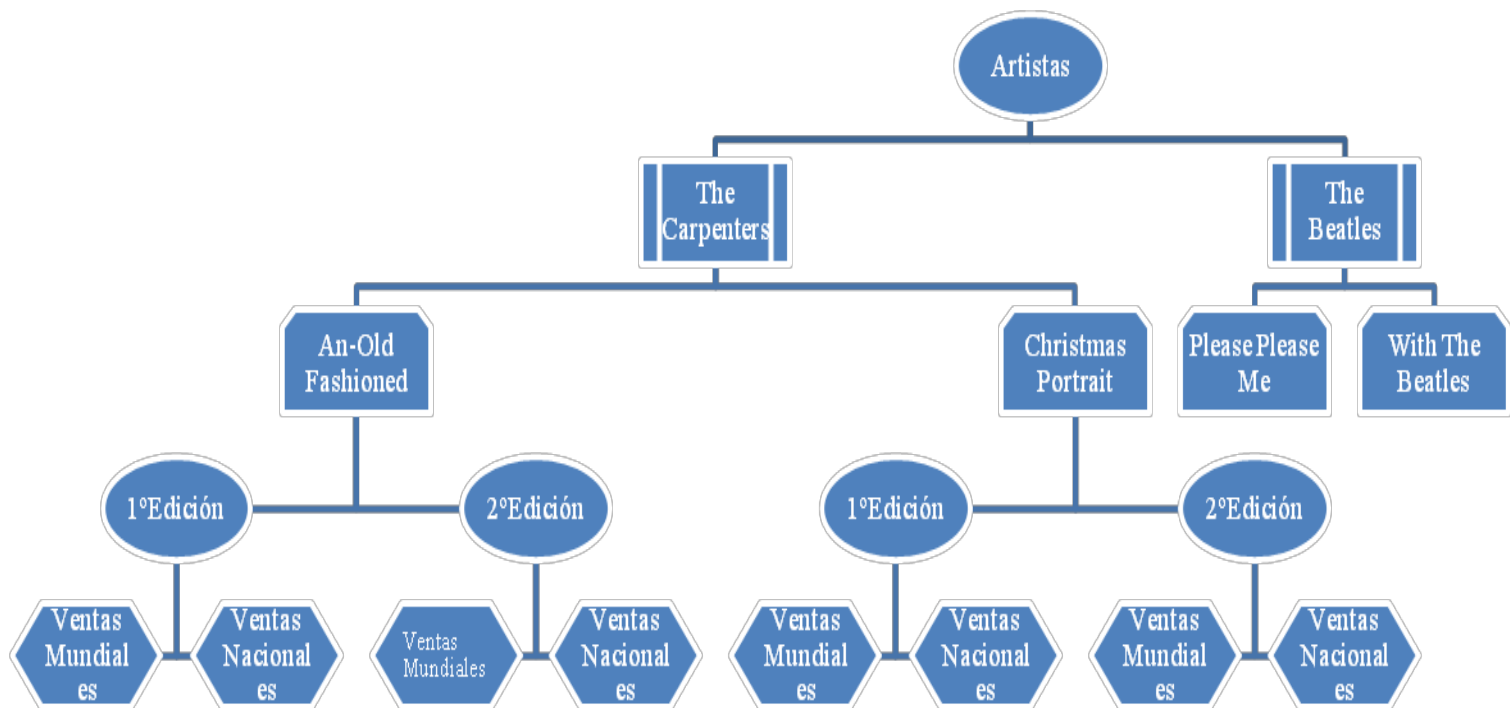
Desde el punto de vista de organización lógica

Bases de datos jerárquicas

Éstas son bases de datos que, como su nombre indica, almacenan su información en una estructura jerárquica. En este modelo los datos se organizan en una forma similar a un árbol (visto al revés), en donde un *nodo padre* de información puede tener varios *hijos*. El nodo que no tiene padres es llamado *raíz*, y a los nodos que no tienen hijos se los conoce como *hojas*.

Las bases de datos jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de información y datos muy compartidos permitiendo crear estructuras estables y de gran rendimiento.

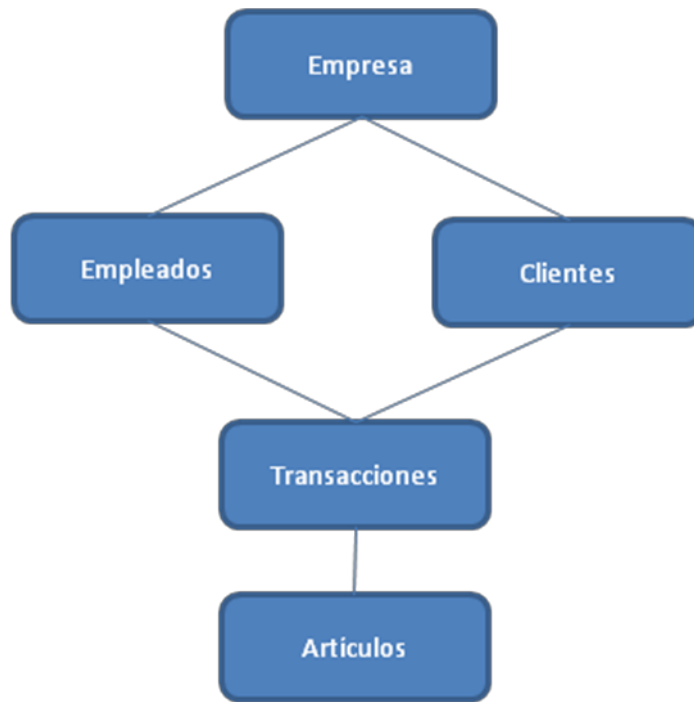
Una de las principales limitaciones de este modelo es su incapacidad de representar eficientemente la redundancia de datos.



Base de datos de red

Éste es un modelo ligeramente distinto del jerárquico; su diferencia fundamental es la modificación del concepto de *nodo*: se permite que un mismo nodo tenga varios padres (posibilidad no permitida en el modelo jerárquico).

Fue una gran mejora con respecto al modelo jerárquico, ya que ofrecía una solución eficiente al problema de redundancia de datos; pero, aun así, la dificultad que significa administrar la información en una base de datos de red ha significado que sea un modelo utilizado en su mayoría por programadores más que por usuarios finales.



Base de datos relacional

Éste es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Tras ser postulados sus fundamentos en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos. Su idea fundamental es el uso de "relaciones". Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "tuplas". Pese a que ésta es la teoría de las bases de datos relacionales creadas por Edgar Frank Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar. Esto es pensando en cada relación como si fuese una tabla que está compuesta por registros (las filas de una tabla), que representarían las tuplas, y campos (las columnas de una tabla).

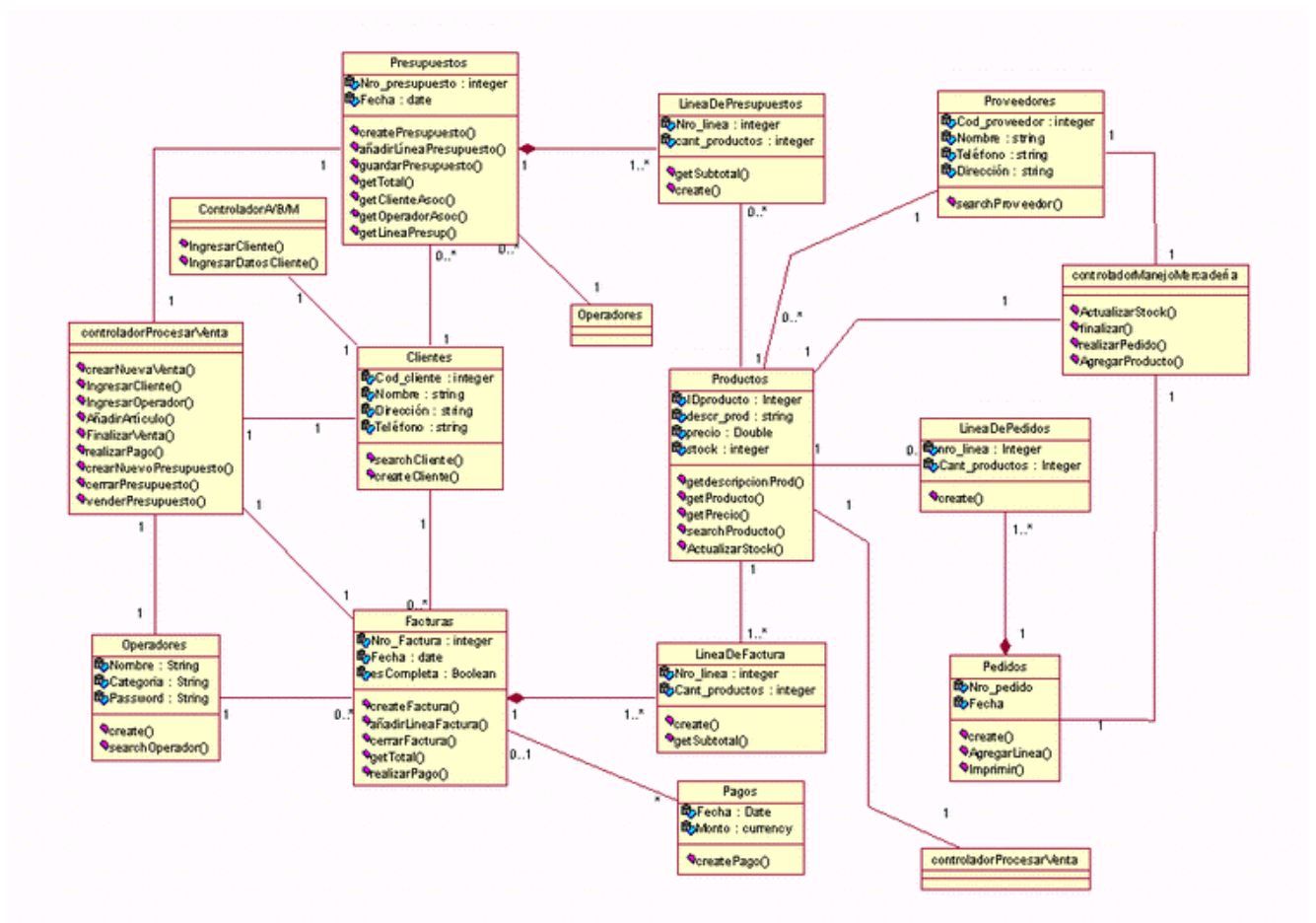
En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario esporádico de la base de datos. La información puede ser recuperada o almacenada mediante "consultas" que ofrecen una amplia flexibilidad y poder para administrar la información.

El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, Structured Query Language o Lenguaje Estructurado de

Consultas, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

Durante su diseño, una base de datos relacional pasa por un proceso al que se le conoce como normalización de una base de datos.

Durante los años '80 (1980-1989) la aparición de dBASE produjo una revolución en los lenguajes de programación y sistemas de administración de datos. Aunque nunca debe olvidarse que dBase no utilizaba SQL como lenguaje base para su gestión.



Bases de datos multidimensionales

Son bases de datos ideadas para desarrollar aplicaciones muy concretas, como creación de [Cubos OLAP](#). Básicamente no se diferencian demasiado de las bases de datos relacionales (una tabla en una base de datos multidimensional podría serlo también en una base de datos multidimensional), la diferencia está más bien a nivel conceptual; en las bases de datos multidimensionales los

campos o atributos de una tabla pueden ser de dos tipos, o bien representan dimensiones de la tabla, o bien representan métricas que se desean estudiar.



Bases de datos orientadas a objetos

Este modelo, bastante reciente, y propio de los modelos informáticos orientados a objetos, trata de almacenar en la base de datos los objetos completos (estado y comportamiento).

Una base de datos orientada a objetos es una base de datos que incorpora todos los conceptos importantes del paradigma de objetos:

Encapsulación - Propiedad que permite ocultar la información al resto de los objetos, impidiendo así accesos incorrectos o conflictos.

Herencia - Propiedad a través de la cual los objetos heredan comportamiento dentro de una jerarquía de clases.

Polimorfismo - Propiedad de una operación mediante la cual puede ser aplicada a distintos tipos de objetos.

En bases de datos orientadas a objetos, los usuarios pueden definir operaciones sobre los datos como parte de la definición de la base de datos. Una operación

(llamada función) se especifica en dos partes. La interfaz (o signatura) de una operación incluye el nombre de la operación y los tipos de datos de sus argumentos (o parámetros). La implementación (o método) de la operación se especifica separadamente y puede modificarse sin afectar la interfaz. Los programas de aplicación de los usuarios pueden operar sobre los datos invocando a dichas operaciones a través de sus nombres y argumentos, sea cual sea la forma en la que se han implementado. Esto podría denominarse independencia entre programas y operaciones.

Se está trabajando en [SQL3](#), que es el estándar de SQL92 ampliado, que soportará los nuevos conceptos orientados a objetos y mantendría compatibilidad con SQL92.

Bases de datos documentales

Permiten la indexación a texto completo, y en líneas generales realizar búsquedas más potentes. Tesauros es un sistema de índices optimizado para este tipo de bases de datos.

Desde el punto de vista de números de usuarios:

- Mono usuarios
- Multiusuarios

Ley Orgánica de Protección de Datos de Carácter Personal de España

La Ley Orgánica 15/1999 de 13 de diciembre de Protección de Datos de Carácter Personal, abreviada como LOPD, es una Ley Orgánica española que tiene por objeto garantizar y proteger, en lo que concierne al tratamiento de los datos personales, las libertades públicas y los derechos fundamentales de las personas físicas, y especialmente de su honor, intimidad y privacidad personal y familiar.

Su objetivo principal es regular el tratamiento de los datos y ficheros, de carácter personal, independientemente del soporte en el cual sean tratados, los derechos

de los ciudadanos sobre ellos y las obligaciones de aquellos que los crean o tratan.

El Real Decreto 994/1999 de Medidas de Seguridad de los ficheros automatizados que contengan datos de carácter personal de 11 de Junio de 1999 (RMS) : Es un reglamento que desarrolla la Ley Orgánica 5/1992, de 29 de octubre, de Regulación del Tratamiento Automatizado de los Datos de Carácter Personal (LORTAD), regula las medidas técnicas y organizativas que deben aplicarse a los sistemas de información en los cuales se traten datos de carácter personal de forma automatizada.

El órgano de control del cumplimiento de la normativa de protección de datos dentro del territorio español, con carácter general es la Agencia Española de Protección de Datos (AEPD), existiendo otras Agencias de Protección de Datos de carácter autonómico, en las Comunidades Autónomas de Madrid, Cataluña y en el País Vasco.

Las sanciones tienen una elevada cuantía, siendo España el país de la Unión Europea que tiene las sanciones más altas en materia de protección de datos. Dichas sanciones dependen de la infracción cometida.

Se dividen en:

- Las sanciones leves van desde 601,01 a 60.101,21 €
- Las sanciones graves van desde 60.101,21 a 300.506,05 €
- Las sanciones muy graves van desde 300.506,05 a 601.012,10 €

Pese al elevado importe de las sanciones, existen muchas empresas en España que todavía no se han adecuado a la misma, o lo han hecho de forma parcial o no revisan de forma periódica su adecuación; por lo que resulta esencial el mantenimiento y revisión de la adecuación realizada.

En el sector público, la citada Ley regula igualmente el uso y manejo de la información y los ficheros con datos de carácter personal utilizados por todas las administraciones públicas.

La Agencia Española de Protección de Datos (AEPD) fue creada en 1994

conforme a lo establecido en la LOPD. Su sede se encuentra en Madrid, si bien dispone de varias agencias autonómicas.

Formas de obtención de los datos, tipos de datos y cancelación

Los datos personales se clasifican en función de su mayor o menor grado de sensibilidad, siendo los requisitos legales y de medidas de seguridad informáticas más estrictos en función de dicho mayor grado de sensibilidad, siendo obligatorio por otro lado, en todo caso la declaración de los ficheros de protección de datos a la "Agencia Española de Protección de Datos".

El tratamiento de los datos de carácter personal requerirá el consentimiento inequívoco del afectado, salvo que la ley disponga otra cosa.

- Se requiere consentimiento expreso y por escrito del afectado respecto a los datos relativos a la **ideología, afiliación sindical, religión y creencias** y sólo podrán ser cedidos con consentimiento expreso.
- Los relativos a **infracciones** penales o administrativas sólo podrán ser incluidos en ficheros de las **Administraciones públicas** competentes.
- Se permite la obtención de otros datos de carácter personal, **sin el consentimiento del afectado** pero éste deberá ser informado de forma expresa, precisa e inequívoca, por el responsable del fichero o su representante, dentro de los **tres meses** siguientes al momento del registro de los datos.

Esta podría ser una **cláusula modelo** de información/consentimiento de derechos amparados por la LOPD:

En cumplimiento de la Ley Orgánica 15/1999, de 13 de diciembre de Protección de Datos de Carácter Personal (LOPD), (sustituir por el nombre del responsable del fichero), como responsable del fichero informa de las siguientes consideraciones:

Los datos de carácter personal que le solicitamos, quedarán incorporados a un fichero cuya finalidad es (describir la finalidad).

Queda igualmente informado de la posibilidad de ejercitar los derechos de acceso, rectificación, cancelación y oposición, de sus datos personales en (sustituir por el domicilio para ejercitar los derechos).

- No será necesaria la comunicación en tres meses de dicha información si los datos han sido recogidos de "**fuentes accesibles al público**",^[6] y se destinan a la actividad de **publicidad** o prospección **comercial**, en este caso "en cada comunicación que se dirija al interesado se le informará del origen de los datos y de la identidad del responsable del tratamiento así como de los derechos que le asisten".

Los datos deben ser destruidos o devueltos al responsable del tratamiento "*una vez cumplida la prestación contractual*".

Los interesados que "sufran daño o lesión en sus bienes o derechos" por incumplimiento de ésta ley "tendrán derecho a ser indemnizados".

Introducción a S.Q.L.

El **Lenguaje de consulta estructurado** (En inglés] Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar -de una forma sencilla- información de interés de una base de datos, así como también hacer cambios sobre la misma.

El SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales permitiendo gran variedad de operaciones sobre los mismos.

Características de S.Q.L.

Lenguaje de definición de datos (DDL)

El lenguaje de definición de datos (en inglés *Data Definition Language*, o *DDL*), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Existen cuatro operaciones básicas: CREATE, ALTER, DROP y TRUNCATE.

CREATE

Este comando crea un objeto dentro de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte.

Ejemplo (crear una tabla)

```
CREATE TABLE 'TABLA_NOMBRE' (  
    'CAMPO_1' INT,  
    'CAMPO_2' STRING  
)  
  
CREATE TABLE 'TABLA_NOMBRE' (  
    'CAMPO_1' APELLIDOP,  
    'CAMPO_2' NOMBRE  
    EDAD)
```

ALTER

Este comando permite modificar la estructura de un objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, modificar un trigger, etc.

Ejemplo (agregar columna a una tabla)

```
ALTER TABLE "TABLA_NOMBRE" (  
    ADD NUEVO_CAMPO INT UNSIGNED  
)
```

DROP

Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte. Se puede combinar con la sentencia ALTER.

Ejemplo

```
ALTER TABLE "TABLA_NOMBRE"  
(
```

```
DROP COLUMN "CAMPO_NOMBRE1"  
  
)
```

TRUNCATE

Este comando trunca todo el contenido de una tabla. La ventaja sobre el comando DELETE, es que si se quiere borrar todo el contenido de la tabla, es mucho más rápido, especialmente si la tabla es muy grande, la desventaja es que TRUNCATE solo sirve cuando se quiere eliminar absolutamente todos los registros, ya que no se permite la cláusula WHERE. Si bien, en un principio, esta sentencia parecería ser DML (Lenguaje de Manipulación de Datos), es en realidad una DDL, ya que internamente, el comando truncate borra la tabla y la vuelve a crear y no ejecuta ninguna transacción.

Ejemplo

```
TRUNCATE TABLE "TABLA_NOMBRE1"
```

Lenguaje de manipulación de datos (DML)

Un lenguaje de manipulación de datos (*Data Manipulation Language*, o *DML* en inglés) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios de la misma llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.

El lenguaje de manipulación de datos más popular hoy día es SQL, usado para recuperar y manipular datos en una base de datos relacional. Otros ejemplos de DML son los usados por bases de datos IMS/DL1, CODASYL u otras.

INSERT

Una sentencia *INSERT* de **SQL** agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

Forma básica

```
INSERT INTO "tabla" ("columna1", ["columna2,... "]) VALUES  
("valor1", ["valor2,..."])
```

Las cantidades de columnas y valores deben ser las mismas. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados (o implícitos) por la sentencia INSERT deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.

Ejemplo

```
INSERT INTO agenda_telefonica (nombre, numero) VALUES  
( 'Roberto Jeldrez', '4886850');
```

Cuando se especifican todos los valores de una tabla, se puede utilizar la sentencia acortada:

```
INSERT INTO "tabla" VALUES ("valor1", ["valor2,..."])
```

Ejemplo (asumiendo que 'nombre' y 'número' son las únicas columnas de la tabla 'agenda_telefonica'):

```
INSERT INTO agenda_telefonica VALUES ('Roberto Jeldrez',  
'4886850');
```

Formas avanzadas

Inserciones en múltiples filas

Una característica de SQL (desde SQL-92) es el uso de *constructores de filas* para insertar múltiples filas a la vez, con una sola sentencia SQL:

```
INSERT INTO "tabla" ("columna1", ["columna2,... "])  
  
VALUES ("valor1a", ["valor1b,..."]),  
( "value2a", ["value2b,..."]),...
```

Ejemplo (asumiendo ese 'nombre' y 'número' son las únicas columnas en la tabla 'agenda_telefonica'):

```
INSERT INTO agenda_telefonica VALUES ('Roberto Fernández',  
'4886850'), ('Alejandro Sosa', '4556550');
```

Que podía haber sido realizado por las sentencias

```
INSERT INTO agenda_telefonica VALUES ('Roberto Fernández',  
'4886850');
```



```
INSERT INTO agenda_telefonica VALUES ('Alejandro Sosa',  
'4556550');
```

Notar que las sentencias separadas pueden tener semántica diferente (especialmente con respecto a los triggers), y puede tener diferente rendimiento que la sentencia de inserción múltiple.

Para insertar varias filas en MS SQL puede utilizar esa construcción:

```
INSERT INTO phone_book  
  
SELECT 'John Doe', '555-1212'  
  
UNION ALL  
  
SELECT 'Peter Doe', '555-2323';
```

Tenga en cuenta que no se trata de una sentencia SQL válida de acuerdo con el estándar SQL (SQL: 2003), debido a la cláusula subselect incompleta.

Para hacer lo mismo en Oracle se usa DUAL TABLE, siempre que se trate de solo una simple fila:

```
INSERT INTO phone_book  
  
SELECT 'John Doe', '555-1212' FROM DUAL  
  
UNION ALL  
  
SELECT 'Peter Doe', '555-2323' FROM DUAL
```

Un estándar-conforme implementación de esta lógica se muestra el siguiente ejemplo, o como se muestra arriba:

```
INSERT INTO phone_book  
  
SELECT 'John Doe', '555-1212' FROM LATERAL ( VALUES (1) ) AS t(c)  
  
UNION ALL  
  
SELECT 'Peter Doe', '555-2323' FROM LATERAL ( VALUES (1) ) AS t(c)
```

UPDATE

Una sentencia *UPDATE* de **SQL** es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

Forma básica

```
UPDATE "tabla" SET "columna1" = "valor1" [, "columna2" =  
"valor2", ...]  
  
WHERE "columnaN" = "valorN"
```

Ejemplo

```
UPDATE My_table SET field1 = 'updated value' WHERE field2 =  
'N';
```

DELETE

Una sentencia *DELETE* de **SQL** borra cero o más registros existentes en una tabla,

Forma básica

```
DELETE FROM "tabla" WHERE "columna1" = "valor1"
```

Ejemplo

```
DELETE FROM My_table WHERE field2 = 'N';
```

Lenguaje de control de datos (DCL)

Grant, Revoke. : Para otorgar y revocar permisos sobre objetos de la base de datos a usuarios

Commit, Rollback, Savepoint , Son instrucciones de control de transacciones, utilizadas para validar o deshacer transacciones en la base de datos.

Base de datos Oracle

Oracle es un sistema de gestión de base de datos relacional (o RDBMS por el acrónimo en inglés de Relational Data Base Management System), fabricado por Oracle Corporation.

Se considera a Oracle como uno de los sistemas de bases de datos más completos, destacando su:

- Soporte de transacciones.

- Estabilidad.
- Escalabilidad.
- Es multiplataforma.

Historia

Oracle surge a finales de los 70 bajo el nombre de Relational Software a partir de un estudio sobre SGBD (Sistemas Gestores de Base de Datos) de George Koch. Computer World definió este estudio como uno de los más completos jamás escritos sobre bases de datos. Este artículo incluía una comparativa de productos que erigía a Relational Software como el más completo desde el punto de vista técnico. Esto se debía a que usaba la filosofía de las bases de datos relacionales, algo que por aquella época era todavía desconocido.

En la actualidad, Oracle (Nasdaq: ORCL) todavía encabeza la lista. La tecnología Oracle se encuentra prácticamente en todas las industrias alrededor del mundo y en las oficinas de 98 de las 100 empresas Fortune 100. Oracle es la primera compañía de software que desarrolla e implementa software para empresas 100 por ciento activado por Internet a través de toda su línea de productos: base de datos, aplicaciones comerciales y herramientas de desarrollo de aplicaciones y soporte de decisiones. Oracle es el proveedor mundial líder de software para administración de información, y la segunda empresa de software.

Arquitectura de Oracle

Todo el mundo puede conducir un automóvil sin necesidad de conocer cómo funciona un motor de combustión interna y todos los subsistemas asociados a él. Pero entonces ciertos conceptos como aprovechamiento de la potencia, compresión, endurecimiento de la suspensión, motricidad, etc., le serán ajenos y nunca podrá sacar lo mejor del automóvil. Y si tiene algún problema se quedará tirado en la carretera.

De la misma manera, no podremos aspirar a que nuestras aplicaciones de BD funcionen bien si no conocemos la arquitectura del *motor* de la BD, el servidor. Es indispensable conocer los factores y parámetros que influyen en el funcionamiento de nuestro SGBD para poder solucionar los problemas que se

pueden plantear en cuanto nos salgamos de las aplicaciones estándares y básicas de BD, o en cuanto tengamos algún problema.

Tablas y Columnas

Los datos son almacenados en la BD utilizando tablas. Cada tabla está compuesta por un número determinado de columnas.

Las tablas propiedad del usuario SYS son llamadas tablas del diccionario de datos. Proveen el catálogo del sistema que permite que la BD se gestione a sí misma.

Las tablas se pueden relacionar entre ellas a través de las columnas que las componen. La BD se puede utilizar para asegurar el cumplimiento de esas relaciones a través de la integridad referencial, que se concreta en las restricciones de tablas.

Restricciones de Tablas

Una tabla puede tener asociadas restricciones que deben cumplir todas las filas. Entre las restricciones que se pueden fijar algunas reciben nombres especiales.: *clave primaria, clave ajena*.

La clave primaria de una tabla está compuesta por las columnas que hacen a cada fila de la tabla una fila distinta.

La clave ajena se utiliza para especificar las relaciones entre tablas. De modo que un conjunto de columnas declaradas como clave ajena de una tabla deben tener valores tomados de la clave primaria de otra tabla.

Usuarios

Una cuenta de usuario no es una estructura física de la BD, pero está relacionada con los objetos de la BD: los usuarios poseen los objetos de la BD. Existen dos usuarios especiales: SYS y SYSTEM. El usuarios SYS posee las tablas del diccionario de datos; que almacenan información sobre el resto de las estructuras de la BD. El usuario SYSTEM posee las vistas que permiten acceder a las tablas del diccionario, para el uso del resto de los usuarios de la BD.

Todo objeto creado en la BD se crea por un usuario, en un espacio de tablas y en un fichero de datos determinado. Toda cuenta de la BD puede estar unida a una

cuenta del S.O., lo que permite a los usuarios acceder a la cuenta de la BD sin dar la clave de acceso.

Cada usuario puede acceder a los objetos que posea o a aquellos sobre los que tenga derecho de acceso.

Índices

Un índice es una estructura de la BD utilizada para agilizar el acceso a una fila de una tabla. Cada fila tiene un identificador de fila, `ROWID`, que determina el fichero, bloque y fila dentro del bloque donde está almacenada la fila.

Cada entrada del índice consiste en un valor clave y una `ROWID`. Cada una de estas entradas se almacena en un árbol B⁺.

Los índices se crean automáticamente cuando se define una restricción `UNIQUE` o `PRIMARY KEY`.

Clusters

Las tablas que son accedidas juntas frecuentemente pueden ser almacenadas juntas. Para ello se crea un *cluster*. De este modo se minimiza el número de E/S.

Las columnas que relacionan las tablas de un *cluster* se llaman clave del *cluster*.

Vistas

Conceptualmente, una vista puede considerarse como una máscara que se extiende sobre una o más tablas, de modo que cada columna de la vista se corresponde con una o más columnas de las tablas subyacentes. Cuando se consulta una vista, esta traspasa la consulta a las tablas sobre las que se asienta. Las vistas no se pueden indexar.

Las vistas no generan almacenamiento de datos, y sus definiciones se almacenan en el diccionario de datos.

Vistas Materializadas

Una vista materializada es el resultado de la consulta que se almacena en una tabla caché real, que será actualizada de forma periódica a partir de las tablas originales. Esto proporciona un acceso mucho más eficiente, a costa de un incremento en el tamaño de la base de datos y a una posible falta de sincronía, es

decir, que los datos de la vista pueden estar potencialmente desfasados con respecto a los datos reales. Es una solución muy utilizada en entornos de almacenes de datos (datawarehousing), donde el acceso frecuente a las tablas básicas resulta demasiado costoso.

Tipos

SQL no es orientado a objetos. A pesar de no serlo, nos permite crear tipos de objetos. Un tipo de dato define una estructura y un comportamiento común para un conjunto de datos de las aplicaciones.

Secuencias

Las definiciones de secuencias se almacenan en el diccionario de datos. Son mecanismos para obtener listas de números secuenciales.

Procedimientos y Funciones

Un procedimiento es un bloque de código PL/SQL, que se almacena en el diccionario de datos y que es llamado por las aplicaciones. Se pueden utilizar para implementar seguridad, no dando acceso directamente a determinadas tablas sino es a través de procedimientos que acceden a esas tablas. Cuando se ejecuta un procedimiento se ejecuta con los privilegios del propietario del procedimiento. La diferencia entre un procedimiento y una función es que ésta última puede devolver valores.

Paquetes, Packages

Se utilizan para agrupar procedimientos y funciones. Los elementos dentro de los paquetes pueden ser públicos o privados. Los públicos pueden ser llamados por los usuarios, los privados están ocultos a los usuarios y son llamados por otros procedimientos.

Disparadores, Triggers

Son procedimientos que son ejecutados cuando se produce un determinado evento en la BD. Se pueden utilizar para mejorar y reforzar la integridad y la seguridad de la BD.

Sinónimos

Para identificar completamente un objeto dentro de una BD se necesita especificar el nombre de la máquina, el nombre del servidor, el nombre del propietario y el nombre del objeto. Para hacer transparente todo esto al usuario se pueden utilizar los sinónimos. Éstos apuntarán a los objetos y si el objeto cambia de lugar o propietario, sólo habrá que modificar el sinónimo.

Existen sinónimos públicos y privados. Los públicos son conocidos por todos los usuarios de una BD. Los privados son locales a un usuario.

Privilegios y Roles

Para que un objeto pueda ser accedido por un usuario debe de tener otorgado ese privilegio. Ejemplos de privilegios son INSERT, SELECT, UPDATE, EXECUTE, etc.

Los roles son grupos de privilegios que pueden ser utilizados para facilitar la gestión de los privilegios. Los privilegios se pueden otorgar a un rol, y los roles pueden ser otorgados a múltiples usuarios.

Esquemas

El conjunto de objetos de un usuario es conocido como esquema.

Lenguaje SQL

SqlPlus

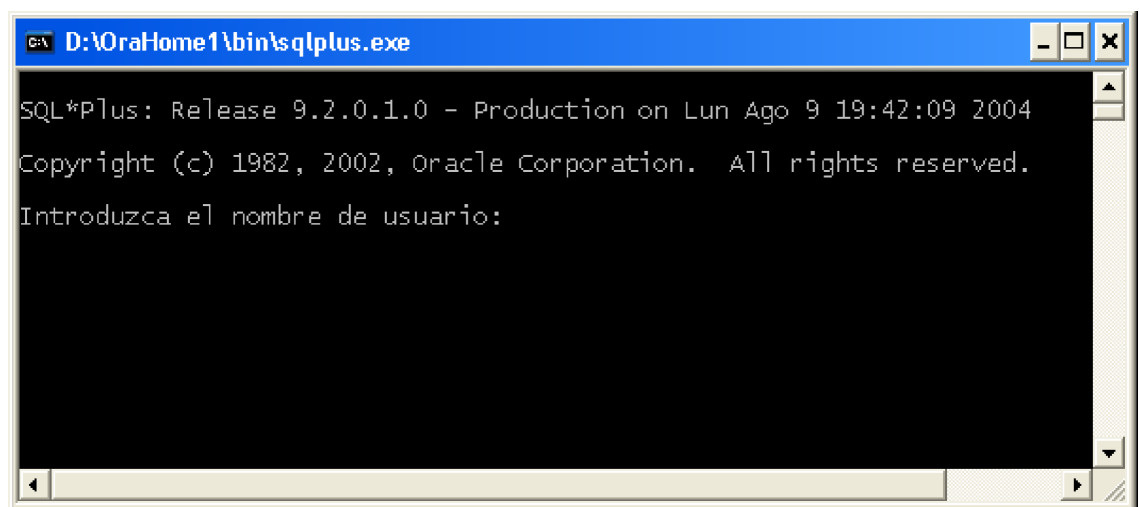
Para poder escribir sentencias SQL al servidor Oracle, éste incorpora la herramienta SQL*Plus. Toda instrucción SQL que el usuario escribe, es verificada por este programa. Si la instrucción es válida es enviada a Oracle, el cual enviará de regreso la respuesta a la instrucción; respuesta que puede ser transformada por el programa SQL*Plus paramodificar su salida.

Para que el programa SQL*Plus funcione en el cliente, el ordenador cliente debe haber sido configurado para poder acceder al servidor Oracle. En cualquier caso

al acceder a Oracle con este programa siempre preguntará por el nombre de usuario y contraseña.

Estos son datos que tienen que nos tiene que proporcionar el administrador (DBA) de la base de datos Oracle.

Para conectar mediante SQL*Plus podemos ir a la línea de comandos y escribir el texto sqlplus. A continuación aparecerá la pantalla:



En esa pantalla se nos pregunta el nombre de usuario y contraseña para acceder a la base de datos (información que deberá indicarnos el administrador o DBA). Tras indicar esa información conectaremos con Oracle mediante SQL*Plus, y veremos aparecer el símbolo:

SQL>

Tras el cual podremos comenzar a escribir nuestros comandos SQL. Ese símbolo puede cambiar por un símbolo con números 1, 2, 3, etc.; en ese caso se nos indica que la instrucción no ha terminado y la línea en la que estamos.

Otra posibilidad de conexión consiste en llamar al programa SQL*Plus indicando contraseña y base de datos a conectar.

El formato es:

sqlplus usuario/contraseña@nombreServicioBaseDeDatos

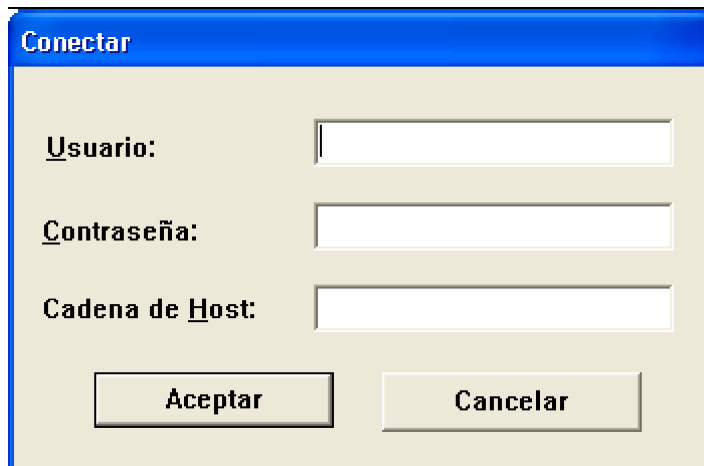
Ejemplo:

sqlplus usr1/miContra@inicial.forempa.net

En este caso conectamos con SQL*Plus indicando que somos el usuario usr1 con contraseña miContra y que conectamos a la base de datos inicial de la red forempa.net. El nombre de la base de datos no tiene porque tener ese formato, habrá que conocer como es el nombre que representa a la base de datos como servicio de red en la red en la que estamos.

Versión gráfica de SQL*Plus

Oracle incorpora un programa gráfico para Windows para utilizar SQL*Plus. Se puede llamar a dicho programa desde las herramientas instaladas en el menú de programas de Windows, o desde la línea de programas escribiendo **sqlplusw**. Al llamarle aparece esta pantalla:

The image shows a graphical user interface window titled "Conectar" (Connect). It has a blue title bar. The main area is light beige. There are three labels with corresponding text input fields: "Usuario:" (with a blue underline), "Contraseña:" (with a blue underline), and "Cadena de Host:" (with a blue underline). Below these fields are two buttons: "Aceptar" (Accept) and "Cancelar" (Cancel). The buttons have a 3D effect with a grey border and a light beige fill.

Como en el caso anterior, se nos solicita el nombre de usuario y contraseña. La *cadena de Host* es el nombre completo de red que recibe la instancia de la base de datos a la que queremos acceder en la red en la que nos encontramos.

También podremos llamar a este entorno desde la línea de comandos utilizando la sintaxis comentada anteriormente. En este caso:

sqlplusw usuario/contraseña@nombreServicioBaseDeDatos

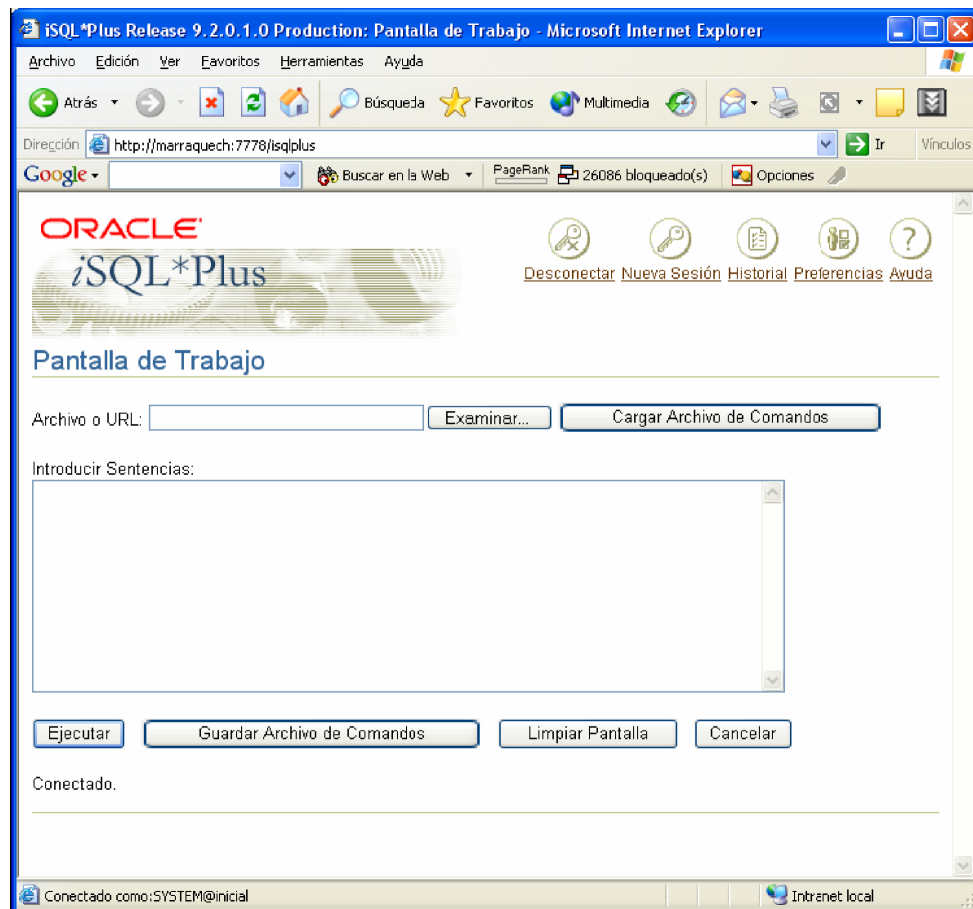
Esta forma de llamar al programa permite entrar directamente sin que se nos pregunte por el nombre de usuario y contraseña.

iSQL*Plus

Es un producto ideado desde la versión 9i de Oracle. Permite acceder a las bases de datos Oracle desde un navegador. Para ello necesitamos tener configurado un

servidor web Oracle que permita la conexión con la base de datos. Utilizar iSQL*Plus es indicar una dirección web en un navegador, esa dirección es la de la página iSQL*Plus de acceso a la base de datos.

Desde la página de acceso se nos pedirá nombre de usuario, contraseña y nombre de la base de datos con la que conectamos (el nombre de la base de datos es el nombre con el que se la conoce en la red). Si la conexión es válida aparece esta pantalla:



Otras Aplicaciones de desarrollo SQL

Golden: Golden es una herramienta de consulta con muchas funciones adicionales, ofrece compatibilidad con SQLPlus.

TOAD: Es una aplicación de software de desarrollo SQL y administración de base de datos, considerada una herramienta útil para los Oracle administradores de base de datos (DBAs) y desarrolladores. Está disponible para las siguientes bases de datos: Oracle, Microsoft SQL Server, IBM DB2, y MySQL.

Esquemas de ejemplo de Oracle

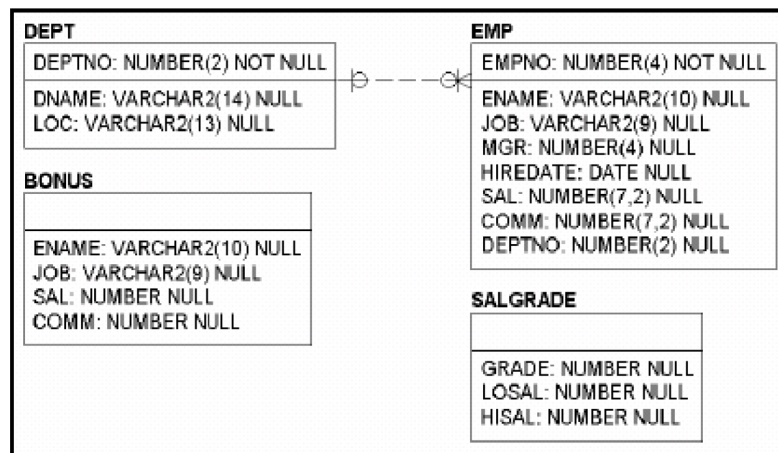
Esquema SCOTT

Para poder iniciar una sesión en el esquema de scott debemos utilizar los siguientes datos:

Usuario scott

Contraseña tiger

Su esquema es el siguiente:



El siguiente script permite consultar el catalogo de scott:

```
SQL> connect scott/tiger
```

```
Connected.
```

```
SQL> select * from cat;
```

```
TABLE_NAME TABLE_TYPE
```

```
-----
```

```
BONUS TABLE
```

```
DEPT TABLE
```

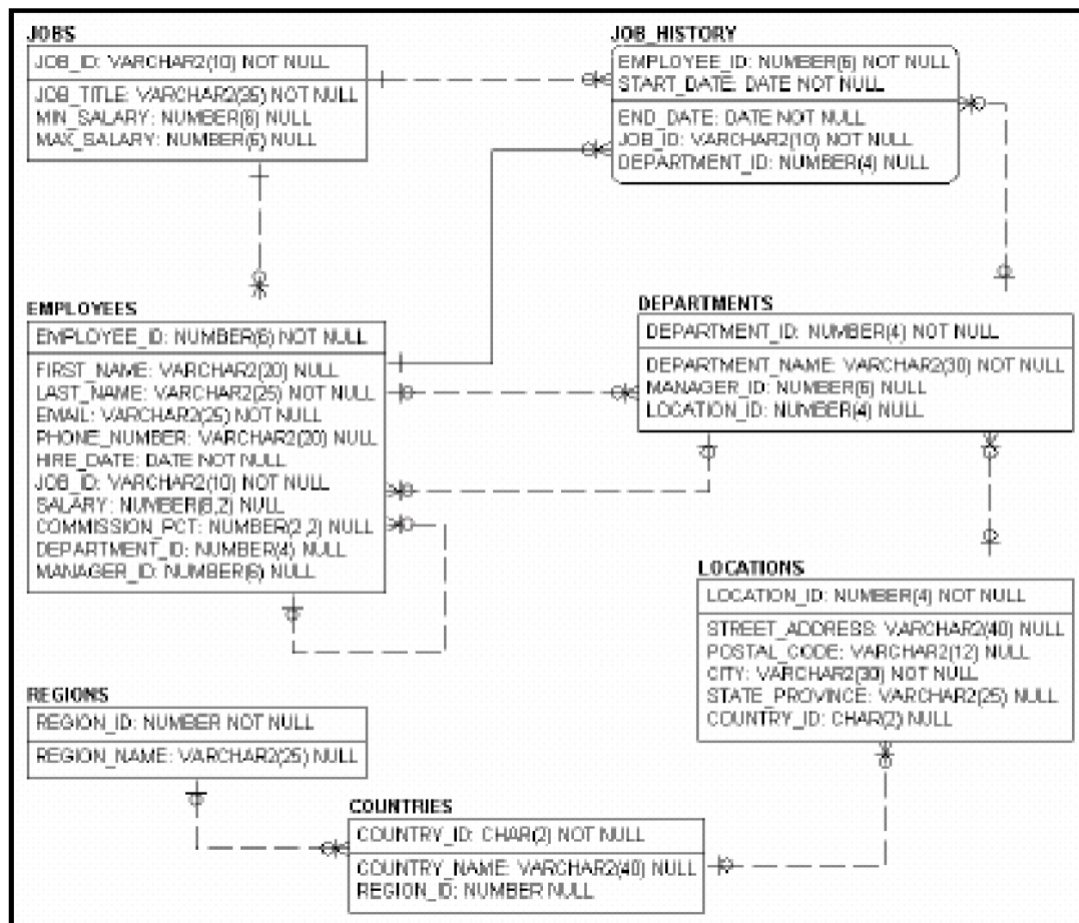
```
EMP TABLE
```

```
SALGRADE TABLE
```

```
4 rows selected.
```

Esquema HR

Su esquema es el siguiente:



La cuenta de usuario HR por defecto está bloqueada, así que lo primero que debemos hacer es desbloquearla, el script es el siguiente:

```
SQL> connect system/manager
```

Connected.

```
SQL> alter user hr
```

2 identified by hr

```
3 account unlock;
```

User altered.

Ahora si podemos consultar el catalogo del esquema HR:

```
SOL> connect hr/hr
```

Connected.

```
SQL> select * from cat;
```

TABLE	NAME	TABLE	TYPE
-------	------	-------	------

```
COUNTRIES TABLE
DEPARTMENTS TABLE
DEPARTMENTS_SEQ SEQUENCE
EMPLOYEES TABLE
EMPLOYEES_SEQ SEQUENCE
EMP_DETAILS_VIEW VIEW
JOBS TABLE
JOB_HISTORY TABLE
LOCATIONS TABLE
LOCATIONS_SEQ SEQUENCE
REGIONS TABLE

11 rows selected.
```

También podemos utilizar la siguiente consulta:

```
SQL> select * from tab;
```

TNAME	TABTYPE	CLUSTERID
COUNTRIES	TABLE	
DEPARTMENTS	TABLE	
EMPLOYEES	TABLE	
EMP_DETAILS_VIEW	VIEW	
JOBS	TABLE	
JOB_HISTORY	TABLE	
LOCATIONS	TABLE	
REGIONS	TABLE	

8 rows selected.

Sentencias DDL(Lenguaje de definición de datos)

Tipos de Datos en Oracle

TIPO	CARACTERÍSTICAS	OBSERVACIONES
CHAR	Cadena de caracteres (alfanuméricos) de longitud fija	Entre 1 y 2000 bytes como máximo. Aunque se introduzca un valor más corto que el indicado en el tamaño, se rellenará al tamaño indicado. Es de longitud fija, siempre ocupará lo mismo, independientemente del valor que contenga
VARCHAR2	Cadena de caracteres de longitud variable	Entre 1 y 4000 bytes como máximo. El tamaño del campo dependerá del valor que contenga, es de longitud variable.
VARCHAR	Cadena de caracteres de longitud variable	En desuso, se utiliza VARCHAR2 en su lugar
NCHAR	Cadena de caracteres de longitud fija que sólo almacena caracteres Unicode	Entre 1 y 2000 bytes como máximo. El juego de caracteres del tipo de datos (datatype) NCHAR sólo puede ser AL16UTF16 ó UTF8. El juego de caracteres se especifica cuando se crea la base de datos Oracle
NVARCHAR2	Cadena de caracteres de longitud variable que sólo almacena caracteres Unicode	Entre 1 y 4000 bytes como máximo. El juego de caracteres del tipo de datos (datatype) NCHAR sólo puede ser AL16UTF16 ó UTF8. El juego de caracteres se especifica cuando se crea la base de datos Oracle

LONG	Cadena de caracteres de longitud variable	<p>Como máximo admite hasta 2 GB (2000 MB). Los datos LONG deberán ser convertidos apropiadamente al moverse entre diversos sistemas.</p> <p>Este tipo de datos está obsoleto (en desuso), en su lugar se utilizan los datos de tipo LOB (CLOB, NCLOB). Oracle recomienda que se convierta el tipo de datos LONG a alguno LOB si aún se está utilizando.</p> <p>No se puede utilizar en cláusulas WHERE, GROUP BY, ORDER BY, CONNECT BY ni DISTINCT</p> <p>Una tabla sólo puede contener una columna de tipo LONG.</p> <p>Sólo soporta acceso secuencial.</p>
LONG RAW	Almacenan cadenas binarias de ancho variable	<p>Hasta 2 GB.</p> <p>En desuso, se sustituye por los tipos LOB.</p>
RAW	Almacenan cadenas binarias de ancho variable	<p>Hasta 32767 bytes.</p> <p>En desuso, se sustituye por los tipos LOB.</p>
LOB (BLOB, CLOB, NCLOB, BFILE)	Permiten almacenar y manipular bloques grandes de datos no estructurados (tales como texto, imágenes, videos, sonidos, etc) en formato binario o del carácter	<p>Admiten hasta 8 terabytes (8000 GB).</p> <p>Una tabla puede contener varias columnas de tipo LOB.</p> <p>Soportan acceso aleatorio.</p> <p>Las tablas con columnas de tipo LOB no pueden ser replicadas.</p>

BLOB	Permite almacenar datos binarios no estructurados	Admiten hasta 8 terabytes
CLOB	Almacena datos de tipo carácter	Admiten hasta 8 terabytes
NCLOB	Almacena datos de tipo carácter	Admiten hasta 8 terabytes. Guarda los datos según el juego de caracteres Unicode nacional.
BFILE	Almacena datos binarios no estructurados en archivos del sistema operativo, fuera de la base de datos. Una columna BFILE almacena un localizador del archivo a uno externo que contiene los datos	Admiten hasta 8 terabytes. El administrador de la base de datos debe asegurarse de que exista el archivo en disco y de que los procesos de Oracle tengan permisos de lectura para el archivo .
ROWID	Almacenar la dirección única de cada fila de la tabla de la base de datos	ROWID físico almacena la dirección de fila en las tablas, las tablas en clúster, los índices, excepto en las índices-organizados (IOT). ROWID lógico almacena la dirección de fila en tablas de índice-organizado (IOT). Un ejemplo del valor de un campo ROWID podría ser: "AAAIugAAJAAC4AhAAI". El formato es el siguiente: Para "OOOOOOFFFB BBBBRRRR", donde:

		<p>OOOOOO: segmento de la base de datos (AAAIug en el ejemplo). Todos los objetos que estén en el mismo esquema y en el mismo segmento tendrán el mismo valor.</p> <p>FFF: el número de fichero del tablespace relativo que contiene la fila (fichero AAJ en el ejemplo).</p> <p>BBBBBB: el bloque de datos que contiene a la fila (bloque AAC4Ah en el ejemplo). El número de bloque es relativo a su fichero de datos, no al tablespace. Por lo tanto, dos filas con números de bloque iguales podrían residir en diferentes datafiles del mismo tablespace.</p> <p>RRR: el número de fila en el bloque (fila AAI en el ejemplo).</p> <p>Este tipo de campo no aparece en los SELECT ni se puede modificar en los UPDATE, ni en los INSERT. Tampoco se puede utilizar en los CREATE. Es un tipo de datos utilizado exclusivamente por Oracle. Sólo se puede ver su valor utilizando la palabra reservada ROWID, por ejemplo:</p> <p>select rowid, nombre, apellidos from clientes</p> <p>Ejemplo 2:</p> <p>SELECT ROWID, SUBSTR(ROWID,15,4) "Fichero",</p>
--	--	--

		<p>SUBSTR(ROWID,1,8) "Bloque", SUBSTR(ROWID,10,4) "Fila" FROM proveedores</p> <p>Ejemplo 3: una forma de saber en cuántos ficheros de datos está alojada una tabla:</p> <p>SELECT COUNT(DISTINCT(SUBSTR(ROWID, 7,3))) "Numero ficheros " FROM facturacion</p>
UROWID	ROWID universal	<p>Admite ROWID a tablas que no sean de Oracle, tablas externas. Admite tanto ROWID lógicos como físicos.</p>
NUMBER	Almacena números fijos y en punto flotante	<p>Se admiten hasta 38 dígitos de precisión y son portables a cualquier entre los diversos sistemas en que funcione Oracle.</p> <p>Para declarar un tipo de datos NUMBER en un CREATE ó UPDATE es suficiente con:</p> <p>nombre_columna NUMBER</p> <p>opcionalmente se le puede indicar la precisión (número total de dígitos) y la escala (número de dígitos a la derecha de la coma, decimales, los cogerá de la precisión indicada):</p> <p>nombre_columna NUMBER (precision, escala)</p> <p>Si no se indica la precisión se tomará en función del número a guardar, si no se</p>

		<p>indica la escala se tomará escala cero.</p> <p>Para no indicar la precisión y sí la escala podemos utilizar:</p> <p>nombre_columna NUMBER (*, escala)</p> <p>Para introducir números que no estén en el formato estándar de Oracle se puede utilizar la función TO_NUMBER.</p>
FLOAT	Almacena tipos de datos numéricos en punto flotante	Es un tipo NUMBER que sólo almacena números en punto flotante
DATE	Almacena un punto en el tiempo (fecha y hora)	<p>El tipo de datos DATE almacena el año (incluyendo el siglo), el mes, el día, las horas, los minutos y los segundos (después de medianoche).</p> <p>Oracle utiliza su propio formato interno para almacenar fechas.</p> <p>Los tipos de datos DATE se almacenan en campos de longitud fija de siete octetos cada uno, correspondiendo al siglo, año, mes, día, hora, minuto, y al segundo.</p> <p>Para entrada/salida de fechas, Oracle utiliza por defecto el formato <i>DD-MMM-AA</i>. Para cambiar este formato de fecha por defecto se utiliza el parámetro <i>NLS_DATE_FORMAT</i>.</p> <p>Para insertar fechas que no estén en el mismo formato de fecha estándar de Oracle, se puede utilizar la función <i>TO_DATE</i> con una máscara del formato:</p>

		TO_DATE (el “13 de noviembre de 1992”, “DD del MES, YYYY”)
TIMESTAMP	Almacena datos de tipo hora, fraccionando los segundos	
TIMESTAMP WITH TIME ZONE	Almacena datos de tipo hora incluyendo la zona horaria (explícita), fraccionando los segundos	
TIMESTAMP WITH LOCAL TIME ZONE	Almacena datos de tipo hora incluyendo la zona horaria local (relativa), fraccionando los segundos	Cuando se usa un SELECT para mostrar los datos de este tipo, el valor de la hora será ajustado a la zona horaria de la sesión actual
XMLType	Tipo de datos abstracto. En realidad se trata de un CLOB.	Se asocia a un esquema XML para la definición de su estructura.

De los tipos anteriores, los comunmente utilizados son: **VARCHAR2** (cadenas de texto no muy grandes), **DATE** (fechas, horas), **NUMBER** (números), **BLOB** (ficheros de tipo word, excel, access, video, sonido, imágenes, etc) y **CLOB** (cadenas de texto muy grandes).

Create

Create Table

La estructura de la sentencia de creación de tablas es:

```

CREATE [GLOBAL TEMPORARY] TABLE [esquema.]tabla

    columna datatype [DEFAULT expr]
[column_constraint(s)]

[,columna datatype [,...]]

table_constraint

table_ref_constraint

[ON COMMIT {DELETE|PRESERVE} ROWS]

storage_options    [COMPRESS int|NOCOMPRESS]

[LOB_storage_clause][varray_clause][nested_storage_clause
] [XML_type_clause]

Partitioning_clause

[[NO]CACHE] [[NO]ROWDEPENDENCIES] [[NO]MONITORING]
[PARALLEL parallel_clause]

[ENABLE enable_clause | DISABLE disable_clause]

{ENABLE|DISABLE} ROW MOVEMENT

[AS subquery]

```

Queda mas claro con unos ejemplos:

```

create table T_PRODUCTOS

(

    numproduct number,

    desproduct varchar2(10)

);

```

Es posible definir restricciones (constraint) con la sentencia CREATE.

```

create table T_CLIENTES

(

    numclie number primary key,

    desclie varchar2(10)

)      ;

create table T_PEDIDOS

```

```
(
    numpedido    number primary key,
    fecpedido    date,
    numclient    references T_CLIENTES
);
```

Una clave primaria (primary key) necesita tener asociado un índice único (unique index). Es posible especificar el tablespace donde queremos crear el índice.

```
create table T_PEDIDOS(
    numpedido    number primary key    using index tablespace
users          ,
    fecpedido    date,
    numclient    references T_CLIENTES
)
```

Organization external , el siguiente script crea una table externa:

```
create table (....)
organization external (
    type          oracle_loader
    default directory some_dir
    access parameters (
        records delimited by newline
        fields terminated by ','
        missing field are values null
    )
    location ('fichero.csv')
)
reject limit unlimited;
```

Nested tables

```
create or replace type item as object (
    item_id Number ( 6 ),
```

```

        descr    varchar2(30 ),
        quant    Number    ( 4,2)
    );
/

create or replace type items as table of item;
/

create table bag_with_items (
    bag_id                number(7)        primary key,
    bag_name              varchar2(30)    not null,
    the_items_in_the_bag  items
)
nested table the_items_in_the_bag store as bag_items_nt;

```

Create Index

Los índices se usan para mejorar el rendimiento de las operaciones sobre una tabla.

En general mejoran el rendimiento las SELECT y empeoran (minimamente) el rendimiento de los INSERT y los DELETE.

Una vez creados no es necesario nada más, oracle los usa cuando es posible (ver EXPLAIN PLAN).

En oracle existen tres tipos de índices:

1) Table Index:

```

CREATE [UNIQUE|BITMAP] INDEX [esquema.]index_name
    ON [esquema.]table_name [tbl_alias]
    (col [ASC | DESC]) index_clause index_attribs

```

2) Bitmap Join Index:

```

CREATE [UNIQUE|BITMAP] INDEX [esquema.]index_name
    ON [esquema.]table_name [tbl_alias]

```

```

        (col_expression [ASC | DESC])
        FROM [esquema.]table_name [tbl_alias]
        WHERE condition [index_clause]
index_attribs

```

3) Cluster Index:

```

CREATE [UNIQUE|BITMAP] INDEX [esquema.]index_name
ON CLUSTER [esquema.]cluster_name index_attribs

```

Las clausulas posibles para los índices son:

```

LOCAL STORE IN (tablespace)

```

```

LOCAL STORE IN (tablespace)

```

```

(PARTITION [partition
[LOGGING|NOLOGGING]
[TABLESPACE {tablespace|DEFAULT}]
[PCTFREE int]
[PCTUSED int]
[INITRANS int]
[MAXTRANS int]
[STORAGE storage_clause]
[STORE IN {tablespace_name|DEFAULT}]
[SUBPARTITION [subpartition [TABLESPACE
tablespace]]]])

```

```

LOCAL (PARTITION [partition
[LOGGING|NOLOGGING]
[TABLESPACE {tablespace|DEFAULT}]
[PCTFREE int]
[PCTUSED int]
[INITRANS int]
[MAXTRANS int]

```



```
[STORAGE storage_clause]

[STORE IN {tablespace_name|DEFAULT}

[SUBPARTITION [subpartition [TABLESPACE
tablespace]]]])
```

```
GLOBAL PARTITION BY RANGE (col_list)

( PARTITION partition VALUES LESS THAN (value_list)

[LOGGING|NOLOGGING]

[TABLESPACE {tablespace|DEFAULT}]

[PCTFREE int]

[PCTUSED int]

[INITRANS int]

[MAXTRANS int]

[STORAGE storage_clause] )
```

```
INDEXTYPE IS indextype [PARALLEL int|NOPARALLEL]
[PARAMETERS ('ODCI_Params')]
```

{Esto es solo para table index, no para bitmap join
Index}

Y además index_attrihs puede ser cualquier combinación de
los siguientes:

NOSORT|SORT

REVERSE

COMPRESS int

NOCOMPRESS

COMPUTE STATISTICS

[NO] LOGGING

ONLINE

TABLESPACE {tablespace|DEFAULT}

PCTFREE int

PCTUSED int

INITTRANS int

MAXTRANS int

STORAGE storage_clause

PARALLEL parallel_clause

Si usamos la opcion PARALLEL esta debe estar al final.

create index es una de las pocas sentencias que pueden usar nologging option.

create index requiere un segmento temporal si no hay espacio en memoria suficiente.

Crear indices basados en funciones require que query_rewrite_enabled este a true y query_rewrite_integrity este a trusted.

Un ejemplo de indices basados en funciones para busquedas en mayusculas:

```
CREATE INDEX idx_case_ins ON my_table(UPPER(empname));
```

```
SELECT * FROM my_table WHERE UPPER (empname) = 'KARL';
```

Storage clause

Configuración del almacenamiento de tablas (CREATE TABLE), indices (CREATE INDEX), etc... en oracle.

STORAGE opciones

Opciones:

INITIAL int K | M

NEXT int K | M

MINEXTENTS int

MAXEXTENTS int

MAXEXTENTS UNLIMITED

PCTINCREASE int

FREELISTS int

FREELIST GROUPS int

```

        OPTIMAL

        OPTIMAL int K | M

        OPTIMAL NULL

        BUFFER POOL {KEEP|RECYCLE|DEFAULT}

storage (

    initial          65536

    next             1048576

    minextents       1

    maxextents       2147483645

    pctincrease      0

    freelists        1

    freelist groups  1

    optimal          7k

    buffer_pool      default

)

```

Esta clausula aparece al final de la definición de los objetos de almacenamiento de la base de datos (tablas, indices, etc...).

Cuando creamos un tablespace (CREATE TABLESPACE) podemos definir un storage por defecto para los objetos que se creen dentro de el.

However, a default storage clause can not be specified for locally managed tablespaces. Dictionary managed tablespaces allow to have a storage clause, but without freelists, freelist groups and buffer_pool.

Initial: Especifica el tamaño (en bytes) de la primera extensión.

Next: Especifica el tamaño (en bytes) de la segunda extensión.

Pctincrease: Especifica el % de incremento en el tamaño de las siguientes extensiones.

Especifica el incremento en el tamaño de las siguientes extensiones. El tamaño de una nueva extension es el tamaño de la anterior multiplico por pctincrease. Debe ser 0 para reducir la fragmentación en los tablespaces.

Minextents: Especifica el numero inicial de extensiones cuando se crea el objeto.

Maxextents: Especifica el número máximo de extensiones que el objeto puede tener.

Freelists: Especifica el número de freelists. Este parámetro solo se puede usar con CREATE TABLE or CREATE INDEX.

Freelist groups: Especifica el numero de freelist groups. Este parámetro solo se puede usar con CREATE TABLE or CREATE INDEX.

Buffer_pool: El valor de buffer_pool debe ser uno de: keep, recycle, default. Este parámetro solo se puede usar con CREATE TABLE, CREATE INDEX, CREATE CLUSTER, ALTER TABLE, ALTER INDEX Y ALTER CLUSTER.

Optimal: Solo se puede especificar para los rollback segments.

Create Sequence

Crea un objeto capaz de darnos numeros consecutivos unicos.

```
CREATE SEQUENCE secuencia  
  
    INCREMENT BY n  
  
    START WITH n  
  
    {MAX VALUE n | NOMAXVALUE}  
  
    {MIN VALUE N | NOMINVALUE}  
  
    {CYCLE | NOCYCLE}  
  
    {CACHE N | NOCACHE}  
  
    {ORDER | NOORDER};
```

En realida es un generador de indentificadores unicos que no bloquea transacciones.

Es muy util para generar primary keys.

Si no nos gusta perder números usamos NOCACHE.

```
CREATE SEQUENCE S_PROVEEDORES MINVALUE 1 START WITH 1  
  
    INCREMENT BY 1 NOCACHE;
```

Si nos interesa la velocidad:

```
CREATE SEQUENCE S_PROVEEDORES MINVALUE 1 START WITH 1  
INCREMENT BY 1 CACHE 20;
```

Asi obtenemos el siguiente valor:

```
SELECT S_PROVEEDORES.NEXTVAL FROM DUAL;
```

Tambien podemos obtener el valor actual:

```
SELECT S_PROVEEDORES.CURRVAL FROM DUAL;
```

Create View

Esta sentencia sirve para crear una vista de una tabla o tablas.

Una vista es una tabla lógica basada en los datos de otra tabla.

Ejemplo:

```
CREATE VIEW V_PEDIDOS (NUMPEDIDO, FECPEDIDO, NUNCLIEN  
TE, NOMCLIEN  
TE)  
FROM  
SELECT A.NUMPEDIDO,A.FECPEDIDO,A.NUMCLIEN  
TE, B.NOMCLIEN  
TE  
FROM T_PEDIDOS A, T_CLIEN  
TE B  
WHERE A.NUMCLIEN  
TE=B.NUMCLIEN  
TE;
```

Esta vista sacará lo datos de los pedidos con el nombre de cliente.

Al ser lógica no necesita espacio de almacenamiento para los datos. Ademas es instantanea, una vez modificados los datos de las tablas origen, los tenemos disponibles en la vista.

Create Global Temporary Tables

Crea una tabla temporal personal para cada sesion. Eso significa que los datos no se comparten entre sesiones y se eliminan al final de la misma.

```
CREATE GLOBAL TEMPORARY TABLE tabla_temp (  
columna datatype [DEFAULT expr]  
[column_constraint(s)]  
[,columna datatype [...]]
```

```
) {ON COMMIT DELETE ROWS | ON COMMIT PRESERVE ROWS};
```

Por ejemplo;

```
CREATE GLOBAL TEMPORARY TABLE tabla_temp (  
    columna number  
)  
ON COMMIT DELETE ROWS ;
```

```
CREATE GLOBAL TEMPORARY TABLE tabla_temp2 (  
    columna number  
)  
ON COMMIT PRESERVE ROWS ;
```

Con la opción ON COMMIT DELETE ROWS se borran los datos cada vez que se hace COMMIT en la sesión.

Con la opción ON PRESERVE DELETE ROWS los datos no se borran hasta el final de la sesión.

Create materialized view

El SQL de las bases de datos Oracle permite crear vistas materializadas o materialized views. Estas vistas materializadas, a parte de almacenar la definición de la vista propiamente dicha, también almacenan los registros que resultan de la ejecución de la sentencia SELECT que define la vista. Como las vistas normales, la sentencia SELECT es la base de la vista, pero la sentencia SQL se ejecuta cuando se crea la vista y los resultados se almacenan físicamente constituyendo una tabla real que ocupa sitio en el disco duro. Esta tabla puede definirse utilizando los mismos parámetros de almacenamiento que se pueden utilizar para una tabla normal (tablespace, etcétera). Las vistas materializadas también admiten índices, esta funcionalidad resulta muy útil a la hora de mejorar el rendimiento de las sentencias PLSQL o SQL que utilicen vistas materializadas.

Cuando una sentencia SQL o PL/SQL accede a una vista materializada el servidor de la base de datos Oracle, transforma la sentencia dirigiéndose directamente a los datos de la vista que están ya almacenados, en lugar de utilizar los datos de las diferentes tablas utilizadas en la definición de dicha vista.

Evidentemente, si una vista (*view*) utiliza muchas tablas base enlazadas de forma compleja, y dicha vista va a ser utilizada frecuentemente, será muy conveniente definirla como una vista materializada o *materialized view*. Esto contribuirá enormemente a mejorar el rendimiento de la base de datos, ya que la sentencia SQL base de la vista sólo se ejecutará una vez.

Por otro lado, está el inconveniente de que si la vista materializada o *materialized view* va a tener que reutilizarse en el futuro, entonces necesitaremos un mecanismo para actualizar o refrescar dicha vista materializada, ya que las tablas base de la vista pueden haber sufrido modificaciones desde la creación de la misma.

Por todo esto, a la hora de determinar si una vista debe definirse como vista o es mejor definirla como vista materializada, debemos valorar los costes de tener que ejecutar la sentencia SQL base de una vista normal siempre que se acceda a dicha vista, frente a los costes de almacenamiento y actualización de una vista materializada.

Sintaxis del comando SQL utilizado para crear vistas materializadas

```
CREATE MATERIALIZED VIEW nombre_vistam
[TABLESPACE nombre_ts]
[PARALLEL (DEGREE n)]
[BUILD {IMMEDIATE|DEFERRED}]
[REFRESH {FAST|COMPLETE|FORCE|NEVER|ON COMMIT}]
[{ENABLE|DISABLE} QUERY REWRITE]
AS SELECT ... FROM ... WHERE ...
```

Los valores por defecto de las distintas opciones están subrayados.

Si se elige la opción `BUILD IMMEDIATE`, entonces la tabla asociada con la vista materializada se puebla con datos en el momento de la ejecución del comando `SQL CREATE`. Por el contrario, si se utiliza `BUILD DEFERRED`, el comando `CREATE` creará sólo la estructura de la vista, pero la tabla física asociada no se poblará con datos hasta que se realice el primer refresco o actualización de la vista materializada.

La opción *REFRESH* permite indicar el mecanismo que la base de datos utilizará para refrescar o actualizar la vista materializada. Los diferentes mecanismos y la forma en que una vista materializada o *materialized view* puede refrescarse, serán objeto de otro artículo en este blog. Como anticipo diré que un refresco completo o *COMPLETE*, significa que la tabla asociada con la vista materializada se borra completamente, volviéndose a insertar todos los registros devueltos por la ejecución de la sentencia `SQL` base de la vista, y que un refresco rápido o *FAST*, significa que la vista materializada se actualiza sólo según hayan sido los cambios realizados sobre las tablas base de la vista desde el último refresco. Para poder utilizar el refresco rápido o *FAST*, hay que crear previamente los *logs* de la vista materializada utilizando el comando *CREATE MATERIALIZED VIEW LOG*.

La opción `ENABLE/DISABLE QUERY REWRITE` determina si el optimizador Oracle puede o no reescribir las sentencias `SQL` de manera que, de ser posible, en la fase de ejecución se utilice la vista materializada en lugar de las tablas base de la vista incluidas en la sentencia `SQL` original. Este es un tema ciertamente complejo y que será objeto de un artículo completo en este blog. Como anticipo indicaré que la reescritura de sentencias `SQL` sólo está disponible cuando se utiliza el optimizador Oracle basado en costes.

Create Synonym

Crea un sinonimo para algun objeto de la base de datos.

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM
[esquema.]sinonimo
FOR [esquema.]objeto [@dblink]
```


Con la opción 'PUBLIC' se crea un sinonimo público accesible a todos los usuarios, siempre que tengan los privilegios adecuados para el mismo. (ver GRANT)

Sirve para no tener que usar la notación 'esquema.objeto' para referirse a un objeto que no es propiedad de usuario.

```
CREATE PUBLIC SYNONYM T_PEDIDOS FOR  
PROGRAMADOR.T_PEDIDOS;
```

No es necesario recompilarlos cuando se redefinen las tablas, de hecho puedes existir sin que exista el objeto al que refererencian.

El acceso es un poco mas eficiente cuando se accede por sinonimos públicos.

Cuando en una sentencia no citamos el nombre del esquema, Oracle resuelve los nombres en el siguiente orden:

- usuario actual
- private synonym
- public synonym

Tambien podemos usarlo para cambiar el objeto que usamos sin tener que cambiar la programacion.

Asi cambiamos la tabla:

```
CREATE PUBLIC SYNONYM T_PEDIDOS FOR  
PROGRAMADOR.T_PEDIDOS_PRUEBA;
```

Create User

Esta sentencia sirve para crear un usuario oracle.

Un usuario es un nombre de acceso a la base de datos oracle. Normalmente va asociado a una clave (password).

Lo que puede hacer un usuario una vez ha accedido a la base de datos depende de los permisos que tenga asignados ya sea directamente (GRANT) como sobre algun rol que tenga asignado (CREATE ROLE).

El perfil que tenga asignado influye en los recursos del sistema de los que dispone un usuario a la hora de ejecutar oracle (CREATE PROFILE).

La sintaxis es:

```
CREATE USER username

        IDENTIFIED {BY password | EXTERNALLY | GLOBALLY AS
'external_name'}

        options;
```

Donde options:

```
DEFAULT TABLESPACE tablespace

TEMPORARY TABLESPACE tablespace

QUOTA int {K | M} ON tablespace

QUOTA UNLIMITED ON tablespace

PROFILE profile_name

PASSWORD EXPIRE

ACCOUNT {LOCK|UNLOCK}
```

Alterar un usuario

```
ALTER USER NOMBRE_USUARIO

IDENTIFIED BY CLAVE_ACCESO

[DEFAULT TABLESPACE ESPACIO_TABLA]

[TEMPORARY TABLESPACE ESPACIO_TABLA]

[QUOTA {ENTERO {K | M } | UNLIMITED } ON ESPACIO_TABLA]

[PROFILE PERFIL];
```

Crea un usuario sin derecho a guardar datos o crear objetos:

```
CREATE USER usuariolimitado IDENTIFIED BY miclavesecreta;
```

Crea un usuario con todos los derechos para guardar datos o crear objetos:

```
DROP USER miusuario CASCADE;
```

```
CREATE USER miusuario IDENTIFIED BY miclavesecreta
```

```
DEFAULT TABLESPACE data  
TEMPORARY TABLESPACE temp  
QUOTA UNLIMITED ON data;
```

```
CREATE ROLE programador;
```

```
GRANT CREATE session, CREATE table, CREATE view,  
CREATE procedure,CREATE synonym,  
ALTER table, ALTER view, ALTER procedure,ALTER  
synonym,  
DROP table, DROP view, DROP procedure,DROP synonym,  
TO conn;
```

```
GRANT programador TO miusuario;
```

Es necesario crear el usuario antes de asignar permisos con GRANT o un ROLE por defecto.

Create Role

Esta sentencia sirve para crear un rol de usuario.

Un rol es una forma de agrupar permisos (o privilegios) para asignarlos luego a los usuarios.

Cada usuario puede tener varios roles.

Ejemplo de creación de un rol:

```
CREATE ROLE MI_PROPIO_ROLE
```

Crea un rol sin password:

```
CREATE ROLE role NOT IDENTIFIED
```

Crea un rol con password:

```
CREATE ROLE role IDENTIFIED BY password
```

Crea un rol de aplicación:

```
CREATE ROLE role IDENTIFIED USING [schema.]package
```

Crea un rol basado en uno del S.O.:

```
ALTER ROLE role IDENTIFIED EXTERNALLY
```

Crea un rol basado en el servicio de directorio:

```
ALTER ROLE role IDENTIFIED GLOBALLY
```

Ejemplo para crear un script que asigna todos los permisos de actual esquema

```
SELECT decode(object_type,
'TABLE','GRANT SELECT, INSERT, UPDATE, DELETE ,
REFERENCES ON'||&OWNER||'.',
'VIEW','GRANT SELECT ON '||&OWNER||'.',
'SEQUENCE','GRANT SELECT ON '||&OWNER||'.',
'PROCEDURE','GRANT EXECUTE ON '||&OWNER||'.',
'PACKAGE','GRANT EXECUTE ON '||&OWNER||'.',
'FUNCTION','GRANT EXECUTE ON'||&OWNER||'.')
||object_name||' TO MI_PROPIO_ROLE ;'
FROM user_objects
WHERE
OBJECT_TYPE IN ( 'TABLE', 'VIEW', 'SEQUENCE',
'PROCEDURE', 'PACKAGE', 'FUNCTION')
ORDER BY OBJECT_TYPE
```

Create profile

Esta sentencia sirve para crear un perfil de usuario.

Un perfil de usuario es una forma de limitar los recursos que puede utilizar un usuario.

Cada usuario puede tener un único perfil.

Antes de asignar un perfil a un usuario es necesario que este perfil exista en la base de datos.

Un perfil se asigna en la creación de un usuario CREATE USER o modificandolo ALTER USER.

Un ejemplo de script sería:

```

CREATE PROFILE app_user LIMIT

SESSIONS_PER_USER                2      --

CPU_PER_SESSION                  10000   -- decimas de segundo

CPU_PER_CALL                     1      -- decimas de segundo

CONNECT_TIME                     UNLIMITED -- minutos

IDLE_TIME                        30      -- minutos

LOGICAL_READS_PER_SESSION        DEFAULT -- DB BLOCKS

LOGICAL_READS_PER_CALL           DEFAULT -- DB BLOCKS

-- COMPOSITE_LIMIT                DEFAULT --

PRIVATE_SGA                     20M     --

FAILED_LOGIN_ATTEMPTS            3      --

PASSWORD_LIFE_TIME               30     -- dias

PASSWORD_REUSE_TIME              12     --

PASSWORD_REUSE_MAX               UNLIMITED --

PASSWORD_LOCK_TIME               DEFAULT -- dias

PASSWORD_GRACE_TIME              2      -- dias

PASSWORD_VERIFY_FUNCTION         NULL;

```

Los recursos que limitamos son recursos del kernel: uso de la CPU, duración de sesión,...

Y tambien limites de uso de las claves de acceso (passwords): duración, intentos de acceso, reuso, ...

Por ejemplo:

```
ALTER PROFILE default LIMIT IDLE_TIME 20;
```

Limita el perfil por defecto a 20 minutos. IDLE_TIME: Es el tiempo que puede estar una sesión sin hacer nada antes de ser cerrada.

Drop

Utilice Drop para mover objetos a la papelera de reciclaje de Oracle:

Ejemplos:

Borrar una tabla:

```
DROP TABLE [schema.]table [CASCADE CONSTRAINTS]
[PURGE];
```

Borrar una Funcion:

```
DROP FUNCTION [schema.]function
```

Borrar un índice:

```
DROP INDEX [schema.]index [FORCE]
```

FORCE se puede utilizar para borrar índices que estas siendo utilizados

Borrar una vista materializada:

```
DROP MATERIALIZED VIEW [schema.] materialized_view
```

Borrar un log de vista materializada:

```
DROP MATERIALIZED VIEW LOG ON [schema.]table;
```

Borrar un paquete:

```
DROP PACKAGE [BODY] [schema.]package_name;
```

Borrar un procedimiento:

```
DROP PROCEDURE [schema.]procedure_name
```

Borrar un perfil:

```
DROP PROFILE profile_name [CASCADE]
```

Si un usuario esta asociado a un profile, esté no puede ser borrado, utilice CASCADE para desasignar antes los profiles de los usuarios.

Borrar un rol:

```
DROP ROLE role
```

Borrar un segmento:

```
DROP ROLLBACK SEGMENT rbs_name
```

Borrar una secuencia:

```
DROP SEQUENCE [schema.]sequence_name
```

Borrar un sinonimo:

```
DROP [PUBLIC] SYNONYM [schema.]synonym [FORCE]
```

Borrar un espacio de tablas:

```
DROP TABLESPACE tablespace_name [INCLUDING CONTENTS [AND  
DATAFILES]  
[CASCADE CONSTRAINTS]];
```

Borrar un disparador:

```
DROP TRIGGER [schema.]trigger
```

Borrar un usuario:

```
DROP USER username [CASCADE]
```

Borrar una vista:

```
DROP VIEW [schema.]view [CASCADE CONSTRAINTS]
```

Alter table

Sirve para cambiar la definición de una tabla. Podemos cambiar tanto columnas como restricciones (ver CONSTRAINTS).

La sintaxis es:

```
ALTER TABLE [esquema.]tabla {ADD|MODIFY|DROP}...
```

Añadir una columna a una tabla:

```
ALTER TABLE T_PEDIDOS ADD TEXTOPEDIDO Varchar2(35);
```

Cambiar el tamaño de una columna en una tabla:

```
ALTER TABLE T_PEDIDOS MODIFY TEXTOPEDIDO Varchar2(135);
```

Hacer NOT NULL una columna en una tabla:

```
ALTER TABLE T_PEDIDOS MODIFY (TEXTOPEDIDO NOT NULL);
```

Eliminar una columna a una tabla:

```
ALTER TABLE T_PEDIDOS DROP COLUMN TEXTOPEDIDO;
```

Valor por defecto de una columna:

```
ALTER TABLE T_PEDIDOS MODIFY TEXTOPEDIDO Varchar2(135)
DEFAULT 'ABC...';
```

Añade dos columnas:

```
ALTER TABLE T_PEDIDOS
ADD (SO_PEDIDOS_ID INT, TEXTOPEDIDO Varchar2(135));
```

Tipos de Constraints

Clave primaria

La clave primaria se utiliza para identificar en forma única cada línea en la tabla. Puede ser parte de un registro real, o puede ser un campo artificial (uno que no tiene nada que ver con el registro real). Una clave primaria puede consistir en uno o más campos en una tabla. Cuando se utilizan múltiples campos como clave primaria, se los denomina claves compuestas.

Las claves primarias pueden especificarse cuando se crea la tabla (utilizando CREATE TABLE) o cambiando la estructura existente de la tabla (utilizando ALTER TABLE).

Ejemplo para la especificación de una clave primaria cuando se crea una tabla:

```
CREATE TABLE CLIENTES
(SID integer PRIMARY KEY,
Last_Name varchar(30),
First_Name varchar(30));
```


A continuación se presentan ejemplos para la especificación de una clave primaria al modificar una tabla:

```
ALTER TABLE CLIENTES ADD PRIMARY KEY (SID);
```

Clave externa

Una clave externa es un campo (o campos) que señala la clave primaria de otra tabla. El propósito de la clave externa es asegurar la integridad referencial de los datos. En otras palabras, sólo se permiten los valores que se esperan que aparezcan en la base de datos.

Por ejemplo, digamos que tenemos dos tablas, una tabla CLIENTES que incluyen todos los datos del CLIENTES, y la tabla ÓRDENES que incluye los pedidos de CLIENTES.

La restricción aquí es que todos los pedidos deben asociarse con un CLIENTES que ya se encuentra en la tabla CUSTOMER. En este caso, colocaremos una clave externa en la tabla ÓRDENES y la relacionaremos con la clave primaria de la tabla CLIENTES. De esta forma, nos aseguramos que todos los pedidos en la tabla ÓRDENES estén relacionadas con un CLIENTES en la tabla CLIENTES. En otras palabras, la tabla ÓRDENES no puede contener información de un CLIENTES que no se encuentre en la tabla CLIENTES.

La estructura de estas dos tablas será la siguiente:

Tabla CLIENTES

nombre de columna	característica
SID	Clave Primaria
Last_Name	
First_Name	

Tabla ÓRDENES

nombre de columna	característica
-------------------	----------------

Order_ID	Clave Primaria
Order_Date	
Customer_SID	Clave Externa
Amount	

En el ejemplo anterior, la columna Customer_SID en la tabla ORDENES es una clave externa señalando la columna SID en la tabla CLIENTES.

Ejemplo de cómo especificar la clave externa a la hora de crear la tabla ÓRDENES:

```
CREATE TABLE ORDENES
(Order_ID integer primary key,
Order_Date date,
Customer_SID integer references CLIENTES (SID),
Amount double);
```

Ejemplo para la especificación de una clave externa al modificar una tabla: Esto asume que se ha creado la tabla ORDERS, y que la clave externa todavía no se ha ingresado:

```
ALTER TABLE ORDENES
ADD (CONSTRAINT fk_orders1) FOREIGN KEY (customer_sid)
REFERENCES CLIENTES (SID);
```

NOT NULL

En forma predeterminada, una columna puede ser NULL. Si no desea permitir un valor NULL en una columna, querrá colocar una restricción en esta columna especificando que NULL no es ahora un valor permitido.

Por ejemplo, en la siguiente instrucción,

```
CREATE TABLE clientes
(SID integer NOT NULL,
```

```
Last_Name varchar (30) NOT NULL,  
First_Name varchar(30));
```

Las columnas “SID” y “Last_Name” no incluyen NULL, mientras que “First_Name” puede incluir NULL.

UNIQUE

La restricción UNIQUE asegura que todos los valores en una columna sean distintos.

Por ejemplo, en la siguiente instrucción,

```
CREATE TABLE clientes  
(SID integer Unique,  
Last_Name varchar (30),  
First_Name varchar(30));
```

La columna “SID” no puede incluir valores duplicados, mientras dicha restricción no se aplica para columnas “Last_Name” y “First_Name”.

Por favor note que una columna que se especifica como clave primaria también puede ser única. Al mismo tiempo, una columna que es única puede o no ser clave primaria.

CHECK

La restricción CHECK asegura que todos los valores en una columna cumplan ciertas condiciones.

Por ejemplo, en la siguiente instrucción,

```
CREATE TABLE clientes  
(SID integer CHECK (SID > 0),  
Last_Name varchar (30),  
First_Name varchar(30));
```

La columna “SID” sólo debe incluir enteros mayores a 0.

Por favor note que la restricción **CHECK** no sea ejecutada por MySQL en este momento.

Modificar Constraints

Para cambiar las restricciones y la clave primaria de una tabla debemos usar ALTER TABLE.

Crear una clave primaria (primary key):

```
ALTER TABLE T_PEDIDOS ADD CONSTRAINT PK_PEDIDOS
PRIMARY KEY (numpedido,lineapedido);
```

Crear una clave externa, para integridad referencial (foreign key):

```
ALTER TABLE T_PEDIDOS ADD CONSTRAINT FK_PEDIDOS_CLIENTES
FOREIGN KEY (codcliente) REFERENCES T_CLIENTES
(codcliente));
```

Crear un control de valores (check constraint):

```
ALTER TABLE T_PEDIDOS ADD CONSTRAINT CK_ESTADO
CHECK (estado IN (1,2,3));
```

Crear una restricción UNIQUE:

```
ALTER TABLE T_PEDIDOS ADD CONSTRAINT UK_ESTADO
UNIQUE (correosid);
```

Normalmente una restricción de este tipo se implementa mediante un índice unico (ver CREATE INDEX).

Borrar una restricción:

```
ALTER TABLE T_PEDIDOS DROP CONSTRAINT CON1_PEDIDOS;
```

Deshabilita una restricción:

```
ALTER TABLE T_PEDIDOS DISABLE CONSTRAINT CON1_PEDIDOS;
```

Habilita una restricción:

```
ALTER TABLE T_PEDIDOS ENABLE CONSTRAINT CON1_PEDIDOS;
```

La sintaxis ALTER TABLE para restricciones es:

```
ALTER TABLE [esquema.]tabla
constraint_clause,...
[ENABLE enable_clause | DISABLE disable_clause]
[{ENABLE|DISABLE} TABLE LOCK]
```

```
[{ENABLE|DISABLE} ALL TRIGGERS];
```

Donde `constraint_clause` puede ser alguna de las siguientes entradas:

```
ADD out_of_line_constraint(s)

ADD out_of_line_referential_constraint

DROP PRIMARY KEY [CASCADE] [{KEEP|DROP} INDEX]

DROP UNIQUE (column,...) [{KEEP|DROP} INDEX]

DROP CONSTRAINT constraint [CASCADE]

MODIFY CONSTRAINT constraint constrnt_state

MODIFY PRIMARY KEY constrnt_state

MODIFY UNIQUE (column,...) constrnt_state

RENAME CONSTRAINT constraint TO new_name
```

Donde a su vez `constrnt_state` puede ser:

```
[[NOT] DEFERRABLE] [INITIALLY {IMMEDIATE|DEFERRED}]

[RELY | NORELY] [USING INDEX using_index_clause]

[ENABLE|DISABLE] [VALIDATE|NOVALIDATE]

[EXCEPTIONS INTO [schema.]table]
```

Borrar una restricción:

```
ALTER TABLE T_PEDIDOS DROP CONSTRAINT CON1_PEDIDOS;
```

Truncate

Quita todas las filas de una tabla sin registrar las eliminaciones individuales de filas. `TRUNCATE TABLE` es similar a la instrucción `DELETE` sin una cláusula `WHERE`; no obstante, `TRUNCATE TABLE` es más rápida y utiliza menos recursos de registros de transacciones y de sistema.

```
TRUNCATE TABLE [esquema.]tabla

[{PRESERVE|PURGE} MATERIALIZED VIEW LOG]

[{DROP | REUSE} STORAGE]

TRUNCATE CLUSTER [esquema.]cluster
```

```
[{DROP | REUSE} STORAGE]
```

La instrucción Truncate no necesita “Commit”

La instrucción truncate libera el espacio utilizado por los datos de la tabla.

Rename

Cambia el nombre a una tabla, vista, secuencia o sinonimo privado.

```
RENAME old TO new
```

Sentencias DCL (Lenguaje de control de datos)

Grant

Grant (dar permisos)

Esta sentencia sirve para dar permisos (o privilegios) a un usuario o a un rol.

Un permiso, en oracle, es un derecho a ejecutar un sentencia (system privileges) o a acceder a un objeto de otro usuario (object privileges).

La sintaxis es:

```
GRANT <ROL> TO <URUARIO / ROL/public > [WITH GRANT  
OPTION]
```

La opcion WITH GRANT OPTION otorga al usuario la posibilidad de otorgar el rol a otro usuario o rol.

El conjunto de permisos es fijo, esto quiere decir que no se pueden crear nuevos tipos de permisos.

Si un permiso se asigna a rol especial PUBLIC significa que puede ser ejecutado por todos los usuarios.

Permisos para acceder a la base de datos (permiso de sistema):

```
GRANT CREATE SESSION TO miusuario;
```

Permisos para usuario de modificación de datos (permiso sobre objeto):

```
GRANT SELECTS, INSERT, UPDATE, DELETE ON T_PEDIDOS TO  
miusuario;
```

Permisos de solo lectura para todos:

```
GRANT SELECT ON T_PEDIDOS TO PUBLIC;
```

Permisos de sistema (system privileges)

Los permisos del sistema pueden ser:

CREATE SESSION - Permite conectar a la base de datos

UNLIMITED TABLESPACE - Uso de espacio ilimitado del tablespace.

SELECT ANY TABLE - Consultas en tables, views, or mviews en cualquier esquema

UPDATE ANY TABLE - Actualizar filas en tables and views en cualquier esquema

INSERT ANY TABLE - Insertar filas en tables and views en cualquier esquema

Permisos de administrador para CREATE, ALTER o DROP:

Cluster, context, database, link, dimension, directory, index,
materialized view, operator, outline, procedure, profile, role,
rollback segment, sequence, session, synonym, table, tablespace,
trigger, type, user, view.

Los roles predefinidos son:

SYSDBA, SYSOPER, OSDBA, OSOPER, EXP_FULL_DATABASE,
IMP_FULL_DATABASE, SELECT_CATALOG_ROLE,
EXECUTE_CATALOG_ROLE, DELETE_CATALOG_ROLE,
AQ_USER_ROLE, AQ_ADMINISTRATOR_ROLE - manejo de la cola

SNMPAGENT - Agente inteligente.

RECOVERY_CATALOG_OWNER - rman

HS_ADMIN_ROLE - servicios heterogeneos

Los roles CONNECT, RESOURCE y DBA ya no deben usarse (aunque estan soportados).

Permisos sobre objetos (object privileges)

Los permisos sobre objetos mas importantes son: SELECT, UPDATE, INSERT, DELETE, ALTER, DEBUG, EXECUTE, INDEX, REFERENCES

```
GRANT object_priv [(column, column,...)]
    ON [schema.]object
    TO {user, | role, |PUBLIC} [WITH GRANT OPTION]
[WITH HIERARCHY OPTION]

GRANT ALL PRIVILEGES [(column, column,...)]
    ON [schema.]object
    TO {user, | role, |PUBLIC} [WITH GRANT OPTION]
[WITH HIERARCHY OPTION]

GRANT object_priv [(column, column,...)]
    ON DIRECTORY directory_name
    TO {user, | role, |PUBLIC} [WITH GRANT OPTION]
[WITH HIERARCHY OPTION]

GRANT object_priv [(column, column,...)]
    ON JAVA [RE]SOURCE [schema.]object
    TO {user, | role, |PUBLIC} [WITH GRANT OPTION]
[WITH HIERARCHY OPTION]
```

Con la opcion WITH HIERARCHY OPTION damos permisos sobre todos los subobjetos, incluso sobre los que se creen despues de ejecutar el GRANT.

Con la opción WITH GRANT OPTION damos permiso para que el que los recibe los pueda a su vez asignar a otros usuarios y roles.

La opción "GRANT ALL PRIVILEGES..." se puede escribir tambien como "GRANT ALL..."

Podemos obtener la lista de permisos de las tablas asi:

```
SELECT * FROM ALL_TAB_PRIVS_MADE;
```


Es posible asignar varios Object_Privs en un solo comando GRANT.

```
GRANT SELECT (empno), UPDATE (sal) ON pepe.tabla TO  
miusuario
```

Permisos del rol SYSDBA:

```
CREATE DATABASE  
  
CREATE SPFILE  
  
STARTUP and SHUTDOWN  
  
ALTER DATABASE: open, mount, back up, or change  
character set  
  
ARCHIVELOG and RECOVERY  
  
Includes the RESTRICTED SESSION privilege
```

Permisos del rol SYSOPER:

```
CREATE SPFILE  
  
STARTUP and SHUTDOWN  
  
ALTER DATABASE: open, mount, back up  
  
ARCHIVELOG and RECOVERY  
  
Includes the RESTRICTED SESSION privilege
```

Cada tipo de objeto tiene su propio conjunto de permisos:

Tables: select, insert, update, delete, alter, debug, flashback, on commit refresh, query rewrite, references, all

Views: select, insert, update, delete, under, references, flashback, debug

Sequence: alter, select

Packages, Procedures, Functions (Java classes, sources...): execute, debug

Materialized Views: delete, flashback, insert, select, update

Directories: read, write

Libraries: execute

User defined types: execute, debug, under

Operators: execute

Indextypes: execute

Revoke (quitar permisos)

Esta sentencia sirve para quitar permisos (o privilegios) a un usuario o a un rol.

No dejamos nada:

```
REVOKE ALL PRIVILEGES FROM miusuario;
```

Quitamos todo:

```
REVOKE ALL ON T_PEDIDOS FROM miusuario;
```

Sintaxis, quitar un rol asignado:

```
REVOKE role FROM {user, | role, |PUBLIC}
```

Quitar un permiso de sistema:

```
REVOKE system_priv(s) FROM {user, | role, |PUBLIC}
```

```
REVOKE ALL FROM {user, | role, |PUBLIC}
```

Quitar un permiso de objeto:

```
REVOKE object_priv [(column1, column2..)] ON  
[schema.]object
```

```
FROM {user, | role, |PUBLIC} [CASCADE  
CONSTRAINTS] [FORCE]
```

```
REVOKE object_priv [(column1, column2..)] ON  
[schema.]object
```

```
FROM {user, | role, |PUBLIC} [CASCADE  
CONSTRAINTS] [FORCE]
```

```
REVOKE object_priv [(column1, column2..)] ON DIRECTORY  
directory_name
```

```
FROM {user, | role, |PUBLIC} [CASCADE
CONSTRAINTS] [FORCE]
```

```
REVOKE object_priv [(column1, column2..)] ON JAVA
[RE]SOURCE [schema.]object
```

```
FROM {user, | role, |PUBLIC} [CASCADE
CONSTRAINTS] [FORCE]
```

La opción FORCE, quita todos los privilegios y descompila todos sus objetos.

Sentencias DML(Lenguaje de manipulación de datos)

Select

La selección sobre una tabla consiste en elegir un subconjunto de filas que cumplan (o no) algunas condiciones determinadas. La sintaxis de una sentencia de este tipo es la siguiente:

```
SELECT */ column1, columna2,....
FROM nombre-tabla
[WHERE condición]
[GROUP BY column1, columna2.... ]
[HAVING condición-selección-grupos ]
[ORDER BY column1 [DESC], columna2 [DESC]... ]
```

Si ejecutamos:

```
SELECT * FROM T_PEDIDOS;
```

Nos da la salida:

COD_PEDIDO	NOMBRE	ESTADO
1	JUAN	0
2	ANTONIO	1
3	PEPE	0
...		

* / columna1, columna2,... Si se escribe *, selecciona todas las columnas. Si se desea seleccionar sólo algunas columnas de la tabla, se debe poner los nombres de cada una de ellas, separadas por una coma.

nombre-tabla Nombre de la(s) tabla(s) de la(s) cual(es) se van a seleccionar los valores.

GROUP BY columna1, columna2....

Se utiliza para agrupar resultados por una determinada columna, específicamente cuando se utilizan funciones de columna y los resultados se desean obtener por grupos (SQL lanza un sort para generar los grupos).

HAVING condición-selección-grupos

Se utiliza con la cláusula “GROUP BY”, cuando se quiere poner condiciones al resultado de un grupo.

ORDER BY colum1 [DESC], colum2 [DESC...]

Sirve para ordenar el resultado. Todas las columnas por las que se desee realizar el orden tienen que encontrarse en la sentencia “Select” de la consulta. El orden de las columnas puede ser ascendente, (por omisión, ASC), o descendente, (DESC).

SENTENCIA SELECT (JOIN)

Consiste en la unión de campos de dos o más tablas. Dichas tablas tendrán por lo menos una columna común que sirva de nexo del join.

```
SELECT columna1, columna2,...  
  
FROM nombre-tabla1, nombre-tabla2
```

columna1, columna2,... Para diferenciar las columnas con el mismo nombre se antepondrá el nombre de la tabla a la que pertenecen, utilizando el punto como separador. Por ejemplo:

```
SELECT Tabla1.Columna2, Tabla2.Columna2, Columna3.....  
  
FROM Tabla1, Tabla2  
  
WHERE Tabla1.Columna1 = Tabla2.Columna1
```

La Columna1 de cada una de las tablas respectivas son las columnas de nexo o columnas de join.

SENTENCIA SELECT DISTINCT

Recupera las filas de una tabla eliminando los valores de la columna duplicados.

```
SELECT DISTINCT columna1, columna2, ....  
  
FROM nombre-tabla1, nombre-tabla2  
  
[GROUP BY columna1, columna2....]  
  
[HAVING condición-selección-grupos]  
  
[ORDER BY columna1 [DESC], columna2 [DESC]...]
```

SENTENCIA SELECT TOP N FILAS DE UNA TABLA

En Oracle⁸ⁱ podemos usar la sintaxis siguiente, con una cláusula ORDER BY, para elegir filas con los valores máximos o mínimos de un campo:

```
SELECT *  
  
FROM (SELECT * FROM my_table ORDER BY col_name_1 DESC)  
  
WHERE ROWNUM < 10;
```

FUNCIONES SOBRE COLUMNAS

COUNT. Indica el número de filas que cumplen una determinada condición, o el número de valores diferentes que posee una columna.

```
COUNT(*) o COUNT(DISTINCT columna)
```

SUM. Suma los valores de una columna.

```
SUM(columna)
```

AVG. Entrega la media de los valores de una columna.

```
AVG(columna)
```

MIN. Entrega el valor mínimo de una columna.

```
MIN(columna)
```

MAX. Entrega el valor máximo de una columna.

```
MAX(columna)
```

SUBSELECTS

Permite realizar comparaciones con valores obtenidos en otra sentencia select anidada, a la que se denomina “Subselect” o “Subselect interna”.

```
SELECT columna1 >, columna2, ....
```

```
FROM nombre-tabla1, nombre-tabla2

WHERE columna1 = (SELECT columna1

FROM nombre-tabla1, nombre-tabla2

WHERE condición)
```

(Cuando en la condición se pone el operador =, la subselect deberá recuperar un sólo registro).

Funciones estandar

Ver capítulo **Funciones estandar** PL-SQL

Insert

Insertando Filas

Inserciones una Sola Fila:

```
SQL> connect hr/hr

Connected.

SQL> insert into

2 departments(department_id, department_name, manager_id,

location_id)

3 values(300, 'Departamento 300', 100, 1800);

1 row created.

SQL> commit;

Commit complete.
```

Insertando Filas con Valores Nulos

Método Implícito: Se omiten las columnas que aceptan valores nulos.

```
SQL> insert into

2 departments(department_id, department_name)

3 values(301, 'Departamento 301');

1 row created.

SQL> commit;
```

Commit complete.

Método Explicito: Especificamos la palabra clave NULL en las columnas donde queremos insertar un valor nulo.

```
SQL> insert into departments
2 values(302, 'Departamento 302', NULL, NULL);
1 row created.
SQL> commit;
Commit complete.
```

Insertando Valores Especiales

```
SQL> insert into employees (employee_id,
2 first_name, last_name,
3 email, phone_number,
4 hire_date, job_id, salary,
5 commission_pct, manager_id,
6 department_id)
7 values(250,
8 'Gustavo', 'Coronel',
9 'gcoronel@miempresa.com', '511.481.1070',
10 sysdate, 'FI_MGR', 14000,
11 NULL, 102, 100);
1 row created.
SQL> commit;
Commit complete.
```

Insertando Valores Específicos de Fecha

```
SQL> insert into employees
2 values(251, 'Ricardo', 'Marcelo',
3 'rmarcelo@techsoft.com', '511.555.4567',
```

```
4 to_date('FEB 4, 2005', 'MON DD, YYYY'),
5 'AC_ACCOUNT', 11000, NULL, 100, 30);

1 row created.

SQL> commit;

Commit complete.
```

Usando & Sustitución para el Ingreso de Valores

```
SQL> insert into

2 departments (department_id, department_name,
3 location_id)

4 values (&department_id, '&department_name',
5 &location_id);

Enter value for department_id: 3003
Enter value for department_name: Departamento 303
Enter value for location_id: 2800

old 3: values (&department_id, '&department_name',
&location_id)

new 3: values (3003, 'Departamento 303', 2800)

1 row created.

SQL> commit;

Commit complete.
```

Copiando Filas Desde Otra Tabla

```
SQL> create table test

2 (

3 id number(6) primary key,

4 name varchar2(20),

5 salary number(8,2)

6 );

Table created.

SQL> insert into test (id, name, salary)

2 select employee_id, first_name, salary

3 from employees
```



```
4 where department_id = 30;

7 rows created.

SQL> commit;

Commit complete.
```

Insertando en Múltiples Tablas

Primero creamos las siguientes tablas: test50 y test80.

```
SQL> create table test50

2 (

3 id number(6) primary key,

4 name varchar2(20),

5 salary number(8,2)

6 );

Table created.

SQL> create table test80

2 (

3 id number(6) primary key,

4 name varchar2(20),

5 salary number(8,2)

6 );

Table created.
```

Luego limpiamos la tabla **test**.

```
SQL> delete from test;

rows deleted.

SQL> commit;

Commit complete.
```

Ahora procedemos a insertar datos en las tres tablas a partir de la tabla **employees**.

```
SQL> insert all

2 when department_id = 50 then
```

```

3 into test50 (id, name, salary)
4 values(employee_id, first_name, salary)
5 when department_id = 80 then
6 into test80 (id, name, salary)
7 values (employee_id, first_name, salary)
8 else
9 into test(id, name, salary)
10 values(employee_id, first_name, salary)
11 select department_id, employee_id, first_name, salary
12 from employees;
109 rows created.

SQL> commit;

Commit complete.

```

Update

Actualizando una Columna de una Tabla

Incrementar el salario de todos los empleados en 10%.

```

SQL> update employees
2 set salary = salary * 1.10;
109 rows updated.

SQL> Commit;

Commit complete.

```

Seleccionando las Filas a Actualizar

Ricardo Marcelo (Employee_id=251) ha sido trasladado de departamento de Compras (Department_id = 30) al departamento de Ventas (Department_id = 80).

```

SQL> select employee_id, first_name, department_id,
salary
2 from employees
3 where employee_id = 251;

```

```

EMPLOYEE_ID FIRST_NAME DEPARTMENT_ID SALARY
-----
251 Ricardo 30 12100

SQL> update employees

2 set department_id = 80

3 where employee_id = 251;

1 row updated.

SQL> select employee_id, first_name, department_id,
salary
2 from employees
3 where employee_id = 251;

EMPLOYEE_ID FIRST_NAME DEPARTMENT_ID SALARY
-----
251 Ricardo 80 12100

SQL> commit;

Commit complete.

```

Actualizando Columnas con Subconsultas

Gustavo Coronel (Employee_id = 250) ha sido trasladado al mismo departamento del empleado 203, y su salario tiene que ser el máximo permitido en su puesto de trabajo.

```

SQL> select employee_id, first_name, last_name,
department_id, job_id, salary
2 from employees
3 where employee_id = 250;

EMPLOYEE_ID FIRST_NAME LAST_NAME DEPARTMENT_ID JOB_ID
SALARY
-----
250 Gustavo Coronel 100 FI_MGR 15400

SQL> update employees

2 set department_id = (select department_id from
employees

```

```

3 where employee_id = 203),
4 salary = (select max_salary from jobs
5 where jobs.job_id = employees.job_id)
6 where employee_id = 250;

1 row updated.

SQL> commit;

Commit complete.

SQL> select employee_id, first_name, last_name,
department_id, job_id, salary

2 from employees

3 where employee_id = 250;

EMPLOYEE_ID FIRST_NAME LAST_NAME DEPARTMENT_ID JOB_ID
SALARY

-----
-----

250 Gustavo Coronel 40 FI_MGR 16000

```

Actualizando Varias Columnas con una Subconsulta

Asumiremos que tenemos la tabla **resumen_dept**, con la siguiente estructura:

Columna	Tipo Dato	Nulos	Descripcion
Department_id	Number(4)	No	Código de Departamento.
Emps	Number(4)	Si	Cantidad de Empleados en el departamento.
Planilla	Number(10,2)	Si	Emporte de la planilla en el departamento.

Esta tabla guarda la cantidad de empleados y el importe de la planilla por departamento.

Este script crea la tabla resumen_det e inserta los departamentos.

```
SQL> create table resumen_dept
2 (
3 department_id number(4) primary key,
4 emps number(4),
5 planilla number(10,2)
6 );
Table created.
SQL> insert into resumen_dept (department_id)
2 select department_id from departments;
31 rows created.
SQL> commit;
Commit complete.
```

Este script actualiza la tabla **resumen_dept**.

```
SQL> update resumen_dept
2 set (emps, planilla) = (select count(*), sum(salary)
3 from employees
4 where employees.department_id =
resumen_dept.department_id);
31 rows updated.
SQL> commit;
Commit complete.
```

Error de Integridad Referencial

```
SQL> update employees
2 set department_id = 55
3 where department_id = 110;
update employees
*
ERROR at line 1:
```

ORA-02291: integrity constraint (HR.EMP_DEPT_FK) violated
- parent key not found

Delete

Eliminar Todas la Filas de una Tabla

```
SQL> select count(*) from test;

COUNT(*)
-----
30

SQL> delete from test;

30 rows deleted.

SQL> commit;

Commit complete.

SQL> select count(*) from test;

COUNT(*)
-----
0
```

Seleccionando las Filas a Eliminar

Creando una tabla de prueba

```
SQL> create table copia_emp
2 as select * from employees;

Table created.
```

Eliminando una sola fila

```
SQL> delete from copia_emp
2 where employee_id = 190;

1 row deleted.

SQL> commit;

Commit complete.
```

Eliminando un grupo de filas

```
SQL> delete from copia_emp
```

```
2 where department_id = 50;
```

```
44 rows deleted.
```

```
SQL> commit;
```

```
Commit complete.
```

Eliminar los empleados que tienen el salario máximo en cada puesto de trabajo.

```
SQL> delete from copia_emp
```

```
2 where salary = (select max_salary from jobs
```

```
3 where jobs.job_id = copia_emp.job_id);
```

```
1 row deleted.
```

```
SQL> commit;
```

```
Commit complete.
```

Error de Integridad Referencial

```
SQL> delete from departments
```

```
2 where department_id = 50;
```

```
delete from departments
```

```
*
```

```
ERROR at line 1:
```

```
ORA-02292: integrity constraint (HR.EMP_DEPT_FK) violated
```

```
- child record found
```

Truncando una Tabla

```
SQL> select count(*) from copia_emp;
```

```
COUNT(*)
```

```
-----
```

```
64
```

```
SQL> truncate table copia_emp;
```

```
Table truncated.
```

```
SQL> select count(*) from copia_emp;
```

```
COUNT(*)
```

```
-----
```

```
0
```

Commit

Guarda los cambios de la transacción en curso.

Libera los recursos bloqueados por cualquier actualización hecha con la transacción actual (LOCK TABLE).

```
COMMIT [WORK] [COMMENT 'comment_text']  
  
COMMIT [WORK] [FORCE 'force_text' [,int] ]  
  
Si ejecutamos:  
  
DELETE FROM T_PEDIDOS WHERE COD_PEDIDO=15;  
  
COMMIT;
```

Borrar un registro y guarda los cambios.

Rollback

Deshace los cambios de la transacción en curso.

Libera los recursos bloqueados por cualquier actualización hecha con la transacción actual (LOCK TABLE).

```
ROLLBACK [WORK] [TO  
[SAVEPOINT] 'savepoint_text_identifier'];  
  
ROLLBACK [WORK] [FORCE 'force_text'];
```

Si ejecutamos:

```
DELETE FROM T_PEDIDOS WHERE COD_PEDIDO=15;  
  
COMMIT;
```

Borrar un registro pero cancela los cambios. Queda como si no hubiesemos hecho nada.

Savepoint

Sirve para marca un punto de referencia en la transacción para hacer un ROLLBACK parcial.

```
SAVEPOINT identificador;
```


Un ejemplo de uso es:

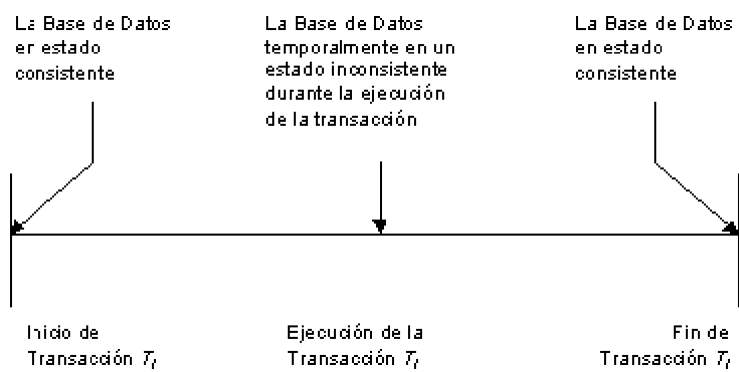
```
UPDATE T_PEDIDOS  
  
SET NOMBRE='jorge'  
  
WHERE CODPEDIDO=125;  
  
SAVEPOINT solouno;  
  
UPDATE T_PEDIDOS  
  
SET NOMBRE = 'jorge';  
  
SAVEPOINT todos;  
  
SELECT * FROM T_PEDIDOS;  
  
ROLLBACK TO SAVEPOINT todos;  
  
COMMIT;
```

Solo guardamos la primera modificación.

Transacciones

Una **transacción** es un grupo de acciones que hacen transformaciones consistentes en las tablas

preservando la consistencia de la base de datos. Una base de datos está en un estado **consistente** si obedece todas las restricciones de integridad definidas sobre ella. Los cambios de estado ocurren debido a actualizaciones, inserciones, y eliminaciones de información. Por supuesto, se quiere asegurar que la base de datos nunca entre en un estado de inconsistencia. Sin embargo, durante la ejecución de una transacción, la base de datos puede estar temporalmente en un estado inconsistente. El punto importante aquí es asegurar que la base de datos regresa a un estado consistente al fin de la ejecución de una transacción.



Lo que se persigue con el manejo de transacciones es por un lado tener una transparencia adecuada de las acciones concurrentes a una base de datos y por otro lado tener una transparencia adecuada en el manejo de las fallas que se pueden presentar en una base de datos.

Propiedades de una Transacción

Una transacción debe tener las propiedades ACID, que son las iniciales en inglés de las siguientes características: Atomicity, Consistency, Isolation, Durability.

Atomicidad

Una transacción constituye una unidad atómica de ejecución y se ejecuta exactamente una vez; o se realiza todo el trabajo o nada de él en absoluto.

Coherencia

Una transacción mantiene la coherencia de los datos, transformando un estado coherente de datos en otro estado coherente de datos. Los datos enlazados por una transacción deben conservarse semánticamente.

Aislamiento

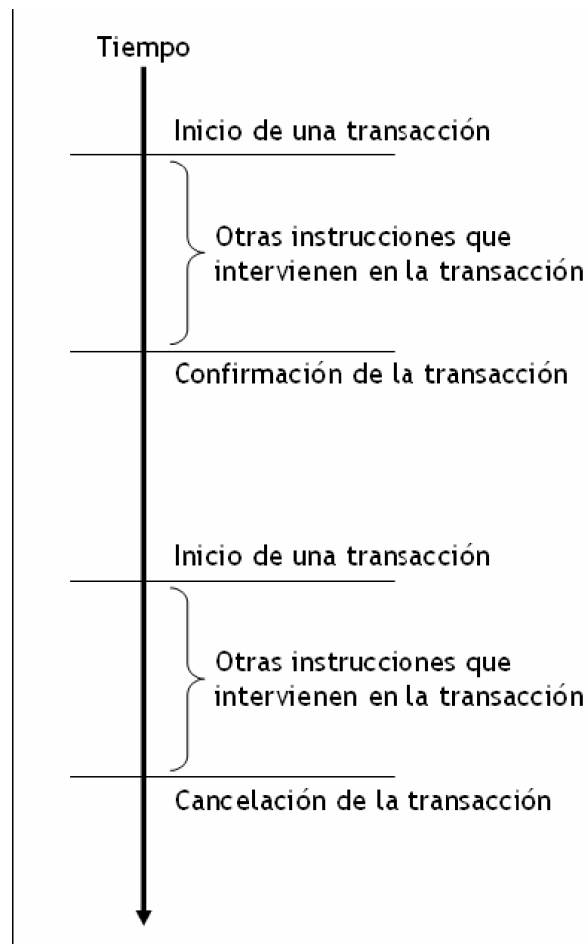
Una transacción es una unidad de aislamiento y cada una se produce aislada e independientemente de las transacciones concurrentes. Una transacción nunca debe ver las fases intermedias de otra transacción.

Durabilidad

Una transacción es una unidad de recuperación. Si una transacción tiene éxito, sus actualizaciones persisten, aun cuando falle el equipo o se apague. Si una transacción no tiene éxito, el sistema permanece en el estado anterior antes de la transacción.

Operación de Transacciones

El siguiente gráfico ilustra el funcionamiento de una transacción, cuando es confirmada y cuando es cancelada.



Inicio de una transacción

El inicio de una transacción es de manera automática cuando ejecutamos una sentencia insert, update, ó delete. La ejecución de cualquiera de estas sentencias da inicio a una transacción. Las instrucciones que se ejecuten a continuación formaran parte de la misma transacción.

Confirmación de una transacción

Para confirmar los cambios realizados durante una transacción utilizamos la sentencia commit.

Cancelar una transacción

Para cancelar los cambios realizados durante una transacción utilizamos la sentencia rollback.

Incrementar el salario al empleado Ricardo Marcelo (employee_id = 251) en 15%.

```
SQL> select employee_id, salary
```

```
2 from employees
```

```
3 where employee_id = 251;
```

```
EMPLOYEE_ID SALARY
```

```
-----
```

```
251 12100
```

```
SQL> update employees
```

```
2 set salary = salary * 1.15
```

```
3 where employee_id = 251;
```

```
1 row updated.
```

```
SQL> select employee_id, salary
```

```
2 from employees
```

```
3 where employee_id = 251;
```

```
EMPLOYEE_ID SALARY
```

```
-----
```

```
251 13915
```

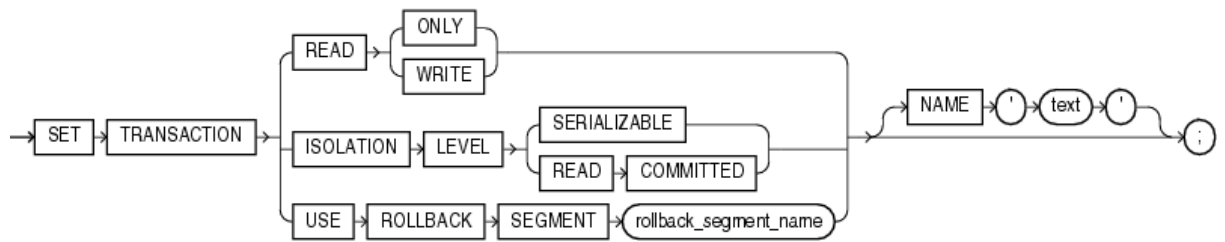
```
SQL> commit;
```

```
Commit complete.
```

Set transaction

SET TRANSACTION inicia una transaccion como “read-only (solo lectura)” o “read-write” (escritura / lectura), establece el “isolation level”, y opcionalmente asigna un segment de rollback para la. Las transacciones de “Solo Lectura (Read-only) son usadas, normalmente, para ejecutar multiples consultas sobre una o mas tablas, mientras otros usuarios modifican los datos de las tablas.

La sintaxis del comando es:



READ ONLY

Establece la transacción como “solo lectura” de forma que las subsiguientes consultas verán solo los datos confirmados antes de la instrucción “Set transaction”.

READ WRITE

Establece la transacción como “Lectura-Escritura”, el uso de READ-WRITE, no afecta a otros usuarios. Si la transacción ejecuta manipulación de datos, Oracle asigna un segmento de rollback para la transacción.

ISOLATION LEVEL

Especifica el comportamiento de bloqueo de la transacción para la conexión

SERIALIZABLE: If a serializable transaction tries to execute a SQL data manipulation statement that modifies any table already modified by an uncommitted transaction, the statement fails.

To enable **SERIALIZABLE** mode, your DBA must set the Oracle initialization parameter **COMPATIBLE** to 7.3.0 or higher.

READ COMMITTED: Si la transacción de manipulación de datos requiere un bloqueo de filas que están actualmente bloqueadas por otro usuario, la transacción espera hasta que las filas sean liberadas.

USE ROLLBACK SEGMENT

Asigna a la transacción un segmento de rollback y establece la transacción como “Read-Write”, no se puede usar esta opción con “Read – Only”

NAME

Establece el nombre o comentario para la transacción.

Notas:

SET TRANSACTION debe ser la primera sentencia en la transacción y puede aparecer solamente una vez en la transacción.

Ejemplo:

```
DECLARE

    daily_order_total    NUMBER(12,2);

    weekly_order_total   NUMBER(12,2);

    monthly_order_total  NUMBER(12,2);

BEGIN

    COMMIT; -- finalize la transacción anterior

    SET TRANSACTION READ ONLY NAME 'Calculando total de
transacciones';

    SELECT SUM (order_total) INTO daily_order_total FROM
orders

        WHERE order_date = SYSDATE;

    SELECT SUM (order_total) INTO weekly_order_total FROM
orders

        WHERE order_date = SYSDATE - 7;

    SELECT SUM (order_total) INTO monthly_order_total FROM
orders

        WHERE order_date = SYSDATE - 30;

    COMMIT; -- finalize la transacción "read-only"
transaction

END;

/
```

Restricciones de SET TRANSACTION

Solo las sentencias SELECT INTO, OPEN, FETCH, CLOSE, LOCK TABLE, COMMIT, ROLLBACK pueden utilizarse en transacciones "Read-Only".

Administración básica de Oracle

Instalación de Oracle 10G Standard/Enterprise Edition

Oracle es un sistema de gestión de base de datos relacional (RDBMS Relational Data Base Management System), fabricado por Oracle Corporation. Oracle es uno de los sistemas de bases de datos más completos.

A continuación os explicamos cómo instalar y configurar las opciones básicas de Oracle 10g en Windows XP (válido para cualquier versión de Windows: Windows 2000, Windows 2003, etc).

En primer lugar descargaremos el fichero de instalación de Oracle, desde su página web (es gratuito siempre que no se utilice con fines comerciales) , el enlace es:

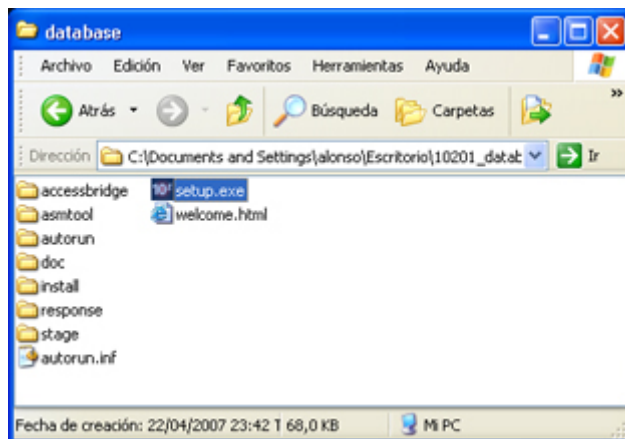
<http://www.oracle.com/technology/software/products/database/oracle10g/index.html>

Seleccionaremos el enlace:

Oracle Database 10g Release 2 (10.2.0.1.0) for Microsoft Windows

o el correspondiente a la versión de Windows que se utilice. Será necesario ser usuario registrado de Oracle (es gratuito) para realizar la descarga. Se descargará el fichero 10201_database_win32.zip, de unos 655 MB.

Una vez descargado el fichero de instalación y descomprimido, ejecutaremos el fichero "setup.exe" para iniciar el programa de instalación:



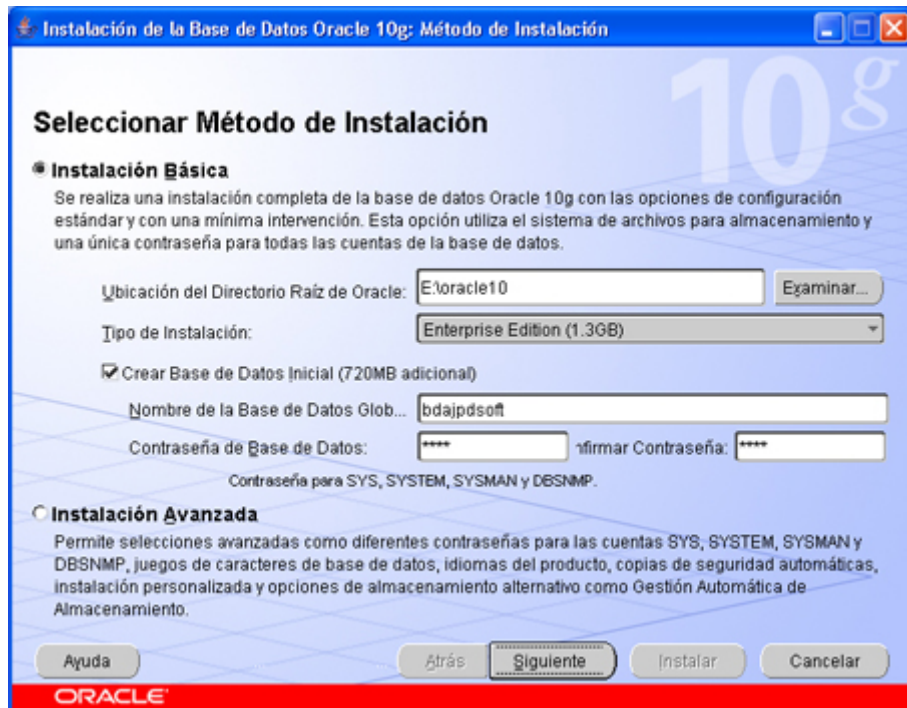
Seleccionaremos el método de instalación, en nuestro caso "Instalación Básica", al seleccionar este método nos pedirá unos datos:

- **Instalación Básica:** seleccione este método de instalación si desea instalar rápidamente la base de datos Oracle 10g. Este método necesita una intervención mínima del usuario. Instala el software y, opcionalmente, crea una base de datos de uso general con el esquema SAMPLE y el tablespace EXAMPLE, con la información especificada en la pantalla inicial. Nota: Si no especifica toda la información necesaria, Installer muestra las pantallas de instalación avanzada. Utiliza los valores especificados como valores por defecto en las pantallas correspondientes.
- **Instalación Avanzada:** seleccione este método de instalación para cualquiera de las siguientes tareas: realizar una instalación personalizada del software o seleccionar una configuración diferente de la base de datos. Instalar Oracle Real Application Clusters. Actualizar una base de datos existente. Seleccionar un juego de caracteres de la base de datos o idiomas de producto diferentes. Crear una base de datos en otro sistema de archivos del software. Configurar la gestión automática de almacenamiento (ASM) o utilizar dispositivos raw para el almacenamiento en la base de datos. Especificar contraseñas diferentes para esquemas administrativos. Configurar copias de seguridad automáticas o notificaciones de Oracle Enterprise Manager.

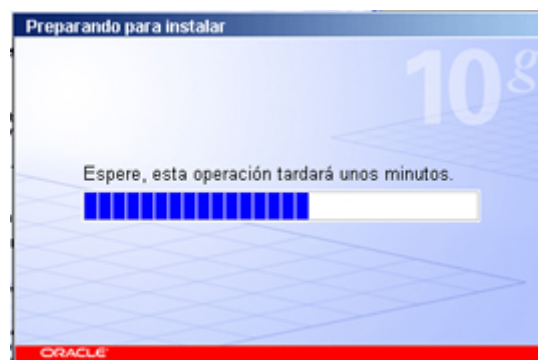
Para la Instalación Básica, las opciones son:

- **Ubicación del Directorio Raíz de Oracle:** unidad y carpeta donde se realizará la instalación de Oracle 10g.
- **Tipo de instalación:**
 - **Enterprise Edition:** este tipo de instalación está diseñado para aplicaciones a nivel de empresa. Está diseñado para el Procesamiento de Transacciones en Línea (OLTP) de alta seguridad y de importancia crítica y para entornos de almacenes de datos. Si selecciona este tipo de instalación, se instalan todas las opciones de Enterprise Edition con licencias independientes.
 - **Standard Edition:** este tipo de instalación está diseñado para aplicaciones a nivel de departamento o grupo de trabajo o para pequeñas y medianas empresas. Está diseñado para proporcionar las opciones y servicios de gestión de bases de datos relacionales esenciales. Si selecciona este tipo de instalación, deberá adquirir licencias adicionales para instalar otras opciones de Enterprise Edition.
 - **Personal Edition** (sólo para Sistemas Operativos Windows): este tipo de instalación instala el mismo software que el tipo de instalación Enterprise Edition, pero sólo soporta un entorno de desarrollo y despliegue monousuario que debe ser totalmente compatible con Enterprise Edition y Standard Edition.
- **Crear base de datos inicial:** creará una base de datos de uso general durante la instalación. Si no la selecciona, Installer sólo instala el software. Si no desea crear una base de datos durante la instalación, puede utilizar el Asistente de Configuración de Bases de Datos (DBCA) para crearla después de instalar el software.
 - **Nombre de la Base de Datos Global:** nombre con el que se identificará la base de datos, máximo 8 caracteres.
 - **Contraseña de Base de Datos:** contraseña que se asignará a los usuarios SYS, SYSTEM, SYSMAN y DBSNMP.

Tras rellenar estos datos pulsaremos "Siguiente" para continuar con la instalación (en nuestro caso hemos seleccionado Instalación Básica):



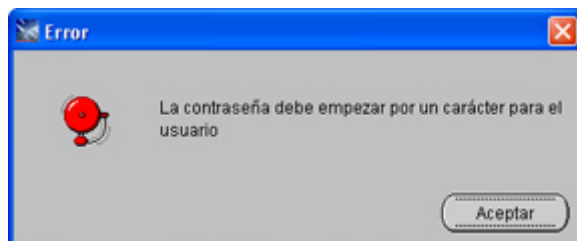
Nos mostrará una barra de progreso indicando que se está preparando para instalar:



El asistente de instalación verificará si el entorno cumple todos los requisitos mínimos para instalar y configurar los productos seleccionados. Si hay algún elemento marcado con advertencia se deberá comprobar manualmente. Pulsaremos "Siguiente" para continuar:



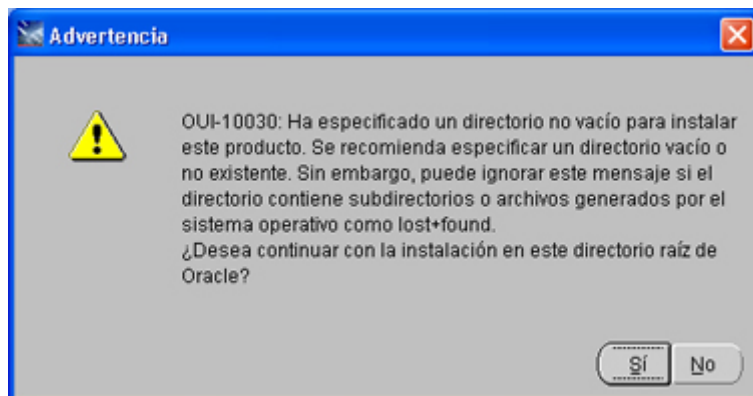
En nuestro caso, hemos introducido una contraseña que no cumple las políticas de seguridad, nos indica que la contraseña debe empezar por un carácter para el usuario, pulsaremos "Aceptar":



y volveremos a introducir una contraseña válida



Nos mostrará una aviso indicando que el directorio donde se va a instalar Oracle no está vacío, pulsaremos "Sí" para continuar:

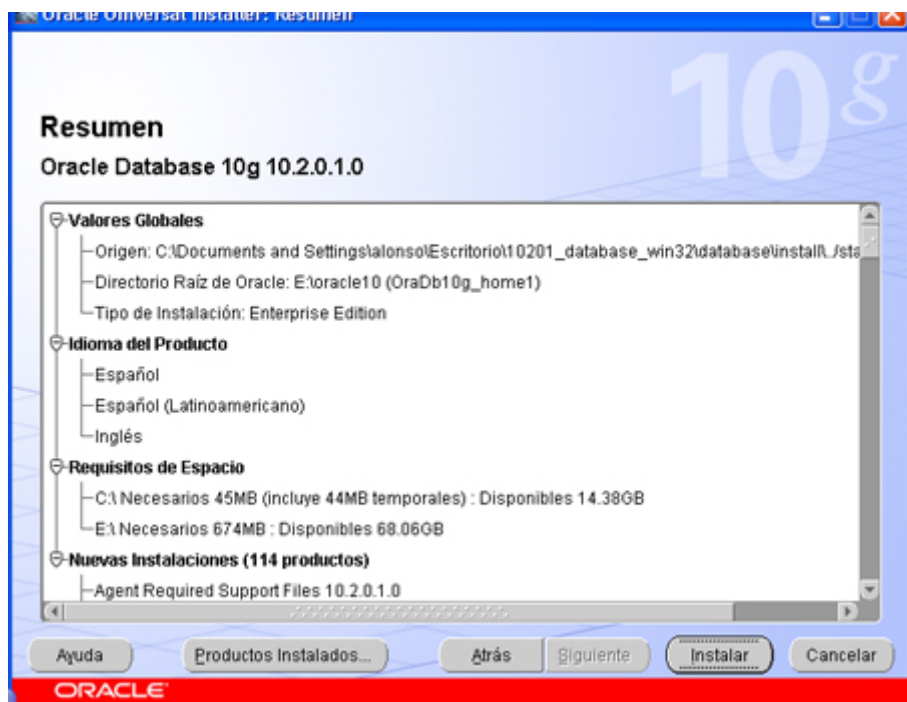


Con el texto: *OUI-10030: ha especificado un directorio no vacío para instalar este producto. Se recomienda especificar un directorio vacío o no existente. Sin embargo, puede ignorar este mensaje si el directorio contiene subdirectorios o archivos generados por el sistema operativo como lost+found. ¿Desea continuar con la instalación en este directorio raíz de Oracle?*

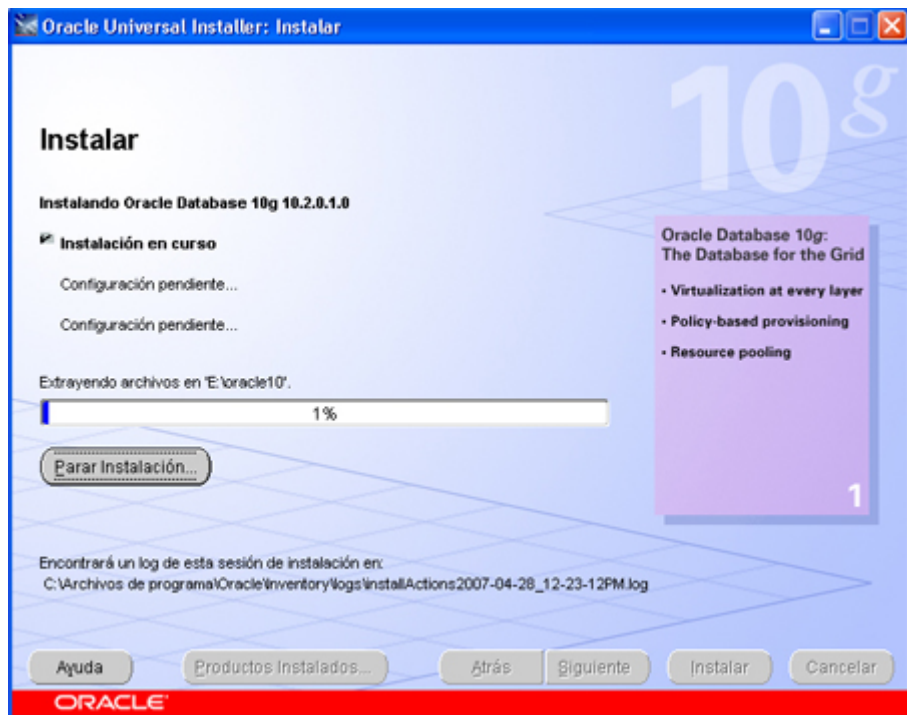
Nos volverá a mostrar la ventana de comprobaciones de requisitos específicos del producto, pulsaremos "Siguiente" para continuar:



Nos mostrará una ventana indicando los productos Oracle Database 10g 10.2.0.1.0 que se instalarán. Pulsaremos "Instalar" para iniciar la instalación:



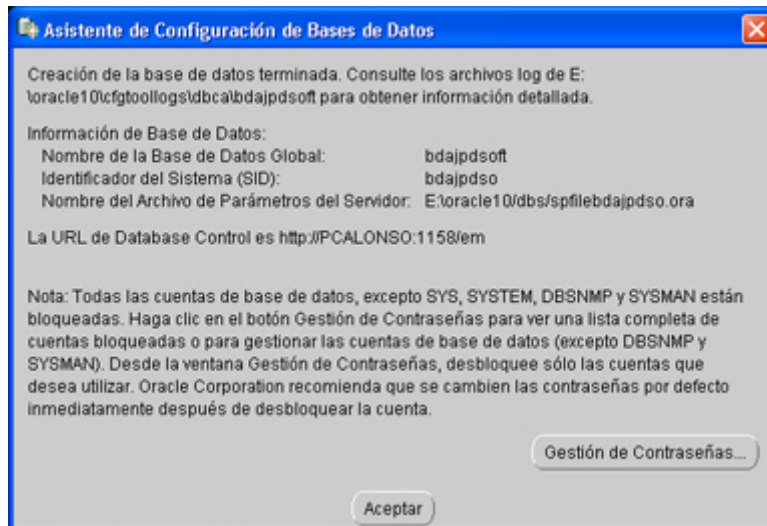
Nos mostrará una ventana con el progreso de la instalación



Si hemos marcado la opción de creación de la base de datos nos mostrará el progreso de este proceso:

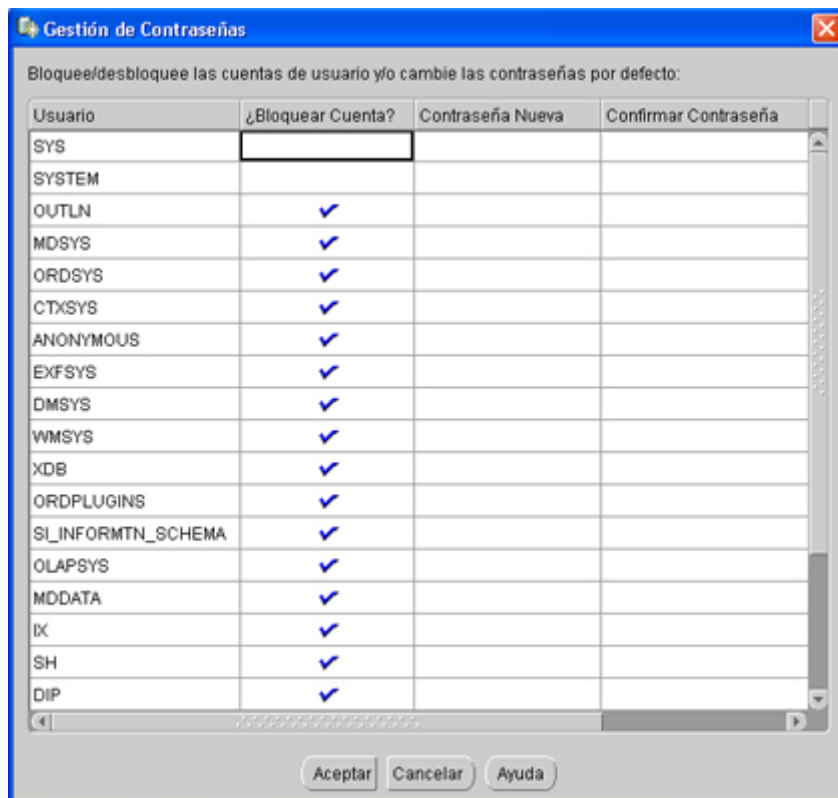


Una vez creada la base de datos nos mostrará una ventana para poder gestionar las contraseñas de cada usuario, también podremos activarlos o desactivarlos, para ello pulsaremos en "Gestión de Contraseñas":



Con el texto: creación de la base de datos terminada. Consulte los archivos de log de E:/oracle10/cfgtoollogs/dbca/bdajpdsoft para obtener información detallada. La URL de Database Control es http://PCALONSO:1158/em. Nota: todas las cuentas de base de datos, excepto SYS, SYSTEM, DBSNMP y SYSMAN están bloqueadas. Haga clic en el botón Gestión de Contraseñas para ver una lista completa de cuentas bloqueadas o para gestionar las cuentas de base de datos (excepto DBSNMP y SYSMAN). Desde la ventana Gestión de Contraseñas, desbloquee sólo las cuentas que desea utilizar. Oracle Corporation recomienda que se cambien las contraseñas por defecto inmediatamente después de desbloquear la cuenta.

Si hemos pulsado en Gestión de Contraseñas en la ventana anterior nos mostrará esta otra ventana, donde podremos activar/desactivar los usuarios que Oracle crea por defecto y cambiar sus contraseñas (OUTLN, MDSYS, ORDSYS, CTXSYS, ANONYMOUS, EXFSYS, DMSYS, WMSYS, XDB, ORDPLUGINS, SI_INFORMTN_SCHEMA, OLAPSYS, MDDATA, IX, SH, DIP, etc):



Tras la instalación, el asistente mostrará una serie de datos. Pulsaremos en "Salir" para terminar la instalación:



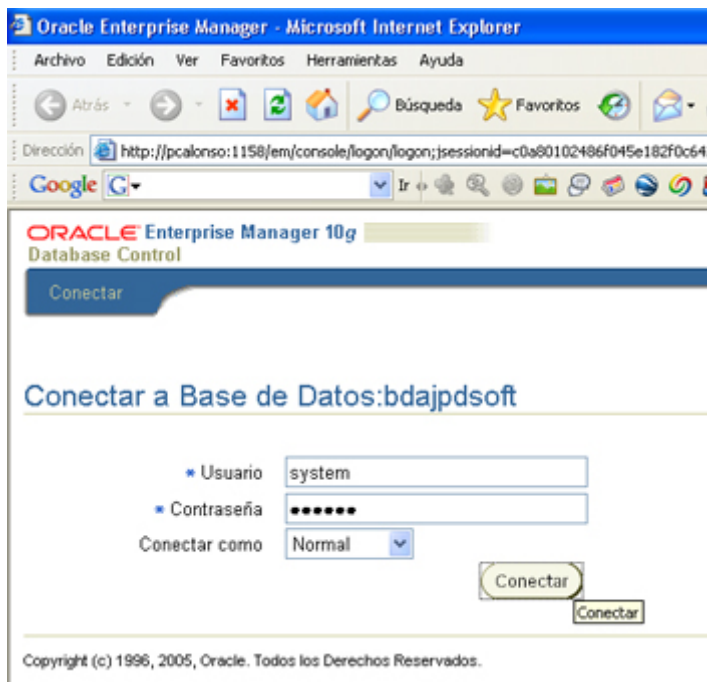
Pulsaremos en "Sí" para cerrar el asistente de instalación de Oracle Database 10g:



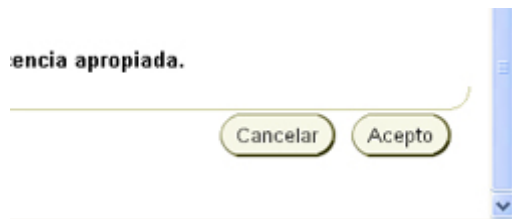
Para probar y configurar Oracle Database 10g, a diferencia de otras versiones (que era una aplicación gráfica), ahora se configura desde el navegador de Internet (Internet Explorer, Mozilla Firefox, etc). Abriremos nuestro navegador de Internet y pondremos la siguiente dirección URL:

http://nombre_pc:1158/em

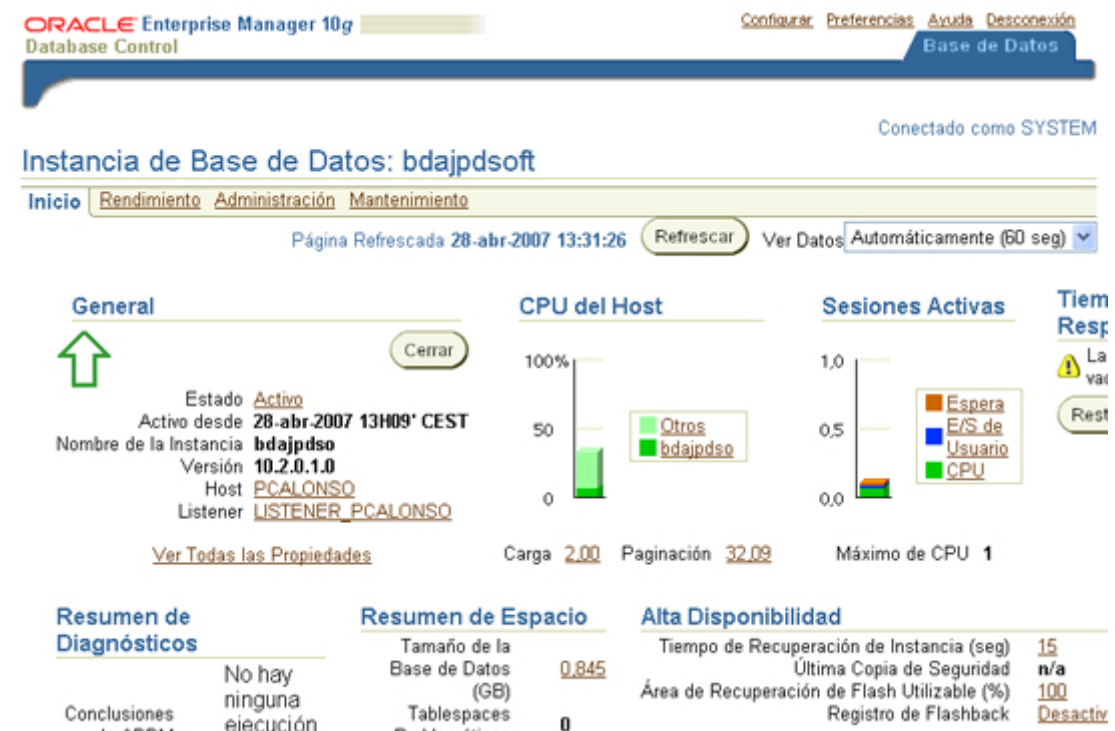
Donde habrá que cambiar (lógicamente) "PCALONSO" por la IP o el nombre del equipo donde hayamos instalado Oracle 10g. Tras abrir esta URL del nuevo Oracle Enterprise Manager 10g (Database Control) nos mostrará una web como la siguiente, donde deberemos introducir el usuario y la contraseña para el acceso, en nuestro caso utilizaremos el usuario "system". Pulsaremos el botón "Conectar":



La primera vez que ejecutamos Database Control nos pedirá que leamos y aceptemos el acuerdo de licencia. Si estamos de acuerdo con los términos de la licencia pulsaremos en "Acepto":



El nuevo Oracle Enterprise Manager 10g Database Control nos mostrará una ventana inicial con una especie de cuadro de mandos, con las estadísticas de uso de la CPU, las sesiones activas, el estado de la base de datos, versión, host, listener, el nombre de la instancia, estadísticas sobre rendimiento de la base de datos, resumen de espacio, resumen de diagnósticos, etc:



Podremos ver estadísticas de rendimiento en tiempo real pulsando en "Rendimiento":

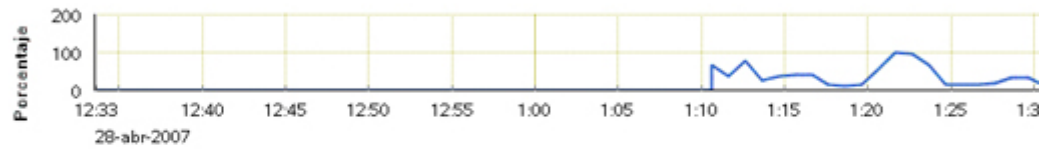
Instancia de Base de Datos: bdajpdsoft

Inicio Rendimiento Administración Mantenimiento

Visualización mejorada con el último plugin SVG

Ver Datos Tiempo Real: Refrescamiento Manual

Host



Media de Sesiones Activas

Ejecutar ADDM Ahora



Podremos administrar las opciones de configuración de la base de datos pulsando en "Administración" (archivos de control, tablespaces, grupos de tablespaces temporales, archivos de datos, segmentos de rollback, grupos de redo logs, archive logs, parámetros de memoria, gestión de deshacer, todos los parámetros de inicialización, uso de funciones de la base de datos, trabajos, cadenas, planificaciones, programas, clases de trabajos, ventanas, grupos de ventanas, atributos globales, repositorio de carga de trabajo automática, gestionar estadísticas del optimizador, migrar a ASM, gestionar tablespace locamente, monitores, grupos de consumidores, asignaciones de grupos de consumidores, etc):

Instancia de Base de Datos: bdajpds01

[Inicio](#) [Rendimiento](#) [Administración](#) [Mantenimiento](#)

El separador Administración muestra enlaces que le permiten administrar objetos de base de datos e iniciar operaciones de base de datos en una base de datos Oracle. El separador Mantenimiento muestra enlaces que proporcionan funciones que controlan los datos entre o fuera de bases de datos Oracle.

Administración de la Base de Datos

Almacenamiento

[Archivos de Control](#)
[Tablespaces](#)
[Grupos de Tablespaces Temporales](#)
[Archivos de Datos](#)
[Segmentos de Rollback](#)
[Grupos de Redo Logs](#)
[Archive Logs](#)

Gestión de Estadísticas

[Repositorio de Carga de Trabajo](#)
[Automática](#)
[Gestionar Estadísticas del Optimizador](#)

Políticas

[Biblioteca de Políticas](#)
[Violaciones de Política](#)

Configuración de la Base de Datos

[Parámetros de Memoria](#)
[Gestión de Deshacer](#)
[Todos los Parámetros de Inicialización](#)
[Uso de Funciones de la Base de Datos](#)

Cambiar Base de Datos

[Migrar a ASM](#)
[Gestionar Tablespace Localmente](#)

Planificador de Base de Datos

[Trabajos](#)
[Cadenas](#)
[Planificaciones](#)
[Programas](#)
[Clases de Trabajos](#)
[Ventanas](#)
[Grupos de Ventanas](#)
[Atributos Globales](#)

Gestor de Recursos

[Monitores](#)
[Grupos de Consumidores](#)
[Asignaciones de Grupos de Consumidores](#)
[Planes](#)

Esquema

Por ejemplo, si pulsamos sobre "Archivos de control" (en la ventana anterior) podremos ver y cambiar los archivos de control de la base de datos Oracle. Por defecto, el asistente de instalación habrá creado tres (CONTROL01.CTL, CONTROL02.CTL y CONTROL03.CTL):

ORACLE Enterprise Manager 10g Database Control [Configurar](#) [?](#)

Instancia de Base de Datos: bdajpds01 > Archivos de Control

Archivos de Control

[General](#) [Avanzado](#) [Sección de Registros](#)

[Copiar](#)

Imágenes Duplicadas de Archivo de Control

Oracle recomienda que la base de datos tenga un mínimo de dos archivos de control en discos diferentes. Si se daña un disco de control debido a un fallo en disco, se podría restaurar utilizando la copia intacta del archivo de control del otro disco. Puede especificar las ubicaciones en el archivo de parámetros de inicialización de la base de datos.

Válido	Nombre de Archivo	Directorio de Archivos
VALID	CONTROL01.CTL	E:\ORADATA\BDAJPDS01\
VALID	CONTROL02.CTL	E:\ORADATA\BDAJPDS01\
VALID	CONTROL03.CTL	E:\ORADATA\BDAJPDS01\

[General](#) [Avanzado](#) [Sección de Registros](#)

Oracle recomienda que la base de datos tenga un mínimo de dos archivos de control en discos diferentes. Si se daña un disco de control debido a un fallo en disco, se podría restaurar utilizando la copia intacta del archivo de control del otro disco. Puede especificar las ubicaciones en el archivo de parámetros de inicialización de la base de datos.

También podremos ver y configurar los Tablespaces, pulsando en "Tablespaces". Por defecto, el instalador de Oracle 10g crea los siguientes tablespaces:

- EXAMPLE
- SYSAUX
- SYSTEM
- TEMP
- UNDOTBS1
- USERS

Base de Datos: bdaipdsoft > Tablespaces Conectado como SYSTEM

Tablespaces

Tipo de Objeto: Tablespace

Seleccione un tipo de objeto y, opcionalmente, introduzca un nombre de objeto para filtrar los datos que aparecerán en el juego de resultados. Nombre del Objeto:

Ir

Por defecto, la búsqueda devuelve todas las coincidencias en mayúsculas que comienzan por la cadena introducida. Para ejecutar una búsqueda de coincidencia exacta entre mayúsculas/minúsculas, introduzca la cadena de búsqueda entre comillas. Puede utilizar el carácter comodín (%) en la cadena entrecomillada.

Modo de Selección: Simple Crear

Editar Vista Suprimir Acciones Agregar Archivo de Datos Ir

Seleccionar	Nombre	Tamaño (MB)	Usado (MB)	Usado (%)	Libre (MB)	Estado	Archivos de Datos	Tipo	Gestión de Extensiones	Gestión de Segmentos
<input checked="" type="radio"/>	EXAMPLE	100,0	77,4	<div><div></div></div> 77,4	22,6	✓	1	PERMANENT	LOCAL	AUTO
<input type="radio"/>	SYSAUX	230,0	228,7	<div><div></div></div> 99,4	1,3	✓	1	PERMANENT	LOCAL	AUTO
<input type="radio"/>	SYSTEM	480,0	472,5	<div><div></div></div> 98,4	7,5	✓	1	PERMANENT	LOCAL	MANUAL
<input type="radio"/>	TEMP	20,0	0,0	<div><div></div></div> 0,0	20,0	✓	1	TEMPORARY	LOCAL	MANUAL
<input type="radio"/>	UNDOTBS1	30,0	0,6	<div><div></div></div> 2,1	29,4	✓	1	UNDO	LOCAL	MANUAL
<input type="radio"/>	USERS	5,0	3,2	<div><div></div></div> 65,0	1,8	✓	1	PERMANENT	LOCAL	AUTO

Espacio Total (MB): **865,0**
 ✓ Online
 ✗ Offline
 🔍 Sólo Lectura

Espacio Libre (MB): **787,4**

Los archivos de datos de la base de datos, podremos verlos y configurarlos pulsando en "Archivos de datos". Por defecto, el instalador de Oracle Database 10g creará los siguientes: example01.dbf, sysaux01.dbf, system01.dbf, temp01.dbf, undotb.dbf, users01.dbf.

chivos de Datos

Tipo de Objeto Archivo de Datos

Buscar

Seleccione un tipo de objeto y, opcionalmente, introduzca un nombre de objeto para filtrar los datos que aparecerán en el juego resultados.

Nombre del Objeto



Por defecto, los archivos de datos realizan búsquedas sensibles a mayúsculas/minúsculas. Para ejecutar una búsqueda de coincidencia exacta, introduzca una cadena de búsqueda entre comillas. Puede utilizar el carácter comodín (%) en la cadena entrecomillada.

<div> <div>Editar</div> <div>Vista</div> <div>Suprimir</div> <div>Acciones</div> <div>Crear como</div> </div>						
Seleccionar	Nombre de Archivo	Tablespace	Estado	Tamaño (MB)	Usado (MB)	Usado (%)
<input checked="" type="radio"/>	E:\ORADATA\BDA\JPDSO\EXAMPLE01.DBF	EXAMPLE	ONLINE	100,000	77,375	<div></div>
<input type="radio"/>	E:\ORADATA\BDA\JPDSO\SYSAUX01.DBF	SYSAUX	ONLINE	230,000	228,688	<div></div>
<input type="radio"/>	E:\ORADATA\BDA\JPDSO\SYSTEM01.DBF	SYSTEM	SYSTEM	480,000	472,500	<div></div>
<input type="radio"/>	E:\ORADATA\BDA\JPDSO\TEMP01.DBF	TEMP	ONLINE	20,000	18,000	<div></div>
<input type="radio"/>	E:\ORADATA\BDA\JPDSO\UNDOTBS01.DBF	UNDOTBS1	ONLINE	30,000	0,438	<div></div>
<input type="radio"/>	E:\ORADATA\BDA\JPDSO\USERS01.DBF	USERS	ONLINE	5,000	3,250	<div></div>

Por ejemplo, podremos ampliar el tamaño asignado a un fichero de datos, lo seleccionaremos en la lista anterior (Seleccionar) y pulsaremos el botón "Editar". En la nueva ventana que aparece podremos poner el fichero de datos en modo Online o Offline y modificar el tamaño en "Tamaño del Archivo". También podremos indicarle a Oracle que incremente el tamaño del archivo de datos automáticamente, marcando "Ampliar automáticamente el archivo de datos cuando esté lleno (AUTOEXTEND):

Directorio de Archivos E:\ORADATA\BDA\JPDSO\

Tablespace EXAMPLE

Estado ☒ Online ☐ Offline

Tamaño del Archivo 102400 KB

Almacenamiento

☒ Ampliar automáticamente el archivo de datos cuando esté lleno (AUTOEXTEND)

Incremento 640 KB

Tamaño Máximo de Archivo ☐ Ilimitado

☒ Valor 3000 MB

Acciones

Crear como

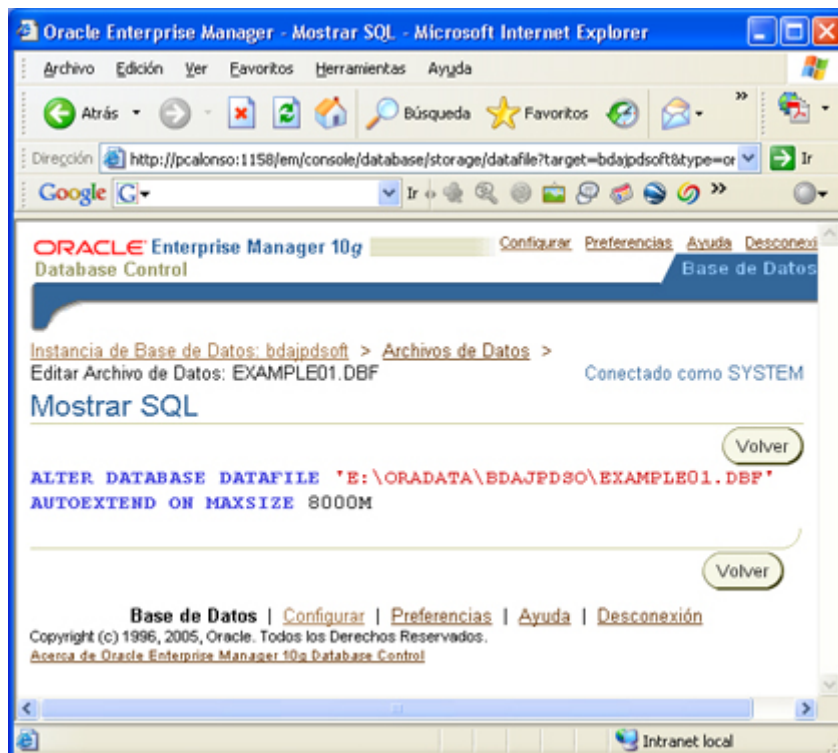


Mostrar SQL

Revertir

Aplicar

Si queremos ver la consulta SQL que se ejecutará para realizar la acción anterior (ampliar el tamaño de un archivo de datos) pulsaremos en "Mostrar SQL" (de la ventana anterior). Para ver el SQL resultante de incrementar el tamaño del un archivo de datos pulsa aquí:



En la sección "Mantenimiento" podremos: planificar copias de seguridad, realizar recuperaciones, gestionar copias de seguridad actuales, gestionar puntos de restauración, ver informes de copia de seguridad, valores de copia de seguridad, valores de recuperación, valores del catálogo de recuperación, exportar archivos de exportación, importar archivos de exportación, importar base de datos, cargar datos de archivos de usuario, clonar base de datos, etc:

Instancia de Base de Datos: bdajpdsoft

[Inicio](#) [Rendimiento](#) [Administración](#) [Mantenimiento](#)

El separador Administración muestra enlaces que le permiten administrar objetos de base de datos e iniciar operaciones de base de datos en una base de datos Oracle. El separador Mantenimiento muestra enlaces que proporcionan funciones de mantenimiento de base de datos entre o fuera de bases de datos Oracle.

Alta Disponibilidad

Copia de Seguridad/Recuperación

[Planificar Copia de Seguridad](#)
[Realizar Recuperación](#)
[Gestionar Copias de Seguridad](#)
[Actuales](#)
[Gestionar Puntos de Restauración](#)
[Informes de Copia de Seguridad](#)

Valores de Copia de Seguridad/Recuperación

[Valores de Copia de Seguridad](#)
[Valores de Recuperación](#)
[Valores del Catálogo de Recuperación](#)

Movimiento de Datos

Mover Datos de Fila

[Exportar a Archivos de Exportación](#)
[Importar de Archivos de Exportación](#)
[Importar de Base de Datos](#)
[Cargar Datos de Archivos de Usuario](#)
[Controlar Trabajos de Importación y Exportación](#)

Mover Archivos de Base de Datos

[Clonar Base de Datos](#)
[Transportar Tablespaces](#)

Flujos

[Configuración](#)
[Gestión](#)

También podremos iniciar el nuevo iSQL*Plus, en la parte inferior de la ventana, en "Enlaces Relacionados" aparecerá "iSQL*Plus":

Trabajos planificados para iniciarse hace no más de 7 días

Ejecuciones Planificadas 0Ejecuciones en Ejecución 0Ejecuciones

[Inicio](#) [Rendimiento](#) [Administración](#) [Mantenimiento](#)

Enlaces Relacionados

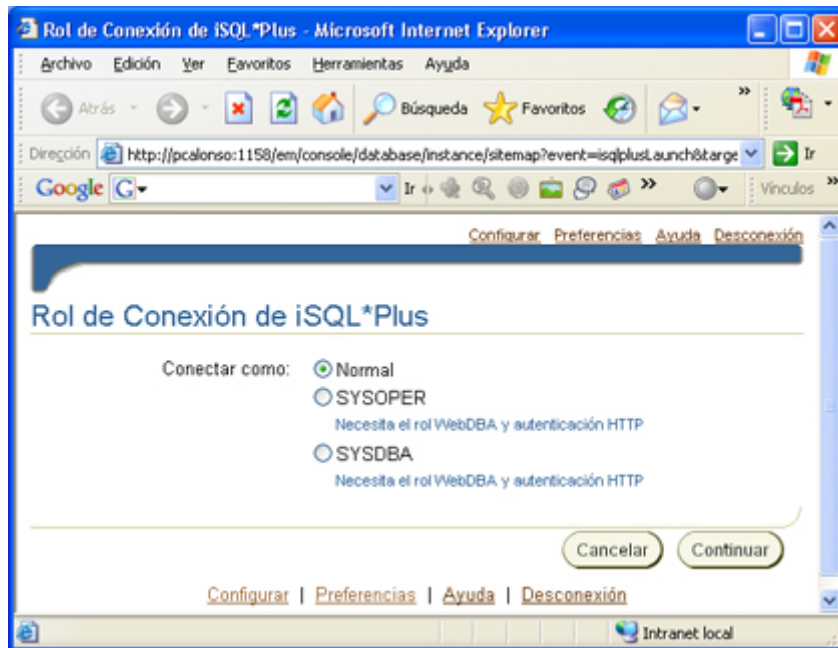
Central de Asesores	Configuración de Control
Controlar Modo de Acceso a Memoria	Errores de Recopilación
Historial de Alertas	Historial SQL
iSQL*Plus	Líneas Base de Métricas
Todas las Métricas	Trabajos

Base de Datos | [Configurar](#) | [Preferencias](#)

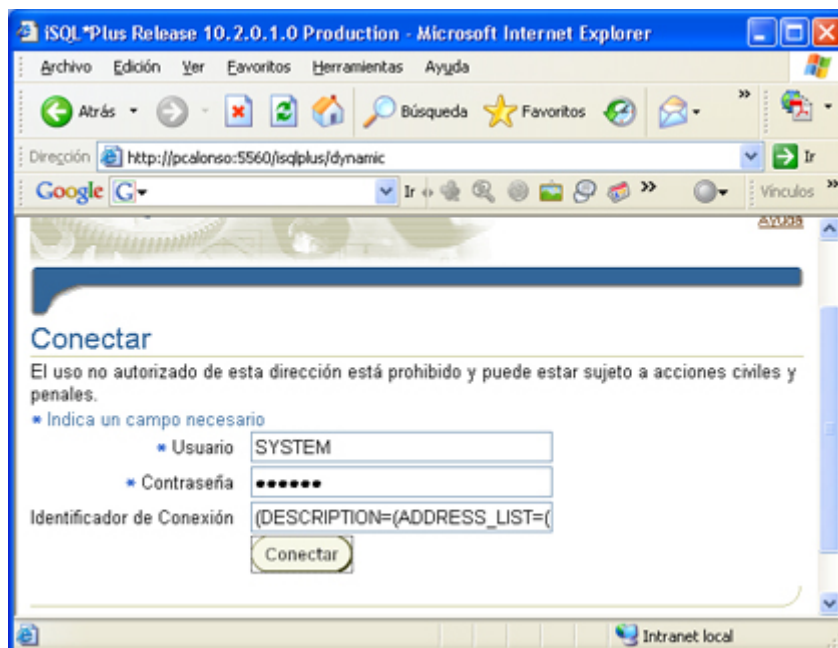
Copyright (c) 1996, 2005, Oracle. Todos los Derechos Reservados.

[Acerca de Oracle Enterprise Manager 10g Database Control](#)

Para conectarnos a iSQL*Plus marcaremos "Normal" salvo que queramos realizar alguna acción que requiera "SYSOPER" o "SYSDBA", pulsaremos "Continuar":

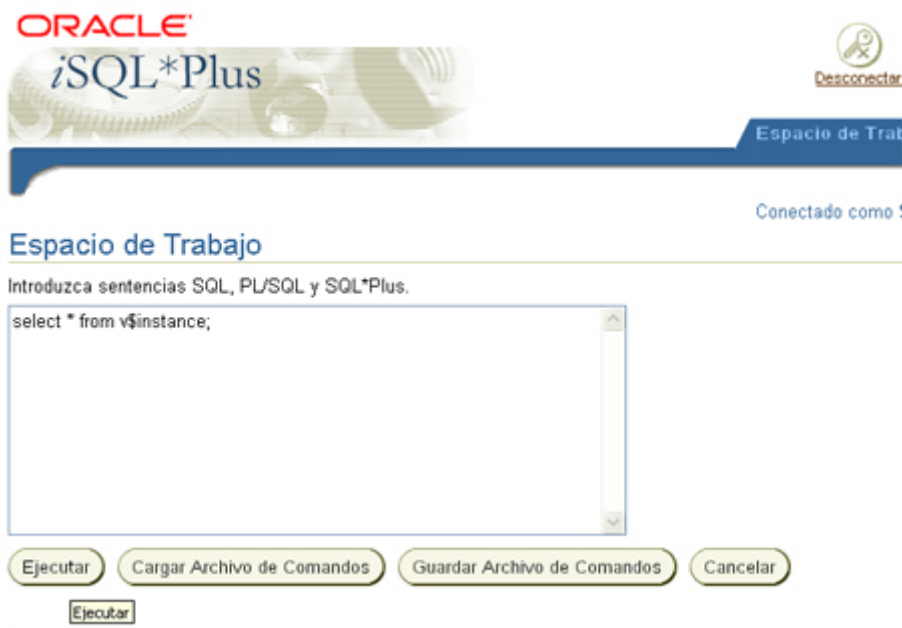


Introduciremos usuario y contraseña para iniciar sesión con iSQL*Plus. nos creará automáticamente el Identificador de Conexión:



Por ejemplo, podremos ejecutar cualquier consulta SQL, como:

*select * from v\$instance;*



En la parte inferior mostrará el resultado de la ejecución de la consulta SQL:

INSTANCE_NUMBER	INSTANCE_NAME	HOST_NAME	VERSION	STARTUP_	STATUS	PARALLEL	THREAD#	ARCHIV
1	bdajpdso	PCALONSO	10.2.0.1.0	28/04/07	OPEN	NO	1	STOPPE

Otro ejemplo:

select status, name from v\$datafile;

Mostrará como resultado todos los archivos de datos de la base de datos:

STATUS	NAME
SYSTEM	E:\ORADATA\BDAJPDSON\SYSTEM01.DBF
ONLINE	E:\ORADATA\BDAJPDSON\UNDOTBS01.DBF
ONLINE	E:\ORADATA\BDAJPDSON\SYSAU01.DBF
ONLINE	E:\ORADATA\BDAJPDSON\USERS01.DBF
ONLINE	E:\ORADATA\BDAJPDSON\EXAMPLE01.DBF

Instalar Oracle Database 10g XE Express Edition en Windows XP

Esta nueva versión *gratuita* está dirigida a estudiantes, pequeñas empresas y desarrolladores que quieran embeberla junto con sus aplicaciones. Esta versión

limitada de Oracle 10g sólo podrá correr en servidores con 1 sólo procesador y con hasta 1 Gb de RAM, y podrá manejar un tamaño máximo de 4 Gb de almacenamiento en el disco.

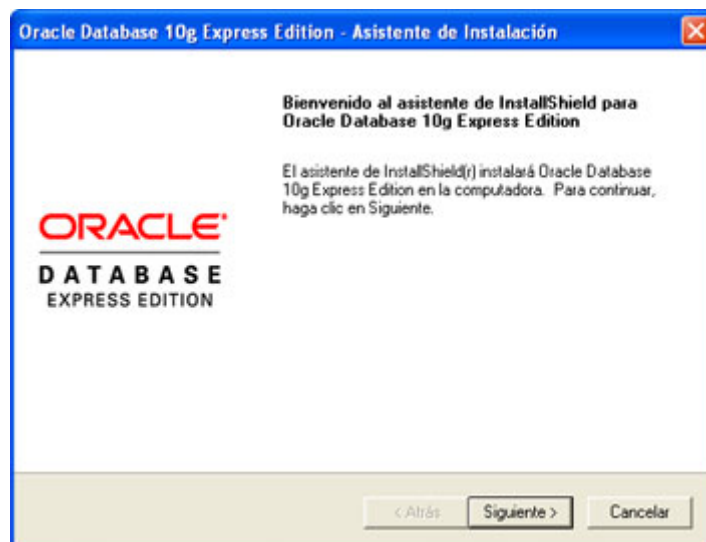
A continuación os mostramos, paso a paso, cómo instalar Oracle 10g Express Edition en un PC con Windows XP:

1. Descargaremos el fichero OracleXE.exe de la web de Oracle:

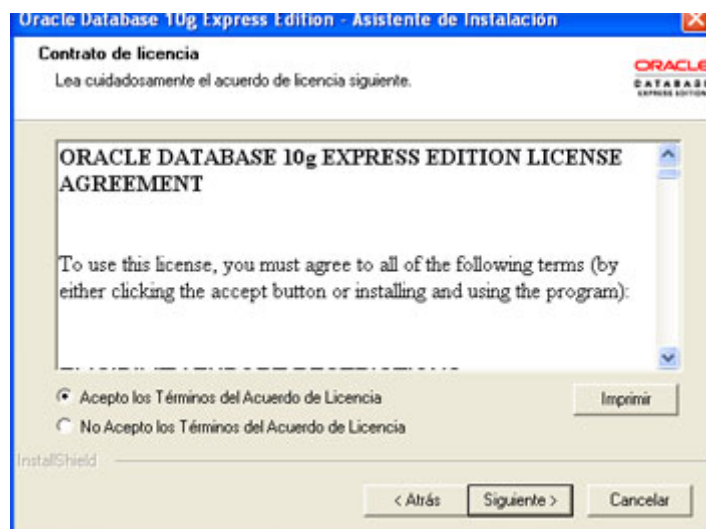
<http://www.oracle.com/technology/software/products/database/xs/index.html>

(necesitaremos ser usuarios registrados de Oracle, el registro es gratuito)

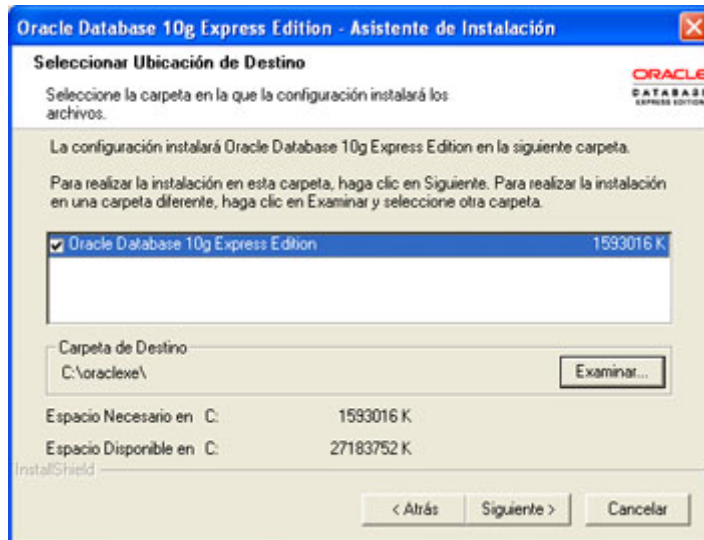
2. Ejecutaremos el fichero descargado, pulsaremos *Siguiente* para iniciar la instalación:



3. Aceptaremos el contrato de licencia y pulsaremos *Siguiente*:

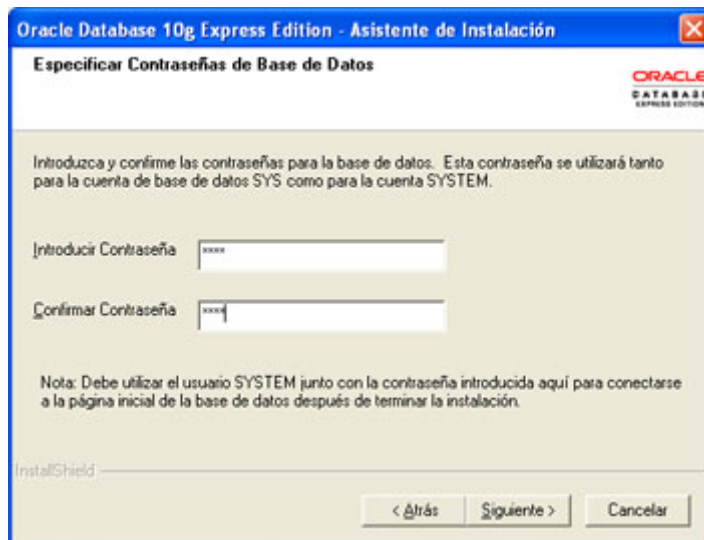


4. Marcaremos *Oracle Database 10g Express Edition* y especificaremos la ruta de instalación de Oracle, pulsando el botón *Examinar* podremos cambiar la ruta por defecto: *C:/oraclexe*:

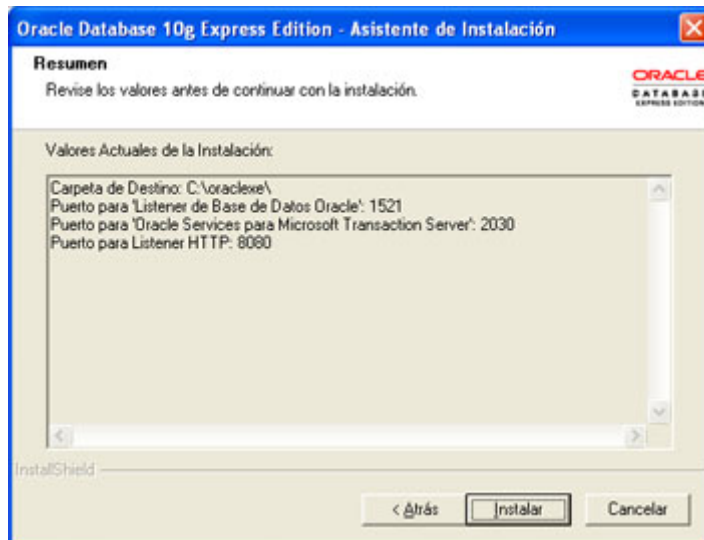


Nota: necesitará un espacio mínimo de 1,6 GB.

5. Introduciremos la contraseña para el usuario *sys* y para el usuario *SYSTEM* y pulsaremos *Siguiente*:



6. A continuación aparecerá una ventana con las opciones de instalación elegidas, pulsaremos *Instalar* para iniciar el proceso:



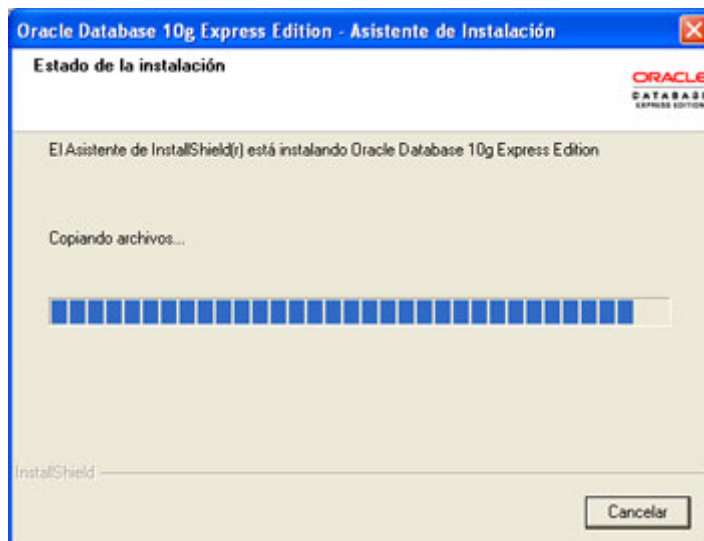
Carpeta de Destino: C:/oraclexe/

Puerto para 'Listener de Base de Datos Oracle': 1521

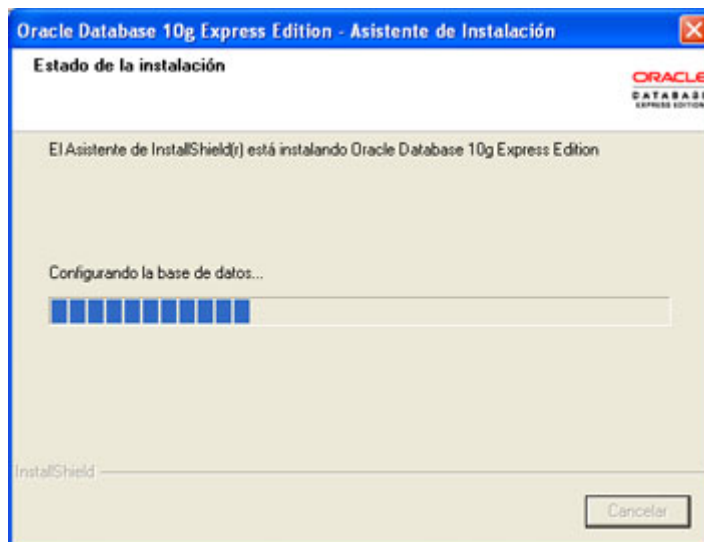
Puerto para 'Oracle Services para Microsoft Transaction Server': 2030

Puerto para Listener HTTP: 8080

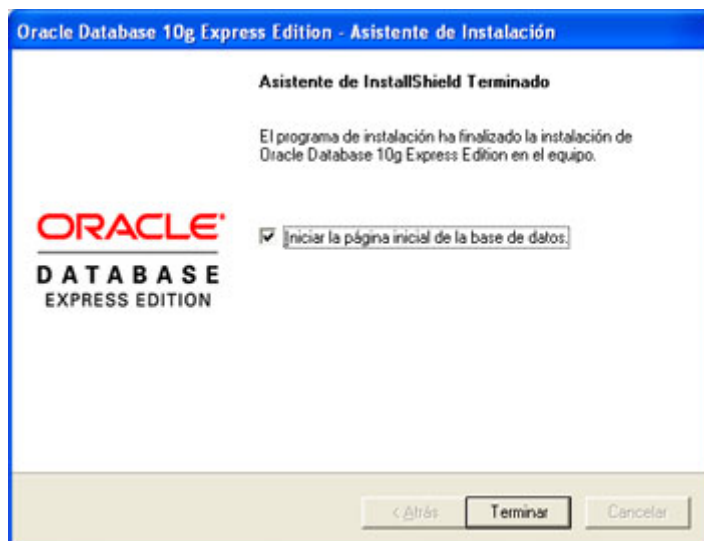
Se iniciar el proceso de copia de ficheros:



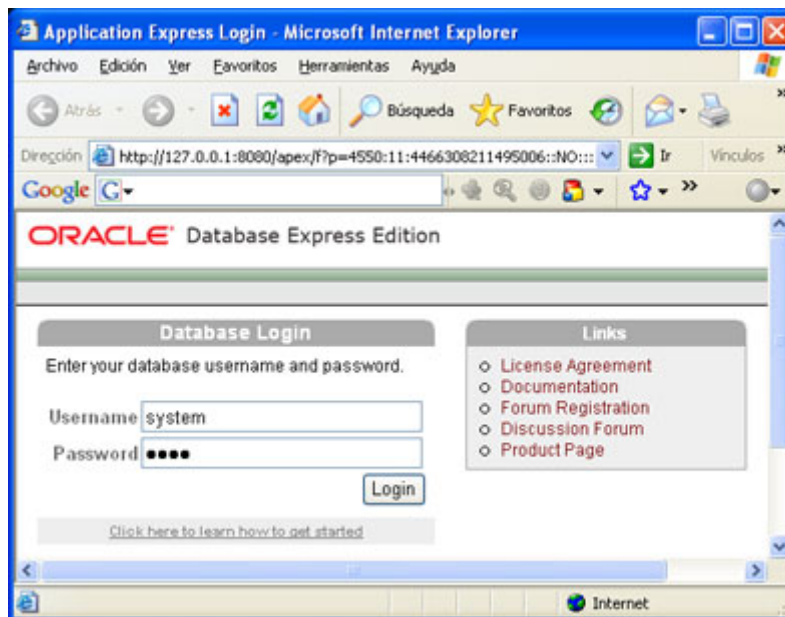
y el proceso de configuración automática de la base de datos. Por defecto, el instalador de Oracle 10g Express Edition, crea y configura una base de datos:



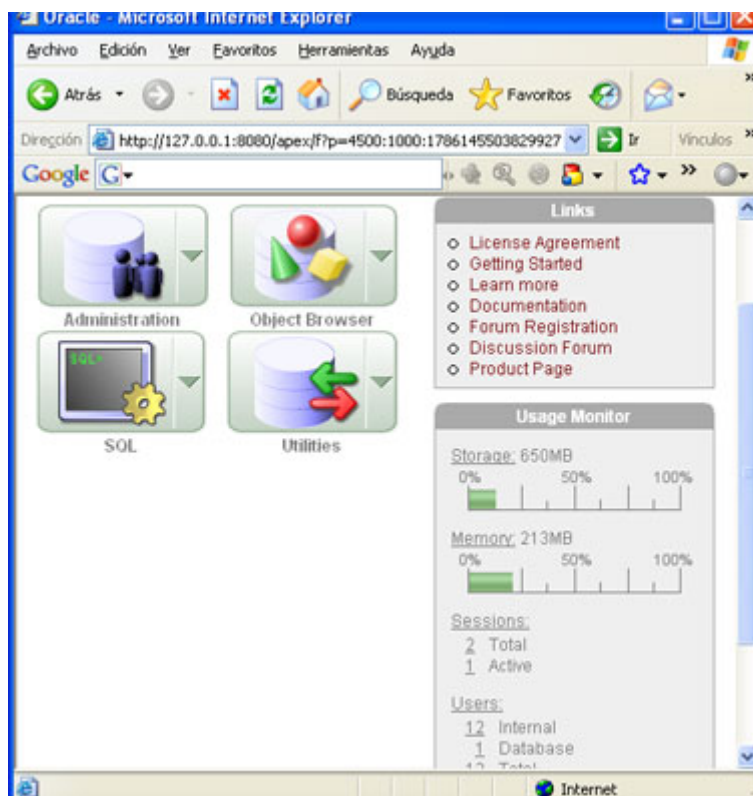
7. Tras la finalización del proceso de creación de la base de datos, el asistente permite iniciar la página de configuración de la base de datos, lo dejaremos chequeado y pulsaremos en *Terminar*:



Tras unos segundos nos aparecerá esta página web para administrar Oracle 10g Express Edition, accesible introduciendo en el explorador de Internet: <http://127.0.0.1:8080/apex>. En *Username* introduciremos el nombre del usuario (*system* ó *sys*) y en *Password* introduciremos la contraseña especificada en el paso 5 de este manual:



Nos aparecerá una ventana de administración (limitada con respecto a las versiones completas) con varias opciones: Administration (para configurar las opciones de almacenamiento, memoria, usuarios y monitorización), Object Browser (para visualizar, modificar y crear tablas, vistas, índices, funciones, triggers, procedimientos, paquetes, secuencias, etc), SQL (para ejecutar consultas SQL, scripts, etc), Utilities (exportación, importación, papelera de reciclaje, informes, generación de sentencias DDL, etc):



El programa de instalación de Oracle 10g Express Edition habrá creado los siguientes servicios:

Nombre	Ubicación	Inicio
OracleJobSchedulerXE	c:/oraclexe/app/oracle/product/10.2.0/server/Bin/extjob.exe XE	Deshabilitado
OracleMTSRecoveryService	C:/oraclexe/app/oracle/product/10.2.0/server/BIN/omtsreco.exe "OracleMTSRecoveryService"	Manual
OracleService-XE	c:/oraclexe/app/oracle/product/10.2.0/server/bin/ORACLE.EXE XE	Automático
OracleXEClrAgent	C:/oraclexe/app/oracle/product/10.2.0/server/bin/OraClrAgnt.exe	Manual
OracleXETNSListener	C:/oraclexe/app/oracle/product/10.2.0/server/BIN/tnslsnr.exe	Automático

Los accesos directos que crea la aplicación:

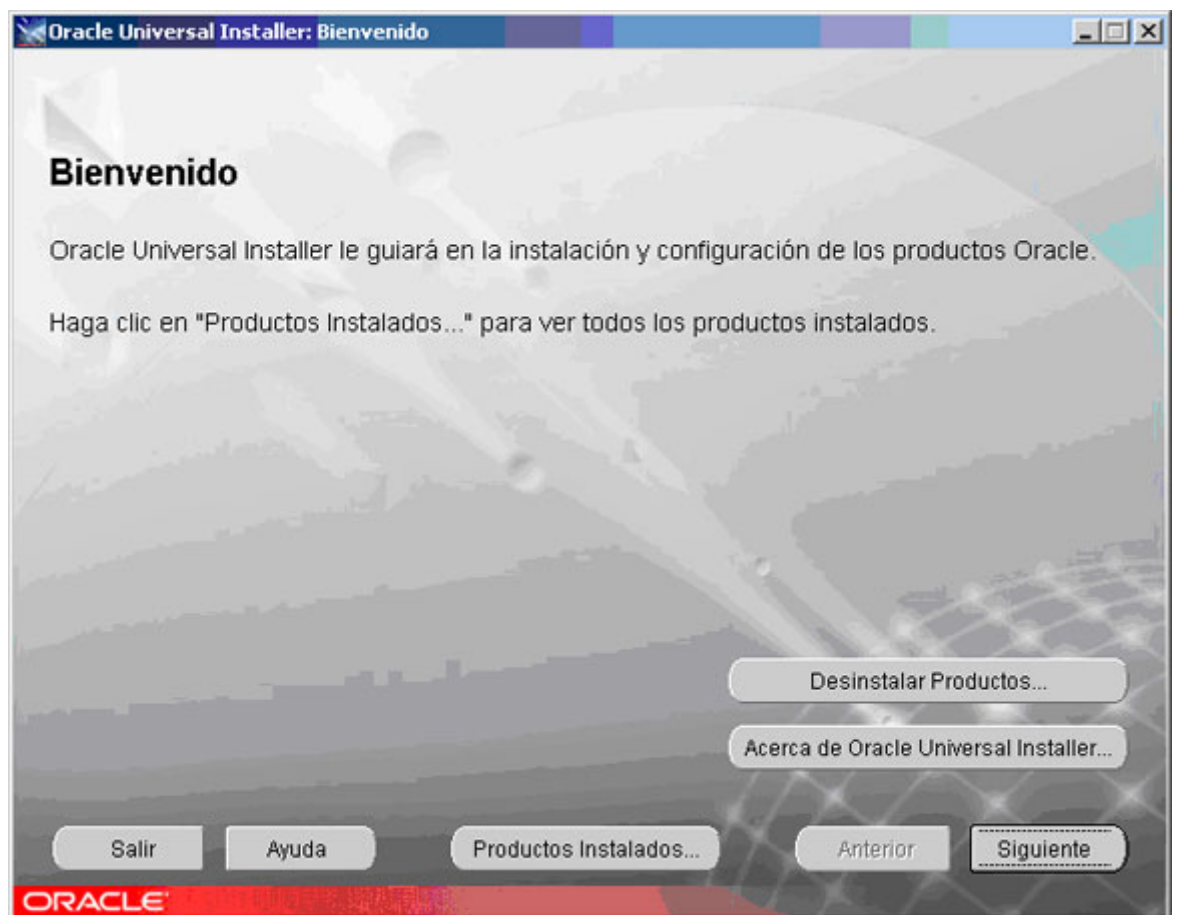
Acceso directo	Descripción
Introducción	Muestra una ventana de ayuda en formato html
Ir a Página Inicial de Base de Datos	Muestra la página de configuración y administración de Oracle
Parar Base de Datos	Detiene la base de datos
Realizar Copia de Seguridad de la Base de Datos	Copia de seguridad de la base de datos
Restaurar Base de Datos	Restaura una copia de seguridad de la base de datos

Obtener Ayuda	Ayuda
Ejecutar Línea de Comandos SQL	Permite ejecutar comandos SQL desde una ventana de MS-DOS
Iniciar Base de Datos	Inicia la base de datos previamente detenida

Cómo instalar oracle client en windows (cliente de oracle)

Este artículo muestra paso a paso cómo instalar Oracle Client (utilidad de Oracle que se ha de instalar en los PCs que queramos que tengan acceso al Servidor de Oracle):

En primer lugar necesitaremos disponer de los CDs de instalación que están disponibles gratuitamente en la web: Oracle.



Pulsaremos en "Siguiente" y seleccionamos el destino (carpeta) de instalación:



Seleccionaremos "Oracle9i Client" :



Dependiendo de las utilidades que queramos instalar seleccionaremos "Administrador" (instala todas las herramientas para administrar Oracle desde

un PC Cliente), "Runtime" (instala las herramientas básicas para acceso a Oracle, es la recomendada) ó "Personalizada" (permite seleccionar las herramientas a instalar):



Dejaremos el puerto por defecto 2030 para "Oracle Services para Microsoft Transaction Server". Este puerto no es relevante si no tenemos un Cluster de servidores:



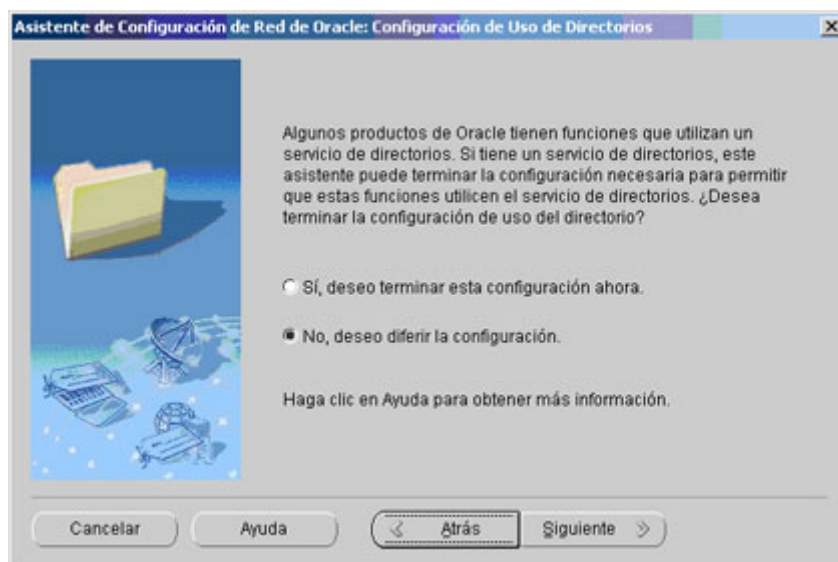
Una vez comprobadas las herramientas que se van a instalar pulsaremos en "Instalar":



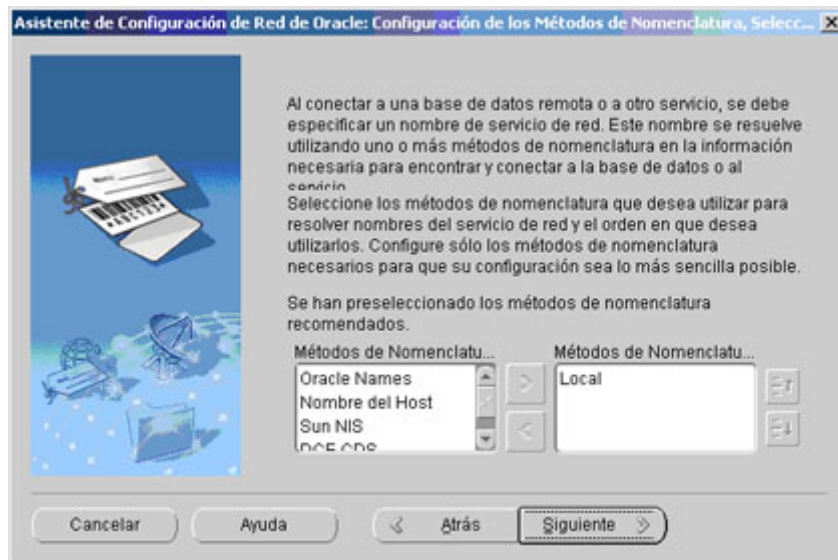
Tras la instalación realizaremos la configuración de red de Oracle:



Desmarcaremos la opción "Realizar una configuración típica" y pulsaremos en "Siguiente".



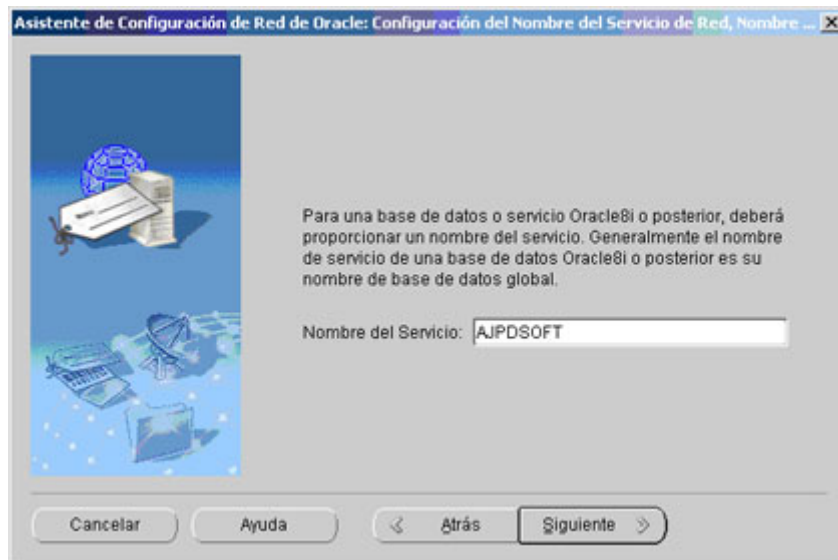
Marcamos "No, deseo diferir la configuración" y pulsamos en "Siguiente".



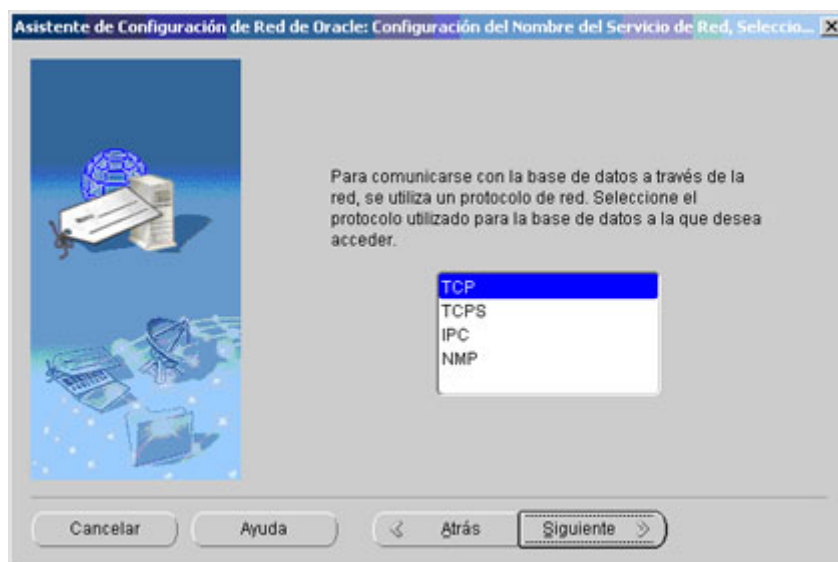
Pulsamos en "Siguiente".



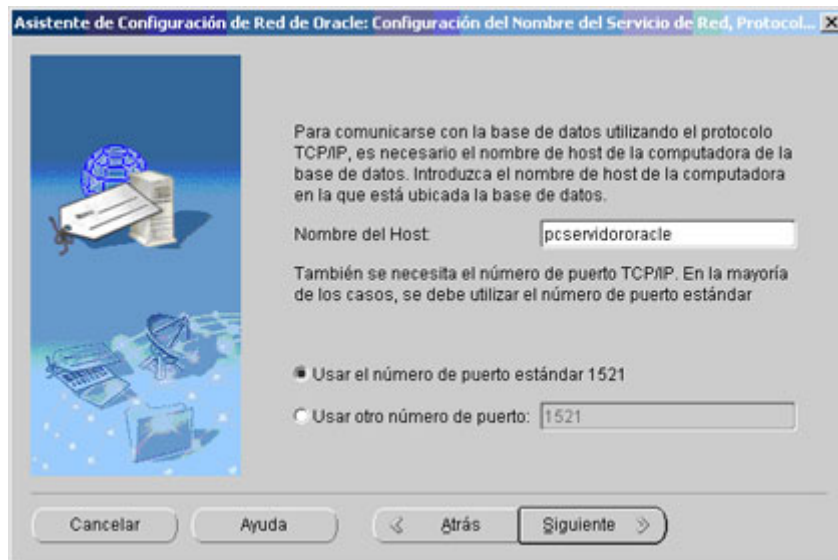
Marcamos "Base de datos o servicio Oracle8i o posterior" y pulsamos en "Siguiente".



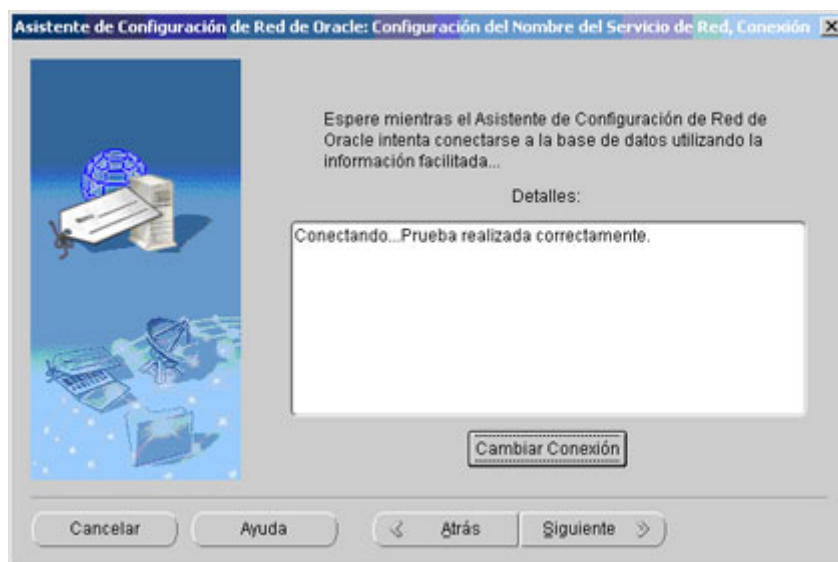
Introducimos el nombre del servicio que, normalmente, coincidirá con el nombre de la base de datos a la que nos conectaremos:



Seleccionamos el protocolo TCP y pulsamos en "Siguiente".



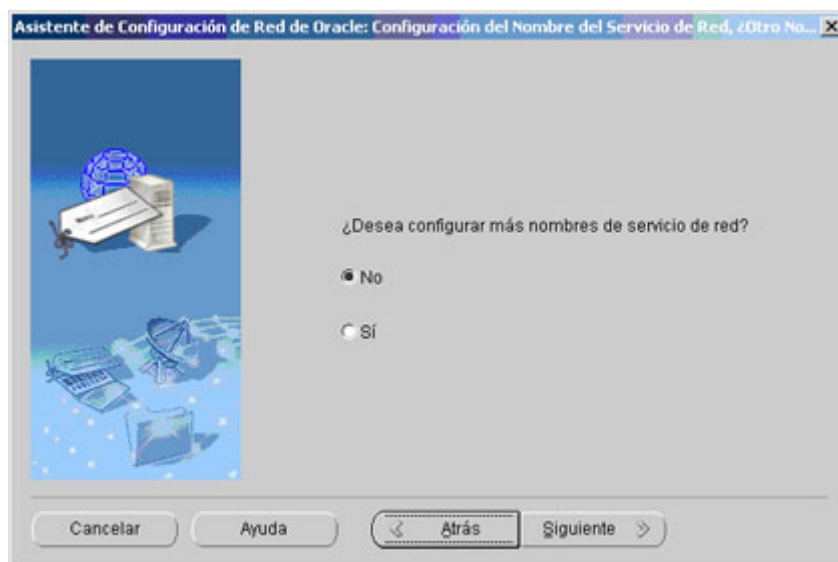
Introducimos el nombre o IP del PC que tiene la base de datos de Oracle (servidor de Oracle) y pulsamos en "Siguiente".



Si aparece algún error en la primera prueba de conexión es habitual pues utiliza el usuario y contraseña que Oracle configura por defecto. Para probar la conexión correctamente pulsaremos en "Cambiar Conexión" e introduciremos un usuario y contraseña existentes en la Base de Datos. Si no hay problemas mostrará "Conectado... Prueba realizada correctamente".

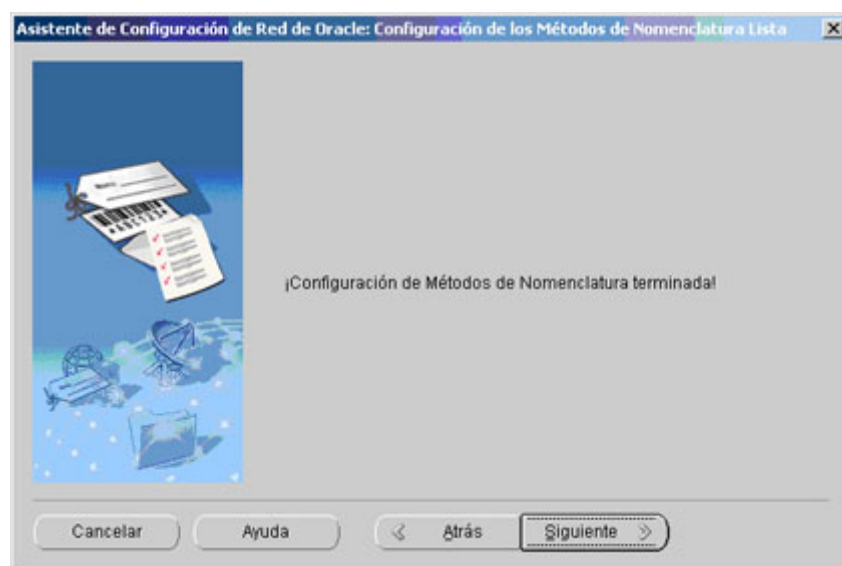


Introduciremos el nombre de red, que por defecto será el mismo que el nombre de la base de datos y pulsamos en "Siguiente".



Marcamos "No" y pulsamos en "Siguiente".

Pulsamos en "Siguiente".



Pulsamos en "Siguiente".

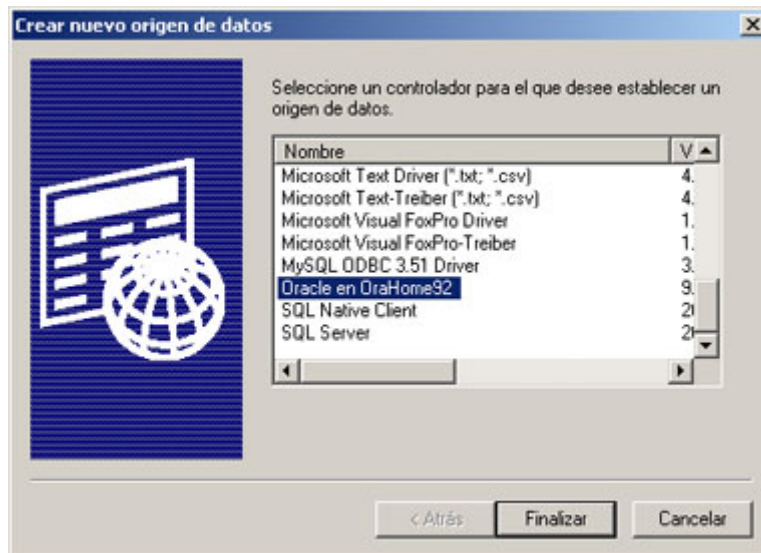


Pulsamos en "Terminar".



Pulsamos en "Salir".

Con esto habremos concluido la instalación de Oracle Client, ahora nos aparecerá un nuevo Driver ODBC:



el cual podremos utilizar para realizar la conexión con Oracle.

Tablas del diccionario de datos:

SQL muy útiles para el administrador de Oracle (estado de la base de datos Oracle, parámetros generales, ficheros de control, conexiones actuales a Oracle, nombre del ejecutable que se utiliza, nombre del usuario, diccionario de datos (vistas y tablas)...

Vista que muestra el estado de la base de datos:

```
select * from v$instance
```

Consulta que muestra si la base de datos está abierta

```
select status from v$instance
```

Vista que muestra los parámetros generales de Oracle

```
select * from v$system_parameter
```

Versión de Oracle

```
select value from v$system_parameter where name =  
'compatible'
```

Ubicación y nombre del fichero spfile

```
select value from v$system_parameter where name =  
'spfile'
```

Ubicación y número de ficheros de control

```
select value from v$system_parameter where name =  
'control_files'
```

Nombre de la base de datos

```
select value from v$system_parameter where name =  
'db_name'
```

Vista que muestra las conexiones actuales a Oracle Para visualizarla es necesario entrar con privilegios de administrador

```
select osuser, username, machine, program  
  
from v$session  
  
order by osuser
```

Vista que muestra el número de conexiones actuales a Oracle agrupado por aplicación que realiza la conexión

```
select program Aplicacion, count(program) Numero_Sesiones  
  
from v$session  
  
group by program  
  
order by Numero_Sesiones desc
```

Vista que muestra los usuarios de Oracle conectados y el número de sesiones por usuario

```
select username Usuario_Oracle, count(username)  
Numero_Sesiones  
  
from v$session  
  
group by username  
  
order by Numero_Sesiones desc
```

Propietarios de objetos y número de objetos por propietario

```
select owner, count(owner) Numero  
  
from dba_objects  
  
group by owner  
  
order by Numero desc
```

Diccionario de datos (incluye todas las vistas y tablas de la Base de Datos)

```
select * from dictionary
```

```
select table_name from dictionary
```

Muestra los datos de una tabla especificada (en este caso todas las tablas que lleven la cadena "EMPLO")

```
select * from ALL_ALL_TABLES where upper(table_name) like  
'%EMPLO%'
```

Tablas propiedad del usuario actual

```
select * from user_tables
```

Todos los objetos propiedad del usuario conectado a Oracle

```
select * from user_catalog
```

Consulta SQL para el DBA de Oracle que muestra los tablespaces, el espacio utilizado, el espacio libre y los ficheros de datos de los mismos:

```
Select t.tablespace_name "Tablespace", t.status  
"Estado",  
  
ROUND(MAX(d.bytes)/1024/1024,2) "MB Tamaño",  
  
ROUND((MAX(d.bytes)/1024/1024) -  
  
(SUM(decode(f.bytes, NULL,0, f.bytes))/1024/1024),2)  
"MB Usados",  
  
ROUND(SUM(decode(f.bytes, NULL,0,  
f.bytes))/1024/1024,2) "MB Libres",  
  
t.pct_increase "% incremento",  
  
SUBSTR(d.file_name,1,80) "Fichero de datos"  
  
FROM DBA_FREE_SPACE f, DBA_DATA_FILES d, DBA_TABLESPACES  
t  
  
WHERE t.tablespace_name = d.tablespace_name AND  
  
f.tablespace_name(+) = d.tablespace_name  
  
AND f.file_id(+) = d.file_id GROUP BY  
t.tablespace_name,  
  
d.file_name, t.pct_increase, t.status ORDER BY 1,3  
DESC
```

Productos Oracle instalados y la versión:

```
select * from product_component_version
```

Roles y privilegios por roles:

```
select * from role_sys_privs
```

Reglas de integridad y columna a la que afectan:

```
select constraint_name, column_name from  
sys.all_cons_columns
```

Tablas de las que es propietario un usuario, en este caso "HR":

```
SELECT table_owner, table_name from sys.all_synonyms  
where table_owner like 'HR'
```

Otra forma más efectiva (tablas de las que es propietario un usuario):

```
SELECT DISTINCT TABLE_NAME  
  
FROM ALL_ALL_TABLES  
  
WHERE OWNER LIKE 'HR'
```

Parámetros de Oracle, valor actual y su descripción:

```
SELECT v.name, v.value value, decode(ISSYS_MODIFIABLE,  
'DEFERRED',  
  
      'TRUE', 'FALSE') ISSYS_MODIFIABLE,  
decode(v.isDefault, 'TRUE', 'YES',  
  
      'FALSE', 'NO') "DEFAULT",  DECODE (ISSES_MODIFIABLE,  
'IMMEDIATE',  
  
      'YES','FALSE', 'NO', 'DEFERRED', 'NO', 'YES')  
SES_MODIFIABLE,  
  
      DECODE(ISSYS_MODIFIABLE, 'IMMEDIATE', 'YES',  
'FALSE', 'NO',  
  
      'DEFERRED', 'YES','YES') SYS_MODIFIABLE ,  
v.description  
  
FROM V$PARAMETER v  
  
WHERE name not like 'nls%'  ORDER BY 1
```

Usuarios de Oracle y todos sus datos (fecha de creación, estado, id, nombre, tablespace temporal,...):

```
Select  * FROM dba_users
```

Tablespaces y propietarios de los mismos:

```
select owner, decode(partition_name, null, segment_name,  
  
      segment_name || ':' || partition_name) name,
```

```

        segment_type, tablespace_name, bytes, initial_extent,
        next_extent, PCT_INCREASE, extents, max_extents
from dba_segments
Where 1=1 And extents > 1 order by 9 desc, 3

```

Últimas consultas SQL ejecutadas en Oracle y usuario que las ejecutó:

```

select distinct vs.sql_text, vs.sharable_mem,
        vs.persistent_mem, vs.runtime_mem, vs.sorts,
        vs.executions, vs.parse_calls, vs.module,
        vs.buffer_gets, vs.disk_reads, vs.version_count,
        vs.users_opening, vs.loads,
        to_char(to_date(vs.first_load_time,
        'YYYY-MM-DD/HH24:MI:SS'),'MM/DD HH24:MI:SS')
first_load_time,
        rawtohex(vs.address) address, vs.hash_value hash_value
,
        rows_processed , vs.command_type, vs.parsing_user_id
,
        OPTIMIZER_MODE , au.USERNAME parseuser
from v$sqlarea vs , all_users au
where (parsing_user_id != 0) AND
        (au.user_id(+) = vs.parsing_user_id)
and (executions >= 1) order by buffer_gets/executions
desc

```

Todos los ficheros de datos y su ubicación:

```
select * from V$DATAFILE
```

Ficheros temporales:

```
select * from V$TEMPFILE
```

Tablespaces:

```
select * from V$TABLESPACE
```

Otras vistas muy interesantes:

```
select * from V$BACKUP
```



```
select * from V$ARCHIVE
```

```
select * from V$LOG
```

```
select * from V$LOGFILE
```

```
select * from V$LOGHIST
```

```
select * from V$ARCHIVED_LOG
```

```
select * from V$DATABASE Memoria Share_Pool libre y usada
```

```
select name,to_number(value) bytes
```

```
from v$parameter where name ='shared_pool_size'
```

```
union all
```

```
select name,bytes
```

```
from v$srgastat where pool = 'shared pool' and name =  
'free memory'
```

Cursores abiertos por usuario

```
select b.sid, a.username, b.value Cursores_Abiertos
```

```
from v$session a,
```

```
v$sesstat b,
```

```
v$statname c
```

```
where c.name in ('opened cursors current')
```

```
and b.statistic# = c.statistic#
```

```
and a.sid = b.sid
```

```
and a.username is not null
```

```
and b.value >0
```

```
order by 3
```

Aciertos de la caché (no debe superar el 1 por ciento)

```
select sum(pins) Ejecuciones, sum(reloads) Fallos_cache,
```

```
trunc(sum(reloads)/sum(pins)*100,2) Porcentaje_aciertos
```

```
from v$librarycache
```

```
where namespace in ('TABLE/PROCEDURE','SQL  
AREA','BODY','TRIGGER');
```

Sentencias SQL completas ejecutadas con un texto determinado en el SQL

```
SELECT c.sid, d.piece, c.serial#, c.username, d.sql_text  
FROM v$session c, v$sqltext d  
WHERE c.sql_hash_value = d.hash_value  
and upper(d.sql_text) like '%WHERE CAMPO LIKE%'  
ORDER BY c.sid, d.piece
```

Una sentencia SQL concreta (filtrado por sid)

```
SELECT c.sid, d.piece, c.serial#, c.username, d.sql_text  
FROM v$session c, v$sqltext d  
WHERE c.sql_hash_value = d.hash_value  
and sid = 105  
ORDER BY c.sid, d.piece
```

//Tamaño ocupado por la base de datos

```
select sum(BYTES)/1024/1024 MB from DBA_EXTENTS
```

//Tamaño de los ficheros de datos de la base de datos

```
select sum(bytes)/1024/1024 MB from dba_data_files
```

//Tamaño ocupado por una tabla concreta sin incluir los índices de la misma

```
select sum(bytes)/1024/1024 MB from user_segments  
where segment_type='TABLE' and  
segment_name='NOMBRETABLA'
```

//Tamaño ocupado por una tabla concreta incluyendo los índices de la misma

```
select sum(bytes)/1024/1024 Table_Allocation_MB from  
user_segments  
where segment_type in ('TABLE','INDEX') and  
(segment_name='NOMBRETABLA' or segment_name in  
(select index_name from user_indexes where  
table_name='NOMBRETABLA'))
```

//Tamaño ocupado por una columna de una tabla

```
select sum(vsize('NOMBRECOLUMNA'))/1024/1024 MB from  
NOMBRETABLA
```

//Espacio ocupado por usuario

```
SELECT owner, SUM(BYTES)/1024/1024 FROM DBA_EXTENTS MB  
  
group by owner
```

//Espacio ocupado por los diferentes segmentos (tablas, índices, undo, rollback, cluster, ...)

```
SELECT SEGMENT_TYPE, SUM(BYTES)/1024/1024 FROM  
DBA_EXTENTS MB  
  
group by SEGMENT_TYPE
```

//Espacio ocupado por todos los objetos de la base de datos, muestra los objetos que más ocupan primero

```
SELECT SEGMENT_NAME, SUM(BYTES)/1024/1024 FROM  
DBA_EXTENTS MB  
  
group by SEGMENT_NAME  
  
order by 2 desc
```

//Obtener todas las funciones de Oracle: NVL, ABS, LTRIM, ...

```
SELECT distinct object_name  
  
FROM all_arguments  
  
WHERE package_name = 'STANDARD'  
  
order by object_name
```

//Obtener los roles existentes en Oracle Database:

```
select * from DBA_ROLES
```

//Obtener los privilegios otorgados a un rol de Oracle:

```
select privilege  
  
from dba_sys_privs  
  
where grantee = 'NOMBRE_ROL'
```

Estructura de ficheros de Oracle

- **Control files (Ficheros de Control):**

Los ficheros de control tiene un tamaño de entre 1 y 5 Megas. Son utilizados para mantener la consistencia interna y guiar las operaciones de recuperación. Son imprescindibles para que la BD se pueda arrancar. Contienen:

- Información de arranque y parada de la BD.
- Nombres de los archivos de la BD y redo log.
- Información sobre los checkpoints.
- Fecha de creación y nombre de la BD.
- Estado online y offline de los archivos.

Debe haber múltiples copias en distintos discos, mínimo dos, para protegerlos de los fallos de disco. La lista de los ficheros de control se encuentra en el parámetro `CONTROL_FILES`, que debe modificarse con la BD parada.

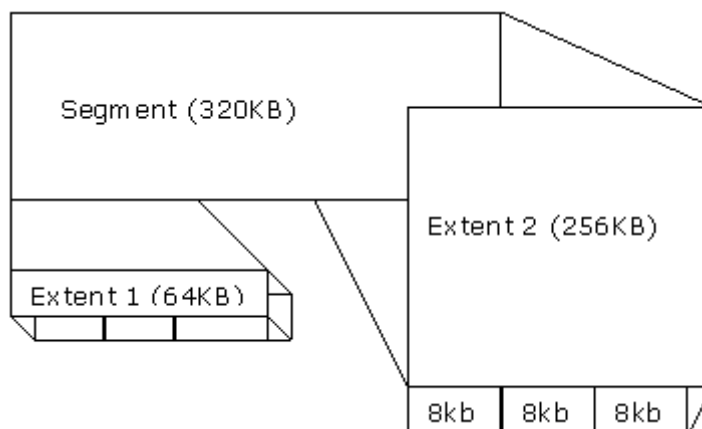
- **Data files(fichero de datos):**

En estos ficheros reside la información de la BD. Solo son modificados por el DBWR. A ellos se vuelcan los bloques sucios de la SGA cuando se hace una validación o cuando sucede un checkpoint. Las validaciones de las transacciones no producen un volcado inmediato, sino lo que se conoce por un commit diferido. Toda actualización se guarda en los ficheros de redo log, y se lleva a la BD física cuando tenemos una buena cantidad de bloques que justifiquen una operación de E/S. Almacenan los segmentos (datos, índices, rollback) de la BD. Están divididos en bloques ($\text{Bloque Oracle} = c * \text{Bloque SO}$), cada uno de los cuales se corresponde con un buffer del buffer cache de la SGA. En el bloque de cabecera no se guardan datos de usuario, sino la marca de tiempo del último checkpoint realizado sobre el fichero.

Relación entre segmentos, extensions y bloques de datos:

- Tablespaces (Espacios de tablas) : Es la unidad de almacenamiento superior, puede estar compuesta de uno o mas Data Files
- Segments (Segmentos): Un “Data file”, esta compuesto por varios segmentos.
- Extents (Extensiones): Un segmento esta compuesto por varias extensiones.
- Data blocks (Bloque de datos): Es la unidad mas pequeña de almacenamiento, su tamaño se define durante la creación de la base de datos y no se puede modificar.

La siguiente figura muestra la relación entre segmentos, extensions y bloques de datos:



Redo Log files:

En ellos se graba toda operación que se efectue en la BD y sirven de salvaguarda de la misma. Tiene que haber por lo menos 2, uno de ellos debe estar activo, *online*, y se escribe en ellos de forma cíclica. Existe la posibilidad de almacenar los distintos ficheros de *redo log* en el tiempo mediante el modo ARCHIVER. Así, se puede guardar toda la evolución de la BD desde un punto dado del tiempo.

Una opción es la utilización de archivos *redo log* multiplexados:

Permite al LGWR escribir simultáneamente la misma información en múltiples archivos *redo log*.

Se utiliza para protegerse contra fallos en el disco.

Da una alta disponibilidad a los archivos *redo log* activos u *online*.

Esto se hace definiendo el número de *grupos* y de *miembros* de archivos *redo log* que van a funcionar en paralelo:

grupos: funcionan como ficheros *redo log* normales, uno de ellos está activo y el resto espera su turno.

Su nombre lleva incorporado una numeración.

Deben contener todos el mismo número de miembros.

miembros: cada escritura de un registro *redo log* se lleva a cabo en todos los miembros del grupo activo en ese momento. Los miembros deben:

tener el mismo tamaño y el mismo número de secuencia.

deben tener nombres similares y estar en diferentes discos para proteger contra fallos de una manera efectiva.

Cuando se produce algún fallo en los ficheros de *redo log* o en el proceso LGWR:

Si la escritura en un fichero *redo log* falla pero el LGWR puede escribir al menos en uno de los miembros del grupo, lo hace , ignorando el fichero inaccesible y registrando un fallo en un fichero de traza o alerta.

Si el siguiente grupo no ha sido archivado (modo ARCHIVELOG) antes del cambio de grupo que lo pone activo, ORACLE espera hasta que se produzca el archivado.

Si fallan todos los miembros de un grupo mientras el LGWR trata de escribir, la instancia se para y necesita recuperación al arrancar.

Ficheros de Traza

Oracle crea ficheros de texto llamados de traza para ayudar en la diagnosis de problemas y en el ajuste del SGBD. Cada proceso del servidor escribe en un fichero de traza asociado cuando es necesario. Los procesos de usuarios también pueden tener asociados ficheros de traza. La situación de estos ficheros de traza del sistema se especifica por el parámetro `BACKGROUND_DUMP_DEST`, y los de

usuario por `USER_DUMP_DEST`. Oracle crea ficheros de traza automáticamente cuando ocurre algún error.

Un parámetro muy frecuentemente utilizado por los desarrolladores Oracle es el `SQL_TRACE`, que cuando está puesto a `TRUE` produce que toda sentencia SQL ejecutada genere información en los ficheros de traza. Este parámetro se puede variar con el siguiente comando:

```
alert_<>.log:
```

- **Spfile.ora init.ora :**

El `spfile` es un fichero binario que contiene la especificación de los parámetros de inicialización de la base de datos. El archivo **`spfile.ora`** Este es el primer archivo que va a "buscar" Oracle en su arranque de base de datos. Si no encuentra este archivo entonces irá a buscar el archivo **`init.ora` (fichero de texto)**

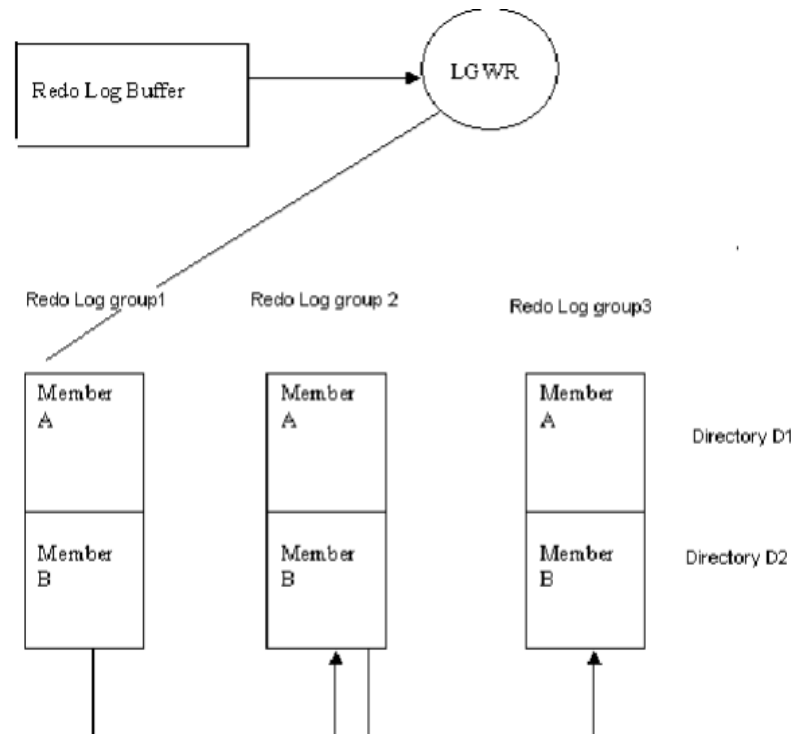
A partir de la versión 9, Oracle introdujo el fichero de parámetros de arranque (más conocido como '`spfile`') como mejora a los antiguos arranques con los '`init.ora`'.

Las ventajas que tienen estos nuevos ficheros sobre los antiguos son, entre otras:

- Los '`spfile`' son binarios, mientras que los '`init.ora`' eran ficheros de texto.
- Con los '`spfile`' se puede arrancar una instancia en modo remoto, mientras que con los '`init.ora`' el fichero debe estar accesible (residir en el mismo sistema de ficheros) desde el que se realiza el arranque.
- Los '`ALTER SYSTEM`' son guardados (si se quiere) automáticamente en los '`spfile`', mientras que en los '`init.ora`' no.
- Los '`spfile`' pueden ser incluidos en los '`backups`' de RMAN, mientras que los '`init.ora`' no.

No obstante, y debido a que los '`init.ora`' pueden ser aun de gran utilidad (arrancar una instancia cuando hay problemas, sobre todo), Oracle los ha mantenido (aunque recomienda encarecidamente el uso de los '`spfile`').

La siguiente figura muestra el uso de los ficheros “Redo Log” :



Localización de ficheros clave:

```
select 'control' tipo, substr(name,1,70) nombre FROM  
v$controlfile  
  
union all  
  
select 'datos' tipo, substr(name,1,70) nombre from  
v$datafile  
  
union all  
  
select 'redolog' tipo, substr(member,1,70) nombre from  
v$logfile
```

La instancia oracle

Una instancia de Oracle está conformada por varios procesos y espacios de memoria compartida que son necesarios para acceder a la información contenida en la base de datos.

La instancia está conformada por procesos del usuario, procesos que se ejecutan en el background de Oracle y los espacios de memoria que comparten estos procesos.

Estructura de la memoria de oracle

El Área Global del Sistema (SGA)

El SGA es un área de memoria compartida que se utiliza para almacenar información de control y de datos de la instancia. Se crea cuando la instancia es levantada y se borra cuando ésta se deja de usar (cuando se hace *shutdown*). La información que se almacena en esta área consiste de los siguientes elementos, cada uno de ellos con un tamaño fijo:

El buffer de caché (*database buffer cache*)

- Almacena los bloques de datos utilizados recientemente (se hayan o no confirmado sus cambios en el disco). Al utilizarse este buffer se reducen las operaciones de entrada y salida y por esto se mejora el rendimiento.
- El buffer de *redo log*: Guarda los cambios efectuados en la base de datos. Estos *buffers* escriben en el archivo físico de redo log tan rápido como se pueda sin perder eficiencia. Este último archivo se utiliza para recuperar la base de datos ante eventuales fallas del sistema.
- El área *shared pool*: Esta sola área almacena estructuras de memoria compartida, tales como las áreas de código SQL compartido e información interna del diccionario. Una cantidad insuficiente de espacio asignado a esta área podría redundar en problemas de rendimiento.

En esta zona se encuentran las sentencias SQL que han sido analizadas. El análisis sintáctico de las sentencias SQL lleva su tiempo y Oracle mantiene las estructuras asociadas a cada sentencia SQL analizada durante el tiempo que pueda para ver si puede reutilizarlas. Antes de analizar una sentencia SQL, Oracle mira a ver si encuentra otra sentencia exactamente igual en la zona de SQL compartido. Si es así, no la analiza y pasa directamente a ejecutar la que mantiene en memoria. De esta manera se premia la uniformidad en la programación de las aplicaciones. La igualdad se entiende que es lexicográfica, espacios en blanco y variables incluidas. El contenido de la zona de SQL compartido es:

- Plan de ejecución de la sentencia SQL.
- Texto de la sentencia.
- Lista de objetos referenciados.

Los pasos de procesamiento de cada petición de análisis de una sentencia SQL son:

- Comprobar si la sentencia se encuentra en el área compartida.
- Comprobar si los objetos referenciados son los mismos.
- Comprobar si el usuario tiene acceso a los objetos referenciados.

Si no, la sentencia es nueva, se analiza y los datos de análisis se almacenan en la zona de SQL compartida.

También se almacena en la zona de SQL compartido el *caché del diccionario*. La información sobre los objetos de la BD se encuentra almacenada en las tablas del diccionario. Cuando esta información se necesita, se leen las tablas del diccionario y su información se guarda en el caché del diccionario de la SGA.

- El caché de biblioteca se utiliza para almacenar código SQL compartido. Aquí se manejan los árboles de *parsing* y el plan de ejecución de las *queries*. Si varias aplicaciones utilizan la misma sentencia SQL, esta área compartida garantiza el acceso por parte de cualquiera de ellas en cualquier instante.
- El caché del diccionario de datos está conformado por un grupo de tablas y vistas que se identifican la base de datos. La información que se almacena aquí guarda relación con la estructura lógica y física de la base de datos. El diccionario de datos contiene información tal como los privilegios de los usuarios, restricciones de integridad definidas para algunas tablas, nombres y tipos de datos de todas las columnas y otra información acerca del espacio asignado y utilizado por los objetos de un esquema.

El Área Global de Programas (PGA)

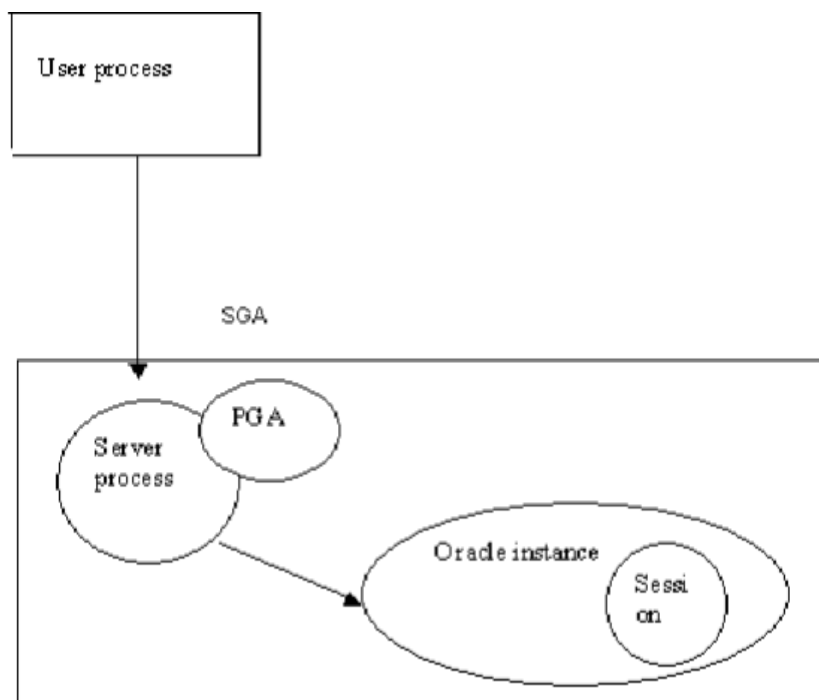
Esta área de memoria contiene datos e información de control para los procesos que se ejecutan en el servidor de Oracle (relacionados con la base de datos, por supuesto). El tamaño y contenido de la PGA depende de las opciones del servidor que se hayan instalado.

Procesos de Una Instancia de Oracle

Existen dos tipos categorías de procesos

- Procesos de usuario
- Procesos de sistema

La siguiente figura muestra la relación entre los procesos de usuario, procesos de servidor, PGA y session:



La primera interacción con Oracle, comienza desde el PC del usuario que crea el proceso de usuario, este se comunica con el proceso de servidor, la PGA es usada para almacenar información asociada con la sesión.

Oracle background processes (Procesos en segundo Plano)

Oracle cuenta con un gran número de procesos en Segundo plano, estos están categorizados en:

- Requeridos
- Opcionales

Algunos procesos en Segundo Plano son:

DBWR

El proceso DBWR es el responsable de gestionar el contenido de los buffers de datos y del caché del diccionario. Él lee los bloques de los archivos de datos y los almacena en la SGA. Luego escribe en los archivos de datos los bloques cuyo

contenido ha variado. La escritura de los bloques a disco es diferida buscando mejorar la eficiencia de la E/S.

Es el único proceso que puede escribir en la BD. Esto asegura la integridad. Se encarga de escribir los bloques de datos modificados por las transacciones, tomando la información del *buffer* de la BD cuando se valida una transacción. Cada validación no se lleva a la BD física de manera inmediata sino que los bloques de la BD modificados se vuelcan a los ficheros de datos periódicamente o cuando sucede algún *checkpoint* o punto de sincronización: grabación *diferida*:

- Los bloques del *buffer* de la BD (bloques del segmento de *rollback* y bloques de datos) menos recientemente utilizados son volcados en el disco continuamente para dejar sitio a los nuevos bloques.
- El bloque del segmento de *rollback* se escribe SIEMPRE antes que el correspondiente bloque de datos.
- Múltiples transacciones pueden solapar los cambios en un sólo bloque antes de escribirlo en el disco.

Mientras, para que se mantenga la integridad y coherencia de la BD, todas las operaciones se guardan en los ficheros de redo log. El proceso de escritura es asíncrono y puede realizar grabaciones multibloque para aumentar la velocidad.

LGWR

El proceso LGWR es el encargado de escribir los registros *redo log* en los ficheros *redo log*. Los registros *redo log* siempre contienen el estado más reciente de la BD, ya que puede que el DBWR deba esperar para escribir los bloques modificados desde el buffer de datos a los ficheros de datos.

Conviene tener en cuenta que el LGWR es el único proceso que escribe en los ficheros de *redo log* y el único que lee directamente los buffers de *redo log* durante el funcionamiento normal de la BD.

Coloca la información de los *redo log buffers* en los ficheros de *redo log*. Los *redo log buffers* almacenan una copia de las transacciones que se llevan a cabo en la BD. Esto se produce:

- a cada validación de transacción, y antes de que se comunique al proceso que todo ha ido bien,
- cuando se llena el grupo de *buffers* de *redo log*
- cuando el DBWR escribe *buffers* de datos modificados en disco.

Así, aunque los ficheros de DB no se actualicen en ese instante con los buffers de BD, la operación queda guardada y se puede reproducir. Oracle no tiene que consumir sus recursos escribiendo el resultado de las modificaciones de los datos en los archivos de datos de manera inmediata. Esto se hace porque los registros de *redo log* casi siempre tendrán un tamaño menor que los bloques afectados por las modificaciones de una transacción, y por lo tanto el tiempo que emplea en guardarlos es menor que el que emplearía en almacenar los bloques sucios resultado de una transacción; que ya serán trasladados a los ficheros por el DBWR. El LGWR es un proceso único, para asegurar la integridad. Es asíncrono. Además permite las grabaciones multibloque.

CKPT

Este proceso escribe en los ficheros de control los *checkpoints*. Estos puntos de sincronización son referencias al estado coherente de todos los ficheros de la BD en un instante determinado, en un punto de sincronización. Esto significa que los bloques sucios de la BD se vuelcan a los ficheros de BD, asegurándose de que todos los bloques de datos modificados desde el último *checkpoint* se escriben realmente en los ficheros de datos y no sólo en los ficheros *redo log*; y que los ficheros de *redo log* también almacenan los registros de *redo log* hasta este instante. La secuencia de puntos de control se almacena en los ficheros de datos, *redo log* y control. Los *checkpoints* se producen cuando:

- un espacio de tabla se pone inactivo, *offline*,
- se llena el fichero de *redo log* activo,
- se para la BD,
- el número de bloques escritos en el *redo log* desde el último *checkpoint* alcanza el límite definido en el parámetro `LOG_CHECKPOINT_INTERVAL`,
- cuando transcurra el número de segundos indicado por el parámetro `LOG_CHECKPOINT_TIMEOUT` desde el último *checkpoint*.

PMON

Este proceso restaura las transacciones no validadas de los procesos de usuario que abortan, liberando los bloqueos y los recursos de la SGA. Asume la identidad del usuario que ha fallado, liberando todos los recursos de la BD que estuviera utilizando, y anula la transacción cancelada. Este proceso se despierta regularmente para comprobar si su intervención es necesaria..

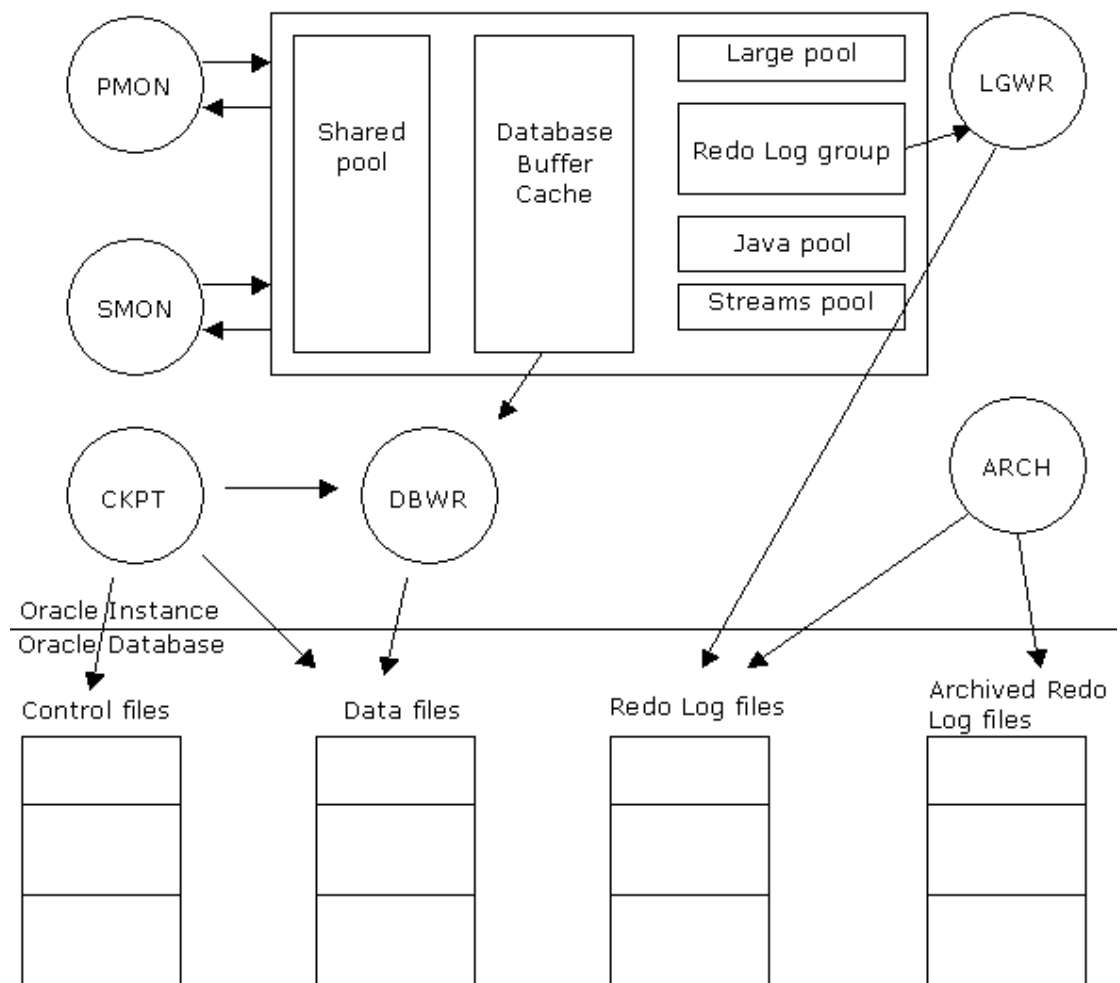
SMON

El SMON es el supervisor del sistema y se encarga de todas las recuperaciones que sean necesarias durante el arranque. Esto puede ser necesario si la BD se paró inesperadamente por fallo físico, lógico u otras causas. Este proceso realiza la recuperación de la instancia de BD a partir de los ficheros *redo log*. Además limpia los segmentos temporales no utilizados y compacta los huecos libres contiguos en los ficheros de datos. Este proceso se despierta regularmente para comprobar si debe intervenir.

ARCH:

El proceso archivador tiene que ver con los ficheros *redo log*. Por defecto, estos ficheros se reutilizan de manera cíclica de modo que se van perdiendo los registros *redo log* que tienen una cierta antigüedad. Cuando la BD se ejecuta en modo ARCHIVELOG, antes de reutilizar un fichero *redo log* realiza una copia del mismo. De esta manera se mantiene una copia de todos los registros *redo log* por si fueran necesarios para una recuperación. Este es el trabajo del proceso archivador.

La siguiente figura muestra la estructura de Oracle en detalle:



La figura anterior muestra los diferentes components del SGA, Procesos en segundo plano y sus interacciones con los ficheros de control, los ficheros Redo Log, y los

Listener

TNS Listener es un proceso servidor que provee la conectividad de red con la base de datos Oracle. El listener está configurado para escuchar la conexión en un puerto específico en el servidor de base de datos. Cuando una se pide una conexión a la base de datos, el listener devuelve la información relativa a la

conexión. La información de una conexión para una instancia de una base de datos provee el nombre de usuario, la contraseña y el SID de la base de datos. Si estos datos no son correctos se devolverá un mensaje de error.

- Por defecto el puerto del listener es el 1521
- El listener no limita el número de conexiones a la base de datos

Toda la información del listener la contiene un archivo denominado listener.ora (\$ORACLE_HOME/network/admin.)

El comando para gestionar el listener es **lsnrctl**. Mediante este comando podemos:

- Parar el listener.
- Ver el estado del listener.
- Arrancar el listener.
- Rearrancar el listener.

Seguridad listener oracle 10g (securing the listener)

El principal paso para realizar la seguridad en el listener es ponerle una contraseña **password**.

El primer método para poner una contraseña al listener es editando el fichero listener.ora y escribiendo la siguiente línea:

```
PASSWORDS_LISTENER = orapass
```

Cuando guardemos el fichero con los cambios realizamos un reload del listener

```
lsnrctl> reload
```

Nota: El comando para entrar en el listener es lsnrctl (\$ORACLE_HOME/bin)

El segundo método para poder cambiar la contraseña al listener es el siguiente:

```
lsnrctl> change_password
```

Este comando te pedirá la clave antigua y la nueva clave.

Si es la primera vez que ejecutas este comando la contraseña antigua (old password) habrá que dejarla en blanco.

El comando **SET** y **SAVE CONFIG** permite guardar esos cambios en el listener porque ahora el listener está gobernado por un **password**.

```
lsnrctl > set password
```

```
lsnrctl > save config
```

La información antigua se guardará en listener.bck y listener.ora se actualizará con los nuevos datos.

Ejemplo de configuración del listener.ora

```
LISTENER9 =  
(DESCRIPTION_LIST =  
(DESCRIPTION =  
(ADDRESS_LIST =  
(ADDRESS = (PROTOCOL = TCP) (HOST = 193.168.4.220) (PORT =  
2484) )  
)  
)  
)  
SID_LIST_LISTENER9 =  
(SID_LIST =  
(SID_DESC =  
(GLOBAL_DBNAME = orasite)  
(ORACLE_HOME = /oracle9/product/9.2.0)  
(SID_NAME = orasite)  
)  
)
```

Parámetros del archivo:

HOST: Dirección ip del servidor de base de datos

PORT: Puerto de escucha de la base de datos (por defecto suele ser el 1521)

CLOBAL_DB_NAME: Nombre de la base de datos

ORACLE_HOME: Directorio de instalación de ORACLE (ORACLE_HOME)

SID_NAME: SID de la base de datos (muchas veces coincide con el GLOBAL_DB_NAME)

Este archivo incluye:

- Direcciones de protocolo en las que acepta solicitudes de conexión.
- Servicios de base de datos
- Parámetros de control utilizados por el listener.

Actualización de los parámetros del sistema

Introducción

Oracle es una base de datos configurable mediante una serie de parámetros, el administrador puede optimizar los valores de esta base de datos. Estos parámetros de optimización y configuración de base de datos se almacenan en un fichero. Este fichero es el primero al que se accede al arrancar la base de datos oracle. El fichero de parámetros del que estamos hablando se denomina **init.ora**. En este fichero como hemos indicado anteriormente escribiremos los parámetros de configuración de oracle, pero si en este archivo alguno de los parámetros de oracle configurables no se encuentra este parámetro tomará el valor que oracle tenga por defecto.

Mostrar la lista de todos los parámetros

```
SELECT p.name,  
       p.type,  
       p.value,  
       p.isises_modifiable,  
       p.issys_modifiable,  
       p.isinstance_modifiable  
FROM v$parameter p  
ORDER BY p.name;
```

Tipos de parámetros existentes

Existen tres tipos de parámetros en oracle:

1. **Parámetros “fijos”:** Son parámetros que una vez instalada la base de datos no se pueden volver a modificar / configurar. El juego de caracteres es un claro ejemplo.
2. **Parámetros Estáticos:** Son parámetros que se pueden modificar, pero su modificación implica cerrar la base de datos y volverla a abrir para que los lea del fichero y pueda realizar el cambio.
3. **Parámetros Dinámicos:** Son parámetros cuyo valor se puede cambiar sin necesidad de cerrar la base de datos a diferencia de los estáticos.

Para saber si un parámetro es fijo, estático o dinámico os remito a la documentación oficial de oracle: [parametros inicializacion](#)

Ubicación y nomenclatura del fichero init.ora

El archivo init.ora lo podemos encontrar en Windows dentro del directorio ORACLE_HOME\database y en UNIX dentro del directorio ORACLE_HOME/dbs. El nombre del archivo siempre corresponderá a initsid.ora siendo sid el nombre de la base de datos.(Este es el nombre que oracle buscará al arrancar la base de datos)

Spfile.ora

Init.ora no es el único archivo de parámetros que podemos encontrar en las base de datos oracle. A partir de la **versión 9** encontramos el archivo **spfile.ora**. Este es el primer archivo que va a "buscar" oracle en su arranque de base de datos. Si no encuentra este archivo entonces irá a buscar el archivo **init.ora** Este archivo está codificado y las modificaciones en él se realizarán mediante una serie de comandos oracle que posteriormente indicaremos. Es cierto que este archivo

podemos intentar abrirlo con el notepad solo que probablemente quede corrupto o inservible. La ubicación de este archivo es la misma que la del init.ora

Cambio de los valores de los parámetros

Si queremos realizar algún cambio en algún parámetro de base de datos tenemos que diferenciar dos cosas:

1. Si el cambio es en el init.ora o spfile.ora
2. Tipo de parámetro sobre el que se quiere hacer el cambio

Cambios en el init.ora

Vamos a explicar como realizar un cambio en el fichero init.ora, para ello tenemos que tener en cuenta el tipo de parámetro que vamos a cambiar. Como hemos visto al principio de este artículo, existen tres tipos de parámetros, dejando a un lado los parámetros fijos (aquellos que no se pueden cambiar una vez instalada la base de datos) nos quedan los parámetros estáticos y los dinámicos. Para modificar un parámetro estático nos basta con editar el fichero init.ora y modificar o añadir ahí el parámetro nuevo reiniciando la base de datos para que coja estos cambios. En cuando a los parámetros dinámicos podemos cambiarlos en tiempo real sin parar la base de datos mediante la siguiente sentencia.

```
SQL> ALTER SYSTEM SET PARAMETRO = VALOR;
```

Este cambio pasa automáticamente a ser efectivo en la base de datos, aunque tenemos que tener en cuenta que la próxima vez que la base de datos sea iniciada lo que va a leer es el fichero de parámetros init.ora y si este cambio no se ha realizado en este fichero la base de datos obtendrá lo que en él ponga.

Cambios en el spfile.ora

Como hemos comentado este fichero lo podemos encontrar en las bases de datos a partir de la versión 9 y como también hemos comentado es un fichero no editable por lo que los cambios se realizan a través del comando ALTER

SYSTEM añadiendo la cláusula SCOPE con una serie de valores que detallaremos a continuación con un ejemplo:

Vamos a cambiar el parámetro shared_pool_size a 150 Megas

```
SQL> ALTER SYSTEM set shared_pool_size= 150 SCOPE=spfile
```

En este caso hemos cambiado el parámetro y estos cambios se han recogido en el archivo spfile, por lo tanto tomarán su cambio cuando sea reiniciada la base de datos.

```
SQL> ALTER SYSTEM set shared_pool_size= 150 SCOPE=memory
```

En este caso se ha cambiado el parámetro y estos cambios se han recogido solamente en memoria, esto quiere decir que se hacen efectivos al momento (si el tipo de parámetro lo permite) pero este cambio no se ver reflejado en el archivo de parámetros por lo tanto cuando volvamos a reiniciar la base de datos tomará el valor que en este tenía (el antiguo).

```
SQL> ALTER SYSTEM set shared_pool_size= 150 SCOPE=both
```

En este caso el parámetro se cambia tanto en el spfile como en memoria

Perdida del spfile

Como hemos comentado en un par de ocasiones el archivo spfile no es un archivo editable y si se edita con notepad y se vuelve a guardar lo mas probable es que se corrompa. En caso de perdida de este archivo es bueno tener un init.ora a partir del cual podemos recuperarlo o recrearlo. Si tenemos un init.ora la sentencia para hacer esto es la siguiente:

```
SQL> CREATE SPFILE [= 'spfile_name'] FROM PFILE  
[='Spfile_name'];
```

Control de sesiones (kill session)

Creación y administración de tablespaces

Copia de seguridad de una base de datos Oracle (rman)

RMAN

Es una utilidad que se sirve con Oracle Database, que nos permite realizar copias de seguridad de la base de datos en “Caliente”. Mostraremos como realizar una copia de seguridad de la base de datos con RMAN.

En primer lugar hay que decir que para poder realizar una copia de seguridad en caliente mediante RMAN la base de datos debe estar en modo ARCHIVELOG (archive log automático). Si aún no lo está deberá cambiarla a este modo, puede ver un manual de cómo hacerlo pulsando aquí.

Iniciaremos RMAN desde una ventana de MS-DOS, para ello pulsaremos en "Inicio" -"Ejecutar" y escribiremos "cmd", pulsaremos INTRO, nos aparecerá la ventana de consola de MS-DOS, escribiremos

```
rman
```

Para conectarnos a la base de datos de la que queramos hacer copia de seguridad introduciremos el comando:

```
connect target  
nombre_usuario/contraseña@Nombre_Base_Datos;
```

El resultado del comando será:

```
conectado a la base de datos destino: XE  
(DBID=2475292301)
```

A continuación ejecutaremos el siguiente script, tal y como os explicamos:

Introducimos la siguiente línea y pulsamos INTRO:

```
run {
```

Introduciremos la siguiente línea y pulsaremos INTRO:

```
allocate channel C1 device type DISK format
'c:/temp/csbd_%d_%u_%t.bak';
```

donde

- "C:/temp" será la carpeta de destino de la copia (debe existir previamente).
- %d: le indicamos con este parámetro que nos incluya en el nombre del fichero el nombre de la base de datos.
- %u: le indicamos que incluya el identificador de la base de datos.
- %t: le indicamos que incluya la fecha y hora de creación del fichero.

Introduciremos la siguiente línea y pulsaremos INTRO:

```
backup database include current controlfile plus
archivelog delete all input;
```

Introduciremos la siguiente línea y pulsaremos INTRO, tras hacerlo se iniciará la copia de seguridad:

```
}
```

El script completo quedará de la siguiente forma:

```
run {
2> allocate channel C1 device type DISK format
'c:/temp/csbd_%d_%u_%t.bak';
3> backup database include current controlfile plus
archivelog delete all input;
4> }
```

El resultado del comando en Oracle 9/10:

```
using target database controlfile instead of recovery
catalog
allocated channel: C1
channel C1: sid=17 devtype=DISK

Starting backup at 14-JUL-06
current log archived
channel C1: starting archive log backupset
channel C1: specifying archive log(s) in backup set
```


input archive log thread=1 sequence=395 recid=1
stamp=595773970
channel C1: starting piece 1 at 14-JUL-06
channel C1: finished piece 1 at 14-JUL-06
piece handle=C:/TEMP/CSBD_BDLOCAL_03HO5IGJ_595773971.BAK
comment=NONE
channel C1: backup set complete, elapsed time: 00:00:02
channel C1: deleting archive log(s)
archive log filename=C:/ORACLE/ORA92/RDBMS/ARC00395.001
recid=1 stamp=595773970
Finished backup at 14-JUL-06

Starting backup at 14-JUL-06
channel C1: starting full datafile backupset
channel C1: specifying datafile(s) in backupset
including current SPFILE in backupset
including current controlfile in backupset
input datafile fno=00001
name=F:/BDORACLE/BDLOCAL/SYSTEM01.DBF
input datafile fno=00002
name=F:/BDORACLE/BDLOCAL/UNDOTBS01.DBF
input datafile fno=00005
name=F:/BDORACLE/BDLOCAL/EXAMPLE01.DBF
input datafile fno=00010
name=F:/BDORACLE/BDLOCAL/XDB01.DBF
input datafile fno=00006
name=F:/BDORACLE/BDLOCAL/INDX01.DBF
input datafile fno=00009
name=F:/BDORACLE/BDLOCAL/USERS01.DBF
input datafile fno=00003
name=F:/BDORACLE/BDLOCAL/CWMLITE01.DBF
input datafile fno=00004
name=F:/BDORACLE/BDLOCAL/DRSYS01.DBF
input datafile fno=00007
name=F:/BDORACLE/BDLOCAL/ODM01.DBF
input datafile fno=00008
name=F:/BDORACLE/BDLOCAL/TOOLS01.DBF
channel C1: starting piece 1 at 14-JUL-06
channel C1: finished piece 1 at 14-JUL-06
piece handle=C:/TEMP/CSBD_BDLOCAL_04HO5IGM_595773974.BAK
comment=NONE

channel C1: backup set complete, elapsed time: 00:07:26
Finished backup at 14-JUL-06
Starting backup at 14-JUL-06
current log archived
channel C1: starting archive log backupset
channel C1: specifying archive log(s) in backup set
input archive log thread=1 sequence=396 recid=2
stamp=595774421
channel C1: starting piece 1 at 14-JUL-06
channel C1: finished piece 1 at 14-JUL-06
piece handle=C:/TEMP/CSBD_BDLOCAL_05HO5IUL_595774421.BAK
comment=NONE
channel C1: backup set complete, elapsed time: 00:00:02
channel C1: deleting archive log(s)
archive log filename=C:/ORACLE/ORA92/RDBMS/ARC00396.001
recid=2 stamp=595774421
Finished backup at 14-JUL-06
released channel: C1

El resultado del comando en Oracle XE:

se utiliza el archivo de control de la base de datos
destino en lugar del catálogo de recuperación
canal asignado: C1
canal C1: sid=22 devtype=DISK
Iniciando backup en 23/07/06
log actual archivado
canal C1: iniciando juego de copias de seguridad de
archive log
canal C1: especificando archive log(s) en el juego de
copias de seguridad
thread de archive log de entrada=1 secuencia=22 recid=1
marca=596579800
canal C1: iniciando parte 1 en 23/07/06
canal C1: parte terminada 1 en 23/07/06
manejador de parte=C:/TEMP/CSBD_XE_01HOU5EV_596579807.BAK
etiqueta=TAG20060723T2
03641 comentario=NONE
canal C1: juego de copias de seguridad terminado, tiempo
transcurrido: 00:00:08
canal C1: suprimiendo archive log(s)
archive log

nombre=C:/ORACLEXE/APP/ORACLE/FLASH_RECOVERY_AREA/XE/ARCHIVELOG2006
_07_2301_MF_1_22_2D7JFNX9_.ARC recid=1 marca=596579800
backup terminado en 23/07/06

Iniciando backup en 23/07/06
canal C1: iniciando juego de copias de seguridad de
archivo de datos completo
canal C1: especificando archivo(s) de datos en el juego
de copias de seguridad
archivo de datos de entrada fno=00001
nombre=C:/ORACLEXE/ORADATA/XE/SYSTEM.DBF
archivo de datos de entrada fno=00003
nombre=C:/ORACLEXE/ORADATA/XE/SYSAUX.DBF
archivo de datos de entrada fno=00002
nombre=C:/ORACLEXE/ORADATA/XE/UNDO.DBF
archivo de datos de entrada fno=00004
nombre=C:/ORACLEXE/ORADATA/XE/USERS.DBF
canal C1: iniciando parte 1 en 23/07/06
canal C1: parte terminada 1 en 23/07/06
manejador de parte=C:TEMPCSBD_XE_02HOU5FA_596579818.BAK
etiqueta=TAG20060723T2
03657 comentario=NONE
canal C1: juego de copias de seguridad terminado, tiempo
transcurrido: 00:00:55
canal C1: iniciando juego de copias de seguridad de
archivo de datos completo
canal C1: especificando archivo(s) de datos en el juego
de copias de seguridad
incluyendo el archivo de control actual en el juego de
copias de seguridad
incluyendo SPFILE actual en el juego de copias de
seguridad
canal C1: iniciando parte 1 en 23/07/06
canal C1: parte terminada 1 en 23/07/06
manejador de parte=C:/TEMP/CSBD_XE_03HOU5H1_596579873.BAK
etiqueta=TAG20060723T2
03657 comentario=NONE
canal C1: juego de copias de seguridad terminado, tiempo
transcurrido: 00:00:03
backup terminado en 23/07/06
Iniciando backup en 23/07/06

```

log actual archivado
canal C1: iniciando juego de copias de seguridad de
archive log
canal C1: especificando archive log(s) en el juego de
copias de seguridad
thread de archive log de entrada=1 secuencia=23 recid=2
marca=596579877
canal C1: iniciando parte 1 en 23/07/06
canal C1: parte terminada 1 en 23/07/06
manejador de parte=C:/TEMP/CSBD_XE_04HOU5H6_596579878.BAK
etiqueta=TAG20060723T2
03758 comentario=NONE
canal C1: juego de copias de seguridad terminado, tiempo
transcurrido: 00:00:02
canal C1: suprimiendo archive log(s)
archive log
nombre=C:/ORACLEXE/APP/ORACLEFLASH_RECOVERY_AREAXEARCHI-
VELOG2006
_07_2301_MF_1_23_2D7JJ5H4_.ARC recid=2 marca=596579877
backup terminado en 23/07/06
canal liberado: C1

```

Podremos comprobar que habrá creado los ficheros correspondientes a la copia de seguridad en la carpeta especificada

El contenido completo de la ventana de MS-DOS para Oracle XE:

```

C:>rman

Recovery Manager : Release 10.2.0.1.0 - Production on Dom
Jul 23 20:21:50 2006

Copyright (c) 1982, 2005, Oracle. All rights reserved.

RMAN> connect target /@XE;

conectado a la base de datos destino: XE
(DBID=2475292301)

RMAN> run {
2> allocate channel C1 device type DISK format
'c:tempcsbd_%d_%u_%t.bak';
3> backup database include current controlfile plus
archivelog delete all input;

```

4> }

se utiliza el archivo de control de la base de datos
destino en lugar del cat blo
go de recuperaci n
canal asignado: C1
canal C1: sid=22 devtype=DISK

Iniciando backup en 23/07/06
log actual archivado
canal C1: iniciando juego de copias de seguridad de
archive log
canal C1: especificando archive log(s) en el juego de
copias de seguridad
thread de archive log de entrada=1 secuencia=22 recid=1
marca=596579800
canal C1: iniciando parte 1 en 23/07/06
canal C1: parte terminada 1 en 23/07/06
manejador de parte=C:/TEMP/CSBD_XE_01HOU5EV_596579807.BAK
etiqueta=TAG20060723T2
03641 comentario=NONE
canal C1: juego de copias de seguridad terminado, tiempo
transcurrido: 00:00:08
canal C1: suprimiendo archive log(s)
archive log
nombre=C:/ORACLEXE/APP/ORACLEFLASH_RECOVERY_AREAXEARCHI-
VELOG2006
_07_2301_MF_1_22_2D7JFNX9_.ARC recid=1 marca=596579800
backup terminado en 23/07/06

Iniciando backup en 23/07/06
canal C1: iniciando juego de copias de seguridad de
archivo de datos completo
canal C1: especificando archivo(s) de datos en el juego
de copias de seguridad
archivo de datos de entrada fno=00001
nombre=C:/ORACLEXE/ORADATA/XESYSTEM.DBF
archivo de datos de entrada fno=00003
nombre=C:/ORACLEXE/ORADATA/XESYSAUX.DBF
archivo de datos de entrada fno=00002
nombre=C:/ORACLEXE/ORADATA/XEUNDO.DBF
archivo de datos de entrada fno=00004
nombre=C:/ORACLEXE/ORADATA/XEUSERS.DBF

canal C1: iniciando parte 1 en 23/07/06
canal C1: parte terminada 1 en 23/07/06
manejador de parte=C:/TEMP/CSBD_XE_02HOU5FA_596579818.BAK
etiqueta=TAG20060723T2
03657 comentario=NONE
canal C1: juego de copias de seguridad terminado, tiempo
transcurrido: 00:00:55
canal C1: iniciando juego de copias de seguridad de
archivo de datos completo
canal C1: especificando archivo(s) de datos en el juego
de copias de seguridad
incluyendo el archivo de control actual en el juego de
copias de seguridad
incluyendo SPFILE actual en el juego de copias de
seguridad
canal C1: iniciando parte 1 en 23/07/06
canal C1: parte terminada 1 en 23/07/06
manejador de parte=C:/TEMP/CSBD_XE_03HOU5H1_596579873.BAK
etiqueta=TAG20060723T2
03657 comentario=NONE
canal C1: juego de copias de seguridad terminado, tiempo
transcurrido: 00:00:03
backup terminado en 23/07/06
Iniciando backup en 23/07/06
log actual archivado
canal C1: iniciando juego de copias de seguridad de
archive log
canal C1: especificando archive log(s) en el juego de
copias de seguridad
thread de archive log de entrada=1 secuencia=23 recid=2
marca=596579877
canal C1: iniciando parte 1 en 23/07/06
canal C1: parte terminada 1 en 23/07/06
manejador de parte=C:/TEMP/CSBD_XE_04HOU5H6_596579878.BAK
etiqueta=TAG20060723T2
03758 comentario=NONE
canal C1: juego de copias de seguridad terminado, tiempo
transcurrido: 00:00:02
canal C1: suprimiendo archive log(s)
archive log
nombre=C:/ORACLEXE/APP/ORACLEFLASH_RECOVERY_AREAXEARCHI-

```
VELOG2006
_07_2301_MF_1_23_2D7JJ5H4_.ARC recid=2 marca=596579877
backup terminado en 23/07/06
canal liberado: C1
```

Otros comandos interesantes de rman:

- *report obsolete*: muestra un listado con los juegos de copias marcadas como obsoletas, como caducadas, como no válidas.
- *delete obsolete*: elimina todos los juegos de copias de seguridad marcadas como obsoletas. El número de juegos de copias dependerá del parámetro *retention policy*.
- *show all*: muestra todos los parámetros de configuración de retención actual.

Cómo activar el modo de archive log (archivado automático) de una base de datos Oracle

Este manual muestra y explica cómo activar el modo de archivelog de una base de datos Oracle

Abriremos la aplicación *SQL Plus* de Oracle desde una ventana de MS-DOS ("Inicio" - "Ejecutar" - "cmd"):

```
sqlplus /nolog
```

Nos conectamos con un usuario con suficientes privilegios a la base de datos Oracle a la que queramos activarle el modo Archive Log (ARCHIVELOG), con el comando:

```
connect usuario/contraseña@NOMBRE_BASE_DATOS as sysdba
```

Si ejecutamos este comando y la base de datos está en modo OPEN (abierta):

```
alter database archivelog;
```

Nos dará el siguiente error:

```
alter database archivelog
```

```
ERROR at line 1:
ORA-01126: database must be mounted EXCLUSIVE and not
open for this operation
```

Indicando que para poder cambiar la base de datos Oracle a modo archivado (archive log) hay que iniciarla en modo "mount" (montado). Para ello hay que detenerla e iniciarla con los siguientes comandos:

```
shutdown immediate;
```

Resultado comando:

```
Database closed.  
Database dismounted.  
ORACLE instance shut down.
```

La iniciamos en modo "mount" con el comando:

```
startup mount;
```

Resultado comando:

```
ORACLE instance started.  
Total System Global Area 135338868 bytes  
Fixed Size 453492 bytes  
Variable Size 109051904 bytes  
Database Buffers 25165824 bytes  
Redo Buffers 667648 bytes  
Database mounted.
```

Para cambiar a modo archivado ejecutaremos el siguiente comando:

```
alter database archive log;
```

Resultado comando:

```
Database altered.
```

Volveremos a iniciar la base de datos:

```
alter database open;
```

Resultado comando:

```
Database altered.
```

Para activar el archivado automático comprobamos el valor del parámetro "log_archive_start", si está a "false" lo pondremos a "true". Para consultar el valor actual del parámetro ejecutaremos el siguiente comando:

```
show parameter log_archive_start;
```


Devolverá el siguiente resultado:

```
NAME TYPE VALUE
-----
log_archive_start boolean FALSE
```

Modificaremos el parámetro "log_archive_start" a "true" con el siguiente comando:

```
alter system set LOG_ARCHIVE_START=TRUE SCOPE=spfile;
```

Resultado comando:

```
System altered.
```

Al ejecutar "SCOPE=spfile" haremos que los cambios se guarden definitivamente.

Para que los cambios tengan efecto es recomendable parar y volver a iniciar la base de datos:

```
shutdown immediate;
```

Iniciaremos la base de datos en modo normal:

```
startup;
```

Notas:

Para comprobar en qué modo está la base de datos:

```
select log_mode from v$database;
```

Resultado:

```
LOG_MODE
-----
NOARCHIVELOG (está en modo no archivado)

LOG_MODE
-----
ARCHIVELOG (está en modo archivado)
```

Otra forma:

```
archive log list;
```

Resultado:

```
Database log mode Archive Mode
Automatic archival Enabled
```

```
Archive destination C:/oracle/ora92/RDBMS
Oldest online log sequence 395
Next log sequence to archive 397
Current log sequence 397
```

Para ver el valor parámetro de archivado automático:

```
show parameter log_archive_start;

NAME TYPE VALUE
-----
log_archive_start boolean FALSE
```

Desactivar el modo archive log

Abriremos la aplicación *SQL Plus* de Oracle desde una ventana de MS-DOS ("Inicio" - "Ejecutar" - "cmd"):

```
sqlplus /nolog
```

Nos conectamos con un usuario con suficientes privilegios a la base de datos Oracle a la que queramos desactivarle el modo Archive Log (ARCHIVELOG), con el comando:

```
connect usuario/contraseña@NOMBRE_BASE_DATOS as sysdba
```

Ejecutamos el siguiente comando para desactivar el modo archive log:

```
alter system archive log stop;
```

Mostrará el siguiente resultado:

```
System altered.
```

Para comprobar que se ha desactivado correctamente:

```
archive log list;
```

Mostrará el siguiente resultado:

```
Database log mode Archive Mode
Automatic archival Disabled
Archive destination C:/oracle/oradata/bdtest/archive
Oldest online log sequence 70
Next log sequence to archive 72
Current log sequence 72
```

A continuación os mostramos todo el contenido de la ventana de MS-DOS:

C:>sqlplus /nolog

SQL*Plus: Release 9.2.0.1.0 - Production on Fri Jul 14
12:30:36 2006

Copyright (c) 1982, 2002, Oracle Corporation. All rights
reserved.

SQL> connect /@BDLOCAL as sysdba;

Connected.

SQL> select log_mode from v\$database;

LOG_MODE

NOARCHIVELOG

SQL> show parameter log_archive_start;

NAME TYPE VALUE

log_archive_start boolean FALSE

SQL> alter database archivelog;

alter database archivelog

*

ERROR at line 1:

ORA-01126: database must be mounted EXCLUSIVE and not
open for this operation

SQL> shutdown immediate;

Database closed.

Database dismounted.

ORACLE instance shut down.

SQL> startup mount

ORACLE instance started.

Total System Global Area 135338868 bytes

Fixed Size 453492 bytes

Variable Size 109051904 bytes

Database Buffers 25165824 bytes

Redo Buffers 667648 bytes

Database mounted.

SQL> alter database archivelog;

Database altered.

SQL> select log_mode from v\$database;

LOG_MODE

ARCHIVELOG

SQL> alter database open;

Database altered.

SQL> show parameter log_archive_start;

NAME TYPE VALUE

log_archive_start boolean FALSE

SQL> alter system set LOG_ARCHIVE_START=TRUE

SCOPE=spfile;

Sistema modificado.

SQL> show parameter log_archive_dest

NAME TYPE VALUE

log_archive_dest string

log_archive_dest_1 string

log_archive_dest_10 string

log_archive_dest_2 string

log_archive_dest_3 string

log_archive_dest_4 string

log_archive_dest_5 string

log_archive_dest_6 string

log_archive_dest_7 string

log_archive_dest_8 string

log_archive_dest_9 string

NAME TYPE VALUE

log_archive_dest_state_1 string enable

log_archive_dest_state_10 string enable

log_archive_dest_state_2 string enable

log_archive_dest_state_3 string enable

log_archive_dest_state_4 string enable

log_archive_dest_state_5 string enable

log_archive_dest_state_6 string enable

```
log_archive_dest_state_7 string enable
log_archive_dest_state_8 string enable
log_archive_dest_state_9 string enable
SQL> show parameter log_archive_dest

SQL> ALTER SYSTEM ARCHIVE LOG STOP;
System altered.

SQL> archive log list;
Database log mode Archive Mode
Automatic archival Disabled
Archive destination C:/oracle/oradata/bdtest/archive
Oldest online log sequence 70
Next log sequence to archive 72
Current log sequence 72
```

Export / import

¿Qué es un export/import en oracle?

Export/Import es una utilidad de Oracle para realizar **backups lógicos de Oracle** (y luego poderlos restaurar). Esto significa que copian el contenido de la BD pero sin almacenar la posición física de los datos. Para realizar estas operaciones la base de datos tiene que estar abierta.

Para crear el fichero de backup se utiliza la utilidad **export** y para importar el contenido o recuperar la base de datos se realiza **import**.

Este tipo de backup se utiliza en los siguientes casos:

1. Para realizar backups de bases de datos (pequeñas/medianas bases de datos)
2. Para corregir "Row Migration & Row Chaining"
3. Detectar alguna corrupción en la base de datos, puesto que al hacer el export se lee toda la bd.
4. Para "migrar" una base de datos a otro servidor

Export en oracle 10g

Mediante unos ejemplos vamos a explicar cómo realizar algunos export en una versión de base de datos 10g, por supuesto existirán más formas de hacer exports según los argumentos que se le pasen:

El **comando** para realizar export en Oracle es **exp**

Para ver todos los argumentos y significado de ellos de un export basta con realizar un **exp help=yes**

Ejemplo1: Copia completa de la base de datos

```
C:\exp file=/oracle9/export_orasite.dmp full=yes  
log=/oracle9/log/export_orasite.log buffer=1000000
```

file=/oracle9/export_orasite.dmp > Nombre y ubicación del archivo del export (el usuario oracle tiene que tener permisos para escribir ahí. La extensión de este archivo es dmp.

full=yes> Con esto indicamos que el export es completo, todos los esquemas de la base de datos y sus datos permisos, privilegios ..

log=/oracle9/log/export_orasite.log > fichero para tener el log del export, el usuario oracle también tiene que tener permisos en ese directorio para escribir.

buffer=1000000 > reservamos buffer para la operación.

Ejemplo2: Copia de tablas específicas de un usuario

```
C:\exp scott/tiger file=orasitescott.dmp  
tables=(emp,dept) buffer=1000000
```

En este caso realizamos sólo un backup de las tablas especificadas en el argumento tables del usuario scott

Ejemplo3: Copiar tablas de un usuario con una condición específica

```
C:\exp scott/tiger file=c:\orasitempleados.dmp tables=emp  
query=\"where deptno=10\"
```

Exportamos la tabla emp del usuario scott y en el argumento query especificamos una condición para realizar el export de esa tabla

"Export interactivo"

Otra forma de realizar un export es poniendo simplemente en la línea de comando exp y esperar a que te vaya pidiendo la utilidad los parámetros que requiere.

De esta forma te pedirá el usuario, contraseña, si quieres hacer copia sólo de la estructura, con datos, sin datos, nombre del archivo ... etc.

Recomendaciones

Se pueden hacer **exports con diferentes versiones de Oracle**, aunque es recomendable realizar el export con la misma versión de la base de datos. En todo caso si se hace con un cliente, si la versión es superior o la misma a la de la base de datos, Oracle "asegura" que no existe ningún problema en realizarlo.

Import oracle 10g

Partiendo de un archivo realizado con la utilidad export podemos recuperar datos de toda la base de datos, de ciertas tablas, etc.

El **comando** para realizar export en Oracle 9i es **imp**

Para ver todos los argumentos y significado de ellos de un export basta con realizar un `imp help=yes`

Ejemplo1: Importar todo el archivo exportado

```
C:\imp system/manager file=c:\orasitefull.dmp full=yes  
ignore=yes  
log=c:\orasite\log\import_log.log buffer=1000000
```

Importamos con el usuario system que tiene permisos para importar el archivo orasitefull.dmp dejando un log de dicha importación en import_log.log reservando un buffer de 100000

Ejemplo2: Importar una tabla de un usuario concreto

```
C:\imp scott/tiger file=orasitempleados.dmp  
fromuser=scott touser=scott tables=dept
```

Importamos del archivo orasitempleados.dmp sólo del usuario scott de ese archivo al esquema del usuario scott la tabla departamento

"Import interactivo"

Otra forma de realizar un import al igual que el export es introducir en la línea de comando **imp** y esperar a que te vaya pidiendo la utilidad los parámetros que requiere.

De esta forma te pedirá el usuario, contraseña, el archivo para importar, qué quieres importar del archivo ... etc.

PL-SQL

Identificadores

Un identificador es un nombre que se le pone a un objeto que interviene en un programa, que puede ser variable, constante, procedimientos, excepciones, cursores... Debe tener un máximo de 30 caracteres que empiece siempre por una letra, y puede contener letras, números, los símbolos \$, #, _, y mayúsculas y minúsculas indiferentemente. Los identificadores no pueden ser palabras reservadas (SELECT, INSERT, DELETE, UPDATE, DROP).

Operadores

Operador de asignación

:= (dos puntos + igual)

Operadores aritméticos + (suma)

- (resta)

* (multiplicación)

/ (división)

** (exponente)

Operadores relacionales o de comparación

= (igual a)

<>, != (distinto de)

< (menor que)

> (mayor que)

>= (mayor o igual a)

<= (menor o igual a)

Operador de concatenación || Comentarios /* comentario de dos o más líneas */

-- comentario de una línea

Variables

Las variables son nombres para procesar los elementos de los datos.

Declaración:

```
Nombre_variable tipo [NOT NULL] [:= valor | DEFAULT  
valor]
```

:= y **DEFAULT** son lo mismo. Si ponemos **NOT NULL** es obligatorio inicializar la variable.

Ejemplos:

```
num_dep NUMBER(2) NOT NULL :=20  
num_emple VARCHAR2(15) DEFAULT 'Pedro'
```

También se puede definir una variable a partir de un campo mediante los atributos **%TYPE** y **%ROWTYPE**, con esto damos el tipo y longitud a la variable de otra variable u objeto ya definido.

%TYPE es la que se utiliza normalmente, **%ROWTYPE** es para claves de registro. El **NOT NULL** y el valor inicial no se heredan, sólo el tipo de dato y longitud de ese dato.

Por ejemplo:

```
num_dep emple.dept_no%TYPE
```

Constantes

Las constantes son como las variables pero no puede modificarse su valor. Se declaran de la siguiente manera:

```
nombre_constante CONSTANT tipo_de_dato := valor
```

Por ejemplo, el IVA es un valor fijo, y para declararlo lo haríamos de la siguiente manera:

```
Imp_iva constant number(2,2) := 12,5
```

Funciones integradas de PL/SQL

PL/SQL tiene un gran número de funciones incorporadas, sumamente útiles. A continuación vamos a ver algunas de las más utilizadas.

SYSDATE

Devuelve la fecha del sistema:

```
SELECT SYSDATE FROM DUAL;
```

NVL

Devuelve el valor recibido como parámetro en el caso de que expresión sea NULL, o expresión en caso contrario.

```
NVL(<expresion>, <valor>)
```

El siguiente ejemplo devuelve 0 si el precio es nulo, y el precio cuando está informado:

```
SELECT CO_PRODUCTO, NVL(PRECIO, 0) FROM PRECIOS;
```

DECODE

Decode proporciona la funcionalidad de una sentencia de control de flujo *if-elseif-else*.

```
DECODE(<expr>, <cond1>, <val1>[, ..., <condN>, <valN>],  
<default>)
```

Esta función evalúa una expresión "<expr>", si se cumple la primera condición "<cond1>" devuelve el valor1 "<vall>", en caso contrario evalúa la siguiente condición y así hasta que una de las condiciones se cumpla. Si no se cumple ninguna condición se devuelve el valor por defecto.

Es muy común escribir la función DECODE identada como si se tratase de un bloque IF.

```
SELECT DECODE (co_pais, /* Expresion a evaluar */
               'ESP', 'ESPAÑA', /* Si co_pais = 'ESP' ==>
               'ESPAÑA' */
               'MEX', 'MEXICO', /* Si co_pais = 'MEX' ==>
               'MEXICO' */
               'PAIS '||co_pais)/* ELSE ==> concatena */
FROM PAISES;
```

TO_DATE

Convierte una expresión al tipo fecha. El parámetro opcional formato indica el formato de entrada de la expresión no el de salida.

TO_DATE(<expresion>, [<formato>])

En este ejemplo convertimos la expresion '01/12/2006' de tipo CHAR a una fecha (tipo DATE). Con el parámetro formato le indicamos que la fecha está escrita como día-mes-año para que devuelve el uno de diciembre y no el doce de enero.

```
SELECT TO_DATE ('01/12/2006',
               'DD/MM/YYYY')
FROM DUAL;
```

Este otro ejemplo muestra la conversión con formato de día y hora.

```
SELECT TO_DATE('31/12/2006 23:59:59',
               'DD/MM/YYYY HH24:MI:SS')
FROM DUAL;
```

TO_CHAR

Convierte una expresión al tipo CHAR. El parámetro opcional formato indica el formato de salida de la expresión.

```
TO_CHAR(<expresion>, [<formato>])
```

```
SELECT TO_CHAR(SYSDATE, 'DD/MM/YYYY')
FROM DUAL;
```

TO_NUMBER

Convierte una expresión alfanumérica en numérica. Opcionalmente podemos especificar el formato de salida.

```
TO_NUMBER(<expresion>, [<formato>])
```

```
SELECT TO_NUMBER('10')
FROM DUAL;
```

TRUNC

Trunca una fecha o número.

Si el parámetro recibido es una fecha elimina las horas, minutos y segundos de la misma.

```
SELECT TRUNC(SYSDATE) FROM DUAL;
```

Si el parámetro es un número devuelve la parte entera.

```
SELECT TRUNC(9.99) FROM DUAL;
```

LENGTH

Devuelve la longitud de un tipo CHAR.

```
SELECT LENGTH('HOLA MUNDO') FROM DUAL;
```

INSTR

Busca una cadena de caracteres dentro de otra. Devuelve la posición de la ocurrencia de la cadena buscada.

Su sintaxis es la siguiente:

```
INSTR(<char>, <search_string>, <startpos>, <occurrence> )
```

```
SELECT INSTR('AQUI ES DONDE SE BUSCA', 'BUSCA', 1, 1 )  
FROM DUAL;
```

REPLACE

Reemplaza un texto por otro en una expresión de búsqueda.

```
REPLACE(<expresion>, <busqueda>, <reemplazo>)
```

El siguiente ejemplo reemplaza la palabra 'HOLA' por 'VAYA' en la cadena 'HOLA MUNDO'.

```
SELECT REPLACE ('HOLA MUNDO', 'HOLA', 'VAYA') -- devuelve  
VAYA MUNDO  
FROM DUAL;
```

SUBSTR

Obtiene una parte de una expresión, desde una posición de inicio hasta una determinada longitud.

```
SUBSTR(<expresion>, <posicion_ini>, <longitud> )
```

```
SELECT SUBSTR('HOLA MUNDO', 6, 5) -- Devuelve MUNDO  
FROM DUAL;
```

UPPER

Convierte una expresion alfanumerica a mayúsculas.

```
SELECT UPPER('hola mundo') -- Devuelve HOLA MUNDO  
FROM DUAL;
```

LOWER

Convierte una expresion alfanumerica a minúsculas.

```
SELECT LOWER('HOLA MUNDO') -- Devuelve hola mundo  
FROM DUAL;
```

ROWIDTOCHAR

Convierte un ROWID a tipo caracter.

```
SELECT ROWIDTOCHAR(ROWID)  
FROM DUAL;
```

RPAD

Añade N veces una determinada cadena de caracteres a la derecha una expresión. Muy util para generar ficheros de texto de ancho fijo.

```
RPAD(<expresion>, <longitud>, <pad_string> )
```

El siguiente ejemplo añade puntos a la expresion 'Hola mundo' hasta alcanzar una longitud de 50 caracteres.

```
SELECT RPAD('Hola Mundo', 50, '.')
FROM DUAL;
```

LPAD

Añade N veces una determinada cadena de caracteres a la izquierda de una expresión. Muy útil para generar ficheros de texto de ancho fijo.

```
LPAD(<expresion>, <longitud>, <pad_string> )
```

El siguiente ejemplo añade puntos a la expresión 'Hola mundo' hasta alcanzar una longitud de 50 caracteres.

```
SELECT LPAD('Hola Mundo', 50, '.')
FROM DUAL;
```

RTRIM

Elimina los espacios en blanco a la derecha de una expresión

```
SELECT RTRIM ('Hola Mundo   ')
FROM DUAL;
```

LTRIM

Elimina los espacios en blanco a la izquierda de una expresión

```
SELECT LTRIM ('   Hola Mundo')
FROM DUAL;
```

TRIM

Elimina los espacios en blanco a la izquierda y derecha de una expresión

```
SELECT TRIM ('   Hola Mundo   ')
FROM DUAL;
```

MOD

Devuelve el resto de la división entera entre dos números.

```
MOD(<dividendo>, <divisor> )
```

```
SELECT MOD(20,15) -- Devuelve el modulo de dividir 20/15
FROM DUAL
```

Bloque PL/SQL

Bloque es la unidad de estructura básica en los programas PL/SQL. Supone una mejora en el rendimiento, pues se envían los bloques completos al servidor para ser procesados en lugar de enviar cada secuencia SQL.

Partes de un bloque:

- Zona de declaraciones: zona opcional. Se declaran los objetos locales (variables, constantes...).
- Zona de instrucciones: zona obligatoria.
- Zona de tratamiento de excepciones: zona opcional. Se tratan excepciones en el programa.

Forma de crear un bloque:

```
[ DECLARE      |      IS / AS ]
      <declaraciones>

BEGIN

      <instrucciones>

[ EXCEPTION ]
      <tratamiento de excepciones>

END;

/
```

La barra "/" siempre se pone al final para ejecutar el bloque.

Excepciones en PL/SQL

En PL/SQL una advertencia o condición de error es llamada una excepción.

Las excepciones se controlan dentro de su propio bloque. La estructura de bloque de una excepción se muestra a continuación.

```
DECLARE
    -- Declaraciones
BEGIN
    -- Ejecucion
EXCEPTION
    -- Excepcion
END;
```

Cuando ocurre un error, se ejecuta la porción del programa marcada por el bloque **EXCEPTION**, transfiriéndose el control a ese bloque de sentencias.

El siguiente ejemplo muestra un bloque de excepciones que captura las excepciones **NO_DATA_FOUND** y **ZERO_DIVIDE**. Cualquier otra excepción será capturada en el bloque **WHEN OTHERS THEN**.

```
DECLARE
    -- Declaraciones
BEGIN
    -- Ejecucion
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        -- Se ejecuta cuando ocurre una excepcion de tipo
        NO_DATA_FOUND
    WHEN ZERO_DIVIDE THEN
        -- Se ejecuta cuando ocurre una excepcion de tipo
        ZERO_DIVIDE

    WHEN OTHERS THEN
        -- Se ejecuta cuando ocurre una excepcion de un tipo no
        tratado
        -- en los bloques anteriores

END;
```

Como ya hemos dicho cuando ocurre un error, se ejecuta el bloque **EXCEPTION**, transfiriéndose el control a las sentencias del bloque. Una vez finalizada la ejecución del bloque de **EXCEPTION** no se continua ejecutando el bloque anterior.

Si existe un bloque de excepcion apropiado para el tipo de excepción se ejecuta dicho bloque. Si no existe un bloque de control de excepciones adecuado al tipo de excepcion se ejecutará el bloque de excepcion **WHEN OTHERS THEN** (si existe!). **WHEN OTHERS** debe ser el último manejador de excepciones.

Las excepciones pueden ser definidas en forma interna o explícitamente por el usuario. Ejemplos de excepciones definidas en forma interna son la división por cero y la falta de memoria en tiempo de ejecución. Estas mismas condiciones excepcionales tienen sus propio tipos y pueden ser referenciadas por ellos: **ZERO_DIVIDE** y **STORAGE_ERROR**.

Las excepciones definidas por el usuario deben ser alcanzadas explícitamente utilizando la sentencia **RAISE**.

Con las excepciones se pueden manejar los errores cómodamente sin necesidad de mantener múltiples chequeos por cada sentencia escrita. También provee claridad en el código ya que permite mantener las rutinas correspondientes al tratamiento de los errores de forma separada de la lógica del negocio.

Excepciones predefinidas

PL/SQL proporciona un gran número de excepciones predefinidas que permiten controlar las condiciones de error más habituales.

Las excepciones predefinidas no necesitan ser declaradas. Simplemente se utilizan cuando estas son lanzadas por algún error determinado.

La siguiente es la lista de las excepciones predeterminadas por PL/SQL y una breve descripción de cuándo son accionadas:

Excepcion	Se ejecuta ...	SQLCODE
		DE

ACCESS_INT0_NULL	El programa intentó asignar valores a los atributos de un objeto no inicializado	-6530
COLLECTION_IS_NULL	El programa intentó asignar valores a una tabla anidada aún no inicializada	-6531
CURSOR_ALREADY_OPEN	El programa intentó abrir un cursor que ya se encontraba abierto. Recuerde que un cursor de ciclo FOR automáticamente lo abre y ello no se debe especificar con la sentencia OPEN	-6511
DUP_VAL_ON_INDEX	El programa intentó almacenar valores duplicados en una columna que se mantiene con restricción de integridad de un índice único (unique index)	-1

INVALID_CURSOR	El programa intentó efectuar una operación no válida sobre un cursor	-1001
INVALID_NUMBER	En una sentencia SQL, la conversión de una cadena de caracteres hacia un número falla cuando esa cadena no representa un número válido	-1722
LOGIN_DENIED	El programa intentó conectarse a Oracle con un nombre de usuario o password inválido	-1017
NO_DATA_FOUND	Una sentencia SELECT INTO no devolvió valores o el programa referenció un elemento no inicializado en una tabla indexada	100

NOT_LOGGED_ON	El programa efectuó una llamada a Oracle sin estar conectado	-1012
PROGRAM_ERROR	PL/SQL tiene un problema interno	-6501
ROWTYPE_MISMATCH	Los elementos de una asignación (el valor a asignar y la variable que lo contendrá) tienen tipos incompatibles. También se presenta este error cuando un parámetro pasado a un subprograma no es del tipo esperado	-6504
SELF_IS_NULL	El parámetro SELF (el primero que es pasado a un método MEMBER) es nulo	-30625
STORAGE_ERROR	La memoria se terminó o está corrupta	-6500

SUBSCRIPT_BEYOND_COUNT	El programa está tratando de referenciar un elemento de un arreglo indexado que se encuentra en una posición más grande que el número real de elementos de la colección	-6533
SUBSCRIPT_OUTSIDE_LIMIT	El programa está referenciando un elemento de un arreglo utilizando un número fuera del rango permitido (por ejemplo, el elemento “-1”)	-6532
SYS_INVALID_ROWID	La conversión de una cadena de caracteres hacia un tipo rowid falló porque la cadena no representa un número	-1410
TIMEOUT_ON_RESOURCE	Se excedió el tiempo máximo de espera por un recurso en Oracle	-51

TOO_MANY_ROWS	Una sentencia SELECT INTO devuelve más de una fila	-1422
VALUE_ERROR	Ocurrió un error aritmético, de conversión o truncamiento. Por ejemplo, sucede cuando se intenta calzar un valor muy grande dentro de una variable más pequeña	-6502
ZERO_DIVIDE	El programa intentó efectuar una división por cero	-1476

Excepciones definidas por el usuario

PL/SQL permite al usuario definir sus propias excepciones, las que deberán ser declaradas y lanzadas explícitamente utilizando la sentencia **RAISE**.

Las excepciones deben ser declaradas en el segmento **DECLARE** de un bloque, subprograma o paquete. Se declara una excepción como cualquier otra variable, asignándole el tipo **EXCEPTION**. Las mismas reglas de alcance aplican tanto sobre variables como sobre las excepciones.

```

DECLARE
    -- Declaraciones

    MyExcepcion EXCEPTION;
BEGIN
```

```

-- Ejecucion
EXCEPTION
-- Excepcion
END;

```

Reglas de Alcance

Una excepcion es válida dentro de su ambito de alcance, es decir el bloque o programa donde ha sido declarada. Las excepciones predefinidas son siempre válidas.

Como las variables, una excepción declarada en un bloque es local a ese bloque y global a todos los sub-bloques que comprende.

La sentencia RAISE

La sentencia **RAISE** permite lanzar una excepción en forma explícita.

Es posible utilizar esta sentencia en cualquier lugar que se encuentre dentro del alcance de la excepción.

```

DECLARE
-- Declaramos una excepcion identificada por VALOR_NEGATIVO
VALOR_NEGATIVO EXCEPTION;
valor NUMBER;
BEGIN
-- Ejecucion
valor := -1;
IF valor < 0 THEN
RAISE VALOR_NEGATIVO;
END IF;

EXCEPTION
-- Excepcion
WHEN VALOR_NEGATIVO THEN
dbms_output.put_line('El valor no puede ser negativo');
END;

```


Con la sentencia **RAISE** podemos lanzar una excepción definida por el usuario o predefinida, siendo el comportamiento habitual lanzar excepciones definidas por el usuario.

Recordar la existencia de la excepción **OTHERS**, que simboliza cualquier condición de excepción que no ha sido declarada. Se utiliza comúnmente para controlar cualquier tipo de error que no ha sido previsto. En ese caso, es común observar la sentencia **ROLLBACK** en el grupo de sentencias de la excepción o alguna de las funciones **SQLCODE** – **SQLERRM**, que se detallan en el próximo punto.

Uso de SQLCODE y SQLERRM

Al manejar una excepción es posible usar las funciones predefinidas **SQLCode** y **SQLERRM** para aclarar al usuario la situación de error acontecida.

SQLcode devuelve el número del error de Oracle y un 0 (cero) en caso de éxito al ejecutarse una sentencia SQL.

Por otra parte, **SQLERRM** devuelve el correspondiente mensaje de error.

Estas funciones son muy útiles cuando se utilizan en el bloque de excepciones, para aclarar el significado de la excepción **OTHERS**.

Estas funciones no pueden ser utilizadas directamente en una sentencia SQL, pero sí se puede asignar su valor a alguna variable de programa y luego usar esta última en alguna sentencia.

```
DECLARE

    err_num NUMBER;
    err_msg VARCHAR2(255);
    result  NUMBER;

BEGIN

    SELECT 1/0 INTO result
    FROM DUAL;

EXCEPTION

WHEN OTHERS THEN

    err_num := SQLCODE;
```

```

err_msg := SQLERRM;

DBMS_OUTPUT.put_line('Error: ' || TO_CHAR(err_num));
DBMS_OUTPUT.put_line(err_msg);

END;

```

También es posible entregarle a la función **SQLERRM** un número negativo que represente un error de Oracle y ésta devolverá el mensaje asociado.

```

DECLARE

msg VARCHAR2(255);

BEGIN

msg := SQLERRM(-1403);

DBMS_OUTPUT.put_line(MSG);

END;

```

Excepciones personalizadas en PL/SQL

RAISE_APPLICATION_ERROR

En ocasiones queremos enviar un mensaje de error personalizado al producirse una excepción PL/SQL.

Para ello es necesario utilizar la instrucción

RAISE_APPLICATION_ERROR;

La sintaxis general es la siguiente:

```

RAISE_APPLICATION_ERROR(<error_num>,<mensaje>);

```

Siendo:

- error_num es un entero negativo comprendido entre -20001 y -20999
- mensaje la descripción del error

```

DECLARE

v_div NUMBER;

BEGIN

SELECT 1/0 INTO v_div FROM DUAL;

EXCEPTION

WHEN OTHERS THEN

```

```
RAISE_APPLICATION_ERROR(-20001, 'No se puede dividir por  
cero');  
END;
```

Cursores en PL/SQL

Introducción a cursores PL/SQL

PL/SQL utiliza cursores para gestionar las instrucciones **SELECT**. Un cursor es un conjunto de registros devuelto por una instrucción SQL. Técnicamente los cursores son fragmentos de memoria que reservados para procesar los resultados de una consulta **SELECT**.

Podemos distinguir dos tipos de cursores:

- **Cursores implícitos.** Este tipo de cursores se utiliza para operaciones **SELECT INTO**. Se usan cuando la consulta devuelve un único registro.
- **Cursores explícitos.** Son los cursores que son declarados y controlados por el programador. Se utilizan cuando la consulta devuelve un conjunto de registros. Ocasionalmente también se utilizan en consultas que devuelven un único registro por razones de eficiencia. Son más rápidos.

Un cursor se define como cualquier otra variable de PL/SQL y debe nombrarse de acuerdo a los mismos convenios que cualquier otra variable. Los cursores implícitos no necesitan declaración.

El siguiente ejemplo declara un cursor explícito:

```
declare  
    cursor c_paises is  
        SELECT CO_PAIS, DESCRIPCION  
        FROM PAISES;  
begin  
    /* Sentencias del bloque ... */  
end;
```

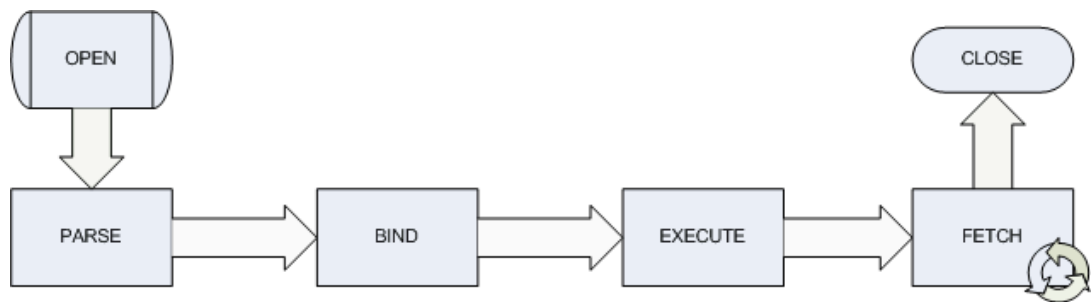
Para procesar instrucciones **SELECT** que devuelvan más de una fila, son necesarios cursores explícitos combinados con un estructura de bloque.

Un cursor admite el uso de parámetros. Los parámetros deben declararse junto con el cursor.

El siguiente ejemplo muestra la declaración de un cursor con un parámetro, identificado por p_continente.

```
declare  
  cursor c_paises (p_continente IN VARCHAR2) is  
  SELECT CO_PAIS, DESCRIPCION  
  FROM PAISES  
  
  WHERE CONTINENTE = p_continente;  
begin  
  /* Sentencias del bloque ... */  
end;
```

El siguiente diagrama representa como se procesa una instrucción SQL a través de un cursor.



Fases para procesar una instrucción SQL

Cursores Implícitos en PL/SQL

Declaración de cursores implícitos.

Los cursores implícitos se utilizan para realizar consultas SELECT que devuelven un único registro.

Deben tenerse en cuenta los siguientes puntos cuando se utilizan cursores implícitos:

- Con cada cursor implícito debe existir la palabra clave **INTO**.

Las variables que reciben los datos devueltos por el cursor tienen que contener el mismo tipo de dato que las columnas de la tabla.

Los cursores implícitos solo pueden devolver una única fila. En caso de que se devuelva más de una fila (o ninguna fila) se producirá una excepción. No se preocupe si aún no sabe que es una excepción, le valdrá conocer que es el medio por el que PL/SQL gestiona los errores.

El siguiente ejemplo muestra un cursor implícito:

```
declare
vdescripcion VARCHAR2(50);
begin
    SELECT DESCRIPCION
    INTO vdescripcion
    from PAISES
    WHERE CO_PAIS = 'ESP';

    dbms_output.put_line('La lectura del cursor es: ' ||
vdescripcion);

end;
```

La salida del programa generaría la siguiente línea:

La lectura del cursor es: ESPAÑA

Excepciones asociadas a los cursores implícitos.

Los cursores implícitos sólo pueden devolver una fila, por lo que pueden producirse determinadas excepciones. Las más comunes que se pueden encontrar son `no_data_found` y `too_many_rows`. La siguiente tabla explica brevemente estas excepciones.

Excepcion	Explicacion
NO_DATA_FOUND	Se produce cuando una sentencia SELECT intenta recuperar datos pero ninguna fila satisface sus condiciones. Es decir, cuando <i>"no hay datos"</i>
TOO_MANY_ROWS	Dado que cada cursor implicito sólo es capaz de recuperar una fila , esta excepcion detecta la existencia de más de una fila.

Cursores Explicitos en PL/SQL

Declaración de cursores explicitos

Los cursores explicitos se emplean para realizar consultas SELECT que pueden devolver cero filas, o más de una fila.

Para trabajar con un cursor explicito necesitamos realizar las siguientes tareas:

Declarar el cursor.

- Abrir el cursor con la instrucción **OPEN**.
- Leer los datos del cursor con la instrucción **FETCH**.
- Cerrar el cursor y liberar los recursos con la instrucción **CLOSE**.

Para declarar un cursor debemos emplear la siguiente sintaxis:

```
CURSOR nombre_cursor IS  
  
instrucción_SELECT
```

También debemos declarar los posibles parametros que requiera el cursor:

```
CURSOR nombre_cursor(param1 tipo1, ..., paramN tipoN) IS  
  
instrucción_SELECT
```

Para abrir el cursor

```
OPEN nombre_cursor;  
  
o bien (en el caso de un cursor con parámetros)  
  
OPEN nombre_cursor(valor1, valor2, ..., valorN);
```

Para recuperar los datos en variables PL/SQL.

```
FETCH nombre_cursor INTO lista_variaciones;  
  
-- o bien ...  
  
FETCH nombre_cursor INTO registro_PL/SQL;
```

Para cerrar el cursor:

```
CLOSE nombre_cursor;
```

El siguiente ejemplo ilustra el trabajo con un cursor explícito. Hay que tener en cuenta que al leer los datos del cursor debemos hacerlo sobre variables del mismo tipo de datos de la tabla (o tablas) que trata el cursor.

```
DECLARE  
  
CURSOR cpaises  
  
IS  
  
SELECT CO_PAIS, DESCRIPCION, CONTINENTE  
FROM PAISES;  
  
  
  
co_pais VARCHAR2(3);  
descripcion VARCHAR2(50);
```

```

        continente  VARCHAR2 (25);
BEGIN
    OPEN cpaíses;
    FETCH cpaíses INTO co_pais, descripcion, continente;
    CLOSE cpaíses;
END;

```

Podemos simplificar el ejemplo utilizando el atributo de tipo [%ROWTYPE](#) sobre el cursor.

```

DECLARE
    CURSOR cpaíses
    IS
        SELECT CO_PAIS, DESCRIPCION, CONTINENTE
        FROM PAISES;

    registro cpaíses%ROWTYPE;
BEGIN
    OPEN cpaíses;
    FETCH cpaíses INTO registro;
    CLOSE cpaíses;
END;

```

El mismo ejemplo, pero utilizando parámetros:

```

DECLARE
    CURSOR cpaíses (p_continente VARCHAR2)
    IS
        SELECT CO_PAIS, DESCRIPCION, CONTINENTE
        FROM PAISES
        WHERE CONTINENTE = p_continente;

    registro cpaíses%ROWTYPE;
BEGIN
    OPEN cpaíses ('EUROPA');
    FETCH cpaíses INTO registro;
    CLOSE cpaíses;
END;

```


Cuando trabajamos con cursores debemos considerar:

- Cuando un cursor está cerrado, no se puede leer.
- Cuando leemos un cursor debemos comprobar el resultado de la lectura utilizando los atributos de los cursores.
- Cuando se cierra el cursor, es ilegal tratar de usarlo.
- Es ilegal tratar de cerrar un cursor que ya está cerrado o no ha sido abierto

Atributos de cursores

Toman los valores TRUE, FALSE o NULL dependiendo de la situación:

Atributo	Antes de abrir	Al abrir	Durante la recuperación	Al finalizar la recuperación	Después de cerrar
%NOT-FOUND	ORA-1001	NULL	FALSE	TRUE	ORA-1001
%FOUND	ORA-1001	NULL	TRUE	FALSE	ORA-1001
%ISOPEN	FALSE	TRUE	TRUE	TRUE	FALSE
%ROW-COUNT	ORA-1001	0	*	**	ORA-1001

* Número de registros que ha recuperado hasta el momento

** Número de total de registros

Manejo del cursor

Por medio de ciclo LOOP podemos iterar a través del cursor. Debe tenerse cuidado de agregar una condición para salir del bucle:

Vamos a ver varias formas de iterar a través de un cursor. La primera es utilizando un

bucle LOOP con una sentencia EXIT condicionada:

```
OPEN nombre_cursor;

LOOP

    FETCH nombre_cursor INTO lista_variables;

    EXIT WHEN nombre_cursor%NOTFOUND;

    /* Procesamiento de los registros recuperados */

END LOOP;

CLOSE nombre_cursor;
```

Aplicada a nuestro ejemplo anterior:

```
DECLARE

CURSOR cpaíses
IS
SELECT CO_PAIS, DESCRIPCION, CONTINENTE
FROM PAISES;

co_pais VARCHAR2(3);
descripcion VARCHAR2(50);
continente VARCHAR2(25);

BEGIN

OPEN cpaíses;

LOOP

    FETCH cpaíses INTO co_pais,descripcion,continente;

    EXIT WHEN cpaíses%NOTFOUND;

    dbms_output.put_line(descripcion);

END LOOP;

CLOSE cpaíses;

END;
```

Otra forma es por medio de un bucle WHILE LOOP. La instrucción FECTH aparece

dos veces.

```
OPEN nombre_cursor;

FETCH nombre_cursor INTO lista_variables;

WHILE nombre_cursor%FOUND
LOOP

    /* Procesamiento de los registros recuperados */

    FETCH nombre_cursor INTO lista_variables;

END LOOP;

CLOSE nombre_cursor;
```

DECLARE

CURSOR cpaises

IS

SELECT CO_PAIS, DESCRIPCION, CONTINENTE
FROM PAISES;

co_pais **VARCHAR2**(3);

descripcion **VARCHAR2**(50);

continente **VARCHAR2**(25);

BEGIN

OPEN cpaises;

FETCH cpaises **INTO** co_pais,descripcion,continente;

WHILE cpaises%found

LOOP

dbms_output.put_line(descripcion);

FETCH cpaises **INTO** co_pais,descripcion,continente;

END LOOP;

CLOSE cpaises;

END;

Por último podemos usar un bucle FOR LOOP. Es la forma más corta ya que el cursor es implícitamente se ejecutan las instrucciones OPEN, FETCH y CLOSE.

```
FOR variable IN nombre_cursor LOOP  
    /* Procesamiento de los registros recuperados */  
END LOOP;
```

```
BEGIN  
    FOR REG IN (SELECT * FROM PAISES)  
    LOOP  
        dbms_output.put_line(reg.descripcion);  
    END LOOP;  
END;
```

Cursores de actualización en PL/SQL

Declaración y utilización de cursores de actualización.

Los cursores de actualización se declaran igual que los cursores explícitos, añadiendo **FOR UPDATE** al final de la sentencia select.

```
CURSOR nombre_cursor IS  
    instrucción_SELECT  
FOR UPDATE
```

Para actualizar los datos del cursor hay que ejecutar una sentencia **UPDATE** especificando la clausula **WHERE CURRENT OF** *<cursor_name>*.

```
UPDATE <nombre_tabla> SET  
<campo_1> = <valor_1>  
[,<campo_2> = <valor_2>]  
WHERE CURRENT OF <cursor_name>
```

El siguiente ejemplo muestra el uso de un cursor de actualización:

```
DECLARE  
  
    CURSOR cpaíses IS  
        select CO_PAIS, DESCRIPCION, CONTINENTE  
        from países  
        FOR UPDATE;  
    co_pais VARCHAR2(3);  
    descripcion VARCHAR2(50);  
    continente VARCHAR2(25);  
  
BEGIN  
    OPEN cpaíses;  
    FETCH cpaíses INTO co_pais,descripcion,continente;  
    WHILE cpaíses%found  
    LOOP  
        UPDATE PAISES  
            SET CONTINENTE = CONTINENTE || '.'  
            WHERE CURRENT OF cpaíses;  
  
        FETCH cpaíses INTO co_pais,descripcion,continente;  
    END LOOP;  
    CLOSE cpaíses;  
    COMMIT;  
  
END;
```

Cuando trabajamos con cursores de actualización debemos tener en cuenta las siguientes consideraciones:

Los cursores de actualización generan bloqueos en la base de datos.

Bloques Anonimos (sin nombre)

Siempre comienza con **DECLARE** o directamente con **BEGIN**.

Ejemplo 1:

```
BEGIN

DBMS_OUTPUT.PUT_LINE ('Hola');

END;

/
```

DBMS_OUTPUT es un depurador de Oracle que sirve para visualizar cualquier cosa, pero antes lo debemos tener activado:

```
SET SERVEROUTPUT ON;
```

Ejemplo 2:

```
DECLARE

v_precio number;

BEGIN

select pvp into v_precio

from tarticulos

where codigo=100;

dbms_output.put_line (v_precio);

END;

/
```

Ejemplo 3:

El siguiente bloque anónimo nos muestra la fecha actual con el formato “Martes, 18 de Marzo de 1998, a las 13:04:55”.

```
DECLARE

        fecha date;

BEGIN

select sysdate into fecha from dual;
```

```

dbms_output.put_line (to_char(sysdate,
'day", "dd" de "month" de "yyyy", a las "hh24:mi:ss'));
END;
/

```

Subprogramas

Se pueden almacenar en la base de datos.

Existen dos tipos de subprogramas:

Procedimientos

Los procedimientos tienen la utilidad de fomentar la reutilización de programas que se usan comúnmente. Una vez compilado, queda almacenado en la base de datos y puede ser utilizado por múltiples aplicaciones.

La sintaxis es la siguiente

```

CREATE [OR REPLACE] PROCEDURE nombre_procedimiento
[nombre_parametro modo tipodatos_parametro ]
IS | AS
bloque de código

```

Donde "modo" puede contener los valores IN, OUT, IN OUT. Por defecto tiene el valor IN si no se pone nada. IN indica que el parámetro es de entrada y no se podrá modificar. OUT indica que el parámetro es de salida con lo que el procedimiento devolverá un valor en él. IN OUT indica que el parámetro es de entrada/salida. Con lo que al llamar al procedimiento se le dará un valor que luego podrá ser modificado por el procedimiento y devolver este nuevo valor.

"tipodatos_parametro indica el tipo de datos que tendrá el parámetro según lo indicado en Tipos de datos Oracle/PLSQL

Para borrar un procedimiento almacenado de la base de datos

```

DROP PROCEDURE nombre_procedimiento

```

Para utilizar un procedimiento almacenado de la base de datos

Simplemente se lo llama desde un bloque anónimo (desde la línea de comandos), previamente habiendo inicializado el/los parametro/s (en caso que existan).

```

DECLARE

    nombre_parametro tipodatos_parametro;

BEGIN

    nombre_parametro tipodatos_parametro :=
valor_de_inicializacion;

    nombre_procedimiento (nombre_parametro =>
nombre_parametro);

END;

/

```

Funciones

Una función es un bloque de código PL/SQL que tiene las mismas características que un procedimiento almacenado. La diferencia estriba que una función devuelve un valor al retornar. Al devolver un valor puede ser llamada como parte de una expresión.

La sintaxis sería

```

CREATE [OR REPLACE] FUNCTION nombre_función

    [nombre_parámetro modo tipodatos_parametro ]

RETURN tipodatos_retorno IS | AS

    bloque de código

```

Donde "modo" puede contener los valores IN, OUT, IN OUT. Por defecto tiene el valor IN si no se pone nada. IN indica que el parámetro es de entrada y no se podrá modificar. OUT indica que el parámetro es de salida con lo que el procedimiento devolverá un valor en él. IN OUT indica que el parámetro es de entrada/salida. Con lo que al llamar al procedimiento se le dará un valor que luego podrá ser modificado por el procedimiento y devolver este nuevo valor. Sin embargo, en este caso solo tendría sentido(por el concepto de función en sí mismo) declarar parámetros del tipo IN y devolver el valor como retorno de la función.

"tipodatos_parametro" y "tipodatos_retorno" indican el tipo de datos que tendrá el parámetro y el valor de retorno de la función respectivamente según lo indicado en Tipos de datos Oracle/PLSQL

Para borrar una función de la base de datos

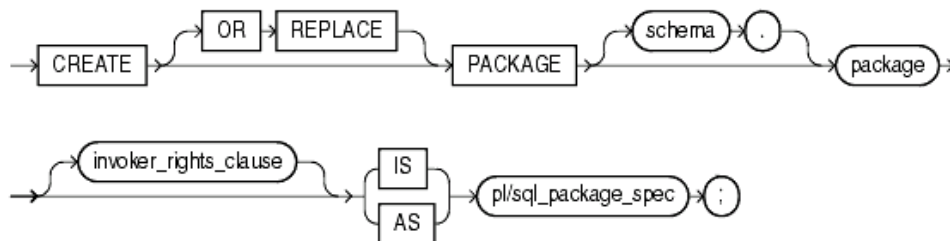
```
DROP FUNCTION nombre_función
```

Los procedimientos y funciones se pueden agrupar en unas estructuras llamadas Paquetes

Paquetes

Los paquetes se utilizan para agrupar procedimiento y funciones,

Ejemplo de declaración de un Paquete de base de datos:



```
CREATE [OR REPLACE] PACKAGE package_name  
    [AUTHID {CURRENT_USER | DEFINER}]  
    {IS | AS}  
    [PRAGMA SERIALLY_REUSABLE;]  
    [collection_type_definition ...]  
    [record_type_definition ...]  
    [subtype_definition ...]  
    [collection_declaration ...]  
    [constant_declaration ...]  
    [exception_declaration ...]  
    [object_declaration ...]  
    [record_declaration ...]  
    [variable_declaration ...]  
    [cursor_spec ...]  
    [function_spec ...]
```

```

    [procedure_spec ...]

    [call_spec ...]

    [PRAGMA RESTRICT_REFERENCES(assertions) ...]

END [package_name];

[CREATE [OR REPLACE] PACKAGE BODY package_name {IS | AS}

    [PRAGMA SERIALLY_REUSABLE;]

    [collection_type_definition ...]

    [record_type_definition ...]

    [subtype_definition ...]

    [collection_declaration ...]

    [constant_declaration ...]

    [exception_declaration ...]

    [object_declaration ...]

    [record_declaration ...]

    [variable_declaration ...]

    [cursor_body ...]

    [function_spec ...]

    [procedure_spec ...]

    [call_spec ...]

[BEGIN

    sequence_of_statements]

END [package_name];]

```

Para Borrar un paquete:

```

DROP PACKAGE nombre_paquEte;

DROP PACKAGE BODY nombre_paquEte; --Elimina el cuerpo del
paquete

```

El uso de OR REPLACE permite sobrescribir un paquete existente. Si se omite, y el paquete existe, se producirá, un error.

Disparadores (Triggers)

Declaración de triggers

Un trigger es un bloque PL/SQL asociado a una tabla, que se ejecuta como consecuencia de una determinada instrucción SQL (una operación DML: INSERT, UPDATE o DELETE) sobre dicha tabla.

La sintaxis para crear un trigger es la siguiente:

```
CREATE [OR REPLACE] TRIGGER <nombre_trigger>
{BEFORE|AFTER}
          {DELETE|INSERT|UPDATE [OF col1, col2, ...,
colN]
          [OR {DELETE|INSERT|UPDATE [OF col1, col2, ...,
colN]...}] }
ON <nombre_tabla>
[FOR EACH ROW [WHEN (<condicion>)]]
DECLARE
    -- variables locales
BEGIN
    -- Sentencias
[EXCEPTION]
    -- Sentencias control de excepcion
END <nombre_trigger>;
```

El uso de OR REPLACE permite sobrescribir un trigger existente. Si se omite, y el trigger existe, se producirá, un error.

Los triggers pueden definirse para las operaciones INSERT, UPDATE o DELETE, y pueden ejecutarse antes o después de la operación. El modificador BEFORE AFTER indica que el trigger se ejecutará antes o después de ejecutarse la sentencia SQL definida por DELETE INSERT UPDATE. Si incluimos el modificador OF el trigger solo se ejecutará cuando la sentencia SQL afecte a los campos incluidos en la lista.

El alcance de los disparadores puede ser la fila o de orden. El modificador FOR EACH ROW indica que el trigger se disparará cada vez que se realizan operaciones sobre una fila de la tabla. Si se acompaña del modificador WHEN, se establece una restricción; el trigger solo actuará, sobre las filas que satisfagan la restricción.

La siguiente tabla resume los contenidos anteriores.

Valor	Descripción
INSERT, DELETE, UPDATE	Define qué tipo de orden DML provoca la activación del disparador.
BEFORE , AFTER	Define si el disparador se activa antes o después de que se ejecute la orden.
FOR EACH ROW	Los disparadores con nivel de fila se activan una vez por cada fila afectada por la orden que provocó el disparo. Los disparadores con nivel de orden se activan sólo una vez, antes o después de la orden. Los disparadores con nivel de fila se identifican por la cláusula FOR EACH ROW en la definición del disparador.

La cláusula WHEN sólo es válida para los disparadores con nivel de fila.

Dentro del ambito de un trigger disponemos de las variables OLD y NEW . Estas variables se utilizan del mismo modo que cualquier otra variable PL/SQL, con la salvedad de que no es necesario declararlas, son de tipo **%ROWTYPE** y contienen una copia del registro antes (OLD) y despues(NEW) de la acción SQL (INSERT, UPDATE, DELTE) que ha ejecutado el trigger. Utilizando esta variable podemos acceder a los datos que se están insertando, actualizando o borrando.

El siguiente ejemplo muestra un trigger que inserta un registro en la tabla PRECIOS_PRODUCTOS cada vez que insertamos un nuevo registro en la tabla PRODUTOS:

```

CREATE OR REPLACE TRIGGER TR_PRODUCTOS_01
  AFTER INSERT ON PRODUCTOS
  FOR EACH ROW
DECLARE
  -- local variables
BEGIN
  INSERT INTO PRECIOS_PRODUCTOS
    (CO_PRODUCTO, PRECIO, FX_ACTUALIZACION)
  VALUES
    (:NEW.CO_PRODUCTO, 100, SYSDATE);
END ;

```

El trigger se ejecutará cuando sobre la tabla PRODUCTOS se ejecute una sentencia INSERT.

```

INSERT INTO PRODUCTOS
(CO_PRODUCTO, DESCRIPCION)
VALUES
('000100', 'PRODUCTO 000100');

```

Orden de ejecución de los triggers

Una misma tabla puede tener varios triggers. En tal caso es necesario conocer el orden en el que se van a ejecutar.

Los disparadores se activan al ejecutarse la sentencia SQL.

- Si existe, se ejecuta el disparador de tipo BEFORE (disparador previo) con nivel de orden.
- Para cada fila a la que afecte la orden:
- Se ejecuta si existe, el disparador de tipo BEFORE con nivel de fila.
- Se ejecuta la propia orden.
- Se ejecuta si existe, el disparador de tipo AFTER (disparador posterior) con nivel de fila.
- Se ejecuta, si existe, el disparador de tipo AFTER con nivel de orden.

Restricciones de los triggers

El cuerpo de un trigger es un bloque PL/SQL. Cualquier orden que sea legal en un bloque PL/SQL, es legal en el cuerpo de un disparador, con las siguientes restricciones:

Un disparador no puede emitir ninguna orden de control de transacciones:

COMMIT, ROLLBACK o SAVEPOINT. El disparador se activa como parte de la ejecución de la orden que provocó el disparo, y forma parte de la misma transacción que dicha orden. Cuando la orden que provoca el disparo es confirmada o cancelada, se confirma o cancela también el trabajo realizado por el disparador.

Por razones idénticas, ningún procedimiento o función llamado por el disparador puede emitir órdenes de control de transacciones.

El cuerpo del disparador no puede contener ninguna declaración de variables LONG o LONG RAW

Utilización de :OLD y :NEW

Dentro del ámbito de un trigger disponemos de las variables OLD y NEW

. Estas variables se utilizan del mismo modo que cualquier otra variable PL/SQL, con la salvedad de que no es necesario declararlas, son de tipo **%ROWTYPE** y contienen una copia del registro antes (OLD) y después (NEW) de la acción SQL (INSERT, UPDATE, DELETE) que ha ejecutado el trigger. Utilizando esta variable podemos acceder a los datos que se están insertando, actualizando o borrando.

La siguiente tabla muestra los valores de OLD y NEW.

ACCION SQL	OLD	NEW
INSERT	No definido; todos los campos toman valor NULL.	Valores que serán insertados cuando se complete la orden.
UPDATE	Valores originales de la fila, antes de la	Nuevos valores que serán escritos cuando se

	actualización.	complete la orden.
DELETE	Valores, antes del borrado de la fila.	No definidos; todos los campos toman el valor NULL.

Los registros OLD y NEW son sólo válidos dentro de los disparadores con nivel de fila.

Podemos usar OLD y NEW como cualquier otra variable PL/SQL.

Utilización de predicados de los triggers: INSERTING, UPDATING y DELETING

Dentro de un disparador en el que se disparan distintos tipos de órdenes DML (INSERT, UPDATE y DELETE), hay tres funciones booleanas que pueden emplearse para determinar de qué operación se trata. Estos predicados son INSERTING, UPDATING y DELETING.

Su comportamiento es el siguiente:

Predicado	Comportamiento
INSERTING	TRUE si la orden de disparo es INSERT; FALSE en otro caso.
UPDATING	TRUE si la orden de disparo es UPDATE; FALSE en otro caso.
DELETING	TRUE si la orden de disparo es DELETE; FALSE en otro caso.

Para eliminar un trigger:

```
DROP TRIGGER nombre_trigger
```

Registros PL/SQL

Cuando vimos los tipos de datos, omitimos intencionadamente ciertos tipos de datos. Estos son:

- Registros
- Tablas de PL
- VARRAY

Declaración de un registro.

Un registro es una estructura de datos en PL/SQL, almacenados en campos, cada uno de los cuales tiene su propio nombre y tipo y que se tratan como una sola unidad lógica.

Los campos de un registro pueden ser inicializados y pueden ser definidos como NOT NULL. Aquellos campos que no sean inicializados explícitamente, se inicializarán a NULL.

La sintaxis general es la siguiente:

```
TYPE <nombre> IS RECORD
(
  campo <tipo_datos> [NULL | NOT NULL]
  [, <tipo_datos>...]
);
```

El siguiente ejemplo crea un tipo PAIS, que tiene como campos el código, el nombre y el continente.

```
TYPE PAIS IS RECORD
(
  CO_PAIS      NUMBER ,
  DESCRIPCION VARCHAR2(50) ,
  CONTINENTE  VARCHAR2(20)

);
```


Los registros son un tipo de datos, por lo que podremos declarar variables de dicho tipo de datos.

DECLARE

```
TYPE PAIS IS RECORD
(
    CO_PAIS      NUMBER ,
    DESCRIPCION VARCHAR2(50),
    CONTINENTE  VARCHAR2(20)
);

/* Declara una variable identificada por miPAIS de tipo PAIS
   Esto significa que la variable miPAIS tendrá los campos
   ID, DESCRIPCION y CONTINENTE.

*/

miPAIS PAIS;

BEGIN

/* Asignamos valores a los campos de la variable.

*/

miPAIS.CO_PAIS := 27;
miPAIS.DESCRIPCION := 'ITALIA';
miPAIS.CONTINENTE := 'EUROPA';

END;
```

Los registros pueden estar anidados. Es decir, un campo de un registro puede ser de un tipo de dato de otro registro.

DECLARE

```
TYPE PAIS IS RECORD
(CO_PAIS      NUMBER ,
 DESCRIPCION VARCHAR2(50),
 CONTINENTE  VARCHAR2(20)
);

TYPE MONEDA IS RECORD
( DESCRIPCION VARCHAR2(50),
```

```

        PAIS_MONEDA PAIS );

miPAIS    PAIS;
miMONEDA  MONEDA;
BEGIN
    /* Sentencias
    */
END;

```

Pueden asignarse todos los campos de un registro utilizando una sentencia SELECT. En este caso hay que tener cuidado en especificar las columnas en el orden conveniente según la declaración de los campos del registro. Para este tipo de asignación es muy frecuente el uso del atributo %ROWTYPE que veremos más adelante.

```

SELECT  CO_PAIS, DESCRIPCION, CONTINENTE
INTO    miPAIS
FROM    PAISES
WHERE   CO_PAIS = 27;

```

Puede asignarse un registro a otro cuando sean del mismo tipo:

```

DECLARE

    TYPE PAIS IS RECORD ...

    miPAIS PAIS;
    otroPAIS PAIS;
BEGIN

    miPAIS.CO_PAIS := 27;
    miPAIS.DESCRIPCION := 'ITALIA';
    miPAIS.CONTINENTE := 'EUROPA';
    otroPAIS := miPAIS;

END;

```

Declaración de registros con el atributo %ROWTYPE

Se puede declarar un registro basándose en una colección de columnas de una tabla, vista o cursor de la base de datos mediante el atributo **%ROWTYPE**.

Por ejemplo, si tengo una tabla PAISES declarada como:

```
CREATE TABLE PAISES (  
  CO_PAIS          NUMBER,  
  DESCRIPCION      VARCHAR2 (50) ,  
  CONTINENTE       VARCHAR2 (20) );
```

Puedo declarar una variable de tipo registro como **PAISES%ROWTYPE**;

```
DECLARE  
  miPAIS PAISES%ROWTYPE;  
BEGIN  
  /* Sentencias ... */  
END;
```

Lo cual significa que el registro miPAIS tendrá la siguiente estructura: CO_PAIS NUMBER, DESCRIPCION VARCHAR2(50), CONTINENTE VARCHAR2(20).

De esta forma se crea el registro de forma dinámico y se podrán asignar valores a los campos de un registro a través de un select sobre la tabla, vista o cursor a partir de la cual se creó el registro.

Tablas PL/SQL

Declaración de tablas de PL/SQL

Las tablas de PL/SQL son tipos de datos que nos permiten almacenar varios valores del mismo tipo de datos.

Una tabla PL/SQL :

- Es similar a un array
- Tiene dos componentes: Un índice de tipo **BINARY_INTEGER** que permite

acceder a los elementos en la tabla PL/SQL y una columna de escalares o registros que contiene los valores de la tabla PL/SQL

- Puede incrementar su tamaño dinámicamente.

La sintaxis general para declarar una tabla de PL es la siguiente:

```
TYPE <nombre_tipo_tabla> IS TABLE OF  
<tipo_datos> [NOT NULL]  
INDEX BY BINARY_INTEGER ;
```

Una vez que hemos definido el tipo, podemos declarar variables y asignarle valores.

```
DECLARE  
  
/* Definimos el tipo PAISES como tabla PL/SQL */  
TYPE PAISES IS TABLE OF NUMBER INDEX BY BINARY_INTEGER ;  
/* Declaramos una variable del tipo PAISES */  
tPAISES PAISES;  
  
BEGIN  
  
tPAISES(1) := 1;  
tPAISES(2) := 2;  
tPAISES(3) := 3;  
  
END;
```

No es posible inicializar las tablas en la inicialización.

El rango de binary integer es -2147483647.. 2147483647, por lo tanto el índice puede ser negativo, lo cual indica que el índice del primer valor no tiene que ser necesariamente el cero.

Tablas PL/SQL de registros

Es posible declarar elementos de una tabla PL/SQL como de tipo registro.

```
DECLARE  
  
TYPE PAIS IS RECORD
```

```

(
    CO_PAIS      NUMBER NOT NULL ,
    DESCRIPCION  VARCHAR2(50),
    CONTINENTE   VARCHAR2(20)
);
TYPE PAISES IS TABLE OF PAIS INDEX BY BINARY_INTEGER ;
tPAISES PAISES;
BEGIN

    tPAISES(1).CO_PAIS := 27;
    tPAISES(1).DESCRIPCION := 'ITALIA';
    tPAISES(1).CONTINENTE := 'EUROPA';

END;

```

Funciones para el manejo de tablas PL/SQL

Cuando trabajamos con tablas de PL podemos utilizar las siguientes funciones:

FIRST. Devuelve el menor índice de la tabla. NULL si está vacía.

LAST. Devuelve el mayor índice de la tabla. NULL si está vacía.

El siguiente ejemplo muestra el uso de FIRST y LAST :

```

DECLARE
    TYPE ARR_CIUDADES IS TABLE OF VARCHAR2(50) INDEX BY
    BINARY_INTEGER;
    misCiudades ARR_CIUDADES;
BEGIN
    misCiudades(1) := 'MADRID';
    misCiudades(2) := 'BILBAO';
    misCiudades(3) := 'MALAGA';

    FOR i IN misCiudades.FIRST..misCiudades.LAST
    LOOP
        dbms_output.put_line(misCiudades(i));
    END LOOP;
END;

```

EXISTS(i). Utilizada para saber si en un cierto índice hay almacenado un valor. Devolverá TRUE si en el índice i hay un valor.

```
DECLARE
TYPE ARR_CIUDADES IS TABLE OF VARCHAR2(50) INDEX BY
BINARY_INTEGER;
misCiudades ARR_CIUDADES;

BEGIN
    misCiudades(1) := 'MADRID';
    misCiudades(3) := 'MALAGA';

    FOR i IN misCiudades.FIRST..misCiudades.LAST
    LOOP
        IF misCiudades.EXISTS(i) THEN
            dbms_output.put_line(misCiudades(i));
        ELSE
            dbms_output.put_line('El elemento no existe:'||TO_CHAR(i));
        END IF;
    END LOOP;
END;
```

COUNT. Devuelve el número de elementos de la tabla PL/SQL.

```
DECLARE
    TYPE ARR_CIUDADES IS TABLE OF VARCHAR2(50) INDEX BY
    BINARY_INTEGER;
    misCiudades ARR_CIUDADES;

BEGIN
    misCiudades(1) := 'MADRID';
    misCiudades(3) := 'MALAGA';
    /* Devuelve 2, ya que solo hay dos elementos con valor */
    dbms_output.put_line(
        'El número de elementos es: '||misCiudades.COUNT);
END;
```

PRIOR (n). Devuelve el número del índice anterior a n en la tabla.

```
DECLARE
    TYPE ARR_CIUDADES IS TABLE OF VARCHAR2(50) INDEX BY
    BINARY_INTEGER;
    misCiudades ARR_CIUDADES;
BEGIN
    misCiudades(1) := 'MADRID';
    misCiudades(3) := 'MALAGA';
    /* Devuelve 1, ya que el elemento 2 no existe */
    dbms_output.put_line(
        'El elemento previo a 3 es:' || misCiudades.PRIOR(3));
END;
```

NEXT (n). Devuelve el número del índice posterior a n en la tabla.

```
DECLARE
    TYPE ARR_CIUDADES IS TABLE OF VARCHAR2(50) INDEX BY
    BINARY_INTEGER;
    misCiudades ARR_CIUDADES;
BEGIN
    misCiudades(1) := 'MADRID';
    misCiudades(3) := 'MALAGA';
    /* Devuelve 3, ya que el elemento 2 no existe */
    dbms_output.put_line(
        'El elemento siguiente es:' || misCiudades.NEXT(1));
END;
```

TRIM. Borra un elemento del final de la tabla PL/SQL.

TRIM(n) borra n elementos del final de la tabla PL/SQL.

DELETE. Borra todos los elementos de la tabla PL/SQL.

DELETE(n) borra el correspondiente al índice n.

DELETE(m,n) borra los elementos entre m y n.

Varrays

Definición de VARRAYS.

Un varray se manipula de forma muy similar a las tablas de PL, pero se implementa de forma diferente. Los elementos en el varray se almacenan comenzando en el índice 1 hasta la longitud máxima declarada en el tipo varray.

La sintaxis general es la siguiente:

```
TYPE <nombre_tipo> IS VARRAY (<tamaño_maximo>) OF  
<tipo_elementos>;
```

Una consideración a tener en cuenta es que en la declaración de un varray el tipo de datos no puede ser de los siguientes tipos de datos:

BOOLEAN

NCHAR

NCLOB

NVARCHAR(n)

REF CURSOR

TABLE

VARRAY

Sin embargo se puede especificar el tipo utilizando los atributos **%TYPE** y **%ROWTYPE**.

Los **VARRAY** deben estar inicializados antes de poder utilizarse. Para inicializar un **VARRAY** se utiliza un constructor (podemos inicializar el VARRAY en la sección DECLARE o bien dentro del cuerpo del bloque):

```
DECLARE  
  
    /* Declaramos el tipo VARRAY de cinco elementos VARCHAR2*/  
    TYPE t_cadena IS VARRAY(5) OF VARCHAR2(50);  
    /* Asignamos los valores con un constructor */  
    v_lista t_cadena:= t_cadena('Aitor', 'Alicia',  
    'Pedro','','');;
```



```

BEGIN
    v_lista(4) := 'Tita';
    v_lista(5) := 'Ainhoa';
END;

```

El tamaño de un VARRAY se establece mediante el número de parámetros utilizados en el constructor, si declaramos un VARRAY de cinco elementos pero al inicializarlo pasamos sólo tres parámetros al constructor, el tamaño del VARRAY será tres. Si se hacen asignaciones a elementos que queden fuera del rango se producirá un error.

El tamaño de un VARRAY podrá aumentarse utilizando la función **EXTEND**, pero nunca con mayor dimensión que la definida en la declaración del tipo. Por ejemplo, la variable `v_lista` que sólo tiene 3 valores definidos por lo que se podría ampliar hasta cinco elementos pero no más allá.

Un VARRAY comparte con las tablas de PL todas las funciones válidas para ellas, pero añade las siguientes:

- **LIMIT** . Devuelve el número máximo de elementos que admite el VARRAY.
- **EXTEND** .Añade un elemento al VARRAY.
- **EXTEND(n)** .Añade (n) elementos al VARRAY.

Varrays en la base de datos

Los VARRAYS pueden almacenarse en las columnas de la base de datos. Sin embargo, un varray sólo puede manipularse en su integridad, no pudiendo modificarse sus elementos individuales de un varray.

Para poder crear tablas con campos de tipo VARRAY debemos crear el VARRAY como un objeto de la base de datos.

La sintaxis general es:

```

CREATE [OR REPLACE]
TYPE <nombre_tipo> IS VARRAY (<tamaño_maximo>) OF
<tipo_elementos>;

```

Una vez que hayamos creado el tipo sobre la base de datos, podremos utilizarlo como un tipo de datos más en la creación de tablas, declaración de variables

Vease el siguiente ejemplo:

```
CREATE OR REPLACE TYPE PACK_PRODUCTOS AS VARRAY(10) OF
VARCHAR2(60);
CREATE TABLE OFERTAS
(
CO_OFERTA NUMBER,
PRODUCTOS PACK_PRODUCTOS,
PRECION NUMBER
);
```

Para modificar un varray almacenado, primero hay que seleccionarlo en una variable PL/SQL. Luego se modifica la variable y se vuelve a almacenar en la tabla.

La utilización de VARRAYS en la base de datos está completamente desaconsejada.

Bulk collect

PL/SQL nos permite leer varios registros en una tabla de PL con un único acceso a través de la instrucción **BULK COLLECT**.

Esto nos permitirá reducir el número de accesos a disco, por lo que optimizaremos el rendimiento de nuestras aplicaciones. Como contrapartida el consumo de memoria será mayor.

```
DECLARE
TYPE t_descripcion IS TABLE OF PAISES.DESCRIPCION%TYPE;
TYPE t_continente IS TABLE OF PAISES.CONTINENTE%TYPE;

v_descripcion t_descripcion;
v_continente t_continente;
```

```

BEGIN
    SELECT DESCRIPCION,
           CONTINENTE
    BULK COLLECT INTO v_descripcion, v_continente
    FROM PAISES;

    FOR i IN v_descripcion.FIRST .. v_descripcion.LAST LOOP
        dbms_output.put_line(v_descripcion(i) || ', ' ||
v_continente(i));
    END LOOP;

END;
/

```

Podemos utilizar **BULK COLLECT** con registros de PL.

```

DECLARE
    TYPE PAIS IS RECORD (CO_PAIS      NUMBER ,
                        DESCRIPCION VARCHAR2(50),
                        CONTINENTE  VARCHAR2(20));
    TYPE t_paises IS TABLE OF PAIS;
    v_paises t_paises;

BEGIN
    SELECT CO_PAIS, DESCRIPCION, CONTINENTE
    BULK COLLECT INTO v_paises
    FROM PAISES;

    FOR i IN v_paises.FIRST .. v_paises.LAST LOOP
        dbms_output.put_line(v_paises(i).DESCRIPCION ||
                                ', ' || v_paises(i).CONTINENTE);
    END LOOP;

END;
/

```

Tambien podemos utilizar el atributo **ROWTYPE**.

```

DECLARE

    TYPE t_paises IS TABLE OF PAISES%ROWTYPE;


```

```

v_paises t_paises;

BEGIN
    SELECT CO_PAIS, DESCRIPCION, CONTINENTE
    BULK COLLECT INTO v_paises
    FROM PAISES;

    FOR i IN v_paises.FIRST .. v_paises.LAST LOOP
        dbms_output.put_line(v_paises(i).DESCRIPCION ||
                               ', ' || v_paises(i).CONTINENTE);
    END LOOP;

END;
/

```

Transacciones autónomas

En ocasiones es necesario que los datos escritos por parte de una transacción sean persistentes a pesar de que la transacción se deshaga con **ROLLBACK**.

PL/SQL permite marcar un bloque con **PRAGMA AUTONOMOUS_TRANSACTION**. Con esta directiva marcamos el subprograma para que se comporte como transacción diferente a la del proceso principal, llevando el control de **COMMIT** o **ROLLBACK** independiente.

Observe el siguiente ejemplo. Primero creamos un procedimiento y lo marcamos con **PRAGMA AUTONOMOUS_TRANSACTION**.

```

CREATE OR REPLACE PROCEDURE Grabar_Log(descripcion VARCHAR2)
IS
    PRAGMA AUTONOMOUS_TRANSACTION;

BEGIN
    INSERT INTO LOG_APLICACION
    (CO_ERROR, DESCRIPCION, FX_ERROR)
    VALUES
    (SQ_ERROR.NEXTVAL, descripcion, SYSDATE);

    COMMIT; -- Este commit solo afecta a la transaccion autonoma
END ;

```

A continuación utilizamos el procedimiento desde un bloque de PL/SQL:

```
DECLARE

    producto PRECIOS%TYPE;

BEGIN

    producto := '100599';

    INSERT INTO PRECIOS

        (CO_PRODUCTO, PRECIO, FX_ALTA)

    VALUES

        (producto, 150, SYSDATE);

    COMMIT;

EXCEPTION

WHEN OTHERS THEN

    Grabar_Log(SQLERRM) ;

    ROLLBACK;

    /* Los datos grabados por "Grabar_Log" se escriben en la
base
de datos a pesar del ROLLBACK, ya que el procedimiento
está
marcado como transacción autonoma.
*/

END;
```

Es muy común que, por ejemplo, en caso de que se produzca algún tipo de error queramos insertar un registro en una tabla de log con el error que se ha producido y hacer ROLLBACK de la transacción. Pero si hacemos ROLLBACK de la transacción también lo hacemos de la inserción del log.

SQL Dinamico

Sentencias DML con SQL dinamico

PL/SQL ofrece la posibilidad de ejecutar sentencias SQL a partir de cadenas de caracteres. Para ello debemos emplear la instrucción **EXECUTE IMMEDIATE**.

Podemos obtener información acerca de número de filas afectadas por la instrucción ejecutada por **EXECUTE IMMEDIATE** utilizando **SQL%ROWCOUNT**.

El siguiente ejemplo muestra la ejecución de un comando SQL dinamico.

```

DECLARE

ret NUMBER;

FUNCTION fn_execute RETURN NUMBER IS

    sql_str VARCHAR2(1000);

BEGIN

    sql_str := 'UPDATE DATOS SET NOMBRE = ''NUEVO NOMBRE''
                WHERE CODIGO = 1';

    EXECUTE IMMEDIATE sql_str;

    RETURN SQL%ROWCOUNT;

END fn_execute ;

BEGIN

    ret := fn_execute();

    dbms_output.put_line(TO_CHAR(ret));

END;

```

Podemos además parametrizar nuestras consultas a través de variables host. Una variable host es una variable que pertenece al programa que está ejecutando la sentencia SQL dinámica y que podemos asignar en el interior de la sentencia SQL con la palabra clave **USING** . Las variables host van precedidas de dos puntos ":".

El siguiente ejemplo muestra el uso de variables host para parametrizar una sentencia SQL dinamica.

```

DECLARE

ret NUMBER;

FUNCTION fn_execute (nombre VARCHAR2, codigo NUMBER) RETURN
NUMBER
IS

    sql_str VARCHAR2(1000);

BEGIN

    sql_str := 'UPDATE DATOS SET NOMBRE = :new_nombre
                WHERE CODIGO = :codigo';

    EXECUTE IMMEDIATE sql_str USING nombre, codigo;

    RETURN SQL%ROWCOUNT;

END fn_execute ;

BEGIN

    ret := fn_execute('Devjoker',1);

    dbms_output.put_line(TO_CHAR(ret));

```

END;

Cursores con SQL dinámico

Con SQL dinámico también podemos utilizar cursores.

Para utilizar un cursor implícito solo debemos construir nuestra sentencia SELECT en una variable de tipo carácter y ejecutarla con EXECUTE IMMEDIATE utilizando la palabra clave INTO.

DECLARE

str_sql **VARCHAR2**(255);

l_cnt **VARCHAR2**(20);

BEGIN

str_sql := 'SELECT count(*) FROM PAISES';

EXECUTE IMMEDIATE str_sql **INTO** l_cnt;

dbms_output.put_line(l_cnt);

END;

Trabajar con cursores explícitos es también muy fácil. Únicamente destacar el uso de **REF CURSOR** para declarar una variable para referirnos al cursor generado con SQL dinámico.

DECLARE

TYPE CUR_TYP **IS REF CURSOR**;

c_cursor **CUR_TYP**;

fila PAISES%**ROWTYPE**;

v_query **VARCHAR2**(255);

BEGIN

v_query := 'SELECT * FROM PAISES';

OPEN c_cursor **FOR** v_query;

LOOP

FETCH c_cursor **INTO** fila;

EXIT WHEN c_cursor%**NOTFOUND**;

dbms_output.put_line(fila.DESCRIPCION);

END LOOP;

CLOSE c_cursor;

END;

Las variables host tambien se pueden utilizar en los cursores.

```
DECLARE

  TYPE cur_typ IS REF CURSOR;
  c_cursor      CUR_TYP;
  fila PAISES%ROWTYPE;
  v_query       VARCHAR2(255);
  codigo_pais   VARCHAR2(3) := 'ESP';

BEGIN

  v_query := 'SELECT * FROM PAISES WHERE CO_PAIS = :cpais';
  OPEN c_cursor FOR v_query USING codigo_pais;
  LOOP
    FETCH c_cursor INTO fila;
    EXIT WHEN c_cursor%NOTFOUND;
    dbms_output.put_line(fila.DESCRIPCION);
  END LOOP;
  CLOSE c_cursor;
END;
```

PL/SQL y Java

Otra de la virtudes de PL/SQL es que permite trabajar conjuntamente con Java.

PL/SQL es un excelente lenguaje para la gestion de información pero en ocasiones, podemos necesitar de un lenguaje de programación más potente. Por ejemplo podríamos necesitar consumir un servicio Web, conectar a otro servidor, trabajar con Sockets Para estos casos podemos trabajar conjuntamente con PL/SQL y Java.

Para poder trabajar con Java y PL/SQL debemos realizar los siguientes pasos:

Crear el programa Java y cargarlo en la base de datos.

Crear un program de recubrimiento (Wrapper) de PL/SQL.

Creacion de Objetos Java en la base de datos ORACLE.

ORACLE incorpora su propia versión de la máquina virtual Java y del JRE. Esta versión

de Java se instala conjuntamente con **ORACLE**.

Para crear objetos Java en la base de datos podemos utilizar la utilidad LoadJava de **ORACLE** desde línea de comandos o bien crear objetos **JAVA SOURCE** en la propia base de datos.

La sintaxis para la creación de **JAVA SOURCE** en **ORACLE** es la siguiente.

```
CREATE [OR REPLACE] AND COMPILE
JAVA SOURCE
NAMED <JavaSourceName>
AS
public class <className>
{
    <java code>
    ...
};
```

El siguiente ejemplo crea y compila una clase Java OracleJavaClass en el interior de **JAVA SOURCE** FuentesJava. **Un aspecto muy a tener en cuenta es que los métodos de la clase java que queramos invocar desde PL/SQL deben ser estaticos.**

```
CREATE OR REPLACE AND COMPILE
JAVA SOURCE
NAMED FuentesJava
AS
public class OracleJavaClass
{
    public static String Saluda(String nombre)
    {
        return ("Hola desde Java" + nombre);
    }
}
;
```

Un mismo **JAVA SOURCE** puede contener varias clases de Java.

```

CREATE OR REPLACE AND COMPILE
JAVA SOURCE
NAMED FuentesJava
AS
public class OracleJavaClass
{
    public static String Saluda(String nombre)
    {
        return ("Hola desde Java" + nombre);
    }
}

public class OracleJavaMejorada
{
    public static String SaludoMejorado(String nombre)
    {
        return ("Saludo mejorado desde Java para:" + nombre);
    }
}
;

```

La otra opción sería guardar nuestro código Java en el archivo OracleJavaClass.java, compilarlo y cargarlo en **ORACLE** con LoadJava.

A continuación se muestran ejemplos del uso de la utilidad LoadJava

-help

-u usuario/password -v -f -r OracleJavaClass.class

-u usuario/password -v -f -r OracleJavaClass.java

Ejecución de programas Java con PL/SQL

Una vez que tenemos listo el programa de Java debemos integrarlo con PL/SQL. Esto se realiza a través de subprogramas de recubrimiento llamados Wrappers.

No podemos crear un Wrapper en un bloque anónimo.

La sintaxis general es la siguiente:

```

CREATE [OR REPLACE]

```

```

FUNCTION|PROCEDURE <name> [( <params>,...)]
[RETURN <tipo>]
IS|AS
LANGUAGE JAVA NAME
'<clase>.<metodo> [return <tipo>]' ;

```

El siguiente ejemplo muestra el Wrapper para nuestra función Saludo.

```

CREATE OR REPLACE
FUNCTION Saluda_wrap (nombre VARCHAR2)
RETURN VARCHAR2
AS
LANGUAGE JAVA NAME
'OracleJavaClass.Saluda(java.lang.String) return
java.lang.String';

```

Una vez creado el wrapper, podremos ejecutarlo como cualquier otra funcion o procedure de PL/SQL. Debemos crear un wrapper por cada función java que queramos ejecutar desde PL/SQL.

Cuando ejecutemos el wrapper, es decir, la función "Saluda_wrap", internamente se ejecutará la clase java y se invocará el método estático "OracleJavaClass.Saluda".

Un aspecto a tener en cuenta es que es necesario proporcionar el nombre del tipo java completo, es decir, debemos especificar java.lang.String en lugar de únicamente String.

```

SELECT SALUDA_WRAP ('DEVJOKER')
FROM DUAL;

```

La ejecución de este ejemplo en SQL*Plus genera la siguiente salida:

```

SQL> SELECT SALUDA_WRAP ('DEVJOKER') FROM DUAL;

SALUDA_WRAP ('DEVJOKER')
-----
Hola desde JavaDEVJOKER

```

Una recomendación de diseño sería agrupar todos los Wrapper en un mismo paquete. En el caso de que nuestro programa Java necesitase de packages Java adicionales, deberíamos cargarlos en la base de datos con la utilidad LoadJava.

Buenas prácticas trabajando con PL-SQL

Porque necesitamos Buenas Practicas en el desarrollo del Back End de las aplicaciones?

Simplemente, por las mismas razones que se necesitan en el desarrollo de cualquier tipo de software: Mantenición, Legibilidad, Rehusó y Modificabilidad del código. Pero, los programadores de PL/SQL, debemos estar especialmente atentos en este sentido, por las siguientes dos razones:

Con el potencial que nos ofrece PL/SQL es importante estar atentos al abanico de posibilidades del que disponemos. No sucede lo mismo en otros lenguajes PL, ya que estos se encuentran mucho mas limitados en sus alternativas de expresión.

Es sumamente fácil, tal vez demasiado a juicio de los gurúes, escribir SQL en PL/SQL, por cuanto no necesitamos de una interfaz ODBC/JDBC etc. Esto, sumado a la creencia de que el modelo de datos esta libre de la evolución propia del software, genera código difícil de mantener y evolucionar.

EL TESTING DE UNIDAD

Algunas consideraciones al respecto:

LOS DEVELOPERS GASTAN EL 80% DEL TIEMPO DE DESARROLLO EN IDENTIFICAR Y CORREGIR DEFECTOS!

Si bien existen muchas clases de test, solamente existe un responsable para los Test de Unidad: el Programador!

Nuestro código no se va a testear solo, por lo que debemos aceptar la responsabilidad y ser disciplinados!!

Seis pasos hacia un testing de unidad exitoso:

1. Describir toda la funcionalidad requerida del programa
 2. Definir el Header del programas: Nombre, Lista de Parámetros, Valor de retorno.
 3. Elaborar los casos de test para el programa
 4. Construir el programa que testee nuestro programa
 5. Programar la unidad de programa
 6. Testear, Debuguear y Corregir. Testear, Debuguear y Corregir...
- Repetir del 3-6 generando un Bug Report.

No debemos preocuparnos por comprar o desarrollar código para hacer nuestros test de unidad, podemos descargar gratuitamente del sitio oficial de Oracle un sencillo frame PL/SQL a tales efectos.

Es interesante mencionar que los casos de uso deben escribirse ANTES que la unidad de programa. Si escribimos las pruebas después de la unidad, vamos a inclinar la balanza hacia la búsqueda de aciertos y no de los fallos, existe de hecho, una tendencia psicológica a seleccionar casos de test que el desarrollador sepa tratados en su unidad.

Ahora, que ocurre cuando nuestra aplicación en producción necesita una modificación leve? Podemos hacerla y comprobar con nuestro test de unidad si la unidad es correcta, pero existe un beneficio mayor: Como sabemos que la aplicación en conjunto sigue funcionando correctamente? Los test de Unidad almacenados nos permitirán realizar lo que se denomina **Test de Regresión**. Se

vuelven a disparar los test de unidad y de esta manera nos aseguramos que el cambio en la funcionalidad de las partes no afectó el funcionamiento del todo.

Aumentando el rendimiento con FORALL y BULK COLLECT

FORALL: Resulta sumamente útil en operaciones DML, por cuanto la sentencia FORALL nos permite tomar todos los datos de una colección y llevarlos a la base de datos de una sola vez.

Cuando se ejecuta un UPDATE, se parsea el bloque en el motor de pl/sql y después en el motor SQL una vez por cada fila updeteadada, estos N cambios de contexto representan mucho overhead. En cambio, en el FOR ALL UPDATE se produce solo un cambio de contexto en cada parser.

Ejemplo:

Esta sentencia por estar dentro de un bloque PL/SQL requiere que la interprete el motor PL/SQL y después el de SLQ POR CADA fila.

```
FOR rec IN emp_cur LOOP

    UPDATE employee

    SET ...

    WHERE ...

LOOP;
```

Se parsea solo una vez, pues se hace un cambio de contexto entre el motor PL y el moto SQL, solamente.

ORALL indice IN coleccion.FIRST .. coleccion.LAST

```
UPDATE employee  
  
SET ...  
  
WHERE ...
```

Observaciones sobre el uso de FORALL:

El motor lee el contenido de la colección de manera secuencial, por lo que si encuentra una posición sin un valor definido disparará la excepción ORA-22160. Si no queremos este problema, y disponemos de Oracle 10G → podemos agregar la sentencia INDICES OF en la sentencia FORALL. Por ejemplo:

```
FORALL i IN INDICES OF miColección  
  
INSERT INTO ...
```

Solo se permiten DML simples para FORALL.

SQL%BULK_ROWCOUNT retorna el número de filas procesadas para la consulta.

Antes de Oracle 10G, las colecciones debían llenarse secuencialmente.

BULK COLLECT INTO:

Como es sabido, cuando queremos recuperar el valor de una fila utilizamos la sentencia

INTO, pero esto no es posible cuando el resultado de la consulta retorna mas de una fila. Existe a tales efectos, dos soluciones. Una, es el uso de cursores. La otra, es la sentencia BULK COLLECT INTO que podrá recuperar N filas y colocarlas en una estructura adecuada, siendo esta una colección de lo que sea que necesitemos, Rowtypes, Objects, etc.

También podemos hacer abrir una variable de cursor sobre una estructura, usando la sentencia BULK COLLECT INTO.

Un caso en que podemos usar BULK COLLECT INTO

Si necesitamos llevar datos masivamente a la base, entonces necesitaremos hacer un FORALL pero esta sentencia solo funciona con colecciones, no con cursores, por lo que para usarla necesitamos haber hecho preferiblemente un BULK COLLECT INTO sobre alguna colección o array.

Resumiendo, siempre que necesitemos realizar una DML por cada fila de un cursor, o una DML dentro de un LOOP, es mejor utilizar un BULK COLLECT INTO mas allá de si lo hacemos sobre una variable de cursor o directamente con una sentencia SQL.

Cuando usar esta sentencia? Cuando se necesite ejecutar SQL dentro de un LOOP, preferentemente si el número de filas es mayor a 5.

declaramos una tabla de algún tipo de dato, preferentemente un ROWTYPE:
MiTabla

```
SELECT * BULK COLLECT INTO MiTabla
```

Iteramos sobre la tabla.

PAGINACIÓN DE SENTENCIAS DML

Declarar un cursor y abrirlo

En ves de hacer FETCH rec, hacemos FETCH rec BULK COLLECT INTO MiTabla [LIMIT cantFilasQueQuieraTraer].

Procesamos la tabla, y seguimos iterando pero nos traemos de a N filas, en vez de a una.

Los BULK COLLECT pueden ser usados con cursores Implícitos o Explícitos. Las colecciones siempre se rellenan secuencialmente, comenzando de 1.

SOBRE LOS ABUSOS DE SQL

Pl/sql nos permite interactuar con el motor SQL sin otra capa intermedia, como ODBC/JDBC etc. De esta manera, facilita tal vez demasiado el uso de SQL.

Pero, la capa PL/SQL debería permitir una fácil manipulación del modelo de datos, y no constituir un obstáculo para la correcta evolución del mismo. Si todos escriben SQL donde y cuando quieren, entonces el código se torna inmanejable.

Recordemos hardcodear nuestro código, es malo. Por otro lado, es una verdad innegable que TODA SENTENCIA SQL ES HARD-CODING! Por lo que podemos afirmar por transitividad y sin pérdida de generalidad, que SQL es MALO! Que mas sabemos sobre SQL?

La mayoría de los problemas de performance suelen estar en las sentencias SQL.

Provocan errores en tiempo de ejecución

Las sentencias SQL estan sujetas a las reglas del negocio, y las reglas del negocio cambian por lo que nuestras sentencias SQL cambian.

El modelo de datos, es una de los modelos mas volátiles de cualquier aplicación; las sentencias SQL cablean esas estructuras! Sería conveniente que regáramos por toda la aplicación las mismas sentencias SQL cableadas que probablemente mañana tengamos que cambiar? Claro que no lo sería, si hacemos tal cosa, deberemos debuguear, optimizar y mantener las mismas líneas de código donde quiera que aparezcan.

Si abstraemos el modelo de datos en estructuras que lo encapsulen será mucho mas fácil realizar las modificaciones del modelo de datos. Además, se puede hacer un manejo consistente de las excepciones, pues de otra manera, pueden hacerse N manejos de una excepción en distintos lugares.

Para abstraer consultas que retornen una sola fila:

Es importante la utilización de generadores de código para las sentencias simples, como la recuperación de una fila por su PK o por algún índice. Se sugiere la abstracción de una tabla mediante métodos de un paquete que retornen el valor de una columna en particular. También puede ser útil escribir un paquete que se inicialice con la PK de una instancia de la entidad, por ejemplo si tenemos la tabla Empleados, podemos desarrollar un paquete que nos permita acceder cada uno de las columnas mediante una función retornarColumna(PK),

por ejemplo. Facilitará esta abstracción, el uso de un TYPE declarado público en el paquete que abstrae la entidad del modelo que estamos implementando.

Para abstraer consultas que retornen mas de una fila:

Tenemos dos opciones:

1. Encapsular retornando una colección: Podemos hacer un BULK COLLECT en una colección de tipos ROWTYPE y retornar la colección.
2. Variable Cursor: Esto último es sumamente útil para retornar datos en entornos no PL/SQL. Una variable cursor es un ptero que apunta a un result set. Podemos pasar variables cursor de una unidad de programa a otra, inclusive si no son programas PL/SQL.

Pensar en SQL como un SERVICIO provisto por el gestor relacional, y no como líneas de código, puede contribuir a que no rescribamos una y otra vez sentencias cableadas. Repetir SQL es repetir lógica asociada al negocio. Mejor solicitar el servicio cuando se necesite y no redefinirlo cada vez. Esto se logra con una apropiada encapsulación de los objetos de base de datos.

PROGRAMAR PARA EL CAMBIO

Cambian las estructuras de datos ➔ Los programas se marcan como inválidos
➔ Hay que re compilar el código inválido

Por esto es sumamente importante no cablear la definición de una variable.

Usar %TYPE y %ROWTYPE.

Hacer los FETCH de un cursor en un registro de tipo cursor y no en variables!

Asegurarnos de que ninguna variable entrará en conflicto con un objeto existente de la base de datos. Pero como podemos adelantarnos a un futuro

renombramiento de algún objeto de la base? Las convenciones de nomenclatura no nos ofrecen garantías → ***SIEMPRE CUALIFICAR LAS REFERENCIAS A VARIABLES DENTRO DE UNA SENTENCIA SQL!!***

Ejemplo:

Anteponemos el nombre del Procedure a la variable para no tener conflictos.

También podemos arriesgarnos a que las reglas de nomenclatura nos ayuden, esto por supuesto, si no existe la posibilidad de que una columna se renombre con nuestra nomenclación...

MANEJAR LOS ERRORES EFECTIVA Y CONCISTENTEMENTE

Las excepciones pueden ser disparadas y manejadas, pero no pasadas como argumentos a otros programas.

Para comunicarle un mensaje de error a la aplicación que usa nuestro paquete, tenemos que usar `RAISE_APPLICATION_ERROR (NUMERO, MENSAJE)`. Es importante definir de manera centralizada los números de error y los mensajes correspondientes para hacer un manejo consistente.

UN PAR DE CONCEPTOS AVANZADOS SOBRE PACKAGES

La participación de los packages en el diseño de alto nivel:

La fortaleza de los packages radica en la posibilidad que brindan implementar abstracción para nuestras aplicaciones. Nos permiten ocultar la complejidad y

los detalles de una funcionalidad determinada. Por esta razón, es el proceso de diseño debe haber una instancia para generar una visión arquitectural de la definición e interrelación de los packages. Una manera efectiva de lograr esto, es asociar un package a cada necesidad funcional compleja de nuestra aplicación.

No es desatinado intentar una relación entre un diagrama de clases , uno de Entidad / Relación y el diagrama de packages. De hecho, un paquete intenta implementar las principales cualidades de una clase al definir una interfase pública y una implementación privada.

Sobre las interdependencias:

Un diagrama nos permitirá establecer dependencias y relaciones entre los mismos. Nunca deberíamos permitir que el package A tenga una dependencia con el package B si este último tiene una dependencia con el package A. Si no es posible evitar esta interdependencia tenemos que recordar que compilando primero la especificación de ambos paquetes podemos compilar la interdependencia existente. Es importante recordar esto a la hora de escribir un script de instalación. De esto surge como una buena practica mantener en archivos separados la especificación de un package de su body. ***Si ponemos todo junto quizá habrá problemas en los scripts de instalación.***

Por supuesto, no todas las interdependencias son viables. Si la especificación del package A referencia a un elemento del package B, y la especificación del package B referencia un elemento del package A ➔ Sencillamente será imposible compilar estos paquetes, por lo que dicha situación tiene que ser claramente evitada.

Buenas Prácticas:

Un principio fundamental es considerar al resto de developers como Usuarios. Este detalle psicológico tendrá un impacto sustancial en la calidad de nuestras interfaces. Todos los programadores somos concientes de la opinión generalizada que tenemos de los usuarios de nuestras aplicaciones. Para lograr el

confort en el uso de nuestras aplicaciones, debemos tener en mente algunas consideraciones.

Hacer nuestras aplicaciones CASE INSENSITIVE

Siempre que recibamos parámetros de tipo texto, necesitamos asegurarnos de que la manera en que los recibamos no afecte la conducta de nuestra aplicación, de lo contrario veremos muy seguido errores como por ejemplo el ORA-06503: Function returned without value. Simplemente con un UPPER(Parámetro) podemos evitar la incomodidad de recordar como tienen que ser parametrizada nuestra aplicación.

Evitar la parametrización con literales de nuestra unidad.

El usuario estará mucho mas feliz si no tiene que recordar los literales que guían la conducta de nuestra aplicación. Como logramos esto? Tenemos dos opciones.

Una es proveer programas separados para realizar distintas acciones. Esto es viable solamente si la cantidad de acciones a realizar es fija y estemos seguros de que continuará así...

Proveer constantes basadas en packages, preferiblemente el mismo donde tenemos la unidad. Esta es sin duda la mejor opción. Es cierto que de todas formas el usuario tendrá que recordar el nombre del parámetro por mas que esté en un package, pero si algo sale mal se enterará en tiempo de Compilación lo cual es mucho mas ventajoso que enterarse en tiempo de ejecución, teniendo que depurar el código buscando ese misterioso error no detectado causado por escribir con mayúsculas alguna letra del literal que parametriza la unidad.

Flexibilizar nuestros packages mediante Toggles.

La aproximación mas general para flexibilizar una unidad de programa es hacerla altamente parametrizada. El problema es que este enfoque aumenta considerablemente el coupling de la unidad y complejiza la interface.

Numerosos estudios comprueban que el cerebro humano no esta bien equipado para tratar con mas de 7 ítems a la vez... Por lo que tenemos que cuidarnos de sobrecargar en exceso las firmas de nuestras unidades de programa. Además,

que pasaría si quisiéramos afectar TODA la conducta de un package y no solo la de una unidad?

Una mejor aproximación es el uso de Toggles o suichs. Un toggle es un set de tres programas: Dos procedimientos para prender o apagar y una función para retornar el valor del toggle.

Sacar provecho del OverLoad de nombre

Las ventajas de sobrecargar nombres de unidades son claras. Pensemos en la función `To_Char()`. Tenemos que llamar a funciones distintas si el argumento es de tipo integer o de tipo real? No, gracias a su OverLoad. Cuando sobrecargamos el nombre de una unidad, estamos ofreciendo muchas versiones de una misma unidad, simplificando la interface de la unidad, lo que resulta crítico. En esto de ver a los developers como usuarios de nuestros desarrollos, notamos la importancia de programar unidades intuitivas y versátiles.

Sucede a veces, que nos encontramos con 3 o 4 unidades de programas que básicamente realizan la misma tarea a nivel semántico pero que devuelven distintos tipos de datos: Como transformar esas N unidades de diferentes nombres en N unidades de igual nombre sobrecargadas? Un primer intento pudiera ser generar solo una función y pasarle un parámetro enumerado. Una mejor aproximación sería generar las N unidades con igual nombre, proveyéndoles a cada una un parámetro `VALOR_DEFAULT` para el retorno, de manera que los diferentes tipos de datos permitirán evadir la ambigüedad de la firma de las unidades sobrecargadas.

El verdadero desafío es desarrollar una sensibilidad que nos permita hacer del overloading nuestra primera solución intuitiva frente a los problemas donde la facilidad de uso sea un objetivo.

Modularizando para desarrollar Packages mantenibles

Tenemos que resistir la tentación de cortar/pegar código creyendo que así estamos construyendo aplicaciones rápidamente, porque lo que realmente estamos haciendo es cavar una fosa muy profunda donde enterraremos las características evolutivas de nuestra aplicación. Tenemos entonces, una regla:

Nunca repitamos una línea de código. En cambio, hemos de construir una unidad privada y llamarla 2 o N veces.

Vale aclarar, que Overloading y modularidad deben considerarse dos caras de la misma moneda si queremos implementar el package correctamente.

El modelo del Diamante:

Debemos estar atentos para que las N versiones de nuestra unidad sobrecargada no este repitiendo líneas de código. Debemos factorizar esas repeticiones y de esa manera lograr la regla mencionada. Normalmente, se llama a este diseño Diamante, si vemos que en la capa superior tenemos una vista de usuario (Que no se corresponde exactamente a una unidad de programa, sino a las N unidades sobrecargadas). En un nivel medio tenemos N versiones de la unidad, y en la capa inferior tenemos una unidad que factoriza lo común de las anteriores (que normalmente contendrá un IF desarrollado, a esto se le llama Efecto de Lámpara de Lava, porque si abrimos los if da la sensación de modelar una de estas lámparas). **Es altamente recomendable construir nuestro package con este criterio.**

Ocultando datos en el Package

Llamamos Packages Data a cualquier estructura de datos implementada en la especificación (los públicos) o el cuerpo (privados) de un package.

Manejar datos públicos es ciertamente conveniente para algunos casos. Pero puede presentar problemas:

Perdida de Control: Cualquier programador puede cambiar los valores de la estructura por ser ésta pública. Es posible que existan reglas de negocios que afecten el valor de esta variable, pero no podremos hacer nada para proteger esa regla de negocios pues nuestra variable es pública, y cualquiera puede modificarla.

Perdida de Flexibilidad: Al ser conocida su estructura, todos hacen suposiciones sobre la misma. La posible necesidad de hacer un cambio en la estructura generará muchos problemas en los usuarios.

Se recomienda **nunca definir variables en la especificación de un package** si es que tenemos otra alternativa. Lo mejor es hacer públicos los datos pero no las

estructuras que lo contengan. Como se logra esto? Declarando las variables en el cuerpo del package y definiendo métodos Get y Set para manipular sus valores.

Una muy interesante consecuencia de esta buena práctica, es la posibilidad de conocer Que y Cuando se esta modificando el valor de un dato. Como es esto posible? Como solo existe un punto por el que se puede afectar el valor de una variable privada, es decir su método Set, podemos en el mismo realizar una traza. Antes de modificar su valor, mostramos en pantalla o en un archivo el nuevo valor que tomará. Esta aproximación puede ser muy poderosa si mantenemos en un package especialmente dedicado a las trazas, manteniendo el nombre del programa que esta realizando el cambio y suministrando un método al que podamos llamar desde los Sets para realizar la traza. A este package podemos añadirle un Toggle o un Switch para activar a desactivar la traza a nuestro gusto. De esta manera, no pasaremos horas debugueando nuestra aplicación para saber donde y que provocó el cambio del valor de una variable.

Otra ventaja, es cuando cambie la estructura de la variable, no habrá que recompilar todos los packages que la utilicen, cosa que sí ocurre cuando declaramos la variable en la especificación del package.

Al mover los datos al cuerpo del paquete, estaremos simplificando la interface de los métodos allí contenidos, pues estos datos serán ahora globales al package, de modo que probablemente nos ahorremos un parámetro en todas aquellas unidades de código que lo necesiten. *Es cierto que esto hace nuestros packages dependiente de los datos, pero como contrapartida solidifica la idea de que los packages son mas un entorno que una colección de unidades.*

Sobre cuando y donde empaquetar funcionalidad:

Cuando hay que empaquetar funcionalidad? Casi siempre.

Cuando no hay que empaquetar funcionalidad? Las siguientes condiciones dadas simultáneamente pueden modelar una sugerencia:

La unidad de programa es muy usada desde muchos lugares

La unidad de programa, nunca cambia → no necesita ser recompilada

El resto de las unidades de programa dentro del package suele cambiar y en consecuencia el package entero necesita ser recompilado.

En este caso, sería importante separar esa unidad de programa del package, porque si no lo hacemos, cada vez que se modifique el package en cuestión se marcará como inválidos todos los programas que lo usen, aunque estos no hallan cambiado.

NOTA: Sobre lo anterior, es importante saber que si recompilamos el BODY de un package, no habrá necesidad de recompilar todos los programas que referencien el package, pero si recompilamos la especificación del mismo, entonces sí habrá que recompilar todas los objetos de la base que lo referencien.

Entonces, como regla: **Es conveniente separar aquella funcionalidad que rara vez cambia, de aquella que suele hacerlo.** Si no lo hacemos, habrá que compilar el package, acción que invalidará todos los programas que lo usen, y estos también necesitaran ser recompilados.

Sobre la inicialización de packages:

Supongamos una sección de inicialización de un package que tiene también una sección de manejo de excepciones:

```
CREATE OR REPLACE PACKAGE valerr  
  
IS  
  
FUNCTION private_variable RETURN VARCHAR2;  
  
END valerr;  
  
/
```

```

CREATE OR REPLACE PACKAGE BODY valerr
IS
    g_private VARCHAR2(1) := 'ABCDE'; --PROBOCARÁ EL DISPARO DE UNA
    EXCEPCIÓN!!!

    FUNCTION private_variable RETURN VARCHAR2
    IS
    BEGIN
        RETURN g_private;
    END private_variable;

    BEGIN
        DBMS_OUTPUT.PUT_LINE ('Before I show you v...');
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE ('Trapped the error!');
    END valerr;
/

```

Que ocurrirá cuando ejecutemos el programa?

*

```

ERROR at line 1:
ORA-06502: PL/SQL: numeric or
value error: character string
buffer too small

```

Pero no había una sección de cacheo de errores? Por otro lado, para hacerlo mas confuso, si corremos este programa una vez mas, este no dará error! Porque sucede esto?

La sección EXCEPTION de la inicialización de un package, solo captura errores que se produzcan en la sección ejecutable del bloque de inicialización.

Cuando la inicialización de un package falla, Oracle marca el paquete como inicializado. Y como todos los paquetes se inicializan solo una vez por sesión, este no vuelve a inicializarse durante la vida de la sesión que lo inicializó.

Resumiendo el enfoque orientado a la Performance

1. Mirar las áreas donde pueda ser posible la aplicación de FOR ALL & BULK COLLECT.
2. Buscar todas las sentencias DML. Si estas están dentro de algún tipo de LOOP, externalizarlas en un Procedure de ámbito Local en la sección de declaración, buscando aumentar el uso del procesamiento BULK.

10 Mitos que los programadores de PL/SQL deberíamos conocer!

Nunca pedir ayuda.

Grave error, sin importar nuestro nivel de Seniority siempre debemos hacerlo. La siguiente sugerencia puede producir un equilibrio práctico: Si no podemos encontrar el problema después de 30 minutos... Pidamos ayuda!

La cantidad de cafeína consumida incide en forma directamente proporcional a nuestras facultades de concentración y productividad.

Siempre es mejor beber agua para hidratar nuestro cuerpo que abusar de la cafeína...

Nunca compartir nuestro conocimiento! Esto nos da poder y seguridad.

De hecho, desde Oracle.Com podemos disponer de muchos recursos tales como Papers, Código, Sugerencias, etc de manera libre y gratuita, algunas de las presentes sugerencias están tomadas de allí.

Nunca dudar de los gurus.

Siempre debemos dejar que nuestra realidad termine de aceptar los consejos que recibimos, ellos también se equivocan.

Nunca ocultar los detalles de nuestro código

Pero ocultar no es sinónimo de algo negativo en el contexto de la programación? No si refiere a detalles de la implementación. Siempre debemos ocultar los detalles, solo así podremos programar para el cambio.

Nunca permitamos que alguien mas lea y observe nuestro código!

La revisión de código constituye un arma sumamente útil contra los bugs y problemas de diseño a bajo nivel.

Nunca te preocupes por el mañana, codifiquemos para el hoy!

Es vital comprender la naturaleza dinámica y cambiante de las reglas del negocio representadas desde el modelo de datos hacia el front end.

Nunca corrigas bugs que otros programadores no hallan encontrado...

Falso. El Bug de hoy, es el bug de mañana. Por supuesto que un repositorio de código soportado por un controlador de versiones nos dará mas valentía para tal empresa.

Nunca asumamos que un Bug de Oracle, fue corregido.

Oracle sugiere confiar en la disponibilidad de la plataforma y la evolución positiva de las misma.

Nunca digas Nunca!

En el contexto del desarrollo de software, podemos asegurar que ese enunciado es falso. Sirva de ejemplo, las siguientes verdades universales en el mundo del desarrollo:

Las cosas nunca permanecerán iguales... o a lo que es equivalente

Las cosas siempre cambiarán...

Unas palabras sobre Refactorizar

Al terminar de desarrollar una unidad, no esta de mas dedicar unos minutos al análisis detenido de nuestro código buscando secciones que requieran una refactorización. La belleza esta en los detalles...

Podemos definir Refactorización, como el proceso de transformación del código escrito mediante el cual sin modificar la conducta externa y esperada, se mejora la estructura y la escritura del código. Es decir, nos abstraemos del Que, y nos concentramos en el Como.

Dos objetivos son sumamente relevantes en el proceso de refactorización:

Simplificar DONDE las cosas son realizadas. (Reorganizar las estructuras del código, moviendo este de un lugar a otro a efectos de aumentar su legibilidad, modificabilidad y rehusos)

Simplificar COMO las cosas son realizadas (Haciendo el código mas elegante, mas compacto y fácil de seguir).

Pautas para refactorizar nuestros desarrollos:

Si en una unidad se llama con mucha frecuencia a la funcionalidad de un package, debemos preguntarnos si no sería mas apropiado que la unidad este contenida en dicho paquete.

Prestar atención a los puntos donde se utilice lógica booleana compleja. Es muy probable que mediante las reglas matemáticas de la lógica Booleana

normalmente conocida por los profesionales, una gran expresión sea equivalente a otra mucho mas simple. Por ejemplo supongamos una función que retorna lo siguiente

```
RETURN ((V_booleana1 And V_booleana2) OR (NOT V_booleana1 And NOT V_booleana2))
```

El sentido común así como la lógica booleana nos dicen que esa expresión es equivalente a la siguiente:

```
RETURN (V_booleana1 = V_booleana2)
```

Observar aquellos lugares donde el código se extiende demasiado. Esta claro que existen problemas que requieren la escritura de miles de líneas, no es el volumen de las líneas lo que esta en discusión, sino la estructura que se le da a las mismas. Es muy probable que podamos ocultar detalles del mismo en Procedure o Function anidadas y de esta manera simplificar las cosas. **Como referencia, debemos pensar que desde el BEGIN al END nunca deberíamos tener mas de 50 líneas**, es mas, un promedio de 20 a 30 líneas es ampliamente recomendado. Si esto se hace con un criterio Top Down, seguramente será mucho mas sencillo de seguir que un monolítico bloque ejecutable, o muchas unidades encadenadas sin criterio arquitectural.

Cuidar el acoplamiento de las unidades. Si tenemos unidades de programa exageradamente parametrizadas, además de restarle claridad al código, es muy probable que tengamos un problema de coupling en nuestro diseño.

Cuidar de mantener simples las estructuras condicionales:

Las expresiones son a menudo mas legibles y entendibles cuando se escriben en forma positiva, es decir el abuso del operador NOT puede restar claridad a nuestro código.

Mantener el nivel de anidamientos bajo. Preferiblemente una estructura If anidada no debería tener mas de dos o tres niveles. Si es necesario ocultar mas niveles de anidamiento, deberíamos usar Procedimientos Locales Anidados.

Cuando una vez no es suficiente:.. *Los loops o interacciones, se cuentan entre los elementos mas sensibles a la performance y los errores.* Por supuesto, Oracle provee varias estructuras para implementar iteraciones, y el uso de cada una dependerá de las circunstancias y no es materia de discusión de las presentes líneas. Ahora bien, existe una guía que bien puede ser tomada como una máxima a la hora de diseñar nuestras estructuras condicionales: **Una forma de entrar, Una forma de salir.** Esto significa que solo existe una manera de entrar a una iteración o de comenzar ese Loop, y que existe un solo punto donde la iteración termina. Una estructura Loop que tenga tres RETURN, no estaría respetando este principio.

Optimización de SQL-PL-SQL Trazas – tkprof

Cómo obtener el plan de ejecución de una sentencia SQL o PL/SQL

Una de las formas más usuales de mejorar el rendimiento de una sentencia SQL o PL/SQL es analizar el plan de ejecución que devuelve el optimizador Oracle. En SQL*Plus se puede obtener dicho plan de ejecución, además de algunas estadísticas referentes al resultado de la ejecución de la sentencia SQL o PLSQL, utilizando el comando *AUTOTRACE*. Para obtener el plan de ejecución no hay necesidad de ejecutar dicho comando pero, ciertamente, si no lo utilizamos, la poca *amigabilidad* del comando que debemos ejecutar (*EXPLAIN PLAN*), el formato de dicho comando y lo complejo que resulta analizar el contenido de la tabla V\$SQL_PLAN, hacen que, por mi parte, recomiende encarecidamente el uso del comando SQL*Plus *AUTOTRACE*.

En mi opinión, *AUTOTRACE* es una buenísima herramienta de diagnóstico y una excelente ayuda para optimizar sentencias SQL y PL/SQL. El comando *AUTOTRACE* es puramente declarativo, por lo que es mucho más fácil de utilizar que el comando *EXPLAIN PLAN*. La sintaxis del comando

AUTOTRACE es como sigue:

SET AUTOTRACE OFF - Deshabilita el análisis (trazado) de las sentencias SQL.

SET AUTOTRACE ON - Habilita el análisis (trazado) de las sentencias SQL.

SET AUTOTRACE TRACEONLY - Habilita el análisis (trazado) de las sentencias SQL pero no devuelve la salida de dicha sentencia. Su uso es recomendable si sólo estamos analizando el rendimiento de la sentencia y no nos interesa conocer los registros que pueda devolver.

SET AUTOTRACE ON/TRACEONLY EXPLAIN - Muestra el plan de ejecución de la sentencia pero no muestra las estadísticas.

SET AUTOTRACE ON STATISTICS - Muestra las estadísticas pero no muestra el plan de ejecución de la sentencia.

Nota: Si se omiten las opciones *EXPLAIN* y *STATISTICS*, entonces al ejecutar una sentencia SQL se mostrarán tanto el plan de ejecución como las estadísticas.

Para poder utilizar la opción *EXPLAIN* del comando *AUTOTRACE*, es necesario crear la tabla *PLAN_TABLE* en el esquema del usuario, es por eso que este comando sólo puede ser ejecutado por determinados usuarios, aquellos para los que la mencionada tabla ya ha sido creada. Es importante pues, conocer los usuarios Oracle que han sido configurados para poder ejecutar el comando *AUTOTRACE*.

Por otro lado, para acceder a las estadísticas, hay que tener acceso a varias tablas del sistema en las que se almacenan los datos del rendimiento de las sentencias SQL. Los DBA pueden dar este acceso utilizando el *script plustrce.sql*. El nombre de este *script* puede variar dependiendo del sistema operativo. El DBA tiene que ejecutar dicho *script* como usuario *SYS* y, asignar al usuario en cuestión, el papel (*role*) correspondiente.

Una vez que se ha configurado convenientemente un usuario para que pueda acceder al plan de ejecución y a las estadísticas, basta habilitar el *AUTOTRACE* para que, al ejecutar una sentencia SQL, nos aparezca el plan de ejecución así como los correspondientes valores estadísticos.

Los valores estadísticos más importantes mostrados por la base de datos Oracle, una vez activado el comando *AUTOTRACE*, son los siguientes:

- *DB block gets*: Número de operaciones de entrada/salida realizadas sobre la memoria caché.
- *Consistent gets*: Número de operaciones de entrada/salida realizadas sobre los segmentos de *rollback* debido a cambios en la memoria caché.
- *Physical reads*: Número de bloques leídos desde el disco.
- *Sorts (memory)*: Número de operaciones realizadas en memoria para ordenar los datos.
- *Sorts (disk)*: Número de operaciones realizadas en disco para ordenar los datos.

A la hora de mejorar el rendimiento de una sentencia SQL o PL/SQL, debemos conseguir que el número de *db block gets*, *consistent gets* y *physical reads* sea bajo comparado con el número de registros devueltos por dicha sentencia. Por otro lado, la ordenación de los datos debe realizarse, siempre que sea posible, en memoria.

En cuanto a lo que se refiere al plan de ejecución, desde este enlace podéis acceder a un ejemplo de sentencia SQL con su correspondiente plan de ejecución y una breve interpretación de dicho plan: Ejemplo de plan de ejecución.

Como un primer consejo a la hora de analizar un plan de ejecución, me gustaría indicar que lo primero que hay que evitar son los *FULL SCAN* (recorrido de todos los registros de una tabla). No obstante, hay determinadas circunstancias bajo las que un *FULL SCAN* puede ser recomendable; así, cuando una tabla

tiene pocos registros, puede ser conveniente realizar un *FULL SCAN*, en vez de acceder a la misma a través de un índice.

El optimizador PL/SQL basado en normas (Rule-Based Optimizer)

Veremos algunas de las características del **optimizador PL/SQL basado en normas** (*Rule-Based Optimizer*). Lo primero que quiero mencionar es que Oracle recomienda utilizar el optimizador PLSQL basado en costes (*cost-based optimizer*), no obstante, en algunos casos, el hecho de tener que activar las estadísticas de la base de datos para poder utilizar este último optimizador, puede hacer que resulte interesante utilizar el optimizador basado en normas y dejar las estadísticas desactivadas para no afectar al rendimiento de la base de datos.

El optimizador PLSQL basado en normas utiliza siempre que puede los índices, incluso cuando las tablas son pequeñas o cuando el número de registros que devuelve la sentencia *SELECT* es un porcentaje elevado con respecto al número total de registros de la tabla, casos para los que es mejor realizar un escaneado total (*full scan*) ya que la respuesta es más rápida (mejora el rendimiento). Esto es debido a que el optimizador basado en normas no hace uso de valores estadísticos, tales como el número total de registros de una tabla.

El optimizador PL/SQL basado en normas hace uso del siguiente orden de prioridades para determinar cual va a ser la forma de acceder a las tablas y determinar finalmente cual va a ser el plan de ejecución:

Prio	Forma de acceso
1	Single row by ROWID
2	Single row by cluster join
3	Single row by hash cluster key with unique or primary key
4	Single row by unique or primary key
5	Cluster join
6	Hash cluster key
7	Indexed cluster key
8	Composite index

- 9 Single-column index
- 10 Bounded range search on indexed column
- 11 Unbounded range search on indexed column
- 12 Sort-merge join
- 13 MAX or MIN of indexed column
- 14 ORDER BY on indexed column
- 15 Full table scan

Los distintos métodos de acceso los he dejado en inglés, ya que es bastante complicado traducir esta terminología. En el presente artículo no voy a explicar cuales son las diferencias existentes entre las distintas formas de acceso. No obstante, en sucesivos artículos pondré algunos ejemplos que permitirán diferenciar estos conceptos.

Siguiendo con el tema que concierne a este *post*, el optimizador basado en normas analiza la sintaxis de la sentencia SQL para establecer los distintos métodos de acceso a las tablas. Básicamente lo que hace es determinar todas las formas de acceso posibles y escoger aquella que tiene una prioridad menor.

Este esquema siempre asume que un escaneado total (*full scan*) es el peor método de acceso (prioridad 15). Sin embargo, ya he mencionado al principio del artículo que esto no siempre es verdad.

Estos métodos de acceso, así como otros adicionales, están también disponibles para el optimizador PL/SQL basado en costes. Sin embargo, este optimizador ignora las prioridades, y determina el coste esperado de ejecución de la sentencia SQL para cada uno de las formas de acceso posibles basándose en las estadísticas, escogiendo después aquella forma de acceso con el menor coste estimado. Muchas funcionalidades del Oracle, como los *hash joins*, *star queries* e histogramas, sólo están disponibles para el optimizador PLSQL basado en costes.

Hints en PL/SQL para el modo de optimización

Los *hints* son pistas que se dan al optimizador SQL de Oracle para que elabore el plan de ejecución de una sentencia DML (sentencias de manipulación de datos como select, insert, update, delete, etc) según nosotros le aconsejemos. En este primer artículo sobre los *hints* voy a empezar hablando de aquellos que se utilizan para seleccionar el modo de trabajar del optimizador Oracle. Estos *hints*, hablando desde un punto de vista práctico, no son muy utilizados, aunque no por ello pueden dejar de ser útiles en determinadas circunstancias.

Los *hints* se incorporan a una sentencia DML en forma de comentario y deben ir justo detrás del comando principal. Por ejemplo, si se tratara de una sentencia SELECT el formato sería el siguiente:

```
SELECT /*+ COMANDO-HINT */ ...
```

Estos son los *hints* que se pueden utilizar para cambiar el modo de optimización del optimizador Oracle:

/*+ RULE */ - Fuerza a que se utilice el optimizador basado en normas (rule-based optimizer). Con este optimizador los planes de ejecución cambian según sea la sintaxis de la sentencia DML, no utiliza las estadísticas asociadas con las tablas de la base de datos Oracle y no calcula los costes del plan de ejecución. El optimizador basado en normas no ha cambiado desde la versión 6 de Oracle.

/*+ CHOOSE */ - Fuerza a que se utilice el optimizador basado en costes (cost-based optimizer). Este optimizador construye los planes de ejecución basándose en las estadísticas almacenadas en el diccionario de datos. Tiene en consideración el número de lecturas lógicas (el factor más importante), la utilización de la CPU junto con los accesos a disco y a memoria, y el uso de la red (cuando los datos residen en diferentes servidores). Una de las ventajas de utilizar el optimizador basado en costes es que Oracle lo está mejorándolo continuamente.

/*+ ALL_ROWS */ - Fuerza a que se utilice el optimizador basado en costes y

optimiza el plan de ejecución de la sentencia DML para que devuelva todas las filas en el menor tiempo posible. Es la opción por defecto del optimizador basado en costes y es la solución apropiada para procesos en masa e informes, en los que son necesarias todas las filas para empezar a trabajar con ellas.

/*+ FIRST_ROWS (n) */ - También fuerza a que se utilice el optimizador basado en costes y optimiza el plan de ejecución de la sentencia DML para que devuelva las "n" primeras filas en el menor tiempo posible. Esto es idóneo para procesos iterativos y bucles, en los que podemos ir trabajando con las primeras filas mientras se recuperan el resto de resultados. Obviamente este *hint* no será considerado por el optimizador si se utilizan funciones de grupo como MAX, SUM, AVG, etc.

Hints en PL/SQL para determinar el método de acceso

Ya hemos hablado de los *hints* para el modo optimización. En este apartado continuaremos hablando de los *hints* pero, en concreto, de aquellos que permiten indicar al optimizador Oracle el modo en que se debe acceder a los datos de las tablas. Este tipo de *hints* resultan extremadamente eficaces a la hora de optimizar una sentencia SQL.

En su día ya indiqué cual es la sintaxis de los *hints* pero creo que no está de más que la muestre de nuevo:

```
{ DELETE | INSERT | SELECT | UPDATE } /*+ HINT (parámetros) */
```

o

```
{ DELETE | INSERT | SELECT | UPDATE } --+ HINT (parámetros)
```

Los *hints* básicos que sirven para determinar el método de acceso a los datos de una tabla Oracle son los siguientes:

/*+ FULL (nombre_tabla) */ - Fuerza a que se realice la búsqueda accediendo a todos los registros de la tabla indicada. Cuando las tablas tienen un número reducido de registros puede resultar bueno para el rendimiento de una sentencia

DML el forzar un escaneado completo de la tabla en lugar de que el optimizador decida acceder a dicha tabla mediante un índice, ya que, en estos casos, el acceso por índice suele ser más lento.

/*+ ROWID (nombre_tabla) */ - Fuerza a que se acceda a la tabla utilizando el ROWID (identificativo único de los registros de una tabla). Este tipo de *hint*, por si solo, no es muy útil.

/*+ INDEX (nombre_tabla [nombre_índice] ...) */ - Fuerza a que se acceda a la tabla utilizando, en sentido ascendente, el índice indicado. Muchos problemas de rendimiento vienen causados por el hecho de que el optimizador Oracle decide acceder a una tabla utilizando un índice incorrecto. Mediante este *hint* podemos indicarle al optimizador que utilice el índice que nosotros consideremos adecuado.

/*+ INDEX_DESC (nombre_tabla [nombre_índice] ...) */ - Idéntico al anterior *hint* pero en este caso el acceso a través del índice se hace en sentido descendente.

/*+ AND_EQUAL (nombre_tabla [nombre_índice] ...) */ - Este *hint* se utiliza para forzar el uso de más de un índice (se utilizarían los índices indicados como parámetros) y, después, fusionar los índices quedándose con los registros encontrados en todas las búsquedas por índice realizadas.

/*+ INDEX_FFS (nombre_tabla [nombre_índice] ...) */ - Fuerza el acceso a los datos de la tabla mediante una búsqueda (*Scan*) rápida (*Fast*) y total (*Full*) sobre el índice indicado. Es parecido a utilizar el *hint FULL* pero sobre un índice en lugar de una tabla, lo cual, difícilmente, puede ser bueno para el rendimiento de una sentencia DML.

/*+ NO_INDEX (nombre_tabla [nombre_índice] ...) */ - Indica al optimizador que no se utilicen los índices indicados. Puede ser útil cuando no tengamos claro cual es el mejor índice que debe ser utilizado para acceder a una tabla pero, por contra, sepamos que podemos tener problemas de rendimiento si

se accede a la tabla por un determinado índice y queramos evitar que esto ocurra.

Puesta a punto de sentencias SQL

Si la siguiente sentencia SELECT:

```
SELECT *  
FROM empleados  
WHERE nombre = 'Francisco'  
AND estado_civil = 'S' -- Soltero
```

nos está dando tiempos de ejecución largos, esto querrá decir que:

1. Obviamente la tabla empleados es de un tamaño considerable.
2. La tabla no está adecuadamente indexada o que, aún habiéndose creado el índice adecuado, lógicamente un índice sobre la columna "nombre", el optimizador SQL decide utilizar el índice sobre otra columna.

Si tenemos problemas de "performance" con la sentencia SELECT de arriba y suponiendo que tenemos un índice sobre la columna "nombre", esto nos debe llevar a pensar que existe otro índice sobre la columna "estado_civil" que es el que el optimizador está utilizando para ejecutar la sentencia.

Los índices sobre columnas de este tipo, con un rango de valores pequeño (soltero, casado, viudo,...) y con una distribución de valores más o menos homogénea para algunos de estos valores (podemos pensar que el número de solteros va a ser similar al de casados y bastante mayor que el de viudos), van a causar problemas en los tiempos de ejecución. Este tipo de índices sólo son interesantes cuando uno de los posibles valores que puede tomar la columna es mucho menos numeroso que el resto y la sentencia SQL la vamos a limitar en base a dicho valor. Por ejemplo, si nuestra sentencia fuera:

```
SELECT *  
FROM empleados  
WHERE nombre = 'Francisco'  
AND estado_civil = 'V' -- Viudo
```

Podemos tener la certeza de que los tiempos de ejecución no serían tan malos. En cambio, cuando hacemos la comparación "estado_civil = 'S'", estamos prácticamente haciendo un "full scan" sobre un índice y esto es incluso peor que hacer un "full scan" sobre una tabla.

¿Qué tenemos que hacer para arreglar la primera sentencia? Debemos forzar al optimizador a usar el índice sobre la columna "nombre", para ello tenemos dos opciones:

```
SELECT *  
FROM empleados  
WHERE nombre = 'Francisco'  
AND estado_civil!!'' = 'S' -- Soltero
```

Al añadir "!!" el optimizador no podrá utilizar el índice sobre la columna estado_civil.

También, suponiendo que el identificador del índice sobre la columna "nombre" es INXEMPLEADOSNOMBRE, podemos añadir el siguiente "hint" en nuestra sentencia SELECT:

```
SELECT * /*+ INDEX(empleados INXEMPLEADOSNOMBRE) */  
FROM empleados  
WHERE nombre = 'Francisco'  
AND estado_civil = 'S' -- Soltero
```

Esto hace que el optimizador utilice el índice sobre la columna "nombre" aunque exista otro índice sobre la columna "estado_civil".

Cómo usar la utilidad de trazado del SQL de Oracle

La utilidad de trazado del SQL de las bases de datos Oracle nos permite analizar el rendimiento de un determinado programa PL/SQL. Esta funcionalidad nos va a permitir obtener información acerca del rendimiento de todas las sentencias SQL que se ejecuten durante la ejecución del programa PLSQL.

Para utilizar la herramienta de trazado del PL/SQL de Oracle deberemos seguir cinco pasos:

- 1) Inicializar los parámetros relativos a esta funcionalidad SQL.
- 2) Activar la traza SQL.
- 3) Ejecutar la aplicación que queremos analizar y desactivar la traza cuando termine.
- 4) Formatear el fichero producido por la traza SQL con el comando *TKPROF*.
- 5) Interpretar la salida del comando *TKPROF* y, si es necesario, optimizar nuestro programa PLSQL.

En este artículo voy a hablar sobre los tres primeros apartados.

Inicialización de los parámetros de trazado Oracle

La utilidad de trazado puede, opcionalmente, proporcionar información acerca de los tiempos de ejecución. Para que esta información quede almacenada es necesario activar el parámetro *TIMED_STATISTICS*.

Dicho parámetro se puede activar a nivel de base de datos mediante su inclusión en el fichero de parámetros de la base de datos Oracle (nota: una vez incluido el parámetro es necesario reinicializar la base de datos para que el cambio tenga efecto):

```
TIMED_STATISTICS = TRUE
```

Este parámetro también se puede asignar dinámicamente a nivel de sesión ejecutando el siguiente comando:

```
SQL> ALTER SESSION SET timed_statistics=true;
```

La activación de este parámetro puede afectar ligeramente al rendimiento de la base de datos por lo que normalmente este parámetro está desactivado

Existen otros dos parámetros que nos permiten controlar el tamaño y el nombre del directorio donde se generará el fichero de trazado:

```
MAX_DUMP_FILE_SIZE = n
```

```
USER_DUMP_DEST = nombre_directorio
```

El valor por defecto del parámetro *MAX_DUMP_FILE_SIZE* es 500, es decir, que nuestro fichero de trazado podrá ocupar 500 bloques del disco duro. Este parámetro puede cambiar también a nivel de sesión con el comando *ALTER SESSION*.

El valor por defecto del parámetro *USER_DUMP_DEST* depende del sistema operativo y no puede ser cambiado a nivel de sesión. Por lo tanto, al ser un parámetro global del sistema, su valor sólo pueden cambiarlo los administradores de la base de datos utilizando el comando *ALTER SYSTEM*.

Para obtener información acerca de los valores que toman los distintos parámetros podemos ejecutar la siguiente sentencia:

```
SELECT name, value  
FROM v$parameter  
WHERE name LIKE '%dump%'
```

Obviamente se necesita tener acceso a la vista (view) *V\$PARAMETER* para poder visualizar esta información.

Activación de la traza SQL

Podemos activar la traza a nivel de sistema mediante la inclusión en el fichero de parámetros de la siguiente línea:

```
SQL_TRACE = TRUE
```

Esta posibilidad es poco recomendable y no se debe implementar en ningún sistema en producción. La activación de la traza a nivel de sistema puede afectar seriamente el rendimiento de la base de datos Oracle.

Por ello es mucho más recomendable activar la traza a nivel de sesión con el comando:

```
SQL> ALTER SESSION SET sql_trace = true;
```

También se puede utilizar el paquete estándar *DBMS_SESSION*. Esta posibilidad es particularmente útil si queremos activar y desactivar la traza dentro del un procedimiento o función PL/SQL.

```
SQL> EXECUTE dbms_session.set_sql_trace (true);
```

Por otro lado, los DBA (*Database Administrators*) pueden activar la traza sobre una sesión de usuario concreta utilizando el siguiente comando:

```
SQL> EXECUTE dbms_system.set_sql_trace_in_session (session_id,  
serial_id, true);
```

Los valores *session_id* y *serial_id* adecuados los tiene que identificar el DBA mediante el análisis de los registros incluidos en la vista *V\$SESSION*, los campos de esta vista que se corresponden con estos valores son *SID* y *SERIAL#*.

Desactivación de la traza SQL

Cuando la ejecución del programa que estamos optimizando termina, debemos

proceder a desactivar la traza utilizando cualquiera de los métodos mencionados anteriormente, sustituyendo la palabra *TRUE* por *FALSE*.

Si la traza se activó a nivel de sesión, entonces cuando la sesión termina, la traza se desactiva automáticamente.

Identificación del fichero de trazado

Para identificar los ficheros de trazado debemos ir al directorio especificado por el parámetro *USER_DUMP_DEST* y, normalmente, el fichero de trazado será aquel que se ha generado más recientemente.

La identificación puede complicarse cuando hay varios usuarios generando ficheros de trazado al mismo tiempo. En este caso podemos utilizar un *script* estándar denominado *readtrace.sql*. Este *script* crea un procedimiento que abre nuestro fichero de trazado utilizando el paquete *UTL_FILE*. El nombre por defecto del fichero de trazado que se genera es *username.trc* pero puede cambiarse fácilmente.

```
SQL> @readtrace.sql
SQL> ALTER SESSION SET sql_trace = true;
SQL> SELECT * FROM nombre_tabla;
SQL> execute gettrace ('nombre_fichero_trazado');
```

El siguiente paso sería utilizar el programa *TKPROF* para poder interpretar el contenido binario del fichero de trazado, pero esto será objeto de otro artículo.

Sql loader

Sql Loader es una utilidad que proporciona Oracle para cargar datos a una base de datos desde un fichero externo, normalmente un fichero de texto aunque también pueden ser ficheros binarios.

Al SQLLoader (**sqlldr**) se le pasan como parametros (los más importantes) el fichero que contiene los datos que se van a cargar y la ruta del fichero de control que contiene las acciones a realizar. El formato de los datos, donde se cargaran y cualquier otro tipo de control.

? Durante la ejecución genera tres tipos de ficheros:

- 1.- Fichero Log.
- 2.- Fichero Bad.
- 3.- Fichero Discard.

Las características de estos ficheros son:

- 1.- Log ? Muestra estadísticas de la carga.
- 2.- Bad ? Almacena los registros rechazados por datos incorrectos.
- 3.- Discard ? Almacena los registros que no han cumplido los criterios de selección de la cláusula WHEN.

La cláusula WHEN carga aquellas filas que cumplen los criterios especificados en ella.

? SQL-Loader utiliza dos mecanismos para cargar datos, estos mecanismos son:

- 1.- Path Convencional.
- 2.- Path Directo.

1.- Path Convencional

Este mecanismo utiliza un buffer llamado BIND ARRAY. El mecanismo se basa en lo siguiente: Se leen varios registros que son almacenados en el Bind array, cuando este se llena o no quedan más datos para leer se pasan a Oracle para que inserte los datos en las tablas.

2.- Path Directo.

Este mecanismo consiste en colocar los registros en bloques de la Base de datos ya formateados y escritos directamente en la Base de datos, pasando por alto la mayor parte de procesamiento RDBMS.

Este mecanismo es más rápido que el convencional, pero tiene muchas más restricciones.

? Para realizar una carga correcta de la información se necesita un fichero que interprete los campos de los registros con el fin de almacenarlos de forma adecuada. Pues este fichero es el fichero de control.

? Por otra parte tenemos los datos. Estos pueden ir en el mismo fichero de control. Los datos se pueden cargar en diferentes formatos:

- 1.- Formato Binario ? No puede tratar campos o columnas de longitud variable.
- 2.- Formato Carácter ? Puede tratar tanto campos o columnas de longitud fija como variable.

Para los formatos variables, el Sistema Operativo utiliza una marca de final de registro (como el newline).

? Existen dos tipos de limitadores que se utilizan para delimitar los campos:

- 1.- Terminated. ? Campos seguidos por un carácter especificado.
Ejemplo la coma: 1,2,3.....
- 2.- Enclosed. ? Campos precedidos y seguidos por un carácter especificado.
Ejemplo las comillas: "ENCLOSED".

? Se distingue entre dos tipos de registros:

- 1.- Registros físicos ? Se consideran registros físicos cada línea del fichero de donde deseamos cargar los datos a Oracle.
- 2.- Registros lógicos ? Se consideran registros lógicos cada fila de la tabla de Base de datos.

Se conoce como continuation fields aquella combinación de registros físicos que forman un sólo registro lógico.

ESTRUCTURA DEL ARCHIVO DE CONTROL

El archivo de control suele tener como mínimo la siguiente estructura:

LOAD DATA

INFILE

INTO TABLE

[BEGIN DATA]

.....
.....
.....

Explicación del uso de las sentencias anteriores:

LOAD DATA ? Esta declaración es obligatoria en el fichero de control. Siempre debe aparecer al principio de este.

INFILE ? En este parámetro se especifica de donde se van a extraer los datos a almacenar en las tablas. Es decir, si se pone un “*”, le estamos diciendo al SQL-Loader que los datos se encuentran en el mismo fichero de control, justo después de la sentencia BEGIN DATA. Por otro lado, se puede indicar, al SQL-Loader, que coja los datos de un fichero de la siguiente manera: INFILE

‘nombre_fichero’.

- Se pueden poner tantos INFILE’s como ficheros de datos deseemos cargar.
- Sí no se especifica la extensión del fichero en INFILE, por defecto se utilizará “.dat”.

Ejemplos:

? Carga de datos a partir de unos datos incluidos en el fichero de control:

INFILE *

...
...

BEGIN DATA

.....
.....

? Carga de datos a partir de un fichero de datos especificado:

INFILE ‘C:\fichero_de_datos.dat’

.....
.....

[BEGIN DATA] ? No se debe poner

INTO TABLE ? En esta sentencia, se especifican las tablas que serán cargadas con la información. Dependiendo del tipo de carga que se vaya a realizar, se deben especificar los campos (columnas) de la tabla y su correspondiente formato.

[BEGIN DATA] ? Se pone esta sentencia cuando los datos a cargar en las tablas se encuentran en el fichero de control. Esta sentencia se debe poner siempre después de la última especificación de control, sino es así, podría suceder que se interpretasen datos como sentencias de control.

Seguidamente, se muestra un fichero de control más detallado y con los parámetros más usuales:

LOAD DATA ? Sentencia obligatoria

INFILE '<fichero_datos.dat>'

[INFILE] '<fichero_datos2.dat>'

.....

.....

[BADFILE] '<fichero_bad.bad>'

[DISCARDFILE] '<fichero_discard.dsc>'

[INSERT] ? Por defecto, la tabla debe estar vacía sino da error.

[APPEND] ? Añade nuevas filas a la tabla.

[REPLACE] ? Elimina las filas existentes y añade las nuevas.

INTO TABLE <nombre_tabla>

[BEGIN DATA] ? Obligatorio sí y sólo sí, se ha especificado '*' en INFILE.

[WHEN] ? Obligatorio sí y sólo sí, se ha declarado DISCARDFILE.

¿Cómo ejecutar SQL-Loader?

Dependiendo de la versión de Oracle en la que estemos trabajando, existe un determinado SQL-Loader. Normalmente, el fichero ejecutable para Windows NT suele llamarse sqlldrXX.exe donde XX es el número de la versión del producto. Este fichero se suele almacenar en el directorio: C:\orant\bin .

Ejemplos:

Para la versión de Oracle 7.3.4.0.0 seria sqlldr73.exe

Para la versión de Oracle 8.0.5.0.0 seria sqlldr80.exe

Etc...

Para ejecutar el SQL-Loader, debemos llamar al programa anterior desde la línea de comandos. La llamada seria algo como:

```
C:\> sqlldrXX argumento=valor [, argumento=valor,] ...
```

Para saber que argumentos podemos utilizar con SQL-Loader, podemos utilizar la ayuda de la siguiente manera:

```
C:\> sqlldrXX help=y
```

Imaginemos que los argumentos que estamos utilizando, son los que utilizamos en la mayoría de ejecuciones de SQL-Loader. Pues, existe una manera de hacer que estos argumentos sean almacenados en un fichero externo y que por medio de la cláusula PARFILE sea llamado como si estuviese incluido en el mismo fichero de control. Este método nos ahorra el tener que escribir largas líneas de argumentos en la línea de comandos. Además de facilitar su entendimiento.

Lista de Argumentos

USERID ? Se debe poner el usuario y clave de Oracle, sino se pone el programa lo solicita.

CONTROL ? Se debe poner el nombre del fichero de control. La extensión es .CTL.

LOG ? Se debe poner el nombre del fichero de Log que guarda información del proceso de carga. La extensión por defecto es .LOG.

BAD ? Se debe poner el nombre del fichero donde se almacenarán las causas de los errores que provocaron las inserciones durante la carga. La extensión por defecto es .BAD.

DATA ? Se debe poner el nombre del fichero de datos a cargar. Por defecto se coge el nombre del fichero de control + la extensión .DAT.

[DISCARD] ? Nombre del fichero opcional de descartados. La extensión es .DSC.

[DISCARDMAX] ? Parámetro que indica el número de descartados que se permiten antes de acabar la carga. Para parar en el primer registro se debe poner “0”. Por defecto su valor es “todos”.

[SKIP] ? Especifica el número de registros o filas que desde el principio del fichero no deben ser cargados en la ejecución del SQL-Loader. Por defecto es “0”.

[LOAD] ? Especifica el número máximo de registros que serán cargados, después de saltar los especificados en la cláusula SKIP. Por defecto cargará todos.

[ERRORS] ? Especifica el número máximo de errores que permiten que se produzcan antes de acabar la carga. Para no permitir ningún error poner =0, para permitirlos todos poner un valor extremadamente elevado. Por defecto, este argumento permite 50 errores.

[ROWS] ? En Path convencional, este argumento especifica el número de filas del BIND ARRAY (por defecto 64). Por otra parte, en Path directo este argumento indica la cantidad entera de registros que deseamos cargar en la Base de datos (Por defecto es salvar todos de una sólo vez al final de la carga).

[BINDSIZE] ? Especifica el tamaño máximo en Bytes del BIND ARRAY, este argumento machaca el valor del argumento ROWS.

[SILENT] ? Suprime los mensajes que aparecen durante la ejecución. Tales como cabeceras, pies de página, errores, descartes etc...

[DIRECT] ? Indica el modo Path a utilizar. Si su valor es TRUE usará el Path Directo, si el valor es FALSE usará el Path Convencional (Por defecto).

[PARFILE] ? Especifica el nombre del fichero que contiene otros parámetros de la línea de comandos.