

# Bases de Datos

Clase 07 - 2: SQL Avanzado

# Clase anterior

```
SELECT <atributos>  
FROM <relaciones>  
WHERE <condiciones / selecciones>
```

# Clase anterior

```
SELECT <atributos>  
FROM <relaciones>  
WHERE <condiciones / selecciones>
```

Además, existen operadores como **LIKE**,  
**DISTINCT**, **ORDER**, **UNION**, etc.

# En esta clase

- Consultas anidadas
- Agregación - Agrupación
- **UPDATE**

# Consultas Anidadas

```
SELECT Bandas.nombre
FROM Bandas, Estudiantes_UC
WHERE Bandas.vocalista = Estudiantes_UC.nombre
      AND Bandas.nombre =
          (SELECT Festivales.banda
           FROM Festivales
           WHERE Festivales.nombre = 'Lollapalooza')
```

# Consultas Anidadas

```
SELECT Bandas.nombre
FROM Bandas, Estudiantes_UC
WHERE Bandas.vocalista = Estudiantes_UC.nombre
      AND Bandas.nombre =
          (SELECT Festivales.banda
           FROM Festivales
           WHERE Festivales.nombre = 'Lollapalooza')
```

¿Y si la subconsulta retorna más de un valor?

# Consultas Anidadas

Operador **IN**

```
SELECT Bandas.nombre
FROM Bandas, Estudiantes_UC
WHERE Bandas.vocalista = Estudiantes_UC.nombre
      AND Bandas.nombre IN
      (SELECT Festivales.banda
       FROM Festivales
       WHERE Festivales.nombre = 'Lollapalooza')
```

# Consultas Anidadas

Operador **IN**

```
SELECT Bandas.nombre
FROM Bandas, Estudiantes_UC
WHERE Bandas.vocalista = Estudiantes_UC.nombre
      AND Bandas.nombre IN
      (SELECT Festivales.banda
       FROM Festivales
       WHERE Festivales.nombre = 'Lollapalooza')
```

Comprobamos que **Banda.nombre** esté dentro de un listado de valores.



# Consultas Anidadas

Quitando la anidación

```
SELECT Bandas.nombre  
FROM Bandas, Estudiantes_UC, Festivales  
WHERE Bandas.vocalista = Estudiantes_UC.nombre  
      AND Bandas.nombre = Festivales.banda  
      AND Festivales.nombre = 'Lollapalooza'
```

# Consultas Anidadas

Quitando la anidación

```
SELECT Bandas.nombre  
FROM Bandas, Estudiantes_UC, Festivales  
WHERE Bandas.vocalista = Estudiantes_UC.nombre  
      AND Bandas.nombre = Festivales.banda  
      AND Festivales.nombre = 'Lollapalooza'
```

¿Por qué no es equivalente a la consulta anterior?  
Hint: ¡Cuidado con los duplicados!

# Consultas Anidadas

```
SELECT Bandas.nombre
FROM Bandas, Estudiantes_UC
WHERE Bandas.vocalista = Estudiantes_UC.nombre
      AND Bandas.nombre IN
          (SELECT Festivales.banda
           FROM Festivales
           WHERE Festivales.nombre = 'Lollapalooza')
```

En caso de subconsulta con duplicados, es equivalente a:

```
SELECT DISTINCT Bandas.nombre
FROM Bandas, Estudiantes_UC, Festivales
WHERE Bandas.vocalista = Estudiantes_UC.nombre
      AND Banda.nombre = Festivales.banda
      AND Festivales.nombre = 'Lollapalooza'
```

# Consultas Anidadas

ALL, ANY, EXISTS

Podemos usar:

- $S > ALL\ R$
- $S > ANY\ R$
- $EXISTS\ R$

# Consultas Anidadas

ALL, ANY

Cervezas más baratas que la Austral Calafate

```
SELECT Cervezas.nombre  
FROM Cervezas  
WHERE Cervezas.precio < ALL  
      (SELECT Cervezas2.precio  
       FROM Cervezas AS Cervezas2  
       WHERE Cervezas2.nombre = 'Austral Calafate')
```

# Consultas Anidadas

ALL, ANY

Cerveza que no sea la más cara

```
SELECT Cervezas.nombre
FROM Cervezas
WHERE Cervezas.precio < ANY
      (SELECT Cervezas2.precio
       FROM Cervezas AS Cervezas2)
```

# Consultas Anidadas

Podemos expresar estas consultas con **SELECT**  
– **FROM** – **WHERE**?

Hint: Las consultas SFW son **monótonas**. Una consulta con **ALL** no es monótona. Una consulta con **ANY** lo es

# Anidando Consultas Relacionadas

Títulos de películas que se repiten en años diferentes

Películas(título, año, director, actor)

```
SELECT p.título
FROM Películas AS p
WHERE año <> ANY
  (SELECT año
   FROM Películas
   WHERE título = p.título)
```



# Anidando Consultas Relacionadas

Títulos de películas que se repiten en años diferentes

Películas(título, año, director, actor)

```
SELECT p.título
FROM Películas AS p
WHERE año <> ANY
    (SELECT año
     FROM Películas
     WHERE título = p.título)
```

p sigue activa a medida que se anidan las consultas!

# Consultas Anidadas

EXISTS

Distribuidores con cervezas más baratas de 100

```
SELECT Distribuidores.nombre
FROM Distribuidores AS Dist
WHERE EXISTS
    (SELECT Cervezas.nombre
     FROM Cervezas
     WHERE Cervezas.nombre_distribuidor =
           Dist.nombre AND
           Cervezas.precio < 100)
```

# Consultas Anidadas

EXISTS

Distribuidores con cervezas más baratas de 100

```
SELECT Distribuidores.nombre
FROM Distribuidores AS Dist
WHERE EXISTS
    (SELECT Cervezas.nombre
     FROM Cervezas
     WHERE Cervezas.nombre_distribuidor =
           Dist.nombre AND
           Cervezas.precio < 100)
```

Es importante no olvidar el alias para no perder la referencia!

# Anidar tuplas

```
SELECT título
FROM Películas
WHERE (título, año) IN
      (SELECT título, año
       FROM Programación
       WHERE cine = 'Cinemark')
```

# Anidar tuplas

```
SELECT título
FROM Películas
WHERE (título, año) IN
      (SELECT título, año
       FROM Programación
       WHERE cine = 'Cinemark')
```

Esto no funciona en todos los sistemas

# Consultas Anidadas

Más ejemplos

Productos junto a su fabricante que son más caros que todos los productos hechos por el mismo fabricante antes del 2000

Productos(nombre, precio, categoría, fabricante, año)

```
SELECT DISTINCT p.nombre, p.fabricante
FROM Productos AS p
WHERE p.precio > ALL
    (SELECT p2.precio
     FROM Productos AS p2
     WHERE p.fabricante = p2.fabricante
     AND p2.año < 2000)
```

# Agregación

¿Qué hace esta consulta?

```
SELECT AVG(precio)
FROM Productos
WHERE fabricante = 'Toyota'
```

# Agregación

¿Qué hace esta consulta?

```
SELECT AVG(precio)
FROM Productos
WHERE fabricante = 'Toyota'
```

También podemos usar SUM, MIN, MAX



# Operaciones Aritméticas

Podemos hacer operaciones aritméticas

`Compra(producto, fecha, precio, cantidad)`

```
SELECT SUM(precio*cantidad)
FROM Compra
WHERE producto = 'tomate'
```

# Count

Podemos contar tuplas

```
SELECT COUNT(*)  
FROM Productos  
WHERE año > 2012
```

# Count

Cuidado! Se cuentan los duplicados

# Count

Cuidado! Se cuentan los duplicados

Estas consultas se comportan igual

```
SELECT COUNT(fabricante)
FROM Productos
WHERE año > 2012
```

```
SELECT COUNT(*)
FROM Productos
WHERE año > 2012
```

# Count

Contar los productos de cada fabricante

# Count

Contar los productos de cada fabricante

```
SELECT fabricante, COUNT(fabricante)
FROM Productos
WHERE año > 2012
GROUP BY fabricante
```

# Count

```
SELECT fabricante, COUNT(fabricante)
FROM Productos
WHERE año > 2012
GROUP BY fabricante
```

Esta consulta:

- Computa los resultados según el FROM y WHERE
- Agrupa los resultados según los atributos del GROUP BY
- Para cada grupo se aplica independientemente la agregación

# Ejemplo

Compra(producto, fecha, precio, cantidad)

producto	fecha	precio	cantidad
tomates	07/02	100	6
tomates	06/07	150	4
zapallo	08/02	800	1
zapallo	09/07	1000	2
zapallo	01/01	600	3



# Ejemplo

Una de las consultas más típicas:

# Ejemplo

Una de las consultas más típicas:

```
SELECT producto, SUM(precio*cantidad) AS ventaTotal  
FROM Compra  
WHERE fecha > '10/01'  
GROUP BY producto
```

```
SELECT producto, SUM(precio*cantidad) AS ventaTotal
FROM Compra
WHERE fecha > '10/01'
GROUP BY producto
```

1) Se computa el FROM y el WHERE

producto	fecha	precio	cantidad
tomates	07/02	100	6
tomates	06/07	150	4
zapallo	08/02	800	1
zapallo	09/07	1000	2

```
SELECT producto, SUM(precio*cantidad) AS ventaTotal
FROM Compra
WHERE fecha > '10/01'
GROUP BY producto
```

## 2) Agrupar según el GROUP BY

producto	fecha	precio	cantidad
tomates	07/02	100	6
	06/07	150	4
zapallo	08/02	800	1
	09/07	1000	2

```
SELECT producto, SUM(precio*cantidad) AS ventaTotal
FROM Compra
WHERE fecha > '10/01'
GROUP BY producto
```

3) Agregar por grupo y ejecutar la proyección

producto	ventaTotal
tomates	1200
zapallo	2800

# HAVING

Misma consulta, pero sólo consideramos los productos que se vendieron más de 100 veces

```
SELECT producto, SUM(precio*cantidad) AS ventaTotal
FROM Compra
WHERE fecha > '10/01'
GROUP BY producto
HAVING SUM(cantidad) > 100
```

¿Por qué usamos **HAVING** y no lo incluimos en el **WHERE**?

# Consultas con Agregación

```
SELECT <S>  
FROM R1, ..., Rn  
WHERE <Condición 1>  
GROUP BY a1, ..., ak  
HAVING <Condición 2>
```

- S puede contener atributos de  $a_1, \dots, a_k$  y/o agregados, pero ningún otro atributo (¿Por qué?)
- Condición 1 es una condición que usa atributos de  $R_1, \dots, R_n$
- Condición 2 es una condición de agregación de los atributos de  $R_1, \dots, R_n$

# Consultas con Agregación

## Evaluación

```
SELECT <S>  
FROM  $R_1, \dots, R_n$   
WHERE <Condición 1>  
GROUP BY  $a_1, \dots, a_k$   
HAVING <Condición 2>
```

- Se computa el FROM - WHERE de  $R_1, \dots, R_n$
- Agrupar la tabla por los atributos de  $a_1, \dots, a_k$
- Computar los agregados de la Condición 2 y mantener grupos que satisfacen
- Computar agregados de S y entregar el resultado



# Ejemplo

Autor(login, nombre)

Documento(url, título)

Escribe(login, url)

Menciona(url, palabra)

# Ejemplo

Versión no deseable

Encuentre todos los autores que escribieron al  
menos 10 documentos

# Ejemplo

Versión no deseable

Encuentre todos los autores que escribieron al menos 10 documentos

```
SELECT DISTINCT *  
FROM (SELECT Autor.login, COUNT(*) AS countAutor  
      FROM Escribe, Autor  
      WHERE Escribe.login = Autor.login  
      GROUP BY Autor.login) AS Foo  
WHERE countAutor >= 10
```

# Ejemplo

¡La respuesta puede mejorar! Estamos haciendo una anidación no necesaria

**Obs.** El uso del alias (en este caso Foo) en el FROM es necesario para el uso de subconsultas

# Ejemplo

Versión elegante

Encuentre todos los autores que escribieron al  
menos 10 documentos

# Ejemplo

Versión elegante

Encuentre todos los autores que escribieron al menos 10 documentos

```
SELECT Autor.nombre  
FROM Autor, Escribe  
WHERE Autor.login = Escribe.login  
GROUP BY Autor.nombre  
HAVING COUNT(Escribe.url) >= 10
```

# Ejemplo

Versión elegante

Encuentre todos los autores que escribieron al menos 10 documentos

```
SELECT Autor.nombre  
FROM Autor, Escribe  
WHERE Autor.login = Escribe.login  
GROUP BY Autor.nombre  
HAVING COUNT(Escribe.url) >= 10
```

**Obs.** No necesitamos DISTINCT gracias al GROUP BY

# Ejemplo

Encuentre todos los autores con un vocabulario  
de más de 10.000 palabras



# Ejemplo

Encuentre todos los autores con un vocabulario de más de 10.000 palabras

```
SELECT Autor.nombre
FROM Autor, Escribe, Menciona
WHERE Autor.login = Escribe.login
      AND Escribe.url = Menciona.url
GROUP BY Autor.nombre
HAVING COUNT(DISTINCT Menciona.palabra) >= 10000
```

# Insert

Para insertar valores:

# Insert

Para insertar valores:

```
INSERT INTO R  
VALUES <Valores>
```

# Insert

Para insertar valores de otra consulta:

```
INSERT INTO Peliculas(título)
VALUES (
    SELECT DISTINCT Película
    FROM Programación
    WHERE Programación.título NOT IN
        (SELECT título
         FROM películas)
)
```

Estamos insertando sólo el título, ¿Qué pasa con los demás valores?

# Update

Para actualizar valores de una tabla:

# Update

Para actualizar valores de una tabla:

```
UPDATE Viajes  
SET aerolínea = 'LaTam'  
WHERE aerolínea LIKE '%Lan%'  
      OR aerolínea LIKE '%Tam%'
```

# Update

Estudiantes(nombre, apellido, Rut)

```
UPDATE Estudiantes  
SET nombre = 'apellido' || 'nombre'  
WHERE Rut LIKE '%K%'
```

Quizás después:

```
ALTER TABLE Estudiantes DROP apellido
```

# Update

Forma general

```
UPDATE R  
SET <Nuevos valores>  
WHERE <Condición sobre R>
```

$\langle \text{Nuevos valores} \rangle := (\text{atributo}_1 = \text{nuevoValor}_1, \dots, \text{atributo}_n = \text{nuevoValor}_n)$



# Delete

Para borrar tuplas que satisfagan una condición:

```
DELETE FROM R  
WHERE <Condición sobre R>
```

Para borrar todo:

```
DELETE FROM R
```