Bases de Datos

SQL (Structured Query Language)

SQL

Lenguaje de consultas de BD, que está compuesto por dos submódulos:

i. Módulo para definición del modelo de datos, denominado **DDL** (Data Definition Language).

i. Módulo para la operatoria normal de la BD, denominado **DML** (Data Manipulation Language).

SQL

Estructura básica de una consulta SQL:

SELECT lista_de_atributos
FROM lista_de_tablas
WHERE (predicado) /*Opcional*/

lista_de_atributos indica los nombres de los atributos que serán presentados en el resultado.

lista_de_tablas indica las tablas de la BD necesarias para resolver la consulta.

predicado indica que condición deben cumplir las tuplas de las tablas para estar en el resultado final de la consulta.

SQL-Analogía con Álgebra Relacional

AR representa la base teórica de SQL, por lo tanto las consultas expresadas en ambos lenguajes son similares en aspectos semánticos.

```
SELECT atr1, atr2, atr3
FROM tabla1, tabla2
WHERE (atr4 = 'valor')
```

Equivale a la siguiente consulta en AR:

$$\pi_{atr1,\ atr2,\ atr3}\ (\sigma_{atr4\ =\ 'valor'}\ (tabla1 \times tabla2))$$

SQL-Operadores

*: indica que todos los atributos de las tablas definidas en el FROM, serán presentados en el resultado de la consulta.

SELECT *
FROM producto

DISTINCT: elimina **tuplas** repetidas.

SELECT DISTINCT(codProducto)
FROM detalle
WHERE (precio> 10000)

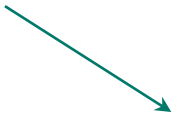
SQL-Operadores

BETWEEN: permite verificar si un valor se encuentra en un rango determinado de valores.

SELECT nroTicket

FROM factura

WHERE (total **BETWEEN** 5000 and 6000)



Se incluyen los extremos.

SQL-Operadores

Los atributos utilizados en el SELECT de una consulta SQL pueden tener asociados operaciones válidas para sus dominios.

SELECT nroTicket, total* 0.21

FROM factura

WHERE (Year(factura.fecha)=2020)

SQL-Operadores

Producto Cartesiano (,): para realizar un producto cartesiano, basta con poner en la cláusula FROM dos o más tablas separadas por coma.

SELECT f.nroTicket as ticket, f.total as totalCompra
FROM producto p, factura f , detalle d
WHERE (p.codigo = d.codProducto)
and (d.idFactura=f.id)

AS: Renombre de atributos.

Se filtran las tuplas con sentido.

Alias definido para una tabla.

SQL-Operadores

- **UNION**: misma interpretación que en AR. No retorna tuplas duplicadas.
- **UNION ALL**: misma interpretación que la UNION pero retorna las tuplas duplicadas.
- **EXCEPT**: cláusula definida para la diferencia de conjuntos.
- **INTERSECT**: cláusula para la operación de intersección

SQL-Operadores

LIKE: brinda gran potencia para aquellas consultas que requieren manejo de Strings. Se puede combinar con:

%: representa cualquier cadena de caracteres, inclusive la cadena vacía.

_ (guión bajo): sustituye solo el carácter del lugar donde aparece.

SELECT nombre
FROM producto
WHERE (nombre **LIKE** “Agen%”)

SELECT nombre
FROM producto
WHERE (descripcion **LIKE** “utilit _ _ _ _”)

SQL-Operadores

ORDER BY: permite ordenar las tuplas resultantes por el atributo que se le indique. Por defecto ordena de menor a mayor (operador **ASC**). Si se desea ordenar de mayor a menor, se utiliza el operador **DESC**.

SELECT DISTINCT nombre, descripcion, precio_unitario
FROM producto p, factura f, detalle d
WHERE (f.total =5000) **AND** (p.codigo = d.codProducto)
and (d.idFactura=f.id)
ORDER BY nombre **,** precio_unitario **DESC**

Dentro de la cláusula ORDER BY se pueden indicar más de un criterio de ordenación. El segundo criterio se aplica en caso de empate en el primero y así sucesivamente.

SQL-Operadores

IS NULL (su negación **IS NOT NULL**): verifica si un atributo contiene el valor de NULL, valor que se almacena por defecto si el usuario no define otro.

SELECT nombre
FROM producto
WHERE (descripcion **IS NOT NULL**)

SQL-Funciones de agregación

Funciones de Agregación: operan sobre un conjunto de tuplas de entrada y producen un único valor de salida.

- **AVG**: promedio del atributo indicado para todas las tuplas del conjunto.
- **COUNT**: cantidad de tuplas involucradas en el conjunto de entrada.
- **MAX**: valor más grande dentro del conjunto de tuplas para el atributo indicado.
- **MIN**: valor más pequeño dentro del conjunto de tuplas para el atributo indicado.
- **SUM**: suma del valor del atributo indicado para todas las tuplas del conjunto.

SQL-Funciones de agrupamiento

GROUP BY: agrupa las tuplas de una consulta por algún criterio con el objetivo de aplicar alguna función de agregación.

p.Codigo	p.Nombre	p.Monto	d.Codigo	d.monto
1	Plancha	50	1	40
1	Plancha	50	1	35
1	Plancha	50	1	50
2	Heladera	100	2	90
2	Heladera	100	2	100
3	Heladera	120	3	120

SELECT nombre, **SUM**(d.monto) as Suma
FROM producto p, detalle d
WHERE (p.codigo = d.codProducto)
GROUP BY p.codigo, p.nombre

nombre	Suma
Plancha	125
Heladera	190
Heladera	120

SELECT nombre, **COUNT**(*) as cantidad
FROM producto p, detalle d
WHERE (p.codigo = d.codProducto)
GROUP BY p.codigo, p.nombre

nombre	Cantidad
Plancha	3
Heladera	2
Heladera	1

Cuál es la relación entre condiciones de agrupamiento e información que se muestra?

SQL-Subconsulta

Subconsulta: consiste en ubicar una consulta SQL dentro de otra. SQL define operadores de comparación para subconsultas:

- = **(igualdad)**: cuando una subconsulta retorna un único resultado, es posible compararlo contra un valor simple.
- IN **(pertenencia)**: comprueba si un elemento es parte o no de un conjunto. Negación (**NOT IN**).
- **=SOME**: igual a alguno.
- **>ALL**: mayor que todos.
- **<=SOME**: menor o igual que alguno

SQL-Subconsulta

```
SELECT DISTINCT f.nroTicket, f.total
FROM factura f , detalle d
WHERE (d.idFactura=f.id)
and d.codProducto=SOME(SELECT  codigo
FROM producto WHERE nombre like
'Jug%')
```

SQL-Cláusula Exist

EXIST: se utiliza para comprobar si una subconsulta generó o no alguna tupla como respuesta. El resultado de la cláusula EXIST es verdadero si la subconsulta tiene al menos una tupla, y falso en caso contrario. Negación (**NOT EXIST**)

SQL-Cláusula Exist

SELECT p.nombre
FROM producto p
WHERE EXIST (SELECT * FROM detalle d **WHERE**
(d.codProducto=p.codigo**))**

Condición de la consulta principal

SQL-Producto Natural

INNER JOIN: producto natural clásico, reúne las tuplas de las relaciones que tienen sentido. El producto natural se realiza en la cláusula FROM indicando la tablas involucradas en dicho producto, y luego de la sentencia **ON** la condición que debe cumplirse.

SELECT DISTINCT p.nombre, p.descripcion, p.precio_unitario
FROM producto p
INNER JOIN detalle d **ON** (d.codProducto=p.codigo)

SQL-Producto Natural

LEFT JOIN: contiene todos los registros de la tabla de la izquierda, aún cuando no exista un registro correspondiente en la tabla de la derecha, para uno de la izquierda. Retorna un valor nulo (NULL) en caso de no correspondencia.

RIGHT JOIN: es la inversa del LEFT JOIN.

SELECT DISTINCT p.nombre, p.descripcion, p.precio_unitario
FROM producto p
LEFTJOIN detalle d **ON** (d.codProducto=p.codigo)

SQL-ABM

INSERT INTO: agrega tuplas a una tabla.

DELETE FROM: borra una tupla o un conjunto de tuplas de una tabla.

UPDATE … SET: modifica el contenido de uno o varios atributos de una tabla.

SQL-EJEMPLOS

Modelo Físico

Producto(codigo, nombre, descripcion,precio_unitario) // listado de todos los productos

Detalle(codProducto, idFactura, cantidad, precio) // detalle de la venta

Factura(id, nroTicket, fecha, total) //listado de facturas de venta

SQL-EJEMPLOS

Mostrar el nombre de todos los productos vendidos en mayo de 2020.

Producto(codigo, nombre, descripcion,precio_unitario) // listado de todos los productos

Detalle(codProducto, idFactura, cantidad, precio) // detalle de la venta

Factura(id, nroTicket, fecha, total) //listado de facturas de venta

```
SELECT DISTINCT p.nombre
FROM producto p
INNER JOIN detalle d ON (p.codigo = d.codProducto)
INNER JOIN  factura f ON (d.idFactura=f.id)
WHERE  YEAR (f.fecha)=2020 and MONTH(f.fecha)=5
```

SQL-EJEMPLOS

Mostrar nroTicket, la fecha y el monto total de facturas en las que se vendió algún producto cuya descripción contenga el string ´za´. El resultado debe estar ordenado por fecha

Producto(codigo, nombre, descripcion,precio_unitario) // listado de todos los productos

Detalle(codProducto, idFactura, cantidad, precio) // detalle de la venta

Factura(id, nroTicket, fecha, total) //listado de facturas de venta

```
SELECT  DISTINCT nroTicket, fecha, total
FROM factura f
INNER JOIN detalle d ON (d.idFactura=f.id)
INNER JOIN producto p ON (p.codigo = d.codProducto)
WHERE (p.descripcion LIKE  '%za%')
ORDER BY  fecha
```

SQL-EJEMPLOS

Se debe calcular el monto total facturado durante el transcurso del año 2019.

Producto(codigo, nombre, descripcion,precio_unitario) // listado de todos los productos

Detalle(codProducto, idFactura, cantidad, precio) // detalle de la venta

Factura(id, nroTicket, fecha, total) //listado de facturas de venta

SELECT SUM (total) as monto total
FROM factura f
WHERE (fecha **BETWEEN** "2019-01-01" and "2019-12-31")

SQL-EJEMPLOS

Informar nombre, descripción y precio unitario de productos que solo fueron vendidos durante marzo de 2020

Producto(codigo, nombre, descripcion,precio_unitario) // listado de todos los productos

Detalle(codProducto, idFactura, cantidad, precio) // detalle de la venta

Factura(id, nroTicket, fecha, total) //listado de facturas de venta

```
SELECT DISTINCT nombre, descripcion, precio_unitario
FROM producto p
INNER JOIN detalle d ON (p.codigo = d.codProducto)
INNER JOIN  factura f ON (d.idFactura=f.id)
WHERE  YEAR (f.fecha)=2020 and MONTH(f.fecha)=3 and p.codigo  NOT IN
            (SELECT d.codigoProducto
            FROM detalle d
            INNER JOIN  factura f ON (d.idFactura=f.id)
            WHERE (YEAR(f.fecha)=2020 and MONTH(f.fecha)<>3)  or YEAR(f.fecha)<> 2020))
```

SQL-EJEMPLOS

Informar para cada producto: el nombre, descripción y la cantidad de facturas en las que se vendió.

Producto(codigo, nombre, descripcion,precio_unitario) // listado de todos los productos

Detalle(codProducto, idFactura, cantidad, precio) // detalle de la venta

Factura(id, nroTicket, fecha, total) //listado de facturas de venta

SELECT nombre, descripcion, **COUNT**(d.idFactura) as cant_facturas
FROM producto p
LEFT JOIN detalle d **ON** (p.codigo = d.codProducto)
GROUP BY p.codigo, nombre, descripción

Otra solución es: usar inner join en vez de left join y luego unir con la selección de productos que no fueron vendidos forzando el valor 0 en la cantidad de facturas del producto

SQL-EJEMPLOS

Informar nroTicket, fecha y total de facturas donde se vendió el producto con nombre ‘X’ o que no se vendió el producto ‘Y’.

Producto(codigo, nombre, descripcion,precio_unitario) // listado de todos los productos

Detalle(codProducto, idFactura, cantidad, precio) // detalle de la venta

Factura(id, nroTicket, fecha, total) //listado de facturas de venta

```
SELECT  DISTINCT nroTicket, fecha, total
FROM factura f
INNER JOIN detalle d ON (d.idFactura=f.id)
INNER JOIN producto p ON (p.codigo = d.codProducto)
WHERE (p.nombre = ‘X’)

UNION

SELECT  nroTicket, fecha, total
FROM factura f
WHERE  f.id not exist (SELECT *
                       FROM detalle d1
                      INNER JOIN producto p1 ON (p1.codigo = d1.codProducto)
                      WHERE f.id=d1.idFactura and p1.nombre=‘Y’)
```

SQL-EJEMPLOS

Informar nombre y descripción de aquellos productos cuyo valor total en ventas supere los $50000000.

Producto(codigo, nombre, descripcion,precio_unitario) // listado de todos los productos

Detalle(codProducto, idFactura, cantidad, precio) // detalle de la venta

Factura(id, nroTicket, fecha, total) //listado de facturas de venta

```
SELECT nombre, descripcion
FROM producto p
INNER JOIN detalle d ON (p.codigo = d.codProducto)
GROUP BY p.codigo, p.nombre, p.descripcion
HAVING SUM(cantidad * precio) > 50000000
```

Condiciones de agrupamiento

Porque se usa inner join??

SQL-EJEMPLOS

Informar nroTicket y total de facturas en las que se vendieron los productos ‘C’ y ‘D’

Producto(codigo, nombre, descripcion,precio_unitario) // listado de todos los productos

Detalle(codProducto, idFactura, cantidad, precio) // detalle de la venta

Factura(id, nroTicket, fecha, total) //listado de facturas de venta

```
SELECT  nroTicket, fecha, total
FROM factura f
INNER JOIN detalle d ON (d.idFactura=f.id)
INNER JOIN producto p ON (p.codigo = d.codProducto)
WHERE (p.nombre = ‘C’)

INTERSECT

SELECT  nroTicket, fecha, total
FROM factura f
INNER JOIN detalle d ON (d.idFactura=f.id)
INNER JOIN producto p ON (p.codigo = d.codProducto)
WHERE (p.nombre = ‘D’)
```