**UNIVERSAD MARIANO GALVEZ DE GUATEMALA**

**CENTRO UNIVERSITARIO DE EL PROGRESO, GUASTATOYA**

**FACULTAD DE INGENIERIA EN SISTEMAS Y CIENCIAS DE LA COMPUTACION**

**Ingeniero:** Carlos Amílcar Tezo Palencia

**Curso:** Base de Datos 1



**UMG**

**Estudiante:**

Melvin Geovanny Tebalan Donis

**Carne:**

1890 – 15 – 1673

**Ciclo:** Sexto Semestre

**“Practica SQL”**

**Guatemala, 10 de septiembre de 2,022**

# 5 - Recuperar algunos campos (select)

Hemos aprendido cómo ver todos los registros de una tabla, empleando la instrucción "select".

La sintaxis básica y general es la siguiente:

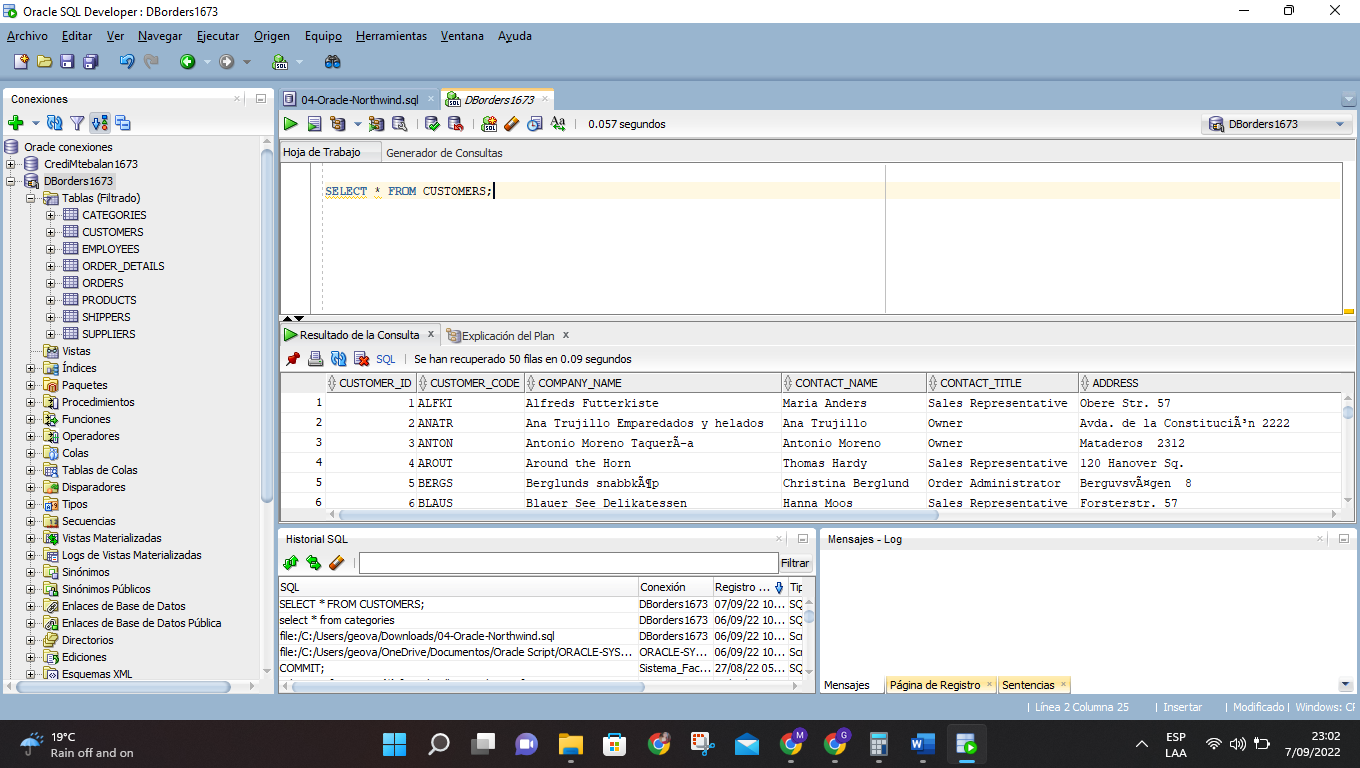
select \*from NOMBRETABLA;

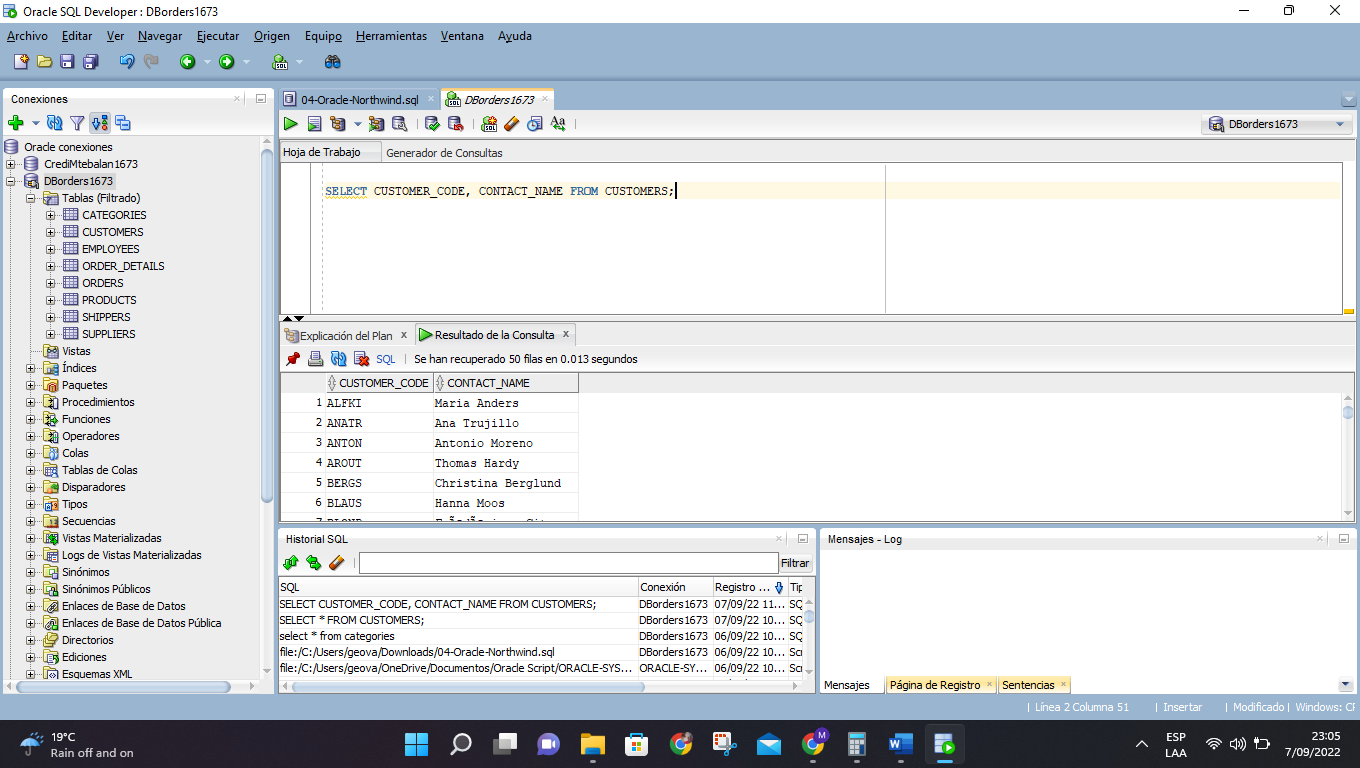
El asterisco (\*) indica que se seleccionan todos los campos de la tabla.

Podemos especificar el nombre de los campos que queremos ver, separándolos por comas:

select titulo,autor from libros;

La lista de campos luego del "select" selecciona los datos correspondientes a los campos nombrados. En el ejemplo anterior seleccionamos los campos "titulo" y "autor" de la tabla "libros", mostrando todos los registros.





# 6 - Recuperar algunos registros (where)

Hemos aprendido a seleccionar algunos campos de una tabla.

También es posible recuperar algunos registros.

Existe una cláusula, "where" con la cual podemos especificar condiciones para una consulta "select". Es decir, podemos recuperar algunos registros, sólo los que cumplan con ciertas condiciones indicadas con la cláusula "where". Por ejemplo, queremos ver el usuario cuyo nombre es "Marcelo", para ello utilizamos "where" y luego de ella, la condición:

select nombre, clave

from usuarios

where nombre='Marcelo';

La sintaxis básica y general es la siguiente:

select NOMBRECAMPO1, ..., NOMBRECAMPOn

from NOMBRETABLA

where CONDICION;

Para las condiciones se utilizan operadores relacionales (tema que trataremos más adelante en detalle). El signo igual(=) es un operador relacional. Para la siguiente selección de registros especificamos una condición que solicita los usuarios cuya clave es igual a "River":

select nombre,clave

from usuarios

where clave='River';

Si ningún registro cumple la condición establecida con el "where", no aparecerá ningún registro.

Entonces, con "where" establecemos condiciones para recuperar algunos registros.

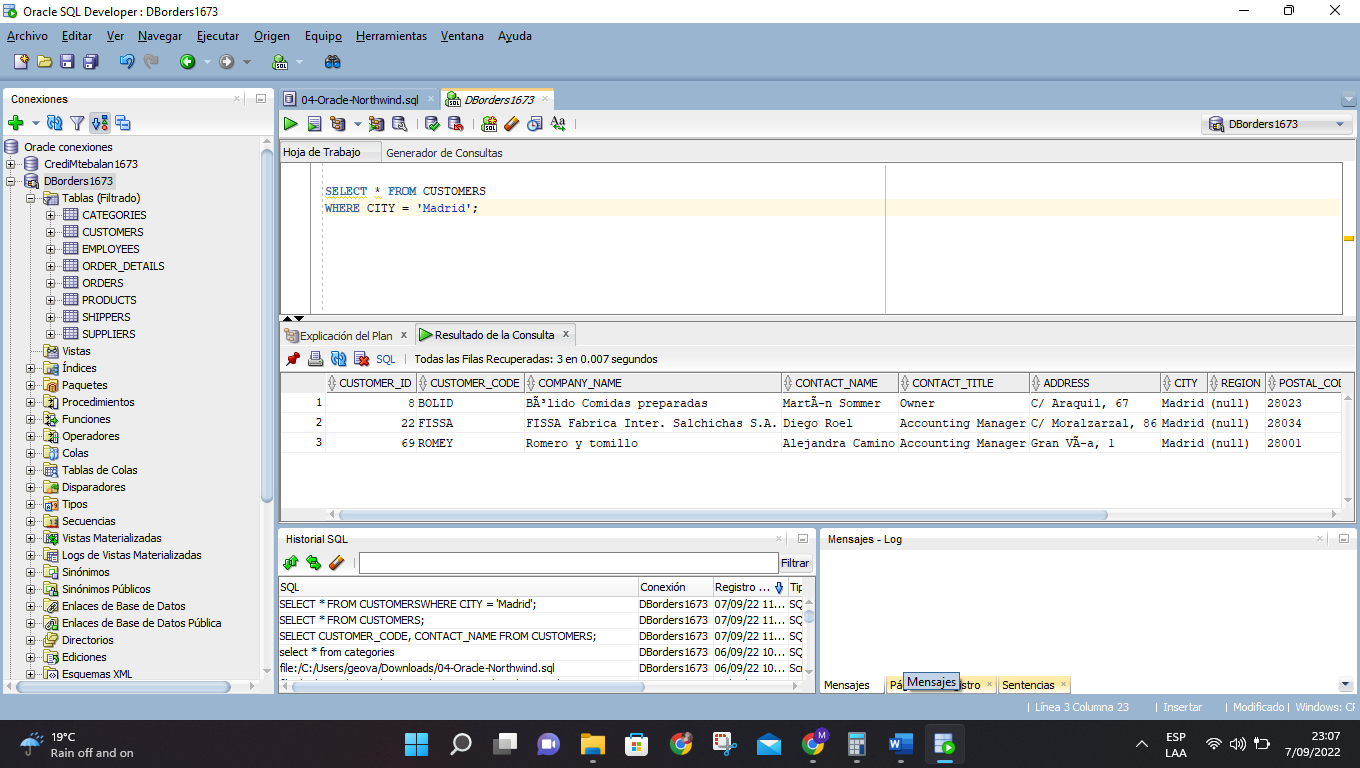
Para recuperar algunos campos de algunos registros combinamos en la consulta la lista de campos y la cláusula "where":

select nombre

from usuarios

where clave='River';

En la consulta anterior solicitamos el nombre de todos los usuarios cuya clave sea igual a "River".



# 7 - Operadores relacionales

Los operadores son símbolos que permiten realizar operaciones matemáticas, concatenar cadenas, hacer comparaciones.

Oracle reconoce de 4 tipos de operadores:

1) relacionales (o de comparación)

2) aritméticos

3) de concatenación

4) lógicos

Por ahora veremos solamente los primeros.

Los operadores relacionales (o de comparación) nos permiten comparar dos expresiones, que pueden ser variables, valores de campos, etc.

Hemos aprendido a especificar condiciones de igualdad para seleccionar registros de una tabla; por ejemplo:

select \*from libros

where autor='Borges';

Utilizamos el operador relacional de igualdad.

Los operadores relacionales vinculan un campo con un valor para que Oracle compare cada registro (el campo especificado) con el valor dado.

Los operadores relacionales son los siguientes:

= igual

<> distinto

> mayor

< menor

>= mayor o igual

<= menor o igual

Podemos seleccionar los registros cuyo autor sea diferente de "Borges", para ello usamos la condición:

select \* from libros

where autor<>'Borges';

Podemos comparar valores numéricos. Por ejemplo, queremos mostrar los títulos y precios de los libros cuyo precio sea mayor a 20 pesos:

select titulo, precio

from libros

where precio>20;

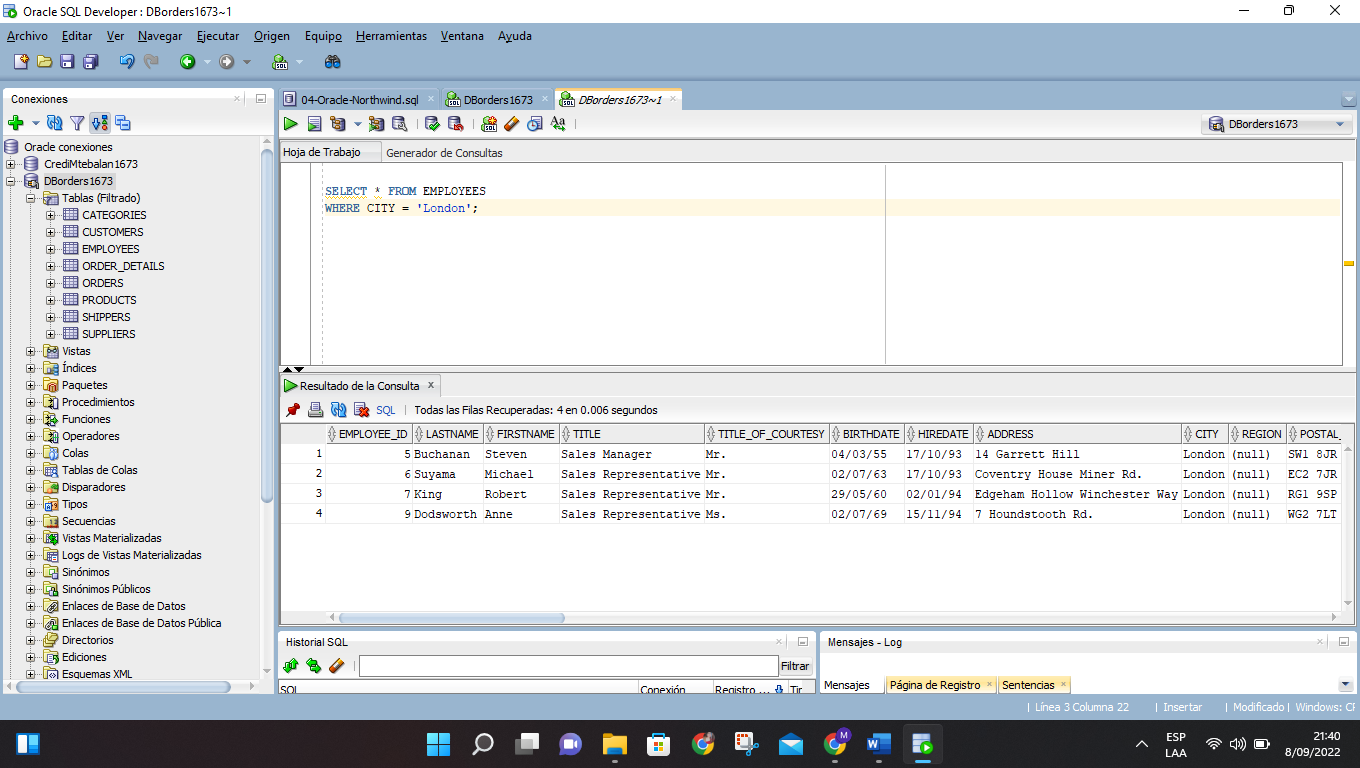
Queremos seleccionar los libros cuyo precio sea menor o igual a 30:

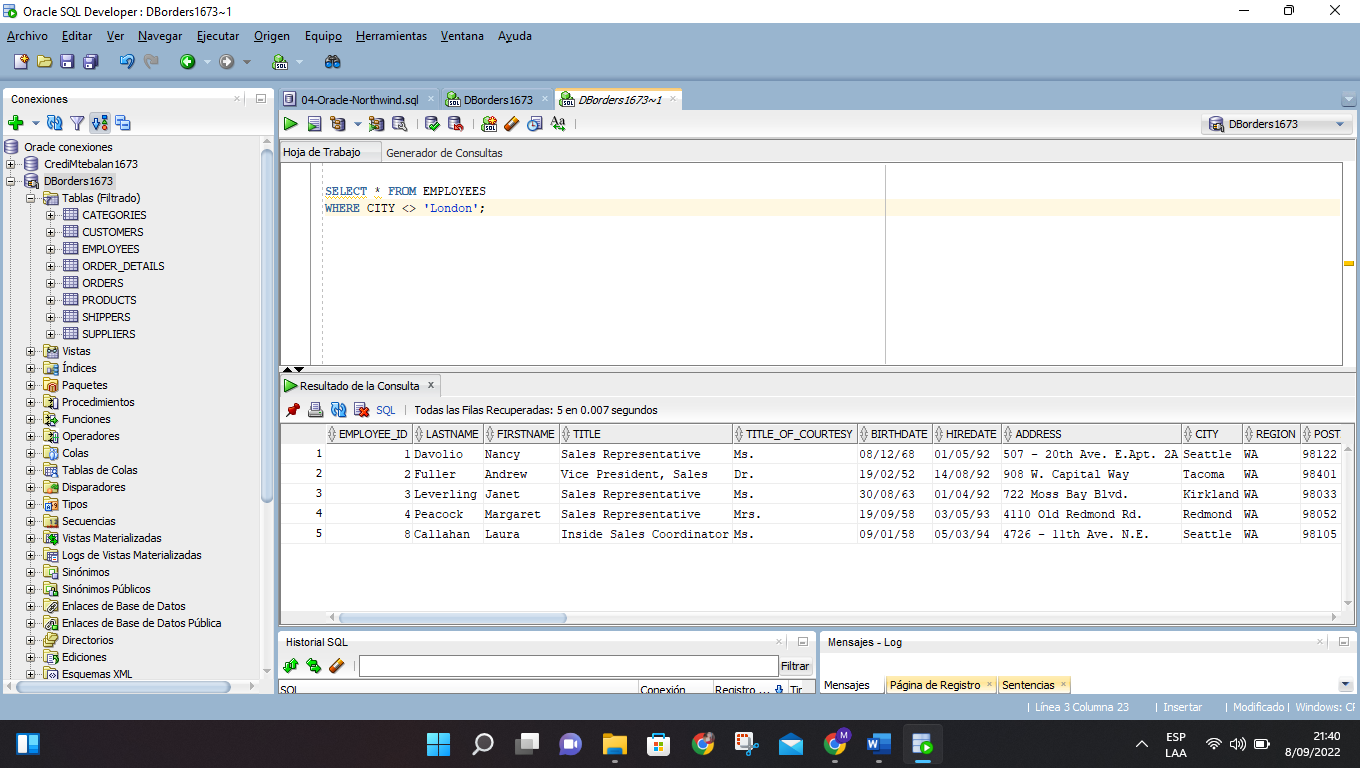
select \*from libros

where precio<=30;

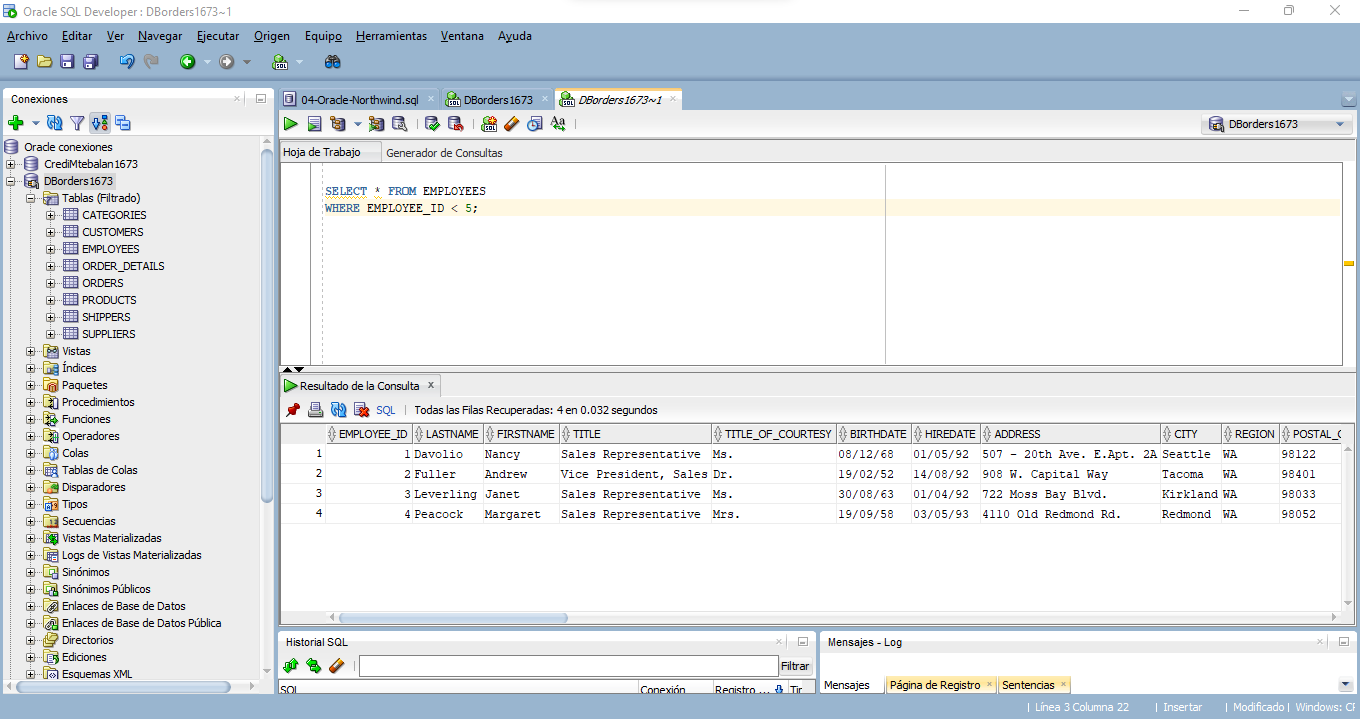
Los operadores relacionales comparan valores del mismo tipo. Se emplean para comprobar si un campo cumple con una condición.

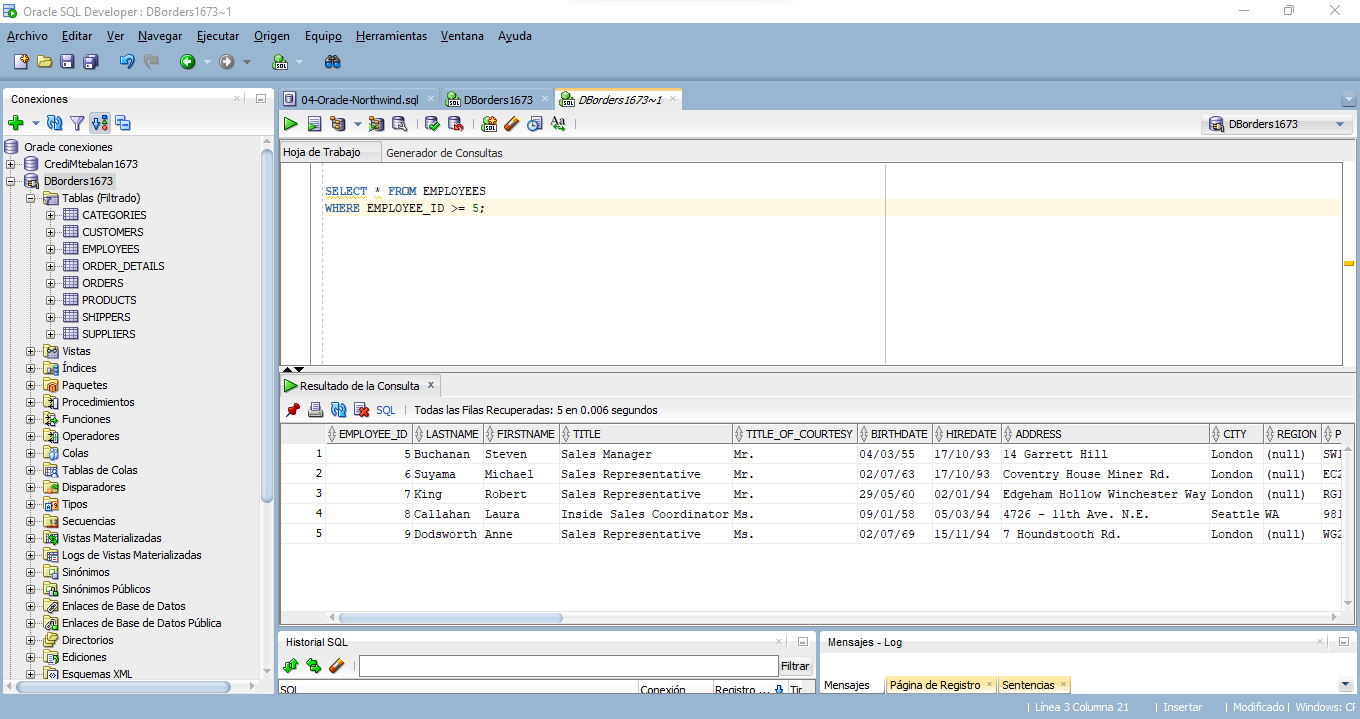
No son los únicos, existen otros que veremos mas adelante.

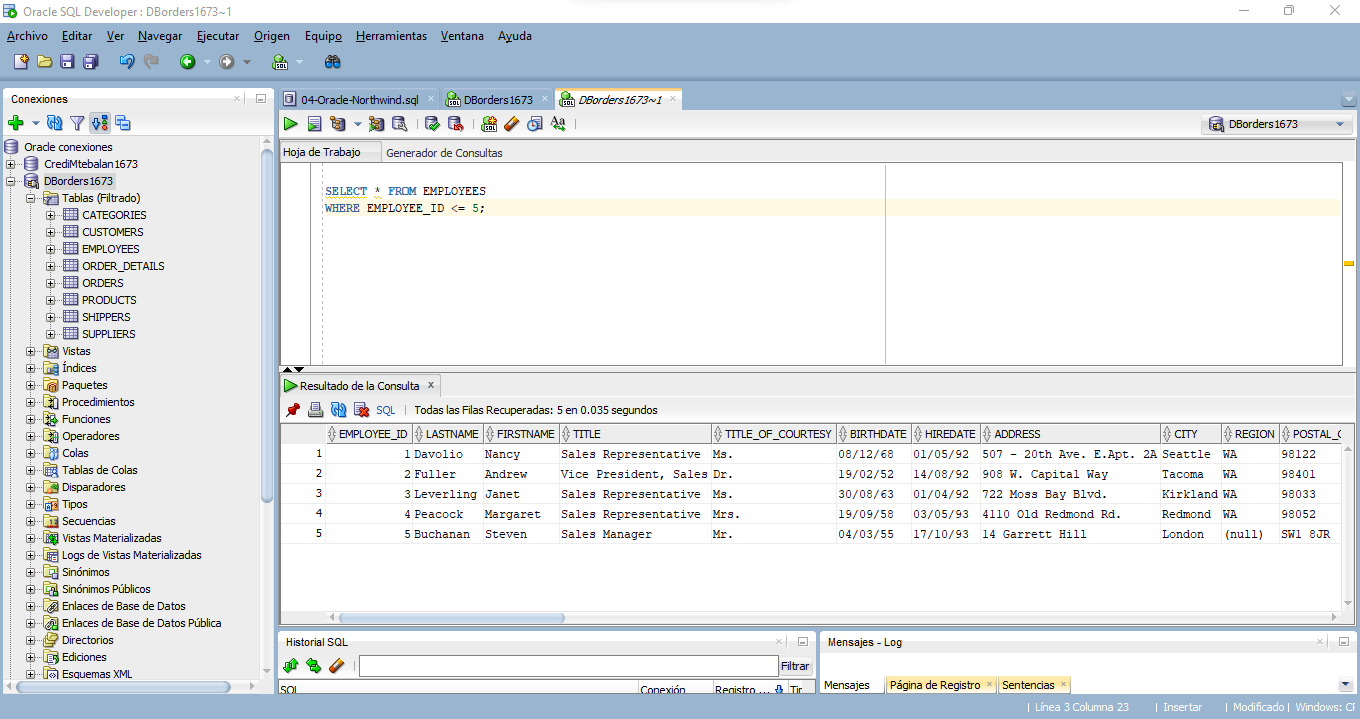












# 8 - Borrar registros (delete)

Para eliminar los registros de una tabla usamos el comando "delete".

Sintaxis básica:

delete from NOMBRETABLA;

Se coloca el comando delete seguido de la palabra clave "from" y el nombre de la tabla de la cual queremos eliminar los registros. En el siguiente ejemplo se eliminan los registros de la tabla "usuarios":

delete from usuarios;

Luego, un mensaje indica la cantidad de registros que se han eliminado.

Si no queremos eliminar todos los registros, sino solamente algunos, debemos indicar cuál o cuáles; para ello utilizamos el comando "delete" junto con la clausula "where" con la cual establecemos la condición que deben cumplir los registros a borrar.

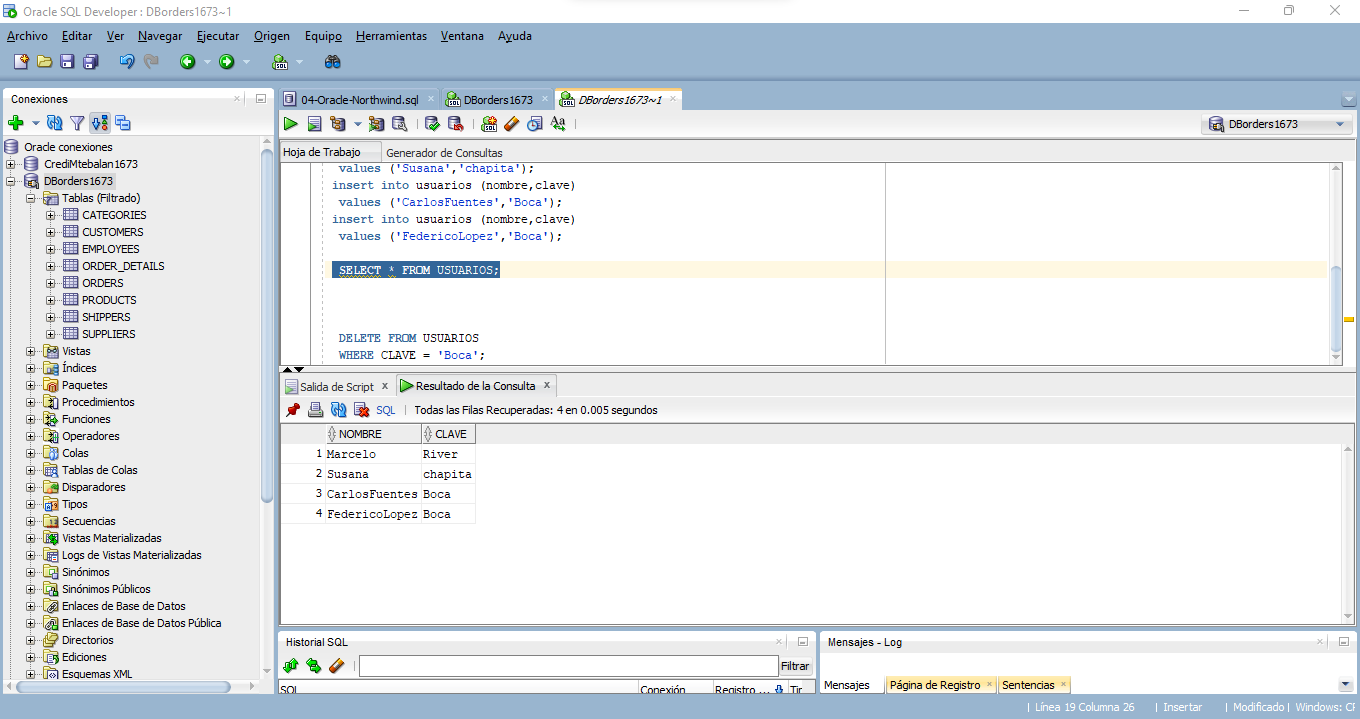
Por ejemplo, queremos eliminar aquel registro cuyo nombre de usuario es "Marcelo":

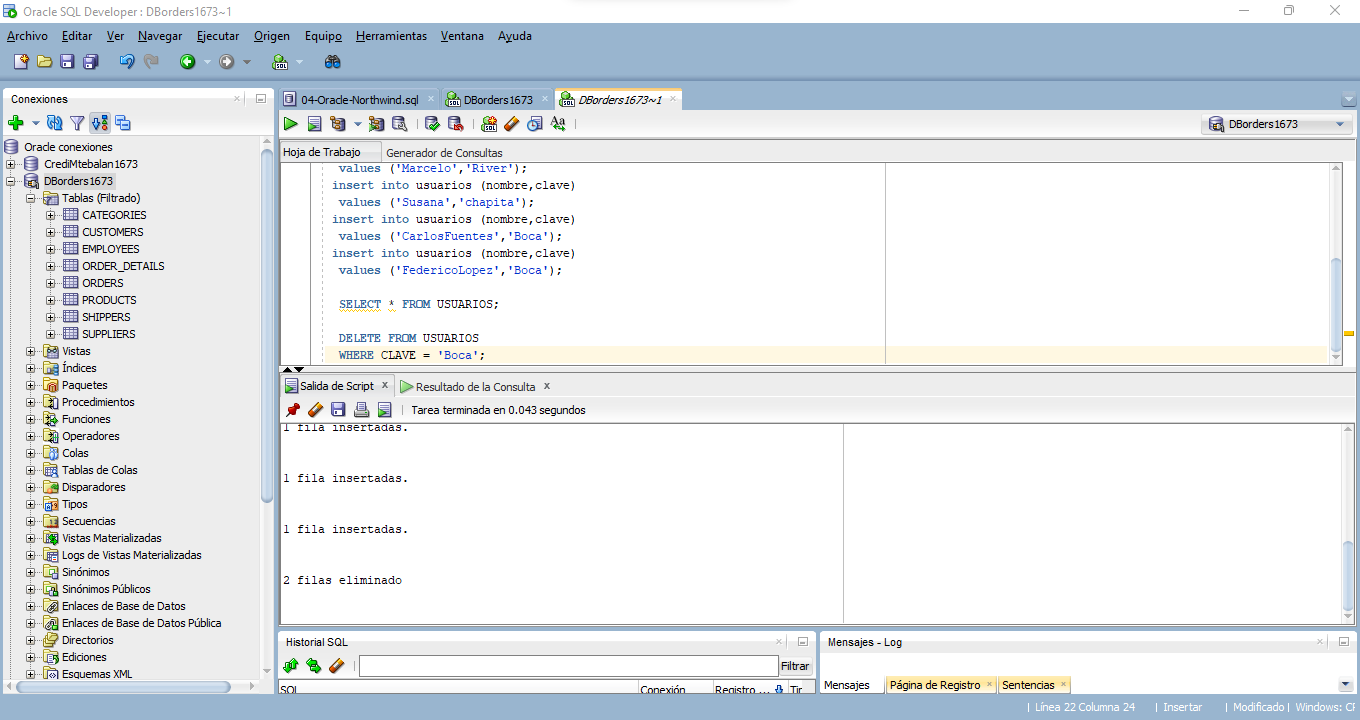
delete from usuarios

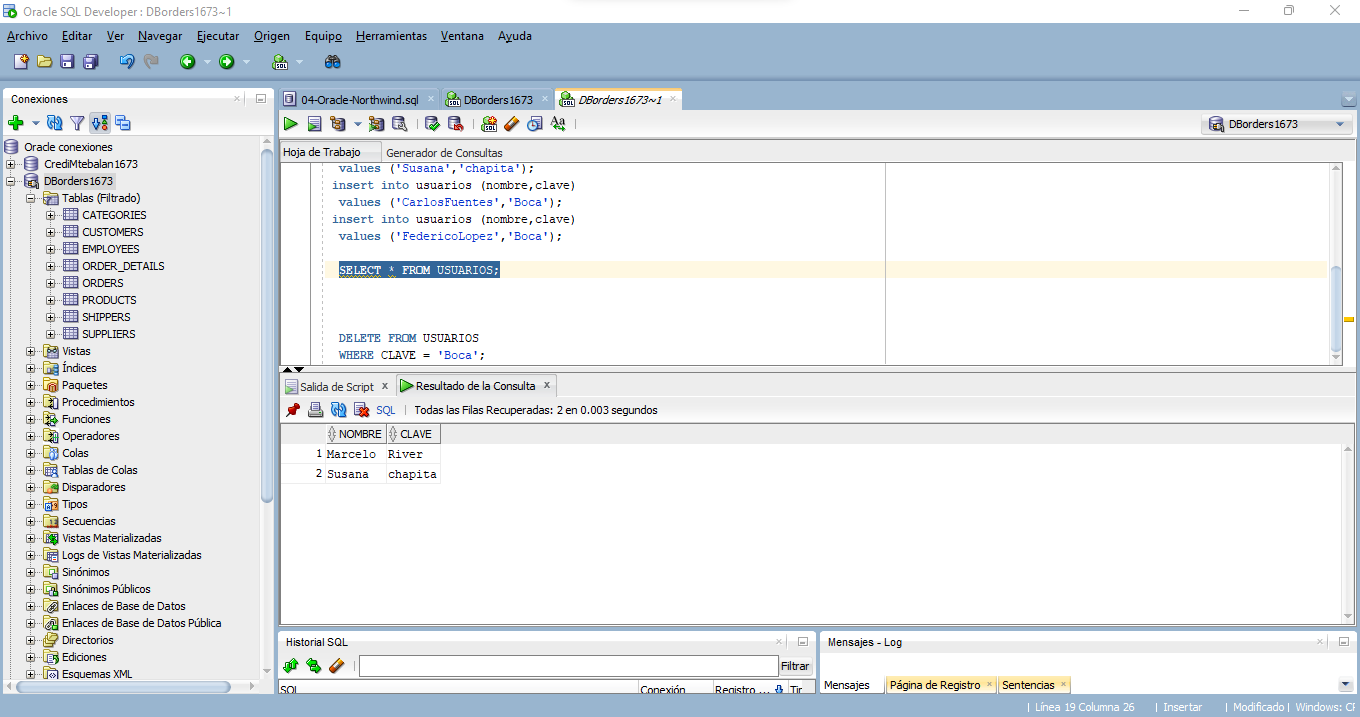
where nombre='Marcelo';

Si solicitamos el borrado de un registro que no existe, es decir, ningún registro cumple con la condición especificada, aparecerá un mensaje indicando que ningún registro fue eliminado, pues no encontró registros con ese dato.

Tenga en cuenta que si no colocamos una condición, se eliminan todos los registros de la tabla especificada.







# 9 - Actualizar registros (update)

Decimos que actualizamos un registro cuando modificamos alguno de sus valores.

Para modificar uno o varios datos de uno o varios registros utilizamos "update" (actualizar).

Sintaxis básica:

update NOMBRETABLA set CAMPO=NUEVOVALOR;

Utilizamos "update" junto al nombre de la tabla y "set" junto con el campo a modificar y su nuevo valor.

El cambio afectará a todos los registros.

Por ejemplo, en nuestra tabla "usuarios", queremos cambiar los valores de todas las claves, por "RealMadrid":

update usuarios set clave='RealMadrid';

Podemos modificar algunos registros, para ello debemos establecer condiciones de selección con "where".

Por ejemplo, queremos cambiar el valor correspondiente a la clave de nuestro usuario llamado "Federicolopez", queremos como nueva clave "Boca", necesitamos una condición "where" que afecte solamente a este registro:

update usuarios set clave='Boca'

where nombre='Federicolopez';

Si Oracle no encuentra registros que cumplan con la condición del "where", un mensaje indica que ningún registro fue modificado.

Las condiciones no son obligatorias, pero si omitimos la cláusula "where", la actualización afectará a todos los registros.

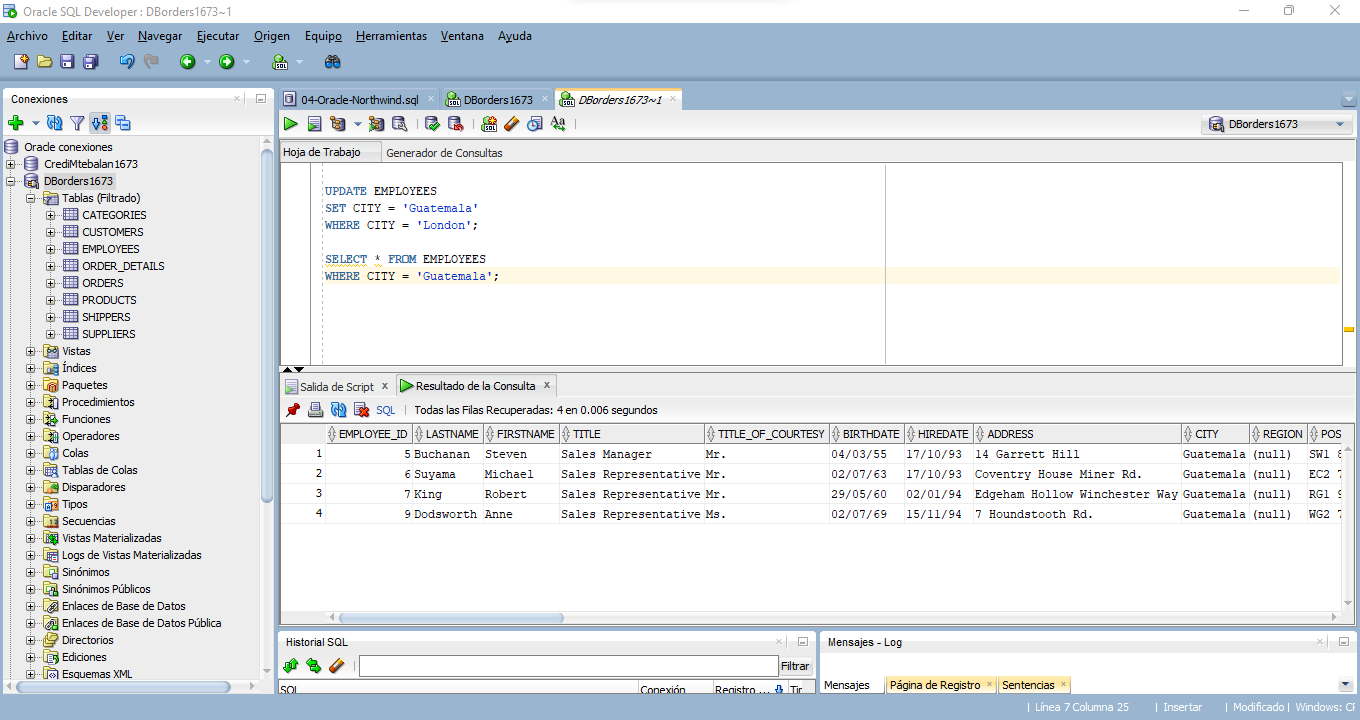
También podemos actualizar varios campos en una sola instrucción:

update usuarios set nombre='Marceloduarte', clave='Marce'

where nombre='Marcelo';

Para ello colocamos "update", el nombre de la tabla, "set" junto al nombre del campo y el nuevo valor y separado por coma, el otro nombre del campo con su nuevo valor.





# 10 – Comentarios

Para aclarar algunas instrucciones, en ocasiones, necesitamos agregar comentarios.

Es posible ingresar comentarios en la línea de comandos, es decir, un texto que no se ejecuta; para ello se emplean dos guiones (--):

select \*from libros;--mostramos los registros de libros

en la línea anterior, todo lo que está luego de los guiones (hacia la derecha) no se ejecuta.

Para agregar varias líneas de comentarios, se coloca una barra seguida de un asterisco (/\*) al comienzo del bloque de comentario y al finalizarlo, un asterisco seguido de una barra (\*/)

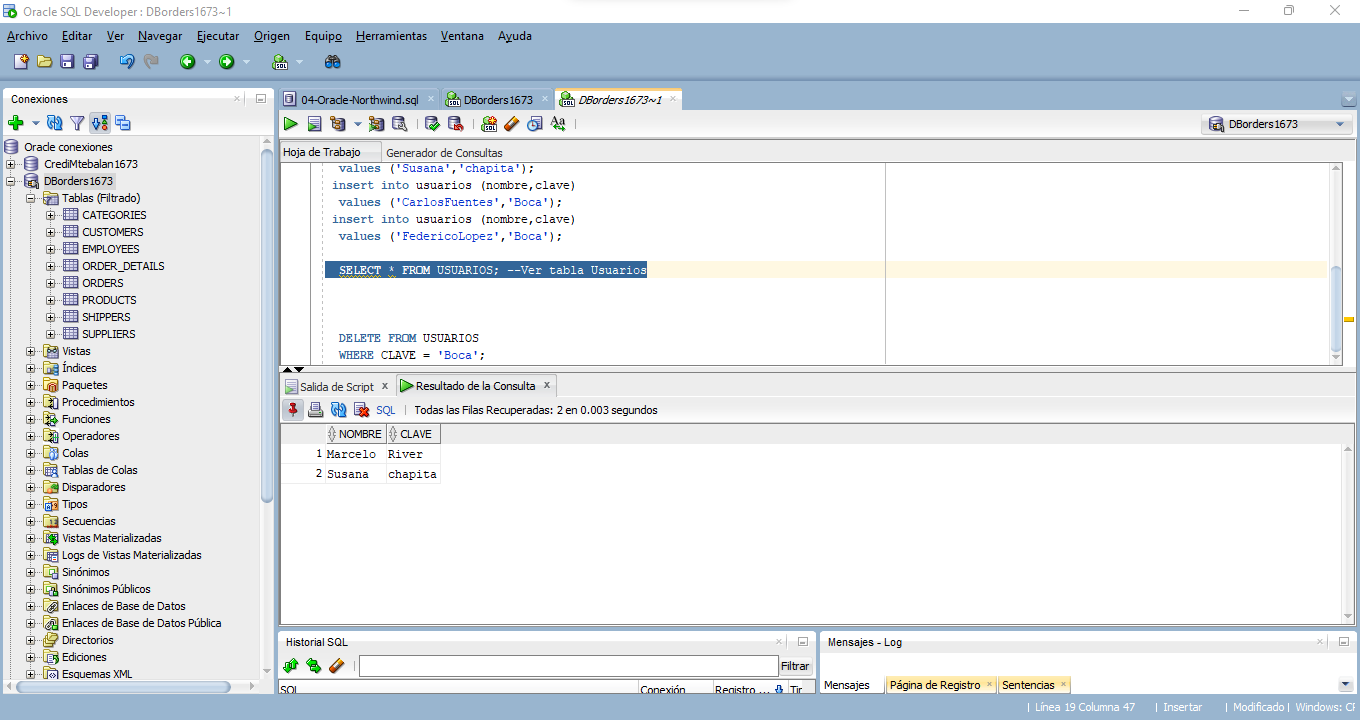
select titulo, autor

/\*mostramos títulos y

nombres de los autores\*/

from libros;

todo lo que está entre los símbolos "/\*" y "\*/" no se ejecuta.



# 11 - Valores nulos (null)

"null' significa "dato desconocido" o "valor inexistente".

A veces, puede desconocerse o no existir el dato correspondiente a algún campo de un registro. En estos casos decimos que el campo puede contener valores nulos.

Por ejemplo, en nuestra tabla de libros, podemos tener valores nulos en el campo "precio" porque es posible que para algunos libros no le hayamos establecido el precio para la venta.

En contraposición, tenemos campos que no pueden estar vacíos jamás.

Veamos un ejemplo. Tenemos nuestra tabla "libros". El campo "titulo" no debería estar vacío nunca, igualmente el campo "autor". Para ello, al crear la tabla, debemos especificar que tales campos no admitan valores nulos:

create table libros(

titulo varchar2(30) not null,

autor varchar2(20) not null,

editorial varchar2(15) null,

precio number(5,2)

);

Para especificar que un campo NO admita valores nulos, debemos colocar "not null" luego de la definición del campo.

En el ejemplo anterior, los campos "editorial" y "precio" si admiten valores nulos.

Cuando colocamos "null" estamos diciendo que admite valores nulos (caso del campo "editorial"); por defecto, es decir, si no lo aclaramos, los campos permiten valores nulos (caso del campo "precio").

Cualquier campo, de cualquier tipo de dato permite ser definido para aceptar o no valores nulos. Un valor "null" NO es lo mismo que un valor 0 (cero) o una cadena de espacios en blanco (" ").

Si ingresamos los datos de un libro, para el cual aún no hemos definido el precio podemos colocar "null" para mostrar que no tiene precio:

insert into libros (titulo,autor,editorial,precio)

values('El aleph','Borges','Emece',null);

Note que el valor "null" no es una cadena de caracteres, NO se coloca entre comillas.

Entonces, si un campo acepta valores nulos, podemos ingresar "null" cuando no conocemos el valor.

También podemos colocar "null" en el campo "editorial" si desconocemos el nombre de la editorial a la cual pertenece el libro que vamos a ingresar:

insert into libros (titulo,autor,editorial,precio)

values('Alicia en el pais','Lewis Carroll',null,25);

Una cadena vacía es interpretada por Oracle como valor nulo; por lo tanto, si ingresamos una cadena vacía, se almacena el valor "null".

Si intentamos ingresar el valor "null" (o una cadena vacía) en campos que no admiten valores nulos (como "titulo" o "autor"), Oracle no lo permite, muestra un mensaje y la inserción no se realiza; por ejemplo:

insert into libros (titulo,autor,editorial,precio)

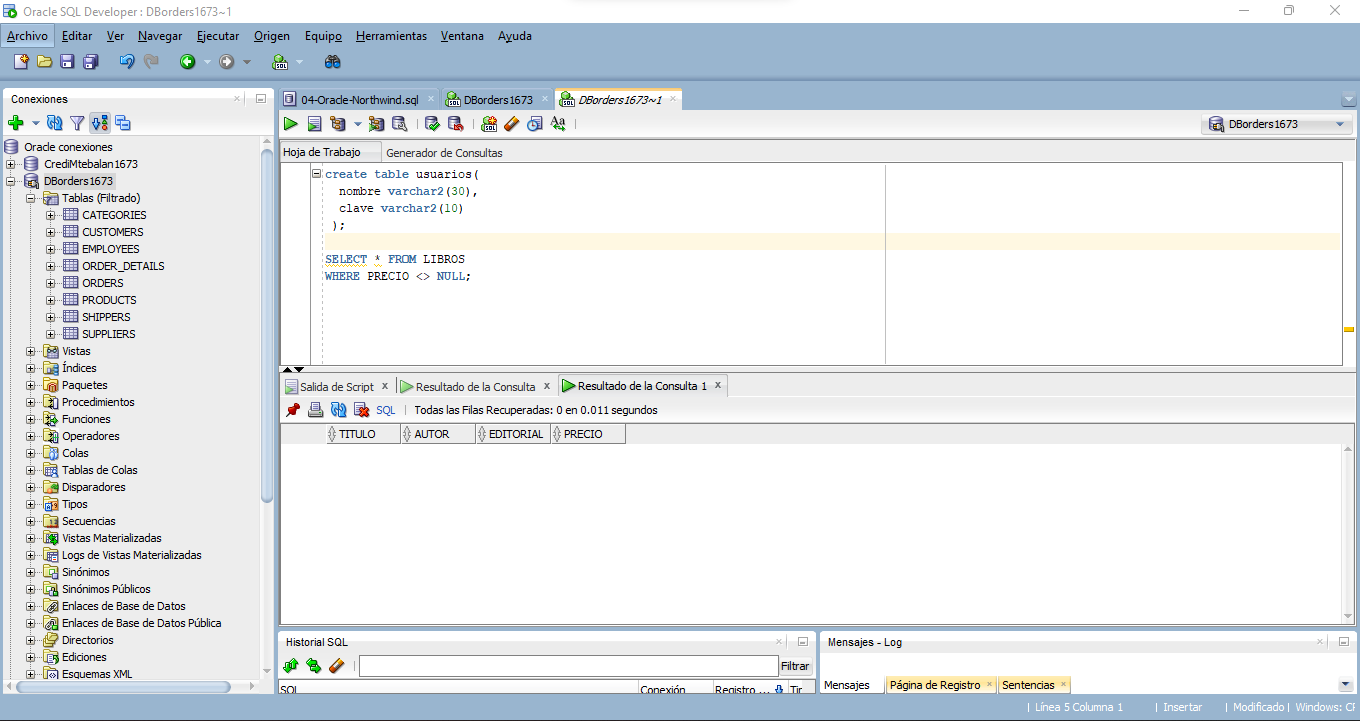
values(null,'Borges','Siglo XXI',25);

Cuando vemos la estructura de una tabla con "describe", en la columna "Null", aparece "NOT NULL" si el campo no admite valores nulos y no aparece en caso que si los permita.

Para recuperar los registros que contengan el valor "null" en algún campo, no podemos utilizar los operadores relacionales vistos anteriormente: = (igual) y <> (distinto); debemos utilizar los operadores "is null" (es igual a null) y "is not null" (no es null).

Los valores nulos no se muestran, aparece el campo vacío.

Entonces, para que un campo no permita valores nulos debemos especificarlo luego de definir el campo, agregando "not null". Por defecto, los campos permiten valores nulos, pero podemos especificarlo igualmente agregando "null".



# 12 - Operadores relacionales (is null)

Para recuperar los registros que contengan el valor "null" en algún campo, no podemos utilizar los operadores relacionales vistos anteriormente: = (igual) y <> (distinto); debemos utilizar los operadores "is null" (es igual a null) y "is not null" (no es null).

Con la siguiente sentencia recuperamos los libros que contienen valor nulo en el campo "editorial":

select \*from libros

where editorial is null;

Recuerde que los valores nulos no se muestran, aparece el campo vacío.

Las siguientes sentencias tendrán una salida diferente:

select \*from libros where editorial is null;

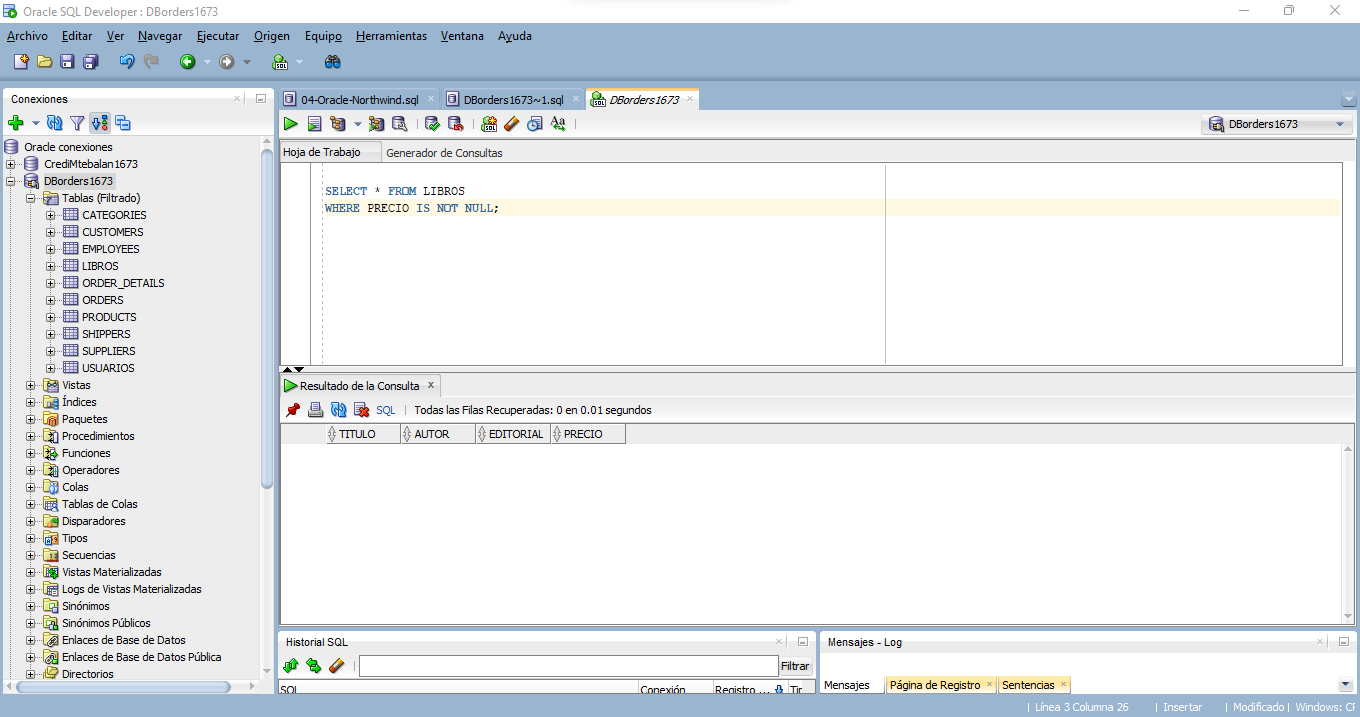
select \*from libros where editorial=' ';

Con la primera sentencia veremos los libros cuya editorial almacena el valor "null" (desconocido); con la segunda, los libros cuya editorial guarda una cadena de 3 espacios en blanco.

Para obtener los registros que no contienen "null", se puede emplear "is not null", esto mostrará los registros con valores conocidos.

Para ver los libros que NO tienen valor "null" en el campo "precio" tipeamos:

select \*from libros where precio is not null;



# 13 - Clave primaria (primary key)

Una clave primaria es un campo (o varios) que identifica un solo registro (fila) en una tabla.

Para un valor del campo clave existe solamente un registro.

Veamos un ejemplo, si tenemos una tabla con datos de personas, el número de documento puede establecerse como clave primaria, es un valor que no se repite; puede haber personas con igual apellido y nombre, incluso el mismo domicilio (padre e hijo por ejemplo), pero su documento será siempre distinto.

Si tenemos la tabla "usuarios", el nombre de cada usuario puede establecerse como clave primaria, es un valor que no se repite; puede haber usuarios con igual clave, pero su nombre de usuario será siempre diferente.

Podemos establecer que un campo sea clave primaria al momento de crear la tabla o luego que ha sido creada. Vamos a aprender a establecerla al crear la tabla. No existe una única manera de hacerlo, por ahora veremos la sintaxis más sencilla.

Tenemos nuestra tabla "usuarios" definida con 2 campos ("nombre" y "clave").

La sintaxis básica y general es la siguiente:

create table NOMBRETABLA(

CAMPO TIPO,

...,

CAMPO TIPO,

PRIMARY KEY (CAMPO)

);

Lo que hacemos agregar, luego de la definición de cada campo, "primary key" y entre paréntesis, el nombre del campo que será clave primaria.

En el siguiente ejemplo definimos una clave primaria, para nuestra tabla "usuarios" para asegurarnos que cada usuario tendrá un nombre diferente y único:

create table usuarios(

nombre varchar2(20),

clave varchar2(10),

primary key(nombre)

);

Una tabla sólo puede tener una clave primaria. Cualquier campo (de cualquier tipo) puede ser clave primaria, debe cumplir como requisito, que sus valores no se repitan ni sean nulos. Por ello, al definir un campo como clave primaria, automáticamente Oracle lo convierte a "not null".

Luego de haber establecido un campo como clave primaria, al ingresar los registros, Oracle controla que los valores para el campo establecido como clave primaria no estén repetidos en la tabla; si estuviesen repetidos, muestra un mensaje y la inserción no se realiza. Es decir, si en nuestra tabla "usuarios" ya existe un usuario con nombre "juanperez" e intentamos ingresar un nuevo usuario con nombre "juanperez", aparece un mensaje y la instrucción "insert" no se ejecuta.

Igualmente, si realizamos una actualización, Oracle controla que los valores para el campo establecido como clave primaria no estén repetidos en la tabla, si lo estuviese, aparece un mensaje indicando que se viola la clave primaria y la actualización no se realiza.

Podemos ver el campo establecido como clave primaria de una tabla realizando la siguiente consulta:

select uc.table\_name, column\_name from user\_cons\_columns ucc

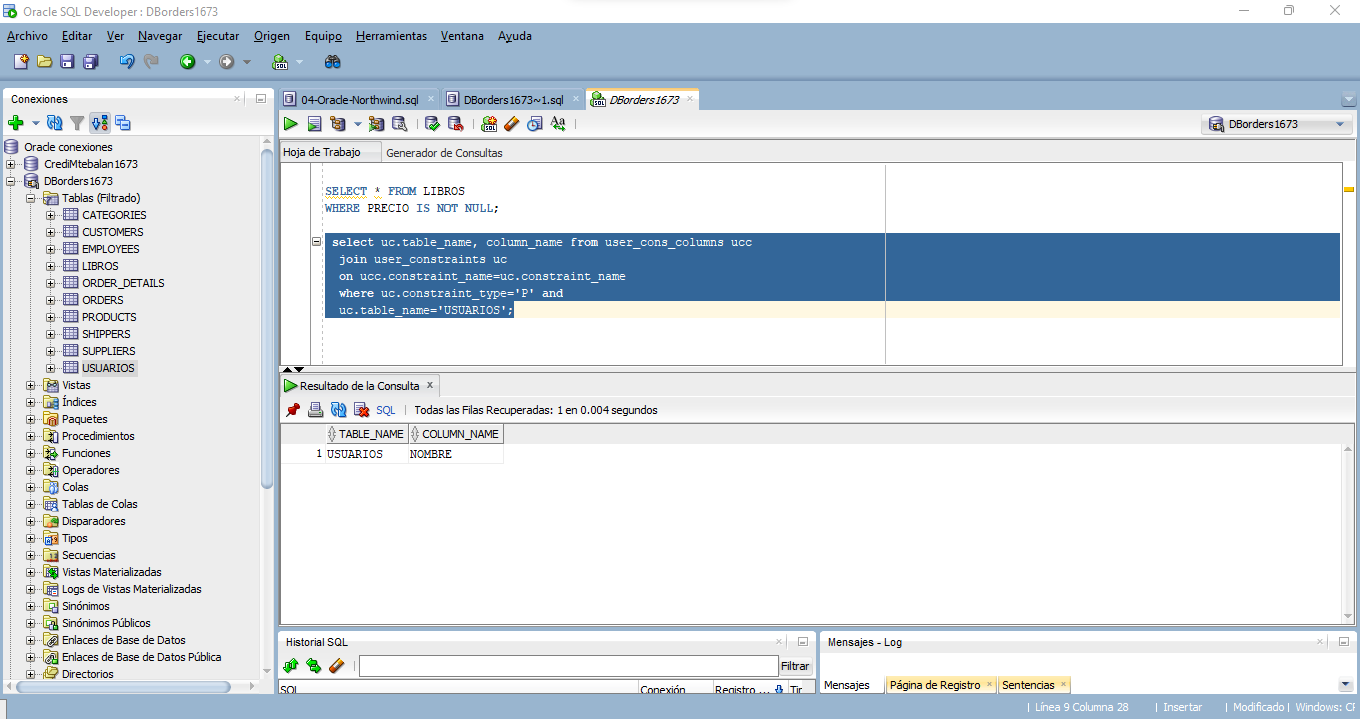
join user\_constraints uc

on ucc.constraint\_name=uc.constraint\_name

where uc.constraint\_type='P' and

uc.table\_name='USUARIOS';

No explicaremos la consulta anterior por el momento, sólo la ejecutaremos; si la consulta retorna una tabla vacía, significa que la tabla especificada no tiene clave primaria. El nombre de la tabla DEBE ir en mayúsculas, sino Oracle no la encontrará.



# 14 - Vaciar la tabla (truncate table)

Aprendimos que para borrar todos los registro de una tabla se usa "delete" sin condición "where".

También podemos eliminar todos los registros de una tabla con "truncate table". Sintaxis:

truncate table NOMBRETABLA;

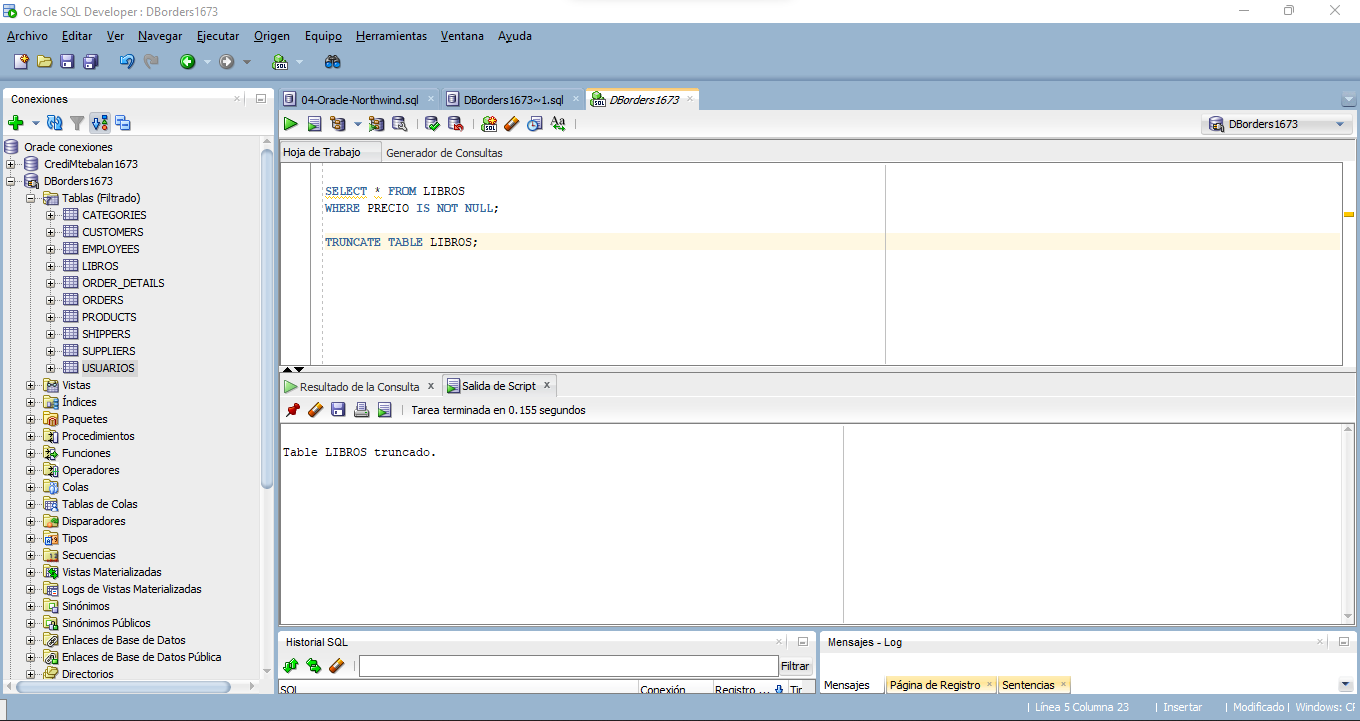
Por ejemplo, queremos vaciar la tabla "libros", usamos:

truncate table libros;

La sentencia "truncate table" vacía la tabla (elimina todos los registros) y conserva la estructura de la tabla.

La diferencia con "drop table" es que esta sentencia elimina la tabla, no solamente los registros, "truncate table" la vacía de registros.

La diferencia con "delete" es la siguiente, al emplear "delete", Oracle guarda una copia de los registros borrados y son recuperables, con "truncate table" no es posible la recuperación porque se libera todo el espacio en disco ocupado por la tabla; por lo tanto, "truncate table" es más rápido que "delete" (se nota cuando la cantidad de registros es muy grande).



# 15 - Tipos de datos alfanuméricos

Ya explicamos que al crear una tabla debemos elegir la estructura adecuada, esto es, definir los campos y sus tipos más precisos, según el caso.

Para almacenar valores alfanuméricos (texto) usamos cadenas de caracteres.

Las cadenas se colocan entre comillas simples.

Podemos almacenar letras, símbolos y dígitos con los que no se realizan operaciones matemáticas, por ejemplo, códigos de identificación, números de documentos, números telefónicos. Tenemos los siguientes tipos:

1) char(x): define una cadena de caracteres de longitud fija determinada por el argumento "x". Si se omite el argumento, por defecto coloca 1. "char" viene de character, que significa caracter en inglés. Su rango es de 1 a 2000 caracteres.

Que sea una cadena de longitud fija significa que, si definimos un campo como "char(10)" y almacenamos el valor "hola" (4 caracteres), Oracle rellenará las 6 posiciones restantes con espacios, es decir, ocupará las 10 posiciones; por lo tanto, si la longitud es invariable, es conveniente utilizar el tipo char; caso contrario, el tipo varchar2.

Si almacenamos "hola" en un campo definido "char(10)" Oracle almacenará "hola ".

2) varchar2(x): almacena cadenas de caracteres de longitud variable determinada por el argumento "x" (obligatorio). Que sea una cadena de longitud variable significa que, si definimos un campo como "varchar2(10)" y almacenamos el valor "hola" (4 caracteres), Oracle solamente ocupa las 4 posiciones (4 bytes y no 10 como en el caso de "char"); por lo tanto, si la longitud es variable, es conveniente utilizar este tipo de dato y no "char", así ocupamos menos espacio de almacenamiento en disco. Su rango es de 1 a 4000 caracteres.

3) nchar(x): es similar a "char" excepto que permite almacenar caracteres ASCII, EBCDIC y Unicode; su rango va de 1 a 1000 caracteres porque se emplean 2 bytes por cada caracter.

4) nvarchar2(x): es similar a "varchar2", excepto que permite almacenar caracteres Unicode; su rango va de 1 a 2000 caracteres porque se emplean 2 bytes por cada caracter.

5 y 6) varchar(x) y char2(x): disponibles en Oracle8.

7) long: guarda caracteres de longitud variable; puede contener hasta 2000000000 caracteres (2 Gb). No admite argumento para especificar su longitud. En Oracle8 y siguientes versiones conviene emplear "clob" y "nlob" para almacenar grandes cantidades de datos alfanuméricos.

8) clob (Character Large OBject) y nclob: puede almacenar hasta 128 terabytes de datos de caracteres en la base de datos

9) blob (Binary Large OBject): puede almacenar hasta 128 terabytes de datos de binarios (imágenes, video clips, sonidos etc.)

Si intentamos almacenar en un campo alfanumérico una cadena de caracteres de mayor longitud que la definida, aparece un mensaje indicando que el valor es demasiado grande y la sentencia no se ejecuta.

Por ejemplo, si definimos un campo de tipo varchar2(10) y le asignamos la cadena 'Aprenda PHP' (11 caracteres), aparece un mensaje y la sentencia no se ejecuta.

Si ingresamos un valor numérico (omitiendo las comillas), lo convierte a cadena y lo ingresa como tal.

Por ejemplo, si en un campo definido como varchar2(5) ingresamos el valor 12345, lo toma como si hubiésemos tipeado '12345', igualmente, si ingresamos el valor 23.56, lo convierte a '23.56'. Si el valor numérico, al ser convertido a cadena supera la longitud definida, aparece un mensaje de error y la sentencia no se ejecuta.

Es importante elegir el tipo de dato adecuado según el caso.

Para almacenar cadenas que varían en su longitud, es decir, no todos los registros tendrán la misma longitud en un campo determinado, se emplea "varchar2" en lugar de "char".

Por ejemplo, en campos que guardamos nombres y apellidos, no todos los nombres y apellidos tienen la misma longitud.

Para almacenar cadenas que no varían en su longitud, es decir, todos los registros tendrán la misma longitud en un campo determinado, se emplea "char".

Por ejemplo, definimos un campo "codigo" que constará de 5 caracteres, todos los registros tendrán un código de 5 caracteres, ni más ni menos.

Para almacenar valores superiores a 4000 caracteres se debe emplear "long".

# 16 - Tipos de datos numéricos

Ya explicamos que al crear una tabla debemos elegir la estructura adecuada, esto es, definir los campos y sus tipos más precisos, según el caso.

Los valores numéricos no se ingresan entre comillas. Se utiliza el punto como separador de decimales.

Para almacenar valores NUMERICOS Oracle dispone de dos tipos de datos:

1) number(t,d): para almacenar valores enteros o decimales, positivos o negativos. Su rango va de 1.0 x 10-130 hasta 9.999...(38 nueves). Definimos campos de este tipo cuando queremos almacenar valores numéricos con los cuales luego realizaremos operaciones matemáticas, por ejemplo, cantidades, precios, etc.

El parámetro "t" indica el número total de dígitos (contando los decimales) que contendrá el número como máximo (es la precisión). Su rango va de 1 a 38. El parámetro "d" indica el máximo de dígitos decimales (escala). La escala puede ir de -84 a 127. Para definir número enteros, se puede omitir el parámetro "d" o colocar un 0.

Un campo definido "number(5,2)" puede contener cualquier número entre -999.99 y 999.99.

Para especificar número enteros, podemos omitir el parámetro "d" o colocar el valor 0.

Si intentamos almacenar un valor mayor fuera del rango permitido al definirlo, tal valor no se carga, aparece un mensaje indicando tal situación y la sentencia no se ejecuta.

Por ejemplo, si definimos un campo de tipo "number(4,2)" e intentamos guardar el valor 123.45, aparece un mensaje indicando que el valor es demasiado grande para la columna. Si ingresamos un valor con más decimales que los definidos, el valor se carga pero con la cantidad de decimales permitidos, los dígitos sobrantes se omiten.

2) binary\_float y binary\_double: almacena números flotantes con mayor precisión:

Value BINARY\_FLOAT BINARY\_DOUBLE

Maximum positive finite value 3.40282E+38F 1.79769313486231E+308

Minimum positive finite value 1.17549E-38F 2.22507485850720E-308

Para ambos tipos numéricos:

- si ingresamos un valor con más decimales que los permitidos, redondea al más cercano.

- si intentamos ingresar un valor fuera de rango, no lo acepta.

- si ingresamos una cadena, Oracle intenta convertirla a valor numérico, si dicha cadena consta solamente de dígitos, la conversión se realiza, luego verifica si está dentro del rango, si es así, la ingresa, sino, muestra un mensaje de error y no ejecuta la sentencia. Si la cadena contiene caracteres que Oracle no puede convertir a valor numérico, muestra un mensaje de error y la sentencia no se ejecuta.

Por ejemplo, definimos un campo de tipo "numberl(5,2)", si ingresamos la cadena '12.22', la convierte al valor numérico 12.22 y la ingresa; si intentamos ingresar la cadena '1234.56', la convierte al valor numérico 1234.56, pero como el máximo valor permitido es 999.99, muestra un mensaje indicando que está fuera de rango. Si intentamos ingresar el valor '12y.25', Oracle no puede realizar la conversión y muestra un mensaje de error.

# 17 - Ingresar algunos campos

Hemos aprendido a ingresar registros listando todos los campos y colocando valores para todos y cada uno de ellos luego de "values".

Si ingresamos valores para todos los campos, podemos omitir la lista de nombres de los campos.

Por ejemplo, si tenemos creada la tabla "libros" con los campos "titulo", "autor" y "editorial", podemos ingresar un registro de la siguiente manera:

insert into libros values ('Uno','Richard Bach','Planeta');

También es posible ingresar valores para algunos campos. Ingresamos valores solamente para los campos "titulo" y "autor":

insert into libros (titulo, autor)

values ('El aleph','Borges');

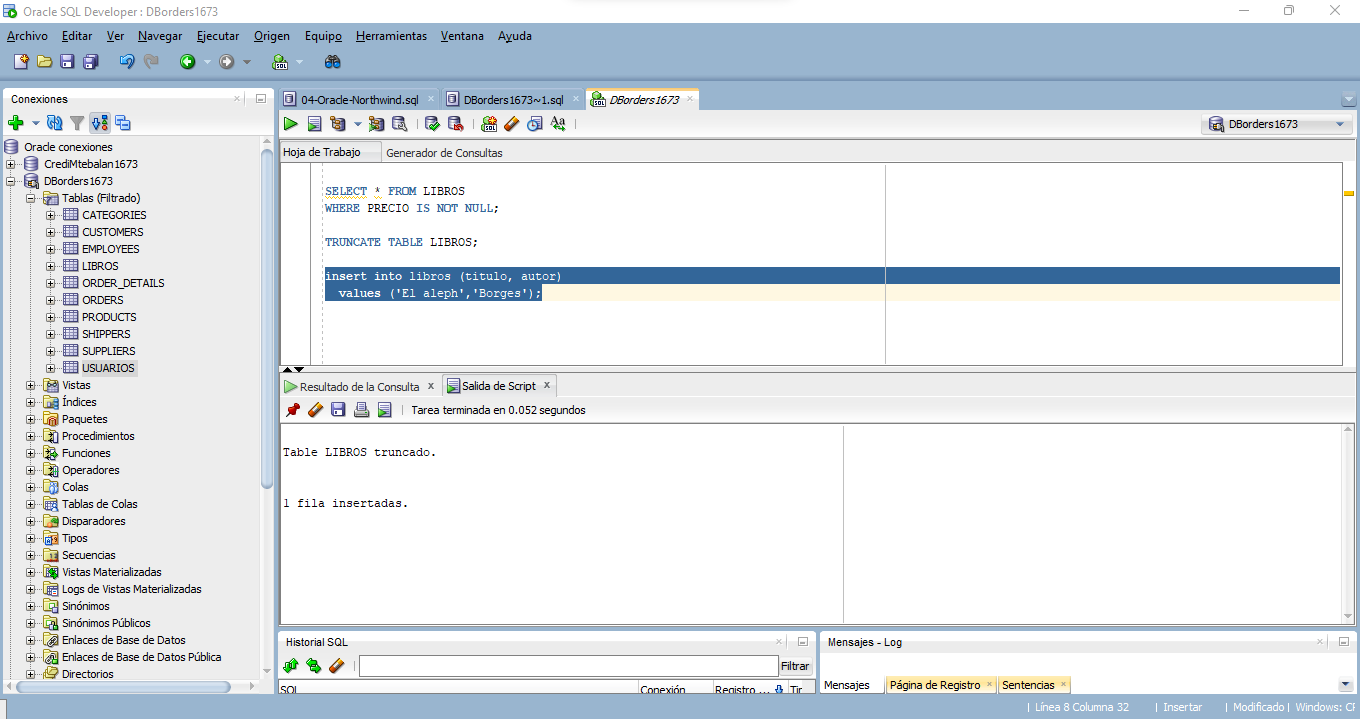
Oracle almacenará el valor "null" en el campo "editorial", para el cual no hemos explicitado un valor.

Al ingresar registros debemos tener en cuenta:

- la lista de campos debe coincidir en cantidad y tipo de valores con la lista de valores luego de "values". Si se listan más (o menos) campos que los valores ingresados, aparece un mensaje de error y la sentencia no se ejecuta.

- si ingresamos valores para todos los campos podemos obviar la lista de campos.

- podemos omitir valores para los campos que permitan valores nulos (se guardará "null"); si omitimos el valor para un campo "not null", la sentencia no se ejecuta.



# 18 - Valores por defecto (default)

Hemos visto que si al insertar registros no se especifica un valor para un campo que admite valores nulos, se ingresa automáticamente "null". A este valor se le denomina valor por defecto o predeterminado.

Un valor por defecto se inserta cuando no está presente al ingresar un registro.

Para campos de cualquier tipo no declarados "not null", es decir, que admiten valores nulos, el valor por defecto es "null". Para campos declarados "not null", no existe valor por defecto, a menos que se declare explícitamente con la cláusula "default".

Podemos establecer valores por defecto para los campos cuando creamos la tabla. Para ello utilizamos "default" al definir el campo. Por ejemplo, queremos que el valor por defecto del campo "autor" de la tabla "libros" sea "Desconocido" y el valor por defecto del campo "cantidad" sea "0":

create table libros(

titulo varchar2(40) not null,

autor varchar2(30) default 'Desconocido' not null,

editorial varchar2(20),

precio number(5,2),

cantidad number(3) default 0

);

Si al ingresar un nuevo registro omitimos los valores para el campo "autor" y "cantidad", Oracle insertará los valores por defecto; en "autor" colocará "Desconocido" y en cantidad "0".

Entonces, si al definir el campo explicitamos un valor mediante la cláusula "default", ése será el valor por defecto.

La cláusula "default" debe ir antes de "not null" (si existiese), sino aparece un mensaje de error.

Para ver si los campos de la tabla "libros" tiene definidos valores por defecto y cuáles son, podemos realizar la siguiente consulta:

select column\_name,nullable,data\_default

from user\_tab\_columns where TABLE\_NAME = 'LIBROS';

Muestra una fila por cada campo, en la columna "data\_default" aparece el valor por defecto (si lo tiene), en la columna "nullable" aparece "N" si el campo no está definido "not null" y "Y" si admite valores "null".

También se puede utilizar "default" para dar el valor por defecto a los campos en sentencias "insert", por ejemplo:

insert into libros (titulo,autor,editorial,precio,cantidad)

values ('El gato con botas',default,default,default,100);

Entonces, la cláusula "default" permite especificar el valor por defecto de un campo. Si no se explicita, el valor por defecto es "null", siempre que el campo no haya sido declarado "not null".

Los campos para los cuales no se ingresan valores en un "insert" tomarán los valores por defecto:

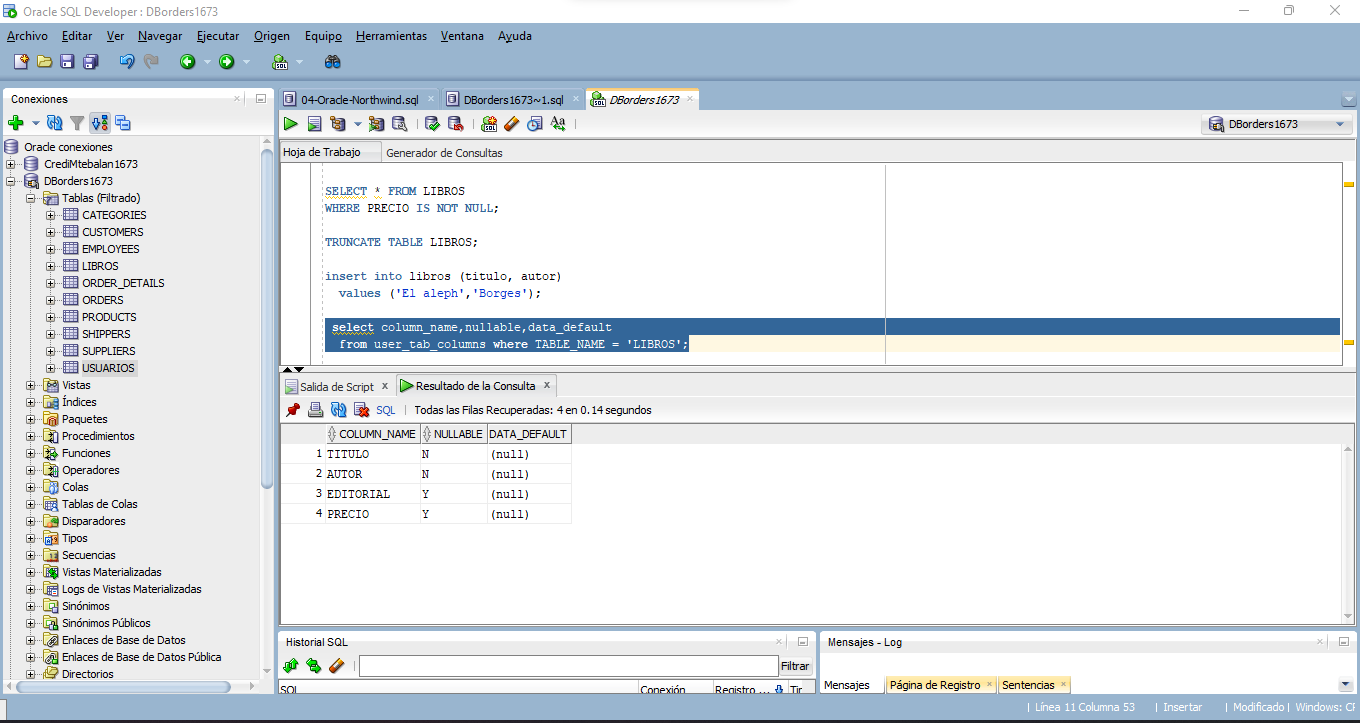
- si permite valores nulos y no tiene cláusula "default", almacenará "null";

- si tiene cláusula "default" (admita o no valores nulos), el valor definido como predeterminado;

- si está declarado explícitamente "not null" y no tiene valor "default", no hay valor por defecto, así que causará un error y el "insert" no se ejecutará.

Un campo sólo puede tener un valor por defecto. Una tabla puede tener todos sus campos con valores por defecto. Que un campo tenga valor por defecto no significa que no admita valores nulos, puede o no admitirlos.

Un campo definido como clave primaria acepta un valor "default", pero no tiene sentido ya que el valor por defecto solamente podrá ingresarse una vez; si intenta ingresarse cuando otro registro ya lo tiene almacenado, aparecerá un mensaje de error indicando que se intenta duplicar la clave.



# 19 - Operadores aritméticos y de concatenación (columnas calculadas)

Dijimos que Oracle tiene 4 tipos de operadores: 1) relacionales o de comparación (los vimos), 2) aritméticos, 3) de concatenación y 4) lógicos (lo veremos más adelante).

Los operadores aritméticos permiten realizar cálculos con valores numéricos.

Son: multiplicación (\*), división (/), suma (+) y resta (-).

Es posible obtener salidas en las cuales una columna sea el resultado de un cálculo y no un campo de una tabla.

Si queremos ver los títulos, precio y cantidad de cada libro escribimos la siguiente sentencia:

select titulo,precio,cantidad

from libros;

Si queremos saber el monto total en dinero de un título podemos multiplicar el precio por la cantidad por cada título, pero también podemos hacer que Oracle realice el cálculo y lo incluya en una columna extra en la salida:

select titulo, precio,cantidad,

precio\*cantidad

from libros;

Si queremos saber el precio de cada libro con un 10% de descuento podemos incluir en la sentencia los siguientes cálculos:

select titulo,precio,

precio-(precio\*0.1)

from libros;

También podemos actualizar los datos empleando operadores aritméticos:

update libros set precio=precio-(precio\*0.1);

Para concatenar cadenas de caracteres existe el operador de concatenación ||.

Para concatenar el título y el autor de cada libro usamos el operador de concatenación ("||"):

select titulo||'-'||autor

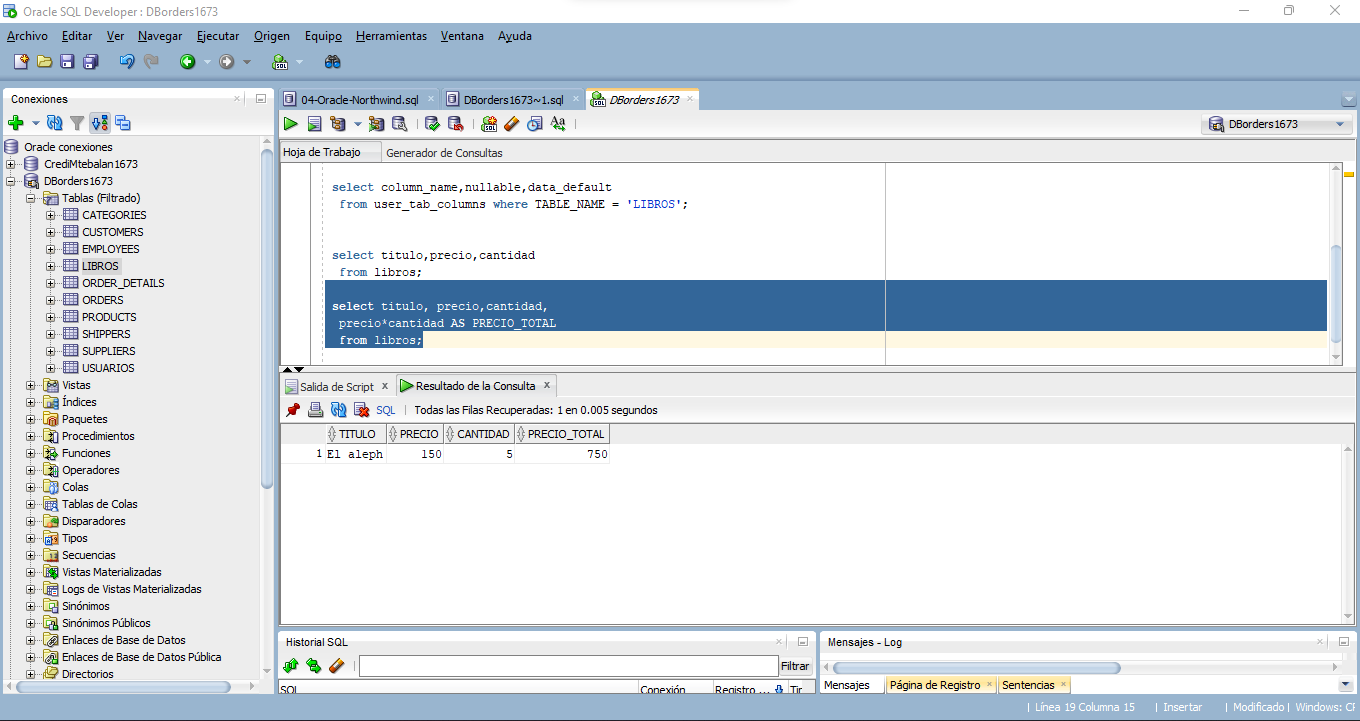
from libros;

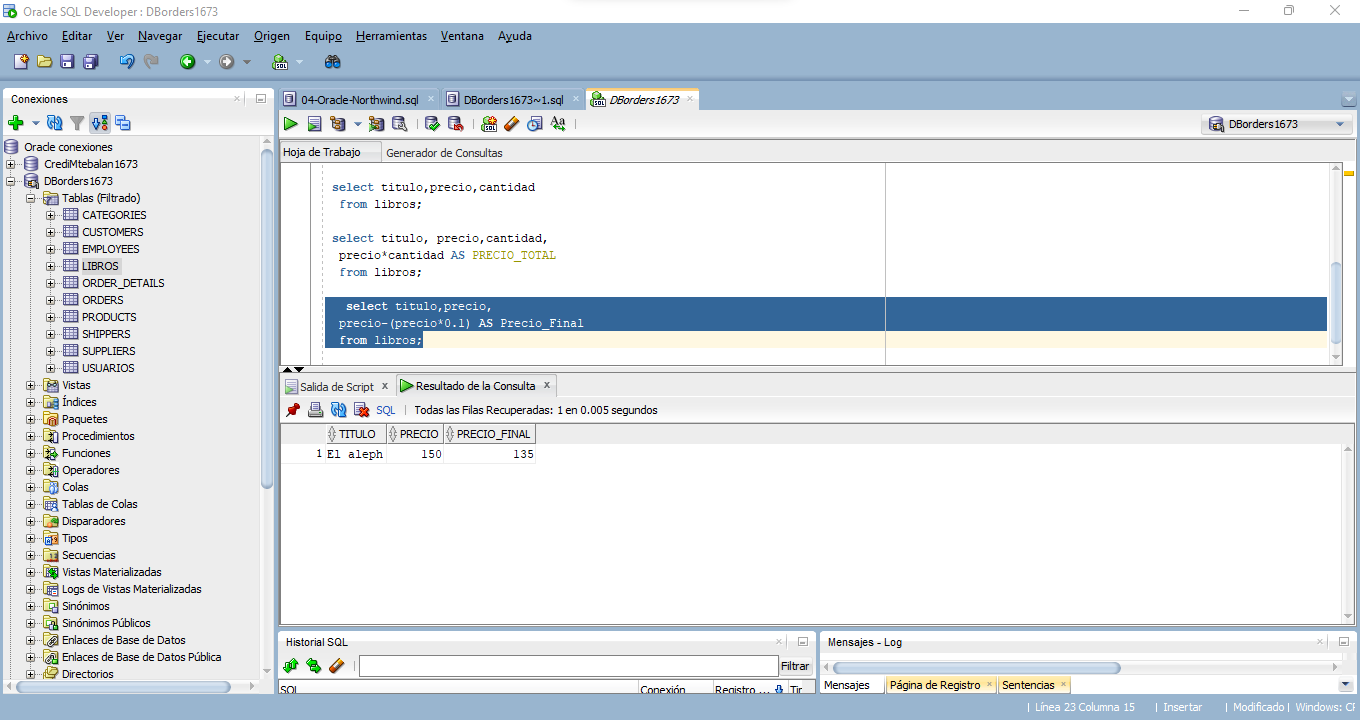
Note que concatenamos además un guión para separar los campos.

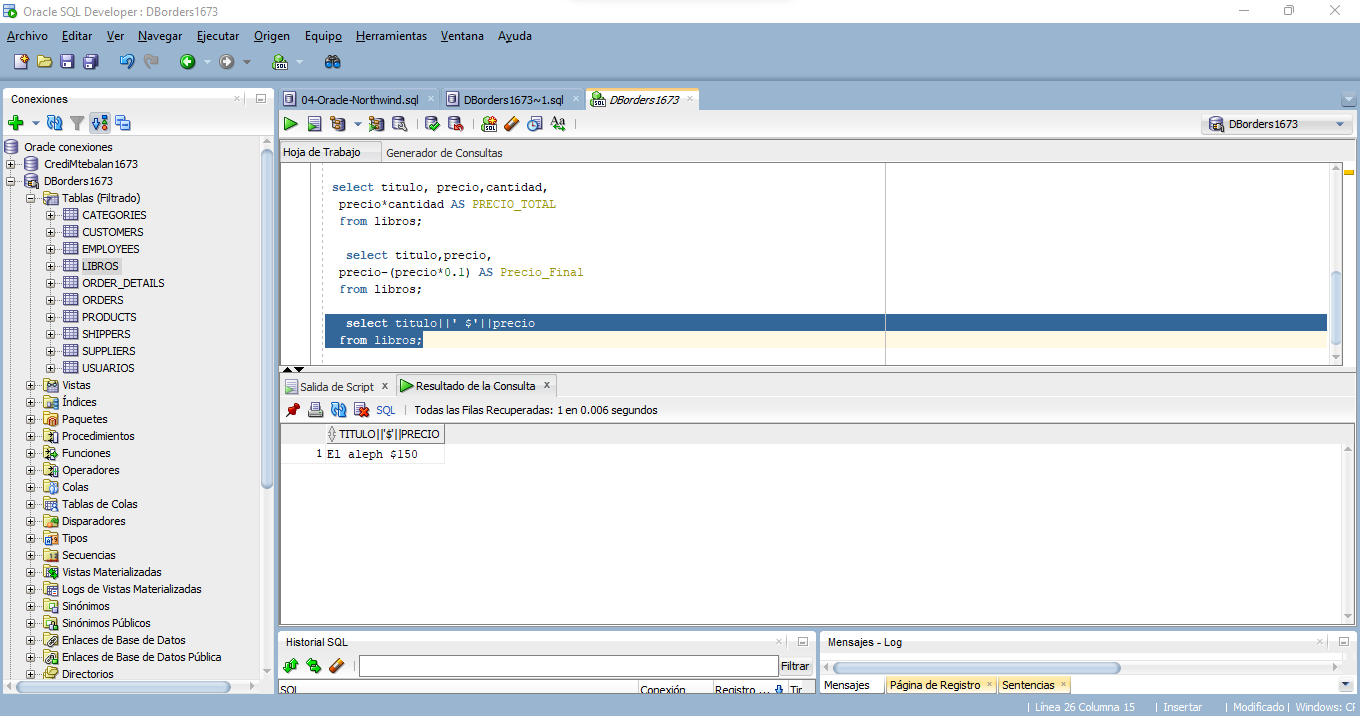
Oracle puede convertir automáticamente valores numéricos a cadenas para una concatenación; por ejemplo, en el siguiente ejemplo mostramos el título y precio de cada libro concatenado con el operador "||":

select titulo||' $'||precio

from libros;







# 20 - Alias (encabezados de columnas)

tenemos la tabla "libros" con un campo "cantidad" (entre otros) en el cual se almacena la cantidad de libros en stock; queremos que al mostrar la información de dicha tabla aparezca como encabezado del campo "cantidad" el texto "stock", para ello colocamos un alias de la siguiente manera:

select titulo,

cantidad as stock,

precio

from libros;

Para reemplazar el nombre de un campo del encabezado por otro, se coloca la palabra clave "as" seguido del texto del encabezado.

Si el alias consta de una sola cadena las comillas no son necesarias, pero si contiene más de una palabra, es necesario colocarla entre comillas dobles:

select titulo,

cantidad as "stock disponible",

precio

from libros;

También se puede crear un alias para columnas calculadas. Por ejemplo:

select titulo,precio,

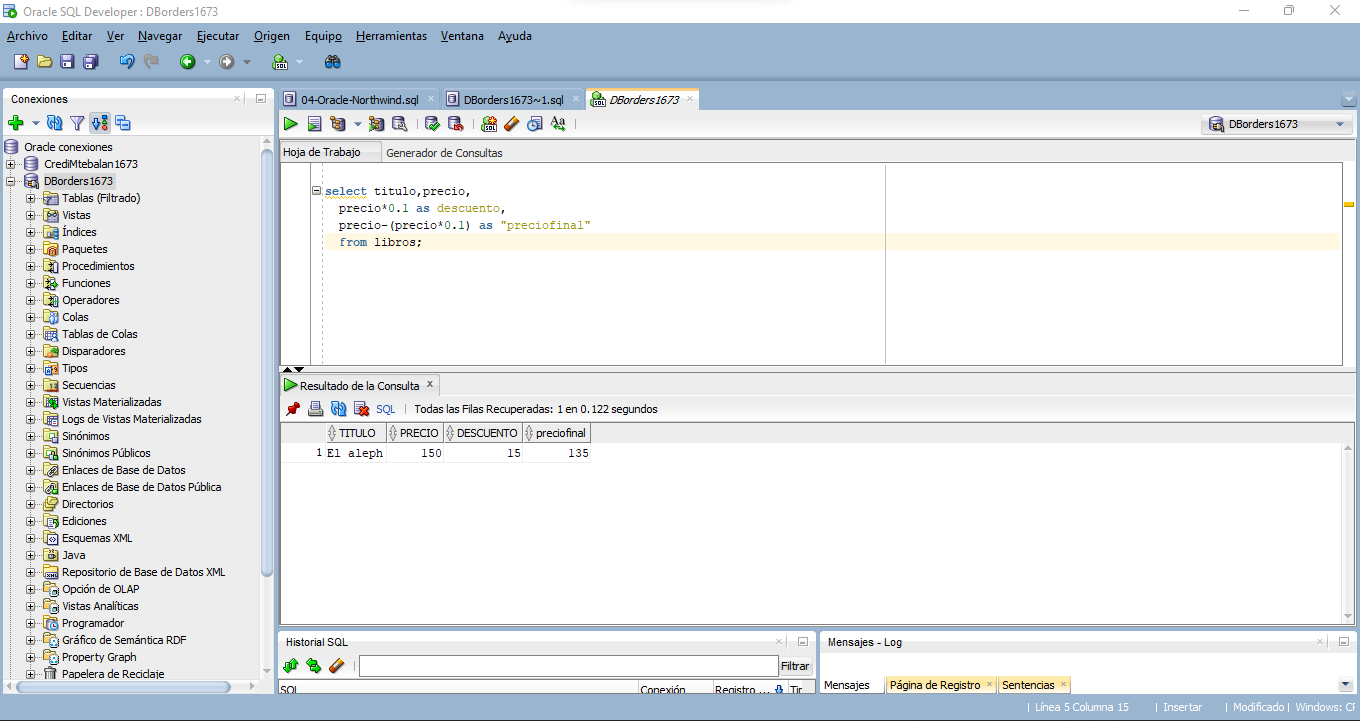
precio\*0.1 as descuento,

precio-(precio\*0.1) as "preciofinal"

from libros;

La palabra clave "as" es opcional, pero es conveniente usarla.

Entonces, un "alias" se usa como nombre de un campo o de una expresión. En estos casos, son opcionales, sirven para hacer más comprensible el resultado.



# 21 - Funciones string

Las funciones de manejo de caracteres alfanuméricos aceptan argumentos de tipo caracter y retornan caracteres o valores numéricos.

Las siguientes son algunas de las funciones que ofrece Oracle para trabajar con cadenas de caracteres:

- chr(x): retorna un caracter equivalente al código enviado como argumento "x". Ejemplo:

select chr(65) from dual;-- retorna 'A'.

select chr(100) from dual;-- retorna 'd'.

Acotaciones

La tabla DUAL es una tabla especial de una sola columna presente de manera predeterminada en todas las instalaciones de bases de datos de Oracle. Se utiliza cuando queremos hacer un select que no necesita consultar tablas. La tabla tiene una sola columna VARCHAR2(1) llamada DUMMY que tiene un valor de 'X'

- concat(cadena1,cadena2): concatena dos cadenas de caracteres; es equivalente al operador ||. Ejemplo:

select concat('Buenas',' tardes') from dual;--retorna 'Buenas tardes'.

- initcap(cadena): retorna la cadena enviada como argumento con la primera letra (letra capital) de cada palabra en mayúscula. Ejemplo:

select initcap('buenas tardes alumno') from dual;--retorna 'Buenas Tardes Alumno'.

- lower(cadena): retorna la cadena enviada como argumento en minúsculas. "lower" significa reducir en inglés. Ejemplo:

select lower('Buenas tardes ALUMNO') from dual;--retorna "buenas tardes alumno".

- upper(cadena): retorna la cadena con todos los caracteres en mayúsculas. Ejemplo:

select upper('www.oracle.com') from dual;-- 'WWW.ORACLE.COM'

- lpad(cadena,longitud,cadenarelleno): retorna la cantidad de caracteres especificados por el argumento "longitud", de la cadena enviada como primer argumento (comenzando desde el primer caracter); si "longitud" es mayor que el tamaño de la cadena enviada, rellena los espacios restantes con la cadena enviada como tercer argumento (en caso de omitir el tercer argumento rellena con espacios); el relleno comienza desde la izquierda. Ejemplos:

select lpad('alumno',10,'xyz') from dual;-- retorna 'xyzxalumno'

select lpad('alumno',4,'xyz') from dual;-- retorna 'alum'

- rpad(cadena,longitud,cadenarelleno): retorna la cantidad de caracteres especificados por el argumento "longitud", de la cadena enviada como primer argumento (comenzando desde el primer caracter); si "longitud" es mayor que el tamaño de la cadena enviada, rellena los espacios restantes con la cadena enviada como tercer argumento (en caso de omitir el tercer argumento rellena con espacios); el relleno comienza desde la derecha (último caracter). Ejemplos:

select rpad('alumno',10,'xyz') from dual;-- retorna 'alumnoxyzx'

select rpad('alumno',4,'xyz') from dual;-- retorna 'alum'

- ltrim(cadena1,cadena2): borra todas las ocurrencias de "cadena2" en "cadena1", si se encuentran al comienzo; si se omite el segundo argumento, se eliminan los espacios. Ejemplo:

select ltrim('la casa de la cuadra','la') from dual;-- ' casa de la cuadra'

select ltrim(' es la casa de la cuadra','la') from dual;-- no elimina ningún caracter

select ltrim(' la casa') from dual;-- 'la casa'

- rtrim(cadena1,cadena2): borra todas las ocurrencias de "cadena2" en "cadena1", si se encuentran por la derecha (al final de la cadena); si se omite el segundo argumento, se borran los espacios. Ejemplo:

select rtrim('la casa lila','la') from dual;-- 'la casa li'

select rtrim('la casa lila ','la') from dual;-- no borra ningún caracter

select rtrim('la casa lila ') from dual; --'la casa lila'

- trim(cadena): retorna la cadena con los espacios de la izquierda y derecha eliminados. "Trim" significa recortar. Ejemplo:

select trim(' oracle ') from dual;--'oracle'

- replace(cadena,subcade1,subcade2): retorna la cadena con todas las ocurrencias de la subcadena de reemplazo (subcade2) por la subcadena a reemplazar (subcae1). Ejemplo:

select replace('xxx.oracle.com','x','w') from dual;

retorna "www.oracle.com'.

- substr(cadena,inicio,longitud): devuelve una parte de la cadena especificada como primer argumento, empezando desde la posición especificada por el segundo argumento y de tantos caracteres de longitud como indica el tercer argumento. Ejemplo:

select substr('www.oracle.com',1,10) from dual;-- 'www.oracle'

select substr('www.oracle.com',5,6) from dual;-- 'oracle'

- length(cadena): retorna la longitud de la cadena enviada como argumento. "lenght" significa longitud en inglés. Ejemplo:

select length('www.oracle.com') from dual;-- devuelve 14.

- instr (cadena,subcadena): devuelve la posición de comienzo (de la primera ocurrencia) de la subcadena especificada en la cadena enviada como primer argumento. Si no la encuentra retorna 0. Ejemplos:

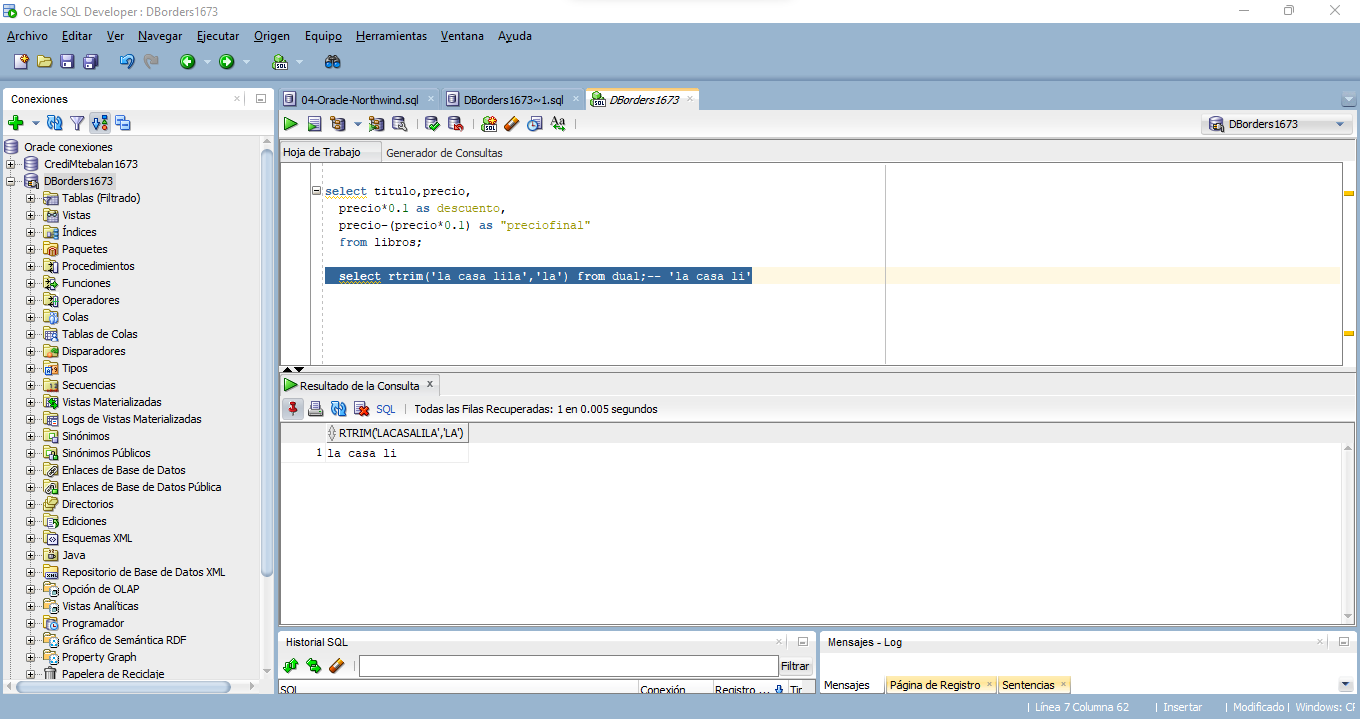
select instr('Jorge Luis Borges','or') from dual;-- 2

select instr('Jorge Luis Borges','ar') from dual;-- 0, no se encuentra

- translate(): reemplaza cada ocurrencia de una serie de caracteres con otra serie de caracteres. La diferencia con "replace" es que aquella trabaja con cadenas de caracteres y reemplaza una cadena completa por otra, en cambio "translate" trabaja con caracteres simples y reemplaza varios. En el siguiente ejemplo se especifica que se reemplacen todos los caracteres "O" por el caracter "0", todos los caracteres "S" por el caracter "5" y todos los caracteres "G" por "6":

select translate('JORGE LUIS BORGES','OSG','056') from dual;--'J0R6E LUI5 B0R6E5'

Se pueden emplear estas funciones enviando como argumento el nombre de un campo de tipo caracter.



# 22 - Funciones matemáticas.

Las funciones matemáticas realizan operaciones con expresiones numéricas y retornan un resultado, operan con tipos de datos numéricos.

Las funciones numéricas aceptan parámetros de entrada de tipo numérico y retornan valores numéricos.

Oracle tiene algunas funciones para trabajar con números. Aquí presentamos algunas.

- abs(x): retorna el valor absoluto del argumento "x". Ejemplo:

select abs(-20) from dual;--retorna 20.

La tabla dual es una tabla virtual que existe en todas las Bases de datos Oracle.

- ceil(x): redondea a entero, hacia arriba, el argumento "x". Ejemplo:

select ceil(12.34) from dual;--retorna 13.

- floor(x): redondea a entero, hacia abajo, el argumento "x". Ejemplo:

select floor(12.34) from dual; --12

- mod(x,y): devuelve el resto de la división x/y. Ejemplos:

select mod(10,3) from dual;--retorna 1.

select mod(10,2) from dual;--retorna 0.

- power(x,y): retorna el valor de "x" elevado a la "y" potencia. Ejemplo:

select power(2,3) from dual;--retorna 8.

- round(n,d): retorna "n" redondeado a "d" decimales; si se omite el segundo argumento, redondea todos los decimales. Si el segundo argumento es positivo, el número de decimales es redondeado según "d"; si es negativo, el número es redondeado desde la parte entera según el valor de "d". Ejemplos:

select round(123.456,2) from dual;-- retorna "123.46", es decir, redondea desde el segundo decimal.

select round(123.456,1) from dual;-- 123.5, es decir, redondea desde el primer decimal.

select round(123.456,-1) from dual;-- 120, redondea desde el primer valor entero (hacia la izquierda).

select round(123.456,-2) from dual;-- 100, redondea desde el segundo valor entero (hacia la izquierda).

select round(123.456) from dual;-- 123.

- sign(x): si el argumento es un valor positivo, retorna 1, si es negativo, devuelve -1 y 0 si es 0. Ejemplo:

select sign(-120) from dual;--retorna -1

select sign(120) from dual;--retorna 1

- trunc(n,d): trunca un número a la cantidad de decimales especificada por el segundo argumento. Si se omite el segundo argumento, se truncan todos los decimales. Si "d" es negativo, el número es truncado desde la parte entera. Ejemplo:

select trunc(1234.5678,2) from dual;--retorna 1234.56

select trunc(1234.5678,-2) from dual;--retorna 1200

select trunc(1234.5678,-1) from dual;--retorna 1230

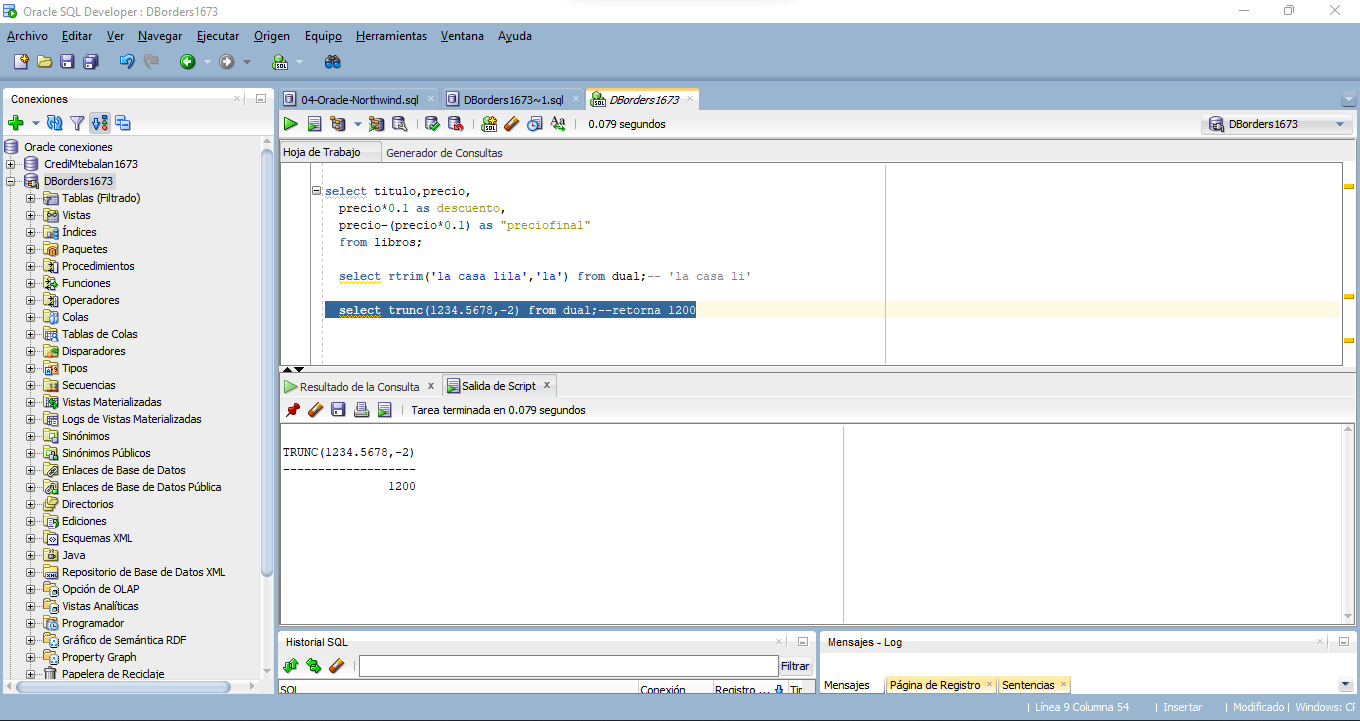
select trunc(1234.5678) from dual;--retorna 1234

- sqrt(x): devuelve la raiz cuadrada del valor enviado como argumento. Ejemplo:

select sqrt(9) from dual;--retorna 3

Oracle dispone de funciones trigonométricas que retornan radianes, calculan seno, coseno, inversas, etc.: acos, asin, atan, atan2, cos, cosh, exp, ln, log, sin, sinh, tan, tanh. No las veremos en detalle.

Se pueden emplear las funciones matemáticas enviando como argumento el nombre de un campo de tipo numérico.



# 23 - Funciones de fechas y horas

Oracle dispone de varias funciones que operan con tipos de datos "date". Estas son algunas:

- add\_months(f,n): agrega a una fecha, un número de meses. Si el segundo argumento es positivo, se le suma a la fecha enviada tal cantidad de meses; si es negativo, se le resta a la fecha enviada tal cantidad de meses. Ejemplo:

select add\_months('10/06/2020',5) from dual; --retorna "10/11/20"

select add\_months('10/06/2020',-5) from dual; --retorna "10/01/20"

select add\_months('30/01/2020',1) from dual;-- retorna "29/02/20" ya que es el último día de ese mes.

- last\_day(f): retorna el ultimo día de mes de la fecha enviada como argumento. Ejemplo:

select last\_day('10/02/2020') from dual;-- "29/02/20"

select last\_day('10/08/2020') from dual;-- "31/08/20"

- months\_between(f1,f2): retorna el numero de meses entre las fechas enviadas como argumento. Ejemplo:

select months\_between('19/05/2020','21/06/20') from dual;-- retorna -1,06451613

- next\_day(fecha,dia): retorna una fecha correspondiente al primer día especificado en "dia" luego de la fecha especificada. En el siguiente ejemplo se busca el lunes siguiente a la fecha especificada:

select next\_day('10/08/2020','LUNES') from dual; -- 17/08/20

- current\_date: retorna la fecha actual. Ejemplo:

select current\_date from dual;

- current\_timestamp: retorna la fecha actual

select current\_timestamp from dual;

Retorna: 22/07/20 07:27:20,973000000 AMERICA/BUENOS\_AIRES

- sysdate: retorna la fecha y hora actuales en el servidor de Oracle.

-systimestamp: retorna fecha y hora actuales.

select systimestamp from dual;

Retorna 22/07/20 07:27:45,012000000 -03:00

- to\_date: convierte una cadena a tipo de dato "date". Ejemplo:

select to\_date ('05-SEP-2019 10:00 AM','DD-MON-YYYY HH:MI AM') from dual;

Retorna 05/09/19

- to\_char: convierte una fecha a cadena de caracteres. Ejemplo:

select to\_char('10/10/2020')from dual;

- extract(parte,fecha): retorna la parte (especificada por el primer argumento) de una fecha. Puede extraer el año (year), mes (month), día (day), hora (hour), minuto (minute), segundo (second), etc. Ejemplo:

select extract(month from sysdate) from dual;

retorna el número mes de la fecha actual.

En Oracle: Los operadores aritméticos "+" (más) y "-" (menos) pueden emplearse con fechas. Por ejemplos:

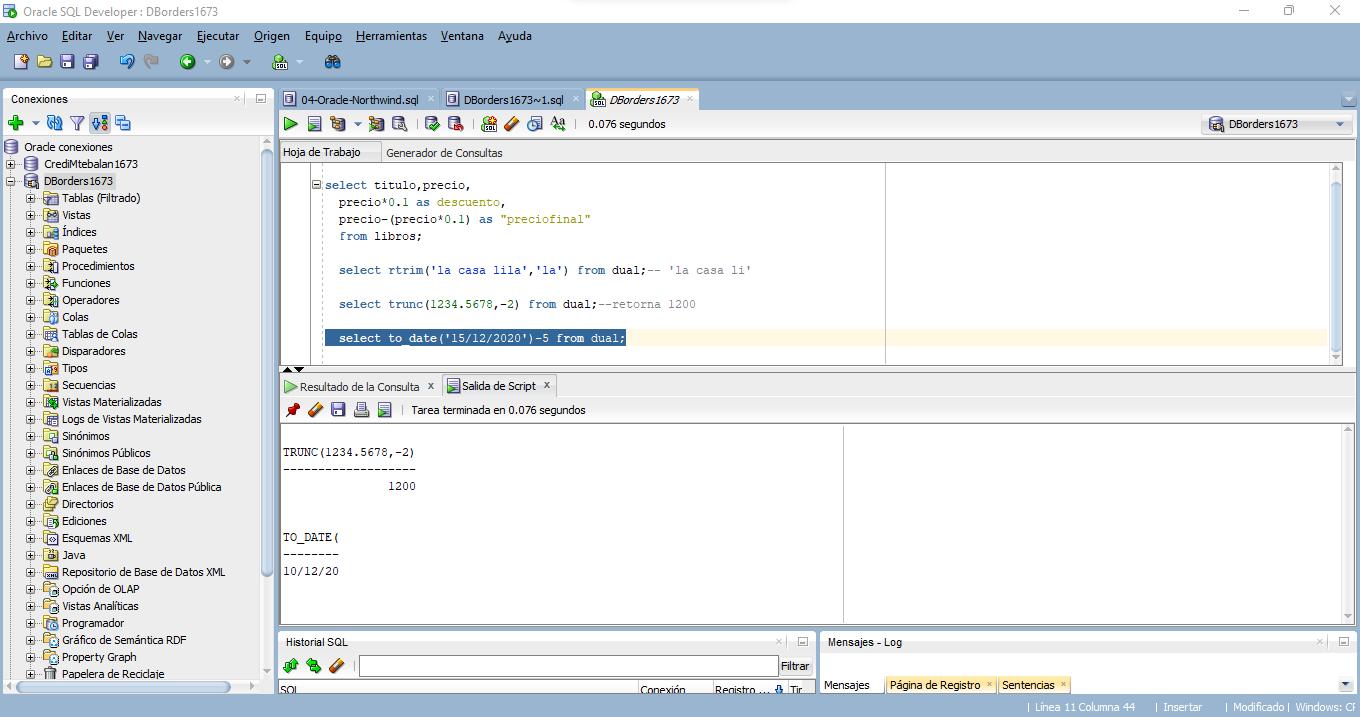
select sysdate-3:

Retorna 3 días antes de la fecha actual.

select to\_date('15/12/2020')-5 from dual;

Retorna 10/12/20

Se pueden emplear estas funciones enviando como argumento el nombre de un campo de tipo date.



# 24 - Ordenar registros (order by)

Podemos ordenar el resultado de un "select" para que los registros se muestren ordenados por algún campo, para ello usamos la cláusula "order by".

La sintaxis básica es la siguiente:

select \*from NOMBRETABLA

order by CAMPO;

Por ejemplo, recuperamos los registros de la tabla "libros" ordenados por el título:

select \*from libros

order by titulo;

Aparecen los registros ordenados alfabéticamente por el campo especificado.

También podemos colocar el número de orden del campo por el que queremos que se ordene en lugar de su nombre, es decir, referenciar a los campos por su posición en la lista de selección. Por ejemplo, queremos el resultado del "select" ordenado por "precio":

select titulo,autor,precio

from libros order by 3;

Si colocamos un número mayor a la cantidad de campos de la lista de selección, aparece un mensaje de error y la sentencia no se ejecuta.

Por defecto, si no aclaramos en la sentencia, los ordena de manera ascendente (de menor a mayor). Podemos ordenarlos de mayor a menor, para ello agregamos la palabra clave "desc":

select \*libros

order by editorial desc;

También podemos ordenar por varios campos, por ejemplo, por "titulo" y "editorial":

select \*from libros

order by titulo,editorial;

Incluso, podemos ordenar en distintos sentidos, por ejemplo, por "titulo" en sentido ascendente y "editorial" en sentido descendente:

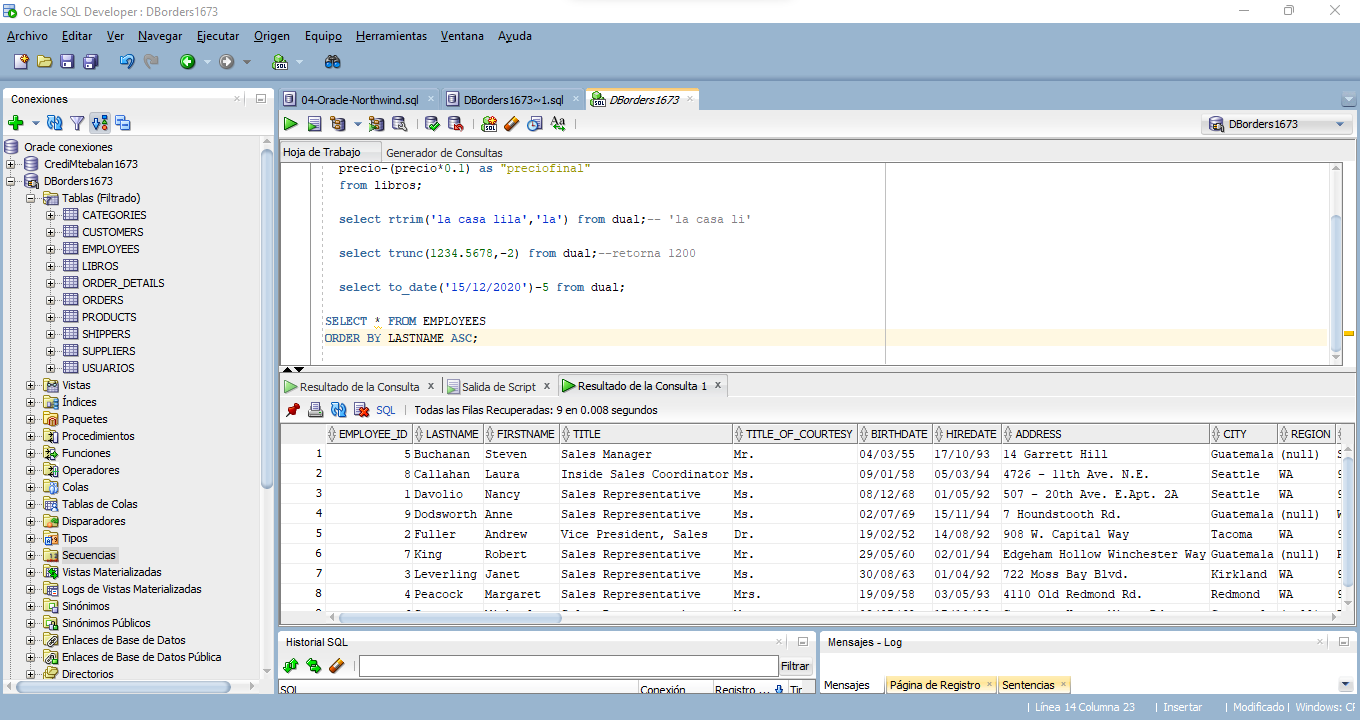
select \*from libros

order by titulo asc, editorial desc;

Debe aclararse al lado de cada campo, pues estas palabras claves afectan al campo inmediatamente anterior.

Es posible ordenar por un campo que no se lista en la selección incluso por columnas calculados.

Se puede emplear "order by" con campos de tipo caracter, numérico y date.



# 25 - Operadores lógicos (and - or - not)

Hasta el momento, hemos aprendido a establecer una condición con "where" utilizando operadores relacionales. Podemos establecer más de una condición con la cláusula "where", para ello aprenderemos los operadores lógicos.

Son los siguientes:

- and, significa "y",

- or, significa "y/o",

- not, significa "no", invierte el resultado

- (), paréntesis

Los operadores lógicos se usan para combinar condiciones.

Si queremos recuperar todos los libros cuyo autor sea igual a "Borges" y cuyo precio no supere los 20 pesos, necesitamos 2 condiciones:

select \*from libros

where (autor='Borges') and

(precio<=20);

Los registros recuperados en una sentencia que une dos condiciones con el operador "and", cumplen con las 2 condiciones.

Queremos ver los libros cuyo autor sea "Borges" y/o cuya editorial sea "Planeta":

select \*from libros

where autor='Borges' or

editorial='Planeta';

En la sentencia anterior usamos el operador "or"; indicamos que recupere los libros en los cuales el valor del campo "autor" sea "Borges" y/o el valor del campo "editorial" sea "Planeta", es decir, seleccionará los registros que cumplan con la primera condición, con la segunda condición y con ambas condiciones.

Los registros recuperados con una sentencia que une dos condiciones con el operador "or", cumplen una de las condiciones o ambas.

Queremos recuperar los libros que NO cumplan la condición dada, por ejemplo, aquellos cuya editorial NO sea "Planeta":

select \*from libros

where not editorial='Planeta';

El operador "not" invierte el resultado de la condición a la cual antecede.

Los registros recuperados en una sentencia en la cual aparece el operador "not", no cumplen con la condición a la cual afecta el "NOT".

Los paréntesis se usan para encerrar condiciones, para que se evalúen como una sola expresión.

Cuando explicitamos varias condiciones con diferentes operadores lógicos (combinamos "and", "or") permite establecer el orden de prioridad de la evaluación; además permite diferenciar las expresiones más claramente.

Por ejemplo, las siguientes expresiones devuelven un resultado diferente:

select \*from libros

where (autor='Borges') or

(editorial='Paidos' and precio<20);

select \*from libros

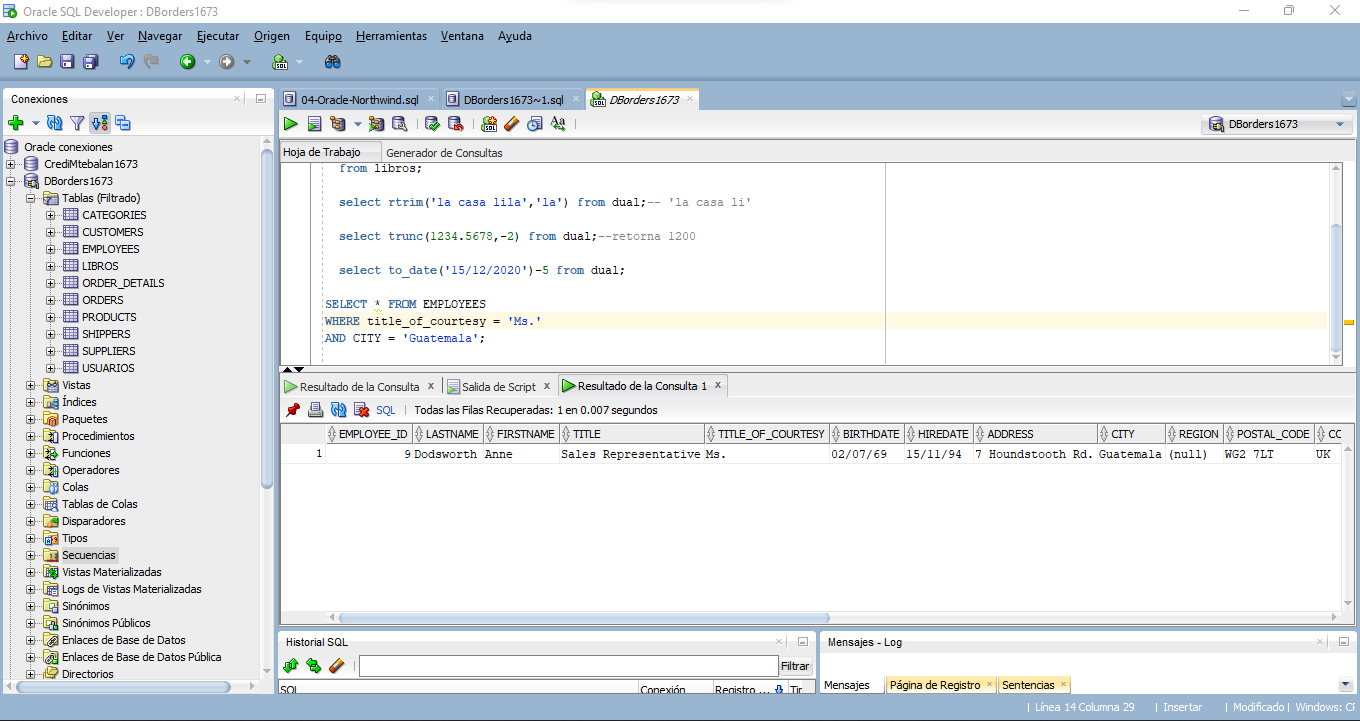
where (autor='Borges' or editorial='Paidos') and

(precio<20);

Si bien los paréntesis no son obligatorios en todos los casos, se recomienda utilizarlos para evitar confusiones.

El orden de prioridad de los operadores lógicos es el siguiente: "not" se aplica antes que "and" y "and" antes que "or", si no se especifica un orden de evaluación mediante el uso de paréntesis. El orden en el que se evalúan los operadores con igual nivel de precedencia es indefinido, por ello se recomienda usar los paréntesis.

Entonces, para establecer más de una condición en un "where" es necesario emplear operadores lógicos. "and" significa "y", indica que se cumplan ambas condiciones; "or" significa "y/o", indica que se cumpla una u otra condición (o ambas); "not" significa "no.", indica que no se cumpla la condición especificada.



# 26 - Otros operadores relacionales (between)

Hemos visto los operadores relacionales: = (igual), <> (distinto), > (mayor), < (menor), >= (mayor o igual), <= (menor o igual), is null/is not null (si un valor es NULL o no).

Otro operador relacional es "between", trabajan con intervalos de valores.

Hasta ahora, para recuperar de la tabla "libros" los libros con precio mayor o igual a 20 y menor o igual a 40, usamos 2 condiciones unidas por el operador lógico "and":

select \*from libros

where precio>=20 and

precio<=40;

Podemos usar "between" y así simplificar la consulta:

select \*from libros

where precio between 20 and 40;

Averiguamos si el valor de un campo dado (precio) está entre los valores mínimo y máximo especificados (20 y 40 respectivamente).

"between" significa "entre". Trabaja con intervalo de valores.

Este operador no tiene en cuenta los valores "null".

Si agregamos el operador "not" antes de "between" el resultado se invierte, es decir, se recuperan los registros que están fuera del intervalo especificado. Por ejemplo, recuperamos los libros cuyo precio NO se encuentre entre 20 y 30, es decir, los menores a 20 y mayores a 30:

select \*from libros

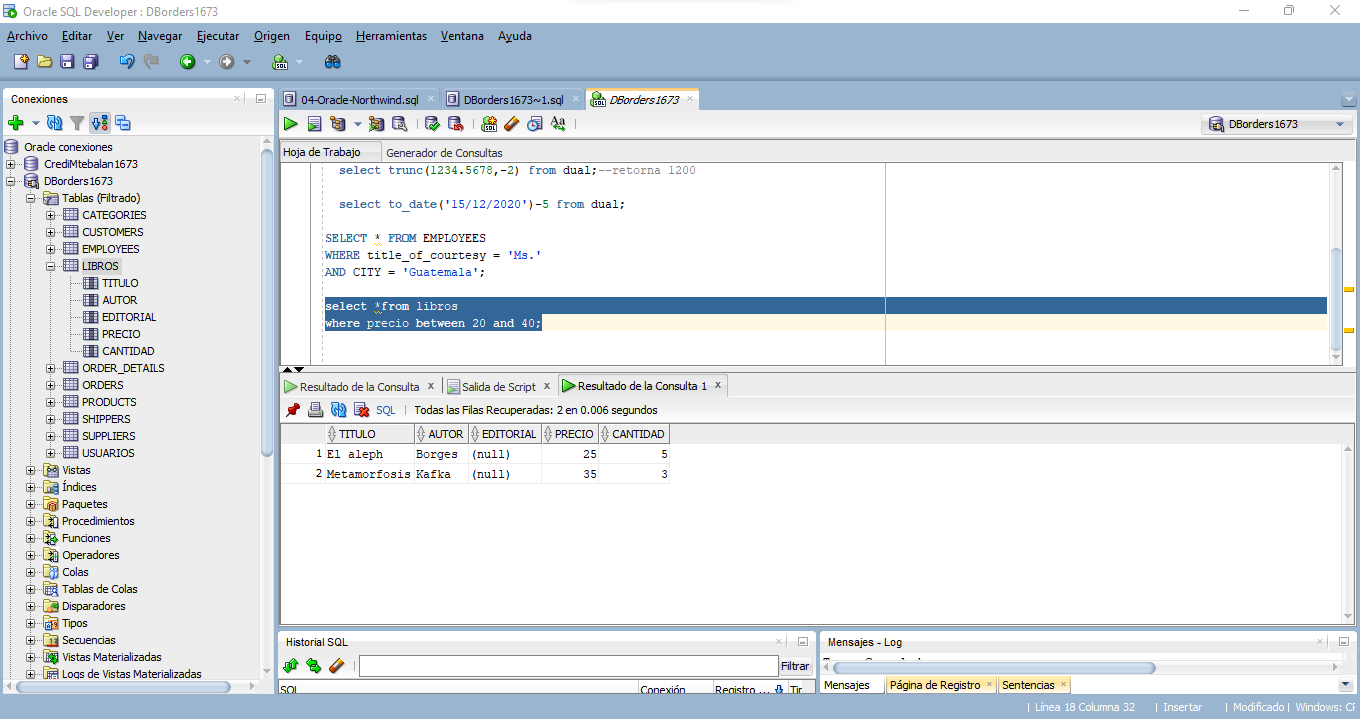
where precio not between 20 and 30;

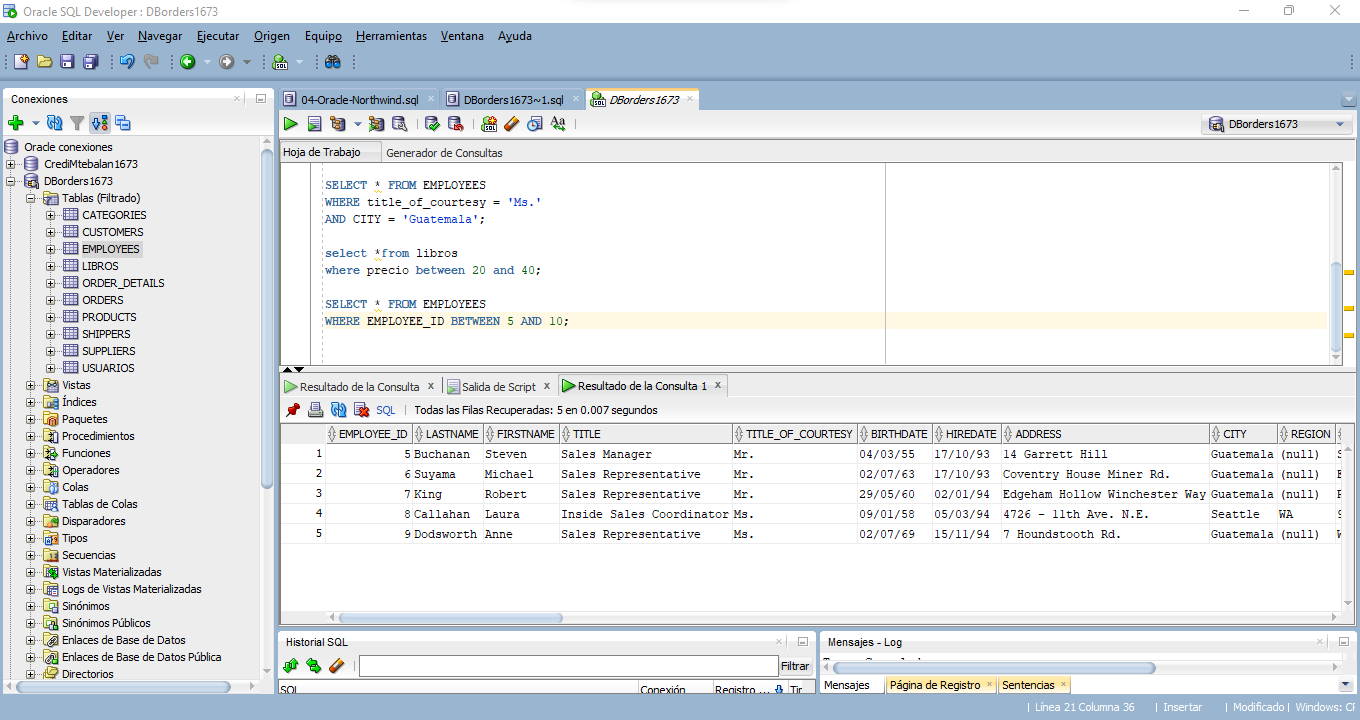
Podemos especificar un intervalo de valores de tipo fecha con "between":

select \*from libros

where edicion between '01/05/2000' and '01/05/2007';

Entonces, empleamos el operador "between" para reducir las condiciones "where".





# 27 - Otros operadores relacionales (in)

Se utiliza "in" para averiguar si el valor de un campo está incluido en una lista de valores especificada.

En la siguiente sentencia usamos "in" para averiguar si el valor del campo autor está incluido en la lista de valores especificada (en este caso, 2 cadenas).

Hasta ahora, para recuperar los libros cuyo autor sea 'Paenza' o 'Borges' usábamos 2 condiciones:

select \*from libros

where autor='Borges' or autor='Paenza';

Podemos usar "in" y simplificar la consulta:

select \*from libros

where autor in('Borges','Paenza');

Para recuperar los libros cuyo autor no sea 'Paenza' ni 'Borges' usábamos:

select \*from libros

where autor<>'Borges' and

autor<>'Paenza';

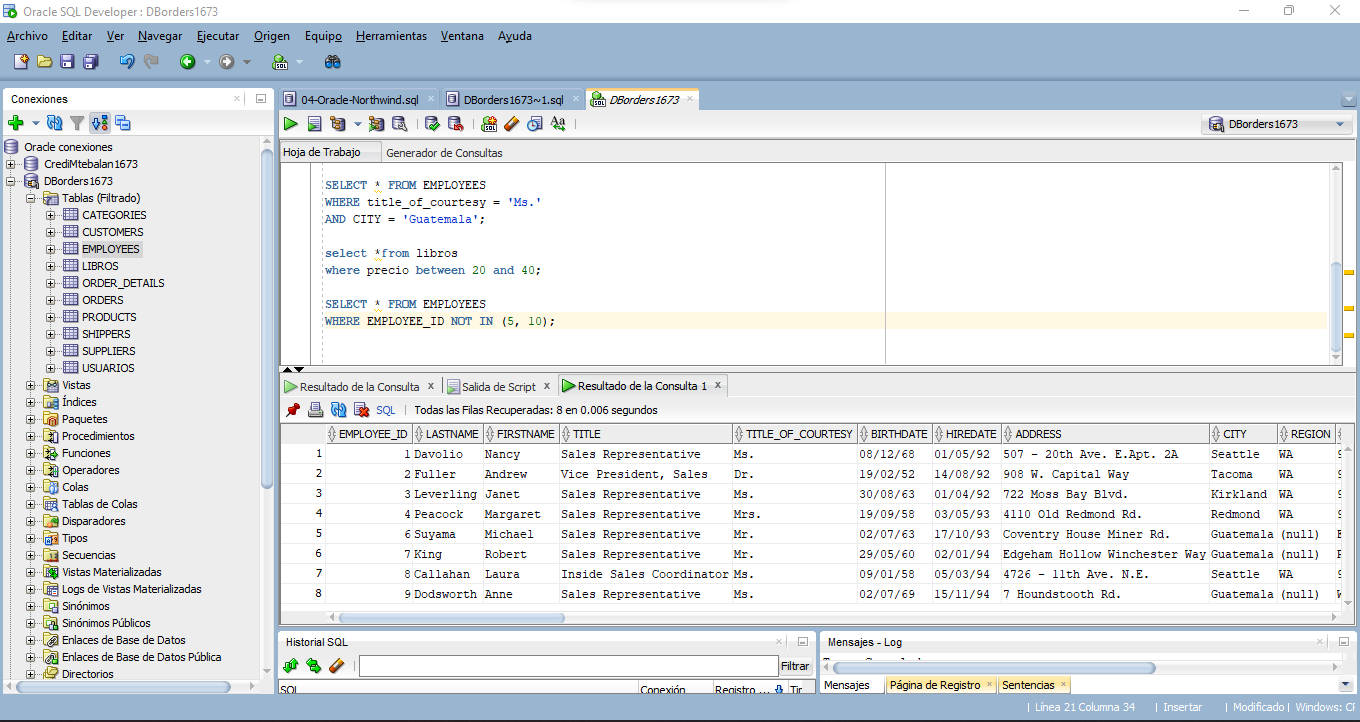
También podemos usar "in" anteponiendo "not":

select \*from libros

where autor not in ('Borges','Paenza');

Empleando "in" averiguamos si el valor del campo está incluido en la lista de valores especificada; con "not" antecediendo la condición, invertimos el resultado, es decir, recuperamos los valores que no se encuentran (no coinciden) con la lista de valores.

Los valores "null" no se consideran.



# 28 - Búsqueda de patrones (like - not like)

Existe un operador relacional que se usa para realizar comparaciones exclusivamente de cadenas, "like" y "not like".

Hemos realizado consultas utilizando operadores relacionales para comparar cadenas. Por ejemplo, sabemos recuperar los libros cuyo autor sea igual a la cadena "Borges":

select \*from libros

where autor='Borges';

El operador igual ("=") nos permite comparar cadenas de caracteres, pero al realizar la comparación, busca coincidencias de cadenas completas, realiza una búsqueda exacta.

Imaginemos que tenemos registrados estos 2 libros:

"El Aleph", "Borges";

"Antologia poetica", "J.L. Borges";

Si queremos recuperar todos los libros de "Borges" y especificamos la siguiente condición:

select \*from libros

where autor='Borges';

sólo aparecerá el primer registro, ya que la cadena "Borges" no es igual a la cadena "J.L. Borges".

Esto sucede porque el operador "=" (igual), también el operador "<>" (distinto) comparan cadenas de caracteres completas. Para comparar porciones de cadenas utilizamos los operadores "like" y "not like".

Entonces, podemos comparar trozos de cadenas de caracteres para realizar consultas. Para recuperar todos los registros cuyo autor contenga la cadena "Borges" debemos tipear:

select \*from libros

where autor like "%Borges%";

El símbolo "%" (porcentaje) reemplaza cualquier cantidad de caracteres (incluyendo ningún caracter). Es un caracter comodín. "like" y "not like" son operadores de comparación que señalan igualdad o diferencia.

Para seleccionar todos los libros que comiencen con "M":

select \*from libros

where titulo like 'M%';

Note que el símbolo "%" ya no está al comienzo, con esto indicamos que el título debe tener como primera letra la "M" y luego, cualquier cantidad de caracteres.

Para seleccionar todos los libros que NO comiencen con "M":

select \*from libros

where titulo not like 'M%';

Así como "%" reemplaza cualquier cantidad de caracteres, el guión bajo "\_" reemplaza un caracter, es otro caracter comodín. Por ejemplo, queremos ver los libros de "Lewis Carroll" pero no recordamos si se escribe "Carroll" o "Carrolt", entonces tipeamos esta condición:

select \*from libros

where autor like "%Carrol\_";

"like" se emplea con tipos de datos caracter y date. Si empleamos "like" con tipos de datos que no son caracteres, Oracle convierte (si es posible) el tipo de dato a caracter. Por ejemplo, queremos buscar todos los libros cuyo precio se encuentre entre 10.00 y 19.99:

select titulo,precio from libros

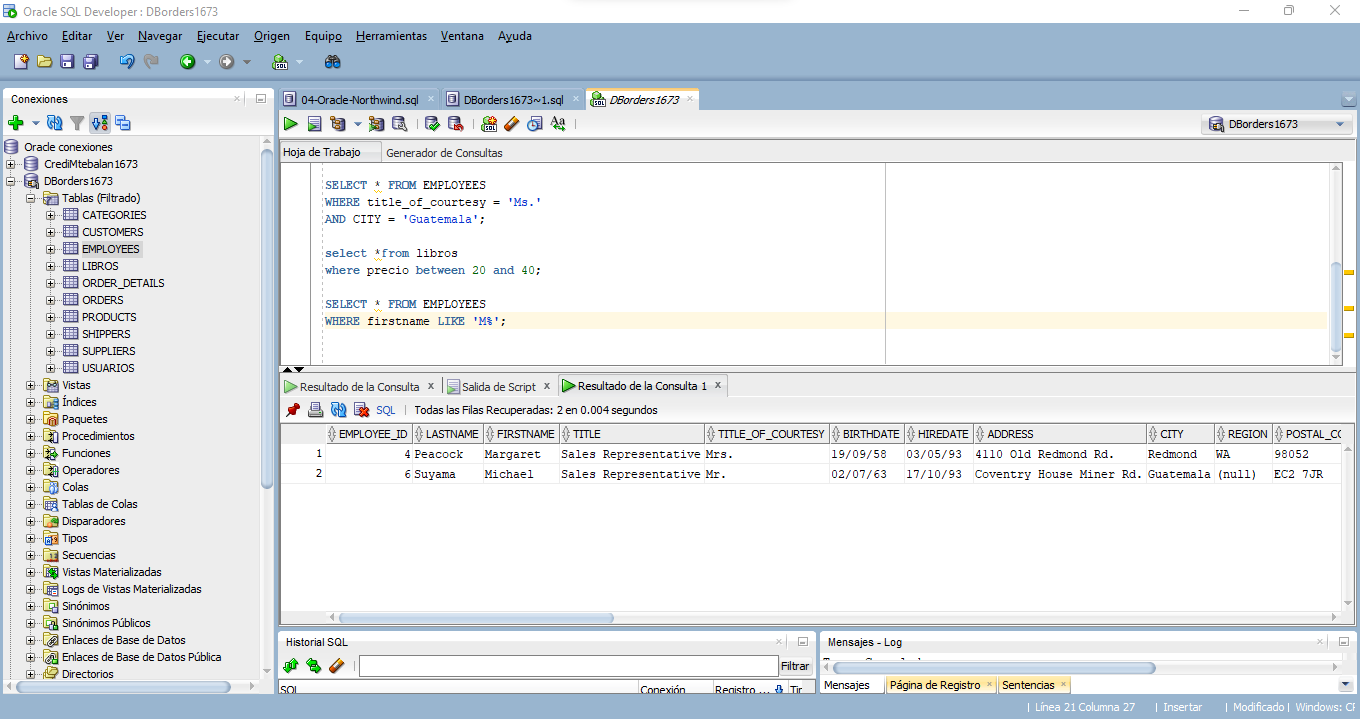
where precio like '1\_,%';

Queremos los libros que NO incluyen centavos en sus precios:

select titulo,precio from libros

where precio not like '%,%';

Los valores nulos no se incluyen en las búsquedas con "like" y "not like".



# 29 - Contar registros (count)

Existen en Oracle funciones que nos permiten contar registros, calcular sumas, promedios, obtener valores máximos y mínimos. Estas funciones se denominan funciones de grupo y operan sobre un conjunto de valores (registros), no con datos individuales y devuelven un único valor.

Imaginemos que nuestra tabla "libros" contiene muchos registros. Para averiguar la cantidad sin necesidad de contarlos manualmente usamos la función "count()":

select count(\*)

from libros;

La función "count()" cuenta la cantidad de registros de una tabla, incluyendo los que tienen valor nulo.

También podemos utilizar esta función junto con la clausula "where" para una consulta más específica. Queremos saber la cantidad de libros de la editorial "Planeta":

select count(\*)

from libros

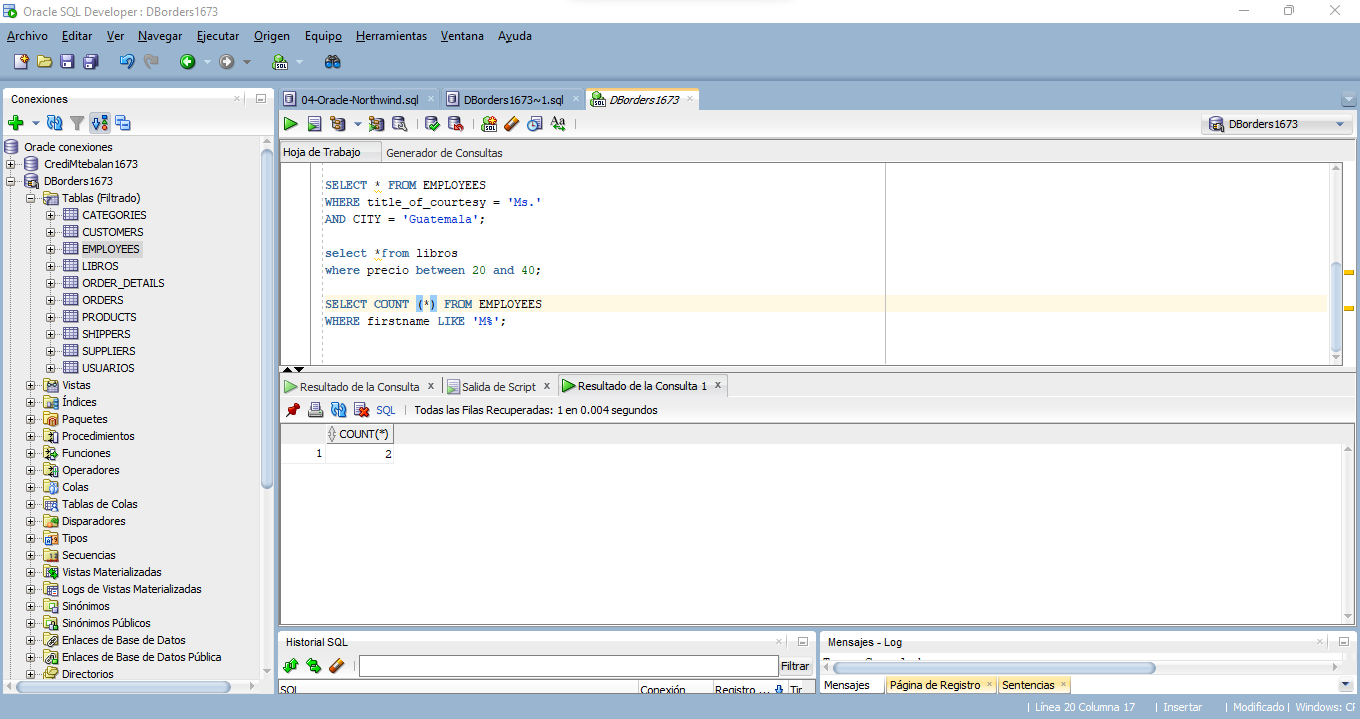
where editorial='Planeta';

Para contar los registros que tienen precio (sin tener en cuenta los que tienen valor nulo), usamos la función "count()" y en los paréntesis colocamos el nombre del campo que necesitamos contar:

select count(precio)

from libros;

Note que "count(\*)" retorna la cantidad de registros de una tabla (incluyendo los que tienen valor "null") mientras que "count(precio)" retorna la cantidad de registros en los cuales el campo "precio" no es nulo. No es lo mismo. "count(\*)" cuenta registros, si en lugar de un asterisco colocamos como argumento el nombre de un campo, se contabilizan los registros cuyo valor en ese campo NO es nulo.



# 30 - Funciones de grupo (count - max - min - sum - avg)

Hemos visto que Oracle dispone de funciones que nos permiten contar registros, calcular sumas, promedios, obtener valores máximos y mínimos, las funciones de grupo. Las funciones de grupo operan sobre un conjunto de valores (registros) y retornan un solo valor.

Ya hemos aprendido una de ellas, "count()", veamos otras.

Se pueden usar en una instrucción "select" y combinarlas con la cláusula "group by" (la veremos posteriormente).

Todas estas funciones retornan "null" si ningún registro cumple con la condicion del "where" (excepto "count" que en tal caso retorna cero).

El tipo de dato del campo determina las funciones que se pueden emplear con ellas.

Las relaciones entre las funciones de agrupamiento y los tipos de datos es la siguiente:

- count: se puede emplear con cualquier tipo de dato.

- min y max: con cualquier tipo de dato.

- sum y avg: sólo en campos de tipo numérico.

La función "sum()" retorna la suma de los valores que contiene el campo especificado. Si queremos saber la cantidad total de libros que tenemos disponibles para la venta, debemos sumar todos los valores del campo "cantidad":

select sum(cantidad)

from libros;

Para averiguar el valor máximo o mínimo de un campo usamos las funciones "max()" y "min()" respectivamente. Queremos saber cuál es el mayor precio de todos los libros:

select max(precio)

from libros;

Entonces, dentro del paréntesis de la función colocamos el nombre del campo del cuál queremos el máximo valor.

La función "avg()" retorna el valor promedio de los valores del campo especificado. Queremos saber el promedio del precio de los libros referentes a "PHP":

select avg(precio)

from libros

where titulo like '%PHP%';

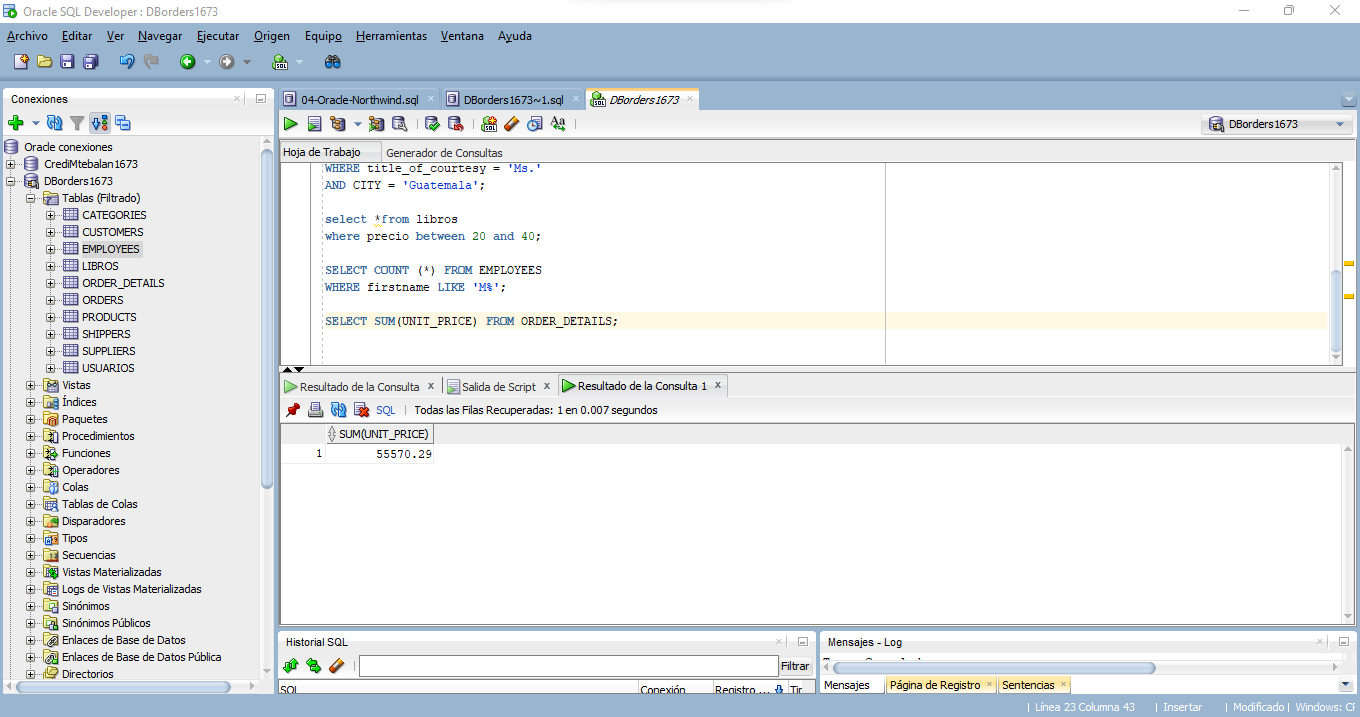
Ahora podemos entender porque estas funciones se denominan "funciones de grupo", porque operan sobre conjuntos de registros, no con datos individuales.

Tratamiento de los valores nulos:

Si realiza una consulta con la función "count" incluyendo entre paréntesis un campo y la tabla contiene 18 registros, 2 de los cuales contienen valor nulo en "precio", el resultado devuelve un total de 16 filas porque no considera aquellos con valor nulo.

Todas las funciones de grupo, excepto "count(\*)", excluye los valores nulos de los campos;

"count(\*)" cuenta todos los registros, incluidos los que contienen "null".



# 31 - Agrupar registros (group by)

Hemos aprendido que las funciones de grupo permiten realizar varios cálculos operando con conjuntos de registros.

Las funciones de grupo solas producen un valor de resumen para todos los registros de un campo.

Podemos generar valores de resumen para un solo campo, combinando las funciones de agregado con la cláusula "group by", que agrupa registros para consultas detalladas.

Queremos saber la cantidad de libros de cada editorial, podemos tipear la siguiente sentencia:

select count(\*) from libros

where editorial='Planeta';

y repetirla con cada valor de "editorial":

select count(\*) from libros

where editorial='Emece';

select count(\*) from libros

where editorial='Paidos';

...

Pero hay otra manera, utilizando la cláusula "group by":

select editorial, count(\*)

from libros

group by editorial;

La instrucción anterior solicita que muestre el nombre de la editorial y cuente la cantidad agrupando los registros por el campo "editorial". Como resultado aparecen los nombres de las editoriales y la cantidad de registros para cada valor del campo.

Los valores nulos se procesan como otro grupo.

Entonces, para saber la cantidad de libros que tenemos de cada editorial, utilizamos la función "count()", agregamos "group by" (que agrupa registros) y el campo por el que deseamos que se realice el agrupamiento, también colocamos el nombre del campo a recuperar; la sintaxis básica es la siguiente:

select CAMPO, FUNCIONDEAGREGADO

from NOMBRETABLA

group by CAMPO;

También se puede agrupar por más de un campo, en tal caso, luego del "group by" se listan los campos, separados por comas. Todos los campos que se especifican en la cláusula "group by" deben estar en la lista de selección.

select CAMPO1, CAMPO2, FUNCIONDEAGREGADO

from NOMBRETABLA

group by CAMPO1,CAMPO2;

Para obtener la cantidad libros con precio no nulo, de cada editorial utilizamos la función "count()" enviándole como argumento el campo "precio", agregamos "group by" y el campo por el que deseamos que se realice el agrupamiento (editorial):

select editorial, count(precio)

from libros

group by editorial;

Como resultado aparecen los nombres de las editoriales y la cantidad de registros de cada una, sin contar los que tienen precio nulo.

Recuerde la diferencia de los valores que retorna la función "count()" cuando enviamos como argumento un asterisco o el nombre de un campo: en el primer caso cuenta todos los registros incluyendo los que tienen valor nulo, en el segundo, los registros en los cuales el campo especificado es no nulo.

Para conocer el total de libros agrupados por editorial:

select editorial, sum(cantidad)

from libros

group by editorial;

Para saber el máximo y mínimo valor de los libros agrupados por editorial:

select editorial,

max(precio) as mayor,

min(precio) as menor

from libros

group by editorial;

Para calcular el promedio del valor de los libros agrupados por editorial:

select editorial, avg(precio)

from libros

group by editorial;

Es posible limitar la consulta con "where".

Si incluye una cláusula "where", sólo se agrupan los registros que cumplen las condiciones.

Vamos a contar y agrupar por editorial considerando solamente los libros cuyo precio sea menor a 30 pesos:

select editorial, count(\*)

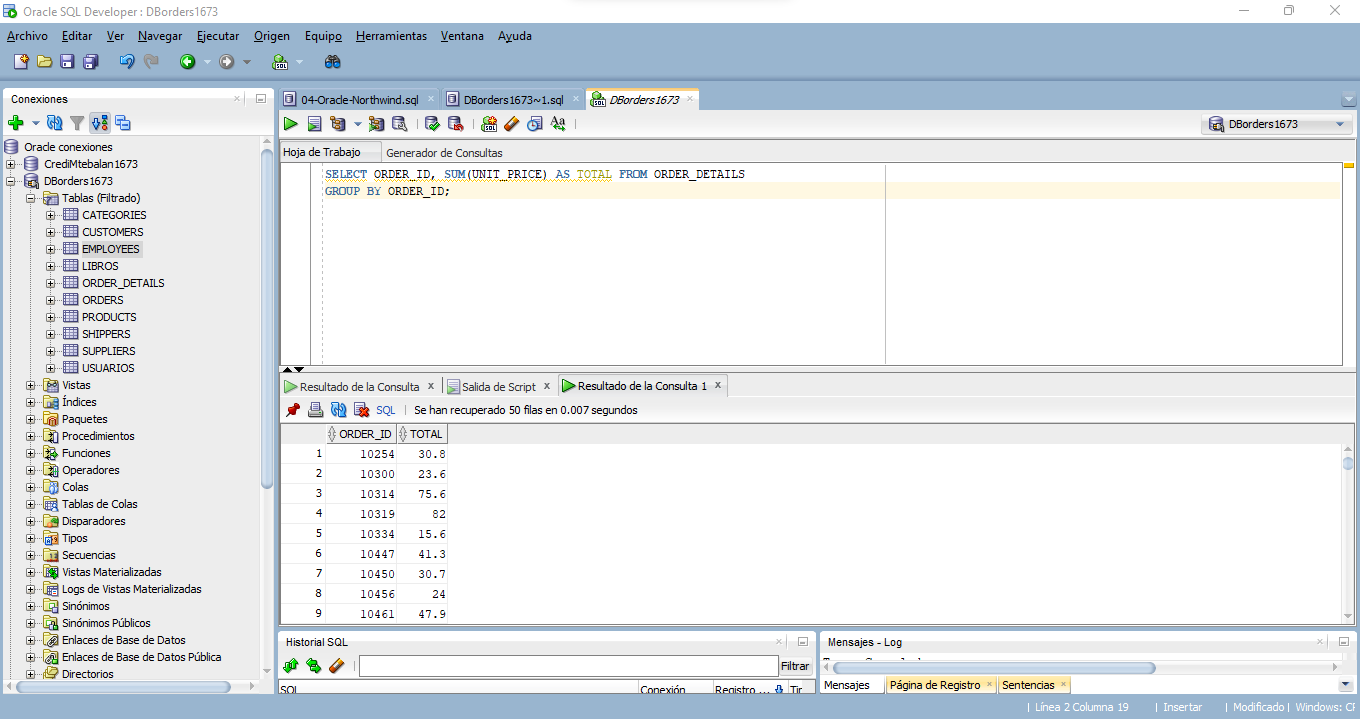
from libros

where precio<30

group by editorial;

Note que las editoriales que no tienen libros que cumplan la condición, no aparecen en la salida.

Entonces, usamos "group by" para organizar registros en grupos y obtener un resumen de dichos grupos. Oracle produce una columna de valores por cada grupo, devolviendo filas por cada grupo especificado.



# 32 - Seleccionar grupos (Having)

Así como la cláusula "where" permite seleccionar (o rechazar) registros individuales; la cláusula "having" permite seleccionar (o rechazar) un grupo de registros.

Si queremos saber la cantidad de libros agrupados por editorial usamos la siguiente instrucción ya aprendida:

select editorial, count(\*)

from libros

group by editorial;

Si queremos saber la cantidad de libros agrupados por editorial pero considerando sólo algunos grupos, por ejemplo, los que devuelvan un valor mayor a 2, usamos la siguiente instrucción:

select editorial, count(\*) from libros

group by editorial

having count(\*)>2;

Se utiliza "having", seguido de la condición de búsqueda, para seleccionar ciertas filas retornadas por la cláusula "group by".

Veamos otros ejemplos. Queremos el promedio de los precios agrupados por editorial, pero solamente de aquellos grupos cuyo promedio supere los 25 pesos:

select editorial, avg(precio) from libros

group by editorial

having avg(precio)>25;

En algunos casos es posible confundir las cláusulas "where" y "having". Queremos contar los registros agrupados por editorial sin tener en cuenta a la editorial "Planeta".

Analicemos las siguientes sentencias:

select editorial, count(\*) from libros

where editorial<>'Planeta'

group by editorial;

select editorial, count(\*) from libros

group by editorial

having editorial<>'Planeta';

Ambas devuelven el mismo resultado, pero son diferentes. La primera, selecciona todos los registros rechazando los de editorial "Planeta" y luego los agrupa para contarlos. La segunda, selecciona todos los registros, los agrupa para contarlos y finalmente rechaza fila con la cuenta correspondiente a la editorial "Planeta".

No debemos confundir la cláusula "where" con la cláusula "having"; la primera establece condiciones para la selección de registros de un "select"; la segunda establece condiciones para la selección de registros de una salida "group by".

Veamos otros ejemplos combinando "where" y "having". Queremos la cantidad de libros, sin considerar los que tienen precio nulo, agrupados por editorial, sin considerar la editorial "Planeta":

select editorial, count(\*) from libros

where precio is not null

group by editorial

having editorial<>'Planeta';

Aquí, selecciona los registros rechazando los que no cumplan con la condición dada en "where", luego los agrupa por "editorial" y finalmente rechaza los grupos que no cumplan con la condición dada en el "having".

Se emplea la cláusula "having" con funciones de grupo, esto no puede hacerlo la cláusula "where". Por ejemplo queremos el promedio de los precios agrupados por editorial, de aquellas editoriales que tienen más de 2 libros:

select editorial, avg(precio) from libros

group by editorial

having count(\*) > 2;

En una cláusula "having" puede haber varias condiciones. Cuando utilice varias condiciones, tiene que combinarlas con operadores lógicos (and, or, not).

Podemos encontrar el mayor valor de los libros agrupados y ordenados por editorial y seleccionar las filas que tengan un valor menor a 100 y mayor a 30:

select editorial, max(precio) as mayor

from libros

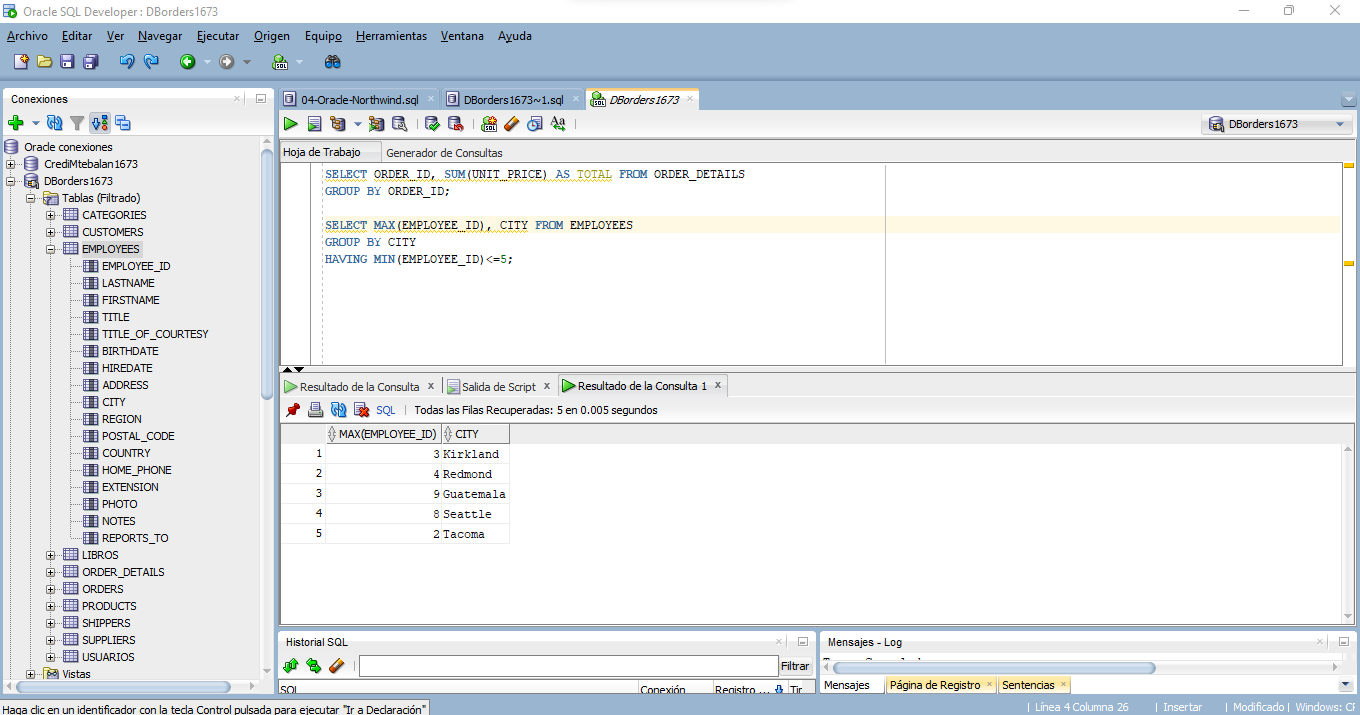
group by editorial

having min(precio)<100 and

min(precio)>30

order by editorial;

Entonces, usamos la clausula "having" para restringir las filas que devuelve una salida "group by". Va siempre después de la cláusula "group by" y antes de la cláusula "order by" si la hubiere.



# 33 - Registros duplicados (Distinct)

Con la cláusula "distinct" se especifica que los registros con ciertos datos duplicados sean obviadas en el resultado. Por ejemplo, queremos conocer todos los autores de los cuales tenemos libros, si utilizamos esta sentencia:

select autor from libros;

Aparecen repetidos. Para obtener la lista de autores sin repetición usamos:

select distinct autor from libros;

También podemos tipear:

select autor from libros

group by autor;

Note que en los tres casos anteriores aparece "null" como un valor para "autor"· Si sólo queremos la lista de autores conocidos, es decir, no queremos incluir "null" en la lista, podemos utilizar la sentencia siguiente:

select distinct autor from libros

where autor is not null;

Para contar los distintos autores, sin considerar el valor "null" usamos:

select count(distinct autor)

from libros;

Note que si contamos los autores sin "distinct", no incluirá los valores "null" pero si los repetidos:

select count(autor)

from libros;

Esta sentencia cuenta los registros que tienen autor.

Podemos combinarla con "where". Por ejemplo, queremos conocer los distintos autores de la editorial "Planeta":

select distinct autor from libros

where editorial='Planeta';

También puede utilizarse con "group by" para contar los diferentes autores por editorial:

select editorial, count(distinct autor)

from libros

group by editorial;

La cláusula "distinct" afecta a todos los campos presentados. Para mostrar los títulos y editoriales de los libros sin repetir títulos ni editoriales, usamos:

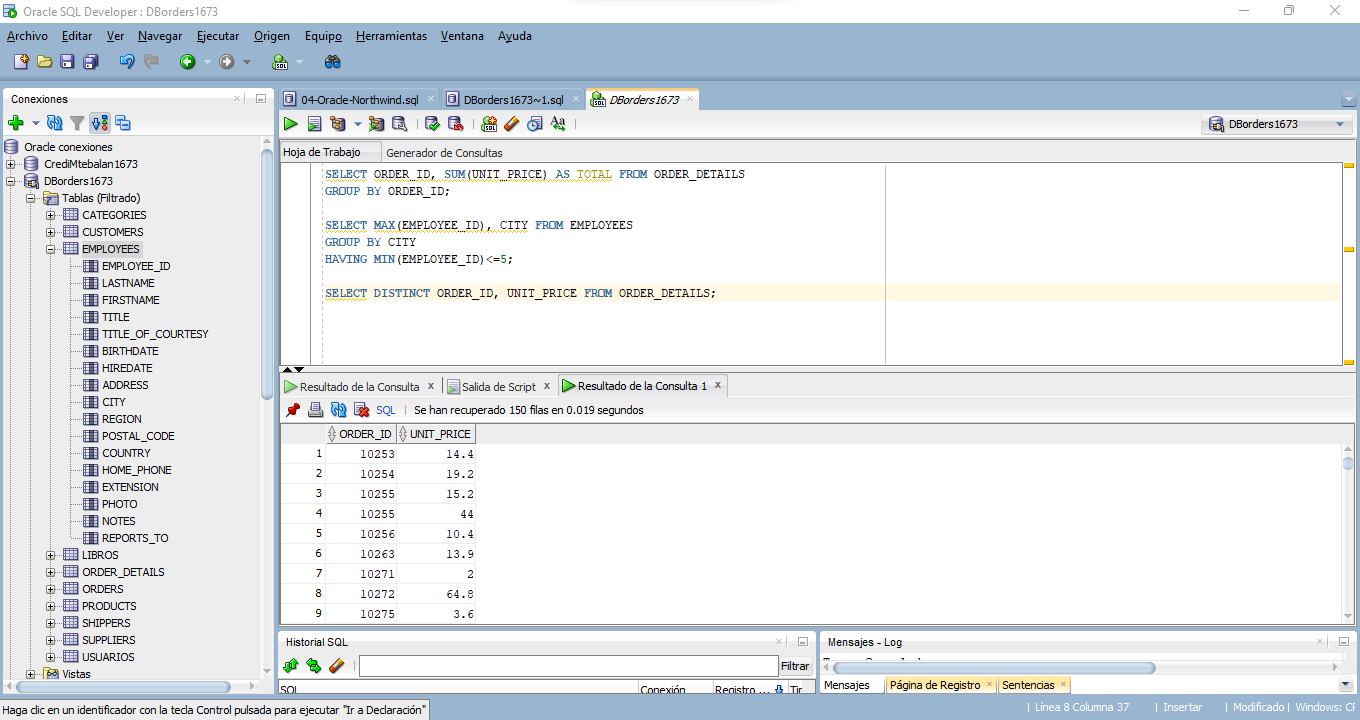
select distinct titulo,editorial

from libros

order by titulo;

Note que los registros no están duplicados, aparecen títulos iguales pero con editorial diferente, cada registro es diferente.

Entonces, "distinct" elimina registros duplicados.



# 34 - Clave primaria compuesta

Las claves primarias pueden ser simples, formadas por un solo campo o compuestas, más de un campo.

Recordemos que una clave primaria identifica un solo registro en una tabla.

Para un valor del campo clave existe solamente un registro. Los valores no se repiten ni pueden ser nulos.

Existe una playa de estacionamiento que almacena cada día los datos de los vehículos que ingresan en la tabla llamada "vehiculos" con los siguientes campos:

- patente char(6) not null,

- tipo char (1), 'a'= auto, 'm'=moto,

- horallegada date,

- horasalida date,

Necesitamos definir una clave primaria para una tabla con los datos descriptos arriba. No podemos usar solamente la patente porque un mismo auto puede ingresar más de una vez en el día a la playa; tampoco podemos usar la hora de entrada porque varios autos pueden ingresar a una misma hora. Tampoco sirven los otros campos.

Como ningún campo, por si sólo cumple con la condición para ser clave, es decir, debe identificar un solo registro, el valor no puede repetirse, debemos usar dos campos.

Definimos una clave compuesta cuando ningún campo por si solo cumple con la condición para ser clave.

En este ejemplo, un auto puede ingresar varias veces en un día a la playa, pero siempre será a distinta hora.

Usamos 2 campos como clave, la patente junto con la hora de llegada, así identificamos unívocamente cada registro.

Para establecer más de un campo como clave primaria usamos la siguiente sintaxis:

create table vehiculos(

patente char(6) not null,

tipo char(1),--'a'=auto, 'm'=moto

horallegada date,

horasalida date,

primary key(patente,horallegada)

);

Nombramos los campos que formarán parte de la clave separados por comas.

Al ingresar los registros, Oracle controla que los valores para los campos establecidos como clave primaria no estén repetidos en la tabla; si estuviesen repetidos, muestra un mensaje y la inserción no se realiza. Lo mismo sucede si realizamos una actualización.

Para ver la clave primaria de una tabla podemos realizar la siguiente consulta:

select uc.table\_name, column\_name, position from user\_cons\_columns ucc

join user\_constraints uc

on ucc.constraint\_name=uc.constraint\_name

where uc.constraint\_type='P' and

uc.table\_name='VEHICULOS';

Entonces, si un solo campo no identifica unívocamente un registro podemos definir una clave primaria compuesta, es decir formada por más de un campo.

