"El producto más valioso que conozco es la información"

08

TALLER DE BASE DE DATOS - INF 272 SEMESTRE 2/2022

PL/SQL Procedural Language Structured Query Language

¿Qué es PL/SQL?

- Es una extensión procedural para el lenguaje SQL de Oracle
- SQL no es un lenguaje procedural (que respeta las normas de un lenguaje estructurado o procedimental)
- PL/SQL permite suplir estas carencias
- Permite una mayor potencia a la hora de acceder a las bases de datos Oracle
- Con PL/SQL, podremos definir bloques, funciones, declarar variables, constantes, hacer condicionantes, loops, etc.

Bloques PL/SQL

- El objeto básico es el bloque
- Entonces PL/SQL es un bloque donde se van a construir y combinar comandos SQL con comandos PL/SQL
- Existen diferentes tipos de componentes en PL/SQL
  - Bloques: Bloques anónimos, Bloques con Nombre
  - Subprogramas: Procedimientos, Funciones y paquetes
  - Disparadores: Triggers

Bloques PL/SQL

- Presentan una estructura compuesta de tres partes bien diferenciadas
  - Sección Declarativa: donde se declaran todas las variables y constantes que se van a emplear en la ejecución del bloque. (Es opcional)
  - Sección de Ejecución: instrucciones a ejecutar en el bloque PL/SQL, pueden ser de tipo DML, DDL, como instrucciones procedimentales. (Es obligatoria)
  - Sección de Excepciones: se definen los manejadores de errores que soportará el bloque. (Es opcional solo se ejecuta si se presenta un error)

Bloques PL/SQL

```
DECLARE
   Declaración de Variables
BEGIN
   Sentencias SQL y PL/SQL
EXCEPTION
   Manejadores de Excepciones
END;
```

```
SET SERVEROUTPUT ON;
DECLARE
    A VARCHAR(10) := '';
BEGIN
   SELECT TO_CHAR(SYSDATE) INTO A FROM DUAL;
   DBMS_OUTPUT.PUT_LINE('LA FECHA ACTUAL ES : ' || A);
EXCEPTION
  WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('HOLA');
END;
```

Un bloque anónimo no tiene asignado un nombre.
Para que la salida pueda verse al ejecutarse debe activarse la variable >>>> SET SERVEROUTPUT ON;
Para mostrar el contenido de una expresión se usa: DBMS_OUTPUT.PUT_LINE (CADENA DE CARACTERES);

Bloques PL/SQL

```
SET SERVEROUTPUT ON;
DECLARE
    A VARCHAR(10) := '';
BEGIN
   SELECT TO_CHAR(SYSDATE) INTO A FROM DUAL;
   DBMS_OUTPUT.PUT_LINE('LA FECHA ACTUAL ES : ' || A);
EXCEPTION
  WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('HOLA');
END;
```

VARIABLES PL/SQL

- Empleadas para almacenar valores durante el desarrollo de un programa
- Pueden ser de distintos tipos
- Pueden ser utilizadas en comandos SQL
- Debe comenzar con una letra
- Pude contener números y letras
- Puede tener algunos caracteres especiales

VARIABLES PL/SQL

- No más de 30 caracteres en el nombre empleado
- No se pude usar palabras reservadas
- Se declaran en la sección DECLARE
- Se pueden pasar como argumentos a otros bloques PL/SQL
- También pueden almacenar resultados de otros bloques PL/SQL
- Cualquier variable que se declare y no se inicialice tiene el valor NULL

```
DECLARE
   NOMBRE_COMPLETO    VARCHAR2(40):='SIN DATOS';
   AGUINALDO          number (10,2):=0;
   FECHA_INGRESO      DATE;
```

CONSTANTES Y NOT NULL

- Son variables que no pueden ser modificadas a lo largo de la ejecución del script.
- Además cuando se declaran constantes estas deben ser inicializadas sino dará error.
- No se puede modificar por que es solo de lectura.

```
SET SERVEROUTPUT ON;
DECLARE
      PRUEBA CONSTANT NUMBER(5) := 2022;
      DIAS     NUMBER NOT NULL := 87;
BEGIN
      DBMS_OUTPUT.PUT_LINE(PRUEBA);
      DBMS_OUTPUT.PUT_LINE(DIAS);
      DIAS := DIAS + 13;
      DBMS_OUTPUT.PUT_LINE(DIAS);
END;
```

BOOLEAN

- Variables que pueden almacenar valores de falso o verdadero
- No existe este tipo de variable en la base de datos.
- Puede tener valores TRUE, FALSE y NULL

```
SET SERVEROUTPUT ON;
DECLARE
  ASISTE BOOLEAN;
  CADENA VARCHAR2(10);
BEGIN
    ASISTE:=TRUE;
    CADENA := CASE WHEN ASISTE THEN 'VERDADERO' ELSE 'FALSO' END;
    DBMS_OUTPUT.PUT_LINE(CADENA);
END;
```

%TYPE

- %TYPE Se usa para declarar una variable que tendrá el mismo tipo que una columna de una tabla o que una variable definida.

```
DECLARE
  V_NOMBRE employees.first_name%TYPE;
  V_APELLIDO employees.last_name%TYPE;
BEGIN
  SELECT FIRST_NAME INTO V_NOMBRE FROM EMPLOYEES WHERE ROWNUM=1;
  V_APELLIDO:='MORALES';
  DBMS_OUTPUT.PUT_LINE('NOMBRE EMPLEADO:'||V_NOMBRE||' APELLIDOS:'||V_APELLIDO);
END;
```

%ROWTYPE

- %ROWTYPE Se usa para declarar un conjunto de variables similares a los tipos de una tabla.

```
SET SERVEROUTPUT ON
DECLARE
    EMPLEADOS    EMPLOYEES%ROWTYPE;
BEGIN
    SELECT * INTO EMPLEADOS
    FROM EMPLOYEES
    WHERE EMPLOYEE_ID=100;
    DBMS_OUTPUT.PUT_LINE(EMPLEADOS.FIRST_NAME||' --->'||EMPLEADOS.SALARY);
END;
```

OPERADORES

- PL/SQL maneja operadores muy similares a los que se manejan en otros lenguajes.
  - + suma
  - - resta
  - * multiplicación
  - / división
  - ** exponente
  - || concatenar

COMENTARIOS

```
-- PERMITE COMENTAR LA LINEA DONDE ESTAMOS

/*   COMENTARIO   */
```

BLOQUES ANIDADOS

- Permite agrupar u ordenar pedazos de bloques dentro de otro bloque mas grande.

```
SET SERVEROUTPUT ON
DECLARE
        XYZ NUMBER := 30;   -- VARIABLE GLOBAL
BEGIN
   DBMS_OUTPUT.PUT_LINE ('NOS ENCONTRAMOS EN EL BLOQUE 1');
   DBMS_OUTPUT.PUT_LINE (XYZ);
   DECLARE
        NUM   NUMBER  :=88;  --VARIABLES LOCALES
        XYZ NUMBER :=100;
   BEGIN
       DBMS_OUTPUT.PUT_LINE ('EN EL BLOQUE 2');
DBMS_OUTPUT.PUT_LINE (NUM);
        DBMS_OUTPUT.PUT_LINE (XYZ);
   END;
END;
```

FUNCIONES

- Una de las características es que las funciones que empleábamos en SQL, pueden ser usadas en PL/SQL.
- Funciones como: INSTR, LENGTH, CONCAT, LOWER, UPPER, ETC. Todas ellas que funcionan en Oracle pueden ser empleadas en código PL/SQL sin ningún problema.
- A pesar de que las funciones que se usan son similares, las que se usan con SQL se ejecutan en el motor de Oracle. Mientras que las que se emplean en PL/SQl se ejecutan en el motor de PL/SQL.

OPERADORES LOGICOS Y RELACIONALES

= IGUAL
<> DISTINTO
< Menor que
> Mayor que
>= Mayor o Igual que
<= Menor o igual que

AND operador y lógico
OR operador o lógico
NOT operador de negación

ESTRUCTURA DE CONTROL IF

```
IF  condición
    THEN
        sentencias;
    ELSE
        sentencias;
END IF;
```

```
IF  condición
    THEN
        sentencias;
    ELSEIF condición
        THEN
            sentencias;
    ELSE
        sentencias;
END IF;
```

ESTRUCTURA DE CONTROL CASE

```
CASE  variable
    WHEN valor1 THEN sentencia1;
    WHEN valor2 THEN sentencia2;
    WHEN valor3 THEN sentencia3;
    WHEN valor4 THEN sentencia4;
    WHEN valor5 THEN sentencia5;
    ELSE sentencia6;
END CASE;
```

ESTRUCTURA DE CONTROL SEARCHED CASE

```
CASE
    WHEN condicion1 THEN sentencia1;
    WHEN condicion2 THEN sentencia2;
    WHEN condicion3 THEN sentencia3;
    WHEN condicion4 THEN sentencia4;
    WHEN condicion5 THEN sentencia5;
    ELSE sentencia6;
END CASE;
```

ESTRUCTURA DE CONTROL BUCLE LOOP

```
LOOP
    sentencias;
END LOOP;
```

Para romper y poder salir del ciclo se usa EXIT

también se puede emplear EXIT WHEN coondicion

ESTRUCTURA DE CONTROL BUCLE FOR

```
FOR variable IN Vi..Vf LOOP
    sentencias;
END LOOP;

FOR i IN 5..15 LOOP            i es una variable PLS INTEGER
  dbms_output.put_line(i);
END LOOP;

FOR i IN REVERSE 5..15 LOOP
  dbms_output.put_line(i);
END LOOP;
```

ESTRUCTURA DE CONTROL BUCLE WHILE

```
WHILE CONDICION LOOP
    sentencias;
END LOOP;

WHILE NOT   CONDICION LOOP
    sentencias;
END LOOP;
```

COMANDO GOTO

Permite romper la secuencia del bloque y llevarnos a otra parte del bloque de sentencias misma que esta identificada con una etiqueta.

```
    sentencia 1;
    sentencia 2;
    sentencia 3;
    GO TO etiqueta1;
    sentencia ....;

  <<etiqueta1>>
    sentencia 10;
```

INSERTS EN PL/SQL

Es idéntico al SQL tradicional

```
DECLARE
  COL1 prueba.campo1%TYPE;
BEGIN
  COL1:=666;
  INSERT INTO prueba (Campo1,Campo2)
  VALUES (COL1,‘zzzzzzzz');
  COMMIT;
END;
```

UPDATES EN PL/SQL

Es idéntico al SQL tradicional

```
DECLARE
  P PRUEBA.Campo1%TYPE;
BEGIN
  P:=99;
  UPDATE PRUEBA SET Campo2=‘kkkkkk' WHERE Campo1=P;
  COMMIT;
END;
```

DELETES EN PL/SQL

Es idéntico al SQL tradicional

```
DECLARE
   P TEST.C1%TYPE;
BEGIN
   P:=20;
   DELETE FROM TEST WHERE Campo1=P;
   COMMIT;
END;
```

Excepciones

Las excepciones nos ayudan a detectar y tratar errores en tiempo de ejecución.
Sirve para definir que se debe hacer frente a errores en sentencias definidas por el usuario, cuando se produce un error se levanta una excepción y pasa el control a la sección de excepción correspondiente del bloque. Hay excepciones de Oracle-PL/SQL o definidas por el usuario.

```
BEGIN
    ..........
    ..........
EXCEPTION
    WHEN <NOMBRE EXCEPCIÓN> THEN INSTRUCCIONES
    .........
END;
```

Excepciones Predefinidas

Son aquellas que se disparan automáticamente cuando se produce un error.

too_many_rows Se produce cuando select devuelve mas de 1 fila

no_data_found Se produce cuando select no devuelve nada

zero_divide Cuando hay una división entre 0

dupval_on_index Cuando se intenta almacenar un valor que crearía duplicados en la clave primaria, o en una columna con restricción UNIQUE

Excepciones Predefinidas

cursor_already_open Cuando se intenta abrir un cursor que ya esta abierto.

invalid_cursor Cuando se efectuo una operación invalida sobre un cursor, por ejemplo cuando se intenta cerrar un cursor y este no esta abierto.

Hay muchas mas, se debe revisar los manuales de referencia.

Excepciones No predefinidas

Existen otros errores internos de Oracle que no tienen asignada una excepción sino un código de error y un mensaje a los que se accede mediante funciones SQLCODE y SQLERRM. Cuando se presenta el control se transfiere directamente a la zona EXCEPTION.

Excepciones No predefinidas

```
SET SERVEROUTPUT ON
DECLARE
  MI_ERROR EXCEPTION;
  PRAGMA EXCEPTION_INIT(MI_ERROR,-937);
  V_ID NUMBER;
  V_SALARIO NUMBER;
BEGIN
   SELECT EMPLOYEE_ID,SUM(SALARY) INTO V_ID,V_SALARIO FROM EMPLOYEES;
   DBMS_OUTPUT.PUT_LINE(V_ID);
EXCEPTION
  WHEN MI_ERROR THEN
     DBMS_OUTPUT.PUT_LINE('FUNCION DE GRUPO INCORRECTA');
  WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('ERROR NO CONTROLADO');
END;
```

PRAGMA ES UNA ESPECIE DE ORDEN AL COMPILADOR PL/SQL

SQLCODE - SQLERRM

SQLCODE – Devuelve el número de error asociado con la excepción lanzada.

SQLERRM – Devuelve el mensaje de error asociado con el error que activo la excepción.

SQLCODE - SQLERRM

```
SET SERVEROUTPUT ON
DECLARE
  EMPL EMPLOYEES%ROWTYPE;
  CODE NUMBER;
  MESSAGE VARCHAR2(100);
BEGIN
  SELECT * INTO EMPL FROM EMPLOYEES;
  DBMS_OUTPUT.PUT_LINE(EMPL.SALARY);
EXCEPTION
  WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLCODE);
      DBMS_OUTPUT.PUT_LINE(SQLERRM);
      CODIGO:=SQLCODE;
      MENSAJE:=SQLERRM;
      INSERT INTO ERRORES VALUES (CODIGO,MENSAJE);
      COMMIT;
END;
```

Excepciones de Usuario

Son excepciones creadas por el desarrollador, quien crea una nueva estructura de errores y se lanzan a través de la sentencia RAISE.

Excepciones de Usuario

```
DECLARE
  reg_max EXCEPTION;
  regn NUMBER;
  regt varchar2(200);
BEGIN
  regn:=101;
  regt:='ASIA';
  IF regn > 100 THEN
      RAISE reg_max;
   ELSE
     insert into regions values (regn,regt);
     commit;
    END IF;
EXCEPTION
 WHEN reg_max THEN
  DBMS_OUTPUT.PUT_LINE('La region no puede ser mayor de 100.');
 WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE('Error indefinido');
END;
```

RAISE_APLICATION_ERROR

Sirve para levantar errores y definir mensajes de error, su formato es:
RAISE_APLICATION_ERROR(NUMERO_ERROR,MENSAJE_ERROR)

El número de error está comprendido entre -20000 y -20999, el mensaje una cadena de caracteres de hasta 512 bytes.
Crea una excepción que solo puede ser tratada en WHEN OTHERS

RAISE_APLICATION_ERROR

```
SET SERVEROUTPUT ON
DECLARE
CONTROL_REGIONES EXCEPTION;
CODIGO NUMBER:=201;
BEGIN
 IF codigo > 200 THEN
     raise control_regiones;
 ELSE
    INSERT INTO REGIONS VALUES (CODIGO,'PRUEBA');
 END IF;
EXCEPTION
WHEN control_regiones THEN
  RAISE_APPLICATION_ERROR(-20001,'El codigo debe ser inferior a 200');
WHEN OTHERS THEN
  dbms_output.put_line(SQLcode);
  dbms_output.put_line(SQLERRM);
END;
```

Cursores

Los cursores se utilizan en SQL para manejar las sentencias SELECT. Un cursor esta formado por un conjunto de registros devueltos por una instrucción SQL del tipo SELECT.

Desde un punto de visto interno a la base de datos Oracle, los cursores son segmentos de memoria utilizados para realizar operaciones con los registros devueltos tras ejecutar una sentencia SELECT.

Un cursor tiene que ser definido previamente como cualquier otra variable y debe asignarle un nombre.

Los cursores admiten el uso de parámetros.

Cursores

Para trabajar con un cursor hay que realizar los siguientes pasos:

Declarar el cursor
CURSOR nombre_cursor IS instrucción_SELECT
CURSOR nombre_cursor(param1 tipo1, ..., paramN tipoN) IS instrucción_SELECT
Abrir el cursor en el servidor
OPEN nombre_cursor;OPEN nombre_cursor(valor1, valor2, ..., valorN);
Recuperar cada una de sus filas (bucle)
FETCH nombre_cursor INTO variables;
Cerrar el cursor
CLOSE **nombre_cursor;**

Declarar el Cursor

Al igual que cualquier otra variable, el cursor se declara en la sección DECLARE.

Se define el nombre que tendrá el cursor y qué consulta SELECT ejecutará. No es más que una declaración.

Una vez que el cursor está declarado ya podrá ser utilizado dentro del bloque de código.

Antes de utilizar un cursor se debe abrir. En ese momento se ejecuta la sentencia SELECT asociada y se almacena el resultado en el área de contexto (estructura interna de memoria que maneja el cursor). Un puntero señala a la primera fila

Abrir el Cursor

Al abrir el cursor se ejecuta la sentencia SELECT asociada y cuyo resultado se guarda en el servidor en un área de memoria interna (tablas temporales) de las cuales se va retornando cada una de las filas según se va pidiendo desde el cliente.

Al abrir un cursor, un puntero señalará al primer registro.

Una vez que el cursor está abierto, se podrá empezar a pedir los resultados al servidor.

Recuperar Filas

Al recuperar un registro, la información recuperada se guarda en una o varias variables.
Si sólo se hace referencia a una variable, ésta se puede declarar con %ROWTYPE.
Si se utiliza una lista de variables, cada variable debe coincidir en tipo y orden con cada una de las columnas de la sentencia SELECT.

Así lo acción más típica es recuperar filas mientras queden alguna por recuperar en el servidor.

Recuperar Filas

```
OPEN nombre_cursor;
LOOP
 FETCH nombre_cursor INTO variables;
 EXIT WHEN nombre_cursor%NOTFOUND;
 --procesar cada una de las filas
END LOOP;
```

```
OPEN nombre_cursor;
FETCH nombre_cursor INTO lista_variables;
WHILE nombre_cursor%FOUND
LOOP
 /* Procesamiento de los registros recuperados */
 FETCH nombre_cursor INTO lista_variables;
END LOOP;
CLOSE nombre_cursor;
```

```
FOR variable IN nombre_cursor LOOP
 /* Procesamiento de los registros recuperados */
END LOOP;
```

Cerrar Cursor

Una vez que se han recuperado todas las filas del cursor, hay que cerrarlo para que se liberen de la memoria del servidor los objetos temporales creados. Si no cerrásemos el cursor, la tabla temporal quedaría en el servidor almacenada con el nombre dado al cursor y si la siguiente vez ejecutásemos ese bloque de código, nos daría la excepción CURSOR_ALREADY_OPEN (cursor ya abierto) cuando intentásemos abrir el cursor.

Atributos de Cursores

Atributo	Tipo	Descripción
%ISOPEN	Booleano	TRUE si el cursor está abierto.
%NOTFOUND	Booleano	TRUE si la recuperación más reciente no devuelve ninguna fila.
%FOUND	Booleano	TRUE si la recuperación más reciente devuelve una fila.
%ROWCOUNT	Número	Proporciona el número total de filas devueltas hasta ese momento.

Procedimientos Almacenados

Conjunto de instrucciones a las que se les da un nombre, se almacena en la base de datos activa.
Permiten agrupar y organizar tareas repetitivas.
Ventajas:
Comparten la lógica de la aplicación con las otras aplicaciones.
Realiza operaciones que los usuarios necesitan sin que tengan acceso a las tablas.

Procedimientos Almacenados

En vez de enviar muchas instrucciones, los usuarios realizan operaciones enviando una única instrucción.
Un procedimiento almacenados puede hacer referencia a objetos que no existen al momento de crearlo. Los objetos deben existir cuando se ejecute el procedimiento almacenado.

Procedimientos Almacenados

Desventajas:

Las instrucciones que podemos utilizar dentro de un procedimiento almacenado no están preparadas para implementar lógicas de negocios muy complejas.

Son difíciles de depurar.

Procedimientos Almacenados

Pueden hacer referencia a tablas, vistas, a funciones definidas por el usuario, a otros procedimientos almacenados.
Pueden incluir cualquier cantidad y tipo de instrucciones DML (de manipulación de datos, como insert, update, delete), no instrucciones DDL (de definición de datos, como create..., drop... alter...).

```
create or replace procedure NOMBREPROCEDIMIENTO
as
begin
  INSTRUCCIONES
end;
/
```

Importante la barra diagonal '/' luego del end, sin esto se genera un error.

Procedimientos Almacenados

Los procedimientos almacenados se eliminan con "drop procedure".

```
drop procedure NOMBREPROCEDIMIENTO;
```

Si el procedimiento que queremos eliminar no existe, aparece un mensaje de error indicando tal situación.

Podemos eliminar una tabla referenciada en un procedimiento almacenado, Oracle lo permite, pero luego, al ejecutar el procedimiento, aparecerá un mensaje de error porque la tabla referenciada no existe.

Procedimientos Almacenados

Pueden recibir y devolver información; para ello se emplean parámetros.
Veamos los primeros. Los parámetros de entrada posibilitan pasar información a un procedimiento. Para que un procedimiento almacenado admita parámetros de entrada se deben declarar al crearlo.

```
create or replace procedure NOM_PROC (PARAMETRO in TIPODEDATO)
 as
 begin
  INSTRUCCIONES;
 end;
 /
```

Pueden declararse varios parámetros por procedimiento, se separan por comas.

PROCEDIMIENTOS

Para crear se debe emplear:

```
CREATE OR REPLACE PROCEDURE nombre IS
    sección de declaración de variables
BEGIN
    ..............
EXCEPTION
    ..................
END nombre;
```

PARA EJECUTAR

```
BEGIN
   nombre;
END;
```

EXECUTE nombre;

PARAMETROS

Desde un bloque PL/SQL se invoca un procedimiento, se pueden pasar parámetros:

IN DE ENTRADA, SI NO SE ESPECIFICA ASUME ESTE
OUT DE SALIDA, PARA DEVOLVER EL RESULTADO
IN/OUT DE ENTRADA Y SALIDA

```
CREATE OR REPLACE PROCEDURE CALC_IMPT (EMPL IN
EMPLOYEES.EMPLOYEE_ID%TYPE, T1 NUMBER)
IS
 IMPTO NUMBER:=0;
 SAL   NUMBER:=0;
BEGIN
  SELECT SALARY INTO SAL FROM EMPLOYEES WHERE EMPLOYEE_ID=EMPL;
  IMPTO:=SAL*T1/100;
  DBMS_OUTPUT.PUT_line('SALARIO:'||SAL);
  DBMS_OUTPUT.PUT_line('IMPUESTO:'||IMPTO);
END;
```

FUNCIONES

Una función es un bloque de código que implementa acciones y que es referenciado por un nombre. Puede recibir argumentos. La diferencia con los procedimientos es que retornan un valor siempre.

Para asignar un valor a una variable, dentro de una función DEBE usarse ":=" (dos puntos e igual).

Si no se le definen parámetros a una función, no deben colocarse los paréntesis.

Las funciones pl/sql se pueden usar en sentencias SQL

FUNCIONES

Para crear se debe emplear:

```
CREATE OR REPLACE FUNCTION nombre-funcion
  (nombre-parámetro {IN|OUT|IN OUT} TIPO DE DATO,...)
  RETURN tipo_de_dato {IS|AS}
    sección de declaración de variables,
    constantes,cursores.
BEGIN
    ..............
EXCEPTION
    ..................
END;
```

FUNCIONES

Para crear se debe emplear:

```
CREATE OR REPLACE FUNCTION nombre-funcion
  (nombre-parámetro {IN|OUT|IN OUT} TIPO DE DATO,...)
  RETURN tipo_de_dato {IS|AS}
    sección de declaración de variables,
    constantes,cursores.
BEGIN
    ..............
EXCEPTION
    ..................
 END;
```

COMO VER EL CODIGO FUENTE

Para ello emplearemos algunas vistas que nos permiten ver esto:

```
Select * from user_objects where object_type IN ('PROCEDURE','FUNCTION';

SELECT OBJECT_TYPE,COUNT(*)  FROM USER_OBJECTS GROUP BY OBJECT_TYPE;

SELECT * FROM USER_SOURCE WHERE NAME='NOMBREPROC' AND TYPE='PROCEDURE';
```

Indices

Objetivo es acelerar la recuperación de información y que es útil cuando la tabla contiene miles de registros, cuando se realizan operaciones de ordenamiento y agrupamiento, etc.

Los campos por los que sería útil crear un índice, son aquellos por los cuales se realizan búsquedas con frecuencia: claves primarias, claves foráneas o campos que combinan tablas. No recomendable crear índices sobre campos que no se usan con frecuencia en consultas o en tablas muy pequeñas.

```
create index NOMBREINDICE
on NOMBRETABLA(CAMPOS);
```

Indices

Si se intenta crear un índice único para un campo que tiene valores duplicados, Oracle no lo permite.

Los campos de tipo "long" y "long raw" no pueden indexarse.

Una tabla puede indexarse por un campo (o varios).

Cuando creamos una restricción "primary key" o "unique" sobre una tabla, Oracle automáticamente crea un índice sobre el campo (o los campos) de la restricción y le da el mismo nombre que la restricción. En caso que la tabla ya tenga un índice, Oracle lo usa, no crea otro.

all_indexes all_constraints VISTAS de CONSULTA

Eliminar Indices

Los índices se eliminan con "drop index"; la siguiente es la sintaxis básica:

```
drop index NOMBREINDICE;
```

Los índices usados por las restricciones "primary key" y "unique" no pueden eliminarse con "drop index", se eliminan automáticamente cuando quitamos la restricción.

Si eliminamos una tabla, todos los índices asociados a ella se eliminan.