Lenguaje de Manipulación de Datos

DML

OPERACIONES BÁSICAS

Comandos que se aplican a una tabla

- **INSERT** para agregar filas a una tabla,
- **UPDATE** para modificar filas de una tabla,
- **DELETE** para borrar filas de una tabla

Comandos que pueden aplicarse a una o más tablas

- **SELECT FROM**

Inserción de Registros

INSERT INTO nombre_tabla [(columnas)]

{VALUES({v1|DEFAULT|NULL}, ..., {vn/DEFAULT/NULL})

INSERT INTO nombre_tabla [(columnas)]

{clausula SELECT};

En el caso de utilizar SELECT se produce una tabla temporal. El SELECT se evalúa antes que inicie la operación INSERT.

MODIFICACIÓN REGISTROS

UPDATE nombre_tabla
SET columna = {expresión|DEFAULT|NULL}
[, columna = {expr|DEFAULT|NULL} ...]
WHERE condiciones;

Borrado de Registros

DELETE FROM nombre_tabla
[**WHERE** condiciones];

CONSULTAS SIMPLES

```
SELECT  [DISTINCT]
*|
nombre_columna_a_seleccionar1 [[AS] alias_col1],
nombre_columna_a_seleccionar 2[[AS] alias_col2]...
nombre_columna_a_seleccionar n[[AS] alias_coln]

FROM tabla_a_consultar [[AS] alias_tabla]

WHERE condiciones;
```

EJEMPLO: CONSULTA SIMPLE ALIAS

Listar los números de CUIT y la Razón Social de los Clientes y Proveedores que tiene registrada la Ferretería.

```
SELECT per.`Cuit`, per.`Razon_Social`
FROM personas per
```

Se ha utilizado el alias per para la tabla personas

EJEMPLO: CONSULTA SIMPLE CON WHERE

Listar las localidades que pertenezcan a la provincia de Santa Fe (cuyo código es 1)

```
SELECT loc.`Cod_Postal`, loc.`ciudad`
FROM localidades loc
WHERE loc.`Id_Pcia` = 1
```

PALABRAS Y SÍMBOLOS PARA CONSULTAS

- **Las expresiones de valores**
  - Suma (+)
  - Resta (-)
  - Multiplicación (*)
  - División (%)
- **Conectores Lógicos**
  - AND OR NOT
- **Predicados**
  - Comparación: (=, <>, <, >, <=, =>)
  - Entre (...BETWEEN...AND...)
  - IN, (NOT IN)
  - LIKE
  - NULL
  - Cuantificador (ALL, SOME, ANY)
  - EXISTS, (NOT EXISTS)

BETWEEN

Para expresar una condición que quiere encontrar un valor entre límites concretos

SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE **columna BETWEEN límite1 AND límite2**;

```
SELECT loc.`Cod_Postal`, loc.`ciudad`
FROM localidades loc
WHERE loc.`Id_Pcia` BETWEEN 1 AND 3
```

LIKE

Para comprobar si una columna de tipo carácter cumple alguna propiedad determinada

SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE **columna LIKE característica**;

```
SELECT *
FROM provincias pro
WHERE pro.`Nom_pcia` LIKE '%sa%'
```

Recuerden probar combinaciones diferentes!!!

IN

Para obtener los registros donde el valor de un atributo coincida con alguno de los elementos de una lista utilizamos **IN**, y para ver que no coincide con ninguno, **NOT IN**:

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna [NOT] IN (valor1, ..., valorN);
```

```
SELECT pro.`Id_Pcia`,pro.`Nom_pcia`
FROM provincias pro
WHERE pro.`Id_Pcia` IN (1,3)
```

IS NULL

Cuando deseamos comprobar si un valor es nulo utilizaremos **IS NULL**, y para averiguar si no lo es, **IS NOT NULL**.

```
SELECT nombre_columnas_a_seleccionar
FROM tabla_a_consultar
WHERE columna IS [NOT] NULL;
```

```
SELECT *
FROM personas per
WHERE per.`Direc_WEB` IS NOT NULL
```

SUBCONSULTAS

Una subconsulta es una consulta incluida dentro de una cláusula WHERE o HAVING de otra consulta.

En ocasiones, para expresar ciertas condiciones no hay otra alternativa que obtener el valor que buscamos como resultado de una consulta.

```
SELECT  nombre_columnas_a seleccionar
FROM tabla_a_consultar
WHERE columna operador_comparación subconsulta;
```

EJEMPLO: SUBCONSULTAS - IN

Listar las localidades(Código Postal y Ciudad) de aquellas Localidades donde no haya Clientes ni Proveedores registrados.

```
SELECT loc.`Cod_Postal`, loc.`ciudad`
FROM localidades loc
WHERE loc.cod_postal
        NOT IN (SELECT per.`Cod_Postal`
                FROM personas per)
```

ANY/SOME - ALL

Para ver si un atributo cumple que:

- (ALL) todas sus filas
- (ANY/SOME) algunas de sus filas

satisfacen una condición, podemos hacer:

SELECT nombre_columnas_a seleccionar
FROM tabla_a_consultar
WHERE columna operador_comparación
{ALL|ANY|SOME} subconsulta;

EJEMPLO: SUBCONSULTAS –ALL – ANY/SOME

Listar los productos que tengan el mayor stock

```
SELECT prod.`ID_Producto`,
       prod.`Nombre_producto`, prod.`Stock`
FROM productos prod
WHERE prod.`Stock` >= ALL (SELECT
       prod.`Stock` FROM productos prod)
```

El complemento de la consulta anterior:

```
SELECT prod.`ID_Producto`,
       prod.` Nombre_producto `, prod.`Stock`
FROM productos prod
WHERE prod.`Stock` < ANY (SELECT
       prod.`Stock` FROM productos prod)
```

EXISTS

Para comprobar si una subconsulta produce alguna fila, podemos utilizar esta sentencia denominada test de existencia **EXISTS**

Para comprobar si una subconsulta no produce ninguna fila, podemos utilizar **NOT EXISTS**.

SELECT nombre_columnas_a_seleccionar

FROM tabla_a_consultar

WHERE [NOT] EXISTS subconsulta;

EJEMPLO: SUBCONSULTAS - EXIST

Listar los pedidos donde se hayan solicitado 10 o más productos en cualquiera de los detalles del mismo.

```
SELECT ped.`ID_Pedido`
FROM pedidos ped
WHERE
    EXISTS
    (SELECT *
     FROM detalle_pedidos det
      WHERE det.`ID_Pedido`= ped.`ID_Pedido`
       AND det.`Cantidad` >= 10)
```

RESULTADOS DE EXISTS - NOT EXISTS - IN

Las cláusulas EXISTS o IN se pueden emplear para generar la **intersección** entre dos consultas

Las cláusulas NOT EXISTS o NOT IN para generar la **diferencia** entre consultas.

Pero existen además cláusulas específicas para obtener operaciones de conjuntos…

INTERSECT

UNION

EXCEPT

EJEMPLO: UNION

Lista de números de CUIT DE personas que hayan tenido actividad en la empresa: Clientes con Pedidos y Proveedores con Productos.

```
SELECT ped.`Cuit`
FROM pedidos ped
UNION
SELECT prodp.`Cuit`
FROM `productos_proveedores` prodp
```

Cómo Combinar tablas

Cuando requerimos datos de tablas relacionadas necesitaremos enlazar las mismas. Podemos hacerlo a través del uso de:

- **PRODUCTO CARTESIANO**
- **NATURAL JOIN**
- **INNER JOIN**
- **LEFT JOIN - RIGHT JOIN**
- **SELF JOIN**

EJEMPLO DE PRODUCTO CARTESIANO *LOCALIDADES*

```
SELECT loc.`Cod_Postal`, loc.`ciudad`,
pro.`Nom_pcia`
FROM `localidades` loc, provincias pro
WHERE loc.`Id_Pcia` = pro.`Id_Pcia`
```

EJEMPLO DE NATURAL JOIN *LOCALIDADES*

```
SELECT loc.`Cod_Postal`, loc.`ciudad`,
pro.`Nom_pcia`
FROM `localidades` loc NATURAL JOIN
provincias pro
```

EJEMPLO DE INNER JOIN *PERSONAS*

Listar los números de CUIT, Razón Social, Dirección, código postal, Nombre de Localidad y Nombre de provincia de los Clientes y Proveedores que tiene registrada la Ferretería y pertenecen a la provincia de Santa Fe (cuyo código es 1)

```
SELECT per.`Cuit`, per.`Razon_Social`,
            per.`Direccion`,loc.`ciudad`,pro.`Nom_pcia
            ` FROM personas per
INNER JOIN localidades loc
ON per.`Cod_Postal` = loc.`Cod_Postal`
INNER JOIN provincias pro
ON pro.`Id_Pcia` = loc.`Id_Pcia`
WHERE pro.`Id_Pcia` = 1
```

RESULTADOS AL UTILIZAR INNER JOIN

INNER JOIN

es una combinación por equivalencia, conocida también como unión interna.

Las combinaciones equivalentes son las más comunes; éstas combinan los registros de dos tablas **siempre que haya concordancia de valores en atributos comunes a ambas tablas**

RESULTADOS DEL EJEMPLO INNER JOIN *PERSONAS*

En el ejemplo anterior las tablas están relacionadas a través de un atributo:

Las provincias tienen Id_Pcia como clave primaria

Las localidades tienen Cod_postal como clave primaria y están relacionadas con la tabla provincias a través del **Id_Pcia** (clave foránea a la tabla provincias)

Las personas tienen CUIT como clave primaria y están relacionadas con la tabla localidades a través del **Cod_Postal**

En las consultas de este tipo basta con poner en la cláusula ON el atributo que vincula las tablas

EJEMPLO DE INNER JOIN: *PRECIOS*

Si deseáramos obtener los productos que vende la ferretería, los proveedores a los que los compra y sus precios, la consulta debería hacerse:

```
SELECT prod.`ID_Producto`,
       prod.`Nombre_producto`, prop.`Cuit`,
       per.`Razon_Social`, prec.`Fecha_Desde`,
       prec.`Precio`
FROM productos prod
INNER JOIN productos_proveedores prop
ON prop.`ID_Producto` = prod.`ID_Producto`
INNER JOIN personas per
ON per.`Cuit` = prop.`Cuit`
INNER JOIN `precios` prec
ON prop.`Cuit` = prec.`Cuit`
AND prop.`ID_Producto` = prec.`ID_Producto`
```

RESULTADOS EJEMPLO INNER JOIN *PRECIOS*

En el ejemplo de precios las tablas están relacionadas de la siguiente forma

La tabla productos tiene Id_prod como clave primaria

La tabla personas tiene CUIT como clave primaria

La tabla productos_proveedor tiene como clave primaria Id_prod + CUIT y está relacionada con:

- productos a través de **Id_prod** (clave foránea a productos)
- Personas a través de **CUIT** (clave foránea a personas)

Los precios están relacionados a productos_proveedor a través de una clave foránea que combina 2 atributos

Id_prod + CUIT

En las consultas de este tipo debemos poner en la cláusula ON todos los atributos que vinculan las tablas utilizando AND

EJEMPLO DE CONSULTA LEFT JOIN *PRECIOS*

El problema con la consulta anterior es que no aparecen aquellos productos de os proveedores para los que no se tienen precios. Para resolverlo utilizamos LEFT JOIN:

```
SELECT prod.`ID_Producto`,
    prod.`Nombre_producto `, prop.`Cuit`,
    per.`Razon_Social`, prec.`Fecha_Desde`,
    prec.`Precio`
FROM productos prod
INNER JOIN productos_proveedores prop
ON prop.`ID_Producto` = prod.`ID_Producto`
INNER JOIN personas per
ON per.`Cuit` = prop.`Cuit`
LEFT JOIN `precios` prec
ON prop.`Cuit` = prec.`Cuit`
AND prop.`ID_Producto` = prec.`ID_Producto`
```

RESULTADOS AL UTILIZAR LEFT JOIN

LEFT JOIN

Como vimos la sintaxis de la sentencia LEFT JOIN es idéntica a la del INNER JOIN pero produce un resultado diferente.

El LEFT JOIN permite que aparezcan en la consulta todos los registros de la tabla de la izquierda aunque los mismos **no tengan ninguna correspondencia** con la tabla con la que se combinan, completando los atributos que se muestran de la tabla de la derecha con **valores NULL.**

RESULTADOS AL UTILIZAR LEFT JOIN

Esto nos puede servir para realizar comparaciones y extraer ,por ejemplo, todos los productos de los proveedores para los que no tenemos registrados precios:

```
SELECT prod.`ID_Producto`, prod.`Descripc_Prod`,
       prop.`Cuit`, per.`Razon_Social`, prec.`Fecha_Desde`,
       prec.`Precio`
FROM productos prod
INNER JOIN productos_proveedores prop
ON prop.`ID_Producto` = prod.`ID_Producto`
INNER JOIN personas per
ON per.`Cuit` = prop.`Cuit`
LEFT JOIN `precios` prec
ON prop.`Cuit` = prec.`Cuit`
AND prop.`ID_Producto` = prec.`ID_Producto`
WHERE prec.`Precio` IS NULL
```

RESULTADOS AL UTILIZAR LEFT JOIN

Además, en este ejemplo se han anidado INNER JOIN con LEFT JOIN

LEFT JOIN o RIGHT JOIN pueden anidarse dentro de INNER JOIN, pero INNER JOIN no puede anidarse dentro de LEFT JOIN o RIGHT JOIN.

Probemos con el ejemplo anterior qué sucede al anidar LEFT e INNER…

ORDEN DE LAS FILAS EN UNA CONSULTA

Si se necesita que las filas aparezcan en un orden determinado se deberá utilizar la cláusula **ORDER BY** en la sentencia SELECT

SELECT nombre_columnas_a seleccionar
FROM tabla_a_consultar
[WHERE condiciones]
ORDER BY columna_orden1 [DESC]
[, columna_orden2 [DESC]…
, columna_ordenn [DESC]…] };

EJEMPLO DE CONSULTA ORDER BY

Listar los números de CUIT, Razón Social, Dirección, código postal, Nombre de Localidad y Nombre de la provincia de los Clientes y Proveedores que tiene registrada la Ferretería ordenados por provincia y por localidad

```
SELECT per.`Cuit`, per.`Razon_Social`,
       per.`Direccion`,loc.`ciudad`,
       pro.`Nom_pcia`
FROM personas per
INNER JOIN localidades loc
ON per.`Cod_Postal` = loc.`Cod_Postal`
INNER JOIN provincias pro
ON pro.`Id_Pcia` = loc.`Id_Pcia`
ORDER BY pro.`Nom_pcia`, loc.`ciudad`
```

Funciones de Agregación

Función	Descripción
COUNT	Nos da el número total de filas seleccionadas
SUM	Suma los valores de una columna
MIN	Nos da el valor mínimo de una columna
MAX	Nos da el valor máximo de una columna
AVG	Calcula el valor promedio de una columna

Ejemplos de Agregación

Cantidad de personas registradas:

```
SELECT COUNT(*)
FROM personas per
```

Cantidad de productos total en stock:

```
SELECT SUM(prod.`Stock`)
FROM productos prod
```

Máximo número de Pedido registrado:

```
SELECT MAX(ped.`ID_Pedido`)
FROM pedidos ped
```

FUNCIONES DE AGRUPACIÓN

Es probable que necesitemos realizar las consultas anteriores pero para determinados valores de un atributo.

Para ello deberemos recurrir a las

funciones de AGRUPACIÓN:

- **GROUP BY** nos sirve para agrupar
- **HAVING** especifica condiciones de búsqueda para grupos de filas; lleva a cabo la misma función que cumple la cláusula WHERE para las filas de toda la tabla, pero ésta aplica las condiciones a los grupos obtenidos.

Ejemplos de Agrupación

Cantidad de personas registradas por localidad:

```
SELECT per.`Cod_Postal`, COUNT(*)
FROM personas per
GROUP BY per.`Cod_Postal`
```

EJEMPLOS DE AGRUPACIÓN

Cantidad de detalles de pedidos por producto de cada proveedor:

```
SELECT  det.`Cuit`,
        det.`ID_Producto`,prod.`Descripc_Prod`,
        SUM(det.`Cantidad`)
FROM `detalle_pedidos` det
INNER JOIN productos prod
ON prod.`ID_Producto` = det.`ID_Producto`
INNER JOIN personas per
ON per.`Cuit` = det.`Cuit`
GROUP BY det.`Cuit`, det.`ID_Producto`
```

Ejemplos de Agrupación - Having

Productos de los cuales se haya solicitado más de 10:

```
SELECT  det.`Cuit`,
        det.`ID_Producto`,prod.`Descripc_Prod`,
        SUM(det.`Cantidad`)
FROM `detalle_pedidos` det
INNER JOIN productos prod
ON prod.`ID_Producto` = det.`ID_Producto`
INNER JOIN personas per
ON per.`Cuit` = det.`Cuit`
GROUP BY det.`Cuit`, det.`ID_Producto`
HAVING SUM(det.`Cantidad`)  > 10
```

EJEMPLO DE AGRUPAMIENTO PARA *PRECIOS*

La tabla precios del ejemplo contiene los precios de los productos de cada proveedor a distintas fechas.

Para obtener cuál es la última fecha a la que se tienen precios de los productos de los proveedores podríamos realizar una consulta agrupada como la siguiente:

```
SELECT prfec.`Cuit`, prfec.`ID_Producto`,
       MAX(prfec.`Fecha_Desde`)
FROM precios prfec
WHERE prfec.`Fecha_Desde` <=CURRENT_DATE
GROUP BY  prfec.`Cuit`, prfec.`ID_Producto`
```

CUIDADO!!! Esta consulta no permite obtener el valor del precio para la máxima fecha ... probar!!!

EJEMPLO DE AGRUPAMIENTO PARA *PRECIOS*

Haciendo algunos cambios, la consulta anterior puede utilizarse como subconsulta para obtener la lista de precios de los proveedores a la fecha de hoy:

```
SELECT *
FROM `precios` prec
WHERE prec.`Fecha_Desde` =

(SELECT MAX(prfec.`Fecha_Desde`)
  FROM precios prfec
  WHERE prfec.`Cuit` = prec.`Cuit`
      AND prfec.`ID_Producto` = prec.`ID_Producto`
      AND prfec.`Fecha_Desde` <=CURRENT_DATE
     GROUP BY  prfec.`Cuit`, prfec.`ID_Producto`)
```

USO DE TABLAS TEMPORALES

El problema de las subconsultas es que se ejecutan **por cada fila de la consulta principal** por lo cual puede resultar muy lento cuando nos encontramos con muchos registros.

En estos casos puede utilizarse una **tabla temporal** (aunque algunos gestores recomiendan no utilizarlas si son muy voluminosas por el acceso a disco y los bloqueos que generan).

Una **tabla temporal** se crea y se elimina igual que una tabla normal, con los comandos:

- **CREATE**
- **DROP**

Pero no afectan al diccionario de datos.

Se pueden insertar los registros con INSERT o se puede utilizar una cláusula SELECT para insertar los registros en el momento de la creación

EJEMPLO DE TABLAS TEMPORALES: *PRECIOS*

Si quisiéramos utilizar una tabla temporal en lugar de una subconsulta para la lista de precios existentes las sentencias serían:

```
DROP TEMPORARY TABLE IF EXISTS preciosahoy;

CREATE TEMPORARY TABLE preciosahoy
(SELECT prec.`Cuit`, prec.`ID_Producto`,
        MAX(prec.`Fecha_Desde`)
FROM precios prec
WHERE prec.`Fecha_Desde` <=CURRENT_DATE
GROUP BY  prec.`Cuit`, prec.`ID_Producto`);
```

EJEMPLO DE TABLAS TEMPORALES: *PRECIOS*

La consulta para el ejemplo de la lista de precios completa utilizando la tabla temporal creada anteriormente:

```
SELECT prod.`ID_Producto`, prod.`Nombre_producto`,
       prop.`Cuit`, per.`Razon_Social`, prec.`Precio`
FROM productos prod
LEFT JOIN productos_proveedores prop
ON prop.`ID_Producto` = prod.`ID_Producto`
LEFT JOIN personas per
ON per.`Cuit` = prop.`Cuit`
LEFT JOIN tt_prfec prfec
ON prfec.`Cuit` = prop.`Cuit`
AND prfec.`ID_Producto` = prop.`ID_Producto`
LEFT JOIN precios prec
ON prec.`Cuit` = prfec.`Cuit`
AND prec.`ID_Producto` = prfec.`ID_Producto`
AND prec.`Fecha_Desde` = prfec.`Fecha_max`
```