Capacitación ORACLE

Lenguaje SQL y Administración Básica

Gestión de Datos

Docentes: Ing. Juan Zaffaroni
Ing. María Cristina Chahin

AGENDA GENERAL MÓDULO 2: SQL Intermedio

- **Día 3**
  - **3- Consultas avanzadas**
    - **Consultas sobre varias tablas. Joins. Alias de tabla.**
    - **Funciones de varias filas**
    - **Agrupamiento y sumarización de datos**
    - **Exclusión de resultados de grupo**
- **Día 4**
  - **3- Consultas avanzadas (cont.)**
    - **Agregados al GROUP BY**
    - **Subconsultas**
- **Día 5**
  - **3- Consultas avanzadas (cont.)**
    - **Cláusula WITH**
- **Día 6**
  - **3- Consultas avanzadas (cont.)**
    - **Operadores de conjunto**

Capacitación ORACLE. Lenguaje SQL y Administración Básica

AGENDA GENERAL MÓDULO 2: SQL Intermedio (cont.)

- **Día 7**
  - **4- Manipulación de Datos**
    - **Inserción**
    - **Actualización**
    - **Merge**
    - **Borrado**
    - **Manejo de transacciones**
- **Día 8**
  - **5- SQL*Plus**
    - **Comandos varios**
    - **Personalización del entorno**
    - **Edición**
    - **Manipulación de archivos de comandos**
    - **Variables de sustitución**
    - **Ejecución**
- **Día 9**
  - **Evaluación Módulos 1 y 2**

3- Consultas avanzadas

- ✓Consultas sobre varias tablas. Joins. Alias de tabla.
- ✓Funciones de varias filas
- ✓Agrupamiento y sumarización de datos
- ✓Extensiones al GROUP BY
- ✓Subconsultas
- ✓Cláusula WITH
- ✓Operadores de conjunto

3- Consultas avanzadas

Introducción

• La sentencia SELECT nos brinda una herramienta para extraer información de una Base de Datos.

• Consultas avanzadas explica funcionalidades adicionales, que permiten combinar características simples, para obtener resultados más complejos. Estos son:

- **Join entre dos o más tablas**
- **Uso de alias de tabla**
- **Uso de funciones de varias filas**
- **Agregado y sumarización de datos**
- **Exclusión de resultados de grupo**
- **Sobreagregado de datos (mediante extensiones al GROUP BY)**
- **Combinación de varias sentencias SELECT en la forma de subconsultas (incluyendo cláusula WITH)**
- **Combinación de varias sentencias SELECT utilizando operadores de conjunto (Unión, Intersección, Diferencia)**

3- Consultas avanzadas: SELECT de varias tablas. Joins. Alias de Tabla.

Objetivo

- Recuperar información de varias tablas de la Base de Datos.
- Diferenciar los conceptos de:
  - *Producto Cartesiano*
  - *Inner join*
  - *Outer join*
  - *Auto join*
- Utilizar alias de tabla.

Ejemplo

➢ Generar un listado con: id de departamento, nombre de departamento, id de empleado, apellido y nombre de empleado, para los departamentos 110 y 190.

➢ Generar un listado con: apellido y nombre de cada empleado, y apellido y nombre de su jefe.

3- Consultas avanzadas: SELECT de varias tablas. Joins. Alias de Tabla.

Sintaxis (Oracle)

INNER JOIN

```
SELECT  t1.columna, t2.columna
FROM    tabla1 t1, tabla2 t2, …
[WHERE  t1.columna = t2.columna]
[AND    condición(es)];
```

Join entre n tablas: **mínimo n-1 condiciones de join**

Join de desigualdad: **BETWEEN … AND …, >=, <=**

OUTER JOIN

```
SELECT  t1.columna, t2.columna
FROM    tabla1 t1, tabla2 t2, …
[WHERE  t1.columna (+) = t2.columna];
SELECT  t1.columna, t2.columna
FROM    tabla1 t1, tabla2 t2, …
[WHERE  t1.columna = t2.columna (+)];
```


3- Consultas avanzadas: SELECT de varias tablas. Joins. Alias de Tabla.

Sintaxis (ANSI SQL 1999)

```
SELECT  t1.columna, t2.columna
FROM    tabla1 t1
[CROSS JOIN tabla2] |
[NATURAL JOIN tabla2] |
[JOIN tabla2 USING (nombre_columna)] |
[JOIN tabla2
  ON (tabla1.nombre_columna = tabla2.nombre_columna)] |
[LEFT | RIGHT | FULL OUTER JOIN tabla2
  ON (tabla1.nombre_columna = tabla2.nombre_columna)];
```

3- Consultas avanzadas: SELECT de varias tablas. Joins. Alias de Tabla.

Aplicación

- **SELECT department_id, department_name,**
  employee_id, last_name, first_name
  FROM employees e, departments d
  WHERE e.department_id (+) = d.department_id
  AND d.department_id in (110, 190);
- **SELECT department_id, department_name, employee_id,**
  last_name, first_name
  FROM employees e
  RIGHT OUTER JOIN departments d
  ON (e.department_id = d.department_id)
  WHERE d.department_id in (110, 190);
- **SELECT e.last_name, e.first_name, j.last_name, j.first_name**
  FROM employees e, employees j
  WHERE e.manager_id = j.employee_id;
- **SELECT e.last_name, e.first_name, j.last_name, j.first_name**
  FROM employees e
  JOIN employees j
  ON e.manager_id = j.employee_id;

3- Consultas avanzadas – Funciones de varias filas

Objetivo

- Utilizar funciones de varias filas para:
  - Operar sobre un juego de filas para proporcionar un *resultado único para el grupo*.
  - Operar sobre *todos* los valores de las columnas, o solamente considerar *no duplicados*.

Ejemplo

- Obtener la fecha de ingreso del empleado más nuevo de la empresa.
- Contar la cantidad de departamentos diferentes a los que pertenecen los empleados de la tabla EMPLOYEES.

Sintaxis

```
SELECT  funcion_de_grupo(columna), …
FROM    tabla
[WHERE  condición(es)];
```

Funciones de grupo: **avg, count, max, min, sum**

Aplicación

- **SELECT MAX(hire_date)**
  FROM employees;
- **SELECT COUNT(DISTINCT department_id)**
  FROM employees;

Objetivo

- Utilizar funciones de varias filas para:
  - Operar sobre un juego de filas para proporcionar un *resultado único para el grupo*.
  - Agregar y sumarizar los datos, subdividiendo los resultados en *grupos más pequeños.*
  - *Anidar* funciones de grupo.

Ejemplo

- Obtener la suma y promedio de los salarios de los empleados para cada departamento, indicando: departamento, suma, promedio.
- Repetir el listado anterior, pero desagregando por departamento y puesto.
- Obtener el promedio de las sumas de salarios por departamento, entre todos los departamentos de la empresa.

Sintaxis

```
SELECT    columna, funcion_de_grupo(columna), …
FROM      tabla
[WHERE    condición(es)]
[GROUP BY expresión_de_agrupamiento]
[ORDER BY columna];
```

Aplicación

```
SELECT      department_id, sum(salary) suma, avg(salary) promedio
FROM        employees
GROUP BY department_id;
```

```
SELECT      department_id, job_id,
            sum(salary) suma, avg(salary) promedio
FROM        employees
GROUP BY department_id, job_id;
```

```
SELECT      avg(sum(salary)) promedio
FROM        employees
GROUP BY department_id;
```


Objetivo

- Utilizar funciones de varias filas para:
  - Agregar y sumarizar los datos, subdividiendo los resultados en *grupos más pequeños.*
  - *Restringir* grupos que formarán parte del resultado.

Ejemplo

Obtener la suma y promedio de los salarios de los empleados para cada departamento, indicando: departamento, suma, promedio, pero informando sólo aquellos departamento cuya suma de salarios supere los $18.000.

Sintaxis

```
SELECT    columna, funcion_de_grupo(columna), …
FROM      tabla
[WHERE    condición(es)]
[GROUP BY expresión_de_agrupamiento]
[HAVING   condición_de_grupo]
[ORDER BY columna];
```


Aplicación

- **SELECT department_id, sum(salary) suma, avg(salary) promedio**
  FROM employees
  GROUP BY department_id
  HAVING sum(salary) > 18000;

Objetivo

- Utilizar extensiones al GROUP BY para:
  - Producir *subtotales*.
  - Producir valores de *tabulación cruzada* (todas las posibles combinaciones de agrupamiento).
  - *Identificar* valores *creados* por estas operaciones.
  - Producir un *único juego* de resultados.

Ejemplo

- Obtener la suma y promedio de los salarios de los empleados indicando: departamento, puesto, suma, promedio con totales para cada departamento y puesto, subtotales por departamento, y total general.
- Obtener la suma y promedio de los salarios de los empleados indicando: departamento, puesto, suma, promedio con totales para cada departamento y puesto, subtotales por departamento, subtotales por puesto y total general.
- Agregar una columna que indique el nivel de agregación.
- Agrupar con criterios específicos dentro de una única consulta.

Sintaxis

Generación de subtotales (de derecha a izquierda)

```
SELECT     columna, funcion_de_grupo(columna), …
FROM       tabla
[WHERE     condición(es)]
[GROUP BY [ROLLUP] expresión_de_agrupamiento]
[HAVING    condición_de_grupo
[ORDER BY columna];
```

Generación de valores de tabulación cruzada

```
SELECT     columna, funcion_de_grupo(columna), …
FROM       tabla
[WHERE     condición(es)]
[GROUP BY [CUBE] expresión_de_agrupamiento]
[HAVING    condición_de_grupo
[ORDER BY columna];
```

Sintaxis (cont.)

Identificación de valores nulos según nivel de agregación

```
SELECT     columna, funcion_de_grupo(columna), GROUPING(expresión), …
FROM       tabla
[WHERE     condición(es)]
[GROUP BY [ROLLUP] [CUBE] expresión_de_agrupamiento]
[HAVING    condición_de_grupo
[ORDER BY columna];
```

Conjuntos de agrupamiento

```
SELECT     columna, funcion_de_grupo(columna), …
FROM       tabla
[WHERE     condición(es)]
[GROUP BY [GROUPING SETS] (conjuntos_de_agrupamiento)]
[HAVING    condición_de_grupo
[ORDER BY columna];
```


Aplicación

- **SELECT department_id, job_id,**
  sum(salary) suma, avg(salary) promedio
  FROM employees
  GROUP BY ROLLUP (department_id, job_id);

- **SELECT department_id, job_id,**
  sum(salary) suma, avg(salary) promedio
  FROM employees
  GROUP BY CUBE (department_id, job_id);

Aplicación (cont.)

```
SELECT      department_id, job_id,
            sum(salary) suma, avg(salary) promedio,
            GROUPING(department_id) dept_grp,
            GROUPING(job_id) job_grp
FROM        employees
GROUP BY ROLLUP (department_id, job_id);
```

```
SELECT      department_id, job_id,
            sum(salary) suma, avg(salary) promedio,
            GROUPING(department_id) dept_grp,
            GROUPING(job_id) job_grp
FROM        employees
GROUP BY CUBE (department_id, job_id);
```


Aplicación (cont.)

```
SELECT      department_id, job_id,
            manager_id, avg(salary) promedio
FROM        employees
GROUP BY GROUPING SETS
            ((department_id, job_id), (job_id, manager_id));
```

```
SELECT      department_id, job_id,
            manager_id, avg(salary) promedio
FROM        employees
GROUP BY ROLLUP ((department_id, job_id), manager_id);
```

Objetivo

• Combinar más de una consulta, para resolver un problema que debe basarse en un resultado desconocido, que está en la base de datos o se puede obtener de ella.

Ejemplo

- Obtener el apellido y nombre de todos los empleados que tienen un salario mayor o igual que el de código 141, exceptuándolo.
- Obtener el apellido y nombre de todos los empleados que tienen un salario mayor o igual que alguno de los empleados del departamento 20, exceptuándolos.
- Obtener el apellido, salario y departamento de los empleados que ganan menos que el salario máximo de su departamento.

Sintaxis

Anidamiento en cláusula WHERE

```
SELECT     lista_select
FROM       tabla1
WHERE      expresión operador
           (SELECT   lista_select
            FROM     tabla2);
```

Anidamiento en cláusula HAVING

```
SELECT     lista_select
FROM       tabla1
GROUP BY   expresión_de_agrupamiento
HAVING     expresión operador
           (SELECT   lista_select
            FROM     tabla2);
```

Anidamiento en cláusula FROM

```
SELECT     lista_select
FROM      (SELECT   lista_select
           FROM     tabla)
[WHERE     condición(es)];
```

Subconsulta correlacionada

```
SELECT  lista_select
FROM    tabla1 principal
WHERE   columna1 operador
       (SELECT lista_select
        FROM   tabla2 secundaria
        WHERE  expresión1 =
               principal expresión2);
```

Operadores de comparación de una sola fila: **=, >, >=, <, <=, <>**

Operadores de comparación de una varias filas: **IN, ANY, ALL**

Operador **[NOT] EXISTS**

Aplicación

```
SELECT  last_name, first_name
FROM    employees
WHERE employee_id <> 141
AND     salary >=
        (SELECT salary
         FROM employees
         WHERE employee_id = 141);
```

```
SELECT  last_name, first_name
FROM    employees
WHERE department_id <> 20
AND     salary >= ANY
        (SELECT salary
         FROM employees
         WHERE department_id = 20);
```

```
SELECT a.last_name, a.salary, a.department_id
FROM   employees a, (SELECT     department_id, max(salary) maxsal
                     FROM       employees
                     GROUP BY department_id) b
WHERE a.department_id = b.department_id
AND     a.salary < b.maxsal;
```


Aplicación (cont.)

El último ejemplo, también se puede resolver con una subconsulta correlacionada.

```
SELECT a.last_name, a.salary, a.department_id
FROM   employees a,
WHERE salary <
      (SELECT     max(salary) maxsal
       FROM       employees
       WHERE    department_id = a.department_id);
```

Objetivo

• Utilizar un mismo bloque de consulta en una sentencia SELECT más compleja.

Ejemplo

➢ Obtener el nombre de departamento y salarios totales de los departamentos cuyos salarios totales sean mayores a sus salarios medios.

Sintaxis

```
WITH
nombre_bloque1 AS
(consulta 1)
[, nombre_bloque2 AS
(consulta 2) [, … ]]
SELECT    expresión
FROM      nombre_bloque1 [, nombre_bloque2]
          [tabla1]
[WHERE    condición(es)]
[GROUP BY expresión_de_agrupamiento]
[HAVING   condición_de_grupo]
[ORDER BY columna];
```

3- Consultas avanzadas - Cláusula WITH

Aplicación

La cláusula WITH sólo se puede usar con SELECT.

```
WITH
costo_depto AS (
    SELECT     d.department_name, SUM(e.salary) AS total_depto
    FROM       employees e, departments d
    WHERE      e.department_id = d.department_id
    GROUP BY d.department_id),
costo_promedio AS (
    SELECT     SUM(total_depto)/COUNT(*) AS prom_depto
    FROM       costo_depto)
SELECT *
FROM    costo_depto
WHERE total_depto >
        (SELECT     prom_depto
         FROM       costo_promedio)
ORDER BY department_name;
```


3- Consultas avanzadas – Operadores de conjunto

Objetivo

• *Combinar los resultados* de dos o más consultas, de forma tal de obtener un nuevo resultado que consista en la unión, intersección o diferencia de los conjuntos que se operan.

Ejemplo

- Obtener el código y puesto de los empleados, tanto los datos actuales como anteriores (consultando las tablas EMPLOYEES y JOB_HISTORY).
- Obtener los códigos de empleado que alguna vez cambiaron de puesto (figuran tanto en la tabla EMPLOYEES como en la JOB_HISTORY)
- Obtener los códigos de empleado que nunca antes habían tenido otro puesto (figuran en la tabla EMPLOYEES pero no en la JOB_HISTORY)

Sintaxis

```
consulta1
UNION / UNION ALL
consulta2;

consulta1
INTERSECT
consulta2;

consulta1
MINUS
consulta2;
```


Aplicación

- **SELECT employee_id, job_id**
  FROM employees
  UNION ALL
  SELECT employee_id, job_id
  FROM job_history;

- **SELECT employee_id**
  FROM employees
  INTERSECT
  SELECT employee_id
  FROM job_history;

- **SELECT employee_id**
  FROM employees
  MINUS
  SELECT employee_id
  FROM job_history;

Resumen

• **Mediante la sentencia SELECT podemos combinar funcionalidades para escribir consultas complejas que permitan:**

- **Recuperar información de *dos o más tablas* mediante el uso del *join*.**
- **Identificar brevemente las tablas involucradas en una consulta mediante el uso de *alias de tabla*.**
- **Agrupar y sumarizar los datos, utilizando *funciones de grupo*, agrupamiento en conjuntos más chicos (*group by*), y restricción sobre esos agrupamientos (*having*).**
- **Obtener *subtotales* sobre consultas agrupadas mediante las *extensiones al GROUP BY*.**
- **Combinar dos o más consultas, *anidándolas* para obtener una consulta compleja, que utilizando una *subconsulta* pueda basarse en un *valor desconocido*, obtenido también de la Base de Datos.**
- **Combinar dos o más consultas, enlazándolas mediante un *operador de conjunto* que permita obtener una *unión*, *intersección* o *diferencia* de conjuntos de resultados.**

4- Manipulación de Datos

- ✓Inserción de datos
- ✓Actualización de datos
- ✓Borrado de datos
- ✓Manejo de transacciones

Introducción

• Las sentencias DML permiten modificar la información contenida en las tablas de la Base de Datos.

• Manipulación de datos explica las operaciones que se pueden hacer para:
- **Insertar filas nuevas en una tabla (INSERT)**
- **Modificar datos de filas existentes (UPDATE)**
- **Combinar la actualización o inserción de filas (MERGE)**
- **Eliminar filas existentes de una tabla (DELETE)**

• Además, se explican las sentencias utilizadas para el manejo de transacciones:
- **Confirmando transacciones (COMMIT)**
- **Deshaciendo transacciones (ROLLBACK)**
- **Marcando puntos de transacción (SAVEPOINT)**

Objetivo

• Agregar nuevas filas a una tabla.
• Se agregan las filas *de a una*, cada vez que se ejecuta la sentencia.
• También se pueden *copiar masivamente*, filas de otra tabla.

Ejemplo

➢ Insertar una nueva fila en la tabla DEPARTMENTS, correspondiente al departamento 120, con nombre “Sistemas”, que actualmente no tiene designado gerente, en la ubicación 1400.
➢ Insertar en la tabla EMPLOYEES_2006 todas las filas que corresponden a los empleados contratados durante el año 2006.

4- Manipulación de datos - Inserción

Sintaxis

```
INSERT INTO tabla [(columna [, columna…])]
VALUES      (valor [, valor…]);
```

Esta sentencia inserta una fila cada vez.

```
INSERT INTO tabla [(columna [, columna…])]
subconsulta;
```

Esta sentencia inserta de una vez todas las filas que devuelve la subconsulta.

4- Manipulación de datos - Inserción

Aplicación

- **INSERT INTO departments**
  (department_id, department_name, location_id)
  VALUES (120, 'Sistemas', 1400);

- **INSERT INTO employees_2006**
  SELECT *
  FROM employees
  WHERE hire_date BETWEEN
  to_date('01/01/2006 00:00:00', 'dd/mm/yyyy hh24:mi:ss')
  AND to_date('31/12/2006 23:59:59', 'dd/mm/yyyy hh24:mi:ss');

Objetivo

- Modificar datos de filas existentes en una tabla.
- Se actualiza *más de una fila* a la vez.
- También se puede actualizar *asignando valores* obtenidos de una *subconsulta*.

Ejemplo

- Modificar los salarios de todos los empleados, incrementándolos en un 10%.
- Cambiar el departamento al que pertenece el empleado 201, pasandolo al departamento 10.
- Asignarle al empleado 149 el mismo departamento que el 102.

Sintaxis

```
UPDATE tabla
SET    columna = valor [, columna = valor, …]
WHERE  condición(es);
```

Esta sentencia actualiza de una vez todas las filas que cumplen las condiciones de la cláusula WHERE.

```
UPDATE tabla
SET    columna = (SELECT columna
                  FROM   tabla
                 [WHERE  condición])
[, columna = (SELECT columna
              FROM   tabla
             [WHERE  condición])]
[WHERE  condición(es)];
```

Esta sentencia actualiza asignando valores obtenidos en una subconsulta.

4- Manipulación de datos - Actualización

Aplicación

- **UPDATE employees**
  SET salary = salary * 1.1;

- **UPDATE employees**
  SET department_id = 10
  WHERE employee_id = 201;

- **UPDATE employees**
  SET department_id = (SELECT department_id
  FROM employees
  WHERE employee_id = 102)
  WHERE employee_id = 149;

4- Manipulación de datos – Sentencia MERGE

Objetivo

• Actualizar o insertar datos en forma condicional en una tabla, realizándolo en una *única sentencia*.

Ejemplo

- Insertar en la tabla copy_emp los datos que no existan de los empleados desde la tabla employees, en base al employee_id. En caso de que el empleado exista, actualizar las columnas: first_name, last_name, email, phone_number, hire_date, job_id, salary, commission_pct, manager_id, department_id.

4- Manipulación de datos – Sentencia MERGE

Sintaxis

```
MERGE INTO tabla alias
   USING (tabla|vista|subconsulta) alias
   ON (condición de join)
   WHEN MATCHED THEN
     UPDATE SET
     columna1 = valor1
     [, columna2 = valor2 …]
   WHEN NOT MATCHED THEN
     INSERT (lista de columnas)
     VALUES (lista de valores);
```

Esta sentencia verifica en la tabla de cláusula INTO la existencia de los datos que se encuentran en la tabla de cláusula USING, en base a la condición de cláusula ON. Si los datos existen, se ejecuta el update de la cláusula WHEN MATCHED THEN. Si no existen, se ejecuta el insert de la cláusula WHEN NOT MATCHED THEN.

4- Manipulación de datos – Sentencia MERGE

Aplicación

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

Objetivo

- Eliminar filas existentes en una tabla.
- Se elimina *más de una fila* a la vez.

Ejemplo

- Eliminar la fila del empleado 102.
- Eliminar las filas de los empleados del departamento “Marketing”.

Sintaxis

```
DELETE [FROM] tabla
[WHERE  condición(es)];
```

Esta sentencia elimina de una vez todas las filas que cumplen las condiciones de la cláusula WHERE.

```
DELETE FROM tabla
WHERE       columna = (SELECT columna
                       FROM   tabla
                      [WHERE  condición])
[AND        condición(es)];
```

Esta sentencia elimina las filas que cumplen la condición, seleccionando de otra tabla.

4- Manipulación de datos - Borrado

Aplicación

- **DELETE FROM employees**
  WHERE employee_id = 102;

- **DELETE FROM employees**
  WHERE department_id =
  (SELECT department_id
  FROM departments
  WHERE department_name = 'Marketing');

4- Manipulación de datos – Manejo de transacciones

Objetivo

• Validar o deshacer la acción de las sentencias contenidas en una transacción.
• Reconocer el inicio y fin de una transacción.

Ejemplo

- Vaciar la tabla DEPARTMENTS.
- Comprobar con una consulta, que está vacía.
- Deshacer la transacción.
- Comprobar con una consulta, que aún tiene datos.
- Insertar una nueva fila en LOCATIONS.
- Marcar este punto en la transacción.
- Vaciar la tabla DEPARTMENTS.
- Deshacer la transacción hasta el punto marcado.
- Validar la transacción.
- Comprobar los datos con una consulta (Departments debe tener datos, y Locations debe tener la fila nueva).

Sintaxis

```
COMMIT;
ROLLBACK;
SAVEPOINT nombre;
ROLLBACK TO SAVEPOINT nombre;
```

Una transacción se valida el emitir una sentencia DDL o confirmarla explícitamente (COMMIT).

Aplicación

- **DELETE FROM departments;**
- **SELECT * FROM departments;**
- **ROLLBACK;**
- **SELECT * FROM departments;**
- **INSERT INTO locations**
  VALUES (1000, null, 1000, 'Buenos Aires', 'CABA', 'BA');
- **SAVEPOINT A;**
- **DELETE FROM departments;**
- **ROLLBACK TO SAVEPOINT A;**
- **COMMIT;**
- **SELECT * FROM departments;**
- **SELECT * FROM locations;**

Resumen

• **Mediante las sentencias DML podemos modificar la información contenida en las tablas de la Base de Datos.**

- **Insertar filas nuevas mediante el *insert*.**
- **Modificar valores en filas existentes mediante el *update*.**
- **Combinar la modificación o inserción de filas mediante el *merge*.**
- **Eliminar filas existentes mediante el *delete*.**

• **Además, mediante el control de transacciones, podemos validar o deshacer los cambios efectuados en sentencias DML.**

- **Validar transacciones mediante el *commit*.**
- **Deshacer transacciones mediante el *rollback*.**
- **Establecer puntos de control mediante el *savepoint*.**

5- SQL*Plus

✓Formateo de salida

✓Variables de comando SET

✓Variables de sustitución

✓Creación y ejecución de archivos de comandos

Introducción

• **La utilización de SQL*Plus permite la comunicación con la Base de Datos, porque reconoce y ejecuta sentencias SQL (estándar) y contiene su propio lenguaje de comandos (propietario de Oracle).**

• **SQL*Plus explica la forma de realizar:**
  - **Personalización del entorno**
  - **Creación y ejecución de archivos de comandos**
  - **Creación de consultas que requieran variables de sustitución**
  - **Formateo de salida de los scripts**

Categorías de comandos

```
Varios (conexión, descripción de tablas, etc.)
Entorno
Edición
Manipulación de archivos
Interacción
Ejecución
```

SQL*Plus reconoce las sentencias y las envía al servidor.
Las palabras clave se pueden abreviar.
Los comandos no realizan manipulación de datos en la base.
No necesita carácter de terminación.

Objetivo

- Conectarse a SQL*Plus.
- Visualizar la estructura de tablas, vistas o sinónimos.
- Salir de SQL*Plus.

Ejemplo

- Conectarse a SQL*Plus mediante la línea de comandos, con usuario *admin*, password *adm01* a la base *desa*.
- Obtener una descripción de las columnas que componen la tabla *employees*.
- Salir de SQL*Plus.

Sintaxis

- **`sqlplus [usuario [/password [@base]]]`**
- **`DESC[RIBE] nombre_tabla|nombre_vista|nombre_sinonimo`**
- **`EXIT`**

5- SQL*Plus – Comandos varios

Aplicación

- **sqlplus admin/adm01@desa**
- **DESC employees**
- **EXIT**

5- SQL*Plus – Personalización del Entorno

Objetivo

- Afectar el comportamiento de la sesión actual de SQL*Plus.
- Visualizar la configuración del entorno de SQL*Plus.

Ejemplo

- Definir que no se muestren los títulos de las columnas.
- Verificar si está activada la visualización de los títulos.

Sintaxis

- **SET variable valor**
- **SHOW variable**

COLSEP { _ | texto }	Define el carácter separador entre columnas.
ECHO { ON | OFF }	Controla si la ejecución de comandos lista cada comando a medida que se ejecutan.
FEED[BACK] { 6 | n | OFF | ON }	Define que se muestre el nro. de registros devueltos cuando se selecciona por lo menos ese nro. de registros.
HEA[DING] { OFF | ON }	Controla si se imprimen los títulos de las columnas.
LIN[ESIZE] { 80 | n }	Define el nro. de caracteres que se muestran en cada línea.
LONG { 80 | n }	Define el ancho máximo en bytes para mostrar valores de tipo CLOB, LONG, etc.
NULL texto	Define el texto que representa un valor nulo en el resultado de un comando SELECT.
PAGES[IZE] { 24 | n }	Define el nro. de líneas de cada página.

Sintaxis (cont.)

SERVEROUT[PUT] { ON | OFF } [SIZE n]	Controla si debe visualizar la salida de los stored procedures o bloques PL/SQL. SIZE define el nro. de bytes de la salida que se pueden almacenar temporalmente. El valor por defecto es 2.000. n debe estar entre 2.000 y 1.000.000.
TERM[OUT] { ON | OFF }	Controla la visualización de la salida de comandos ejecutados desde un script a pantalla.
TI[ME] { ON | OFF }	Controla la visualización de la hora actual.
TIMI[NG] { ON | OFF }	Controla la visualización de estadísticas de tiempo.
TRIM[OUT] { ON | OFF }	Determina si SQL*Plus elimina blancos al final de cada línea mostrada. No afecta a la salida a archivos.
WRA[P] { ON | OFF }	Controla si debe alinear la visualización de una fila a la siguiente línea, si es demasiado larga para el ancho actual de línea. OFF trunca la fila.

5- SQL*Plus – Personalización del Entorno

Aplicación

- **SET HEADING OFF**
- **SHOW HEADING**

5- SQL*Plus – Edición

Objetivo

- Mostrar el buffer de SQL.
- Ejecutar sentencias en el buffer.

Ejemplo

- Mostrar las primeras 5 líneas del buffer de SQL.
- Mostrar y ejecutar la sentencia que actualmente está en el buffer de SQL.

5- SQL*Plus – Edición

Sintaxis

- **L[IST]**
- **L[IST] n**
- **L[IST] m n**
- **R[UN]**

5- SQL*Plus – Edición

Aplicación

- **LIST 5**
- **R**

Objetivo

- Guardar el buffer de SQL.
- Recuperar sentencias al buffer de SQL.
- Ejecutar archivos de comandos
- Editar el contenido del buffer o de un archivo de comandos.
- Almacenar resultados en archivos.

Ejemplo

- Guardar el buffer actual de SQL en el archivo script1.txt.
- Cargar en el buffer de SQL el contenido del archivo script2.sql.
- Ejecutar las sentencias guardadas en el archivo de comandos ajuste.sql.
- Editar el contenido del buffer de SQL.
- Definir el archivo salida.dat para almacenar el resultado de las consultas. Luego, cerrar el archivo.

Sintaxis

- **S[AVE] nombre_archivo[.extensión] [REP[LACE]|APP[END]]**
- **GET nombre_archivo[.extensión]**
- **STA[RT] nombre_archivo [.extensión]**
- **@nombre_archivo[.extensión]**
- **ED[IT] [nombre_archivo[.extensión]]**
- **SPO[OL] [nombre_archivo[.extensión]|OFF]**

La extensión por defecto de los archivos es .sql.

5- SQL*Plus – Manipulación de archivos

Aplicación

- **S script1.txt**
- **GET script2**
- **START ajuste**
- **@ajuste.sql**
- **ED**
- **SPOOL salida.dat … SPOOL OFF**

5- SQL*Plus – Interacción

Objetivo

• Ejecutar comandos que soliciten al usuario los valores para sustituir ciertas variables.

Ejemplo

- Activar la visualización de las variables de sustitución.
- Obtener apellido y nombre del empleado cuyo código se ingresa cada vez.
- Obtener los códigos de todos los empleados que cumplan con la condición solicitada.
- Definir la variable puesto, con el valor inicial 'SA_REP'.
- Utilizando la variable puesto, obtener código y apellido de los empleados con ese código de puesto.
- Indefinir la variable puesto.
- Desactivar la visualización de las variables de sustitución.

Sintaxis

- ***&variable***
- ***&&variable***
- **DEFINE**
- **UNDEFINE**
- **SET VERIFY {ON|OFF}**

Las variables de sustitución se pueden embeber dentro de archivos de comandos, o de una única sentencia SQL.
& utiliza la variable. Si no está definida, la solicita al usuario por cada comando ejecutado.
&& reutiliza la variable una vez que se ingresa.
Si el valor de la variable es de fecha o carácter, y entonces debiera llevar comillas, encerrar la expresión &variable entre comillas.
Se puede utilizar una variable de sustitución en cualquier lugar de la sentencia select, excepto como primera palabra.

Aplicación

- **SET VERIFY ON**
- **SELECT last_name, first_name**
  FROM employees
  WHERE employee_id = &id;
- **SELECT employee_id**
  FROM employees
  WHERE &condicion;
- **DEFINE puesto = 'SA_REP'**
- **SELECT employee_id, last_name**
  FROM employees
  WHERE job_id = &puesto;
- **UNDEFINE puesto**
- **SET VERIFY OFF**

Objetivo

• Ejecutar un archivo de comandos desde SQL*Plus.

Instrucciones

- Crear y probar la sentencia SQL.
- Guardarla en un archivo de comandos.
- Editar el archivo de comandos para agregar formatos de SQL*Plus antes de la sentencia SQL. No colocar comandos SQL*Plus dentro de la sentencia SQL.
- Verificar que la sentencia SQL va seguida de un carácter de ejecución (";" ó "/")
- Agregar comandos para deshacer los formatos luego de la sentencia SQL.
- Guardar el archivo de comandos.
- Ejecutarlo desde SQL*Plus mediante @comandos.sql.

Sintaxis

- **`SET variable valor`**

ARRAY[SIZE] { 20 | n }	Número de filas que SQL*Plus recupera a la vez.
DEF[INE] { & | carácter | ON | OFF }	Define el carácter utilizado para prefijar las variables de sustitución.
TRIMS[POOL] { ON | OFF }	Determina si SQL*Plus elimina blancos al final de cada línea escrita en archivo. No afecta a la salida por pantalla.

- **`SHO[W] { variable | opción }`**

ALL	Lista la configuración de todas las opciones de SHOW.
ERR[ORS]	Muestra errores de compilación (PL/SQL).
PARAMETERS [nombre]	Muestra el valor actual para uno o todos los parámetros de inicialización.
REL[EASE]	Muestra el nro. de release de Oracle.
SPOO[L]	Indica si la salida se está redireccionando.
SQLCODE	Muestra el código de retorno de SQL de la última operación
USER	Muestra el nombre de usuario conectado.

- **`PRO[MPT] [ texto ]`**

5- SQL*Plus – Ejecución

Aplicación

```
SET FEEDBACK OFF
SET HEADING OFF
SET LINESIZE 240
SET PAGESIZE 0
SET TERMOUT OFF
SET TIMING ON
SET WRAP OFF
SPOOL salida.dat
PROMPT
PROMPT Usuario conectado
PROMPT
SHOW USER
PROMPT
@script
SPOOL OFF
SET TIMING OFF
SET TERMOUT ON
SET PAGESIZE 24
SET LINESIZE 80
SET HEADING ON
SET FEEDBACK 6
```


5- SQL*Plus

Resumen

• Mediante SQL*Plus podemos:

- *Conectarnos* a la base de datos desde Windows o la interfaz de comandos.
- Visualizar la *estructura de tablas*.
- *Personalizar* el comportamiento de la *sesión de SQL*Plus*.
- *Editar* y *ejecutar* el contenido del *buffer* de SQL.
- *Guardar* el contenido del *buffer* de SQL.
- *Recuperar el contenido* de un archivo de comandos.
- *Ejecutar el contenido* de un archivo de comandos.
- *Editar el contenido* del buffer o de un archivo de comandos.
- *Almacenar el resultado* de la ejecución en un archivo.
- Escribir comandos que soliciten al usuario el contenido de *variables de sustitución*.
- *Ejecutar* sentencias SQL mediante *archivos de comandos*.

Capacitación ORACLE. Lenguaje SQL y Administración Básica

EVALUACIÓN MÓDULOS 1 y 2