BASE DE DATOS II
UNIDAD I

INTRODUCCIÓN A ORACLE, SQLDEVELOPER, SQL DATA MODELER

ORACLE

- Oracle permite acceder y manipular información de la base de datos definiendo objetos procedurales (subprogramas) que se almacenan en la base de datos. Estos objetos procedurales son unidades de programa PL/SQL: Funciones y Procedimientos almacenados.

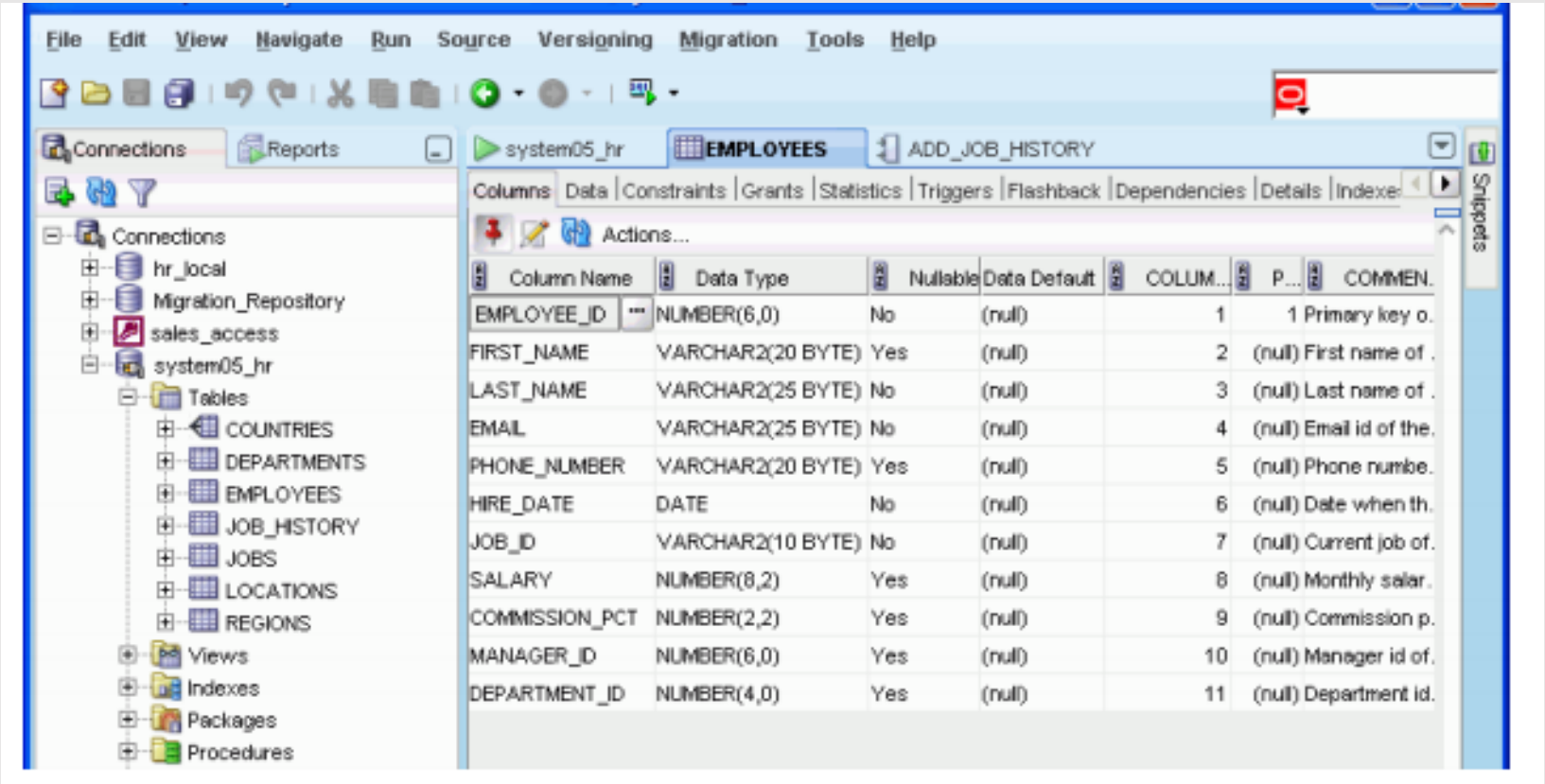
- Los procedimientos o funciones son bloques PL/SQL con nombre, que pueden recibir parámetros y pueden ser invocados desde distintos entornos: SQL*PLUS, Oracle*Forms, desde otros procedimientos y funciones y desde otras herramientas Oracle y aplicaciones.

SQL DEVELOPER

- Oracle SQL Developer es un entorno de desarrollo integrado y gratuito que simplifica el desarrollo y la administración de la base de datos Oracle tanto en implementaciones tradicionales como en la nube. SQL Developer ofrece un desarrollo completo de extremo a extremo de sus aplicaciones PL / SQL, una hoja de trabajo para ejecutar consultas y scripts, una consola DBA para administrar la base de datos, una interfaz de informes, una solución completa de modelado de datos y una plataforma de migración para mover su Bases de datos de terceros a Oracle.

INTERFAZ DE SQL DEVELOPER

- La ventana de SQL Developer generalmente usa el lado izquierdo para la navegación para encontrar y seleccionar objetos y el lado derecho para mostrar información sobre los objetos seleccionados.

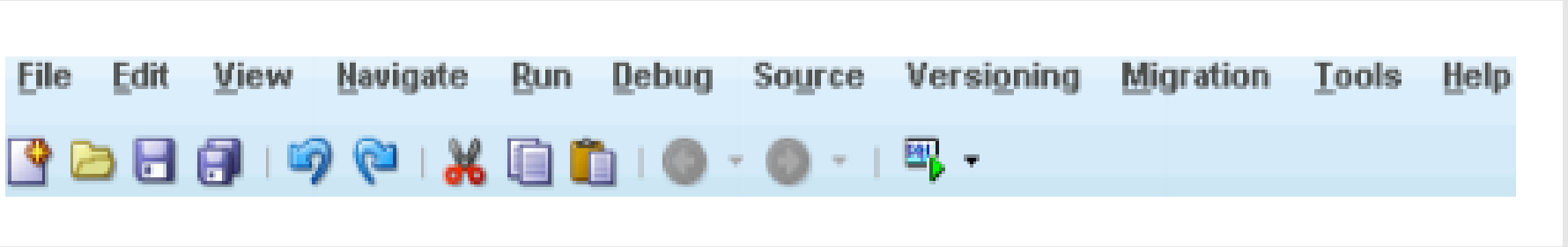


INTERFAZ DE SQL DEVELOPER

Puede usar teclas de método abreviado para acceder a menús y elementos de menú: por ejemplo, Alt + F para
Menú Archivo y Alt + E para el menú Edición; o Alt + H, luego Alt + S para Ayuda, luego Texto Completo
Buscar. También puede visualizar el menú Archivo presionando la tecla F10.
Los íconos debajo de los menús realizan varias acciones, incluyendo las siguientes:

- New: crea un nuevo objeto de base de datos nuevo.
- Open: abre un archivo (consulte la Sección 4.89, "Abrir archivo").
- Save: Guarda cualquier cambio en el objeto seleccionado actualmente.

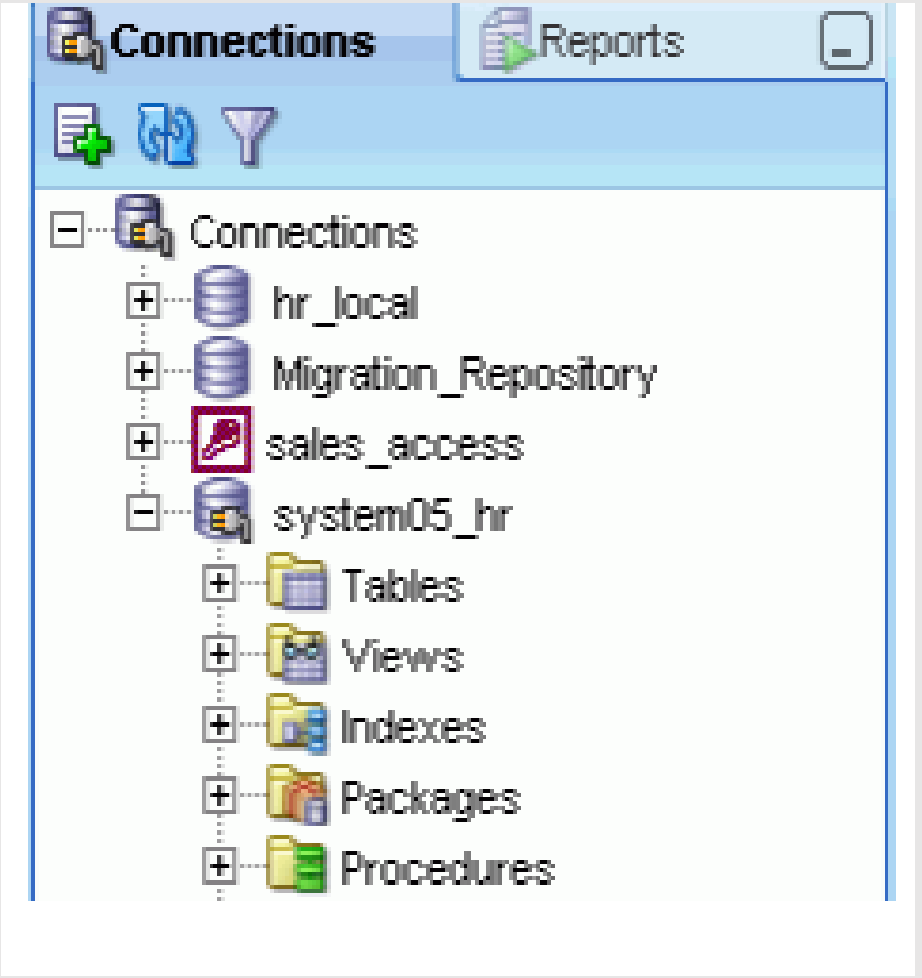
INTERFAZ DE SQL DEVELOPER



- ■ Savel All: guarda cualquier cambio en todos los objetos abiertos.
- ■ Back: se mueve al panel que visitó más recientemente.
- ■ Forward: se mueve al panel después del actual en la lista de paneles visitados.
- ■ Open SQL Worksheet: abre la hoja de cálculo SQL

INTERFAZ DE SQL DEVELOPER

- El navegador de Connections enumera las conexiones de base de datos que se han creado. A crear una nueva conexión de base de datos, importar un archivo XML con definiciones de conexión, o exportar o editar conexiones actuales, hacer clic derecho en el nodo Conexiones y seleccionar el elemento de menú apropiado.



INTERFAZ DE SQL DEVELOPER

Creación de tablas

Para crear una tabla, puede hacer una de las siguientes cosas:

■ Cree la tabla rápidamente agregando columnas y especificando las características usadas frecuentemente.

Para hacer esto, no marque la casilla Avanzado en el cuadro de diálogo Crear tabla.

Para obtener ayuda con las opciones para crear una tabla usando este enfoque rápido.

■ Cree la tabla agregando columnas y seleccionando entre un conjunto de características más grande.

■ Cree la tabla automáticamente desde una hoja de cálculo de Microsoft Excel. Para hacer esto, haga clic con el botón derecho en Tablas debajo de una conexión en el navegador Conexiones, y seleccione Datos de importación. Cuando se le pida el archivo, seleccione un archivo de tipo .xls o .csv.

INTERFAZ DE SQL DEVELOPER

Creación de tablas

Puede realizar las siguientes operaciones en una tabla haciendo clic con el botón derecho en el nombre de la tabla el navegador de Conexiones y seleccionando un elemento del menú:

- Edit: muestra el cuadro de diálogo Crear / editar tabla (con opciones avanzadas).
- Table: las acciones de la tabla incluyen **Renombrar**, **Copiar** (crear una copia con un nombre diferente), **Eliminar** (eliminar la tabla), **Truncar** (eliminar datos existentes sin afectar la tabla definición), **Bloquear** (establece el modo de bloqueo de la tabla: compartir en fila, exclusivo, etc.), **Comentario** (comentario descriptivo que explica el uso o el propósito de la tabla),
- Constraint: incluye opciones para agregar, descartar, habilitar y deshabilitar restricciones
- Index: las opciones incluyen **Crear** (crear un índice en columnas especificadas), **Crear texto** (crear un índice de texto de Oracle en una columna), **crear texto** (crear una función basada índice en una columna) y Drop.

INTERFAZ DE SQL DEVELOPER

Creación de tablas

■ Privileges: si está conectado como usuario de base de datos con suficientes privilegios, puede otorgar o revocar privilegios en la tabla a otros usuarios.

■ Statistics: las opciones incluyen Reunir estadísticas (calcular la tabla y la columna exactas estadísticas y almacenarlos en el diccionario de datos) y validar la estructura (verifica el integridad de cada bloque de datos y fila, y para una tabla organizada por índice también genera el recuento de compresión de prefijo óptimo para el índice de clave principal). El optimizador de Oracle Database usa las estadísticas para elegir la ejecución de plan para declaraciones SQL que acceden a objetos analizados.

■ Trigger: las opciones incluyen Crear, Crear PK desde secuencia (crear un antes-insertar desencadenar para llenar la clave principal usando valores de una secuencia especificada), Habilitar o Deshabilitar todo, Habilitar o Deshabilitar Único y Soltar (eliminar el activador).

INTERFAZ DE SQL DEVELOPER

Creación de tablas

■ Almacenamiento: las opciones incluyen la tabla de contracción (espacio de contracción en una tabla, para segmentos en espacios de tablas con gestión automática de segmentos) y Move Table (a otro tablespace). Las opciones de la Tabla de contracción incluyen Compacta (solo desfragmenta el espacio de segmento y compacta las filas de la tabla para su posterior publicación, pero no lo hace reajuste la marca de agua alta y no libera el espacio inmediatamente) y Cascade (realiza las mismas operaciones en todos los objetos dependientes de la tabla, incluyendo índices secundarios en tablas organizadas por índice).

■ Importar datos: le permite importar datos de una hoja de cálculo de Microsoft Excel (.xls o archivo .csv).

■ Exportar datos: le permite exportar algunos o todos los datos de la tabla a un archivo o al portapapeles del sistema, en cualquiera de los siguientes formatos: XML (etiquetas XML y datos), CSV (valores separados por comas, incluida una fila de encabezado para los identificadores de columna), SQL Insertar (instrucciones INSERT) o SQL Loader (archivo de control del cargador SQL *). Después de ti seleccione un formato, se muestra el cuadro de diálogo Exportar datos de tabla.

DECLARACIÓN DE VARIABLES

- CHAR, VARCHAR, NUMBER, BINARY_INTEGER:

Data Type	Description
CHAR [(*maximum_length*)]	Base type for fixed-length character data up to 32,767 bytes. If you do not specify a maximum length, the default length is set to 1.
VARCHAR2 (*maximum_length*)	Base type for variable-length character data up to 32,767 bytes. There is no default size for VARCHAR2 variables and constants.
NUMBER [(*precision*, *scale*)]	Number having precision *p* and scale *s*. The precision *p* can range from 1 through 38. The scale *s* can range from –84 through 127.
BINARY_INTEGER	Base type for integers between –2,147,483,647 and 2,147,483,647

DECLARACIÓN DE VARIABLES

- PLSQL_INTEGER, BOOLEAN, BINARY_FLOAT, BINARY_DOUBLE

Data Type	Description
PLS_INTEGER	Base type for signed integers between –2,147,483,647 and 2,147,483,647. PLS_INTEGER values require less storage and are faster than NUMBER values. In Oracle Database 10g, the PLS_INTEGER and BINARY_INTEGER data types are identical. The arithmetic operations on PLS_INTEGER and BINARY_INTEGER values are faster than on NUMBER values.
BOOLEAN	Base type that stores one of the three possible values used for logical calculations: TRUE, FALSE, and NULL
BINARY_FLOAT	Represents floating-point number in IEEE 754 format. It requires 5 bytes to store the value.
BINARY_DOUBLE	Represents floating-point number in IEEE 754 format. It requires 9 bytes to store the value.

DECLARACIÓN DE VARIABLES

- DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE.

Data Type	Description
`DATE`	Base type for dates and times. `DATE` values include the time of day in seconds since midnight. The range for dates is between 4712 B.C. and A.D. 9999.
`TIMESTAMP`	The `TIMESTAMP` data type, which extends the `DATE` data type, stores the year, month, day, hour, minute, second, and fraction of second. The syntax is `TIMESTAMP [(precision)]`, where the optional parameter `precision` specifies the number of digits in the fractional part of the seconds field. To specify the precision, you must use an integer in the range 0–9. The default is 6.
`TIMESTAMP WITH TIME ZONE`	The `TIMESTAMP WITH TIME ZONE` data type, which extends the `TIMESTAMP` data type, includes a time-zone displacement. The time-zone displacement is the difference (in hours and minutes) between local time and Coordinated Universal Time (UTC), formerly known as Greenwich Mean Time. The syntax is `TIMESTAMP [(precision)] WITH TIME ZONE`, where the optional parameter `precision` specifies the number of digits in the fractional part of the seconds field. To specify the precision, you must use an integer in the range 0–9. The default is 6.

DECLARACIÓN DE VARIABLES

- TIMESTAMP WITH LOCAL TIME ZONE, INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND

Data Type	Description
TIMESTAMP WITH LOCAL TIME ZONE	The TIMESTAMP WITH LOCAL TIME ZONE data type, which extends the TIMESTAMP data type, includes a time-zone displacement. The time-zone displacement is the difference (in hours and minutes) between local time and Coordinated Universal Time (UTC), formerly known as Greenwich Mean Time. The syntax is TIMESTAMP [(precision)] WITH LOCAL TIME ZONE, where the optional parameter precision specifies the number of digits in the fractional part of the seconds field. You cannot use a symbolic constant or variable to specify the precision; you must use an integer literal in the range 0–9. The default is 6. This data type differs from TIMESTAMP WITH TIME ZONE in that when you insert a value into a database column, the value is normalized to the database time zone, and the time-zone displacement is not stored in the column. When you retrieve the value, the Oracle server returns the value in your local session time zone.
INTERVAL YEAR TO MONTH	You use the INTERVAL YEAR TO MONTH data type to store and manipulate intervals of years and months. The syntax is INTERVAL YEAR [(precision)] TO MONTH, where precision specifies the number of digits in the years field. You cannot use a symbolic constant or variable to specify the precision; you must use an integer literal in the range 0–4. The default is 2.
INTERVAL DAY TO SECOND	You use the INTERVAL DAY TO SECOND data type to store and manipulate intervals of days, hours, minutes, and seconds. The syntax is INTERVAL DAY [(precision1)] TO SECOND [(precision2)], where precision1 and precision2 specify the number of digits in the days field and seconds field, respectively. In both cases, you cannot use a symbolic constant or variable to specify the precision; you must use an integer literal in the range 0–9. The defaults are 2 and 6, respectively.

CREACIÓN ESQUEMAS-USUARIOS

- En cualquier modelo de datos es importante distinguir entre la descripción de la base de datos y la base de datos misma. La descripción se conoce como esquema de la base de datos (o metadatos). Este esquema se especifica durante el diseño y no es de esperar que se modifique muy a menudo.
- Un esquema de base de datos representa la configuración lógica de todo o parte de una base de datos relacional. Un esquema de base de datos indica qué tablas o relaciones componen la base de datos, así como los campos incluidos en cada tabla.
- En la mayoría de los modelos de datos se utilizan ciertas convenciones para representar los esquemas en forma de diagramas, así que la representación de un esquema se denomina diagrama del esquema. A cada uno de los objetos del esquema —como ESTUDIANTE o CURSO— se le llama elemento del esquema.

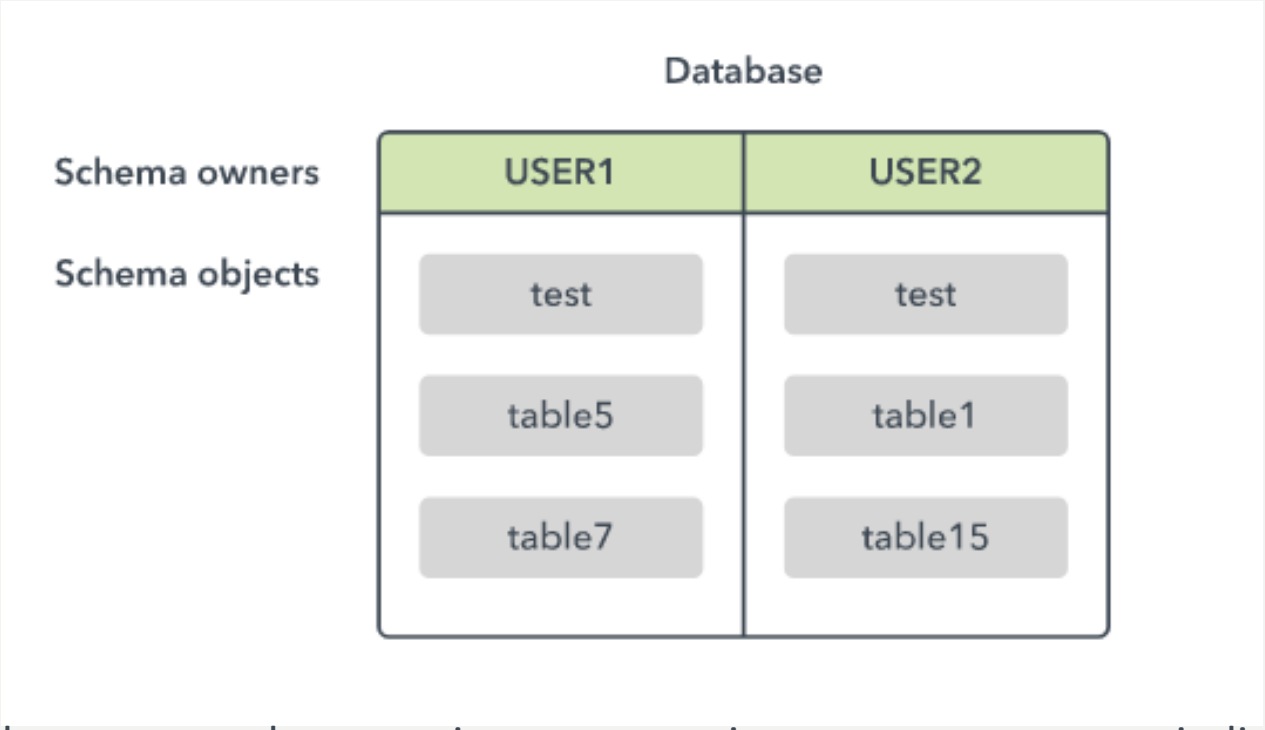
CREACIÓN ESQUEMAS-USUARIOS

En el sistema de base de datos Oracle, el término *esquema de base de datos*, al cual también se lo conoce como "esquema SQL", tiene un significado diferente.

Aquí, una base de datos puede tener esquemas múltiples (o "schemata", como se le dice elegantemente en inglés).

Cada uno de ellos contiene todos los objetos creados por un usuario específico de la base de datos. Esos objetos pueden incluir tablas, vistas, sinónimos y mucho más. Algunos objetos no se pueden incluir en un esquema, tales como usuarios, contextos, roles y objetos del directorio.

CREACIÓN ESQUEMAS-USUARIOS



Se puede conceder acceso a los usuarios para que ingresen a esquemas individuales según cada caso concreto, y la titularidad es transferible

CREACIÓN ESQUEMAS-USUARIOS

- Estos esquemas no necesariamente indican las formas en que los archivos de datos se almacenan físicamente.
- Los esquemas y los espacios de tablas no necesariamente se alinean a la perfección: los objetos de un esquema pueden estar presentes en múltiples espacios de tablas, mientras que un espacio de tablas puede incluir objetos de varios esquemas.
- Los esquemas y las instancias de bases de datos pueden afectarse entre sí a través de un sistema de administración de bases de datos (DBMS). El DBMS asegura que cada instancia de la base de datos cumpla con las restricciones impuestas por los diseñadores de la base en el esquema de la base de datos.

GRANT

Use la declaración GRANT para otorgar privilegios a un usuario o rol específico, o a todos los usuarios, para realizar acciones en los objetos de la base de datos. También puede usar la declaración GRANT para otorgar un rol a un usuario

Se pueden otorgar los siguientes tipos de privilegios:

- Eliminar datos de una tabla específica.
- Inserta datos en una tabla específica.
- Cree una referencia de clave externa a la tabla nombrada o a un subconjunto de columnas de una tabla.
- Seleccione datos de una tabla, vista o un subconjunto de columnas en una tabla.
- Crea un disparador en una tabla.
- Actualice datos en una tabla o en un subconjunto de columnas en una tabla.
- Ejecuta una función o procedimiento especificado.
- Use un generador de secuencia o un tipo definido por el usuario.

GRANT

Para utilizar la sentencia GRANT se debe seguir la siguiente sintaxis:

GRANT <NOMBRE DEL PRIVILEGIO 1>,…<NOMBRE DEL PRIVILEGIO N> TO <NOMBRE DEL ROL O USUARIO AL CUAL SE DESEA ASIGNAR LOS PRIVILEGIOS>;

Ejemplo:

GRANT create session, create any table TO user_carlos;

REVOKE

Esta sentencia sirve para quitar permisos (o privilegios) que se han asignado a un usuario o a un rol. También es utilizada para eliminar un rol que se ha asignado a un usuario.

La sintaxis para utilizar esta sentencia es la siguiente:

REVOKE <NOMBRE DEL PRIVILEGIO 1>,…<NOMBRE DEL PRIVILEGIO N> FROM <NOMBRE DEL ROL O USUARIO AL CUAL SE DESEA ASIGNAR LOS PRIVILEGIOS>;

Ejemplo:

REVOKE create session, create any table FROM user_carlos;

Eliminar todos los privilegios de select, update, insert y delete asignados a tablas de otro usuario

REVOKE ALL ON <NOMBRE DE TABLA> FROM <USUARIO O ROL>;

Sintaxis para quitar un rol asignado:

REVOKE <nombre del rol> FROM <usuario>;

CREACIÓN DE USUARIOS

- Esta sentencia sirve para crear un usuario oracle.
- Un usuario es un nombre de acceso a la base de datos oracle. Normalmente va asociado a una clave (password).
- Lo que puede hacer un usuario una vez ha accedido a la base de datos depende de los permisos que tenga asignados ya sea directamente (GRANT) como sobre algún rol que tenga asignado (CREATE ROLE).
- El perfil que tenga asignado influye en los recursos del sistema de los que dispone un usuario a la hora de ejecutar oracle (CREATE PROFILE).

CREACIÓN DE USUARIOS

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

```
CREATE USER miusuario IDENTIFIED BY contraseña
        DEFAULT TABLESPACE users
        TEMPORARY TABLESPACE temp
        QUOTA UNLIMITED ON users;
CREATE ROLE programador;

GRANT CREATE session, CREATE any table, CREATE any view,
        CREATE any procedure,
        ALTER any table, ALTER any procedure,
        DROP any table, DROP any view, DROP any procedure
        TO programador;
GRANT programador TO miusuario;
```

CREACIÓN DE USUARIOS

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

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

CREACIÓN DE USUARIOS

```
CREATE USER username
        IDENTIFIED {BY password | EXTERNALLY | GLOBALLY AS 'external_name’}
            options;
```

Donde options:

```
DEFAULT TABLESPACE tablespace
TEMPORARY TABLESPACE tablespace
QUOTA int {K | M} ON tablespace
QUOTA UNLIMITED ON tablespace
PROFILE profile_name
PASSWORD EXPIRE
ACCOUNT {LOCK|UNLOCK}
```

CREACIÓN DE ROLES

Un rol es una forma de agrupar permisos (o privilegios) para asignarlos luego a los usuarios.
Cada usuario puede tener varios roles.
Sintaxis: CREATE ROLE <NOMBRE DEL ROL>

Ejemplo de creación de un rol:
CREATE ROLE MI_UNICO_ROLE
Crea un rol sin password:
CREATE ROLE role NOT IDENTIFIED
Crea un rol con password:
CREATE ROLE role IDENTIFIED BY password

BLOQUES ANÓNIMOS

La sintaxis básica para crear un bloque anónimo es la siguiente:

```
DECLARE
        <DECLARACIÓN DE VARIABLES>;
BEGIN
        <INSTRUCCIONES SQL>;
END;
```

TIPOS DE DATOS

- Para declarar una variable de cadena , debe seleccionar uno de los muchos tipos de datos de cadena Oracle, incluyendo CHAR, NCHAR, VARCHAR2, NVARCHAR2, CLOB y NCLOB. Los tipos de datos que tienen el prefijo '*N* se utilizan para almacenar datos de caracteres Unicode.
- Si la cadena puede contener más de 32.767 caracteres , utilice tipos CLOB (o NCLOB)
- Si el valor asignado a una cadena siempre tiene una longitud fija (tal como el Registro Federal de Causantes (RFC), que siempre tiene el mismo formato y longitud , AAAA-NNNNNN-XXX), utilice CHAR(o NCHAR).
- De lo contrario (y , por lo tanto , la mayoría de las veces), utilice el tipo de datos VARCHAR2 (o NVACHAR2, cuando se trabaje con datos Unicode).

TIPOS DE DATOS

Diferentes tamaños máximos.

Hay una serie de diferencias entre SQL y PL/SQL para los tamaños máximos para los tipos de datos de cadena. En PL/SQL, el tamaño máximo para VARCHAR2 es 32.767 bytes, mientras que en SQL el máximo es de 4.000 bytes. En PL/SQL, el tamaño máximo para CHAR es 32.767 bytes , mientras que en SQL el máximo es de 2.000 bytes.

Usar SUBSTR para extraer no más de 4.000 bytes de la cadena, y salvar la subcadena en la tabla. Esta opción tiene claramente un inconveniente: se pierde parte de sus datos.

TIPOS DE DATOS

1.Para tipos numéricos se usa comúnmente el tipo NUMBER, que engloba enteros como números reales.

2.PL/SQL permite variables de tipo BOOLEAN, sin embargo, Oracle no soporta dicho tipo para las columnas de la base de datos.

3.Podemos usar la siguiente sentencia para hacer que la variable 'miCarro' sea declarada con el mismo tipo que tiene la columna 'modelo' en la tabla 'Automovil':

```
miCarro Automovil.modelo%TYPE;
```

TIPOS DE DATOS

- La siguiente sentencia hace que la variable 'miCarro' sea un registro con los campos '*modelo*' y 'year', asumiendo la tabla tiene esos campos, automovil(modelo, year):

```
miCarro Automovil%ROWTYPE;
```

El valor inicial de cualquier variable, sin importar su tipo, es NULL.
Para hacer asignaciones usamos el operador ':='. Ejemplo:

```
DECLARE
 miCarro Automovil%ROWTYPE;
BEGIN
 miCarro.modelo = 'Jetta';
END;
/
```

SENTENCIAS DE CONTROL

En PL/SQL tenemos grupos de sentencias de control:

- Condicional
- Iterativo
- Secuencial

SENTENCIAS DE CONTROL - CONDICIONAL

- La estructura condicional está representada por las sentencias IF y CASE.
- **IF-THEN-ELSE(SI-ENTONCES-CASO CONTRARIO)**
- Ésta sentencia permite la ejecución de una o varias líneas de código según una condición. La condición se especifica en la cláusula IF, en caso de que la condición sea verdadera se ejecuta el bloque de código especificado bajo la cláusula THEN, en caso contrario se ejecuta el bloque de código especificado en la cláusula ELSE. La cláusula ELSE es opcional

SENTENCIAS DE CONTROL - CONDICIONAL

- **Ejemplo**: Manejo de cadenas. El objetivo de este código es demostrar el uso de variables y condiciones

```
DECLARE
  l_variable VARCHAR2(10) := 'Veracruz';
  l_fixed    CHAR (10) := 'Veracruz';
BEGIN
  IF l_variable = l_fixed
  THEN
   DBMS_OUTPUT.put_line ('Igual');
  ELSE
   DBMS_OUTPUT.put_line ('Diferente');
  END IF;
END;
/
```

SENTENCIAS DE CONTROL - CONDICIONAL

```
01  DECLARE
02  /*Para poder ver la utilidad de éste script
03  basta con cambiar los valores de las variables
04  V_A y V_B según queramos. */
05  V_A NUMBER := 30;
06  V_B NUMBER := 10;
07
08  BEGIN
09    /*En caso de que la variable V_A sea mayor
10     se mostrará un mensaje*/
11    IF V_A > V_B THEN
12      dbms_output.put_line('V_A es mayor');
13    END IF;
14  END;
```

```
V_A es mayor
```

SENTENCIAS DE CONTROL - CONDICIONAL

- El operador > (mayor que) permite realizar una comparación lógica entre 2 expresiones, en este caso, 2 variables. Así como existe el operador > (mayor que) existen los operadores <(menor que), = (igual que), <>(diferente que), etc. Éstos operadores son operadores de comparación.
- La condición para la sentencia IF del ejemplo anterior, está dada por la comparación V_A > V_B, en caso de que el valor de V_A sea mayor que el de V_B se mostrará el mensaje "V_A es mayor", en caso de que V_B sea mayor o ambos iguales no se mostrará nada.

SENTENCIAS DE CONTROL - CONDICIONAL

- Uso de la cláusula ELSE:

```
01 DECLARE
02 /*Para poder ver la utilidad de éste script
03 basta con cambiar los valores de las variables
04 V_A y V_B según queramos. */
05 V_A NUMBER := 30;
06 V_B NUMBER := 30;
07 BEGIN
08   /*En caso de que la variable V_A sea mayor
09    se mostrará un mensaje*/
10   IF V_A > V_B THEN
11     dbms_output.put_line('V_A es mayor');
12   ELSE
13     dbms_output.put_line('V_B es mayor o igual que V_A');
14   END IF;
15 END;
```

```
1 V_B es mayor o igual que V_A
```

Para el ejemplo anterior, se ha adicionado la cláusula ELSE, la cual se ejecutará cuando la variable V_A no sea mayor que la variable V_B.

SENTENCIAS DE CONTROL - CONDICIONAL

CASE(CASO)

- La sentencia CASE permite evaluar diferentes opciones, por cada opción se realiza una instrucción específica. Veamos un ejemplo:

```
01  DECLARE
02  V_TIPO_TRABAJADOR VARCHAR2(2) := 'AP';
03  V_SALARIO         NUMBER;
04  BEGIN
05
06    CASE
07      WHEN V_TIPO_TRABAJADOR = 'AP' THEN
08        V_SALARIO := 1500;
09      WHEN V_TIPO_TRABAJADOR = 'AS' THEN
10        V_SALARIO := 2300;
11      WHEN V_TIPO_TRABAJADOR = 'JP' THEN
12        V_SALARIO := 5000;
13    END CASE;
14
15    DBMS_OUTPUT.PUT_LINE('El salario de un ' || V_TIPO_TRABAJADOR
16                         || ' es ' || V_SALARIO);
17  END;
```

```
El salario de un AP es 1500
```

SENTENCIAS DE CONTROL - CONDICIONAL

- También se puede usar la sentencia CASE de manera simple para realizar la asignación de un valor según la evaluación de una variable por diferentes casos:

```
01  DECLARE
02  V_TIPO_TRABAJADOR VARCHAR2(2) := 'AP';
03  V_SALARIO         NUMBER;
04  BEGIN
05
06    V_SALARIO:= CASE V_TIPO_TRABAJADOR
07                  WHEN 'AP' THEN   1500
08                  WHEN 'AS' THEN   2300
09                  WHEN 'JP' THEN   5000
10                END;
11
12    DBMS_OUTPUT.PUT_LINE('El salario de un ' || V_TIPO_TRABAJADOR
13                         || ' es ' || V_SALARIO);
14  END;
```

```
El salario de un AP es 1500
```

SENTENCIAS DE CONTROL - CONDICIONAL

- Finalmente, podemos hacer uso de la cláusula ELSE en caso de que ninguna de las opciones definidas en WHEN sea seleccionada:

```
01  DECLARE
02  V_TIPO_TRABAJADOR VARCHAR2(2) := 'AR';
03  V_SALARIO         NUMBER;
04  V_TIENE_SALARIO   BOOLEAN;
05  BEGIN
06
07    CASE
08      WHEN V_TIPO_TRABAJADOR = 'AP' THEN
09        V_SALARIO := 1500;
10        V_TIENE_SALARIO := TRUE;
11      WHEN V_TIPO_TRABAJADOR = 'AS' THEN
12        V_SALARIO := 2300;
13         V_TIENE_SALARIO := TRUE;
14      WHEN V_TIPO_TRABAJADOR = 'JP' THEN
15        V_SALARIO := 5000;
16        V_TIENE_SALARIO := TRUE;
17      ELSE
18        V_TIENE_SALARIO := FALSE;
19    END CASE;
20
21    IF V_TIENE_SALARIO THEN
22      DBMS_OUTPUT.PUT_LINE('El salario de un ' || V_TIPO_TRABAJADOR
23                           || ' es ' || V_SALARIO);
24    ELSE
25      DBMS_OUTPUT.PUT_LINE(V_TIPO_TRABAJADOR
26                           || ' no tiene un salario definido');
27    END IF;
28  END;
```

```
AR no tiene un salario definido
```

SENTENCIAS DE CONTROL - ITERATIVO

- Las sentencias iterativas permiten ejecutar varias instrucciones múltiples veces. A éstas sentencias también se les conoce por el nombre de bucles repetitivos.
- **FOR-LOOP**
- La sentencia FOR-LOOP permite especificar un rango de números enteros, finalmente ejecuta una secuencia de instrucciones para cada número entero dentro de la lista de números.

```
1  BEGIN
2    FOR i IN 1..8 LOOP
3      DBMS_OUTPUT.PUT_LINE('El número de ésta iteracción es:' || i);
4    END LOOP;
5  END;
```

```
El número de ésta iteracción es:1
El número de ésta iteracción es:2
El número de ésta iteracción es:3
El número de ésta iteracción es:4
El número de ésta iteracción es:5
El número de ésta iteracción es:6
El número de ésta iteracción es:7
El número de ésta iteracción es:8
```

SENTENCIAS DE CONTROL - ITERATIVO

- En el ejemplo anterior, sólo existe una instrucción para imprimir un mensaje en la sentencia LOOP, pero sin embargo en la salida el mensaje se repite 8 veces con los números del 1 al 8. En la cláusula IN la expresión 1..8 genera una lista de números, para éste caso , del 1 al 8.
- Probemos ahora generando la tabla de multiplicar del 7 del 1 al 12:

```
1  BEGIN
2    FOR i IN 1..12 LOOP
3      DBMS_OUTPUT.PUT_LINE('7*' || i || '='|| (7*i));
4    END LOOP;
5  END;
```

```
7*1=7
7*2=14
7*3=21
7*4=28
7*5=35
7*6=42
7*7=49
7*8=56
7*9=63
7*10=70
7*11=77
7*12=84
```

SENTENCIAS DE CONTROL - ITERATIVO

- **WHILE-LOOP**
- La sentencia de control WHILE LOOP, al igual que FOR-LOOP, permite la ejecución de una o varias instrucciones. A diferencia de la sentencia FOR-LOOP, en WHILE-LOOP la ejecución se define a partir de una condición

```
1  DECLARE
2    V_NUM NUMBER := 1;
3  BEGIN
4    WHILE V_NUM <= 13 LOOP
5      DBMS_OUTPUT.PUT_LINE('El valor V_NUM es :' || V_NUM);
6      V_NUM := V_NUM + 2.1;
7    END LOOP;
8  END;
```

```
1  El valor V_NUM es :1
2  El valor V_NUM es :3,1
3  El valor V_NUM es :5,2
4  El valor V_NUM es :7,3
5  El valor V_NUM es :9,4
6  El valor V_NUM es :11,5
```

SENTENCIAS DE CONTROL - ITERATIVO

- En el ejemplo anterior, al igual que en la sentencia FOR-LOOP, el mensaje "El valor de V_NUM es ..." se repite varias veces, pero ésta vez la variable V_NUM se ve incrementada en 2.1 cada vez que la sentencia WHILE-LOOP se ejecuta.
- En la cláusula WHILE, mientras la condición **V_NUM <= 13** es verdadera se ejecutará el código dentro de la cláusula LOOP.
- Finalmente debido al incremento realizado en la línea 6 (V_NUM := V_NUM + 2.1) el código sólo se repetirá 6 veces.

SENTENCIAS DE CONTROL - ITERATIVO

LOOP

- La sentencia LOOP permite ejecutar un bloque de código de manera infinita, a diferencia de las otras sentencias iterativas, en la sentencia LOOP se debe utilizar la cláusula EXIT WHEN para colocar una condición para terminar el bucle:

```
01 DECLARE
02   V_TOTAL    NUMBER := 0;
03   V_CONTADOR NUMBER := 0;
04 BEGIN
05   LOOP
06     V_CONTADOR := V_CONTADOR + 1;
07     V_TOTAL := V_TOTAL + V_CONTADOR;
08     DBMS_OUTPUT.PUT_LINE('Valor de V_TOTAL:' || V_TOTAL);
09     EXIT WHEN V_TOTAL > 20;
10   END LOOP;
11 END;
```

```
Valor de V_TOTAL:1
Valor de V_TOTAL:3
Valor de V_TOTAL:6
Valor de V_TOTAL:10
Valor de V_TOTAL:15
Valor de V_TOTAL:21
```

En el ejemplo anterior, la sentencia LOOP terminará cuando el valor de la variables V_TOTAL sea mayor a 20.

SENTENCIAS DE CONTROL - SECUENCIAL

- La sentencia GOTO permite cambiar la línea secuencial de la ejecución hacia una etiqueta. Las etiquetas son marcadas con los símbolos << y >>. Para ir dirigir la ejecución hacia una etiqueta simplemente hay que usar la sentencia GOTO:

```
01 DECLARE
02   V_TOTAL    NUMBER(9) := 0;
03   V_CONTADOR NUMBER(6) := 0;
04 BEGIN
05
06   <<calcular>>
07   V_CONTADOR := V_CONTADOR + 1;
08   V_TOTAL := V_TOTAL + V_CONTADOR;
09
10   IF V_TOTAL <= 20 THEN
11     DBMS_OUTPUT.PUT_LINE('Valor de V_TOTAL:' || V_TOTAL);
12     GOTO calcular;
13   END IF;
14 END;
```

```
Valor de V_TOTAL:1
Valor de V_TOTAL:3
Valor de V_TOTAL:6
Valor de V_TOTAL:10
Valor de V_TOTAL:15
```

En el ejemplo anterior, la sentencia GOTO redireccionará la ejecución hacia la línea 6 mientras el valor de V_TOTAL sea menor a 20.

SENTENCIA SELECT INTO

La instrucción SELECT INTO recupera datos de una o más tablas de base de datos y asigna los valores seleccionados a variables o colecciones.

En su uso predeterminado (SELECT ... INTO), esta instrucción recupera una o más columnas de una sola fila.

Toda instrucción SELECT dentro de un bloque anónimo debe estar acompañado del comando INTO.

La sintaxis es la siguiente

SELECT <columnas> INTO <variables> FROM <tabla> WHERE <criterios>;

SENTENCIA SELECT INTO

```
DECLARE
  deptid        employees.department_id%TYPE;
  jobid         employees.job_id%TYPE;
  emp_rec       employees%ROWTYPE;
  TYPE emp_tab IS TABLE OF employees%ROWTYPE INDEX BY PLS_INTEGER;
  all_emps      emp_tab;
BEGIN
  SELECT department_id, job_id INTO deptid, jobid
     FROM employees WHERE employee_id = 140;
  IF SQL%FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Dept Id: ' || deptid || ', Job Id: ' || jobid);
  END IF;
  SELECT * INTO emp_rec FROM employees WHERE employee_id = 105;
  SELECT * BULK COLLECT INTO all_emps FROM employees;
  DBMS_OUTPUT.PUT_LINE('Number of rows: ' || SQL%ROWCOUNT);
END;
/
```

CURSORES

Los cursores permiten recorrer los registros que devuelve una consulta SQL.

También las operaciones INSERT, UPDATE Y DELETE definen un cursor implícito, denominado SQL.

Atributos de un cursor:

%ROWCOUNT: Cantidad de registros

%FOUND: Indica que hay datos (tipo boolean)

%NOTFOUND: Indica que no hay datos (tipo boolean)

%ISOPEN: Indica si el cursor está abierto (tipo boolean)

CURSORES

Control de Cursores

- 1°. Crear el cursor en el área específica del DECLARE
- 2°. Identificar el cursor y abrirlo OPEN
- 3°. Cargar la fila actual en variables FETCH
- 4°. Si todavía existen filas sin leer, volver a 3°.
- 5°. Si no existen más filas a leer CLOSE

CURSORES

Declaración del Cursor

Sintaxis:

CURSOR <NOMBRE DEL CURSOR> IS <SENTENCIA SQL A EJECUTAR>;

CURSORES

Ejemplo de Declaración:

```
DECLARE
        CURSOR C1 IS SELECT EMPNO, ENAME, JOB, SAL FROM EMP WHERE
        SAL > 2000;
BEGIN
        <INSTRUCCIONES SQL>
END;
```

CURSORES

- Apertura del cursor

OPEN <NOMBRE DEL CURSOR>;

- Recuperación de datos

FETCH <NOMBRE DEL CURSOR> INTO <VARIABLE 1>,…,<VARIABLE N>|<NOMBREL DEL REGISTRO>

- Cierre del cursor

CLOSE <NOMBRE DEL CURSOR>

FUNCIONES PARA LOS ARREGLOS ASOCIATIVOS

Cuando trabajamos con arreglos asociativos podemos utilizar las siguientes funciones:

- COUNT: Devuelve el número de elementos de la tabla PL/SQL.
- PRIOR (n): Devuelve el número del índice anterior a n en la tabla.
- NEXT(n): Devuelve el número del índice posterior a n en la tabla.
- DELETE(n): Borra el elemento correspondiente al índice n.
- DELETE(m,n): Borra los elementos entre el índice m y n.

EXCEPCIONES

Las instrucciones SQL que se tengan dentro del bloque anónimo pueden ocasionar un error y esto hará que el bloque anónimo no termine su ejecución de forma exitosa. Para evitar que suceda esto y que se pueda realizar alguna acción al presentarse un error, se tiene que hacer uso de las excepciones.

Una excepción es un error que se produce en PL/SQL y que se puede capturar para gestionarlo de acuerdo a las acciones que se definan.

EXCEPCIONES

La sintaxis para utilizar las excepciones es:

```
DECLARE
        <DECLARACIÓN DE VARIABLES>
BEGIN
        <INSTRUCCIONES SQL>;
        EXCEPTION
                WHEN <NOMBRE DE EXCEPCIÓN 1> THEN
                        <INSTRUCCIONES SQL>;
                .
                .
                .
                WHEN <NOMBRE DE EXCEPCIÓN N> THEN
                        <INSTRUCCIONES SQL>;

END;
```

EXCEPCIONES

```
DECLARE
        <DECLARACIÓN DE  VARIABLES>
BEGIN
        <INSTRUCCIONES SQL>;
        EXCEPTION
                WHEN <NOMBRE DE EXCEPCIÓN> THEN
                        <INSTRUCCIONES SQL>;
                WHEN OTHERS THEN
                        <INSTRUCCIONES SQL>;
END;
```

EXCEPCIONES

Dentro del bloque de excepciones podemos utilizar dos variables que se habilitan cuando se genera el error, las variables son:

- SQLCODE= retorna el código del error que se generó, el código es un valor numérico.
- SQLERRM= retorna un mensaje informando del error que se ha generado

SECUENCIAS

Una secuencia (sequence) se emplea para generar valores enteros secuenciales únicos y asignárselos a campos numéricos; se utilizan generalmente para las claves primarias de las tablas garantizando que sus valores no se repitan.

Una secuencia es una tabla con un campo numérico en el cual se almacena un valor y cada vez que se consulta, se incrementa tal valor para la próxima consulta.

Sintaxis general:

CREATE SEQUENCE <NOMBRESECUENCIA>

START WITH <VALOR ENTERO>

INCREMENT BY <VALOR ENTERO>

MAXVALUE <VALOR ENTERO>

MINVALUE <VALOR ENTERO>

CYCLE | NOCYCLE;

SECUENCIAS

La cláusula "start with" indica el valor desde el cual comenzará la generación de números secuenciales. Si no se especifica, se inicia con el valor que indique "minvalue".

La cláusula "increment by" especifica el incremento, es decir, la diferencia entre los números de la secuencia; debe ser un valor numérico entero positivo o negativo diferente de 0. Si no se indica, por defecto es 1.

"maxvalue" define el valor máximo para la secuencia. Si se omite, por defecto es 999999999999999999999999.

"minvalue" establece el valor mínimo de la secuencia. Si se omite será 1.

La cláusula "cycle" indica que, cuando la secuencia llegue a máximo valor (valor de "maxvalue") se reinicie, comenzando con el mínimo valor ("minvalue") nuevamente, es decir, la secuencia vuelve a utilizar los números. Si se omite, por defecto la secuencia se crea "nocycle".

SECUENCIAS

En el siguiente ejemplo creamos una secuencia llamada "sec_codigolibros", estableciendo que comience en 1, sus valores estén entre 1 y 99999 y se incrementen en 1, por defecto, será "nocycle":

```
create sequence sec_codigolibros
start with 1
increment by 1
maxvalue 99999
minvalue 1;
```

Con las secuencias podemos hacer uso de dos funciones:

- .NEXTVAL=obtiene el siguiente valor de la secuencia de acuerdo al valor de incremento
- .CURRVAL=obtiene el valor actual de la secuencia

CAMPOS AUTOINCREMENTABLES

- Creación de la tabla:

```
create table ventas(
    id_venta number(5) primary key,
    id_articulo number(5) references articulos,
    id_empleado number(6) references empleado,
    id_cliente number(5) references cliente,
    cantidad number(10) not null,
    precio number(8) not null,
    total number(8) not null,
    fecha_venta date );
```

CAMPOS AUTOINCREMENTABLES

Para utilizar la secuencia en un campo autoincremental hay dos opciones, la primera es utilizarla dentro de la instrucción insert into y la segunda opción es utilizarla dentro de un trigger para que la secuencia se gestione automáticamente al insertar.

```
create sequence sq_venta
start with 1
 increment by 1     -- aqui lo que se hace es determinar a partir de que valor se va
                    -- a iniciar y cuánto se va a incrementar
order;
```

INSERT INTO ventas VALUES (sq_venta.NEXTVAL, <resto de valores a insertar>);

GESTIÓN DE TRANSACCIONES

Una transacción está formada por una serie de instrucciones DML. Una transacción comienza con la primera instrucción DML que se ejecute y finaliza con alguna de estas circunstancias:

- Una operación COMMIT o ROLLBACK
- Una instrucción DDL (como ALTER TABLE por ejemplo)
- Una instrucción del Lenguaje de Control de Datos (DCL, como GRANT o REVOKE)
- El usuario abandona la sesión
- Caída del sistema

Hay que tener en cuenta que cualquier instrucción DDL o DCL da lugar a un COMMIT implícito, es decir todas las instrucciones DML ejecutadas hasta ese instante pasan a ser definitivas.

GESTIÓN DE TRANSACCIONES

Una transacción es un conjunto de operaciones que se ejecutan en una base de datos, y que son tratadas como una única unidad lógica por el SGBD. Es decir, una transacción es una o varias sentencias SQL que se ejecutan en una base de datos como una única operación, confirmándose o deshaciéndose en grupo.

No todas las operaciones SQL son transaccionales. Sólo son transaccionales las operaciones correspondiente al DML, es decir, sentencias INSERT, UPDATE y DELETE

Para confirmar una transacción se utiliza la sentencia COMMIT. Cuando realizamos COMMIT los cambios se escriben en la base de datos.

Para deshacer una transacción se utiliza la sentencia ROLLBACK. Cuando realizamos ROLLBACK se deshacen todas las modificaciones realizadas por la transacción en la base de datos, quedando la base de datos en el mismo estado que antes de iniciarse la transacción.

GESTIÓN DE TRANSACCIONES

El siguiente ejemplo muestra una transacción bancaria:

```
DECLARE
   importe NUMBER;
   ctaOrigen VARCHAR2(23);
   ctaDestino VARCHAR2(23);
BEGIN
    importe := 100;
    ctaOrigen  := '2530 10 2000 1234567890';
    ctaDestino := '2532 10 2010 0987654321';
    UPDATE CUENTAS SET SALDO = SALDO – importe WHERE CUENTA = ctaOrigen;
    UPDATE CUENTAS SET SALDO = SALDO + importe WHERE CUENTA = ctaDestino;
    INSERT INTO MOVIMIENTOS (CUENTA_ORIGEN, CUENTA_DESTINO,IMPORTE, FECHA_MOVIMIENTO)
    VALUES (ctaOrigen, ctaDestino, importe*(-1), SYSDATE);
    INSERT INTO MOVIMIENTOS (CUENTA_ORIGEN, CUENTA_DESTINO,IMPORTE, FECHA_MOVIMIENTO)
    VALUES (ctaDestino,ctaOrigen, importe, SYSDATE);
    COMMIT;


    EXCEPTION
        WHEN OTHERS THEN
        dbms_output.put_line('Error en la transaccion:'||SQLERRM);
        dbms_output.put_line('Se deshacen las modificaciones);
        ROLLBACK;
END;
```

GESTIÓN DE TRANSACCIONES

Si alguna de las tablas afectadas por la transacción tiene triggers, las operaciones que realiza el trigger están dentro del ámbito de la transacción, y son confirmadas o deshechas conjuntamente con la transacción.

Durante la ejecución de una transacción, una segunda transacción no podrá ver los cambios realizados por la primera transacción hasta que esta se confirme.

ORACLE es completamente transaccional. Siempre debemos especificar si queremos deshacer o confirmar la transacción.

DROP TABLE Elimina una tabla (datos y estructura) y sus índices. No se puede hacer Rollback de esta sentencia.

TRIGGERS

Un trigger es un disparador automático que se asocia a una tabla de la base de datos, el cual se puede ejecutar antes o después de una operación INSERT, UPDATE o DELETE.

La sintaxis es la siguiente:

```
CREATE OR REPLACE TRIGGER <NOMBRE_TRIGGER>
BEFORE|AFTER INSERT|DELETE|UPDATE ON <TABLA>
FOR EACH ROW
DECLARE
        <DECLARACIÓN DE  VARIABLES>
BEGIN
        <INSTRUCCIONES SQL>
END;
```

TRIGGERS

Existen 2 variables que se pueden utilizar en los triggers, estas son:

- :NEW y
- :OLD

Estas variables se pueden utilizar en base a la siguiente tabla:

Data Operations	Old Value	New Value
INSERT	NULL	Inserted value
UPDATE	Value before update	Value after update
DELETE	Value before delete	NULL