



SENTENCIAS DE MANIPULACIÓN Y CREACIÓN DE DATOS

TIPO DE SENTENCIAS

- Las sentencias de base de datos pueden ser de los siguientes tipos:

TIPO	DESCRIPCION
DDL (Data Definition Language)	Sentencias que son usadas para definir la estructura o esquema de la base de datos (crear tablas o modificar su estructura)
DML (Data Manipulation Language)	Sentencias que son usadas para manejar los datos en la base (leer, modificar y borrar datos)
DCL (Data Control Language)	Utilizado para controlar el acceso de un usuario a la base de datos (otorgar privilegios, confirmar transacciones)

SENTENCIAS DML

COMANDO	DESCRIPCION
SELECT	Recupera datos de la base de datos
INSERT	Agrega una nueva fila en la tabla
UPDATE	Modifica filas de la tabla
DELETE	Elimina filas de la tabla
MERGE	Permite modificar una tabla combinando los datos con otra tabla
EXPLAIN PLAN	Describe el camino que toma la base para acceder a una tabla

INSERT: Sintaxis

- Agregue nuevas filas a una tabla usando el comando de **INSERCIÓN**
- Con esta sintaxis, se inserta **UNA SOLA FILA** a la vez

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

Ejemplo de Inserción

- Inserte una nueva fila que contiene valores por cada columna:

```
INSERT INTO B_AREAS VALUES (1, 'GERENCIA GENERAL',  
                             TO_DATE('01-01-1990', 'DD-MM-YYYY'), 'S', NULL);
```

- Como puede notarse, cuando todas las columnas tienen correspondientemente valores, éstas pueden listarse opcionalmente en la cláusula de inserción.
- Si no se especifican las columnas dentro de la cláusula INSERT, los valores que siguen en la cláusula VALUES deben indicarse exactamente en el orden en que las columnas fueron definidas en las tablas.
- Los valores para columnas de tipo carácter y fecha deben ir entre apóstrofes.

Cuando no existen datos para ciertas columnas:

- Método implícito
 - Omite la columna de la lista de las columnas

```
INSERT INTO B_AREAS (ID, NOMBRE_AREA, FECHA_CREA, ACTIVA)
VALUES (1, 'GERENCIA GENERAL', TO_DATE('01-01-1990', 'DD-MM-
YYYY'), 'S');
```

- Método explícito
 - Especifique la palabra NULL en la lista de valores.

```
INSERT INTO B_AREAS VALUES (1, 'GERENCIA GENERAL',
TO_DATE('01-01-1990', 'DD-MM-YYYY'), 'S', NULL);
```

Insertando Valores Especiales

- La función USER registra el nombre del usuario actual
- La función SYSDATE registra la fecha y hora actual

```
INSERT INTO B_DIARIO_CABECERA (ID, FECHA, CONCEPTO,  
                                USUARIO, FECHA_CIERRE, ID_AREA) VALUES (1,  
to_date('03-01-2013', 'dd-mm-yyyy'), 'APORTE DE CAPITAL  
INICIAL', USER, SYSDATE, 3);
```


Insertando Valores específicos de Fecha y Hora

- Función TO_DATE

Permite insertar una fecha y hora específicas.

- El Siglo por defecto es el siglo actual
- La Hora por defecto es la media noche

```
INSERT INTO B_DIARIO_CABECERA (ID, FECHA,  
                                CONCEPTO, USUARIO, FECHA_CIERRE, ID_AREA)  
                                VALUES (1, TO_DATE('01-01-2012 10:00',  
'DD-MM-YYYY HH:MI') , 'APORTE DE CAPITAL  
INICIAL', USER, SYSDATE, 3);
```


Uso de 'variables de sustitución'

- Permite el ingreso interactivo de valores mediante parámetros (variables de sustitución)

```
INSERT INTO B_AREAS (ID, NOMBRE_AREA, FECHA_CREA,  
ACTIVA) VALUES  
(&pid, '&pnombre_area', '&fecha_crea', '&pactiva');
```

Introduzca un valor para pid: 1

Introduzca un valor para pnombre_area :GERENCIA

Introduzca un valor para fecha_crea : 03/04/14

Introduzca un valor para pactiva : S

Creando un script con Prompts personalizados

- El comando ACCEPT del SQL*Plus almacena valores en una variable.
- Con PROMPT puede desplegar un texto personalizado.

```
ACCEPT pid NUMBER PROMPT 'Introduzca un valor para ID: ';
ACCEPT pnombre_area CHAR PROMPT 'Ingrese nombre de Area: ';
ACCEPT pfecha_crea DATE PROMPT 'Ingrese Fecha: ';
ACCEPT pactiva CHAR PROMPT 'Ingrese Estado: ';

INSERT INTO B_AREAS (ID, NOMBRE_AREA, FECHA_CREA, ACTIVA)
VALUES (&pid, '&pnombre_area', '&pfecha_crea', '&pactiva');
```

Copiando Registros de otra Tabla

- Se puede insertar valores utilizando SUBQUERIES
- Cuando se utiliza subconsultas para la inserción no se utiliza la cláusula VALUES
- El número de columnas en la cláusula de inserción debe coincidir con aquellas del subquery

```
Insert into b_empleados (CEDULA, NOMBRE, APELLIDO,  
    FECHA_ING, FECHA_NACIM, TELEFONO, DIRECCION,  
    BARRIO)  
select to_number(cedula), nombre, apellido, sysdate,  
    to_date(fecha_nacimiento, 'dd/mm/yyyy'), telefono,  
    direccion, 'CENTRO' from b_personas where id=1;
```

Inserción a través de un subquery

- Es posible utilizar un subquery en lugar de una tabla en la cláusula INTO:

INSERT INTO

```
(select CEDULA, NOMBRE, APELLIDO, FECHA_ING,  
      FECHA_NACIM, TELEFONO, DIRECCION, BARRIO  
      from b_empleados where cedula = '1309873')  
values ('3524610', 'JUAN', 'PORTILLO', SYSDATE,  
        to_date('02/08/86','dd/mm/yyyy'), '345-865',  
        'ESPAÑA 345', 'CARMELITAS');
```

Inserción a través de un subquery

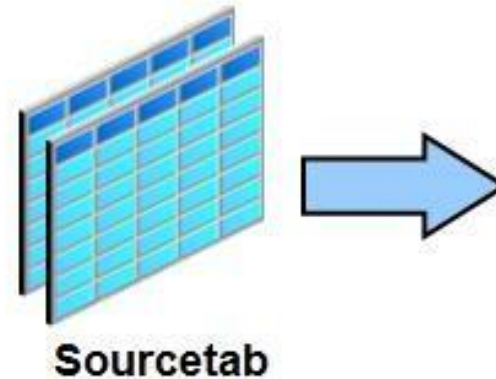
- Use **WITH CHECK OPTION** para evitar que se inserten valores no contemplados en el subquery

```
INSERT INTO  
(select CEDULA, NOMBRE, APELLIDO, FECHA_ING, FECHA_NACIM,  
    TELEFONO, DIRECCION, BARRIO from b_empleados where  
    BARRIO = 'CENTRO' WITH CHECK OPTION)  
values ('3524612', 'JUAN', 'PORTILLO', SYSDATE,  
    to_date('02/08/86', 'dd/mm/yyyy'), '345-865', 'ESPAÑA  
345', 'CENTRO');
```

Inserción en múltiples tablas

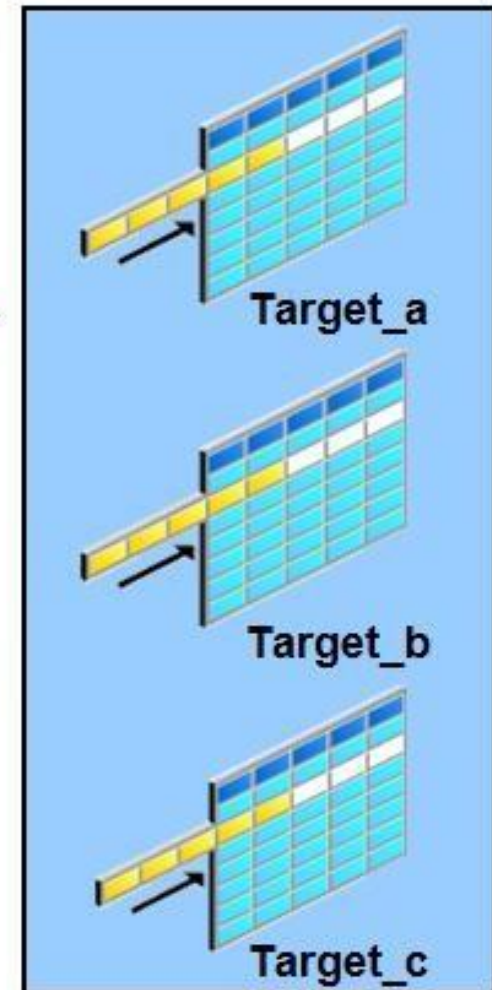
- Oracle permite insertar en una o mas tablas, filas devueltas como resultado de la evaluación de una subconsulta, mediante una sola instrucción `INSERT INTO ... SELECT`
- Esta instrucción puede ser condicional o no condicional
- Reduce los recorridos de tablas y el código PL/SQL necesarios para la realización de múltiples inserciones condicionales
- Su uso principal es para el proceso de ETL en almacenes de datos
- Proporcionan una mejora significativa del rendimiento en:
 - Sentencia única DML frente a diversas sentencias `INSERT...SELECT`
 - Una única sentencia DML frente a un procedimiento para realizar varias inserciones mediante la sintaxis `IF...THEN`

Visión General de Sentencias INSERT de Varias de Tablas



```
INSERT ALL  
  INTO target_a VALUES (...,...,...)  
  INTO target_b VALUES (...,...,...)  
  INTO target_c VALUES (...,...,...)  
SELECT ...  
FROM sourcetab  
WHERE ...;
```

Subconsulta



Tipos de sentencias INSERT en varias tablas

Los diferentes tipos de sentencia INSERT de varias tablas son:

- **INSERT incondicional:** Por cada fila devuelta por la subconsulta, se inserta una fila en cada una de las tablas de destino
- **INSERT ALL condicional:** por cada fila devuelta por la subconsulta, se inserta una fila en cada una de las tablas de destino si se cumple la condición especificada
- **INSERT FIRST condicional:** por cada fila devuelta por la subconsulta, se inserta una fila en la primera tabla de destino en la que se cumple la condición
- **INSERT mediante giro:** Es un caso especial de insert condicional

Insertión en múltiples tablas - Sintaxis

```
INSERT [ALL] [cláusula de insert condicional]  
[valores de la cláusula insert]  
(SUBQUERY)
```

En donde la cláusula de INSERT condicional puede ser
[ALL] [FIRST]
[WHEN condicion THEN] [valores de la cláusula INSERT]
[ELSE] [valores de la cláusula INSERT]

- Cuando se especifica **ALL**, entonces la BD evalúa cada cláusula WHEN, independientemente de las otras. Si se cumple, se ejecuta la cláusula INTO.
- Si se especifica **FIRST**, entonces la BD evalúa cada cláusula WHEN en el orden de aparición. Para el primer WHEN que se cumpla, se ejecuta la cláusula INTO, y los subsecuentes WHEN ya no son evaluados.

INSERT ALL Incondicional

```
CREATE TABLE CLIENTES_CANTIDADES (ID_CLIENTE  
NUMBER, CANTIDAD NUMBER);
```

```
CREATE TABLE CLIENTES_MONTOS (ID_CLIENTE  
NUMBER, MONTO NUMBER);
```

```
INSERT ALL  
  INTO clientes_cantidades VALUES (id_cliente,  
  cantidad)  
  INTO clientes_montos VALUES (id_cliente,  
  monto_total)  
SELECT id_cliente, count(*) cantidad,  
sum(monto_total) monto_total  
FROM b_ventas group by id_cliente;
```

INSERT ALL Incondicional

- Se denomina “incondicional” porque no se aplican más restricciones a las filas que las impuestas por la condición del select.
 - Todas las filas recuperadas se insertan en ambas tablas
 - La clausula VALUES en las sentencias insert especifican las columnas del select que se deben insertar en cada tabla
- Cada fila devuelta por el select da como resultado dos inserciones: una en clientes_cantidades, otra en clientes_montos

INSERT ALL Condicional

```
CREATE TABLE CLIENTES_MAYORISTAS (ID_CLIENTE NUMBER, MONTO NUMBER);  
CREATE TABLE CLIENTES_IDEN (ID_CLIENTE NUMBER, MONTO NUMBER);  
CREATE TABLE CLIENTES_MINORISTAS (ID_CLIENTE NUMBER, MONTO NUMBER);
```

INSERT ALL

```
WHEN monto_total < 1000000 THEN  
  INTO clientes_MINORISTAS
```

Dependiendo del monto de venta,
inserta en clientes_minoristas o
clientes_mayoristas

```
WHEN monto_total >= 1000000 THEN  
  INTO clientes_MAYORISTAS
```

```
WHEN id_cliente < 10 THEN  
  INTO clientes_IDEN
```

Si además el código es menor a 10,
inserta también en clientes_iden

```
SELECT id_cliente, sum(monto_total) monto_total  
  FROM b_ventas  
  group by id_cliente;
```

INSERT ALL Condicional

- A las filas se aplica sólo la condición de recuperación del SELECT
- Las filas se insertan en las tablas respectivas solo si cumplen la condición WHEN. Es posible que la misma fila se inserte en solo una o en varias tablas, dependiendo de si satisfacen o no dicha condición

INSERT FIRST Condicional

INSERT FIRST

WHEN id_cliente < 10 THEN

INTO clientes_IDEN

WHEN monto_total < 1000000 THEN

INTO clientes_MINORISTAS

WHEN monto_total >= 1000000 THEN

INTO clientes_MAYORISTAS

SELECT id_cliente, sum(monto_total) monto_total

FROM b_ventas

group by id_cliente;

Inserta en la tabla con la primera
condicion WHEN que se cumpla

INSERT FIRST Condicional

- Las filas recuperadas por el select se insertan en la primera tabla en la que se cumpla la condición WHEN. Cada fila recuperada por el select se inserta solo en una tabla.

INSERT mediante giro

- Convertir el juego de registros de ventas de la tabla de base de datos no relacional a un formato relacional.

Emp_ID	Week_ID	MON	TUES	WED	THUR	FRI
176	6	2000	3000	4000	5000	6000



Employee_ID	WEEK	SALES
176	6	2000
176	6	3000
176	6	4000
176	6	5000
176	6	6000

INSERT mediante giro o pivotante

```
INSERT ALL
  INTO sales_info VALUES (employee_id, week_id, sales_MON)
  INTO sales_info VALUES (employee_id, week_id, sales_TUE)
  INTO sales_info VALUES (employee_id, week_id, sales_WED)
  INTO sales_info VALUES (employee_id, week_id, sales_THUR)
  INTO sales_info VALUES (employee_id, week_id, sales_FRI)
SELECT EMPLOYEE_ID, week_id, sales_MON, sales_TUE,
       sales_WED, sales_THUR, sales_FRI
FROM sales_source_data;
```

- El giro es una operación de transformación de forma que cada registro de cualquier flujo de entrada, como una tabla no relacional, se convierta en registros para un entorno relacional.

Inserción en múltiples tablas - Restricciones

- Sólo se pueden ejecutar sentencias INSERT en múltiples TABLAS. La sentencia no admite vistas simples o materializadas.
- Tampoco se pueden ejecutar sentencias INSERT múltiples en tablas remotas.

UPDATE: Sintaxis

- Modifique filas existentes con la cláusula UPDATE

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

Actualizando Filas: Ejemplos

- Actualice a la localidad 6 a todas las personas de la localidad 2

```
UPDATE b_personas SET id_localidad = 6  
WHERE id_localidad = 2;
```

- Marque como proveedor y actualice a 22 la localidad de la persona con id=1

```
UPDATE b_personas  
SET es_proveedor='S', id_localidad='22'  
WHERE id = 1;
```

- Si no se especifica WHERE, todas las filas son actualizadas

Actualizando con el resultado del subquery

```
UPDATE B_EMPLEADOS  
SET CEDULA_JEFE = 1309873  
WHERE CEDULA IN  
      (SELECT CEDULA  
        FROM B_EMPLEADOS  
        WHERE CEDULA_JEFE=952160) ;
```


DELETE: Sintaxis

- Elimine filas existentes usando la orden DELETE.

```
DELETE [FROM] tabla  
[WHERE condición];
```

- Ejemplo: Elimine toda la información de empleados que empezaron antes del 1 de enero del 2001.

```
DELETE FROM b_personas  
WHERE fecha_ing<to_date('2001-01-01','yyyy-mm-dd')
```

- Si no especifica WHERE borra todas las filas!!!

MERGE: Sintaxis

- Permite actualizar los datos de una tabla combinándola con los datos de otra tabla

```
MERGE INTO <nombre de tabla>  
USING <vista o sentencia SELECT>  
ON (<condición>)  
WHEN MATCHED THEN <cláusula UPDATE>  
WHEN NOT MATCHED THEN <cláusula INSERT>;
```

MERGE

- Ejemplo: Creemos una tabla de bonificación por cada empleado con la siguiente estructura:

```
CREATE TABLE BONIFICACION  
(CEDULA_EMP      NUMBER(7) ,  
  BONIFICACION    NUMBER(10)) ;
```

- Suponiendo que la tabla contenga la bonificación correspondiente a cada empleado como un porcentaje del 5% sobre las ventas entonces podríamos hacer:

MERGE: Sintaxis

```
MERGE INTO bonificacion B
USING (SELECT cedula_vendedor cedula,
        SUM(monto_total) ventas
        FROM b_ventas
        GROUP by cedula_vendedor) E
ON (B.cedula_emp = E.cedula)
WHEN MATCHED THEN
    UPDATE SET B.bonificacion = ROUND(E.ventas * 1.05)
WHEN NOT MATCHED THEN
    INSERT (B.cedula_emp, B.bonificacion)
    VALUES (E.cedula, ROUND(E.ventas * 1.05));
```

Transacciones de Base de Datos

- **Transacción:** Es la unidad lógica de procesamiento de la Base de Datos.
- Se habla de “Unidad Lógica” porque representa un único proceso desde el punto de vista del usuario, pero puede implicar la ejecución de varias operaciones que efectúan un cambio consistente a los datos de la base
- Para delimitar la “unidad lógica”, debe establecerse un “inicio” y un “fin” de la transacción

Manipulación de Datos y Comandos de Control de Transacciones

COMANDO	DESCRIPCION
COMMIT	Finaliza la transacción actual haciendo que todos los cambios pendientes pasen a ser permanentes.
SAVEPOINT	Establece una "marca" dentro de la transacción en curso, usada por COMMIT o ROLLBACK.
ROLLBACK	Finaliza la transacción en curso descartando todos los cambios pendientes.

Transacciones de base de datos

- INICIO DE LA TRANSACCIÓN

Una transacción comienza cuando se ejecuta la primera sentencia SQL

- FIN DE LA TRANSACCIÓN

La transacción termina cuando sucede cualquiera de los siguientes eventos:

- La ejecución de un comando COMMIT o ROLLBACK
- Se ejecuta un comando DDL o DCL
- Existe una falla de la BD o del Sistema Operativo
- El usuario sale de su sesión SQL*Plus

ESTADO DE LAS TRANSACCIONES ANTES DE UN COMMIT o ROLLBACK

- Las operaciones que ejecuta el usuario se realizan en el buffer de la BD, por lo que la situación anterior puede ser recuperada
- La nueva situación puede ser visualizada únicamente por el usuario concurrente
- Los otros usuarios NO pueden ver los resultados, tampoco pueden realizar cambios en las filas afectadas, que para el efecto son 'bloqueadas' por el ORACLE

El comando COMMIT

- Confirma los cambios en la base de datos de modo permanente.
- Ejemplo: Cree el nuevo departamento «Contabilidad»

```
insert into b_areas(id, nombre_area, fecha_crea,  
activa,id_area_superior)  
values (50, 'Contabilidad', to_date(sysdate, 'dd/mm/yyyy'), 'S', NULL) ;
```

- Agregue a un empleado en el departamento y confirme los cambios.

```
INSERT INTO B_POSICION_ACTUAL  
VALUES (30, 1, 429987, 50, SYSDATE, NULL) ;  
COMMIT ;
```

El comando ROLLBACK

- Se descartan los cambios de los datos y el estado anterior de los datos se restaura
- Se liberan los llaveos en las filas afectadas
- Ejemplo:

```
DELETE FROM B_POSICION_ACTUAL;  
ROLLBACK;
```

El comando SAVEPOINT

- Establece un 'marcador' para controlar la reversión de los cambios
- Ejemplo: Cree una marca dentro de una transacción actual usando SAVEPOINT. Al hacer el rollback, la transacción se deshace hasta el último SAVEPOINT

```
SQL> UPDATE...  
SQL> SAVEPOINT update_done;  
Savepoint created.  
SQL> INSERT...  
SQL> ROLLBACK TO update_done;  
Rollback complete.
```