



# Diseño de Bases de Datos

Curso 2017- SQL

# Structured Query Language(SQL)

Lenguaje de consultas de BD, compuesto por dos submódulos:

- ❖ Módulo para definición del modelo de datos, denominado DDL (Data Definition Language).
- ❖ Módulo para la operatoria normal de la BD, denominado DML (Data Manipulation Language).

# SQL- Definición de datos

- ❖ De base de datos: CREATE|DROP SCHEMA
- ❖ De dominios: CREATE|ALTER|DROP DOMAIN
- ❖ De tablas: CREATE|ALTER|DROP TABLE
- ❖ De vistas: CREATE|DROP VIEW
- ❖ De índices (algunos SGBD): CREATE|DROP INDEX

# SQL- Manipulación de datos

- ❖ SELECT

- ❖ INSERT

- ❖ UPDATE

- ❖ DELETE

# SQL- Manipulación de datos

## Modelo físico de ejemplo

Producto(codP, nombre, descripcion, precio\_unitario)  
// listado de todos los productos

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

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

# 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**(codP)

**FROM** detalle

**WHERE** (cantidad<= 20)

# SQL- Operadores

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

**SELECT** nombre, descripcion

**FROM** producto

**WHERE** (precio\_unitario **BETWEEN** 100 and 200)



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
```

**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, descripcion  
FROM producto  
WHERE (descripcion IS NOT NULL)
```



# SQL- Operadores

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

**AS:** Renombre de atributos.

**SELECT DISTINCT** f.nroTicket , f.total **as** totalFactura

**FROM** producto p, factura f , detalle d

**WHERE** (p.codP = d.codP)

Se filtran las tuplas con sentido.

**and** (d.idFactura=f.idFactura)

**and** (p.nombre='zapatos')

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.

**Las consultas a unir deben tener esquemas compatibles**

**SELECT** nombre, descripcion

**FROM** producto

**WHERE** (descripcion **IS NOT NULL**)

**UNION**

(**SELECT DISTINCT** nombre, descripcion

**FROM** producto p, detalle d

**WHERE** (p.codp=d.codP ) **and** (d.precio=1000))

Que retornaría si cambiamos **UNION** por **UNION ALL**?

# SQL- Operadores

❖ **EXCEPT**: cláusula definida para la diferencia de conjuntos.

```
SELECT nombre, descripcion  
FROM producto  
WHERE (descripcion IS NULL)  
EXCEPT
```

Las consultas deben tener  
esquemas compatibles para **except**  
e **intersect**

```
(SELECT DISTINCT nombre, descripcion  
FROM producto p, detalle d  
WHERE (p.codp=d.codP))
```

Que retornaría si cambiamos **except**  
por **intersect**?

❖ **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 "%pas%")
```

```
SELECT nombre  
FROM producto  
WHERE (descripcion LIKE "__pato")
```

# 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 Year(f.fecha)=2017 AND (p.codP = d.codP)  
AND(d.idFactura=f.idFactura)  
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- 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- Agrupamiento

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

```
SELECT nombre, AVG(f.total) as promedio  
FROM producto p, factura f, detalle d  
WHERE (p.codP = d.codP)  
and (d.idFactura=f.id)  
GROUP BY p.codP, p.nombre
```

Qué información se puede mostrar cuando se realizó un agrupamiento?

Porque es importante agrupar además por PK?

# SQL- Agrupamiento clausula HAVING

La cláusula **HAVING** se usa con la cláusula GROUP BY para restringir los grupos que aparecen en la tabla de resultados mediante alguna condición que deben cumplir los grupos

```
SELECT nombre, AVG(f.total) as promedio  
FROM producto p, factura f, detalle d  
WHERE (p.codP = d.codP)  
and (d.idFactura=f.id)  
GROUP BY p.codP, p.nombre  
HAVING (AVG(f.total)) >10000
```



# SQL- Funciones agregación y agrupamiento

```
SELECT COUNT(*) as  
cantidadFacturas2017  
FROM factura f  
WHERE YEAR(f.fecha)=2017
```

```
SELECT d.codP, COUNT (DISTINCT  
(d.idFactura))as cantVecesVendioProducto  
FROM detalle d  
GROUP BY d.codP
```

```
SELECT nombre,  
SUM(cantidad*precio) as  
MontoVendProducto  
FROM producto p,detalle d  
WHERE (p.codP = d.codP)  
GROUP BY p.codP, p.nombre
```

```
SELECT nombre, MAX(precio) as  
ValorMayorVendioProducto  
FROM producto p,detalle d  
WHERE (p.codP = d.codP)  
GROUP BY p.codP, p.nombre
```

# SQL- Subconsultas

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- Subconsultas

**SELECT DISTINCT** f.nroTicket, f.total

**FROM** factura f , detalle d

**WHERE** (d.idFactura=f.idFactura)


**and d.codP=SOME(SELECT codP FROM**  
**producto WHERE descripcion like '\_\_\_\_**  
**\_\_\_\_\_')**

# SQL- Cláusula Exist

Permite 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**)

```
SELECT p.nombre  
FROM producto p  
WHERE NOT EXIST (SELECT * FROM detalle d  
WHERE (d.codP=p.codP))
```



Condición de la consulta principal

# SQL- Producto natural

**INNER JOIN:** producto natural, 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.codP=p.codP)
```

**NATURAL JOIN:** análogo al producto natural de AR, trabaja por equicombinación.

# 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, d.cantidad as cantidadDetalle  
FROM producto p  
LEFTJOIN detalle d ON (d.codP=p. codP)
```

# SQL- Equivalencia AR

- ❖  $T1 \bowtie T2 \equiv \text{SELECT } * \text{ FROM } T1 \text{ NATURAL JOIN } T2$   
(equicombinación de tuplas de T1 y T2 sobre la *foreign key* de una que referencie a la *primary key* de la otra)
- ❖  $T1 \bowtie T2 \equiv \text{SELECT } * \text{ FROM } T1 \text{ INNER JOIN } T2 \text{ ON } (\dots)$  (equicombinación de tablas sobre los campos que se especifiquen –cuando se quieren igualar campos que no coincidan con la *primary key* de una y la *foreign key* de la otra)
- ❖  $T1 \bowtie_{\theta} T2 \equiv \text{SELECT } * \text{ FROM } T1 \text{ NATURAL JOIN } T2 \text{ WHERE } \dots$
- ❖  $T1 \ltimes T2 \equiv \text{SELECT } * \text{ FROM } T1 \text{ LEFT JOIN } T2 \text{ ON } (\dots)$

# SQL- ABM

❖ **INSERT INTO**: agrega tuplas a una tabla.

**INSERT INTO** producto (nombre, descripcion, precio\_unitario) **VALUES** ('Azucar', azucar Y x 1 kg',20);

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

**DELETE FROM** detalle **WHERE** idFactura=1000;

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

**UPDATE** factura **SET** total=total + total\*0.21





# SQL

**¿Consultas?**