

PASOS PARA INGRESAR AL INTRANET DE ALUMNOS

1. Solicitar al Área de Informes ó Administración de Sistemas UNI su Usuario y Contraseña la cual se podrá ser cambiada al iniciar sesión por primera vez.

IMPORTANTE: Es de suma importancia que usted verifique y actualice sus Apellidos y Nombres de forma correcta en el **Área de Administración**, debido a que los mismos irán en su certificado (cuando lo tramite), caso contrario si los datos están incorrectos (al momento de la entrega del certificado) la emisión del nuevo certificado con los datos corregidos tendrá un costo de 20 soles.

2. Una vez obtenido el Usuario y la Contraseña ingrese a **www. Sistemasuni.edu.pe** y click en Alumnos.



3. Ingresar su Usuario y su Password proporcionado en el Área de Informes ó Administración.

Inicio de sesión

Por favor, introduzca su nombre y contraseña para iniciar sesión.

Ingresa tu correo
Ingresa tu contraseña
Iniciar sesión

- Una vez ingresado Ud. Puede revisar sus cursos que llevo y sus respectivas notas en campo Cursos Libres y Módulos, respectivamente de la modalidad en la cual Ud. Se matriculó. También puede verificar sus Apellidos y Nombres, de estar incompletos o erróneos. Acercarse al Área de Informes ó Administración para su corrección.



Datos generales

Información del perfil del alumno.

CÓDIGO:
APELLIDOS Y NOMBRES:
GENERO:*
FECHA DE NACIMIENTO:*
ESTADO CIVIL:*
TELEFONO:
CELULAR:*
CORREO ELECTRONICO:*
DIRECCIÓN:
ACTUALIZAR MIS DATOS

- De no visualizar sus notas hasta luego de 7 días después de finalizado el curso, comunicarse al teléfono **200-9060 opción 3** (Área de Administración).



Introducción al SQL y PL/SQL

Este es el primer curso que nos permite entrar en contacto, conocimiento, uso y profundización del administrador de bases corporativas por excelencia.

Oracle 11g nos proporcionará las herramientas para gestionar la información y hacerla eficiente para todo tipo de negocio.

Parte I : Contiene capítulos desde el 1 al 4. (Impreso)

Parte II : Contiene los capítulos desde el 1 al 8 (PDF)

Presentación

Oracle Database Server es un software que tiene por finalidad gestionar la información que se almacena en una base de datos, además de protegerla, validarla y ofrecer herramientas para un eficiente manejo de la misma.

Oracle es el servidor de base de datos más flexible y más potente del mundo. Desde que apareció en el mercado informático, Oracle no ha dejado de crecer en calidad y en aplicaciones que están al servicio de todas las áreas de la gestión empresarial.

En esta ocasión ha entrado al mercado informático la versión **Oracle Database 11g**, que sobre todo trae la ventaja de la reducción de costes con **Oracle Database 11g Versión 2**. Esto se da mediante la consolidación de información en grids empresariales, la reducción de la necesidad de almacenamiento y la supresión de la inactividad de los sistemas de respaldo. La versión 2 de Oracle Database 11g cuenta con magníficas características adicionales: funciones valiosas y sencillas que facilitan el grid computing, importantes mejoras de compresión y otra innovación del mercado que permite actualizar las aplicaciones de la base de datos sin desconectar a los usuarios. Venga el 21 de octubre y descubra dos aspectos importantes:

Reducción del gasto en informática gracias a:

- la consolidación de grids empresariales;
- la reducción de las necesidades de almacenamiento;
- la eliminación de los períodos de inactividad en las bases de datos de respaldo

Aumento de la tasa de utilización del hardware para:

- incrementar la eficacia energética al máximo;
- disminuir el coste total de propiedad (TCO).

Sea de los primeros en conocer las novedosas funciones de la versión 2 de Oracle Database 11g, como:

- posibilidad de actualizar las aplicaciones de la base de datos sin desconectar a los usuarios;
- funciones sencillas de utilizar que facilitan el grid computing;
- automatización de las principales actividades de gestión de sistemas; y mucho más.

El presente curso tiene como objetivo principal formar personas que sean capaces de manejar los objetos de la base de datos de Oracle. Estos objetos son responsables no solo del almacenamiento sino de la generación de transacciones que resuelven los problemas del negocio y que los automatizan.

Aprender el lenguaje SQL y su adicional PL/SQL posibilita la construcción de aplicaciones que pueden ser llamadas o utilizadas desde cualquier plataforma. El servidor de base de datos no solo es un almacén de información, sino que será capaz de contener programas de distintas índoles que aprovechen al máximo los recursos de Oracle.

Los objetivos del curso son:

- Utilizar las sentencias SELECT en todas sus modalidades para recuperar información de la base de datos mostrándolas de diversas formas.
- Utilizar la sintaxis DDL del SQL para crear, modificar o destruir objetos de la base de datos de Oracle.
- Reconocer y aplicar las reglas de las restricciones para salvaguardar la integridad de la base de datos.
- Utilizar la sintaxis DML del SQL para insertar, modificar o borrar filas de datos de las tablas y vistas.
- Crear programas utilizando la extensión del PL/SQL en diversas modalidades como son los procediéndooos almacenados, los disparadores o las funciones.

Así, Oracle es un servidor de base de datos (RDBMS) que se encarga de organizar, administrar, proteger y procesar la información de un negocio. Entre las ventajas más saltantes que ofrece encontramos:

- Escalabilidad de departamentos a ubicaciones e-business de empresa.
- Arquitectura robusta, fiable, disponible y segura.
- Un modelo de desarrollo, opciones sencillas de desarrollo.
- Aprovecha el juego de habilidades actual de una organización en toda la plataforma de la base de datos (incluidos SQL, PL/SQL, Java y XML).
- Una interfaz de gestión para todas las aplicaciones.
- Tecnologías del estándar de la industria, sin bloqueo por propiedad.

Generalmente cuando pensamos en un servicio de base de datos solo lo hacemos centrándonos en su servidor de base de datos, pero en realidad, cuando instalamos el software contaremos con la potencia de varios productos integrados que funcionan e interactúan como uno solo:

Database.

La base de datos gestiona la información. No sólo los datos relacionales de objetos que se espera que gestione una base de datos de empresa, sino también los datos sin estructurar, como por ejemplo:

Hojas de cálculo, Documentos de Word, Presentaciones de PowerPoint, XML, Tipos de dato multimedia como MP3, gráficos, vídeo y más.

Los datos ni siquiera tienen que estar en la base. La base de datos tiene servicios a través de los que puede almacenar metadatos con la información almacenada en sistemas de archivo. Puede utilizar el servidor de base de datos para gestionar y servir información dondequiera que esté ubicada.

Enterprise Manager. El **Enterprise Manager** es un servicio adicional de la base de datos que permitirá administrarla de un modo sencillo mediante un grid de gestión

remota. Este servicio, basado en la arquitectura del dbconsole además de permitir realizar todas las tareas de administración, facilitará herramientas para el monitoreo del rendimiento de la base de datos.

En el presente manual se recorrerán aspectos y contenidos relacionados con la manipulación de la base de datos, es decir, de la información.



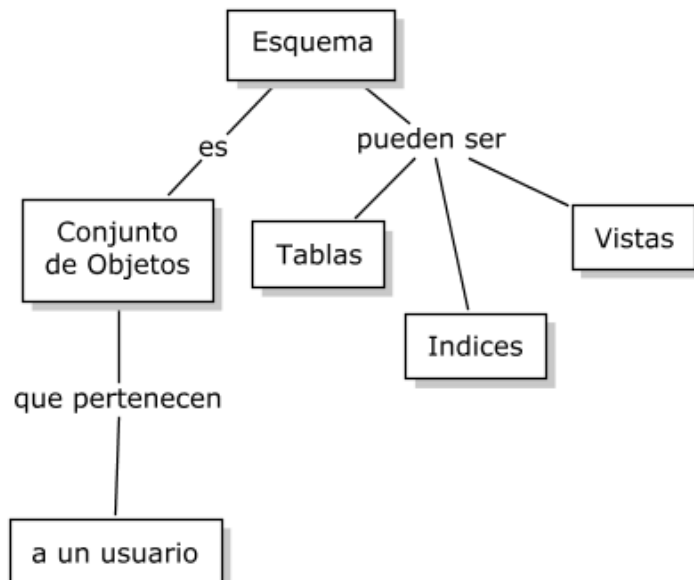
CAPÍTULO 1

EL LENGUAJE SQL

Contenido del Capítulo:

1. Esquema usado para el curso.
2. Herramienta de trabajo: SQL Developer
3. Estructura básica del comando SELECT.
4. Ordenamiento de datos y filtros de filas.
5. Funciones de una sola fila.

1. Esquema usado para el curso.



Un **esquema** es el conjunto de objetos que son pertenecen a un usuario. Es decir, existe la posibilidad que un usuario sea el owner o propietario de un conjunto de tablas. Esto implica que dicho usuario tiene la posibilidad de crear objetos y por lo tanto tiene los privilegios y espacios necesarios dentro de la base de datos.

En este curso utilizamos el esquema HR. Este es un usuario que viene con

unas tablas de ejemplo de la base de datos Oracle y que puede ser activado durante la instalación del producto.

Este esquema de ejemplo simula la información de una base de datos de gestión de personal. Las tablas que contiene son:


```

SQL> connect hr/hr
Connected.
SQL>
SQL>
SQL>
SQL>
SQL>
SQL>
SQL> SELECT TABLE_NAME
      2 FROM USER_TABLES
      3 ;

TABLE_NAME
-----
REGIONS
LOCATIONS
DEPARTMENTS
JOBS
EMPLOYEES
JOB_HISTORY
COUNTRIES

7 rows selected.

SQL>

```

Visualización de las tablas del usuario HR luego de la conexión

REGIONS. Contiene los datos de las regiones. Posee un ID y un nombre de la región.

```

SQL>
SQL> DESC REGIONS;

Name                               Null?    Type
-----
REGION_ID                          NOT NULL NUMBER
REGION_NAME                        VARCHAR2(25)

SQL>

```

LOCATIONS. Contiene los datos de las oficinas o direcciones de la empresa. Almacena un ID, la dirección, el código postal, la ciudad, el estado, y un identificador del país.

```

SQL> DESC LOCATIONS;

Name                               Null?    Type
-----
LOCATION_ID                          NOT NULL NUMBER(4)
STREET_ADDRESS                     VARCHAR2(40)
POSTAL_CODE                        VARCHAR2(12)
CITY                               NOT NULL VARCHAR2(30)
STATE_PROVINCE                     VARCHAR2(25)
COUNTRY_ID                         CHAR(2)

SQL>

```

DEPARTMENTS. Contiene los datos de cada departamento de la empresa. Guarda el ID, el nombre del departamento, el ID del jefe, y el ID de la dirección (relacionado con la tabla LOCATIONS).

```
SQL>
SQL> DESC DEPARTMENTS;
Name                                         Null?      Type
-----
DEPARTMENT_ID                               NOT NULL   NUMBER(4)
DEPARTMENT_NAME                             NOT NULL   VARCHAR2(30)
MANAGER_ID                                  NOT NULL   NUMBER(6)
LOCATION_ID                                   NOT NULL   NUMBER(4)
SQL>
```

JOBS. Contiene la descripción de los cargos o puestos de trabajo de la empresa. Guarda el ID del cargo, su título, el salario mínimo y el máximo.

```
SQL>
SQL>
SQL> DESC JOBS;
Name                                         Null?      Type
-----
JOB_ID                                       NOT NULL   VARCHAR2(10)
JOB_TITLE                                   NOT NULL   VARCHAR2(35)
MIN_SALARY                                  NOT NULL   NUMBER(6)
MAX_SALARY                                  NOT NULL   NUMBER(6)
SQL>
```

EMPLOYEES. Almacena los datos de los empleados de la empresa. Contiene el ID del empleado, primer nombre, apellidos, correo, número de teléfono, fecha de contratación, ID del cargo (relacionado con la tabla JOBS), sueldo, porcentaje de comisión, ID del jefe, ID del departamento para el que trabaja.

```
SQL>
SQL>
SQL>
SQL> DESC EMPLOYEES;
Name                                         Null?      Type
-----
EMPLOYEE_ID                               NOT NULL   NUMBER(6)
FIRST_NAME                                NOT NULL   VARCHAR2(20)
LAST_NAME                                  NOT NULL   VARCHAR2(25)
EMAIL                                       NOT NULL   VARCHAR2(25)
PHONE_NUMBER                              NOT NULL   VARCHAR2(20)
HIRE_DATE                                  NOT NULL   DATE
JOB_ID                                     NOT NULL   VARCHAR2(10)
SALARY                                    NOT NULL   NUMBER(8,2)
COMMISSION_PCT                             NOT NULL   NUMBER(2,2)
MANAGER_ID                                NOT NULL   NUMBER(6)
DEPARTMENT_ID                             NOT NULL   NUMBER(4)
SQL>
```

JOB_HISTORY. Es un historial de todos los puestos de trabajo que ha tenido el empleado. Guarda el ID del empleado, la fecha de inicio en el cargo y la fecha de fin, el ID del cargo y el ID del departamento.

```
SQL>
SQL> DESC JOB_HISTORY;
```

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID	NOT NULL	NUMBER(4)

```
SQL>
```

COUNTRIES. Guarda información de los países donde está la empresa. Guarda el ID del país, el nombre y el ID de la región.

```
SQL>
SQL>
SQL>
SQL> DESC COUNTRIES;
```

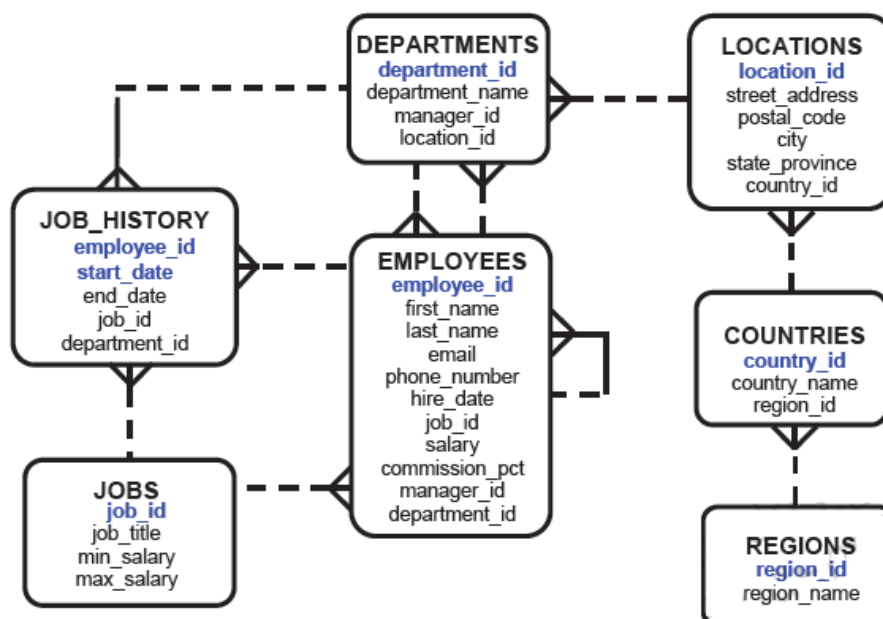
Name	Null?	Type
COUNTRY_ID	NOT NULL	CHAR(2)
COUNTRY_NAME		VARCHAR2(40)
REGION_ID		NUMBER

```
SQL>
```

```
SQL>
```

```
SQL>
```

En versiones anteriores existía el usuario SCOTT con tablas reducidas del mismo sistema de recursos humanos. En versiones anteriores este esquema. SCOTT, ya no es instalado por defecto, por tal razón usamos HR (human resources) para el trabajo del curso. Con estas tablas aprenderemos las sintaxis requeridas por este lenguaje SQL.



2. Herramienta de trabajo: el SQL Developer.



Oracle SQL Developer 3.0 (3.0.04.34)

March 2011

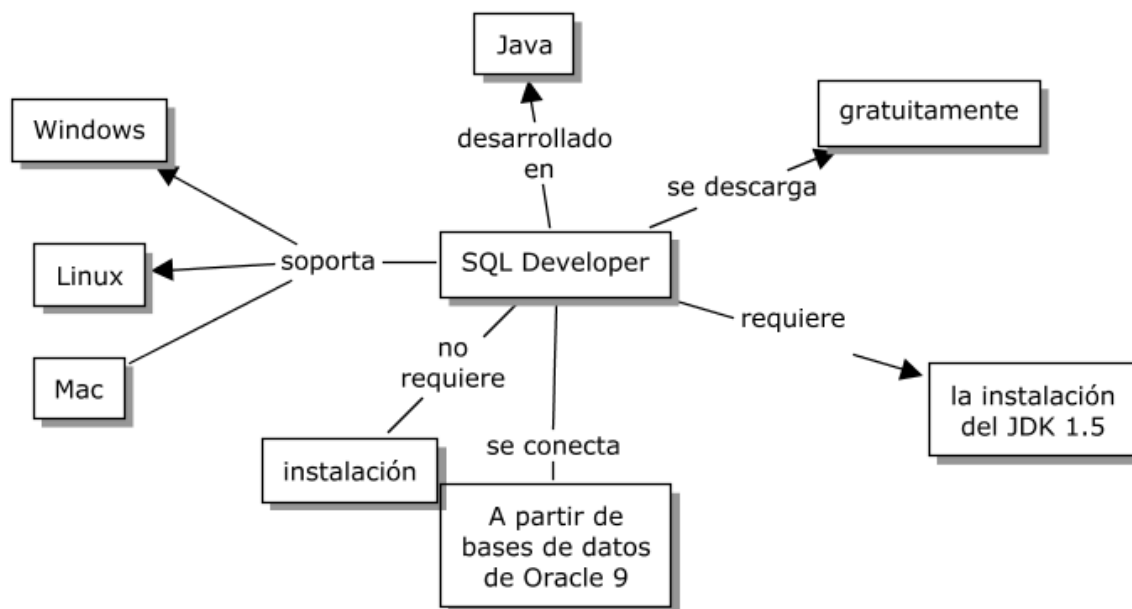
Oracle SQL Developer es la herramienta gráfica gratuita que proporciona Oracle para que no sea necesario utilizar

herramientas de terceros (como el conocido TOAD, o el PL/SQL Developer) para desarrollar, o simplemente para ejecutar consultas o scripts SQL, tanto DML como DDL, sobre bases de datos Oracle. Es decir, es una herramienta de interface gráfica que nos permite manipular la información de una base de datos.

La apariencia y funcionalidad es similar a la de otras herramientas de este tipo, por lo que es una buena opción si no tenemos especial predilección por otras herramientas.

Además, en las últimas versiones ha incorporado mejoras como permitir conectar con bases de datos no Oracle, como SQLServer, MySQL o Access. La conexión con MySQL o SQLServer se realiza a través de JDBC, y de manera bastante sencilla. Una vez establecida la conexión se pueden explorar los objetos de las bases de datos como si se tratara de una de Oracle, y ejecutar sobre ellas sentencias SQL, aunque en cuanto a funcionalidades más avanzadas como la creación de estructuras este tipo de conexión estará mucho más limitado.

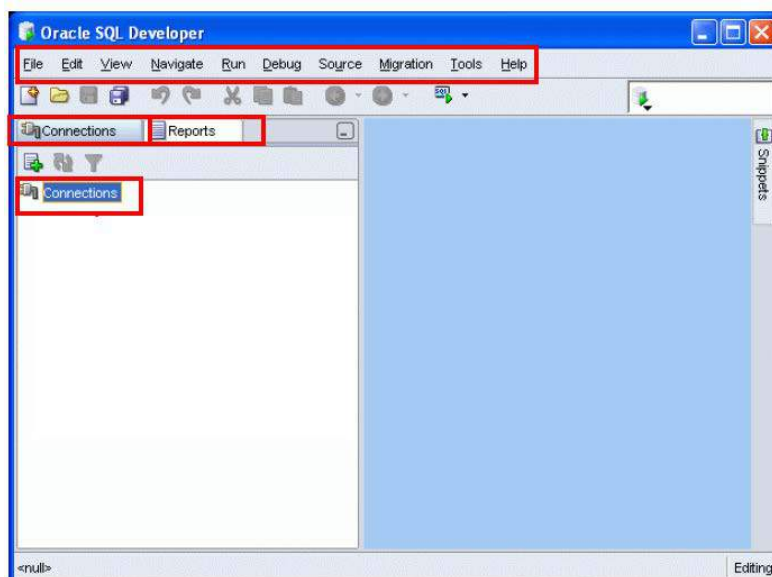
Esta herramienta se descarga desde Internet, desde la web de Oracle y no necesita ser instalada ni configurada, sino que es un archivo ejecutable que activa inmediatamente la interface para conectarse a cualquier base de datos Oracle además de otras plataformas.



La herramienta Oracle SQL Developer trabaja sobre la noción de **conexión**. Una conexión es un objeto parecido a un alias que permite ingresar a la base de datos mediante un usuario y una clase. Un SQL Developer puede tener varias conexiones, y según queramos entrar con un usuario u otro las podemos activar.

Cada vez que entramos con un usuario, entramos al esquema de dicho usuario, es decir, vemos directamente los objetos que le pertenecen y de los cuáles es propietario.

Al empezar el trabajo se debe crear una **conexión**:



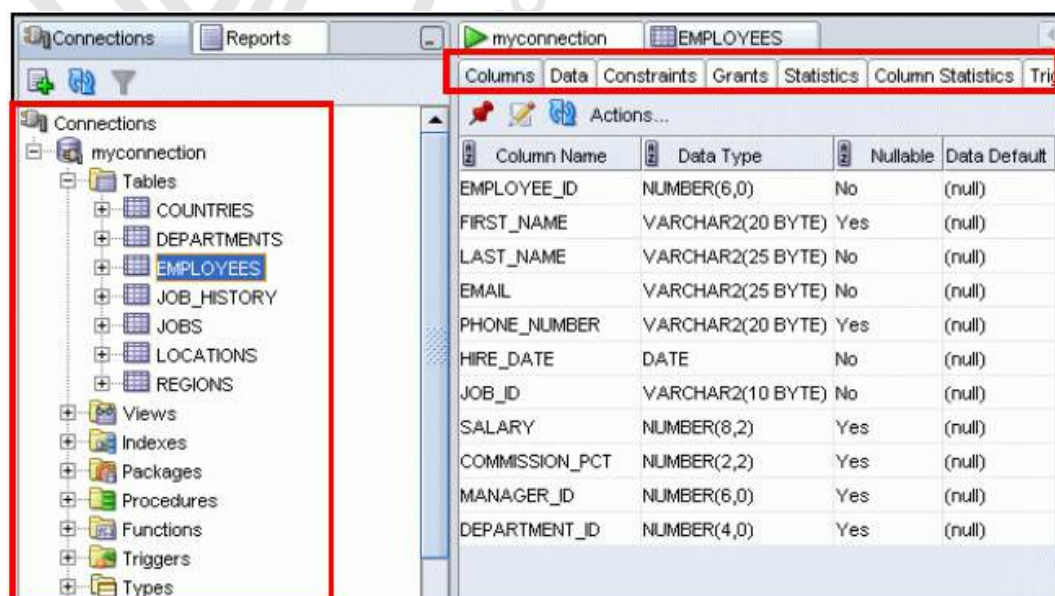
1. Clic en el ícono para crear una **nueva conexión**.
2. Se activa la ventana de propiedades de la conexión:
 - Nombre de la conexión.
 - Usuario.
 - Clave.
 - Si se desea que guarde la clave.

- El hostname, es el nombre o IP del equipo donde está corriendo el servidor Oracle.
- El puerto del listener, que por lo general es el 1521.
- El SID, es decir, el nombre de la instancia. Por lo general se le asigna el nombre de orcl, pero esto se define necesariamente en la instalación.

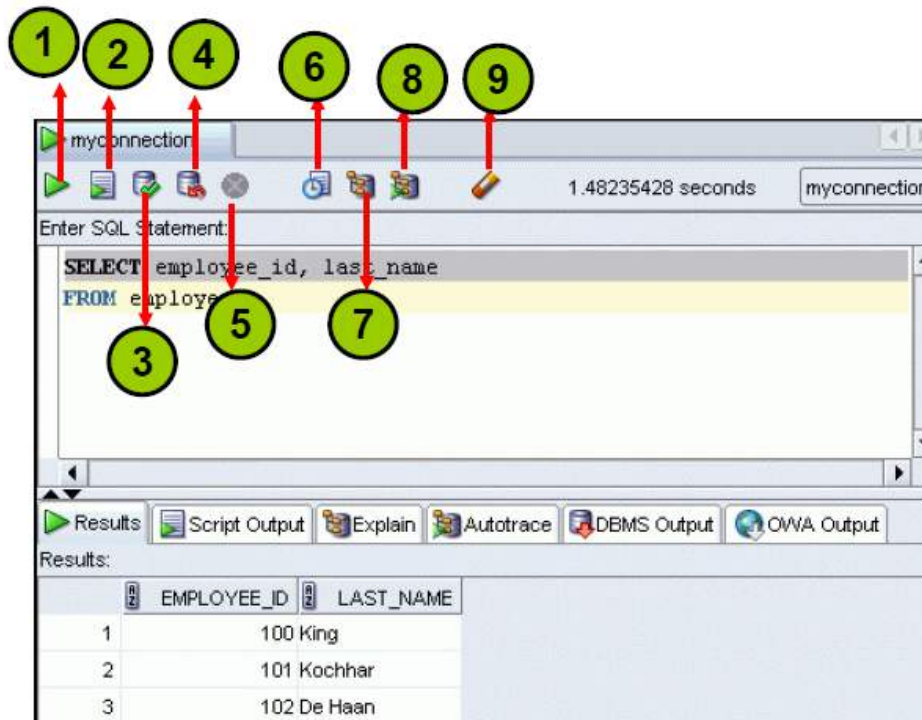
3. Hecho esto, se puede activar cualquiera de los botones:

- Probar la conexión.
- Guardarla.
- Conectarse.

Si nos conectamos con el usuario HR ingresaremos a ver directamente su esquema.



Con esta herramienta además podemos editar comandos en línea. Tiene un entorno para hacer la compilación de esos comandos y ver los resultados.



Contiene los siguientes botones de función:

1. Ejecuta la sentencia y la muestra en una cuadrícula.
2. Ejecuta la sentencia y la muestra como texto.
3. Confirma la ejecución de la sentencia: COMMIT.
4. Cancela los cambios de la sentencia: ROLLBACK.
5. Detiene la ejecución de la sentencia.
6. Muestra el historial de todos los comandos SQL ejecutados hasta el momento.
7. Realiza un seguimiento o traza del comando.
8. Limpia la ventana donde se digitan los comandos.

3. Sentencias básicas del SQL: SELECT

SQL es un lenguaje creado expresamente para manejar la información de la base de datos. Como lenguaje estándar posee cláusulas, sintaxis y funciones que se ajustan a los requerimientos de información.

Los comandos básicos son:

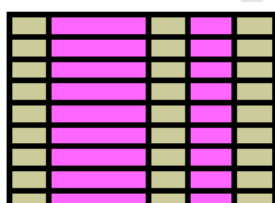
SELECT INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE COMMENT	Data definition language (DDL)
GRANT REVOKE	Data control language (DCL)
COMMIT ROLLBACK SAVEPOINT	Transaction control

Sentencias DML.

Estas sentencias son aquellas que se encargan de la manipulación de los datos contenidos en las tablas. Dentro de esos comandos, el más usado es la sentencia SELECT.

La sentencia SELECT es aquella que tiene por objetivo conectarse a la base de datos para extraer la información de una o más tablas que desea visualizar el usuario.

Esta sentencia funciona de manera individual ya que a modo de consulta y a través de parámetros reconoce cuáles son los campos o columnas que desean ser mostrados en la pantalla. No establece ninguna conexión permanente con la base de datos, y mucho menos genera bloqueos en dichas tablas.



El objeto principal que maneja la sentencia SELECT es la tabla.

Una **tabla** es la unidad mínima de almacenamiento y está compuesta por columnas y filas.

En la imagen se aprecian cinco columnas (vertical), y nueve filas (horizontal). Es importante precisar que las tablas no se almacenan físicamente de manera bidimensional sino se van apilando, pero cada vez que recuperamos la información de una base de datos, la apreciaremos así.

Cada columna guarda un dato de la entidad referida. En la imagen se observa el contenido de la tabla **JOBS**. Esta tiene las columnas JOB_ID que guarda el ID del

cargo, JOB_TITLE guarda el nombre, MIN_SALARY y MAX_SALARY almacenan el sueldo mínimo y máximo respectivamente para cada puesto de trabajo de la empresa. Es decir, esa tabla tiene 4 columnas y muchas filas de información.

```
SQL> SELECT *
      2 FROM JOBS;
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY	
AD_PRES	President	20080	40000	
AD_UP	Administration Vice President	15000	30000	
AD_ASST	Administration Assistant	3000	6000	
FI_MGR	Finance Manager	8200	16000	
FI_ACCOUNT	Accountant	4200	9000	
AC_MGR	Accounting Manager	8200	16000	
AC_ACCOUNT	Public Accountant	4200	9000	En
SA_MAN	Sales Manager	10000	20080	su
SA_REP	Sales Representative	6000	12008	
PU_MAN	Purchasing Manager	8000	15000	
PU_CLERK	Purchasing Clerk	2500	5500	

forma más simple, una sentencia SELECT debe incluir lo siguiente:

- Una cláusula SELECT, que especifica las columnas que se han de mostrar
- Una cláusula FROM, que especifica la tabla que contiene las columnas listadas en la cláusula SELECT

En la sintaxis:

```
SELECT      * | {[DISTINCT] column|expression [alias],...}
FROM table;
```

SELECT	es una lista de una o más columnas de la tabla
*	selecciona todas las columnas
DISTINCT	suprime los duplicados
column expression	selecciona la expresión o columna especificada
alias	asigna cabeceras diferentes a las columnas seleccionadas
FROM table	especifica la tabla que contiene las columnas

Nota:

A lo largo de este curso, las palabras cláusula, sentencia y palabra clave se usan según lo siguiente:

Una palabra clave hace referencia a un elemento SQL individual.
Por ejemplo, SELECT y FROM son palabras clave.

Una cláusula es una parte de una sentencia SQL.
Por ejemplo, SELECT employee_id, last_name, ... es una cláusula.

Una sentencia es una combinación de dos o más cláusulas.
Por ejemplo, SELECT * FROM employees es una sentencia SQL.

Selección de Todas las Columnas de Todas las Filas

Se puede visualizar todas las columnas de datos en una tabla si escribe un asterisco (*) detrás de la palabra clave SELECT.

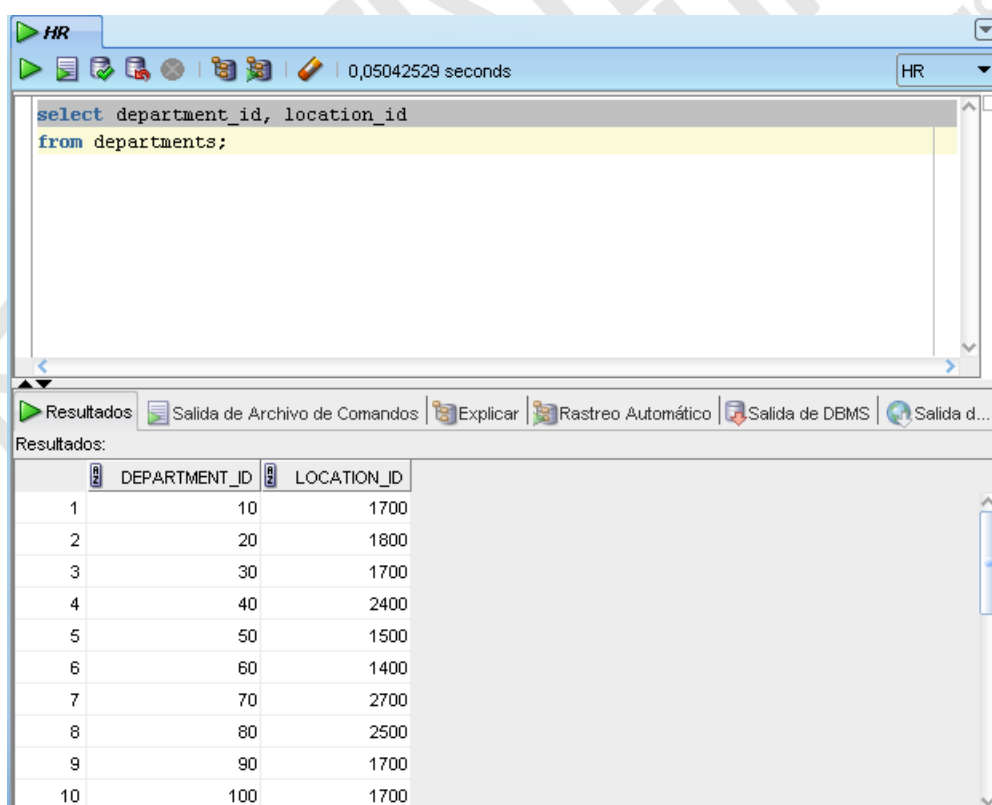
```
SELECT *  
FROM departments;
```

También puede visualizar todas las columnas de la tabla si enumera todas las columnas después de la palabra clave SELECT.

Selección de Columnas específicas de Todas las Filas

Puede utilizar la sentencia SELECT para visualizar columnas específicas de la tabla si indica los nombres de columna, separados por comas. El ejemplo muestra todos los números de departamento y de ubicación de la tabla departments.

```
SELECT department_id, location_id  
FROM departments;
```



The screenshot shows the SQL Developer interface. The top toolbar includes icons for running queries, saving, and other database operations. The main text area contains the SQL query: `select department_id, location_id from departments;`. Below the text area, the 'Resultados' (Results) tab is active, displaying a table with the query results. The table has two columns: 'DEPARTMENT_ID' and 'LOCATION_ID'. It contains 10 rows of data, numbered 1 through 10 in the first column.

	DEPARTMENT_ID	LOCATION_ID
1	10	1700
2	20	1800
3	30	1700
4	40	2400
5	50	1500
6	60	1400
7	70	2700
8	80	2500
9	90	1700
10	100	1700

En la cláusula SELECT, especifique las columnas que desee en el orden en que quiera que aparezcan en el resultado. Por ejemplo, para mostrar la ubicación, antes que el número de departamento, de izquierda a derecha, utilice la siguiente sentencia:

ESCRITURA DE SENTENCIAS SQL

Utilizando las sencillas reglas e instrucciones siguientes, puede construir sentencias válidas que sean fáciles tanto de leer como de editar:

- Las sentencias SQL no son sensibles a mayúsculas/minúsculas a menos que se indique.
- Las sentencias SQL se pueden introducir en una o más líneas.
- Las palabras clave no se pueden dividir entre líneas ni abreviar.
- Normalmente las cláusulas están colocadas en líneas separadas por motivos de legibilidad y facilidad de edición.
- Los sangrados se deben utilizar para que los códigos sean más legibles.
- Generalmente, las palabras clave se introducen en mayúsculas; todas las demás palabras, como los nombres de tabla y columnas se introducen en minúsculas.

EXPRESIONES ARITMÉTICAS

Puede que necesite modificar la forma en la que se muestra la información, realizar cálculos, o examinar supuestos hipotéticos. Todo ello es posible si se utilizan las expresiones aritméticas. Una expresión aritmética puede contener nombres de columna, valores numéricos constantes y operadores aritméticos.

LOS OPERADORES ARITMÉTICOS

Los operadores aritméticos disponibles en SQL son los que se encuentran a continuación:

+	Suma
-	Resta
*	Multiplicación
/	División

Puede utilizar operadores aritméticos en cualquier cláusula de una sentencia SQL excepto en la cláusula FROM.

Uso de los operadores aritméticos.

```
SELECT last_name, salary, salary + 300
FROM employees;
```

El ejemplo de la transparencia utiliza el operador de suma para calcular un incremento del salario de 300 para todos los empleados y muestra una nueva columna en el resultado.

Observe que la columna calculada resultante `salary + 300` no es una columna nueva de la tabla `employees`, sino sólo de visualización. Por defecto, el nombre de una columna nueva surge del cálculo que la generó, en este caso, `salary + 300`.

Nota: El servidor Oracle11g ignora los espacios en blanco anteriores y posteriores al operador aritmético.

Prioridad de Operador

Si una expresión aritmética contiene más de un operador, la multiplicación y la división se evalúan en primer lugar. Si los operadores incluidos en una expresión son de idéntica prioridad, entonces la evaluación se hace de izquierda a derecha.

Puede utilizar paréntesis para forzar que la expresión incluida entre paréntesis se evalúe en primer lugar.



- La multiplicación y la división tienen prioridad sobre la suma y la resta.
- Los operadores de idéntica prioridad se evalúan de izquierda a derecha.
- Los paréntesis se utilizan para forzar evaluaciones prioritarias y para clarificar sentencias.

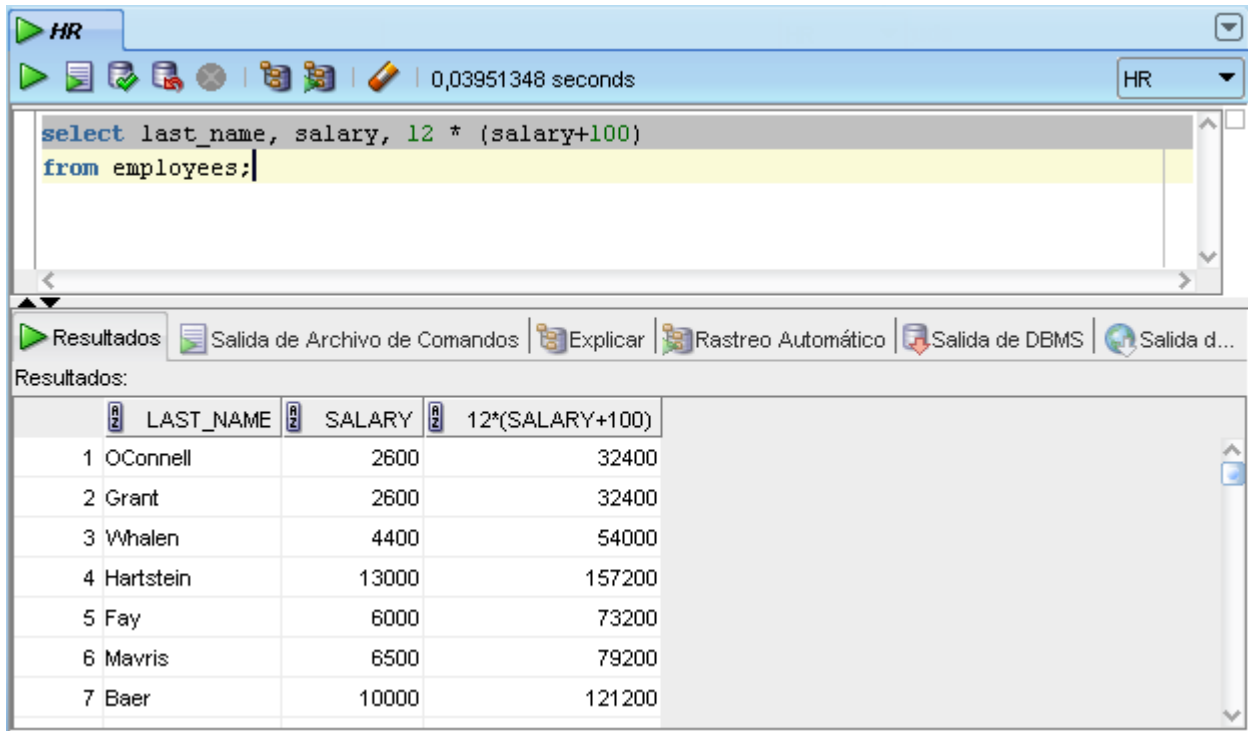
Así las siguientes expresiones darán resultados diferentes:

```
SELECT last_name, salary, 12*salary+100
FROM employees;
```

Se multiplica doce por el sueldo y se le suma 100.

```
SELECT last_name, salary, 12*(salary+100)
FROM employees;
```

Se suma el sueldo más cien y se multiplica por doce



The screenshot shows a SQL query execution window with the following SQL statement:

```
select last_name, salary, 12 * (salary+100)
from employees;
```

The results are displayed in a table with the following columns: LAST_NAME, SALARY, and 12*(SALARY+100). The results are as follows:

	LAST_NAME	SALARY	12*(SALARY+100)
1	OConnell	2600	32400
2	Grant	2600	32400
3	Whalen	4400	54000
4	Hartstein	13000	157200
5	Fay	6000	73200
6	Mavris	6500	79200
7	Baer	10000	121200

LOS VALORES NULOS

Si una fila no tiene el valor de datos de una columna determinada, se dice que ese valor es nulo, o que contiene un null.

Un valor nulo es un valor no disponible, no asignado, desconocido, o no aplicable. Un valor nulo no es lo mismo que cero ni que un espacio. Cero es un número y un espacio es un carácter.

Las columnas de cualquier tipo de dato pueden contener valores nulos. Sin embargo, algunas restricciones, NOT NULL y PRIMARY KEY, evitan que se utilicen valores nulos en la columna.

En la columna COMMISSION_PCT de la tabla EMPLOYEES, observe que sólo el director de ventas o el representante de ventas puede percibir una comisión. Los demás empleados no tienen derecho a ganar comisiones. Un valor nulo representa este hecho.

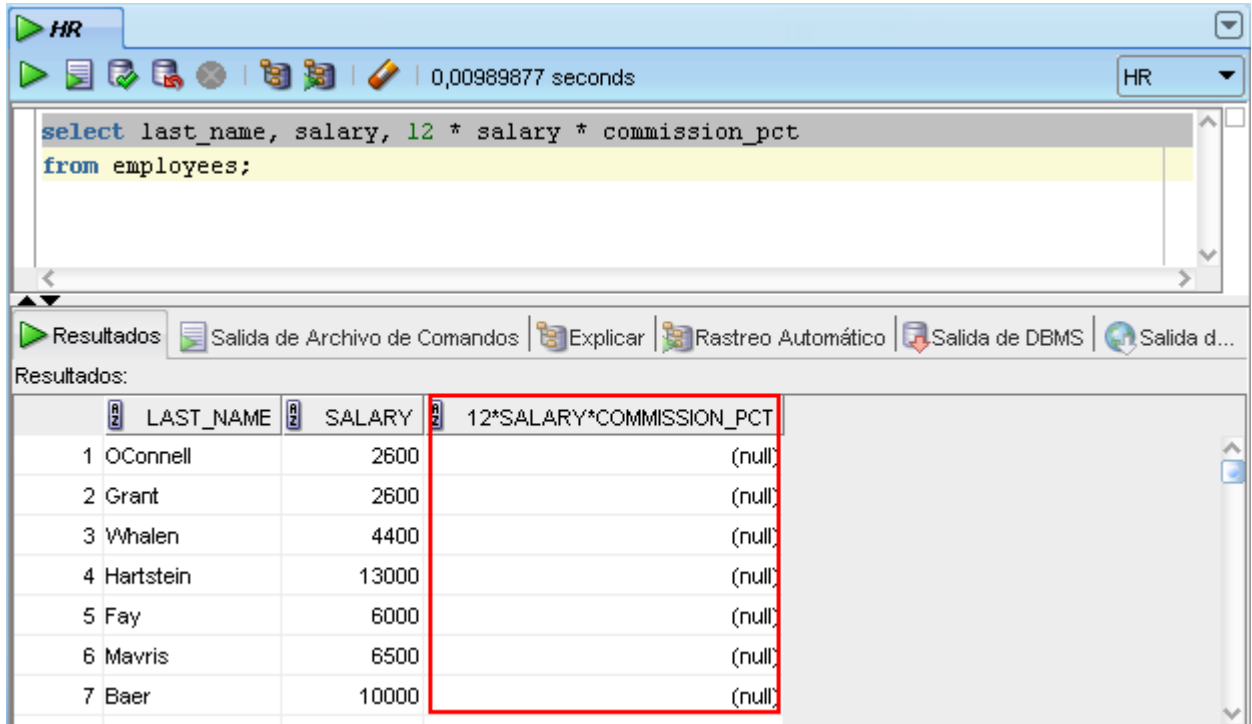
```
SELECT last_name, job_id, salary, commission_pct
FROM employees;
```

El valor nulo es considerado un vacío real, no es ni espacio en blanco ni cero.

Si un valor de columna de una expresión aritmética es nulo, el resultado es null. Por ejemplo, si intenta dividir entre cero, recibirá un mensaje de error. Sin embargo, si divide un número entre un valor nulo, el resultado será nulo o desconocido.

Por eso si realizamos la consulta que se muestra a continuación, las filas que contengan valores nulos tendrán como resultado nulo también.

```
SELECT last_name, 12*salary*commission_pct
FROM employees;
```



The screenshot shows the SQL Developer interface with a query window and a results window. The query window contains the following SQL statement:

```
select last_name, salary, 12 * salary * commission_pct
from employees;
```

The results window displays the following data:

	LAST_NAME	SALARY	12*SALARY*COMMISSION_PCT
1	OConnell	2600	(null)
2	Grant	2600	(null)
3	Whalen	4400	(null)
4	Hartstein	13000	(null)
5	Fay	6000	(null)
6	Mavris	6500	(null)
7	Baer	10000	(null)

En el caso del ejemplo solo tendrán resultados aquellos que tenían valores. Los que asumían un valor NULL seguirían siendo NULL a pesar de tener valores en las otras columnas.

El puntero NULL entonces es un objeto que absorbe los cálculos de cualquier tipo, y por eso hay que tener cuidado en su uso y aplicación.

Alias de Columna

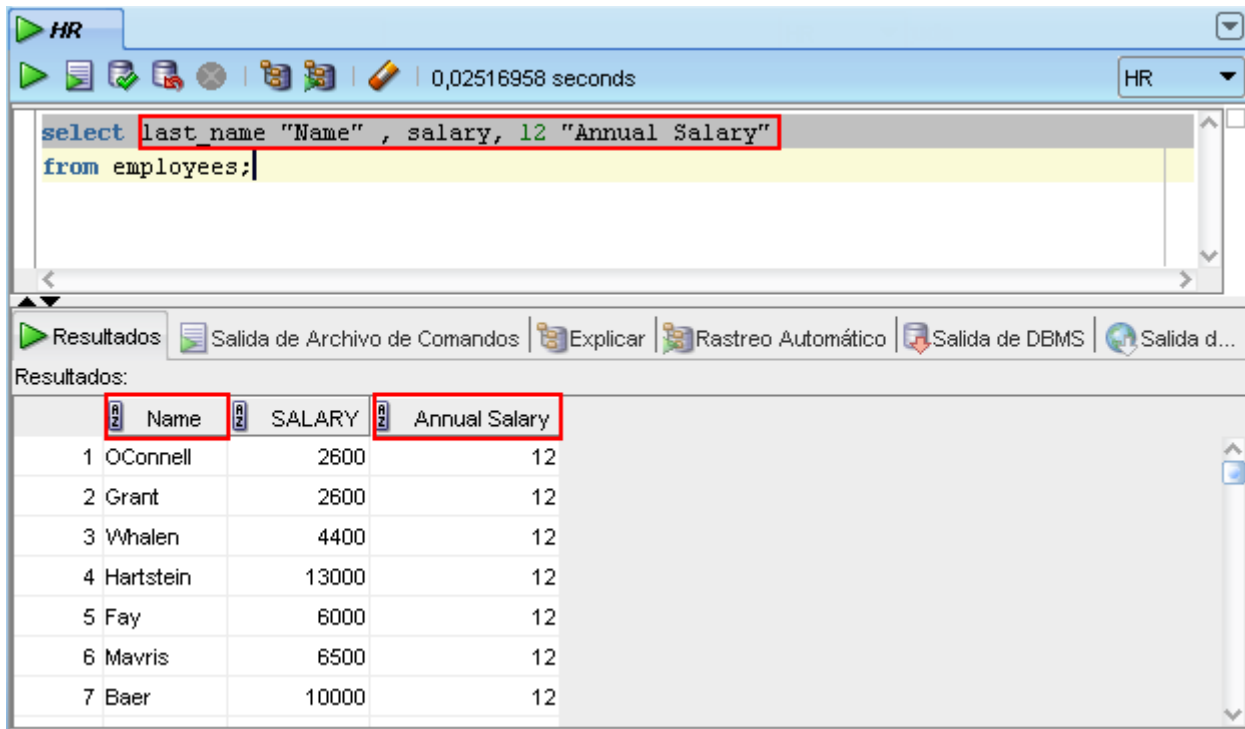
Cuando muestra el resultado de una consulta, normalmente se utiliza el nombre de la columna seleccionada como la cabecera de columna. Es posible que esta cabecera no sea descriptiva y por tanto, sea difícil de comprender. Puede cambiar la cabecera de una columna mediante un alias de columna.

Especifique el alias tras la columna en la lista SELECT utilizando un espacio como separador. Por defecto, las cabeceras de alias aparecen en mayúsculas. Si el alias contiene espacios o caracteres especiales (como # o \$), o es sensible a mayúsculas/minúsculas, escriba el alias entre comillas dobles (" ").

```
SELECT last_name AS name, commission_pct comm
FROM employees;
```

En este primer ejemplo, no se respetan los formatos que se escriben en la sentencia para el alias. Además, se puede observar que el mismo resultado se obtiene si se coloca la palabra AS o no.

```
SELECT last_name "Name", salary*12 "Annual Salary"
FROM employees;
```



The screenshot shows a SQL query execution window with the following query:

```
select last_name "Name" , salary, 12 "Annual Salary"
from employees;
```

The results are displayed in a table with the following columns: Name, SALARY, and Annual Salary. The data is as follows:

	Name	SALARY	Annual Salary
1	OConnell	2600	12
2	Grant	2600	12
3	Whalen	4400	12
4	Hartstein	13000	12
5	Fay	6000	12
6	Mavris	6500	12
7	Baer	10000	12

Cuando se coloca la comilla doble, los identificadores de las alias pueden tener cualquier formato, hasta pueden dejar espacios en blanco o usar otros símbolos que de otra forma estarían prohibidos para las sentencias SQL.

Operador de Concatenación

Un operador de concatenación es aquel que se encarga de:

- Concatenar columnas o cadenas de caracteres a otras columnas.
- Está representado por dos barras verticales (||).
- Crea una columna resultante que es una expresión de caracteres.

Puede enlazar columnas a otras columnas, expresiones aritméticas o valores constantes para crear una expresión de caracteres mediante el operador de concatenación (||). Las columnas a cada lado del operador se combinan para hacer una única columna de resultados.

```
SELECT last_name||job_id AS "Employees"
FROM employees;
```

Este operador de concatenación ofrece la posibilidad de concatenar infinitas columnas para una sola expresión. A esta nueva columna se le puede agregar un alias para hacer legible el resultado.

Concatenación de cadenas de Caracteres Literales

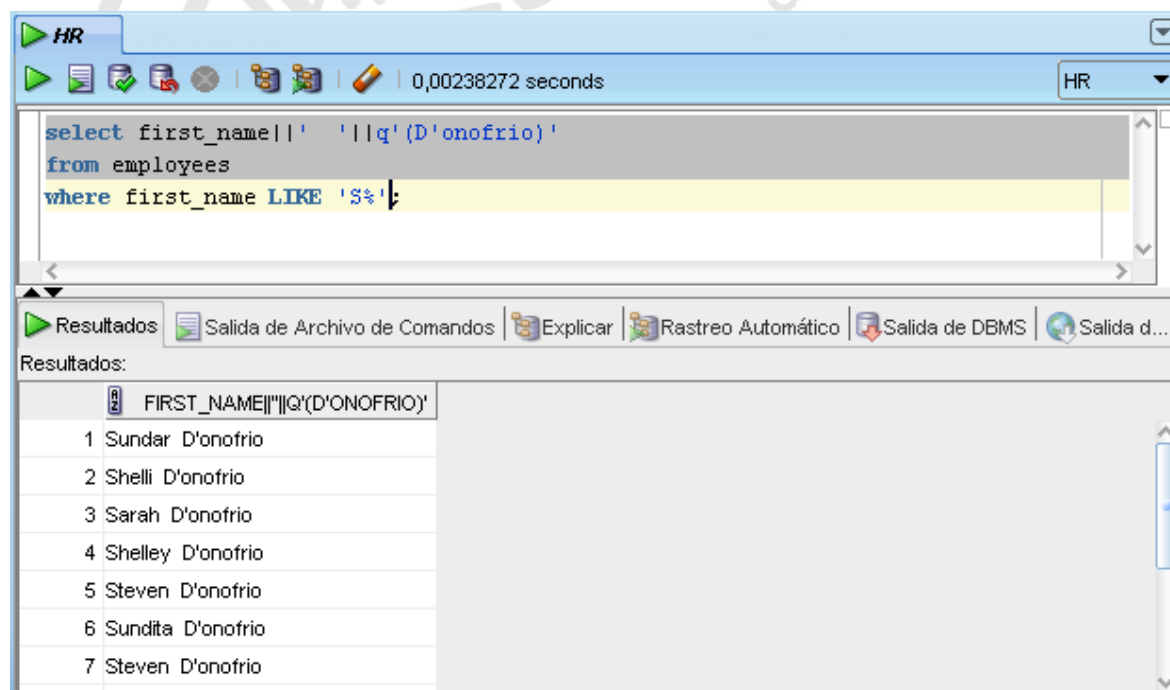
Un literal es un carácter, un número o una fecha que está incluido en la lista SELECT y que no es un nombre de columna o un alias de columna. Un literal está impreso para cada fila devuelta. Las cadenas literales de texto de formato libre se pueden incluir en el resultado de la consulta y son tratados de la misma manera que una columna en la lista SELECT.

Mientras que los literales de caracteres y fecha se deben escribir entre comillas simples (' '), los literales numéricos no.

```
SELECT last_name || ' es un ' || job_id
       AS "Detalles del empleado"
FROM   employees;
```

Existen algunos casos en los que se desea concatenar las comillas dobles o la comilla simple. En ese caso, se debe distinguir las comillas simples del carácter que se desea concatenar para no tener ningún error de interpretación del Oracle. Es por eso que existe el operador "q".

```
SELECT first_name || q'('D' onofrio)'
FROM   employees
WHERE  first_name LIKE 'S%';
```



The screenshot shows the Oracle SQL Developer interface. The top toolbar includes icons for HR, SQL, PL/SQL, and other database tools. The main window displays a SQL query:

```
select first_name || ' ' || q'('D' onofrio)'
from employees
where first_name LIKE 'S%';
```

Below the query editor, the "Resultados" (Results) tab is active, showing a table with the following data:

	FIRST_NAME ' ' Q('D' ONOFRIO)
1	Sundar D'onofrio
2	Shelli D'onofrio
3	Sarah D'onofrio
4	Shelley D'onofrio
5	Steven D'onofrio
6	Sundita D'onofrio
7	Steven D'onofrio

En el ejemplo, el carácter que indica el inicio de la cadena ya no es la comilla simple, sino '('. Esto se le indica con el operador "q", por lo tanto para indicar el fin de la cadena se usa ')'. Para esa expresión el apostrofe es solo considerado un carácter más.

Las filas Duplicadas

A menos que indique lo contrario, Oracle muestra los resultados de una consulta sin eliminar filas duplicadas. Hay muchas tablas que almacenan filas con columnas que poseen valores duplicados. En algunos casos se necesita recuperar esa información sin que se muestren los datos duplicados.

Por ejemplo ejecuten la siguiente consulta:

```
SELECT department_id  
FROM employees;
```

Se observa repetidos los ID de los departamentos porque en un departamento pueden trabajar varios empleados.

Para eliminar las filas duplicadas en el resultado, incluya la palabra clave DISTINCT en la cláusula SELECT inmediatamente detrás de la palabra clave SELECT.

Puede especificar múltiples columnas detrás del cualificador DISTINCT. Este cualificador afecta a todas las columnas seleccionadas y el resultado es cada distinta combinación de columnas.

```
SELECT DISTINCT department_id  
FROM employees;
```

Estructura de una tabla.

Una tabla está formada por columnas. Cada columna almacena datos de un determinado tipo y en algunos casos estas columnas tienen restricciones.

Se puede visualizar la estructura de una tabla mediante el comando DESCRIBE.

El comando muestra los nombres de columna y los tipos de dato, así como si una columna debe contener datos.

En la sintaxis:

```
DESC[RIBE] tablename
```

Tablename es el nombre de cualquier tabla, vista o sinónimo existente accesible para el usuario.

DESCRIBE employees

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

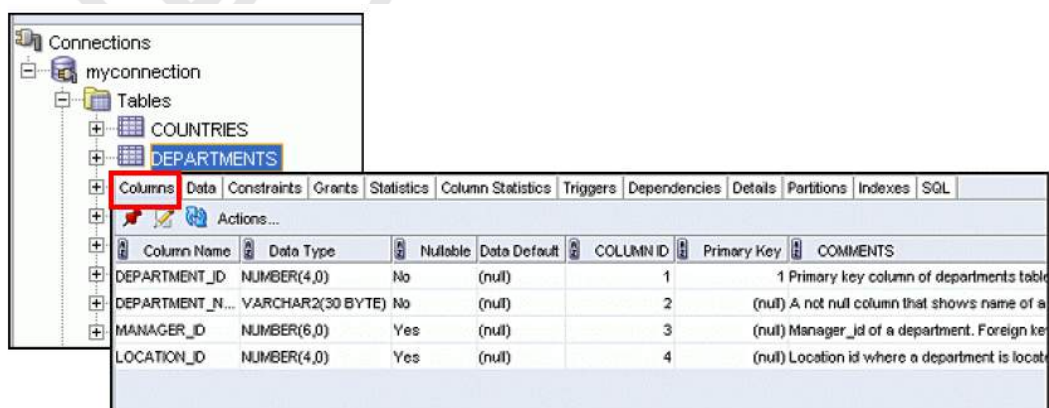
En el resultado se muestran los siguientes datos:

- Null?** Indica si la columna debe contener datos; NOT NULL indica que una columna debe contener datos, pero cuando está vacío indica que la columna es opcional.
- Type** Muestra el tipo de dato de una columna, es decir, el formato de almacenamiento interno.

Los tipos de dato se describen en la siguiente tabla:

- CHAR** Almacena caracteres de longitud fija.
- VARCHAR2** Almacena caracteres de longitud variables, es decir, si el dato es más pequeño que el tamaño del campo Oracle libera los espacios no usados.
- NUMBER** Almacena números.
- DATE** Almacena día, mes, año, hora, minuto y segundo.

La herramienta SQL Developer nos permite ver la estructura de la tabla de forma gráfica:



Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
DEPARTMENT_ID	NUMBER(4,0)	No	(null)	1	1	Primary key column of departments table
DEPARTMENT_N...	VARCHAR2(30 BYTE)	No	(null)	2		(null) A not null column that shows name of a
MANAGER_ID	NUMBER(6,0)	Yes	(null)	3		(null) Manager_id of a department. Foreign ke
LOCATION_ID	NUMBER(4,0)	Yes	(null)	4		(null) Location id where a department is locat

4. Filtro de Filas en una Selección

En algunos casos no se desea ver como resultado todas las filas de una tabla. Suponga que desea mostrar todos los empleados del departamento 90. Las filas con el valor 90 en la columna son las únicas que se devuelven.

Este método de restricción es la base de la cláusula WHERE en SQL.

```
SELECT  *|{ [DISTINCT] column|expression [alias], ... }  
FROM    table  
[WHERE  condition(s)];
```

Puede restringir las filas devueltas por la consulta utilizando la cláusula WHERE. Esta cláusula contiene una condición que se debe cumplir y sigue directamente a la cláusula FROM. Si la condición es verdadera, se devuelve la fila que cumple la condición.

En la sintaxis:

WHERE	Restringe la consulta a las filas que cumplen una Condición.
condition	Está formado por nombres de columna, expresiones, constantes y un operador de comparación

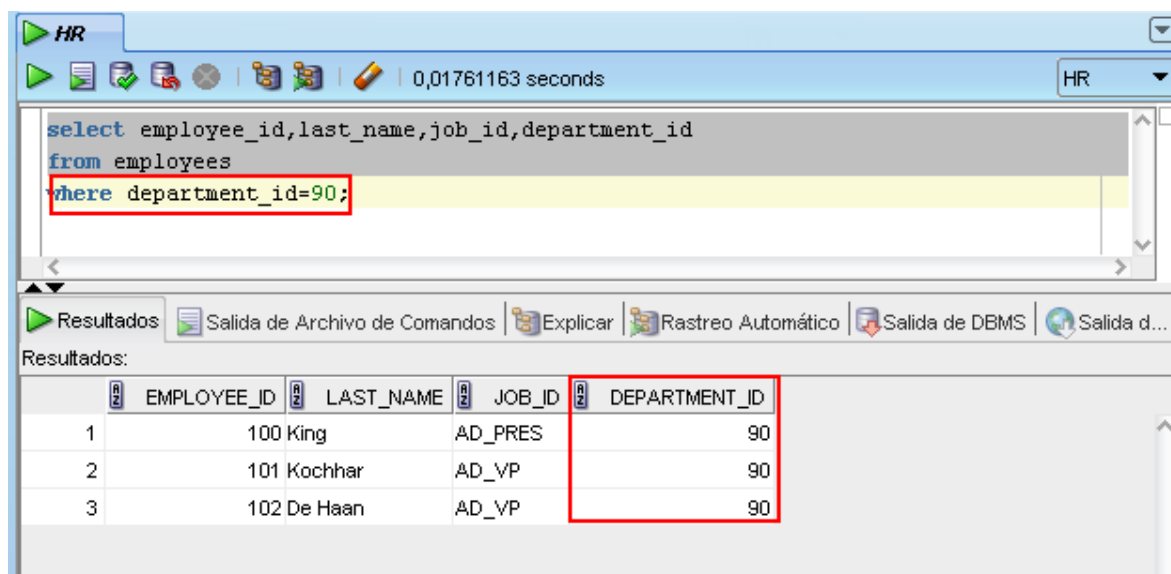
La cláusula WHERE puede comparar valores de columnas, valores literales, expresiones aritméticas o funciones. Consta de tres elementos:

- Nombre de columna
- Condición de comparación
- Nombre de columna, constante o lista de valores

Cuando Oracle procesa las tres cláusulas juntas, existe un orden de compilación que se debe conocer:

1. Ejecuta la cláusula FROM para capturar las tablas.
2. Ejecuta la cláusula WHERE, para establecer los filtros, de tal forma que el filtro está en función de las columnas de las tablas mencionadas anteriormente.
3. Ejecuta la cláusula SELECT. Por lo tanto, puede en el filtro existir una columna que no se tome en cuenta en el SELECT.

```
SELECT employee_id, last_name, job_id, department_id  
FROM   employees  
WHERE  department_id = 90 ;
```



The screenshot shows an Oracle SQL Developer window with a query editor at the top containing the following SQL statement:

```
select employee_id,last_name,job_id,department_id
from employees
where department_id=90;
```

Below the query editor, the 'Resultados' (Results) tab is active, displaying a table with the following data:

	EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
1	100	King	AD_PRES	90
2	101	Kochhar	AD_VP	90
3	102	De Haan	AD_VP	90

En este caso se muestran los ID, apellidos, cargos, y departamentos de todos aquellos empleados que pertenecen al departamento 90.

Manejo de las cadenas de caracteres y fechas en la condición.

- Las cadenas de caracteres y los valores de fechas se escriben entre comillas simples.
- Los valores de caracteres son sensibles a mayúsculas/minúsculas y los de fecha, al formato.
- El formato de fecha por defecto es DD-MON-RR. Este formato indica que se debe ingresar el dato ordenado por día, mes y año.

En la cláusula WHERE, las cadenas de caracteres y las fechas se deben escribir entre comillas simples ("), pero no las constantes numéricas.

Las búsquedas de caracteres son sensibles a mayúsculas/minúsculas.

```
SELECT last_name, job_id, department_id
FROM employees
WHERE last_name = 'Whalen';
```

Las bases de datos Oracle almacenan fechas en formato numérico interno, representando el siglo, el año, el mes, el día, las horas, los minutos y los segundos. El formato de fecha por defecto es DD-MON-RR.

Condiciones de comparación

Las condiciones de comparación se utilizan en condiciones que comparan una expresión con otro valor o expresión.

Se usan en la cláusula WHERE con el siguiente formato:

Sintaxis

... WHERE expresión **operador** valor

Ejemplo

```
... WHERE hire_date='01-JAN-95'
... WHERE salary>=6000
... WHERE last_name='Smith'
```

No se puede utilizar un **alias** en la cláusula WHERE, sino solo campos de la tabla.

Los operadores de comparación son:

Operador	Significado
=	Igual que
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
<>	No igual a

Estos operadores se pueden utilizar no solo con datos numéricos sino de fechas y caracteres.

```
SELECT last_name, salary
FROM employees
WHERE salary <= 3000;
```

OTROS OPERADORES DE COMPARACIÓN.

Operador	Significado
BETWEEN ... AND ...	Entre dos valores (ambos inclusive),
IN (set)	Coincide con cualquiera de una lista de valores
LIKE	Coincide con un patrón de caracteres
IS NULL	Es un valor nulo

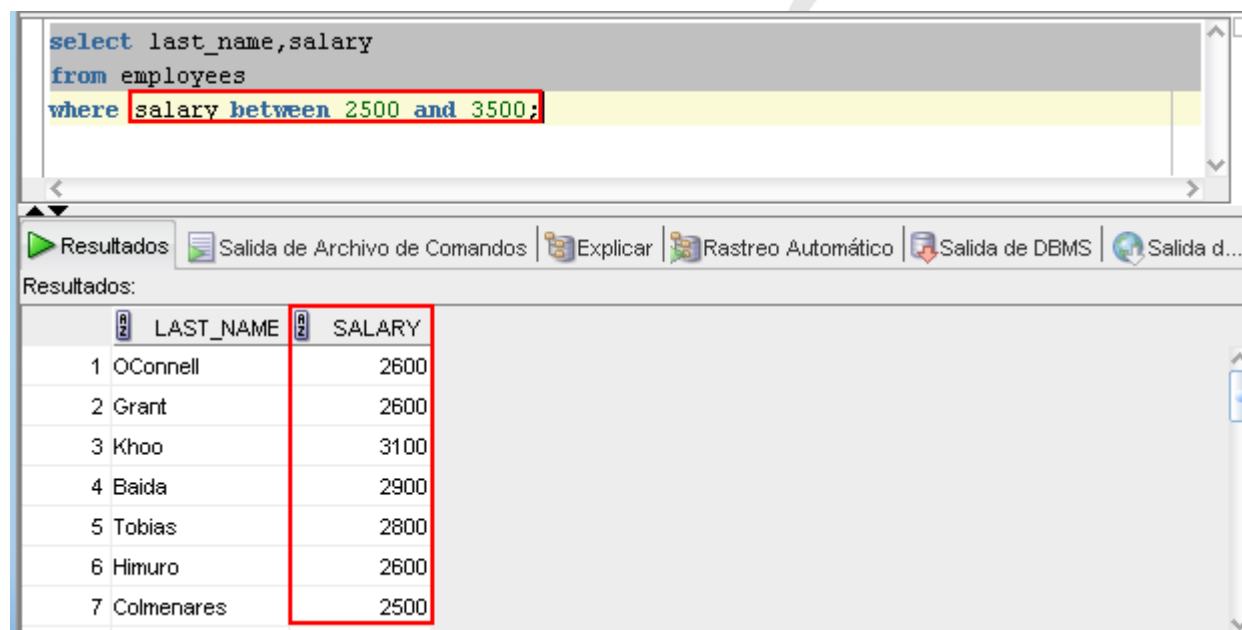
La Condición BETWEEN

Puede mostrar filas incluidas en un rango de valores utilizando la condición de rango BETWEEN. El rango que especifique contiene un límite inferior y uno superior.

La sentencia SELECT del ejemplo devuelve filas de la tabla *employees* para cualquier empleado cuyo salario esté entre 2500 y 3500.

Los valores especificados en la condición BETWEEN están incluidos. Siempre se debe especificar en primer lugar el límite inferior y luego el superior.

```
SELECT last_name, salary
FROM employees
WHERE salary BETWEEN 2500 AND 3500;
```



The screenshot shows a SQL query execution window. The query is: `select last_name,salary from employees where salary between 2500 and 3500;`. The results are displayed in a table with columns LAST_NAME and SALARY. The results are:

	LAST_NAME	SALARY
1	OConnell	2600
2	Grant	2600
3	Khoo	3100
4	Baida	2900
5	Tobias	2800
6	Himuro	2600
7	Colmenares	2500

La Condición IN

Para comprobar si hay valores en un conjunto (o lista) especificado de valores, utilice la condición IN. Esta condición también se conoce como condición de pertenencia. Es decir, los valores que devuelve la consulta es cualquiera de ellos que se encuentra enumerado en la lista.

El ejemplo muestra números de empleado, apellidos, salarios y números de empleado del jefe para todos los empleados cuyos números de empleado del director sean 100, 101 o 201.

```
SELECT employee_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IN (100, 101, 201);
```

Si se utilizan caracteres o fechas en la lista, deben escribirse entre comillas simples ('). Si son números no se utiliza esos caracteres.

La Condición LIKE

No siempre conocerá el valor exacto para buscar. Puede seleccionar filas que coincidan con un patrón de caracteres utilizando la condición LIKE.

La operación de coincidencia del patrón de caracteres se denomina búsqueda con comodines. Se pueden utilizar dos símbolos para construir la cadena de búsqueda.

La sentencia SELECT del ejemplo devuelve el nombre del empleado de la tabla employees para cualquier empleado cuyo nombre empiece por S. Observe que la S está en mayúsculas. No se devolverán nombres que empiecen por s minúsculas.

```
SELECT first_name
FROM employees
WHERE first_name LIKE 'S%';
```

La condición LIKE se puede utilizar como método abreviado de algunas comparaciones BETWEEN. El siguiente ejemplo muestra los apellidos y las fechas de contratación de todos los empleados contratados entre enero y diciembre de 1995:

```
SELECT first_name, hire_date
FROM employees
WHERE hire_date LIKE '%95';
```

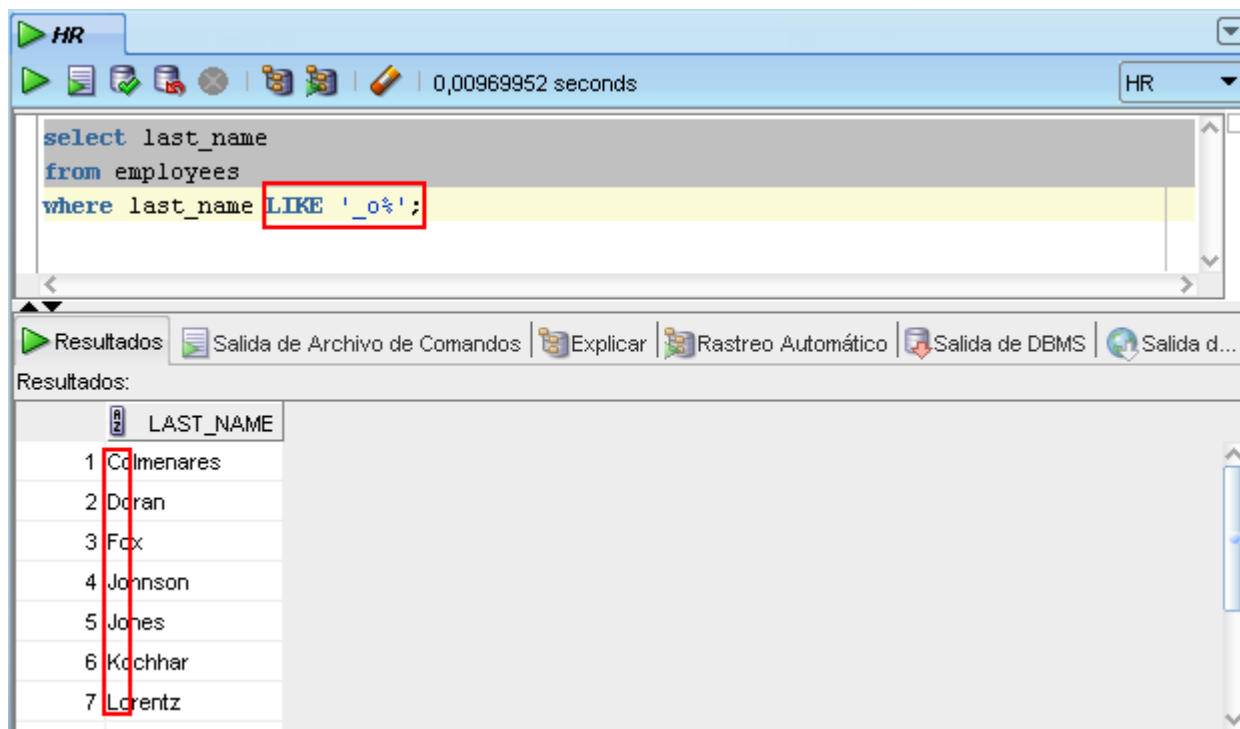
Las condiciones de búsqueda pueden contener caracteres literales o números, por eso es siempre oportuno el uso de los caracteres comodines:

% indica cero o muchos caracteres.

_ indica un carácter.

```
SELECT last_name
FROM employees
WHERE last_name LIKE '_o%';
```

Este ejemplo muestra aquellos empleados cuyos apellidos tienen en la segunda letra el carácter o minúscula.



COMBINACIÓN DE CARACTERES COMODÍN

Pero ¿qué pasa cuándo lo que se quiere buscar en la cadena son los símbolos % y _? En este caso se pueden utilizar en cualquier combinación con caracteres literales pero siempre y cuando se indique que esos caracteres dejan de ser comodines para dicha expresión. Esto se hace con la cláusula ESCAPE.

La opción ESCAPE

Si necesita una coincidencia exacta de los caracteres % y _ reales, utilice la opción ESCAPE. Esta opción especifica cuál es el carácter de escape. Si desea buscar cadenas que contengan 'SA_' (se está buscando el guión bajo que es un comodín), puede utilizar la siguiente sentencia SQL:

```
SELECT last_name, job_id
FROM employees
WHERE last_name LIKE '%SA\_%' ESCAPE '\';
```

Las Condiciones NULL

Las condiciones NULL incluyen la condición IS NULL y la condición IS NOT NULL.

La condición IS NULL comprueba si hay valores nulos. Un valor nulo significa que el valor no está disponible, no está asignado, es desconocido o no es aplicable. Por lo tanto, no puede comprobar con = (igual), pues un valor nulo no puede ser igual o distinto de ningún valor.

El ejemplo recupera los apellidos y los directores de todos los empleados que no tienen director.


```
SELECT last_name, manager_id
FROM employees
WHERE manager_id IS NULL;
```

Para otro ejemplo, para mostrar apellidos, identificador de cargo y comisiones para todos los empleados que NO tienen derecho a recibir una comisión, utilice la siguiente sentencia SQL:

```
SELECT last_name, job_id, commission_pct
FROM employees
WHERE commission_pct IS NULL;
```

CONDICIONES LÓGICAS

Las condiciones lógicas combinan el resultado de dos condiciones componentes para producir un resultado único basado en ellas o invierten el resultado de una única condición. Se devuelve una fila sólo si el resultado global de la condición es verdadero. En SQL están disponibles tres operadores lógicos:

Operador	Significado
AND	Devuelve TRUE si las <i>dos</i> condiciones componentes son verdaderas
OR	Devuelve TRUE si <i>alguna</i> de las condiciones componentes es verdadera
NOT	Devuelve TRUE si la siguiente condición es falsa

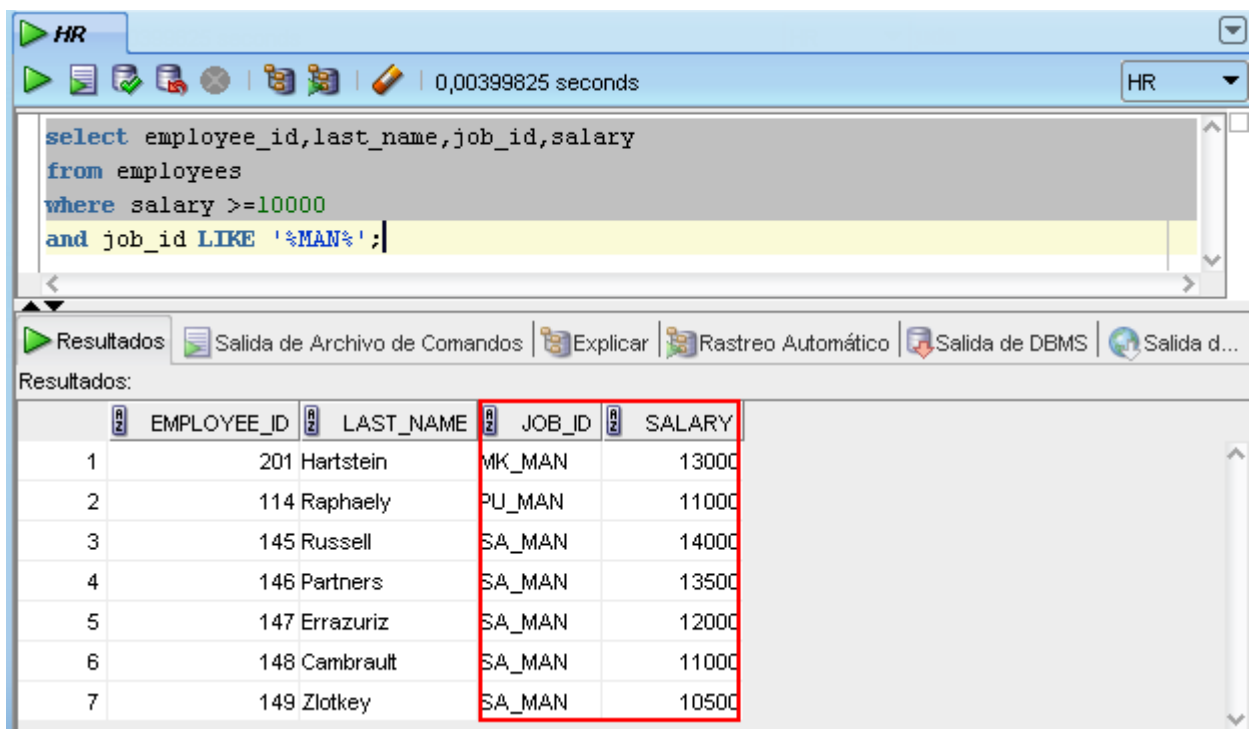
Todos los ejemplos mostrados hasta ahora han especificado solamente una condición en la cláusula WHERE. Una sentencia SQL puede utilizar varias condiciones en una cláusula WHERE utilizando los operadores AND y OR. Estos operadores se van concatenando para validar cada pareja de expresiones.

El Operador AND

En el ejemplo, las dos condiciones deben ser verdaderas para que se seleccione un registro. Por lo tanto, solamente se seleccionarán los empleados cuyo cargo contenga la cadena MAN y ganen más de 10000.

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >=10000
AND job_id LIKE '%MAN%';
```


Todas las búsquedas de caracteres son sensibles a mayúsculas/minúsculas. No se devolverán filas si MAN no está en mayúsculas. Las cadenas de caracteres se deben escribir entre comillas.



The screenshot shows a SQL query execution window with the following query:

```
select employee_id,last_name,job_id,salary
from employees
where salary >=10000
and job_id LIKE '%MAN%';
```

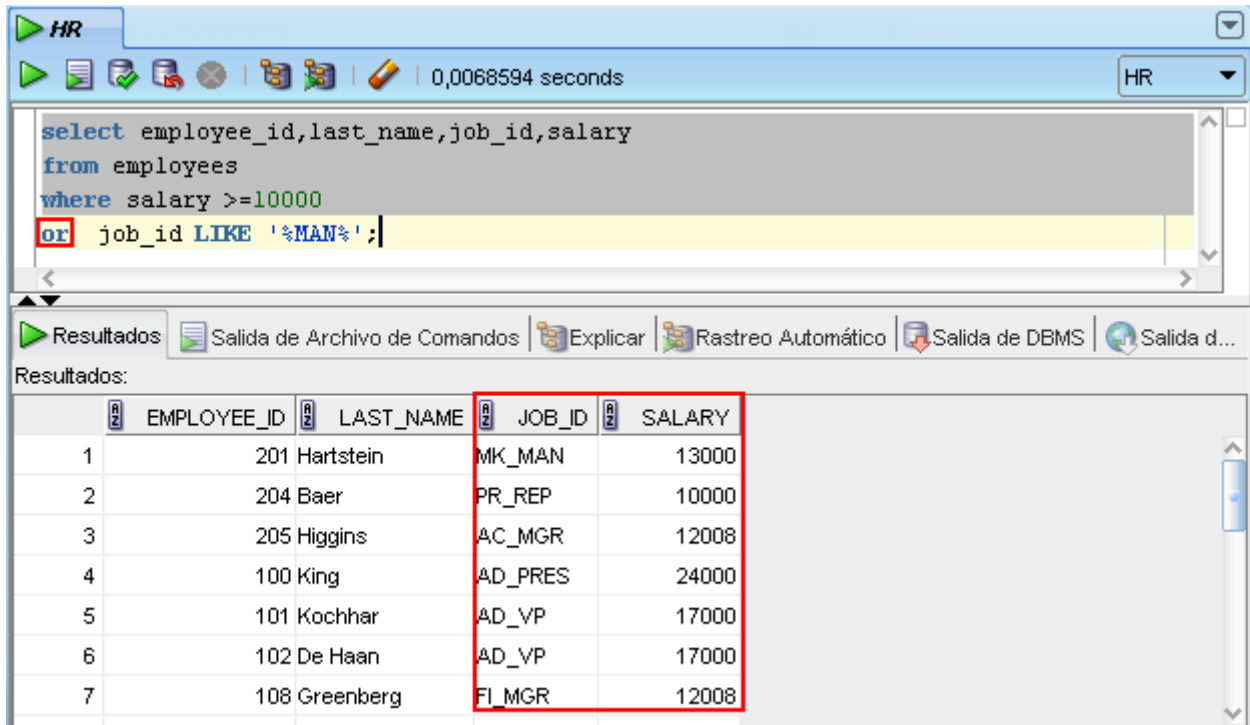
The results are displayed in a table with the following columns: EMPLOYEE_ID, LAST_NAME, JOB_ID, and SALARY. The results are as follows:

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	201	Hartstein	MK_MAN	13000
2	114	Raphaely	PU_MAN	11000
3	145	Russell	SA_MAN	14000
4	146	Partners	SA_MAN	13500
5	147	Errazuriz	SA_MAN	12000
6	148	Cambraut	SA_MAN	11000
7	149	Zlotkey	SA_MAN	10500

El Operador OR

En el ejemplo, cualquiera de las dos condiciones puede ser verdadera para que se seleccione un registro. Por lo tanto, se seleccionará cualquier empleado cuyo identificador de cargo contenga MAN o gane más de \$10.000.

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary >= 10000
OR job_id LIKE '%MAN%';
```



The screenshot shows a SQL query execution window with the following query:

```
select employee_id,last_name,job_id,salary
from employees
where salary >=10000
or job_id LIKE '%MAN%';
```

The results are displayed in a table with the following columns: EMPLOYEE_ID, LAST_NAME, JOB_ID, and SALARY. The results are as follows:

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
201	Hartstein	MK_MAN	13000
204	Baer	PR_REP	10000
205	Higgins	AC_MGR	12008
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
108	Greenberg	FI_MGR	12008

El Operador NOT

El ejemplo muestra el apellido y el identificador de cargo de todos los empleados cuyos identificadores de cargo no sean IT_PROG, ST_CLERK o SA_REP.

```
SELECT last_name, job_id
FROM employees
WHERE job_id
NOT IN ( 'IT_PROG', 'ST_CLERK', 'SA_REP' );
```

El operador NOT también se puede utilizar con otros operadores SQL, como BETWEEN, LIKE y NULL.

```
... WHERE job_id NOT IN ('AC_ACCOUNT','AD_VP')
... WHERE salary NOT BETWEEN 10000 AND 15000
... WHERE last_name NOT LIKE '%A%'
... WHERE commission_pct IS NOT NULL
```

REGLAS DE PRIORIDAD

Las reglas de prioridad determinan el orden en el que se evalúan y se calculan las expresiones. La tabla enumera el orden de prioridad por defecto. Puede sustituir el orden por defecto escribiendo entre paréntesis las expresiones que desee calcular en primer lugar. Es importante al escribir sentencias de condición concatenadas, el

verificar que realmente el orden en el que van a ser evaluadas es el que se requiere, de otra forma el resultado será distinto.

Orden de Evaluación	Operador
1	Operadores aritméticos
2	Operador de concatenación
3	Condiciones de comparación
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	condición lógica NOT
7	condición lógica AND
8	condición lógica OR

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = 'SA_REP'
OR job_id = 'AD_PRES'
AND salary > 15000;
```

En este caso, primero se evalúa el AND, y luego el OR.

```
SELECT last_name, job_id, salary
FROM employees
WHERE (job_id = 'SA_REP'
OR job_id = 'AD_PRES')
AND salary > 15000;
```

En este caso, al usar los paréntesis, se evalúa primero el OR, y luego el AND.

La Cláusula ORDER BY

El orden de filas devuelto por una consulta no está definido.

Se puede utilizar la cláusula ORDER BY para ordenar las filas. Si lo hace, debe ser la última de la sentencia SQL. Puede especificar una expresión o alias o posición de columna como condición de orden.

Recordemos que cuando se guarda los datos o bloques de datos de una tabla, estos no se guardan en orden, sino se apilan en bloques que pueden estar continuos o discontinuos. La bidimensionalidad no existe físicamente. Por eso que cuando se recupera información, esta es mostrada por Oracle en cualquier orden.

Cuando queremos ver la información ordenada por uno o más campos hay que indicarle la cláusula ORDER BY. Este ordenamiento no es físico, sino lógico, se realiza en la memoria del Oracle y solo para esa sentencia.

Sintaxis

```
SELECT          expr
FROM            table
[WHERE          condition(s)]
[ORDER BY {column, expr} [ASC|DESC]];
```

En la sintaxis:

ORDER BY	especifica el orden en el que se muestran las filas recuperadas
ASC	ordena las filas en orden ascendente (es el orden por defecto)
DESC	ordena las filas en orden descendente

Si no se utiliza la cláusula ORDER BY, el orden no está definido y es posible que Oracle Server no recupere las filas en el mismo orden dos veces para la misma consulta. Utilice esta cláusula para mostrar las filas en un orden específico.

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date ;
```

Ordenación por Defecto de Datos

El orden por defecto es ascendente:

- Los valores numéricos se muestran con los valores más bajos primero; por ejemplo, 1 a 999.
- Los valores de fecha se muestran con la fecha anterior primero; por ejemplo, 01-ENE-92 antes de 01-ENE-95.
- Los valores de caracteres se muestran en orden alfabético; por ejemplo, A en primer lugar y Z en último.
- Los valores nulos se muestran los últimos para las secuencias ascendentes y los primeros para las descendentes.

Reversión al Orden por Defecto

Para revertir el orden en el que se muestran las filas, especifique la palabra clave DESC después del nombre de columna en la cláusula ORDER BY. El ejemplo de la transparencia ordena el resultado de manera descendente por el campo fecha.

```
SELECT last_name, job_id, department_id, hire_date
FROM employees
ORDER BY hire_date DESC ;
```

Ordenación según Alias de Columna

Cuando una sentencia SQL tiene la cláusula ORDER BY, esta siempre se ejecuta al último:

1. Primero se ejecuta el FROM.
2. Luego el WHERE.
3. Luego el SELECT, y cuando ya todos los datos están cargados.
4. El ORDER BY. Esto quiere decir que cuando se activa esta sentencia ya se han generado los campos calculados, y por lo tanto estos se pueden utilizar en el ordenamiento.

Puede utilizar un alias de columna en la cláusula ORDER BY. El ejemplo de la transparencia ordena los datos según el salario anual.

```
SELECT employee_id, last_name, salary*12 Anual
FROM employees
ORDER BY Anual;
```

Ordenación según Múltiples Columnas

Puede ordenar el resultado de la consulta según más de una columna. El límite de ordenación es el número de columnas de la tabla dada.

En la cláusula ORDER BY, especifique las columnas y separe los nombres de columna mediante comas. Si desea revertir el orden de una columna, especifique DESC después de su nombre. También puede ordenar según columnas no incluidas en la cláusula SELECT.

```
SELECT last_name, department_id, salary
FROM employees
ORDER BY department_id, salary DESC;
```

En este ejemplo, se muestra el apellido, ID de departamento, sueldo de los empleados ordenados primero por el ID del departamento de forma ascendente, y luego por el salario en forma descendente.

Otra forma de ordenar una consulta es utilizando el número de orden de la columna y no el nombre de la misma. La consulta anterior se escribiría de la siguiente forma:

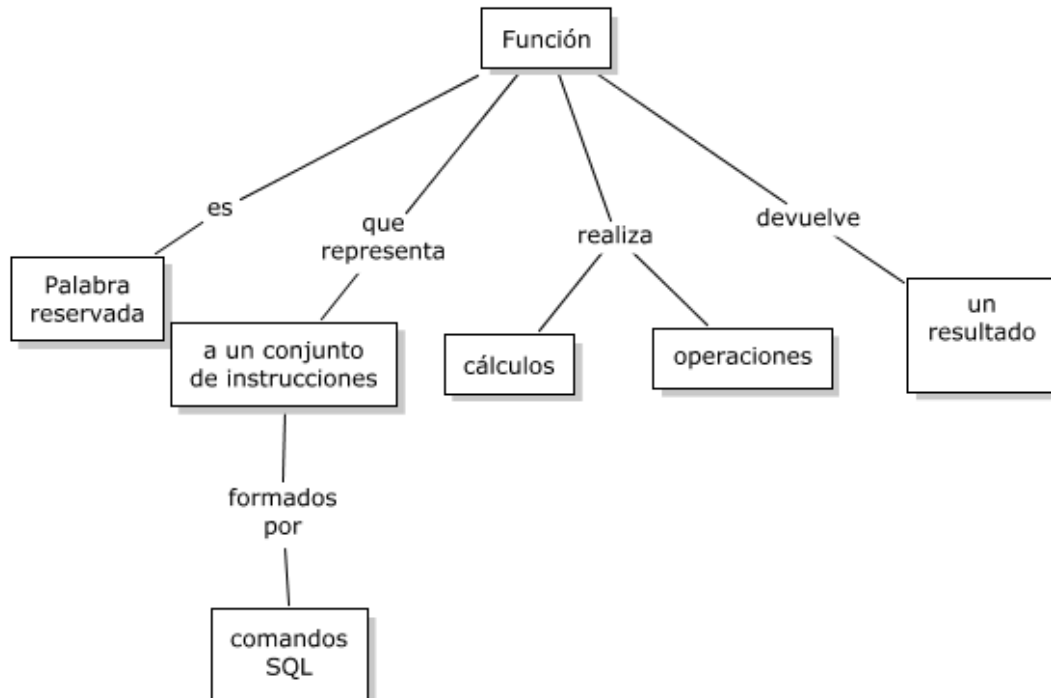
```
SELECT last_name, department_id, salary
FROM employees
ORDER BY 2, 3 DESC;
```

LAS FUNCIONES SQL DE ORACLE 11G

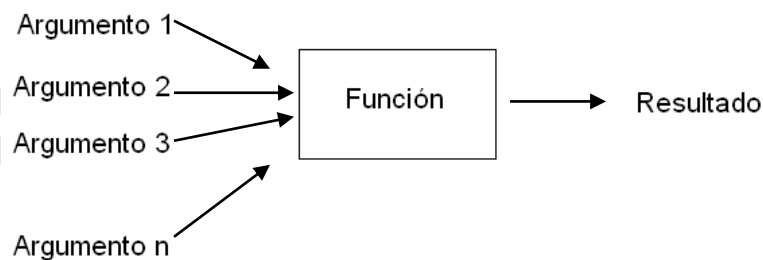
Las funciones son unas funcionalidades potentes de SQL y se pueden utilizar para lo siguiente:

- Realizar cálculos sobre datos
- Modificar elementos de datos individuales

- Manipular el resultado para grupos de filas
- Formatear fechas y números para su visualización
- Convertir tipos de dato de columna



Así, se podría definir sencillamente a las funciones como un programa que contiene un conjunto de instrucciones que realizan un cálculo y devuelven un solo valor.

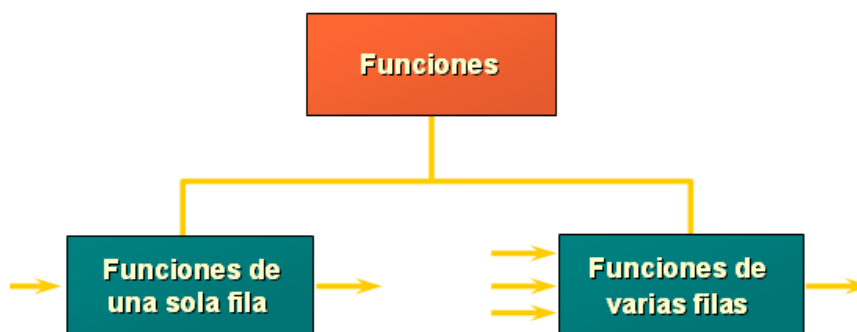


Las funciones SQL a veces toman argumentos y siempre devuelven un valor. Un argumento es el valor que ingresa desde fuera hacia la función y que le proporcionan elementos para ser calculados.

Existen igualmente funciones que no tienen argumentos.

Hay dos tipos distintos de funciones:

- Funciones de una sola fila
- Funciones de varias filas



Funciones de una Sola Fila

Estas funciones solamente operan en una fila y devuelven un resultado por fila. Hay distintos tipos de funciones de una sola fila. Esta lección cubre las siguientes:

- Carácter
- Número
- Fecha
- Conversión

Funciones de Varias Filas

Las funciones pueden manipular grupos de filas para proporcionar un resultado por cada uno de ellos. Estas funciones se conocen como funciones de grupo y se tratan en una lección posterior.

FUNCIONES DE UNA SOLA FILA

Las funciones de una sola fila se utilizan para manipular elementos de datos. Aceptan uno o varios argumentos y devuelven un valor para cada fila devuelta por la consulta. El argumento puede ser uno de los siguientes:

- Valor de variable
- Nombre de columna
- Expresión

Las funcionalidades de las funciones de una sola fila incluyen:

- Actuar sobre cada fila devuelta en la consulta
- Devolver un resultado por fila
- Devolver posiblemente un valor de datos de un tipo diferente al de referencia
- Esperar posiblemente uno o varios argumentos
- Se pueden utilizar en cláusulas SELECT, WHERE y ORDER BY; se pueden anidar varias funciones al mismo tiempo.

En la sintaxis:

function_name	es el nombre de la función.
arg1, arg2	es cualquier argumento que debe utilizar la función.
	Puede venir representado por un nombre de columna o una expresión.

FUNCIONES DE MANIPULACIÓN DE MAYÚSCULAS/MINÚSCULAS

LOWER, UPPER e INITCAP son las tres funciones de conversión de mayúsculas/minúsculas.

- **LOWER:** Convierte cadenas de caracteres en mayúsculas o mezclados a minúsculas.

```
SQL>
SQL>
SQL> SELECT LOWER(last_name)
  2 FROM EMPLOYEES
  3 WHERE employee_id=205;

LOWER(LAST_NAME)
-----
higgins
```

- **UPPER:** Convierte cadenas de caracteres en minúsculas o mezclados a mayúsculas.

```
SQL>
SQL>
SQL> SELECT UPPER(last_name)
  2 FROM EMPLOYEES
  3 WHERE employee_id=205;

UPPER(LAST_NAME)
-----
HIGGINS
```

- **INITCAP:** Convierte la primera letra de cada palabra a mayúsculas y las restantes letras a minúsculas.

```
SQL>
SQL>
SQL> ED
Wrote file afiedt.buf

  1 SELECT INITCAP(last_name)
  2 FROM EMPLOYEES
  3* WHERE employee_id=205
SQL> /

INITCAP(LAST_NAME)
-----
Higgins
```

Las funciones no solo se usan en la sentencia SELECT, como habíamos dicho en párrafos anteriores sino también en la CLAUSULA where.


```
SQL> ed
Wrote file afiedt.buf

 1 SELECT INITCAP<last_name>, UPPER <first_name>, LOWER <job_id>
 2 FROM EMPLOYEES
 3* WHERE UPPER<last_name>='KING'
SQL> /
```

INITCAP<LAST_NAME>	UPPER<FIRST_NAME>	LOWER<JOB_
King	STEVEN	ad_pres
King	JANETTE	sa_rep

Funciones de Manipulación de Caracteres

CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD y TRIM son las funciones de manipulación de caracteres que se tratan en esta lección.

- **CONCAT:** Une valores (con esta función está limitado a utilizar dos parámetros.)

```
SQL>
SQL>
SQL> SELECT CONCAT<LAST_NAME,FIRST_NAME>
 2 FROM EMPLOYEES
 3 WHERE EMPLOYEE_ID=100;
```

CONCAT<LAST_NAME,FIRST_NAME>
KingSteven

- **SUBSTR:** Extrae una cadena de una longitud determinada.

```
SQL>
SQL>
SQL> SELECT LAST_NAME,SUBSTR<LAST_NAME,1,3>
 2 FROM EMPLOYEES
 3 WHERE EMPLOYEE_ID=100;
```

LAST_NAME	SUBSTR<LAST_
King	Kin

- **LENGTH:** Muestra la longitud de una cadena como valor numérico.

```
SQL>
SQL>
SQL> SELECT LAST_NAME,LENGTH<LAST_NAME>
 2 FROM EMPLOYEES
 3 WHERE EMPLOYEE_ID=100;
```

LAST_NAME	LENGTH<LAST_NAME>
King	4

- **INSTR:** Busca la posición numérica de un carácter especificado.

```
SQL>
SQL>
SQL> SELECT LAST_NAME,INSTR<LAST_NAME,'n'>
 2 FROM EMPLOYEES
 3 WHERE EMPLOYEE_ID=100;
```

LAST_NAME	INSTR<LAST_NAME,'N'>
King	3

- **LPAD:** Rellena el valor de caracteres justificado a la derecha.
- **RPAD:** Rellena el valor de caracteres justificado a la izquierda.

```
SQL> SELECT LPAD(SALARY,7,'+') ,RPAD(SALARY,7,'*')
2 FROM EMPLOYEES
3 WHERE EMPLOYEE_ID=100;
```

LPAD(SALARY,7,'+')	RPAD(SALARY,7,'*')
++24000	24000**

- **TRIM:** Recorta caracteres iniciales o finales (o ambos) de una cadena de caracteres. (Si trim_character o trim_source es un literal de carácter, debe escribirlo entre comillas simples.)

```
SQL>
SQL> SELECT LAST_NAME, TRIM<leading 'K' from last_name>
2 FROM EMPLOYEES
3 WHERE EMPLOYEE_ID=100;
```

LAST_NAME	TRIM<LEADING 'K' FROM LAST_N
King	ing

```
SQL> SELECT LAST_NAME, TRIM<trailing 'g' from last_name>
2 FROM EMPLOYEES
3 WHERE EMPLOYEE_ID=100;
```

LAST_NAME	TRIM<TRAILING 'G' FROM LAST_
King	Kin

```
SQL> SELECT LAST_NAME, TRIM<both 'K' from last_name>
2 FROM EMPLOYEES
3 WHERE EMPLOYEE_ID=100;
```

LAST_NAME	TRIM<BOTH 'K' FROM LAST_NAME
King	ing

Todas estas funciones en realidad se pueden combinar dentro de una sentencia SQL.

```
SELECT employee id, CONCAT(first name, last_name) NAME,
job_id, LENGTH (last name),
INSTR(last_name, 'a') "Contains 'a'?"
FROM employees
WHERE SUBSTR(job_id, 4) = 'REP' ;
```

Funciones Numéricas

Las funciones numéricas aceptan entradas numéricas y devuelven valores numéricos. Esta sección describe algunas de las funciones numéricas.

La tabla DUAL. La tabla DUAL es una tabla virtual que no contiene datos específicos sino simplemente sirve para cuando se quieren hacer consultas con datos literales que no son extraídos de ninguna tabla.

ROUND: Redondea el valor a los decimales especificados.

```
SQL>
SQL>
SQL> SELECT ROUND(45.678,2) FROM DUAL;
ROUND(45.678,2)
-----
          45,68

SQL> SELECT ROUND(45.678,1) FROM DUAL;
ROUND(45.678,1)
-----
          45,7

SQL> SELECT ROUND(45.678,-1) FROM DUAL;
ROUND(45.678,-1)
-----
          50
```

TRUNC: Trunca el valor a los decimales especificados.

```
SQL>
SQL>
SQL> SELECT TRUNC(45.678,2) FROM DUAL;
TRUNC(45.678,2)
-----
          45,67
```

MOD: Devuelve el resto de la división.

```
SQL>
SQL> SELECT MOD(100,3) FROM DUAL;
MOD(100,3)
-----
          1
```

ABS: Calcula el valor absoluto de n.

```
SQL>
SQL>
SQL> SELECT ABS(-2) FROM DUAL;
ABS(-2)
-----
          2
```

CEIL: Calcula el menor número entero mayor o igual que n.

FLOOR: Calcula el mayor número entero menor o igual que n.

POWER: Devuelve m elevado a la n potencia, n debe ser entero

```
SQL>
SQL>
SQL> SELECT CEIL<16.7> FROM DUAL;

CEIL<16.7>
-----
         17

SQL> SELECT FLOOR<16.7> FROM DUAL;

FLOOR<16.7>
-----
         16

SQL> SELECT POWER<3,2> FROM DUAL;

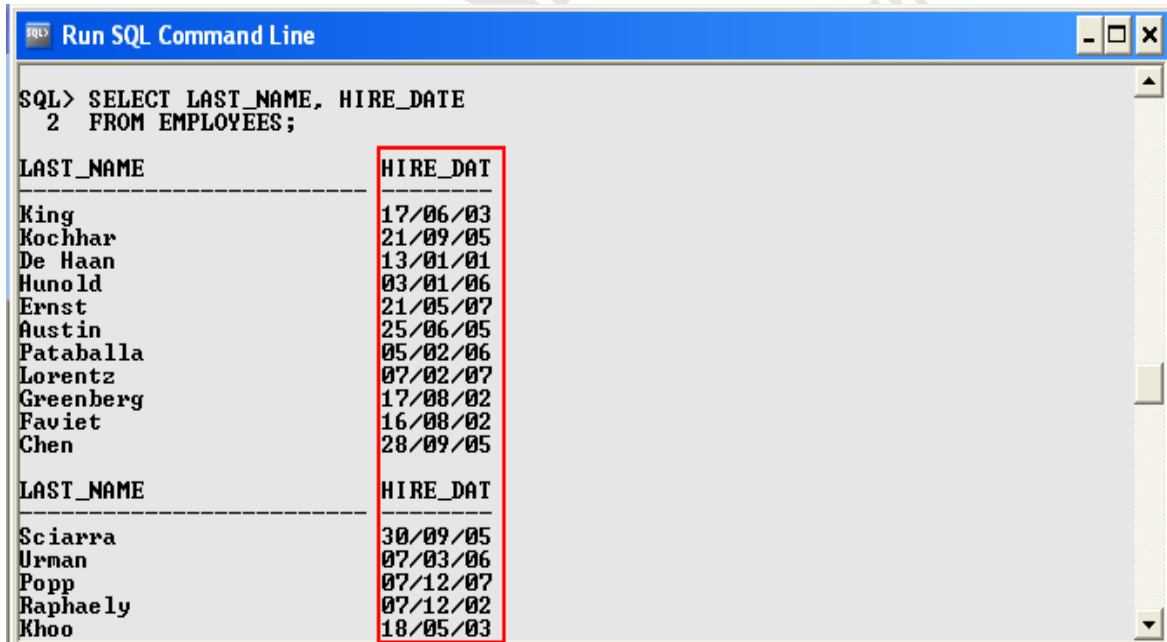
POWER<3,2>
-----
          9
```

Formato de Fechas de Oracle

La base de datos Oracle almacena fechas en un formato numérico interno, representando el siglo, el año, el mes, el día, las horas, los minutos y los segundos.

El formato de entrada y visualización por defecto de cualquier fecha es DD-MON-RR. Las fechas válidas para Oracle están comprendidas entre el 1 de enero de 4712 a.C. y el 31 de diciembre de 9999 d.C.

Así podemos darnos cuenta que las fecha son un tipo de dato especial, que se almacenan de forma numérica, pero se visualiza de forma de carácter.



The screenshot shows a window titled 'Run SQL Command Line' with the following content:

```
SQL> SELECT LAST_NAME, HIRE_DATE
2 FROM EMPLOYEES;
```

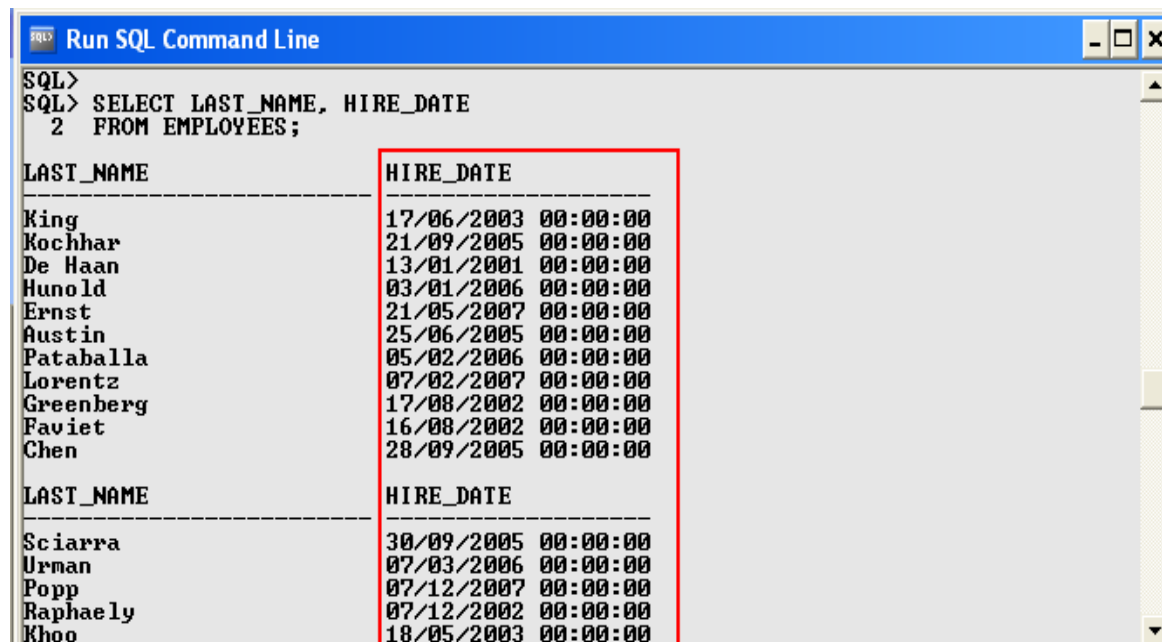
LAST_NAME	HIRE_DATE
King	17/06/03
Kochhar	21/09/05
De Haan	13/01/01
Hunold	03/01/06
Ernst	21/05/07
Austin	25/06/05
Pataballa	05/02/06
Lorentz	07/02/07
Greenberg	17/08/02
Faviet	16/08/02
Chen	28/09/05
Sciarra	30/09/05
Urman	07/03/06
Popp	07/12/07
Raphaely	07/12/02
Khoo	18/05/03

Esto implica una conversión interna que se controla con el parámetro NLS_DATE_FORMAT.

```
SQL>
SQL>
SQL> ALTER SESSION SET NLS_DATE_FORMAT='DD/MM/YYYY HH24:MI:SS';

Session altered.
```

De tal forma que ahora se podría visualizar:



LAST_NAME	HIRE_DATE
King	17/06/2003 00:00:00
Kochhar	21/09/2005 00:00:00
De Haan	13/01/2001 00:00:00
Hunold	03/01/2006 00:00:00
Ernst	21/05/2007 00:00:00
Austin	25/06/2005 00:00:00
Pataballa	05/02/2006 00:00:00
Lorentz	07/02/2007 00:00:00
Greenberg	17/08/2002 00:00:00
Faviet	16/08/2002 00:00:00
Chen	28/09/2005 00:00:00
Sciarra	30/09/2005 00:00:00
Urman	07/03/2006 00:00:00
Popp	07/12/2007 00:00:00
Raphaely	07/12/2002 00:00:00
Khoo	18/05/2003 00:00:00

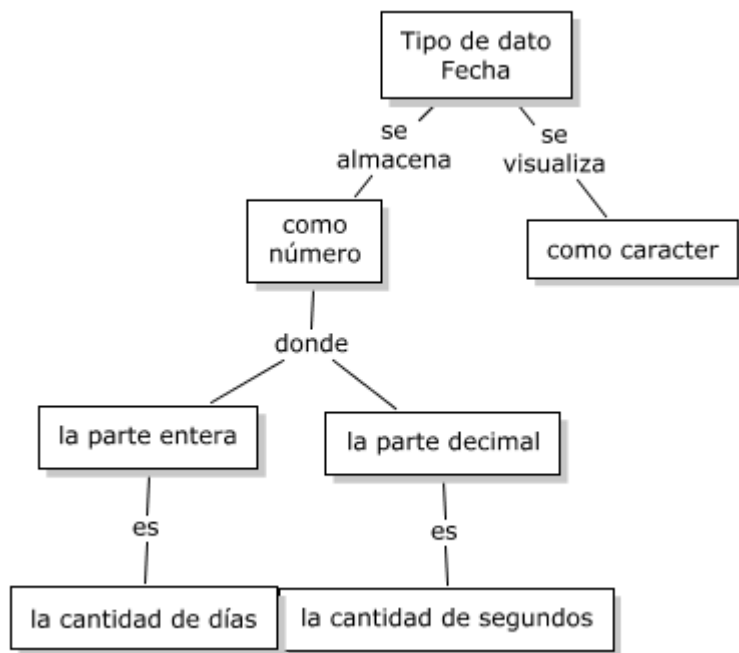
Los datos se almacenan internamente como se indica a continuación:

CENTURY YEAR MONTH DAY HOUR MINUTE SECOND

Siglos y el Año 2000

Oracle Server cumple con el año 2000. Al insertar en una tabla un registro con una columna de fecha, la información de siglo se recoge de la función SYSDATE. Sin embargo, cuando se muestra la columna de fecha en pantalla, el componente siglo no aparece por defecto.

El tipo de dato DATE siempre almacena la información de año internamente como número de cuatro dígitos: dos dígitos para el siglo y otros dos para el año. Por ejemplo, la base de datos Oracle almacena el año como 1996 ó 2001 y no simplemente como 96 ó 01.



La Función SYSDATE

SYSDATE es una función de fecha que devuelve la fecha y la hora actuales del servidor de base de datos. Puede utilizar SYSDATE de la misma forma que utilizaría cualquier otro nombre de columna. Por ejemplo, puede mostrar la fecha actual seleccionando SYSDATE desde una tabla. Lo habitual es seleccionar SYSDATE desde una tabla ficticia llamada DUAL.

Ejemplo

Visualice la fecha actual utilizando la tabla DUAL.

```

SQL>
SQL>
SQL> SELECT SYSDATE FROM DUAL;

SYSDATE
-----
23/04/2011 10:46:28
  
```

Aritmética con Fechas

Partiendo de la premisa que las fechas son números, y que cada número entero es la cantidad de días transcurridos y que cada decimal es la cantidad de segundos transcurridos en un día, entonces podemos hablar de una aritmética de fechas en la cual se calcula todo en base a la unidad DIA.

El ejemplo de la transparencia que se muestra a continuación, muestra el apellido y el número de semanas de empleo de todos los trabajadores del departamento 90. Resta la fecha en la que se contrató al empleado de la fecha actual (SYSDATE) y divide el resultado por 7 para calcular el número de semanas que lleva empleado un trabajador.

Nota: SYSDATE es una función SQL que devuelve la fecha y la hora actuales. Es posible que los resultados que obtenga difieran del ejemplo.

Si a una fecha se le resta otra más reciente, la diferencia es un número negativo.

```
SQL>
SQL>
SQL>
SQL> SELECT LAST_NAME, <SYSDATE-HIRE_DATE>/7 AS SEMANAS
2 FROM EMPLOYEES
3 WHERE DEPARTMENT_ID=90;
```

LAST_NAME	SEMANAS
King	409,63589
Kochhar	291,493032
De Haan	536,064461

También se pueden agregar días a una fecha mediante sumas:

```
SQL>
SQL>
SQL> SELECT LAST_NAME, HIRE_DATE+90 AS NUEVA_FECHA
2 FROM EMPLOYEES
3 WHERE DEPARTMENT_ID=90;
```

LAST_NAME	NUEVA_FECHA
King	15/09/2003 00:00:00
Kochhar	20/12/2005 00:00:00
De Haan	13/04/2001 00:00:00

Funciones de Fecha

Las funciones de fecha operan sobre fechas de Oracle. Todas las funciones de fecha devuelven un valor del tipo de dato DATE excepto MONTHS_BETWEEN, que devuelve un valor numérico.

Función	Descripción
MONTHS_BETWEEN	Número de meses entre dos fechas
ADD_MONTHS	Suma meses de calendario a una fecha
NEXT_DAY	Siguiente día de la fecha especificada
LAST_DAY	Último día del mes
ROUND	Redondea la fecha
TRUNC	Trunca la fecha

MONTHS_BETWEEN(date1, date2): Busca el número de meses entre date1 y date2. El resultado puede ser positivo o negativo. Si date1 es posterior a date2, el resultado es positivo; si date1 es anterior a date2, el resultado es negativo. La parte no entera del resultado representa una porción del mes.


```
SQL>
SQL> SELECT MONTHS_BETWEEN<SYSDATE,HIRE_DATE>
2 FROM EMPLOYEES;

MONTHS_BETWEEN<SYSDATE,HIRE_DATE>
-----
94,2082385
67,0792062
123,337271
```

ADD_MONTHS(date, n): Suma n meses de calendario a date. El valor de n debe ser entero y puede ser negativo.

```
SQL>
SQL> SELECT ADD_MONTHS<SYSDATE,3> FROM DUAL;

ADD_MONTHS<SYSDATE,
-----
23/07/2011 10:57:05
```

NEXT_DAY(date, 'char'): Busca la fecha del siguiente día de la semana especificado ('char') posterior a date. El valor de char puede ser un número que represente un día o una cadena de caracteres.

```
SQL>
SQL>
SQL> SELECT NEXT_DAY<SYSDATE,'VIERNES'> FROM DUAL;

NEXT_DAY<SYSDATE,'U
-----
29/04/2011 10:57:57
```

LAST_DAY(date): Busca la fecha del último día del mes en el que está date.

```
SQL>
SQL> SELECT LAST_DAY<SYSDATE> FROM DUAL;

LAST_DAY<SYSDATE>
-----
30/04/2011 10:58:38
```

ROUND(date[, 'fmt']): Devuelve date redondeado a la unidad especificada por el modelo de formato fmt. Si el modelo de formato fmt está omitido, date se redondea al día más próximo.

```
SQL> SELECT SYSDATE, ROUND<SYSDATE, 'YEAR'> FROM DUAL;

SYSDATE          ROUND<SYSDATE, 'YEAR
-----
23/04/2011 11:00:21 01/01/2011 00:00:00

SQL> SELECT SYSDATE, ROUND<SYSDATE, 'MONTH'> FROM DUAL;

SYSDATE          ROUND<SYSDATE, 'MONT
-----
23/04/2011 11:00:28 01/05/2011 00:00:00
```

TRUNC(date[, 'fmt']): Devuelve date con la parte de hora del día truncada a la unidad especificada por el modelo de formato fmt. Si el modelo de formato fmt está omitido, date se trunca al día más próximo.

```
SQL>
SQL>
SQL> SELECT SYSDATE, TRUNC<SYSDATE, 'YEAR'> FROM DUAL;

SYSDATE                TRUNC<SYSDATE, 'YEAR'
-----
23/04/2011 11:02:29 01/01/2011 00:00:00

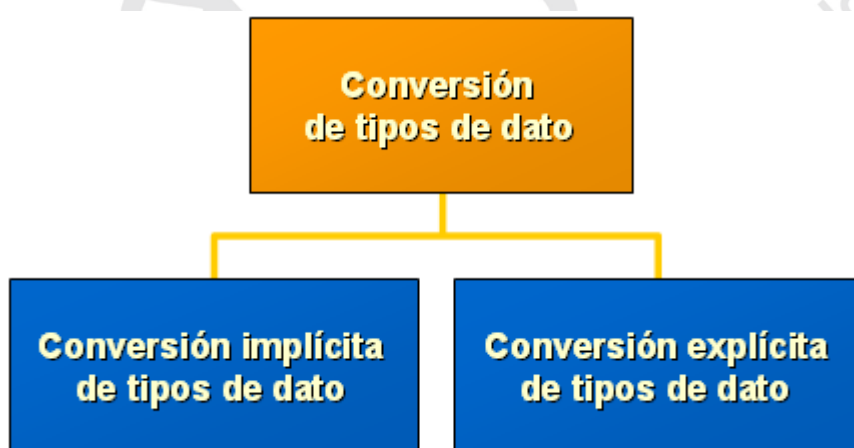
SQL> SELECT SYSDATE, TRUNC<SYSDATE, 'MONTH'> FROM DUAL;

SYSDATE                TRUNC<SYSDATE, 'MONT
-----
23/04/2011 11:02:35 01/04/2011 00:00:00
```

FUNCIONES DE CONVERSIÓN

Además de los tipos de dato Oracle, las columnas de las tablas de las bases de datos Oracle11g se pueden definir utilizando tipos de dato ANSI, DB2 y SQL/DS. Sin embargo, Oracle Server convierte internamente estos tipos de dato en tipos de dato Oracle.

En algunos casos, Oracle Server utiliza datos de un tipo donde espera datos de otro tipo distinto. Cuando esto ocurre, Oracle Server puede convertir automáticamente los datos al tipo de dato esperado. Esta conversión la puede hacer Oracle Server implícitamente o el usuario explícitamente.



Las conversiones implícitas de tipos de dato se realizan de acuerdo con las reglas explicadas en las dos transparencias siguientes.

Las conversiones explícitas de tipos de dato se realizan utilizando las funciones de

conversión, que convierten un valor de un tipo de dato en otro. Generalmente, la forma de los nombres de función sigue la convención data type TO data type. El primer tipo de dato es el de entrada y el último, el de salida.

Nota: Aunque la conversión implícita de tipos de dato está disponible, se recomienda que haga conversiones explícitas de tipos de dato para asegurar la fiabilidad de las sentencias SQL.

Las conversiones implícitas se realizan automáticamente y se pueden dar en dos niveles:

- Cuando se asignan datos. Es decir, en una cláusula de INSERT o UPDATE.

- Cuando se evalúan datos. Es decir, en el WHERE de una cláusula SELECT.

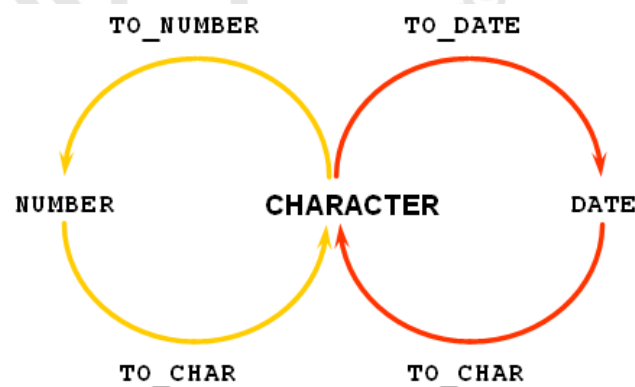
Cuando se asignan datos se puede convertir automáticamente en los siguientes casos:

De	A
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

Por otro lado, si se trata de evaluaciones de expresiones, se puede convertir automáticamente:

De	A
VARCHAR2 or CHAR	NUMBER
VARCHAR2 or CHAR	DATE

Las funciones de conversión se utilizan para los cambios de datos explícitos, y pueden ser las siguientes funciones:



Visualización de una Fecha en un Formato Específico

Anteriormente, todos los valores de fecha de Oracle se mostraban en formato DD-MON-YY. Puede utilizar la función TO_CHAR para convertir una fecha de este formato por defecto al que especifique.

El modelo de formato se debe escribir entre comillas sencillas y es sensible a mayúsculas/minúsculas.

- El modelo de formato puede incluir cualquier elemento de formato de fecha válido. Asegúrese de separar el valor de fecha del modelo de formato mediante una coma.
- Los nombres de los días y los meses en la salida se rellenan automáticamente con espacios en blanco.
- Para eliminar los espacios rellenos o suprimir los ceros a la izquierda, utilice el elemento fm de modo de relleno.
- Puede formatear el campo de caracteres resultante con el comando COLUMN que se cubre en una lección posterior.

```
SQL> SELECT LAST_NAME, TO_CHAR(HIRE_DATE, 'DAY, DD/MM/YYYY') AS FECHA
2 FROM EMPLOYEES
3 WHERE EMPLOYEE_ID=100;
```

LAST_NAME	FECHA
King	MARTES ,17/06/2003

Elementos de la cadena de formatos de la función TO_CHAR

YYYY	Año completo en números.
YEAR	Año completo en letras.
MM	Número del mes.
MONTH	Nombre del mes.
MON	Abreviatura del mes.
DY	Abreviatura del día de la semana.
DAY	Nombre del día de la semana.
DD	Día del mes en número.
HH24	Hora.
MI	Minutos.
SS	Segundos.

```
SELECT last_name,
       TO_CHAR(hire_date, 'fmDD Month YYYY')
       AS HIREDATE
FROM employees;
```

Para usar la función TO_CHAR con valores numéricos se utilizan las siguientes formas:

```
SELECT TO_CHAR(salary, '$99,999.00') SUELDO
FROM employees
WHERE last_name = 'Ernst';
```

9	Representa un número.
0	Obliga un número.
\$	Muestra signo de dólar.
L	Muestra el signo monetario del servidor.

- .
- Separador decimal.
- ,
- Separador de miles.

```
SQL>
SQL>
SQL> SELECT LAST_NAME, TO_CHAR(SALARY, 'L99,999.00') AS SUELDO
2 FROM EMPLOYEES
3 WHERE EMPLOYEE_ID=100;
```

LAST_NAME	SUELDO
King	\$/24,000.00

Funciones TO_NUMBER y TO_DATE

Puede convertir una cadena de caracteres en un número o en una fecha. Para ello, utilice las funciones TO_NUMBER o TO_DATE. El modelo de formato que elija se basa en los elementos de formato mostrados previamente.

El modificador "x" especifica una coincidencia exacta para el argumento de caracteres y el modelo de formato de fecha de una función TO_DATE:

- La puntuación y el texto entrecomillado del argumento de caracteres deben coincidir exactamente (excepto en las mayúsculas/minúsculas) con las partes correspondientes del modelo de formato.
- El argumento de caracteres no puede tener espacios en blanco adicionales. Sin fx, Oracle ignora los espacios en blanco adicionales.
- Los datos numéricos del argumento de caracteres deben tener el mismo número de dígitos que el elemento correspondiente del modelo de formato. Sin fx, los números del argumento de caracteres pueden omitir los ceros a la izquierda.

Visualice los nombres y las fechas de contratación de todos los empleados que empezaron a trabajar el 24 de mayo de 1999. Como se utiliza el modificador fx, se requiere una coincidencia exacta y los espacios después de la palabra 'May' no se reconocen.

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM employees
WHERE hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
```

Funciones de Anidamiento

Las funciones de una sola fila se pueden anidar a cualquier profundidad. Las funciones anidadas se evalúan desde el nivel más interno al más externo. A continuación, se indican algunos ejemplos que muestran la flexibilidad de estas funciones.

```
SELECT last_name,
       NVL(TO_CHAR(manager_id), 'Sin Jefe')
FROM employees
WHERE manager_id IS NULL;
```

Funciones Generales

Estas funciones trabajan con cualquier tipo de dato y están relacionadas con el uso de valores nulos en la lista de expresiones.

La Función NVL

Para convertir un valor nulo en uno real, utilice la función NVL.

NVL (expr1, expr2)

En la sintaxis:

expr1 es el valor o la expresión de origen que puede contener un valor nulo.

expr2 es el valor de destino para convertir el valor nulo.

Puede utilizar la función NVL para convertir cualquier tipo de dato, pero el valor de retorno siempre es del mismo tipo que expr1.

```
SELECT last_name, salary, NVL(commission_pct, 0),  
       (salary*12)+(salary*12*NVL(commission_pct,0)) SUELDO_ANUAL  
FROM employees;
```

La Función NVL2

La función NVL2 examina la primera expresión. Si no es nula, devuelve la segunda expresión. Si es nula, devuelve la tercera.

NVL(expr1, expr2, expr3)

En la sintaxis:

expr1 es el valor o la expresión de origen que puede contener un valor nulo.

expr2 es el valor devuelto si expr1 no es nulo.

expr3 es el valor devuelto si expr1 es nulo.

```
SELECT last_name, salary, commission_pct,  
       NVL2(commission_pct, 'SAL+COMM', 'SAL') resultado  
FROM employees  
WHERE department_id IN (10, 20);
```

```

1 SELECT last_name,salary ,commission_pct,
2        NUL2(commission_pct,'SAL+COMM','SAL') resultado
3 FROM EMPLOYEES
4* WHERE department_id IN (10, 20)
SQL> /

```

LAST_NAME	SALARY	COMMISSION_PCT	RESULTAD
Whalen	4400		SAL
Hartstein	13000		SAL
Fay	6000		SAL

La Función COALESCE

Esta función permite evaluar dos veces dentro de la misma sentencia. En el ejemplo primero se evalúa el campo comisión_pct y luego el campo salary.

```

SELECT last_name,
       COALESCE(commission_pct, salary, 10) comm
FROM employees
ORDER BY commission_pct;

```

La Expresión CASE

```

CASE expr WHEN comparison_expr1 THEN return_expr1
          [WHEN comparison_expr2 THEN return_expr2
          WHEN comparison_exprn THEN return_exprn
          ELSE else_expr]
END

```

Las expresiones CASE le permiten utilizar la lógica IF-THEN-ELSE en sentencias SQL sin tener que llamar a procedimientos.

En una expresión CASE sencilla, Oracle busca el primer par WHEN ... THEN para el que expr es igual a comparison_expr y devuelve return_expr. Si ninguno de los pares WHEN ... THEN cumplen esta condición y existe una cláusula ELSE, Oracle devuelve else_expr. En caso contrario, Oracle devuelve un valor nulo. No puede especificar el literal NULL para todas las expresiones return_expr ni para else_expr.

Todas las expresiones (expr, comparison_expr y return_expr) deben tener el mismo tipo de dato, que puede ser CHAR, VARCHAR2, NCHAR o NVARCHAR2.


```

1 SELECT last_name, job_id, salary,
2         CASE job_id WHEN 'IT_PROG' THEN 1.10*salary
3                     WHEN 'ST_CLERK' THEN 1.15*salary
4                     WHEN 'SA_REP' THEN 1.20*salary
5                     ELSE salary END "NUEVO SUELDO"
6* FROM employees
SQL> /

```

LAST_NAME	JOB_ID	SALARY	NUEVO SUELDO
King	AD_PRES	24000	24000
Kochhar	AD_UP	17000	17000
De Haan	AD_UP	17000	17000
Hunold	IT_PROG	9000	9900
Ernst	IT_PROG	6000	6600
Austin	IT_PROG	4800	5280
Pataballa	IT_PROG	4800	5280
Lorentz	IT_PROG	4200	4620
Greenberg	FI_MGR	12008	12008
Faviet	FI_ACCOUNT	9000	9000
Chen	FI_ACCOUNT	8200	8200

La Función DECODE

```

DECODE(col/expression, search1, result1
      [, search2, result2,...,]
      [, default])

```

La función DECODE descodifica una expresión de forma similar a la lógica IF-THEN-ELSE utilizada en varios idiomas. Esta función descodifica expression después de compararla con cada valor search. Si la expresión es la misma que search, se devuelve result.

Si el valor por defecto está omitido, se devuelve un valor nulo cuando un valor de búsqueda no coincide con ninguno de los valores resultantes.

```

1 SELECT last_name, job_id, salary,
2         DECODE(job_id, 'IT_PROG', 1.10*salary,
3                 'ST_CLERK', 1.15*salary,
4                 'SA_REP', 1.20*salary,
5                 salary)
6         "NUEVO SUELDO"
7* FROM employees
SQL> /

```

LAST_NAME	JOB_ID	SALARY	NUEVO SUELDO
King	AD_PRES	24000	24000
Kochhar	AD_UP	17000	17000
De Haan	AD_UP	17000	17000
Hunold	IT_PROG	9000	9900
Ernst	IT_PROG	6000	6600
Austin	IT_PROG	4800	5280
Pataballa	IT_PROG	4800	5280
Lorentz	IT_PROG	4200	4620
Greenberg	FI_MGR	12008	12008
Faviet	FI_ACCOUNT	9000	9000
Chen	FI_ACCOUNT	8200	8200

Ejercicio práctico

1. Muestre la estructura de la tabla departments. Seleccione todos los datos de la tabla.
2. Muestre la estructura de la tabla EMPLOYEES. Cree una consulta para mostrar el apellido, el código de cargo, la fecha de contratación y el número de empleado para cada empleado, con el número de empleado en primer lugar. Proporcione un alias para la columna. Guarde la sentencia SQL en un archivo llamado lab1_2.sql.
3. Cree una consulta para mostrar códigos de cargo únicos de la tabla EMPLOYEES.
4. Muestre el apellido concatenado con el identificador de cargo, separados por una coma y un espacio y llame a la columna Empleado y Cargo.
5. Cree una consulta para mostrar todos los datos de la tabla EMPLOYEES. Separe cada columna con una coma. Llame a la columna SALIDA.
6. Cree una consulta para mostrar el apellido y el salario de los empleados que ganan más de \$12.000. Coloque la sentencia SQL en un archivo de texto llamado lab1_6.sql. Ejecute la consulta.
7. Cree una consulta para mostrar el apellido del empleado y el número de departamento para el número de empleado 176.
8. Modifique lab1_6.sql para mostrar el apellido y el salario para todos los empleados cuyos salarios no están comprendidos entre \$5.000 y \$12.000. Coloque la sentencia SQL en un archivo de texto llamado lab1_8.sql.
9. Muestre el apellido del empleado, el identificador de cargo y la fecha de inicio de los empleados contratados entre el 20 de febrero de 1998 y el 1 de mayo de 1998. Ordene la consulta en orden ascendente por fecha de inicio.
10. Muestre el apellido y el número de departamento de todos los empleados de los departamentos 20 y 50 en orden alfabético por apellido.
11. Modifique lab1_8.sql para enumerar el apellido y el salario de los empleados que ganan entre \$5.000 y \$12.000, y están en el departamento 20 ó 50. Etiquete las columnas Empleado y Sueldo Mensual, respectivamente.
12. Muestre el apellido y la fecha de contratación de todos los empleados contratados en 1994.
13. Muestre el apellido y el cargo de todos los empleados que no tienen jefe.
14. Muestre el apellido, el salario y la comisión para todos los empleados que ganan comisiones. Ordene los datos en orden descendente de salarios y comisiones.
15. Muestre el apellido, el cargo y el salario de todos los empleados cuyos cargos sean representantes de ventas o encargados de stock y cuyos salarios no sean iguales a \$2.500, \$3.500 ni \$7.000.
16. Modifique lab1_8.sql para mostrar el apellido, el salario y la comisión para todos los empleados cuyas comisiones son el 20 %.
17. Escriba una consulta para mostrar la fecha actual. Etiquete la columna como Date.
18. Para cada empleado, visualice su número, apellido, salario y salario incrementado en el 15 % y expresado como número entero. Etiquete la

columna como Nuevo Sueldo. Ponga la sentencia SQL en un archivo de texto llamado lab1_18.sql.

19. Modifique la consulta lab1_18.sql para agregar una columna que reste el salario antiguo del nuevo. Etiquete la columna como Increase.
20. Escriba una consulta que muestre los apellidos de los empleados con la primera letra en mayúsculas y todas las demás en minúsculas, así como la longitud de los nombres, para todos los empleados cuyos nombres comienzan por J, A o M. Asigne a cada columna la etiqueta correspondiente. Ordene los resultados según los apellidos de los empleados.
21. Para cada empleado, muestre su apellido y calcule el número de meses entre el día de hoy y la fecha de contratación. Etiquete la columna como MESES_TRABAJADOS. Ordene los resultados según el número de meses trabajados. Redondee el número de meses hacia arriba hasta el número entero más próximo.

CAPÍTULO 2

CONSULTAS CON SQL

Contenido del Capítulo:

1. Consultas con varias tablas.
2. Sintaxis ANSI y sintaxis Oracle.
3. Uso de las funciones de agrupación de filas.

1. Visualización de datos de varias tablas.

Datos de Varias Tablas

A veces es necesario utilizar datos de más de una tabla. En la base de datos, en el schema HR existen tablas relacionadas que sirven para extraer datos en conjunto.

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
202	Fay	20
205	Higgins	110
206	Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700

EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
102	90	Executive
205	110	Accounting
206	110	Accounting

- Los identificadores de empleado están en la tabla EMPLOYEES. Es el campo EMPLOYEE_ID.
- Los identificadores de departamento están en las tablas EMPLOYEES y DEPARTMENTS. Esto implica una relación entre ambas tablas pues comparten información.
- Los identificadores de ubicación están en la tabla DEPARTMENTS.

Para producir el informe, debe enlazar las tablas EMPLOYEES y DEPARTMENTS y acceder a los datos de ambas.

Existen muchas formas de acceder a los datos de varias tablas al mismo tiempo:

PRODUCTOS CARTESIANOS

Cuando una condición de unión no es válida o está completamente omitida, el resultado es un producto Cartesiano, en el que se muestran todas las combinaciones de filas. Todas las filas de la primera tabla se unen con todas las filas de la segunda tabla.

Los productos Cartesianos tienden a generar un gran número de filas y es poco frecuente que el resultado sea útil. Debe incluir siempre una condición de unión válida en una cláusula WHERE, a menos que tenga la necesidad específica de combinar todas las filas de todas las tablas.

Los productos Cartesianos son útiles para algunas pruebas en las que se necesita generar un gran número de filas para simular una cantidad razonable de datos.

Se genera un producto Cartesiano si una condición de unión está omitida.

En el ejemplo se muestra el apellido del empleado y el nombre del departamento de las tablas EMPLOYEES y DEPARTMENTS. Como no se ha especificado ninguna cláusula WHERE, todas las filas de la tabla EMPLOYEES se han unido con todas las filas de la tabla DEPARTMENTS, generando muchas filas en la salida.

```
SQL>
SQL> SELECT LAST_NAME, DEPARTMENT_NAME
2 FROM EMPLOYEES,DEPARTMENTS
3 WHERE EMPLOYEE_ID=100;
```

LAST_NAME	DEPARTMENT_NAME
King	Administration
King	Marketing
King	Purchasing
King	Human Resources
King	Shipping
King	IT
King	Public Relations
King	Sales
King	Executive
King	Finance
King	Accounting

Otra manera de obtener el mismo resultado es utilizando la sintaxis del SQL 1999:

La cláusula CROSS JOIN produce varios productos entre dos tablas.

Es lo mismo que un producto Cartesiano entre las dos tablas.


```
SQL>
SQL> SELECT LAST_NAME, DEPARTMENT_NAME
2 FROM EMPLOYEES CROSS JOIN DEPARTMENTS
3 WHERE EMPLOYEE_ID=100;
```

LAST_NAME	DEPARTMENT_NAME
King	Administration
King	Marketing
King	Purchasing
King	Human Resources
King	Shipping
King	IT
King	Public Relations
King	Sales
King	Executive
King	Finance
King	Accounting

TIPOS DE UNIONES

La base de datos Oracle11g ofrece una sintaxis de unión conforme con SQL: 1999. Antes de la versión 11g, la sintaxis de unión era distinta de los estándares ANSI. La nueva sintaxis de unión conforme con SQL: 1999 no ofrece ninguna ventaja de rendimiento con respecto a la sintaxis de unión de propiedad de Oracle que existía en las versiones anteriores.

Las Uniones pueden ser:

- De igualdad
- De no igualdad
- Externas
- Autouniones

DEFINICIÓN DE UNIONES

Cuando son necesarios datos de más de una tabla de la base de datos, se utiliza una condición de unión. Las filas de una tabla se pueden unir a las de otra según valores comunes que existen en las columnas correspondientes, es decir, normalmente columnas de clave primaria y ajena.

Para visualizar datos de dos o más tablas relacionadas, escriba una condición de unión simple en la cláusula WHERE.

```
SELECT    table1.column, table2.column
FROM table1, table2
WHERE     table1.column1 = table2.column2;
```

En la sintaxis :

table1.column indica la tabla y la columna de la que se recuperan los datos

table1.column1 = es la condición que une (o relaciona) las tablas entre sí
table2.column2

Instrucciones

- Al escribir una sentencia SELECT que unas tablas, ponga delante del nombre de la columna el nombre de la tabla para obtener una mayor claridad y para mejorar el acceso a la base de datos.
- Si en más de una tabla aparece el mismo nombre de columna, éste debe llevar el nombre de tabla como prefijo.
- Para unir n tablas, necesita un mínimo de n-1 condiciones de unión. Por ejemplo, para unir cuatro tablas, se requiere un mínimo de tres uniones. Esta regla puede no aplicarse si la tabla tiene una clave primaria concatenada, en cuyo caso se requiere más de una columna para identificar a cada fila de forma exclusiva.

Uniones de Igualdad

Para determinar el nombre de departamento de un empleado, compare el valor de la columna de la tabla EMPLOYEES con los valores de DEPARTMENT_ID de la tabla DEPARTMENTS. La relación entre las tablas EMPLOYEES y DEPARTMENTS es una unión de igualdad, es decir, los valores de la columna DEPARTMENT_ID de ambas tablas deben ser iguales. Con frecuencia, este tipo de unión implica complementos de clave primaria y ajena.

Ejemplo:

```
SELECT employees.employee_id, employees.last_name,  
       employees.department_id, departments.department_id,  
       departments.location_id  
FROM   employees, departments  
WHERE  employees.department_id = departments.department_id;
```

- La cláusula SELECT especifica los nombres de columna que se recuperan:
 - Apellido del empleado, número del empleado y número de departamento, que son columnas de la tabla EMPLOYEES.
 - Número de departamento, nombre de departamento e identificador de ubicación, que son columnas de la tabla DEPARTMENTS.
- La cláusula FROM especifica las dos tablas a las que debe acceder la base de datos:
 - tabla EMPLOYEES.
 - tabla DEPARTMENTS.
- La cláusula WHERE especifica cómo se deben unir las tablas:
 - EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID
 - Como la columna DEPARTMENT_ID es común a ambas tablas, debe tener como prefijo el nombre de la tabla para evitar ambigüedades.

Condiciones de Búsqueda Adicionales

Además de la unión, puede tener criterios para la cláusula WHERE para restringir las filas en consideración para una o más tablas de la unión. Por ejemplo, para visualizar el número y el nombre de departamento del empleado Mato, necesita una condición adicional en la cláusula WHERE.

```
1 SELECT last_name, employees.department_id, department_name
2 FROM employees, departments
3 WHERE employees.department_id=departments.department_id
4* AND last_name= 'Matos'
SQL> /
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Matos	50	Shipping

Cualificación de Nombres de Columna Ambiguos

Es necesario que cualifique los nombres de las columnas de la cláusula WHERE con el nombre de tabla para evitar ambigüedades. Sin los prefijos de tabla, la columna DEPARTMENT_ID podría ser de la tabla DEPARTMENTS o de la tabla EMPLOYEES. Es necesario agregar el prefijo de tabla para ejecutar la consulta.

Si no hay nombres de columna comunes entre dos tablas, no es necesario cualificar las columnas. Sin embargo, el uso del prefijo de tabla mejora el rendimiento ya que indica a Oracle Server dónde debe buscar exactamente las columnas.

La necesidad de cualificar los nombres de columna ambiguos también es aplicable a columnas que pueden ser ambiguas en otras cláusulas, como SELECT u ORDER BY.

ALIAS DE TABLA

La cualificación de nombres de columna con nombres de tabla puede llevar mucho tiempo, especialmente si los nombres de tabla son largos. Puede utilizar en su lugar alias de tabla. Al igual que un alias de columna asigna a una columna otro nombre, un alias de tabla asigna a una tabla otro nombre. Los alias de tabla ayudan a reducir el código SQL, utilizando así menos memoria.

Observe cómo se identifican los alias de tabla en la cláusula FROM del ejemplo. El nombre de tabla se especifica completo, seguido de un espacio y del alias de tabla. A la tabla EMPLOYEES se le ha asignado el alias e y a la tabla DEPARTMENTS el alias d.

Instrucciones

- Los alias de tabla pueden tener hasta 30 caracteres, pero cuanto más pequeños sean, mejor.
- Si se utiliza un alias de tabla para un nombre de tabla en particular en la cláusula FROM, dicho alias se debe sustituir por el nombre de tabla en toda la sentencia SELECT.
- Los alias de tabla deben ser significativos.
- El alias de tabla sólo es válido para la sentencia SELECT actual.

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e, departments d
WHERE  e.department_id = d.department_id;
```

Condiciones de Búsqueda Adicionales

A veces es posible que necesite unir más de dos tablas. Por ejemplo, para visualizar el apellido, el nombre de departamento y la ciudad de cada empleado, tiene que unir las tablas:

EMPLOYEES		DEPARTMENTS		LOCATIONS	
LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID	LOCATION_ID	CITY
King	90	10	1700	1400	Southlake
Kochhar	90	20	1800	1500	South San Francisco
De Haan	90	50	1500	1700	Seattle
Hunold	60	60	1400	1800	Toronto
Ernst	60	80	2500	2500	Oxford
Lorentz	60	90	1700		
Mourgos	50	110	1700		
Rajs	50	190	1700		
Davies	50				
Matos	50				
Vargas	50				
Zlotkey	80				
Abel	80				
Taylor	80				

8 rows selected.

20 rows selected.

```
SQL> Run SQL Command Line

1  SELECT e.last_name, d.department_name, l.city
2  FROM   employees e, departments d, locations l
3  WHERE  e.department_id = d.department_id
4  AND    d.location_id = l.location_id
5* AND    department_name = 'Sales'
SQL> /

LAST_NAME      DEPARTMENT_NAME
-----
CITY
-----
Russell
Oxford        Sales

Partners
Oxford        Sales

Errazuriz
Oxford        Sales
```

UNIONES DE NO IGUALDAD

Una unión de no igualdad es una condición de unión que contiene algo distinto a un operador de igualdad.

La relación entre las tablas EMPLOYEES y JOB_GRADES es un ejemplo de unión de no igualdad. Una relación entre las dos tablas es que la columna SAL de la tabla EMPLOYEES debe estar entre los valores de las columnas lowest y highest de la tabla JOB_GRADES. La relación se obtiene utilizando un operador distinto de igual a (=).

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e, job_grades j
WHERE  e.salary
      BETWEEN j.lowest_sal AND j.highest_sal;
```

DEVOLUCIÓN DE REGISTROS SIN COINCIDENCIA DIRECTA CON UNIONES EXTERNAS

Si una fila no satisface una condición de unión, no aparecerá en el resultado de la búsqueda. Por ejemplo, en la condición de unión de igualdad de las tablas EMPLOYEES y DEPARTMENTS, el empleado Grant no aparece porque no hay identificador de departamento registrado para él en la tabla EMPLOYEES.

USO DE UNIONES EXTERNAS PARA DEVOLVER REGISTROS SIN COINCIDENCIA DIRECTA

Las filas que faltan se pueden devolver si se utiliza un operador de unión externa en la condición de unión. El operador es un signo más entre paréntesis (+) y se coloca “al lado” de la unión que tiene información insuficiente. Este operador tiene el efecto de crear una o varias filas nulas, a las que se pueden unir una o varias filas de la tabla que no tiene información insuficiente.

En la sintaxis:

table1.column = es la condición que une (o relaciona) las tablas entre sí.

table2.column (+) es el símbolo de unión externa, que se puede colocar en cualquier lado de la condición de cláusula WHERE, pero no en ambos (Coloque el símbolo de unión externa a continuación del nombre de la columna en la tabla sin filas coincidentes.)

```
SELECT   table1.column, table2.column
FROM     table1, table2
WHERE    table1.column(+) = table2.column;
```

```
SELECT    table1.column, table2.column
FROM table1, table2
WHERE     table1.column = table2.column(+);
```

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e, departments d
WHERE  e.department_id(+) = d.department_id ;
```

En el ejemplo se muestran los apellidos de los empleados, los identificadores de departamento y los nombres de departamento. El departamento Contracting no tiene empleados. En la salida mostrada aparece el valor vacío.

Restricciones de Unión Externa

- El operador de unión externa puede aparecer solamente en un lado de la expresión, el lado en el que falta información. Devuelve las filas de una tabla que no tienen coincidencia directa en la otra tabla.
- Una condición que implique una unión externa no puede utilizar el operador IN ni estar enlazada a otra condición con el operador OR.

DEFINICIÓN DE UNIONES CON LA SINTAXIS SQL 1999

```
SELECT    table1.column, table2.column
FROM table1
[CROSS JOIN table2] |
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
    ON(table1.column_name = table2.column_name)] |
[LEFT|RIGHT|FULL OUTER JOIN table2
    ON (table1.column_name = table2.column_name)];
```

Al utilizar la sintaxis SQL: 1999, puede obtener resultados iguales a los mostrados en páginas anteriores.

En la sintaxis:

table1.column	Indica la tabla y la columna de la que se recuperan los datos.
CROSS JOIN	Devuelve un producto Cartesiano de las dos tablas.
NATURAL JOIN	Une dos tablas basadas en el mismo nombre de columna.
JOIN table	
USING column_name	Realiza una unión de igualdad basada en el nombre de columna
JOIN table ON	
table1.column_name	Realiza una unión de igualdad basada en la condición de la cláusula ON
= table2.column_name	
LEFT/RIGHT/FULL OUTER	

Autouniones

EMPLOYEES (WORKER)

EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourges	100

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourges

Hay casos en que una misma tabla debe ser leída dos veces para relacionarla consigo misma. Este es el caso de la autounión. En la tabla EMPLOYEES existe el código de jefe MANAGER_ID, que hace referencia a la misma clave EMPLOYEE_ID de la tabla. En ese caso, la tabla puede ser levantada en memoria dos veces para usarla como si fueran dos tablas diferentes.

```
SELECT worker.last_name || ' trabaja para '
       || manager.last_name
FROM   employees worker, employees manager
WHERE  worker.manager_id = manager.employee_id ;
```



```

1 SELECT worker.last_name || ' trabaja para '
2      || manager.last_name
3 FROM   employees worker, employees manager
4* WHERE worker.manager_id = manager.employee_id
SQL> /

```

```

WORKER.LAST_NAME||'TRABAJA PARA' ||MANAGER.LAST_NAME
-----

```

```

Smith trabaja para  Cambrault
Ozer trabaja para   Cambrault
Kumar trabaja para  Cambrault
Fox trabaja para    Cambrault
Bloom trabaja para  Cambrault
Bates trabaja para  Cambrault
Hunold trabaja para De Haan
Uishney trabaja para Errazuriz
Marvins trabaja para Errazuriz
Lee trabaja para    Errazuriz
Greene trabaja para Errazuriz

```

Creación de Uniones Naturales

En las versiones anteriores de Oracle, no era posible realizar una unión sin especificar de forma explícita las columnas en las tablas correspondientes. En Oracle11g se puede dejar que se termine automáticamente la unión basada en columnas de las dos tablas que tienen tipos de dato y nombres coincidentes, utilizando las palabras clave NATURAL JOIN.

Nota: La unión sólo puede ocurrir en columnas que tengan los mismos nombres y tipos de dato en las dos tablas. Si las columnas tienen el mismo nombre, pero distintos tipos de dato, la sintaxis NATURAL JOIN produce un error.

```

SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;

```

La Cláusula USING

Las uniones naturales utilizan todas las columnas con nombres y tipos de dato coincidentes para unir las tablas. La cláusula USING se puede utilizar para especificar solamente las columnas que se deben utilizar para una unión de igualdad. Las columnas de referencia en la cláusula USING no deben tener un cualificador (nombre o alias de tabla) en ningún lugar de la sentencia SQL.

```

SELECT e.employee_id, e.last_name, d.location_id
FROM   employees e JOIN departments d
USING (department id) ;

```

La misma restricción se aplica también a las uniones NATURAL. Por lo tanto, las columnas que tienen el mismo nombre en las dos tablas se deben utilizar sin ningún cualificador.

La Condición ON

Utilice la cláusula ON para especificar una condición de unión. Esto le permite especificar condiciones de unión distintas de cualquier condición de búsqueda o filtro en la cláusula WHERE.

- La condición de unión para la unión natural es básicamente una unión de igualdad de todas las columnas con el mismo nombre.
- Para especificar condiciones arbitrarias o especificar columnas para unir, se utiliza la cláusula ON.
- La condición de unión se separa de otras condiciones de búsqueda.
- La cláusula ON facilita la comprensión del código.

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
FROM   employees e JOIN departments d  
ON     (e.department_id = d.department_id);
```

Uniones en Tres Sentidos

Una unión en tres sentidos es una unión de tres tablas. En la sintaxis conforme con SQL: 1999, las uniones se realizan de izquierda a derecha, por lo que la primera que se realiza es EMPLOYEES JOIN DEPARTMENTS. La primera condición de unión puede hacer referencia a columnas de EMPLOYEES y de DEPARTMENTS, pero no a columnas de referencia en LOCATIONS. La segunda condición de unión puede hacer referencia a columnas de las tres tablas.

También se puede escribir como unión de igualdad en tres sentidos:

```
SELECT employee_id, city, department_name  
FROM   employees e  
JOIN   departments d  
ON     d.department_id = e.department_id  
JOIN   locations l  
ON     d.location_id = l.location_id;
```

Uniones INNER frente a OUTER

En SQL: 1999, la unión de dos tablas que devuelve solamente las filas coincidentes es una unión interna.

Una unión entre dos tablas que devuelve los resultados de la unión interna así como las tablas izquierda (o derecha) de filas no coincidentes es una unión externa izquierda (o derecha).

Una unión entre dos tablas que devuelve los resultados de una unión interna así como los de una unión izquierda y derecha es una unión externa completa.

Ejemplo de LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
LEFT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

Ejemplo de RIGHT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
FULL OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

Ejemplo de ambos lados.

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e
RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

FUNCIONES DE GRUPOS

A diferencia de las funciones de una sola fila, las funciones de grupo operan sobre juegos de filas para proporcionar un resultado por grupo. Estos juegos pueden ser la tabla completa o la tabla dividida en grupos.

Las funciones de grupo son:

- AVG Promedio
- COUNT Cantidad
- MAX Valor mayor
- MIN Valor menor
- STDDEV Desviación estandar
- SUM Sumatoria
- VARIANCE Varianza

```
SELECT   [columnn,] group_function(columnn), ...
FROM     table
[WHERE   condition]
[GROUP BY columnn]
[ORDER BY columnn];
```

Instrucciones para el Uso de Funciones de Grupo

- DISTINCT hace que la función solamente considere valores no duplicados; ALL hace que considere todos los valores incluyendo duplicados. El valor por defecto es ALL y, por lo tanto, no es necesario especificarlo.
- Los tipos de dato para las funciones con un argumento expr pueden ser CHAR, VARCHAR2, NUMBER o DATE.
- Todas las funciones de grupo ignoran los valores nulos. Para sustituir un valor por valores nulos, utilice las funciones NVL, NVL2 o COALESCE.
- Oracle Server ordena implícitamente el juego de resultados en orden ascendente al utilizar una cláusula GROUP BY. Para sustituir este orden por defecto, se puede utilizar DESC en una cláusula ORDER BY.

Funciones de Grupo

Puede utilizar las funciones AVG, SUM, MIN y MAX en columnas que almacenan datos numéricos. En el ejemplo se muestra la media, el salario mayor, el menor y la suma de los salarios mensuales de todos los representantes de ventas.

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

```
1 SELECT AVG(salary), MAX(salary),
2        MIN(salary), SUM(salary)
3 FROM   employees
4* WHERE job_id LIKE '%REP%'
SQL> /
```

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8272,72727	11500	6000	273000

Puede utilizar las funciones MAX y MIN para cualquier tipo de dato. En el ejemplo se muestra al empleado de menor edad y al de mayor edad.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM   employees;
```

En el siguiente ejemplo se muestra el apellido del primer y del último empleado de una lista alfabética de todos los empleados.

```
SQL>
SQL> SELECT MIN(LAST_NAME), MAX(LAST_NAME)
2 FROM EMPLOYEES;
```

MIN(LAST_NAME)	MAX(LAST_NAME)
Abel	Zlotkey

Nota: Las funciones AVG, SUM, VARIANCE y STDDEV sólo se pueden utilizar con tipos de dato numéricos.

La Función COUNT

La función COUNT tiene tres formatos:

- COUNT(*)
- COUNT(expr)
- COUNT(DISTINCT expr)

COUNT(*) devuelve el número de filas de una tabla que satisface los criterios de la sentencia SELECT, incluyendo filas duplicadas y filas que contengan valores nulos en cualquiera de las columnas. Si se incluye una cláusula WHERE en la sentencia SELECT, COUNT(*) devuelve el número de filas que satisface la condición de la cláusula WHERE..

En contraste, COUNT(expr) devuelve el número de valores no nulos de la columna identificada por expr.

COUNT(DISTINCT expr) devuelve el número de valores únicos no nulos de la columna identificada por expr.

```
SELECT COUNT(*)  
FROM employees  
WHERE department_id = 50;
```

Funciones de Grupo y Valores Nulos

Todas las funciones de grupo ignoran los valores nulos de la columna. En el ejemplo, la media se calcula basándose solamente en las filas de la tabla que almacenan un valor válido en la columna COMM. La media se calcula como la comisión total pagada a todos los empleados dividida por el número de empleados que perciben una comisión (cuatro).

```
SELECT COUNT(commission_pct)  
FROM employees  
WHERE department_id = 80;
```

La función NVL fuerza a que las funciones de grupo incluyan valores nulos. En el ejemplo, la media se calcula basándose en todas las filas de la tabla, independientemente de si se almacenan valores nulos en la columna COMMISSION_PCT. La media se calcula como la comisión total pagada a todos los empleados dividida por el número total de empleados de la compañía.

```
SELECT AVG(commission_pct)  
FROM employees;
```

```
SQL>
SQL>
SQL> SELECT AVG<NUL<COMMISSION_PCT,0>>
      2 FROM EMPLOYEES;

AVG<NUL<COMMISSION_PCT,0>>
-----
      .072897196
```

También se puede anidar la cláusula DISTINCT para evitar repetir algunos valores durante los procesos de las funciones de grupo.

```
SELECT COUNT(DISTINCT department_id)
FROM employees;
```

Grupos de Datos

Hasta ahora, todas las funciones de grupo han tratado la tabla como un gran grupo de información. A veces, necesita dividir la tabla de información en grupos más pequeños. Esto se puede realizar utilizando la cláusula GROUP BY.

```
SELECT      column, group_function(column)
FROM        table
[WHERE      condition]
[GROUP BY  group_by_expression]
[ORDER BY  column];
```

A continuación, puede utilizar las funciones de grupo para devolver información de resumen para cada grupo.

En la sintaxis:

group_by_expression especifica columnas cuyos valores determinan la base para agrupar filas

Instrucciones

- Si incluye una función de grupo en una cláusula SELECT, no puede seleccionar también resultados individuales, a menos que la columna individual aparezca en la cláusula GROUP BY. Recibirá un mensaje de error si no incluye la lista de columnas en la cláusula GROUP BY.
- Si utiliza una cláusula WHERE, puede excluir filas antes de dividir las en grupos.
- Debe incluir las columnas en la cláusula GROUP BY.
- No se puede utilizar un alias de columna en la cláusula GROUP BY.
- Por defecto, las filas se ordenan en orden ascendente de las columnas incluidas en la lista GROUP BY. Puede sustituir este orden por defecto utilizando la cláusula ORDER BY.

Al utilizar la cláusula GROUP BY, asegúrese de que todas las columnas de la lista SELECT que no son funciones de grupo están incluidas en la cláusula GROUP BY. En el ejemplo se muestra el número de departamento y el salario medio para cada

departamento. A continuación, se muestra el modo en que se evalúa esta sentencia SELECT, que contiene una cláusula GROUP BY:

- La cláusula SELECT especifica las columnas que se van a recuperar:
 - Columna de número de departamento de la tabla EMP
 - La media de todos los salarios en el grupo que ha especificado en la cláusula GROUP BY
- La cláusula FROM especifica las tablas a las que debe acceder la base de datos: la tabla EMP.
- La cláusula WHERE especifica las filas que se van a recuperar. Como no hay ninguna cláusula WHERE, se recuperan todas las filas por defecto.
- La cláusula GROUP BY especifica cómo se deben agrupar las filas. Las filas se agrupan por número de departamento, por lo que la función AVG que se está aplicando a la columna de salario calculará el salario medio de cada departamento.

```
SELECT  department_id, AVG(salary)
FROM    employees
GROUP BY department_id ;
```

La columna GROUP BY no tiene que estar en la cláusula SELECT. Por ejemplo, la sentencia SELECT de la transparencia muestra los salarios medios para cada departamento sin mostrar los números de los departamentos respectivos. Sin embargo, sin los números de departamento, los resultados no parecen significativos.

```
SELECT  AVG(salary)
FROM    employees
GROUP BY department_id ;
```

Grupos dentro de Grupos

A veces necesita ver resultados para grupos dentro de grupos. Por ejemplo se puede requerir un informe con el salario total que se paga a cada cargo, dentro de cada departamento.

La tabla EMPLOYEES se agrupa en primer lugar por número de departamento y, dentro de dicha agrupación, por cargo.

```
SELECT  department_id dept_id, job_id, SUM(salary)
FROM    employees
GROUP BY department_id, job_id ;
```

Puede devolver resultados resumidos para grupos y subgrupos incluyendo en una lista más de una columna GROUP BY. Puede determinar el orden por defecto de los resultados por el orden de las columnas en la cláusula GROUP BY. A continuación, se muestra el modo en que se evalúa la sentencia SELECT de la transparencia, que contiene una cláusula GROUP BY:

- La cláusula SELECT especifica la columna que se va a recuperar:
 - Número de departamento de la tabla EMP

- Identificador de cargo de la tabla EMP
- La suma de todos los salarios en el grupo que ha especificado en la cláusula GROUP BY
- La cláusula FROM especifica las tablas a las que debe acceder la base de datos: la tabla EMP.
- La cláusula GROUP BY especifica cómo debe agrupar las filas:
 - En primer lugar, las filas se agrupan por número de departamento.
 - En segundo lugar, dentro de los grupos de números de departamento, las filas se agrupan por identificador de cargo.

De esta forma, la función SUM se está aplicando a la columna de salario para todos los identificadores de cargo dentro de cada grupo de números de departamento.

Restricción de Resultados de Grupo

```
SELECT      column, group_function
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[HAVING     group_condition]
[ORDER BY   column];
```

De la misma forma que utiliza la cláusula WHERE para restringir las filas que selecciona, utilice la cláusula HAVING para restringir grupos. Para buscar el salario máximo de cada departamento, pero mostrando solamente los departamentos que tengan un salario máximo de más de 10.000, debe:

1. Buscar el salario medio para cada departamento agrupando por número de departamento.
2. Restringir los grupos a los departamentos con un salario máximo mayor que 10.000.

```
SELECT  department_id, MAX(salary)
FROM    employees
GROUP BY department_id
HAVING  MAX(salary) > 10000 ;
```

En el ejemplo se muestran números de departamento y salarios máximos para los departamentos cuyos salarios máximos son mayores que \$10.000.

Puede utilizar la cláusula GROUP BY sin utilizar una función de grupo en la lista SELECT.

Si restringe las filas basándose en el resultado de una función de grupo, debe tener una cláusula GROUP BY además de la cláusula HAVING.

En el siguiente ejemplo se muestran los números de departamento y los salarios medios para los departamentos cuyos salarios máximos sean mayores que \$10.000:

En el ejemplo se muestra el identificador de cargo y el salario mensual total para cada cargo con una nómina total superior a \$13.000. El ejemplo excluye a representantes de ventas y ordena la lista según el salario mensual total.

```
SELECT    job_id, SUM(salary) PLANILLA
FROM      employees
WHERE     job_id NOT LIKE '%REP%'
GROUP BY  job_id
HAVING    SUM(salary) > 13000
ORDER BY  SUM(salary) ;
```

Anidamiento de Funciones de Grupo

Las funciones de grupo se pueden anidar hasta una profundidad de dos. En el ejemplo de la transparencia se muestra el salario medio máximo.

```
SELECT    MAX(AVG(salary))
FROM      employees
GROUP BY  department_id;
```

Ejercicio práctico

1. Escriba una consulta para visualizar el apellido del empleado, el número y el nombre de departamento para todos los empleados.
2. Cree un listado único de todos los cargos que haya en el departamento 20. Incluya la ubicación del departamento en el resultado.
3. Escriba una consulta para mostrar el apellido del empleado, el nombre de departamento, el identificador de ubicación y la ciudad de todos los empleados que perciben comisión.
4. Visualice el apellido del empleado y el nombre de departamento para todos los empleados que tengan una a (minúsculas) en el apellido.
5. Escriba una consulta para visualizar el apellido, el cargo, el número y el nombre de departamento para todos los empleados que trabajan en Toronto.
6. Visualice el apellido y el número del empleado junto con el apellido y el número de su director. Etiquete las columnas como Employee, Emp#, Manager y Mgr#, respectivamente.
7. Cree una consulta que muestre apellidos de empleado, números de departamento y todos los empleados que trabajan en el mismo departamento que un empleado dado. Asigne a cada columna la etiqueta adecuada.
8. Visualice la estructura de la tabla JOB_GRADES. Cree una consulta en la que pueda visualizar el nombre, el cargo, el nombre de departamento, el salario y el grado de todos los empleados.
9. Cree una consulta para visualizar el apellido y la fecha de contratación de cualquier empleado contratado después del empleado Davies.
10. Visualice los nombres y las fechas de contratación de todos los empleados contratados antes que sus directores, junto con los nombres y las fechas de contratación de estos últimos. Etiquete las columnas como Employee, Emp Hired, Manager y Mgr Hired, respectivamente.
11. Escriba una consulta para visualizar el número de personas con el mismo cargo.
12. Determine el número de directores sin enumerarlos. Etiquete la columna como Number of Managers. Indicación: Utilice la columna MANAGER_ID para determinar el número de directores.
13. Escriba una consulta para visualizar la diferencia entre el salario mayor y el menor. Etiquete la columna DIFERENCIA.
14. Visualice el número de director y el salario del empleado de menor sueldo para dicho director. Excluya a todas las personas con director desconocido. Excluya los grupos donde el salario mínimo sea \$6.000 o inferior. Ordene el resultado en orden descendente de salario.
15. Escriba una consulta para visualizar el nombre, la ubicación, el número de empleados y el salario medio de todos los empleados de cada departamento. Etiquete las columnas como Name, Location, Number of People y Sal, respectivamente. Redondee el salario medio en dos posiciones decimales.
16. Cree una consulta que muestre el número total de empleados y, de ese total, el número de empleados contratados en 1995, 1996, 1997 y 1998. Cree las cabeceras de columna adecuadas.

CAPÍTULO 3

SUBCONSULTAS

Contenido del Capítulo:

1. Definición.
2. Sintaxis de las subconsultas y parámetros.
3. Tipos de subconsultas.
4. Operadores de conjuntos

Definición

Uso de una Subconsulta para Resolver un Problema

Suponga que desea escribir una consulta para averiguar quién gana un salario mayor que el de Abel.

Para resolver este problema, necesita dos consultas: una para buscar lo que gana Abel y una segunda para buscar quién gana más de esta cantidad.

Puede resolver este problema combinando las dos consultas, colocando una consulta dentro de la otra.

La consulta interna o subconsulta devuelve un valor que utiliza la consulta externa o principal. La utilización de una subconsulta es equivalente a realizar dos consultas secuenciales y usar el resultado de la primera como valor de búsqueda en la segunda.

```
SELECT    select_list
FROM table
WHERE     expr operator
          (SELECT    select_list
           FROM      table);
```

Las subconsultas son sentencias SELECT embebidas en cláusulas de otras sentencias SELECT. Con ellas puede crear sentencias potentes a partir de sentencias simples. Esto puede resultar muy útil si necesita seleccionar filas de una tabla con una condición que depende de los datos de la propia tabla.

Puede colocar la subconsulta en diversas cláusulas SQL como, por ejemplo:

- La cláusula WHERE
- La cláusula HAVING
- La cláusula FROM

En la sintaxis:

operator incluye una condición de comparación como >, = o IN

Las condiciones de comparación son de dos clases: operadores de una sola fila (>, =, >=, <, <>, <=) y operadores de varias filas (IN, ANY, ALL).

Con frecuencia se hace referencia a la subconsulta como sentencia SELECT anidada, sub-SELECT o SELECT interna. La subconsulta se suele ejecutar primero y su resultado se utiliza para completar la condición de consulta de la consulta principal o externa.

Ejemplo:

```
SELECT last_name
FROM employees 11000
WHERE salary >
  (SELECT salary
   FROM employees
   WHERE last_name = 'Abel');
```

Instrucciones para Utilizar Subconsultas

- La subconsulta se debe escribir entre paréntesis.
- Coloque la subconsulta a la derecha de la condición de comparación para una mejor legibilidad.
- Antes de la versión Oracle8i, las subconsultas no podían contener una cláusula ORDER BY. Sólo se puede utilizar una cláusula ORDER BY para una sentencia SELECT y, si se especifica, debe ser la última cláusula de la sentencia SELECT principal. A partir de la versión Oracle8i, se puede utilizar una cláusula ORDER BY y es necesaria en la subconsulta para realizar un análisis N principal.
- En las subconsultas se utilizan dos clases de condiciones de comparación: operadores de una sola fila y operadores de varias filas.

TIPOS DE SUBCONSULTAS

- Subconsultas de una sola fila: Consultas que devuelven una sola fila a partir de la sentencia SELECT interna.
- Subconsultas de varias filas: Consultas que devuelven más de una fila a partir de la sentencia SELECT interna.

Subconsultas de una Sola Fila

Una subconsulta de una sola fila es aquella que devuelve una fila a partir de la sentencia SELECT interna. Este tipo de subconsulta utiliza un operador de una sola fila. La transparencia proporciona una lista de operadores de una sola fila.

Ejemplo

Muestre los empleados cuyos identificadores de cargo sean los mismos que el del empleado 141.

```

1 SELECT last_name, job_id
2   FROM EMPLOYEES
3   WHERE job_id =
4         (SELECT job_id
5          FROM employees
6          WHERE employee_id = 141)
SQL> /

```

LAST_NAME	JOB_ID
Nayer	ST_CLERK
Mikkilineni	ST_CLERK
Landry	ST_CLERK
Markle	ST_CLERK
Bissot	ST_CLERK
Atkinson	ST_CLERK
Marlow	ST_CLERK
Olson	ST_CLERK
Mallin	ST_CLERK
Rogers	ST_CLERK
Gee	ST_CLERK

Los operadores de una sola fila son:

=	igual
>	mayor que
<	menor que
>=	mayor o igual
<=	menor o igual
<>	diferente

Se puede utilizar más de una subconsulta dentro de una sentencia SELECT según se necesite.

```

SELECT last_name, job_id, salary
FROM employees
WHERE job_id =
      (SELECT job_id
       FROM employees
       WHERE employee_id = 141)
AND salary >
      (SELECT salary
       FROM employees
       WHERE employee_id = 143);

```

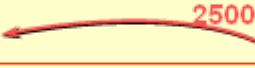
Diagrama de flujo: Se muestra la consulta principal con dos subconsultas. La primera subconsulta (job_id) devuelve 'ST CLERK' y la segunda subconsulta (salary) devuelve '2600'. Las flechas indican que estos valores se comparan con los campos correspondientes de la consulta principal.

En el ejemplo se consultan los nombres, cargos y salarios de aquellos que tienen el mismo cargo que el empleado 141 y que ganan más que el empleado 143.

Uso de Funciones de Grupo en una Subconsulta

Puede mostrar datos de una consulta principal utilizando una función de grupo en una subconsulta para devolver una sola fila. La subconsulta está entre paréntesis y colocada detrás de la condición de comparación.


```
SELECT last_name, job_id, salary
FROM employees
WHERE salary = (SELECT MIN(salary)
                FROM employees);
```



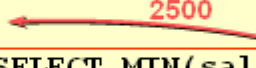
En el ejemplo se muestran los nombres, cargos y sueldo de aquellos que ganan el sueldo menor.

La Cláusula HAVING con Subconsultas

Puede utilizar subconsultas no sólo en la cláusula WHERE, sino también en la cláusula HAVING. Oracle Server ejecuta la subconsulta y los resultados se devuelven a la cláusula HAVING de la consulta principal.

La sentencia SQL del ejemplo muestra todos los departamentos que tienen un salario mínimo mayor que el del departamento 50.

```
SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) > (SELECT MIN(salary)
                      FROM employees
                      WHERE department_id = 50);
```



Subconsultas de Varias Filas

Las subconsultas que devuelven más de una fila se denominan subconsultas de varias filas. Se utiliza un operador de varias filas, en lugar de uno de una sola fila, en una subconsulta de varias filas. El operador de varias filas espera un valor o varios.

Ejemplo

Busque los empleados que ganan el mismo salario que el mínimo de cada departamento.

La consulta interna se ejecuta en primer lugar y proporciona un resultado de consulta. A continuación, se procesa el bloque de consulta principal y utiliza los valores devueltos por la consulta interna para completar su condición de búsqueda. En realidad, la consulta principal aparecería en Oracle Server como se indica a continuación:

```

1 SELECT LAST_NAME, SALARY, DEPARTMENT_ID
2 FROM EMPLOYEES
3* WHERE SALARY IN (SELECT MIN(SALARY) FROM EMPLOYEES GROUP BY DEPARTMENT_ID)
SQL> /

```

LAST_NAME	SALARY	DEPARTMENT_ID
Kochhar	17000	90
De Haan	17000	90
Ernst	6000	60
Lorentz	4200	60
Popp	6900	100
Colmenares	2500	30
Vollman	6500	50
Marlow	2500	50
Olson	2100	50
Patel	2500	50
Vargas	2500	50

Los operadores que se emplean en subconsultas con varias filas son:

Operador	Significado
IN	Igual a cualquier miembro de la lista
ANY	Compara el valor con cada valor devuelto por la subconsulta
ALL	Compara el valor con todos los valores devueltos por la subconsulta

El operador ANY (y su sinónimo, el operador SOME) compara un valor con cada valor devuelto por una subconsulta. El ejemplo de la transparencia muestra los empleados que no son programadores de IT y cuyos salarios son menores que los de cualquier programador de IT. El salario máximo que gana un programador es de \$9.000.

<ANY significa menos que el máximo.

>ANY significa más que el mínimo.

=ANY es equivalente a IN.

<ALL significa menos que el máximo.

>ALL significa más que el mínimo.

Ejemplo

```

SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ANY
      (SELECT salary
       FROM employees
       WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';

```

9000, 6000, 4200

Muestra aquellos cuyo sueldo es menor que alguno de los de la lista.

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL
      (SELECT salary
       FROM employees
       WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

9000, 6000, 4200

Muestra aquellos cuyo sueldo es mayor que todos los de la lista.

Devolución de Valores Nulos en el Juego Resultante de una Subconsulta

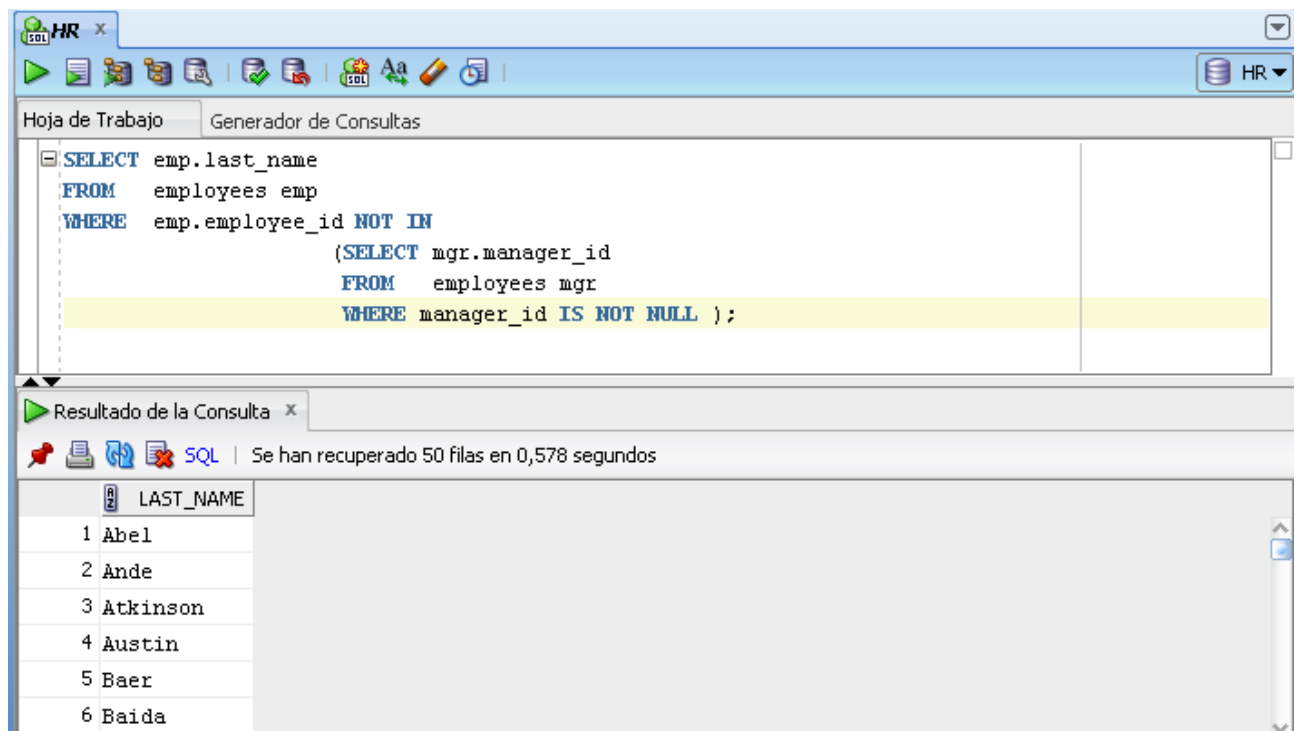
La sentencia SQL del ejemplo intenta mostrar todos los empleados que no tienen subordinados. Lógicamente, esta sentencia SQL debería haber devuelto 12 filas. Sin embargo, no devuelve ninguna. Uno de los valores devueltos por la consulta interna es un valor nulo, por lo que la consulta entera no devuelve ninguna fila. El motivo es que todas las condiciones que comparen un valor nulo producen un valor nulo. Por ello, si es posible que en el juego de resultados de una subconsulta haya valores nulos, no utilice el operador NOT IN. Este operador es equivalente a <> ALL.

Observe que el valor nulo como parte del juego de resultados de una subconsulta no supone un problema si utiliza el operador IN. Este operador es equivalente a =ANY. Por ejemplo, para mostrar los empleados que tienen subordinados, utilice la siguiente sentencia SQL:

```
SELECT emp.last_name
FROM employees emp
WHERE emp.employee_id NOT IN
      (SELECT mgr.manager_id
       FROM employees mgr);
```

no rows selected

Igualmente, se puede incluir una cláusula WHERE en la subconsulta para mostrar todos los empleados que no tienen subordinados:



The screenshot shows an SQL IDE window titled 'HR'. The 'Generador de Consultas' (Query Generator) tab is active, displaying the following SQL query:

```
SELECT emp.last_name
FROM employees emp
WHERE emp.employee_id NOT IN
      (SELECT mgr.manager_id
       FROM employees mgr
       WHERE manager_id IS NOT NULL );
```

Below the query editor, the 'Resultado de la Consulta' (Query Result) tab shows the results of the query. It indicates that 50 rows were retrieved in 0.578 seconds. The results are displayed in a table with one column, 'LAST_NAME'.

	LAST_NAME
1	Abel
2	Ande
3	Atkinson
4	Austin
5	Baer
6	Baida

LOS OPERADORES DE CONJUNTOS

Los operadores SET combinan los resultados de dos o más consultas componentes en un resultado. Las consultas que contienen operadores SET se llaman consultas compuestas.

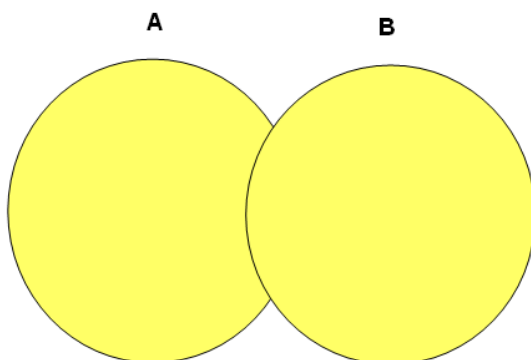
Todos los operadores SET tienen la misma prioridad. Si una sentencia SQL contiene varios operadores SET, Oracle Server los evalúa de izquierda (arriba) a derecha (abajo) si no hay paréntesis que especifiquen explícitamente otro orden. Debe utilizar paréntesis para especificar el orden de evaluación explícitamente en columnas que utilicen el operador INTERSECT con otros operadores SET.

Tablas Utilizadas en Esta Lección

Se utilizan dos tablas en esta lección: EMPLOYEES y JOB_HISTORY.

La tabla EMPLOYEES almacena detalles de empleado. Para los registros de recursos humanos, esta tabla almacena un número de identificación único y una dirección de correo electrónico para cada empleado. También se almacenan detalles del empleado de número de identificación de cargo, salario y director. Algunos de los empleados perciben una comisión además del salario; también se realiza un seguimiento de esta información. La compañía organiza los roles de los empleados en cargos. Algunos empleados llevan mucho tiempo con la compañía y han ido cambiando de cargo. Esto se controla mediante la tabla JOB_HISTORY. Cuando un empleado cambia de cargo, los detalles de las fechas inicial y final del trabajo anterior, el número de identificación de cargo y el departamento se registran en la tabla JOB_HISTORY.

El Operador UNION



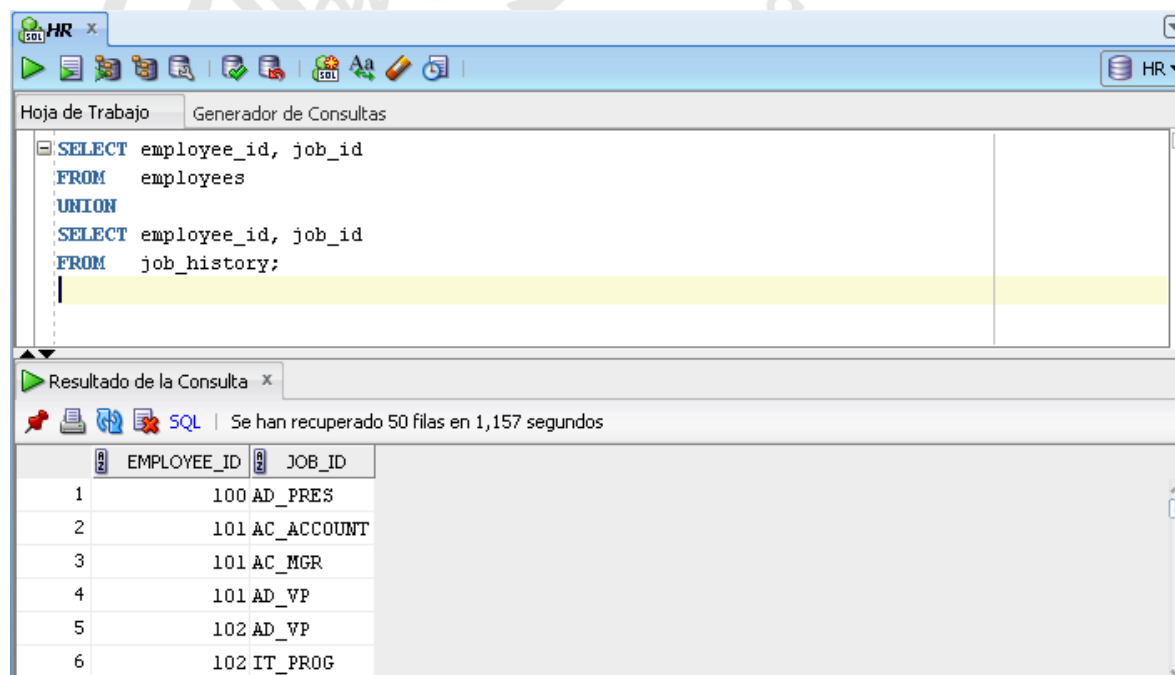
El operador UNION devuelve todas las filas seleccionadas por cualquiera de las consultas. Utilice el operador UNION para devolver todas las filas de varias tablas y eliminar las filas duplicadas.

Instrucciones

- El número de columnas y los tipos de datos de las columnas seleccionadas deben ser idénticos en todas las sentencias SELECT utilizadas en la consulta. Los nombres de las columnas no tienen que ser idénticos.
- UNION opera en todas las columnas seleccionadas.
- Los valores NULL no se ignoran durante la comprobación de duplicados.
- El operador IN tiene una prioridad más alta que el operador UNION.
- Por defecto, el resultado se ordena en orden ascendente por la primera columna de la cláusula SELECT.

El operador UNION elimina los registros duplicados. Si hay registros que se producen tanto en la tabla EMPLOYEES como en la tabla JOB_HISTORY y son idénticos, se mostrarán una sola vez. Observe en el resultado que se muestra que el registro para el empleado EMPNO 200 aparece dos veces porque su JOB_ID es diferente en cada fila.

Observe el siguiente ejemplo:



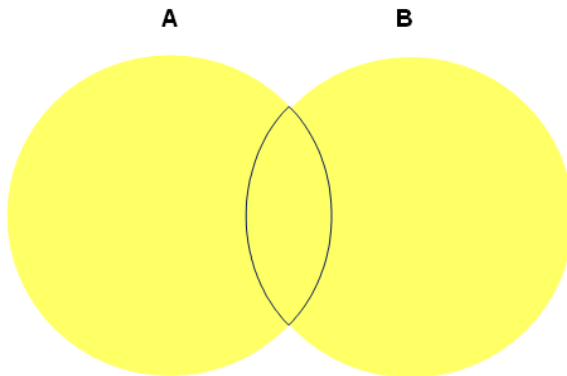
The screenshot shows an SQL IDE window titled 'HR'. The 'Hoja de Trabajo' (Worksheet) tab is active, displaying the following SQL query:

```
SELECT employee_id, job_id
FROM employees
UNION
SELECT employee_id, job_id
FROM job_history;
```

The 'Resultado de la Consulta' (Query Result) tab is also active, showing the results of the query. The status bar indicates 'Se han recuperado 50 filas en 1,157 segundos' (50 rows retrieved in 1.157 seconds). The results are displayed in a table with two columns: EMPLOYEE_ID and JOB_ID.

	EMPLOYEE_ID	JOB_ID
1	100	AD_PRES
2	101	AC_ACCOUNT
3	101	AC_MGR
4	101	AD_VP
5	102	AD_VP
6	102	IT_PROG

El Operador UNION ALL

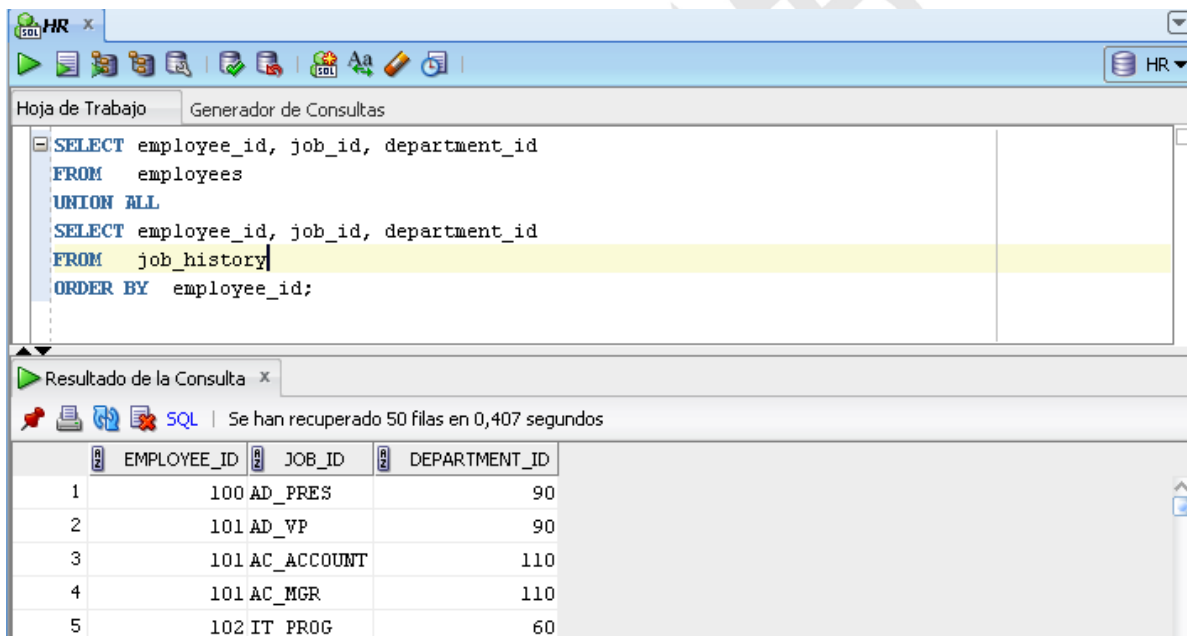


Utilice el operador UNION ALL para devolver todas las filas de varias consultas.

Instrucciones

- A diferencia de UNION, las filas duplicadas no se eliminan y el resultado no se ordena por defecto.
- No se puede utilizar la palabra clave DISTINCT.

Con la excepción de lo anteriormente expuesto, las instrucciones para UNION y UNION ALL son las mismas.



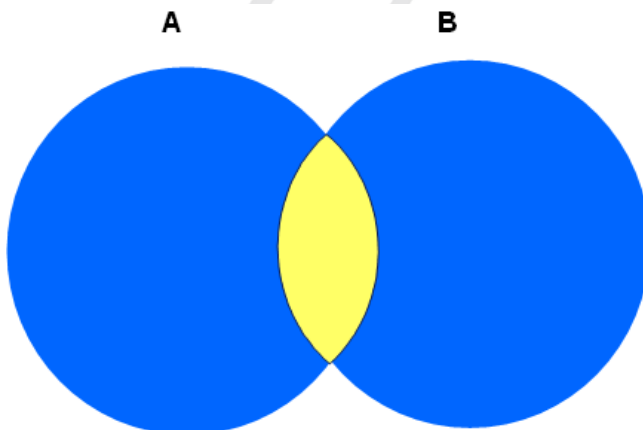
The screenshot shows the SQL Developer interface with a query window titled 'Hoja de Trabajo' and a results window titled 'Resultado de la Consulta'. The query in the window is:

```
SELECT employee_id, job_id, department_id
FROM employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM job_history
ORDER BY employee_id;
```

The results window shows the following data:

EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100 AD_PRES	90
2	101 AD_VP	90
3	101 AC_ACCOUNT	110
4	101 AC_MGR	110
5	102 IT_PROG	60

El Operador INTERSECT



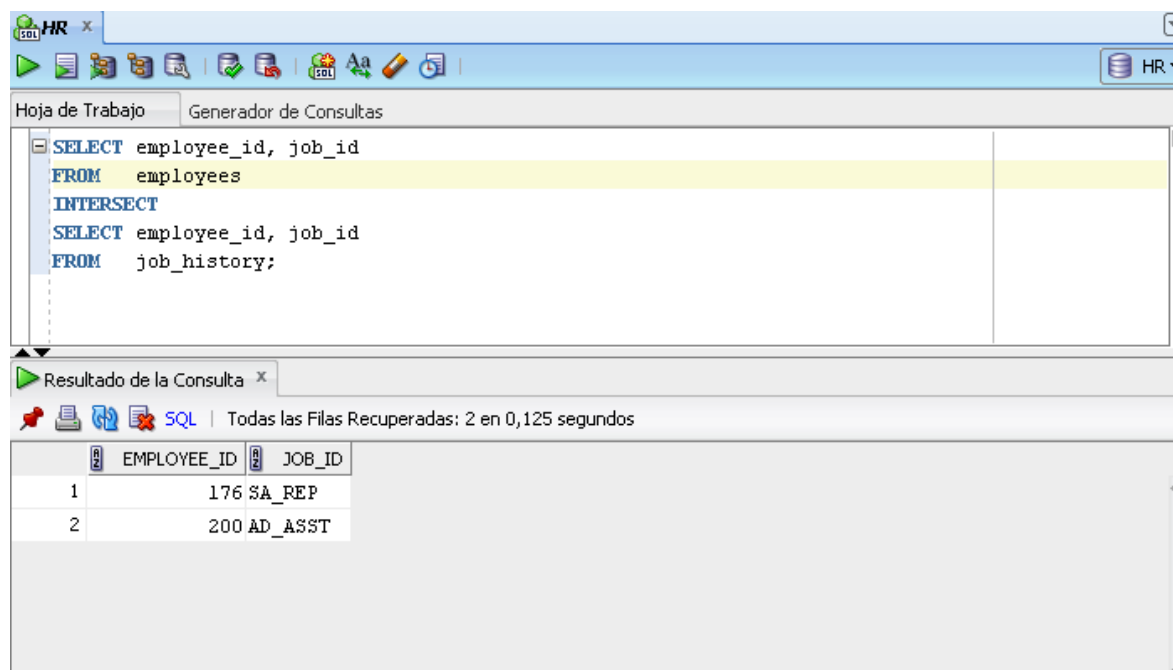
Utilice el operador INTERSECT para devolver todas las filas comunes de varias consultas.

Instrucciones

- El número de columnas y los tipos de datos de las columnas seleccionadas por las sentencias SELECT deben ser idénticos en todas las sentencias SELECT utilizadas en las consultas. Los nombres de las columnas no tienen que ser idénticos.

- Revertir el orden de las tablas intersecadas no modificará el resultado.
- INTERSECT no ignora los valores NULL.

Ejemplo



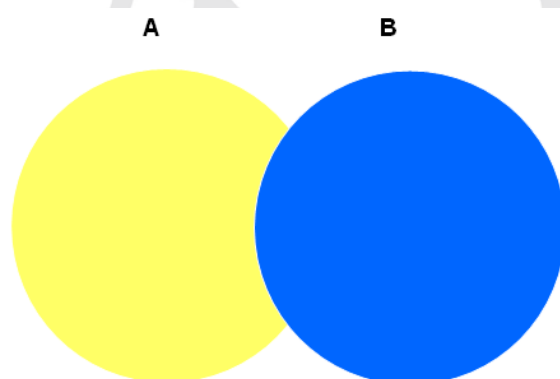
The screenshot shows the SQL Developer interface. The top pane displays the following SQL query:

```
SELECT employee_id, job_id
FROM employees
INTERSECT
SELECT employee_id, job_id
FROM job_history;
```

The bottom pane shows the results of the query in a table with two columns: EMPLOYEE_ID and JOB_ID. The results are as follows:

	EMPLOYEE_ID	JOB_ID
1	176	SA_REP
2	200	AD_ASST

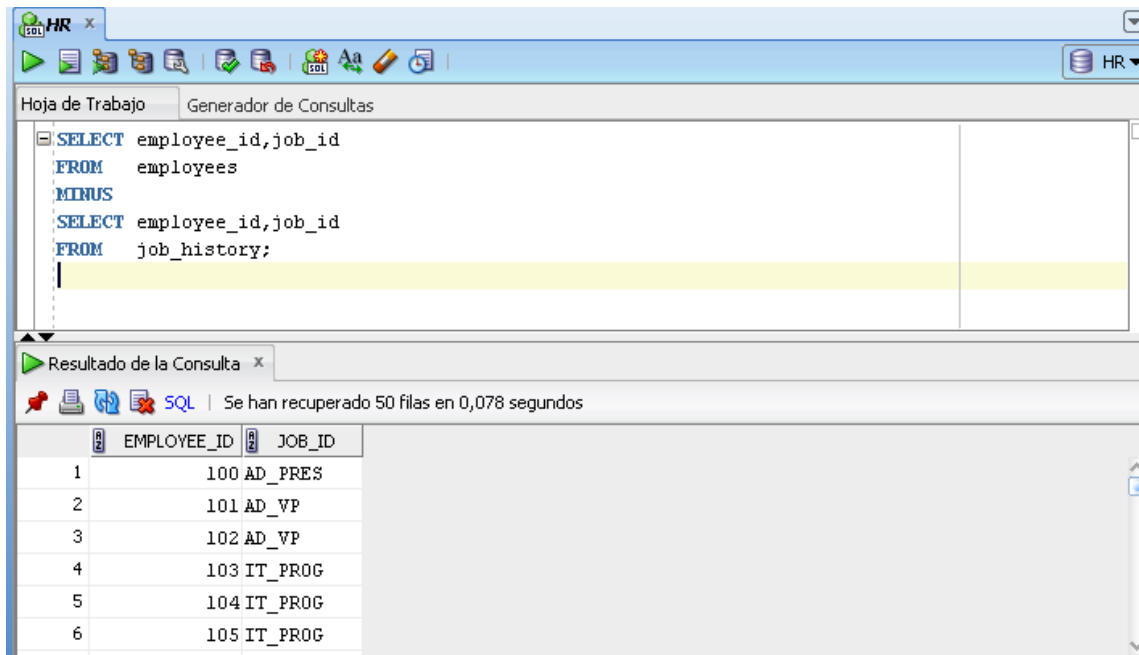
El Operador MINUS



Utilice el operador MINUS para devolver filas devueltas por la primera consulta que no estén presentes en la segunda (la primera sentencia SELECT menos (MINUS) la segunda sentencia SELECT).

Instrucciones

- El número de columnas y los tipos de datos de las columnas seleccionadas por las sentencias SELECT deben ser idénticos en todas las sentencias SELECT utilizadas en las consultas. Los nombres de las columnas no tienen que ser idénticos.
- Todas las columnas de la cláusula WHERE deben estar en la cláusula SELECT para que el operador MINUS funcione.



The screenshot shows the SQL Developer interface. The top pane, titled 'Hoja de Trabajo' and 'Generador de Consultas', contains the following SQL query:

```
SELECT employee_id, job_id
FROM employees
MINUS
SELECT employee_id, job_id
FROM job_history;
```

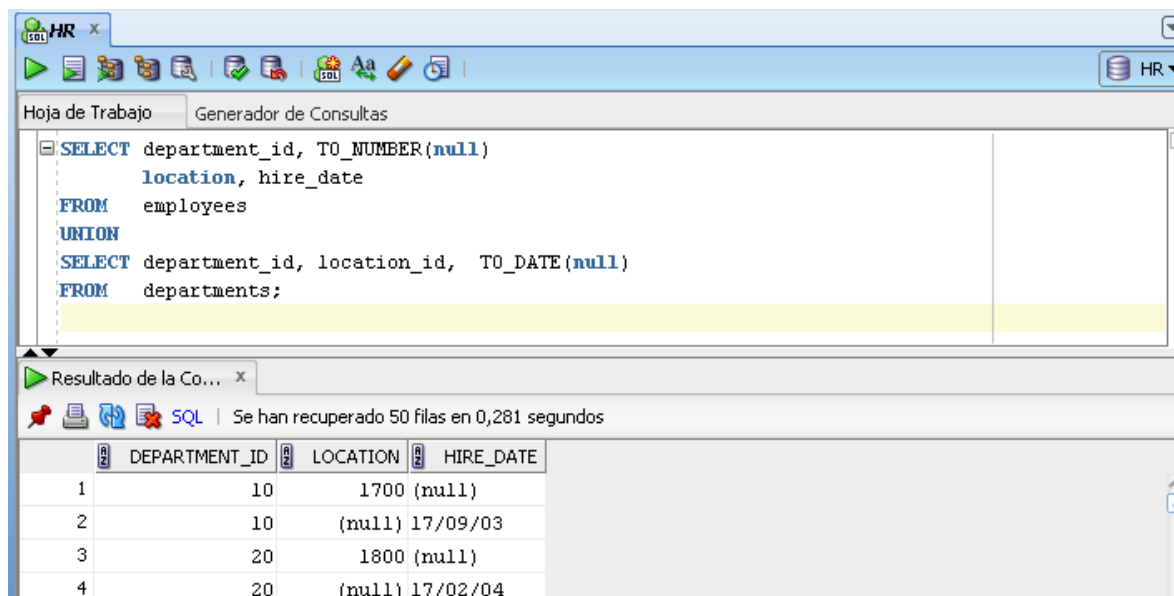
The bottom pane, titled 'Resultado de la Consulta', shows the execution results. It indicates that 50 rows were retrieved in 0.078 seconds. The results are displayed in a table with two columns: EMPLOYEE_ID and JOB_ID.

	EMPLOYEE_ID	JOB_ID
1	100	AD_PRES
2	101	AD_VP
3	102	AD_VP
4	103	IT_PROG
5	104	IT_PROG
6	105	IT_PROG

Instrucciones para Operadores SET

- Las expresiones de las listas SELECT deben coincidir en número y tipo de datos.
- Se pueden utilizar paréntesis para modificar la secuencia de ejecución.
- La cláusula ORDER BY:
 - Puede aparecer sólo justo al final de la sentencia.
 - Aceptará el nombre de columna, los alias de la primera sentencia SELECT o la notación posicional.
- Las filas duplicadas se eliminan automáticamente excepto en UNION ALL.
- Los nombres de columna de la primera consulta aparecen en el resultado.
- El resultado se ordena en orden ascendente por defecto excepto en UNION ALL.

Cuando no coincide la cantidad de columnas o los tipos de datos se puede optar por completarlos con valores NULL.



The screenshot shows the SQL Developer interface with a query window titled 'Hoja de Trabajo' and 'Generador de Consultas'. The query is as follows:

```

SELECT department_id, TO_NUMBER(null)
      location, hire_date
FROM   employees
UNION
SELECT department_id, location_id, TO_DATE(null)
FROM   departments;

```

Below the query, the 'Resultado de la Co...' window shows the results of the query. It indicates that 50 rows were retrieved in 0.281 seconds. The results are displayed in a table with the following columns: DEPARTMENT_ID, LOCATION, and HIRE_DATE.

	DEPARTMENT_ID	LOCATION	HIRE_DATE
1	10	1700 (null)	
2	10	(null) 17/09/03	
3	20	1800 (null)	
4	20	(null) 17/02/04	

También se puede completar con otros valores literales como en este ejemplo:

```

SELECT employee_id, job_id, salary
FROM   employees
UNION
SELECT employee_id, job_id, 0
FROM   job_history;

```

Toda sentencia que tenga UNION solo puede tener un ORDER BY, al final de todas las cláusulas.

Ejercicio práctico

1. Escriba una consulta que muestre el apellido y la fecha de contratación de cualquier empleado del mismo departamento que Zlotkey. Excluya a Zlotkey.
2. Cree una consulta para mostrar los números de empleado y los apellidos de todos los empleados que ganen más del salario medio. Ordene los resultados por salario en orden ascendente.
3. Escriba una consulta que muestre los números de empleado y los apellidos de todos los empleados que trabajen en un departamento con cualquier empleado cuyo apellido contenga una u. Coloque la sentencia SQL en un archivo de texto llamado lab3_3.sql. Ejecute la consulta.
4. Muestre el apellido, el número de departamento y el identificador de cargo de todos los empleados cuyos identificadores de ubicación de departamento sean 1700.
5. Muestre el apellido y el salario de todos los empleados que informen a King.
6. Muestre el número de departamento, el apellido y el identificador de cargo de todos los empleados del departamento Executive.
7. Modifique la consulta en lab3_3.sql para mostrar los números de empleado, los apellidos y los salarios de todos los empleados que ganan más del salario medio y que trabajan en un departamento con un empleado que tenga una u en su apellido.
8. Vuelva a guardar lab3_3.sql como lab3_7.sql. Ejecute la sentencia en lab3_7.sql.
9. Enumere los identificadores de departamento para departamentos que no contienen el cargo ST_CLERK, mediante operadores SET.
10. Mediante operadores SET, visualice el identificador de país y el nombre de los países que no tengan departamentos ubicados en ellos.
11. Cree una lista de puestos para los departamentos 10, 50 y 20, en ese orden. Visualice el identificador de cargo y de departamento, mediante operadores SET.
12. Enumere los identificadores de empleado y de cargo de los empleados que actualmente tengan el cargo que ocupaban antes de comenzar a trabajar con la compañía.
13. Escriba una consulta compuesta que enumere:

Apellidos e identificadores de departamento de todos los empleados de la tabla EMPLOYEES independientemente de si pertenecen o no a algún departamento

CAPÍTULO 4

MANIPULACIÓN DE DATOS EN LAS TABLAS

Contenido del Capítulo:

1. Sentencias DML.
2. Comando INSERT.
3. Comando UPDATE.
4. Comando DELETE.
5. Comando MERGE.
6. Control de transacciones

Lenguaje de Manipulación de Datos

El lenguaje de manipulación de datos (DML) es una pieza central de SQL. Cuando desee agregar, actualizar o suprimir datos de la base de datos, ejecute una sentencia DML. Una recopilación de sentencias DML que forma una unidad de trabajo lógica se llama transacción.

Considere la base de datos de un banco. Cuando un cliente del banco transfiere dinero de una cuenta de ahorros a una de cheques, la transacción puede constar de tres operaciones distintas: disminución de la cuenta de ahorros, aumento de la cuenta de cheques y registro de la transacción en el diario de transacciones. Oracle Server debe garantizar que las tres sentencias SQL se realizan para mantener las cuentas en el equilibrio correcto. Si algo evita la ejecución de una de las sentencias de la transacción, las otras sentencias se deben deshacer.

Una transacción consta de una recopilación de sentencias DML que forman una unidad de trabajo lógica.

Insertar datos

```
INSERT INTO    table [(column [, column...)]]  
VALUES        (value [, value...]);
```

Puede agregar filas nuevas a una tabla emitiendo la sentencia INSERT.

En la sintaxis:

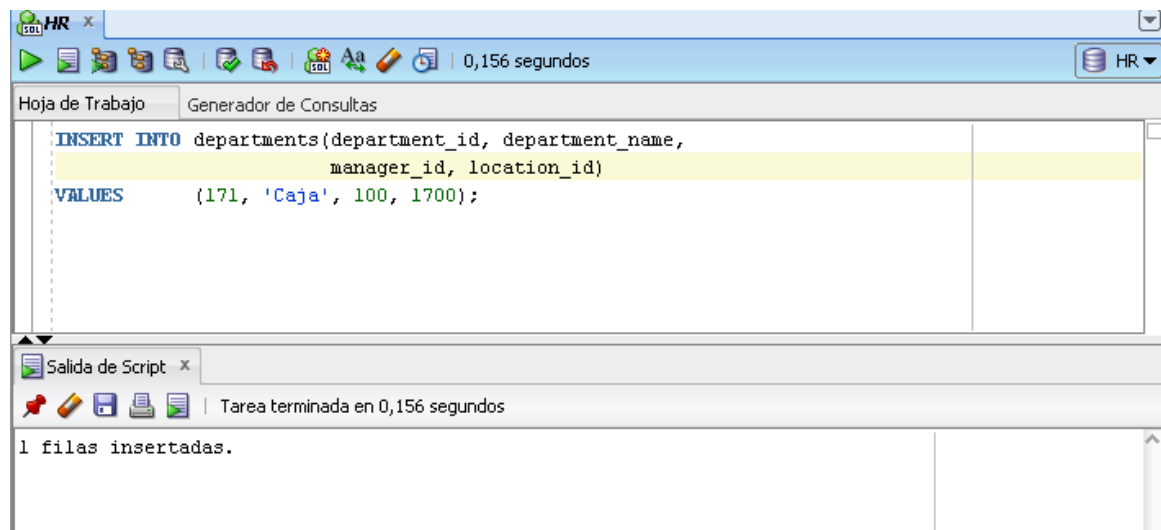
<i>table</i>	es el nombre de la tabla.
<i>column</i>	es el nombre de la columna de la tabla que se ha de rellenar.
<i>value</i>	es el valor correspondiente para la columna.

Esta sentencia con la cláusula VALUES sólo agrega una fila cada vez a una tabla.

Instrucciones

- Inserte una fila nueva que contenga valores para cada columna.
- Enumere los valores en el orden por defecto de las columnas de la tabla.
- Opcionalmente, liste las columnas en la cláusula INSERT.
- Escriba los valores de caracteres y de fecha entre comillas simples.

Ejemplo



The screenshot shows the SQL Developer interface. The top toolbar includes icons for running queries, saving, and other database functions. The main window displays the following SQL statement:

```
INSERT INTO departments(department_id, department_name,
                        manager_id, location_id)
VALUES      (171, 'Caja', 100, 1700);
```

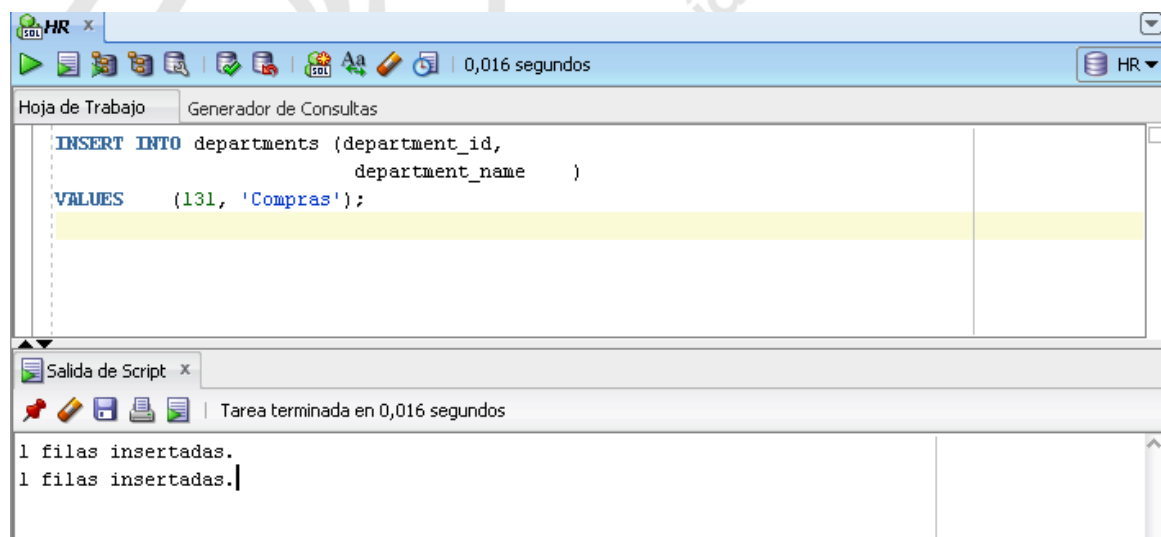
Below the SQL editor, the 'Salida de Script' (Script Output) window shows the execution result:

```
1 filas insertadas.
```

Los valores nulos

Para agregar datos nulos a las columnas de las tablas que lo permitan hay dos métodos:

Método implícito: Omite la columna de la lista de columnas.



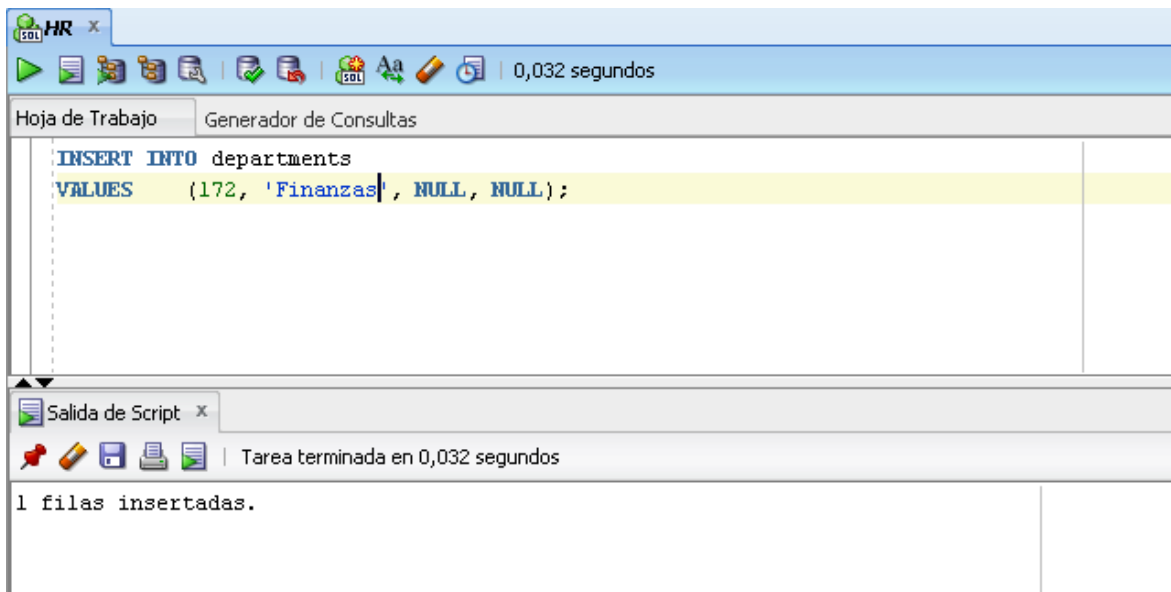
The screenshot shows the SQL Developer interface. The main window displays the following SQL statement:

```
INSERT INTO departments (department_id,
                        department_name )
VALUES      (131, 'Compras');
```

Below the SQL editor, the 'Salida de Script' (Script Output) window shows the execution result:

```
1 filas insertadas.
1 filas insertadas.
```

Método explícito: Especifique la palabra clave NULL en la cláusula VALUES.



Asegúrese de que puede utilizar valores nulos en la columna de destino verificando el estado de Null? con el comando DESCRIBE.

Oracle Server fuerza automáticamente todos los tipos de dato, rangos de datos y restricciones de integridad de datos. Toda columna no listada explícitamente obtiene un valor nulo en la fila nueva.

Errores habituales que se pueden producir durante la entrada del usuario. Estos errores tienen que ver con las reglas de integridad de una base de datos que se definirán en las siguientes lecciones:

- Falta un valor obligatorio para una columna NOT NULL.
- Un valor duplicado viola la restricción de unicidad.
- Se viola una restricción de clave ajena.
- Se viola una restricción CHECK.
- El tipo de dato no coincide.
- El valor es demasiado ancho para la columna.

Ejemplos de inserción

```
INSERT INTO employees (employee_id,  
                        first_name, last_name,  
                        email, phone_number,  
                        hire_date, job_id, salary,  
                        commission_pct, manager_id,  
                        department_id)  
VALUES  
    (113,  
     'Louis ', 'Popp',  
     'LPOPP ', '515.124.4567',  
     SYSDATE, 'AC_ACCOUNT', 6900,  
     NULL, 205, 100);  
  
1 row created.
```


El formato DD-MON-YY se utiliza normalmente para insertar un valor de fecha.

Con este formato, recuerde que el siglo por defecto es el actual. Como la fecha también contiene información de hora, la hora por defecto es la medianoche (00:00:00).

```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
             TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
             'AC_ACCOUNT', 11000, NULL, 100, 30);

1 row created.
```

Si se debe introducir una fecha en un formato distinto del formato por defecto, por ejemplo, con otro siglo, o una hora específica, debe utilizar la función TO_DATE.

En el ejemplo se registra información para el empleado Raphealy en la tabla EMPLOYEES y se define la columna HIREDATE en 3 de febrero de 1999. Si utiliza la siguiente sentencia en lugar de la mostrada en la transparencia, el año de hiredate se interpreta como 2099.

Copia de Filas desde otra Tabla

Puede utilizar la sentencia INSERT para agregar filas a una tabla en la que los valores se derivan desde tablas existentes. En lugar de la cláusula VALUES, utilice una subconsulta.

Sintaxis

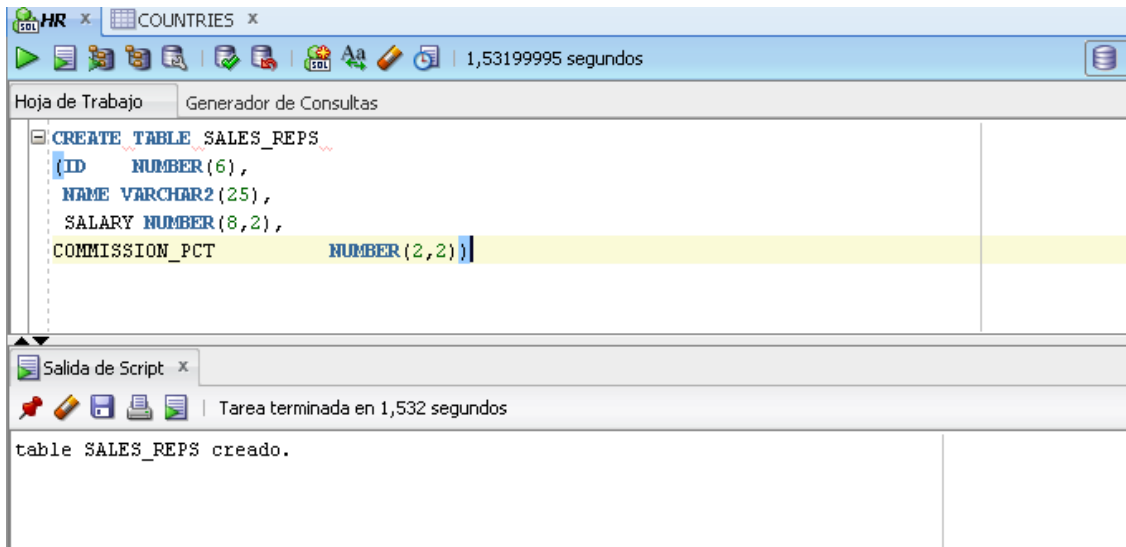
```
INSERT INTO table [ column (, column) ] subquery;
```

En la sintaxis:

- | | |
|----------|---|
| table | es el nombre de la tabla. |
| column | es el nombre de la columna de la tabla que se ha de rellenar. |
| subquery | es la subconsulta que devuelve filas a la tabla. |

El número de columnas y sus tipos de dato en la lista de columnas de la cláusula INSERT debe coincidir con el número de valores y sus tipos de dato en la subconsulta. Para crear una copia de las filas de una tabla, utilice SELECT * en la subconsulta.

Para poder realizar el ejemplo de este caso primero debemos crear una tabla vacía. Para ello utilizaremos el comando CREATE TABLE como se observa a continuación.



HR x COUNTRIES x
1,53199995 segundos

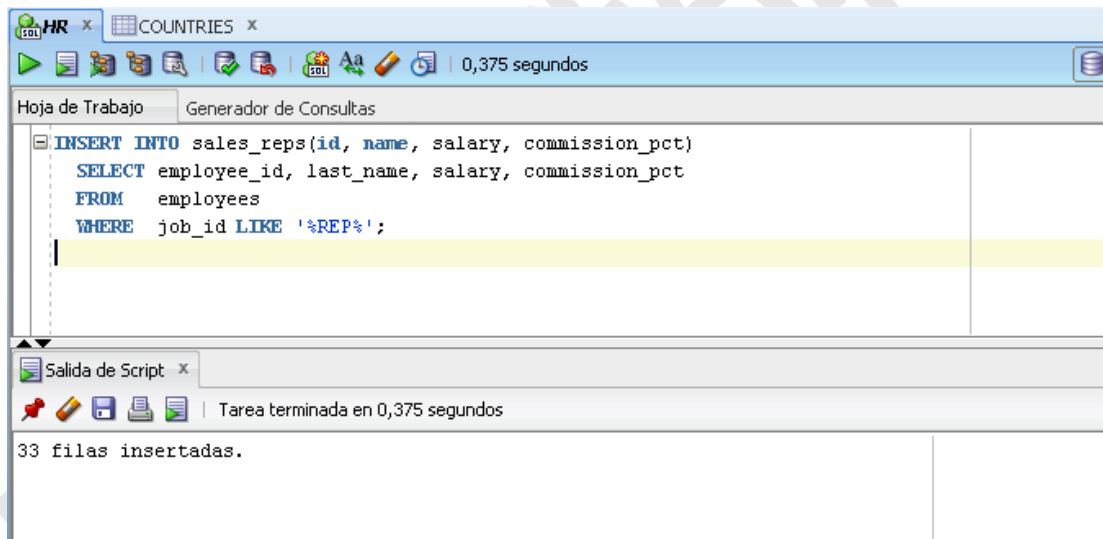
Hoja de Trabajo Generador de Consultas

```
CREATE TABLE SALES_REPS
(
  ID NUMBER(6),
  NAME VARCHAR2(25),
  SALARY NUMBER(8,2),
  COMMISSION_PCT NUMBER(2,2)
)
```

Salida de Script x
Tarea terminada en 1,532 segundos

table SALES_REPS creado.

Luego se podrá realizar la inserción masiva de datos a partir de una subconsulta de otra tabla.



HR x COUNTRIES x
0,375 segundos

Hoja de Trabajo Generador de Consultas

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

Salida de Script x
Tarea terminada en 0,375 segundos

33 filas insertadas.

MODIFICACIÓN DE DATOS EN UNA TABLA

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

Puede modificar las filas existentes utilizando la sentencia UPDATE.

En la sintaxis:

table es el nombre de la tabla.

column es el nombre de la columna de la tabla que se ha de rellenar.

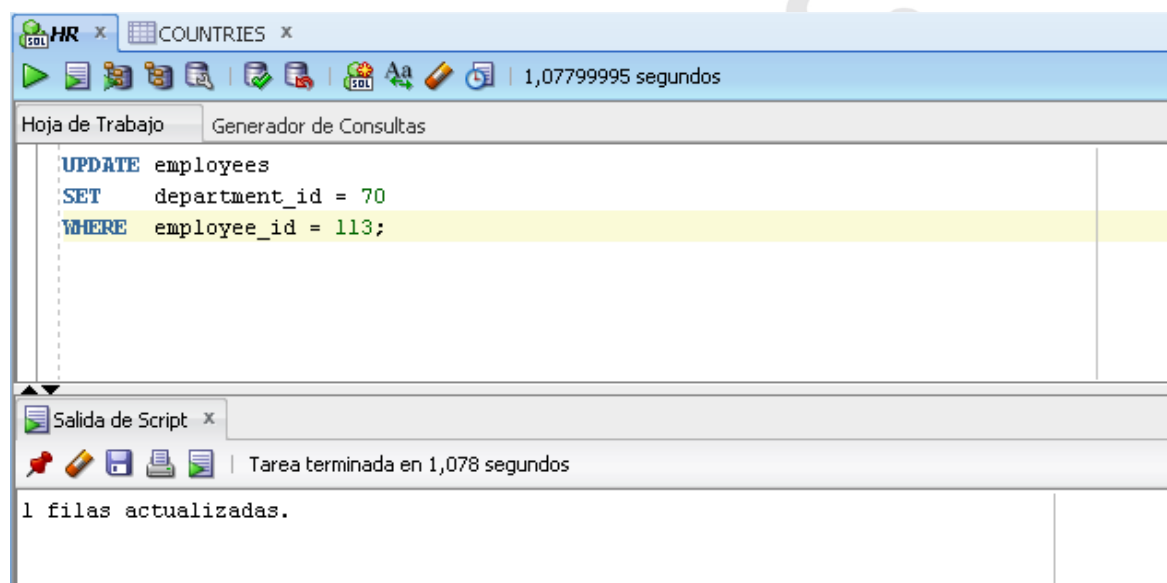
value es la subconsulta o el valor correspondiente para la columna.

condition identifica las filas que se han de actualizar y está compuesta de expresiones de nombres de columna, constantes, subconsultas y operadores de comparación.

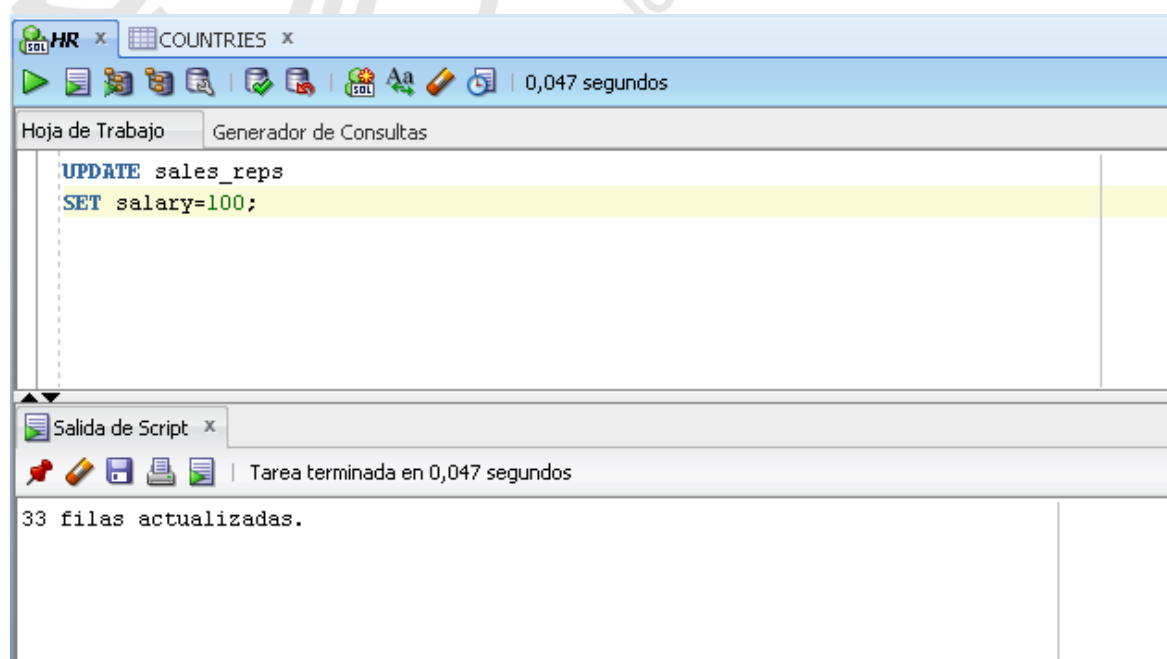
Confirme la operación de actualización mediante la consulta a la tabla para mostrar las filas actualizadas.

En general, utilice la clave primaria para identificar una sola fila. Si usa otras columnas se pueden actualizar varias filas de forma inesperada. Por ejemplo, identificar una sola fila de la tabla EMPLOYEES por el nombre es peligroso, ya que es posible que varios empleados tengan el mismo nombre.

Si incluye la cláusula WHERE, las filas específicas se modifican.



Se modifican todas las filas de la tabla si omite la cláusula WHERE.



ACTUALIZACIÓN DE DOS COLUMNAS CON UNA SUBCONSULTA

Puede actualizar varias columnas en la cláusula SET de una sentencia UPDATE escribiendo varias subconsultas.

Sintaxis

```
UPDATE table
SET   column = (SELECT column
                FROM table
                WHERE condition)
[ , column = (SELECT column
                FROM table
                WHERE condition)]
[WHERE condition];
```

Ejemplo

```
UPDATE employees
SET   job_id = (SELECT job_id
                FROM employees
                WHERE employee_id = 205),
      salary = (SELECT salary
                FROM employees
                WHERE employee_id = 205)
WHERE employee_id = 114;
1 row updated.
```

ACTUALIZACIÓN DE FILAS BASÁNDOSE EN OTRA TABLA

Puede utilizar subconsultas en sentencias UPDATE para actualizar las filas de una tabla. En el ejemplo de la transparencia se actualiza la tabla COPY_EMPLOYEES basándose en los valores de la tabla EMP, se cambia el número de departamento de todos los empleados con el identificador de cargo del empleado 200 por el número de departamento actual del empleado 100.

```
UPDATE copy_emp
SET   department_id = (SELECT department_id
                       FROM employees
                       WHERE employee_id = 100)
WHERE job_id = (SELECT job_id
                 FROM employees
                 WHERE employee_id = 200);
1 row updated.
```

ELIMINACIÓN DE FILAS DE UNA TABLA

Puede eliminar las filas existentes utilizando la sentencia DELETE.

```
DELETE [FROM]    table
[WHERE           condition];
```

En la sintaxis:

table es el nombre de la tabla.

condition identifica las filas que se van a suprimir y está compuesta de nombres de columna, expresiones, constantes, subconsultas y operadores de comparación.

Puede suprimir filas específicas incluyendo la cláusula WHERE en la sentencia DELETE.

En el ejemplo se suprime el departamento Finance de la tabla DEPT. Puede confirmar la operación de supresión si visualiza las filas suprimidas utilizando la sentencia SELECT.

```
DELETE FROM departments
WHERE department_name = 'Finance';
1 row deleted.
```

Si omite la cláusula WHERE, todas las filas de la tabla se suprimen. En el segundo ejemplo se suprimen todas las filas de la tabla COPY_EMP, porque no se ha especificado ninguna cláusula WHERE.

```
DELETE FROM copy_emp;
22 rows deleted.
```

Supresión de Filas Basándose en otra Tabla

Puede utilizar subconsultas para suprimir filas de una tabla basándose en valores de otra tabla. En el ejemplo se suprimen todos los empleados que están en un departamento cuyo nombre contiene la cadena "Public". La subconsulta busca en la tabla DEPARTMENTS el número de departamento basándose en el nombre de departamento que contiene la cadena "Public".

A continuación, proporciona el número de departamento a la consulta principal, la cual suprime filas de datos de la tabla EMPLOYEES basándose en este número de departamento.

```
DELETE FROM employees
WHERE department_id =
    (SELECT department_id
     FROM departments
     WHERE department_name LIKE '%Public%');
1 row deleted.
```

USO DE UNA SUBCONSULTA EN UNA SENTENCIA INSERT

Puede utilizar una subconsulta en lugar del nombre de tabla en la cláusula INTO de la sentencia INSERT.

La lista de selección de esta subconsulta debe tener el mismo número de columnas que la lista de columnas de la cláusula VALUES. Se deben seguir todas las reglas de las columnas de la tabla base para que la sentencia INSERT funcione correctamente. Por ejemplo, no puede poner un identificador de empleado duplicado, ni dejar fuera un valor para una columna obligatoria no nula.

```
INSERT INTO
    (SELECT employee_id, last_name,
         email, hire_date, job_id, salary,
         department_id
     FROM   employees
     WHERE  department_id = 50)
VALUES (99999, 'Taylor', 'DTAYLOR',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ST_CLERK', 5000, 50);

1 row created.
```

La Palabra Clave WITH CHECK OPTION

Especifique WITH CHECK OPTION para indicar que, si se utiliza la subconsulta en lugar de una tabla en una sentencia INSERT, UPDATE o DELETE, en la tabla no se permitirán cambios que produzcan filas que no estén incluidas en la subconsulta.

En el ejemplo mostrado, se utiliza la palabra clave WITH CHECK OPTION. La subconsulta identifica las filas que están en el departamento 50, pero el identificador de departamento no se encuentra en la lista SELECT y no se le proporciona un valor en la lista VALUES. La inserción de esta fila daría como resultado un identificador de departamento de valor nulo, que no se encuentra en la subconsulta.

```
INSERT INTO (SELECT employee_id, last_name, email,
                  hire_date, job_id, salary
             FROM   employees
             WHERE  department_id = 50 WITH CHECK OPTION)
VALUES (99998, 'Smith', 'JSMITH',
        TO_DATE('07-JUN-99', 'DD-MON-RR'),
        'ST_CLERK', 5000);

INSERT INTO
*
ERROR at line 1:
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Valores por Defecto Explícitos

- Con la función valor por defecto explícito, puede utilizar la palabra clave DEFAULT como valor de columna donde desee el valor por defecto de columna.
- Esta función se agrega para asegurar el cumplimiento con el estándar SQL: 1999.

- Esto permite al usuario controlar dónde y cuándo se debe aplicar el valor por defecto a los datos.
- Se pueden utilizar valores por defecto explícitos en sentencias INSERT y UPDATE.

Uso de Valores por Defecto Explícitos

Especifique DEFAULT para definir la columna en el valor especificado previamente como valor por defecto para la columna. Si no se ha especificado ningún valor por defecto para la columna correspondiente, Oracle define la columna en un valor nulo.

En el primer ejemplo mostrado, la sentencia INSERT utiliza un valor por defecto para la columna MGR. Si no hay ningún valor por defecto definido para la columna, se inserta en su lugar un valor nulo.

En el segundo ejemplo se utiliza la sentencia UPDATE para definir la columna MGR en un valor por defecto para el departamento 10. Si no hay ningún valor por defecto definido para la columna, cambia el valor a nulo.

Ejemplo.

```
INSERT INTO departments
  (department_id, department_name, manager_id)
VALUES (300, 'Engineering', DEFAULT);
```

Cuando se hace una modificación también se puede activar el valor por defecto.

```
UPDATE departments
SET manager_id = DEFAULT WHERE department_id = 10;
```

SENTENCIAS MERGE

SQL se ha extendido para incluir la sentencia MERGE.

Con esta sentencia, puede actualizar o insertar una fila condicionalmente en una tabla, evitando así tener múltiples sentencias UPDATE. La decisión sobre si actualizar o insertar en la tabla de destino se basa en una condición en la cláusula ON.

Como el comando MERGE combina los comandos INSERT y UPDATE, necesita los dos privilegios INSERT y UPDATE en la tabla de destino y el privilegio SELECT en la tabla de origen.

La sentencia MERGE es determinista. No puede actualizar la misma fila de la tabla de destino varias veces en la misma sentencia MERGE.

Un enfoque alternativo es utilizar bucles PL/SQL y varias sentencias DML. Sin embargo, la sentencia MERGE es fácil de utilizar y se expresa de forma más simple como una sola sentencia SQL.

La sentencia MERGE es útil en diversas aplicaciones de almacenes de datos. Por ejemplo, en una aplicación de almacenes de datos, tal vez sea necesario trabajar con datos procedentes de varios orígenes, algunos de los cuales pueden estar duplicados. Con la sentencia MERGE, puede agregar o modificar filas condicionalmente.


```
MERGE INTO table_name table_alias
  USING (table/view/sub_query) alias
  ON (join condition)
  WHEN MATCHED THEN
    UPDATE SET
      col1 = col_val1,
      col2 = col2_val
  WHEN NOT MATCHED THEN
    INSERT (column_list)
    VALUES (column_values);
```

Puede actualizar filas existentes e insertar filas nuevas condicionalmente utilizando la sentencia MERGE.

En la sintaxis:

Cláusula INTO	especifica la tabla de destino que está actualizando o en la que está insertando.
Cláusula USING	identifica el origen de los datos que se van a actualizar o insertar; puede ser una tabla, una vista o una subconsulta.
Cláusula ON	la condición sobre la cual la operación MERGE actualiza o inserta.
WHEN MATCHED	indica al servidor cómo responder a los resultados de la condición de unión.
WHEN NOT MATCHED	

Ejemplo

```
MERGE INTO copy_emp c
  USING employees e
  ON (c.employee_id = e.employee_id)
  WHEN MATCHED THEN
    UPDATE SET
      c.first_name      = e.first_name,
      c.last_name       = e.last_name,
      ...
      c.department_id  = e.department_id
  WHEN NOT MATCHED THEN
    INSERT VALUES(e.employee_id, e.first_name, e.last_name,
      e.email, e.phone_number, e.hire_date, e.job_id,
      e.salary, e.commission_pct, e.manager_id,
      e.department_id);
```

En el ejemplo mostrado se compara EMPLOYEE_ID de la tabla COPY_EMP con EMPLOYEE_ID de la tabla EMPLOYEES. Si se encuentra una coincidencia, la fila de

la tabla COPY_EMP se actualiza para que coincida con la fila de la tabla EMPLOYEES. Si no se encuentra la fila, se inserta en la tabla COPY_EMP.

TRANSACCIONES DE BASE DE DATOS

Oracle Server asegura la consistencia de datos basándose en transacciones. Las transacciones le ofrecen más flexibilidad y control al cambiar datos y aseguran una consistencia de datos en caso de fallo del proceso de usuario o fallo del sistema.

Las transacciones constan de sentencias DML que constituyen un cambio consistente en los datos. Por ejemplo, una transferencia de fondos entre dos cuentas debe incluir el débito a una cuenta y el cargo a otra por el mismo importe. Las dos acciones se deben realizar correcta o incorrectamente al mismo tiempo, de forma que el crédito no se debe validar sin el débito.

Tipos de Transacción

- Sentencias DML, que constituyen un cambio consistente en los datos
- Una sentencia DDL
- Una sentencia DCL

¿Cuándo Empieza y Termina una Transacción?

Una transacción comienza cuando se encuentra la primera sentencia DML y termina cuando se produce una de las siguientes opciones:

- Se emite una sentencia COMMIT o ROLLBACK.
- Se emite una sentencia DDL, como por ejemplo, CREATE.
- Se emite una sentencia DCL.
- El usuario sale de la herramienta de trabajo.
- Se produce un fallo de una máquina o en el sistema.

Cuando termina un transacción, la siguiente sentencia SQL ejecutable inicia automáticamente la siguiente transacción.

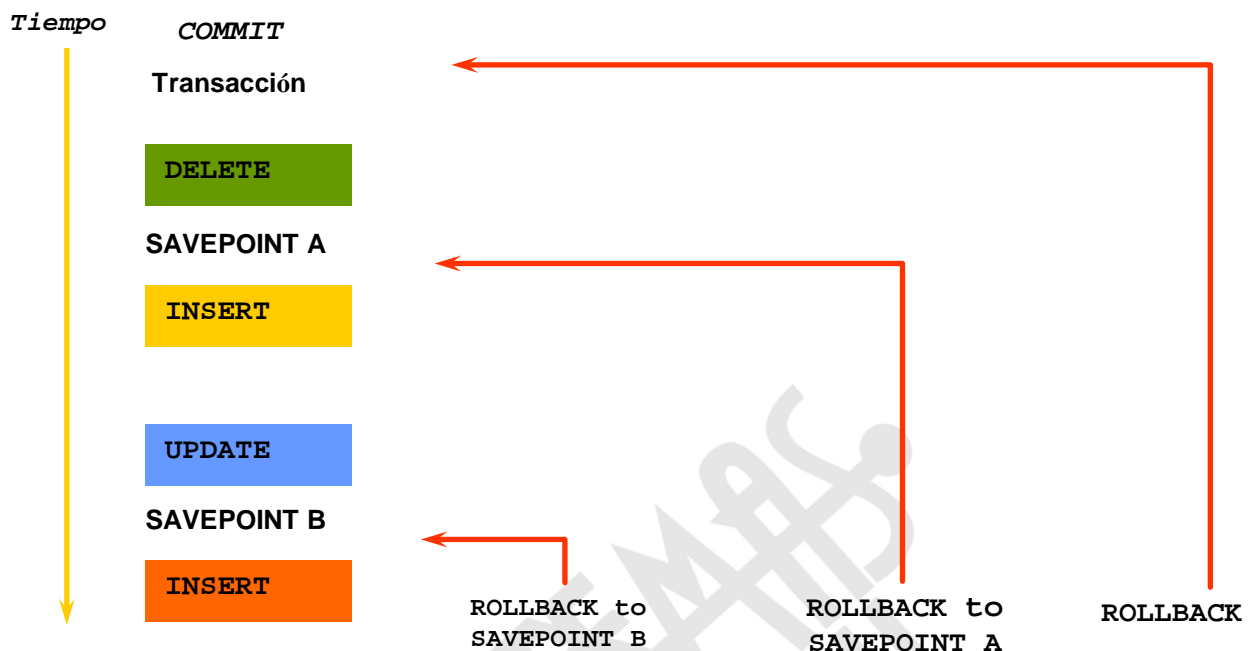
Se valida automáticamente una sentencia DDL o una sentencia DCL y, por lo tanto, finaliza implícitamente una transacción.

Con sentencias COMMIT y ROLLBACK, puede:

- Asegurar la consistencia de datos
- Realizar una presentación preliminar de los cambios de datos antes de hacer que estos sean permanentes
- Agrupar las operaciones relacionadas lógicamente

Sentencias Explícitas de Control de Transacciones

Puede controlar la lógica de las transacciones utilizando las sentencias COMMIT, SAVEPOINT y ROLLBACK.



REALIZACIÓN DE ROLLBACK DE LOS CAMBIOS A UN PUNTO DE GRABACIÓN

Puede crear un marcador en la transacción actual utilizando la sentencia **SAVEPOINT**, que divide la transacción en secciones más pequeñas. A continuación, puede desechar los cambios pendientes hasta dicho marcador utilizando la sentencia **ROLLBACK TO SAVEPOINT**.

Si crea un segundo punto de grabación con el mismo nombre que uno anterior, el primero se suprime.

Procesamiento Implícito de Transacciones

- Se produce una validación automática en las siguientes circunstancias:
 - Se emite una sentencia DDL.
 - Se emite una sentencia DCL.
 - Se sale normalmente de la herramienta de trabajo, sin emitir explícitamente sentencias **COMMIT** o **ROLLBACK**.
- Se realiza un rollback automático tras una terminación anormal de la herramienta de trabajo o un fallo del sistema.

Todo cambio de datos realizado durante la transacción es temporal hasta que se valida la transacción.

Estado de los datos antes de que se emitan sentencias **COMMIT** o **ROLLBACK**:

- Las operaciones de manipulación de datos afectan principalmente al buffer de la base de datos; por lo tanto, el estado anterior de los datos se puede recuperar.
- El usuario actual puede revisar los resultados de las operaciones de manipulación de datos consultando las tablas.
- Otros usuarios no pueden ver los resultados de las operaciones de manipulación de datos realizados por el usuario actual. Oracle Server

establece la consistencia de lectura para asegurar que cada usuario ve los datos como eran en la última validación.

- Las filas afectadas se bloquean; otros usuarios no pueden cambiar los datos de las filas afectadas.

Haga que todos los cambios pendientes sean permanentes utilizando la sentencia COMMIT. Tras una sentencia COMMIT:

- Los cambios de datos se escriben en la base de datos.
- El estado anterior de los datos se pierde permanentemente.
- Todos los usuarios pueden ver los resultados de la transacción.
- Los bloqueos de las filas afectadas se liberan; las filas están ahora disponibles para que otros usuarios realicen nuevos cambios de datos.
- Todos los puntos de grabación se borran.

Ejemplo

Elimine los departamentos 29 y 30 de la tabla DEPARTMENTS y actualice una fila en la tabla COPY_EMP. Haga que el cambio de datos sea permanente.

```
DELETE FROM dept
WHERE deptno IN (29, 30);
```

```
UPDATE copy_emp
SET deptno = 80
WHERE empno = 206;
COMMIT;
```

Estado de los Datos Después de ROLLBACK

Deseche todos los cambios pendientes utilizando la sentencia ROLLBACK:

- Los cambios de datos se deshacen.
- Se restaura el estado anterior de los datos.
- Se liberan los bloqueos de las filas afectadas.

Rollbacks a Nivel de Sentencia

Un rollback implícito puede desechar parte de una transacción si se detecta un error de ejecución de sentencia. Si falla una sola sentencia DML durante la ejecución de una transacción, su efecto lo deshace un rollback a nivel de sentencia, pero los cambios realizados por las sentencias DML anteriores en la transacción no se desechan. El usuario puede validarlas o realizar rollback de ellas explícitamente.

Oracle emite una validación implícita antes y después de cualquier sentencia del lenguaje de definición de datos (DDL). Por ello, aunque la sentencia DDL no se ejecute correctamente, no puede realizar rollback de la sentencia anterior porque el servidor emitió una validación.

Termine las transacciones explícitamente ejecutando una sentencia COMMIT o ROLLBACK.

Consistencia de Lectura

Los usuarios de la base de datos acceden a ésta de dos formas:

- Operaciones de lectura (sentencia SELECT)
- Operaciones de escritura (sentencias INSERT, UPDATE, DELETE)

Necesita consistencia de lectura para que:

- El escritor y el lector de la base de datos se aseguren una visualización consistente de los datos.
- Los lectores no visualicen los datos que están en proceso de cambio.
- Los escritores se aseguren de que los cambios en la base de datos se realizan de forma consistente.
- Los cambios realizados por un escritor no interrumpan a los que está haciendo otro escritor ni entren en conflicto con ellos.

El objetivo de la consistencia de lectura es asegurar que cada usuario ve los datos tal como eran en la última validación, antes de que comenzara una operación DML.

IMPLEMENTACIÓN DE CONSISTENCIA DE LECTURA

La consistencia de lectura es una implementación automática. Mantiene una copia parcial de la base de datos en segmentos deshacer.

Cuando se realiza una operación de inserción, actualización o supresión en la base de datos, Oracle Server realiza una copia de los datos antes de que se cambien y los escribe en un segmento deshacer.

Todos los lectores, excepto el que emitió el cambio, seguirán viendo la base de datos como existía antes de que comenzaran los cambios; ven la “instantánea” del segmento de rollback de los datos.

Antes de que se validen los cambios en la base de datos, sólo el usuario que está modificando los datos ve la base de datos con las modificaciones; los demás ven la instantánea en el segmento deshacer. Esto garantiza que los lectores de los datos leen datos consistentes que no están sufriendo ningún cambio actualmente.

Cuando se valida una sentencia DML, el cambio realizado en la base de datos se hace visible para cualquier persona que ejecute una sentencia SELECT. El espacio ocupado por los datos antiguos en el archivo del segmento deshacer se libera para su nuevo uso.

Si se realiza rollback de la transacción, los cambios se deshacen:

- La versión original (más antigua) de los datos del segmento deshacer se vuelve a escribir en la tabla.
- Todos los usuarios ven la base de datos como existía antes de que comenzara la transacción.

¿Qué Son los Bloqueos?

Los bloqueos son mecanismos que evitan una interacción destructiva entre las transacciones que acceden al mismo recurso, ya sea un objeto de usuario (como tablas o filas) o un objeto de sistema no visible a los usuarios (como estructuras de datos compartidos y filas del diccionario de datos).

Cómo Bloquea Datos la Base de Datos Oracle

El bloqueo de Oracle se realiza automáticamente y no requiere acción del usuario. El bloqueo implícito se produce para las sentencias SQL según sea necesario, dependiendo de la acción solicitada. Este bloqueo se produce para todas las sentencias SQL excepto SELECT.

Los usuarios también pueden bloquear los datos manualmente, lo que se denomina bloqueo explícito.

Bloqueo DML

Al realizar operaciones del lenguaje de manipulación de datos (DML), Oracle Server proporciona simultaneidad de datos a través del bloqueo DML, que se produce a dos niveles:

- Un bloqueo compartido se obtiene automáticamente a nivel de tabla durante operaciones DML: Con el modo de bloqueo compartido, varias transacciones pueden adquirir bloqueos compartidos sobre el mismo recurso.
- Un bloqueo exclusivo se adquiere automáticamente para cada fila modificada por una sentencia DML. Los bloqueos exclusivos evitan que otras transacciones cambien la fila hasta que la transacción se valida o se realiza rollback. Este bloqueo asegura que ningún otro usuario puede modificar la misma fila al mismo tiempo ni sobrescribir los cambios que aún no han sido validados por otro usuario.

Los bloqueos DDL se producen al modificar un objeto de base de datos como una tabla.

Ejercicio práctico

1. Crear la tabla MY_EMPLOYEE en base a la tabla EMPLOYEES que se utilizará para la práctica.
2. Describa la estructura de la tabla MY_EMPLOYEE para identificar los nombres de columna.
3. Agregue la primera fila de datos a la tabla MY_EMPLOYEE. No enumere las columnas en la cláusula INSERT.
4. Rellene la tabla MY_EMPLOYEE con los datos de la tabla EMPLOYEES que coincidan con los empleados que trabajan el departamento 90. Esta vez, enumere las columnas explícitamente en la cláusula INSERT.
5. Confirme la adición a la tabla.
6. Escriba una sentencia INSERT en un archivo de texto llamado loademp.sql para cargar filas en la tabla MY_EMPLOYEE. Concatene la primera letra del nombre con los primeros siete caracteres del apellido para crear el correo del empleado.
7. Confirme las adiciones a la tabla.
8. Haga que las adiciones de datos sean permanentes.
9. Actualice y suprima datos en la tabla MY_EMPLOYEE.
10. Cambie el apellido del empleado 3 por Drexler.
11. Cambie el salario a 1000 para todos los empleados con un salario inferior a 900.
12. Verifique los cambios realizados a la tabla.
13. Confirme los cambios realizados a la tabla.
14. Vacíe toda la tabla.
15. Confirme que la tabla está vacía.
16. Deseche la operación DELETE más reciente sin desechar la operación INSERT anterior.

Í N D I C E

Capítulo 1.....	5
El lenguaje SQL.....	6
1. Esquema usado para el curso.	6
2. Herramienta de trabajo: el SQL Developer.....	10
3. Sentencias básicas del SQL: SELECT.....	12
Sentencias DML.....	13
Escritura de Sentencias SQL	16
Expresiones Aritméticas.....	16
Los Operadores Aritméticos.....	16
Los Valores Nulos.....	18
4. Filtro de Filas en una Selección	24
Condiciones de comparación	25
Otros operadores de comparación.....	26
Combinación de Caracteres Comodín.....	29
Condiciones Lógicas	30
Reglas de Prioridad	32
Las funciones SQL de Oracle 11g	35
Funciones de una Sola Fila.....	37
Funciones de Manipulación de Mayúsculas/Minúsculas	38
Funciones de Conversión	47
CAPÍTULO 2.....	57
Consultas con SQL.....	58
1. Visualización de datos de varias tablas.....	58
Datos de Varias Tablas	58
Productos Cartesianos	59
Tipos de Uniones	60
Definición de Uniones.....	60
Alias de Tabla.....	62
Uniones de No Igualdad.....	64
Devolución de Registros sin Coincidencia Directa con Uniones Externas.....	64
Uso de Uniones Externas para Devolver Registros sin Coincidencia Directa	64
Definición de Uniones con la sintaxis SQL 1999.....	65
Funciones De grupos.....	69
CAPÍTULO 3.....	77
Subconsultas.....	78
Definición	78
Uso de una Subconsulta para Resolver un Problema.....	78

Tipos de Subconsultas.....	79
Los operadores de conjuntos	84
CAPÍTULO 4.....	91
Manipulación de datos en las tablas.....	92
Lenguaje de Manipulación de Datos	92
Modificación de datos en una tabla	96
Actualización de Dos Columnas con una Subconsulta	98
Actualización de Filas basándose en otra Tabla	98
Eliminación de filas de una tabla.....	99
Uso de una Subconsulta en una Sentencia INSERT.....	100
Sentencias MERGE.....	101
Transacciones de Base de Datos.....	103
Realización de Rollback de los Cambios a un Punto de Grabación	104
Implementación de Consistencia de Lectura.....	106