# Sistemas Informáticos

Tema 3: Bases de datos distribuidas

## 3.2 SQL (Structured Query Language)


## Lenguajes de SQL

- **DDL** Lenguaje de definición de datos (Data Definition Language, DDL).
  - Definición de esquemas de relación
  - Borrado de relaciones
  - Creación de índices
  - Modificación de esquemas de relación
  - Órdenes para la definición de vistas
  - Órdenes para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos
  - Órdenes para especificar derechos de acceso para las relaciones y vistas
- **DML** Lenguaje interactivo de manipulación de datos (Data Manipulation Language, DML).
  - Incluye un lenguaje de consultas, basado en el álgebra relacional (y en el cálculo de tuplas)
  - Incluye órdenes para insertar, borrar y modificar tuplas de la base de datos.

# Sistemas Informáticos

Tema 3.2: SQL

## SQL, Data Definition Language (DDL)
## Lenguaje de definición de datos


## Tipos de Datos en SQL

- **char(n).** Cadena de longitud fija. La longitud es n caracteres.
- **varchar(n).** Cadena de longitud variable. La longitud máxima es n caracteres.
- **int/integer.** Entero.
- **smallint.** Entero corto.
- **numeric(p,d).** Numero en formato de coma fija, con precisión de *p* dígitos, con *d* dígitos a la derecha de la coma decimal. (1-> 0.9999)
- **real, double precision.** numero en coma flotante y número en coma flotante con doble precisión.
- **float(n).** Número en coma flotante con una precisión no menor de *n* dígitos.
- El valor NULL esta permitido para todos los atributos a menos que se prohíba explícitamente. **not null** prohíbe el uso del valor NULL.
- La construcción **create domain** en SQL-92 crea tipos de datos definidos por el usuario

  **create domain** *nombre-persona* **char**(20) **not null**

  (define/typedef in C)

## Día y Hora en SQL

- **date.** Fecha (día del año), año (4 dígitos), mes y dia
  - Ej. **date** '2001-7-27'
- **time.** hora del día, en horas, minutos y segundos.
  - E.j. **time** '09:00:30' **time** '09:00:30.75'
- **timestamp**: día y hora
  - E.j. **timestamp** '2001-7-27 09:00:30.75'
- **Interval**: periodo de tiempo
  - E.j. Interval '1' day
  - la diferencia entre "date/time/timestamp" da un "interval"
  - "Interval" se puede sumar a "date/time/timestamp"
- Se pueden extraer valores independientes de "date/time/timestamp"
  - E.j. **extract** (**year from** r.comienzo)


## Creación de Tablas (y destruccion)

- Para crear una tabla se usa la orden CREATE TABLE. Es necesario especificar (al menos) el nombre de la tabla, los nombres de las columnas y el tipo de dato. Por ejemplo:

```
CREATE TABLE tablita (
   nombre1 char(20), -- hola
   nombre2 integer   -- que tal
);
```

- Cuando las tablas no sean necesarias se pueden borrar con la orden DROP TABLE. Por ejemplo:

```
DROP TABLE tablita;
```

- Todos los comandos CREATE tienen una pareja DROP

## Restricciones

- " Check", Restricción arbitraria
- " Not-Null", El atributo no acepta valores nulos
- " Unique", El atributo no acepta valores repetidos
- " Primary Keys", El atributo es clave primaria
- " Foreign Keys", El atributo es clave extranjera


## Ejemplos

- CHECK: Permite especificar que los valores de una columna deben satisfacer una expresión. Por ejemplo ser positivo.

```
CREATE TABLE productos (
    producto_no integer,
    nombre text,
    precio numeric(10,2) CHECK (precio > 0)
);

CREATE TABLE productos (
        producto_no integer,
        nombre text,
        precio numeric(10,2) CONSTRAINT
              precio_positivo CHECK (precio > 0)
    );
```

## Ejemplos

- NOT NULL: Indica que el atributo no puede valer “NULL”

```
CREATE TABLE productos (
    producto_no  integer NOT NULL,
    nombre text NOT NULL,
    precio  numeric(10,2)
);
```

- Pueden existir varias restricciones referidas al mismo atributo, el order no importa.

```
CREATE TABLE productos (
    producto_no integer NOT NULL,
    nombre text NOT NULL,
    precio numeric(10,2) NOT NULL CHECK (precio > 0)
);
```


## Ejemplos

- UNIQUE: Asegura que un determinado valor no está repetido en una columna.

```
CREATE TABLE productos (
    producto_no integer UNIQUE,
    nombre text,
    precio numeric(10,2)
);
```

```
CREATE TABLE productos (
            producto_no integer,
            nombre text,
            precio numeric,
            UNIQUE(producto_no,nombre)
        );
```

## Ejemplos

CLAVE PRIMARIA

```
CREATE TABLE productos (
        producto_no integer PRIMARY KEY,
        nombre text,
        precio numeric
    );

CREATE TABLE ejemplo (
    a integer,
    b integer,
    c integer,
    PRIMARY KEY (a, c)
);
```


## Ejemplos

CLAVE EXTRANJERA

- La restricción “REFERENCES” asegura que los valores de una determinada columna debe ser idénticos a los valores que aparecen en otra determinada columna que puede estar en otra tabla

```
CREATE TABLE productos (
    producto_no integer PRIMARY KEY,
    nombre text,
    precio numeric);
CREATE TABLE pedidos (
    pedido_no integer PRIMARY KEY,
    producto_no integer REFERENCES
             productos(producto_no),
    cantidad integer);
```

- PostgreSQL no nos dejará crear pedidos sobre productos que no existan

## Modificar tablas:"Drop" y "Alter"

- La orden **drop table nombre_de_la_tabla** elimina la tabla nombre_de_la_tabla (y toda información relacionada con ella) de la base de datos.
- La orden **alter table nombre_de_la_tabla** se usa para añadir atributos a una relación.

**alter table** *productos* **add column** *A integer*

El valor inicial de los atributos es NULL (a menos que se especifique un valor por defecto).

- La orden **alter table** se puede usar para borrar atributos de una tabla

**alter table** *productos* **drop** *A*


## Añadir/Modificar una restricción

- `ALTER TABLE productos ADD CHECK (nombre <> '');`
- `ALTER TABLE productos ADD CONSTRAINT some_name UNIQUE (producto_no);`
- `ALTER TABLE productos ADD FOREIGN KEY (producto_group_id) REFERENCES grupo_productos;`

- `ALTER TABLE productos ALTER COLUMN producto_no SET NOT NULL;`
- `ALTER TABLE productos ALTER COLUMN precio SET DEFAULT 7.77;`
- `ALTER TABLE productos RENAME COLUMN producto_no TO product_number;`

## Ejemplo del Banco

```
create table cuenta
   (numero_cuenta  varchar(15),-- not null unique,
    nombre_sucursal  varchar(15) not null,
    saldo    numeric  not null,
    primary key(numero_cuenta));

create table sucursal
   (nombre_sucursal  varchar(15) not null unique,
    ciudad_sucursal  varchar(15) not null,
    capital    numeric  not null,
    primary key(nombre_sucursal));

create table cliente
   (id              varchar(15),
    nombre_cliente  varchar(15) not null,-- unique,
    calle_cliente  varchar(12) not null,
    ciudad_cliente  varchar(15) not null,
    primary key(id));
```

```
create table prestamo
   (numero_prestamo  varchar(15) not null unique,
    nombre_sucursal  varchar(15) not null,
    cantidad    numeric  not null,
    primary key(numero_prestamo));

create table cliente_cuenta
   (id_cliente  varchar(15) not null,
    numero_cuenta  varchar(15) not null,
    primary key(id_cliente, numero_cuenta),
    foreign key(numero_cuenta) references
   cuenta(numero_cuenta),
    foreign key(id_cliente) references
   cliente(id));

create table cliente_prestamo
   (id_cliente  varchar(15) not null,
    numero_prestamo  varchar(15) not null,
    primary key(id_cliente, numero_prestamo),
    foreign key(id_cliente) references cliente(id),
    foreign key(numero_prestamo) references
   prestamo(numero_prestamo));
```


## Modificando la base de Datos: INSERT

**INSERT INTO R($A_1$,$A_2$, …, $A_n$)VALUES ($v_1$,$v_2$, …, $v_n$)**

- Ejemplo:

```
INSERT INTO
   pelicula (id , titulo, agno, puntuacion, votos)
VALUES (1,'Star Wars',1977,8.9,14182);
```

- Los valores pueden provenir de una consulta
- Se puede omitir la lista de atributos si se suministran todos :

```
INSERT INTO pelicula
VALUES (1,'Star Wars',1977,8.9,14182);
```

- en el **mismo orden** en que se definieron
- No es imprescindible proveer valores para todos los atributos
  - La tupla creada tendra el valor por defecto o NULL para todos aquellos atributos a los que no se les asigne un valor
  - De todas formas conviene usar DEFAULT, NULL cuando no se suministre un valor

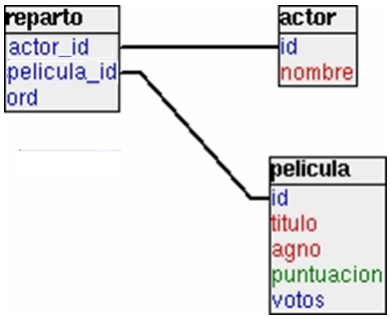
# Sistemas Informáticos

Tema 3.2: SQL

## SQL, Data Manipulation Language (DML)
## Lenguaje interactivo de Manipulación de Datos


### Ejemplo: Base de Datos de Películas

- Para rellenar la base se ha utilizado información proveniente de “The internet Movie- Database” http://www.imdb.com/list
- Todos los datos son anteriores a 1997
- Todas las películas tienen al menos 200 votos
- Los actores deben aparecer en más de una película




## Consultas sencillas en SQL

```
SELECT
FROM
WHERE
ORDER BY
```

- Lista los atributos (`SELECT`) pertenecientes a una (o más relaciones) (`FROM`) que satisfagan una condición (`WHERE`), ordenar la salida (`ORDEN BY`).
- SQL admite el duplicado de tuplas


## Comparación de caracteres

- Se pueden utilizar las expresiones algebraicas "usuales" '< ', '>', '= ', …
- o la instrucción "`like`" que permite el uso de patrones ("wildcards")
  - Comodines: '_' (un carácter) and '%' (varios caracteres)
- Encontrar todas las películas (y su puntuación) que empiezan por 'Star'

```
SELECT titulo, puntuacion
FROM pelicula
WHERE titulo LIKE 'Star%';
```

- Encontrar todas las películas con 's en su titulo

```
SELECT titulo
FROM pelicula
WHERE titulo LIKE '%''s%';
```

```
peliculas=> SELECT titulo, puntuacion
peliculas-> FROM pelicula
peliculas-> WHERE titulo LIKE 'Star%';
                  titulo                  | puntuacion
------------------------------------------+------------
 Star Wars                                |        8.9
 Star Trek: First Contact                 |        8.2
 Star Trek: The Wrath of Khan             |        7.5
 Starship Troopers                        |        7.1
 Star Trek: Generations                   |          7
 Stargate                                 |        6.9
 Star Trek IV: The Voyage Home            |        7.4
 Star Trek: The Motion Picture            |        5.7
 Star Trek III: The Search for Spock      |        6.2
 Star Trek VI: The Undiscovered Country   |        7.3
 Starman                                  |        6.9
 Star Trek: The Next Generation - All Good Things...  |        8.9
 Star Trek: The Next Generation - Encounter at Farpoint  |        7.5
 Star Trek V: The Final Frontier          |        4.7
(14 rows)
```

## [NOT] SIMILAR TO

- Tiene todas las funcionalidades de LIKE y además es capaz de usar (un subconjunto de) expresiones regulares
  - | una de dos alternativas
  - * repetición de lo anterior cero o más veces.
  - + repetición de lo anterior una o más veces.
  - () se usa para agrupar creando una único objeto.
  - [...] especifica una clase.


## Ejemplo: SIMILAR

- Encontrar todas las películas (y su puntuación) que contengan los caracteres 'Star' y no sean de la saga "Star Treck"

```
SELECT titulo, puntuacion
 FROM pelicula
 WHERE titulo NOT SIMILAR TO
          '%(S|s)tar [A-z]rek%' AND
          titulo SIMILAR TO '%Star%';
```

```
peliculas=>   SELECT titulo, puntuacion
peliculas->    FROM pelicula
peliculas->    WHERE titulo NOT SIMILAR TO
peliculas->             '%(S|s)tar [A-z]rek%' AND
peliculas->             titulo SIMILAR TO '%Star%';
                                  titulo                                  | puntuacion
--------------------------------------------------------------------------+------------
 Star Wars                                                                |        8.9
 Starship Troopers                                                        |        7.1
 Lone Star                                                                |        8.4
 Stargate                                                                 |        6.9
 Dark Star                                                                |        7.9
 Starman                                                                  |        6.9
 Last Starfighter, The                                                    |        6.7
                                       (7 rows)
```

## Producto(s) Cartesiano(s)

- Gran parte de la potencia de las bases relacionales se basa en la posibilidad de combinar dos (o más) relaciones.
- El producto cartesiano de dos relaciones se consigue enumerando cada relación en la orden FROM
- Obtener el reparto de 'Pulp Fiction' (el identificador de esta película es el 2)

```
SELECT nombre
FROM actor, reparto
WHERE pelicula_id=2 AND actor_id=id;
```


## Producto(s) Cartesiano(s)

```
peliculas=> select * from pelicula where id = 2;
 id |     titulo                    | agno | puntuacion | votos
----+-------------------------------+------+------------+-------
  2 | Pulp Fiction                  | 1994 |        8.4 | 11693
(1 row)

peliculas=> select * from reparto where pelicula_id =2;
 pelicula_id | actor_id | ord
-------------+----------+-----
           2 |      180 |  43
           2 |      257 |   2
           2 |      959 |  26
           2 |     1033 |  12
           2 |     1071 |   9
           2 |     1099 |   8
           2 |     1479 |  27
           2 |     2196 |  33
           2 |     2567 |  20
           2 |     2581 |  14
           2 |     2778 |   7
           2 |     3308 |  45
           2 |     4520 |   4
           2 |     4612 |  21
           2 |     4708 |  46
           2 |     4789 |  36
           2 |     4846 |  37
           2 |     5161 |   3
           2 |     5181 |  47
           2 |     5328 |  39
           2 |     5639 |  35
           2 |     6702 |  11
           2 |     6816 |  41
           2 |     7006 |  44
           2 |     7183 |  31
           2 |     7502 |  30
           2 |     7657 |  13
           2 |     7810 |   5
           2 |     8148 |  23
           2 |     8483 |   1
           2 |     8735 |  15
           2 |     8767 |  38
           2 |     8838 |  10
(33 rows)
```

```
peliculas=> SELECT nombre
peliculas->       FROM actor, reparto
peliculas->       WHERE pelicula_id=2 AND
peliculas->             actor_id=id;
              nombre
-------------------------------------
 Alexis Arquette
 Amanda Plummer
 Brenda Hillhouse
 Bronagh Gallagher
 Bruce Willis
 Burr Steers
 Christopher Walken
 Don Blakely
 Eric Clark
 Eric Stoltz
 Frank Whaley
 Harvey Keitel
 John Travolta
 Josef Pilato
 Julia Sweeney
 Karen Maruyama
 Kathy Griffin
 Laura Lovelace
 Lawrence Bender
 Linda Kaye
 Maria de Medeiros
 Paul Calderon
 Peter  Greene (I)
 Quentin Tarantino
 Rich Turner
 Robert Ruth
 Rosanna Arquette
 Samuel L. Jackson
 Steve Buscemi
 Tim Roth
 Uma Thurman
 Venessia Valentino
 Ving Rhames
(33 rows)
```

## Producto Natural

- NATURAL JOIN
- Campos con el mismo nombre en las tablas con las que se realiza el producto natural.
- **Suponiendo que el campo clave de actor fuera actor_id:**
- Obtener el reparto de 'Pulp Fiction' (el identificador de esta película es el 2)

```
SELECT nombre
FROM actor NATURAL JOIN reparto
WHERE pelicula_id=2;
```


## Más sobre JOIN

- Toma dos relaciones y devuelve otra relación
- Estas operaciones están típicamente en la parte FROM
- Condición en Join: define qué tuplas de las dos relaciones se corresponden, y cuáles serán los atributos que estarán en la relación resultante.
- Tipo de Join: define cómo las tuplas de una relación que no tiene correspondencia en la otra relación (según la condición de join) son tratadas

| Tipo de Join |
|---|
| **inner join** |
| **left outer join** |
| **right outer join** |
| **full outer join** |

| Condición en Join |
|---|
| **natural** |
| **on** <predicado> |
| **using** ($A_1$, $A_2$, ..., $A_n$) |

## Más sobre JOIN

- Relación *loan*

| *loan-number* | *branch-name* | *amount* |
|---|---|---|
| L-170 | Downtown | 3000 |
| L-230 | Redwood | 4000 |
| L-260 | Perryridge | 1700 |

- Relación *borrower*

| *customer-name* | *loan-number* |
|---|---|
| Jones | L-170 |
| Smith | L-230 |
| Hayes | L-155 |

- Atención: no hay información en borrower para L-260 y no hay información en loan para L-155


## Más sobre JOIN

- *loan* **inner join** *borrower* **on** *loan.loan-number = borrower.loan-number*

| *loan-number* | *branch-name* | *amount* | *customer-name* | *loan-number* |
|---|---|---|---|---|
| L-170 | Downtown | 3000 | Jones | L-170 |
| L-230 | Redwood | 4000 | Smith | L-230 |

- *loan* **left outer join** *borrower* **on** *loan.loan-number = borrower.loan-number*

| *loan-number* | *branch-name* | *amount* | *customer-name* | *loan-number* |
|---|---|---|---|---|
| L-170 | Downtown | 3000 | Jones | L-170 |
| L-230 | Redwood | 4000 | Smith | L-230 |
| L-260 | Perryridge | 1700 | *null* | *null* |

## Más sobre JOIN

- *loan* **natural inner join** *borrower*

| *loan-number* | *branch-name* | *amount* | *customer-name* |
|---|---|---|---|
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |

- loan **natural right outer join** *borrower*

| *loan-number* | *branch-name* | *amount* | *customer-name* |
|---|---|---|---|
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |
| L-155 | null | null | Hayes |


## Más sobre JOIN

- *loan* **full outer join** *borrower* **using** *(loan-number)*

| *loan-number* | *branch-name* | *amount* | *customer-name* |
|---|---|---|---|
| L-170 | Downtown | 3000 | Jones |
| L-230 | Redwood | 4000 | Smith |
| L-260 | Perryridge | 1700 | *null* |
| L-155 | null | null | Hayes |

## Combinando Consultas

- Union: `unión`
- Intersect: `intersección`
- Except: `resta`
- Estos operadores eliminan los duplicados
  - Si se usa ALL los duplicados no se eliminan: e.g., UNION ALL
- Las subconsultas deben ser compatibles

- Ejemplo: Actores comunes a las películas Star Trek IV y Star Trek V


## Ejemplo de combinación de Consultas

```
(SELECT nombre FROM pelicula,actor,reparto WHERE titulo LIKE 'Star
  Trek V:%' AND pelicula_id=pelicula.id AND actor_id=actor.id)
  INTERSECT (SELECT nombre FROM pelicula,actor,reparto WHERE titulo
  LIKE 'Star Trek IV:%'  AND pelicula_id=pelicula.id AND
  actor_id=actor.id );
```

```
peliculas=> (SELECT nombre FROM pelicula,actor,reparto WHERE titulo LIKE 'Star Trek
V:%' AND pelicula_id=pelicula.id AND actor_id=actor.id) INTERSECT (SELECT nombre
FROM pelicula,actor,reparto WHERE titulo LIKE 'Star Trek IV:%'  AND
pelicula_id=pelicula.id AND actor_id=actor.id );
              nombre
-----------------------------------
 DeForest Kelley
 George Takei
 James Doohan
 Leonard Nimoy
 Nichelle Nichols
 Walter Koenig
 William Shatner
(7 rows)
```

# Sistemas Informáticos

Tema 3.2: SQL

## Subconsultas


## Subconsultas

- Hasta ahora las condiciones en `WHERE` involucraban valores escalares
- Pero puede que aparezca `SELECT` como parte de la condición descrita en `WHERE`
  - Esta subconsulta puede devolver un valor
  - O puede devolver una relación que será procesada valor por valor usando IN, EXISTS, ALL, ANY, BETWEEN...

## Subconsultas que Involucran Escalares

- Reparto de 'Star Wars' que tiene id 1 (utilizamos dos tablas)

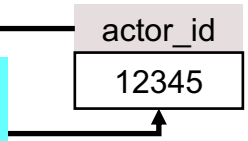
```
SELECT nombre
FROM actor, reparto

WHERE pelicula_id=1 AND actor_id=actor.id;
```

```
        nombre
-------------------------------------
 Alec Guinness
 Angus MacInnes
 Anthony Daniels
 Carrie Fisher
 David Prowse
 Denis Lawson
 Don  Henderson (II)
 Garrick Hagon
 Harrison Ford
 Jack Purvis
 Jeremy Sinden
 Kenny  Baker (I)
 Leslie Schofield
 Mark Hamill
 Peter  Mayhew (II)
 Peter Cushing
 Richard Le Parmentier
 Shelagh Fraser
 William Hootkins
(19 rows)
```


## Subconsultas que Involucran Escalares

- En lugar de realizar un producto escalar podemos hacer primero una “subconsulta” que nos de los identificadores de los actores (utilizamos una tabla inicialmente)

```
SELECT nombre
FROM actor
WHERE  id =
  (SELECT DISTINCT actor_id
  FROM reparto
  WHERE pelicula_id = 1
  );
```




Run-time error, si se genera más de una tupla

## Subconsultas que Involucran Escalares

- Lo anterior sería…

```
SELECT nombre
FROM actor
WHERE  id IN
     (SELECT DISTINCT actor_id
     FROM reparto
     WHERE pelicula_id = 1
     );
```


## El ejemplo con 3 tablas

- Utilizando 3 tablas

```
SELECT nombre
FROM actor
WHERE  id IN
     (SELECT DISTINCT actor_id
     FROM reparto
     WHERE pelicula_id =
               (SELECT DISTINCT id
               FROM pelicula
               WHERE titulo  = 'Star Wars')
     );
```

## Condiciones que Involucran Relaciones

1. `EXISTS` ***R*** : TRUE si R es una relación no vacía (un valor o una lista con varios valores)
2. ***s*** `IN` ***R*** : TRUE si s esta en R

Sea ***s*** un escalar , ***R*** debe ser una relación compuesta por un único atributo

3. ***s*** **op** `ALL` ***R*** , op = {<,>,<>,=, …}: TRUE si s es mayor, menor, etc que TODOS los valores de R.
4. ***s*** **op** `ANY` ***R*** : TRUE si s es mayor, menor, … que al menos un valor de R


## Ejemplo de (not) exists

- Lista de actores que no actúan en ninguna película

```
SELECT nombre
FROM actor
WHERE  NOT EXISTS
    (SELECT actor_id
    FROM reparto
    WHERE actor_id = actor.id
    );
```

## Operadores en Subconsultas

SELECT ... WHERE ... at < ALL/ANY (subconsulta)

≤ ALL/ANY

> ALL/ANY

≥ ALL/ANY

= ALL/ANY

<> ALL/ANY

- Se puede preceder con NOT, e.g.: SELECT ... WHERE ... NOT (at < ALL/ANY ...)
- "at" puede ser tanto un escalar como una tupla.


## Operadores en Subconsultas: IN

- Asumiendo que una subconsulta produzca varias tuplas
- Si la tupla t tiene el mismo número de atributos que una relación R podemos comprobar si t está contenida en R usando el operador IN
- SELECT ... WHERE ... at IN (subconsulta)
- se puede usar en conjunción con NOT: SELECT ... WHERE ... at NOT IN …
- Forma General:

SELECT ... WHERE ... at IN (subconsulta)
donde "at" puede ser tanto un escalar como una tupla.

## Ejemplo con dos versiones

- Películas protagonizadas por 'Harrison Ford'
- **Opción 1:** Usando el producto escalar

```
SELECT titulo
 FROM actor, pelicula, reparto
 WHERE nombre = 'Harrison Ford' AND actor.id=actor_id AND
  pelicula_id=pelicula.id;
```

- **Opción 2:** Usando consultas anidadas y el operador `IN`

```
SELECT titulo
FROM pelicula
WHERE id IN
   (SELECT pelicula_id
    FROM reparto
    WHERE (actor_id) IN
        (SELECT id
        FROM actor
        WHERE nombre = 'Harrison Ford'
        )
   );
```


```
peliculas=> SELECT titulo
peliculas-> FROM pelicula
peliculas-> WHERE id IN
peliculas-> (SELECT pelicula_id
peliculas(>  FROM reparto
peliculas(>  WHERE (actor_id) IN
peliculas(> (SELECT id
peliculas(> FROM actor
peliculas(> WHERE nombre = 'Harrison Ford'
peliculas(> )
peliculas(> );
                                titulo
-----------------------------------------------------------------------
 Star Wars
 Blade Runner
 Empire Strikes Back, The
 Raiders of the Lost Ark
 Apocalypse Now
 Indiana Jones and the Last Crusade
 Indiana Jones and the Temple of Doom
 Witness
 Air Force One
 American Graffiti
 Patriot Games
 Working Girl
 Sabrina
 Presumed Innocent
 Devil's Own, The
 Conversation, The
 Frantic
 Mosquito Coast, The
 Regarding Henry
 Force 10 from Navarone
(20 rows)
```

## Subconsultas que se ejecutan varias veces

- Hasta la fecha las subconsultas se han evaluado una sola vez. Pero esto no es necesariamente así. Es posible evaluar la subpregunta por cada valor del atributo/s que se compare
- Chequeo de consistencia (buscar actores que están repetidos)

```
SELECT Star1.nombre, Star1.id
FROM actor Star1, actor Star2
 WHERE Star1.nombre = Star2.nombre
 AND Star1.id < Star2.id;
```

```
        nombre                |  id
--------------------------------------+------
 John Lurie                           | 4432
 Marc  Lawrence (I)                   | 5565
                (2 rows)
```

## Cuidado!!!

- Chequeo de consistencia

```
SELECT nombre, id
FROM actor Star1
WHERE nombre = ANY
        (SELECT nombre
         FROM actor
         WHERE id < Star1.id AND
            nombre = Star1.nombre
);
```

tuplas que pertenecen a la relación Star1

tuplas que pertenecen a la relacion actor

Para cada iteración de la consulta "principal" se realiza la subconsulta

## Subconsultas en FROM

- También es posible usar una subconsulta en lugar de una relación ya almacenada como entrada de `FROM`
  - Es obligatorio proporcionar un alias
- Películas estrenadas en 1978 por orden de número de actores en el reparto

```
SELECT titulo, cc
FROM (SELECT pelicula_id, COUNT(actor_id)
      AS cc FROM reparto, pelicula
      WHERE agno=1978 AND pelicula_id=id
      GROUP BY pelicula_id) AS Temp, pelicula
WHERE pelicula_id=id
ORDER BY cc;

COUNT: devuelve el numero de tuplas
```


# Sistemas Informáticos

Tema 3.2: SQL

## Agrupaciones y funciones de agregación

## Agregación

- Operaciones que calculan un valor único a partir de una columna de valores
  - SUM, AVG, MIN, MAX, COUNT
- *¿Cuántas películas, actores, relaciones hay en la base?*

```
SELECT  COUNT(*)
  FROM pelicula;
```

- Si queremos estar seguros de que no contamos el mismo actor dos veces

```
SELECT  COUNT(nombre) FROM actor;
SELECT  COUNT(DISTINCT nombre)FROM actor;
```


## Agregación (ii)

- A menudo no queremos agrupar TODA la columna
- *Listado de actores ordenado por numero de veces que son estrellas*
- Queremos una salida como:

| nombre | Count (ord=1) |
|---|---|
| Pedro Pérez | 1 |
| Joe Dalton | 3 |

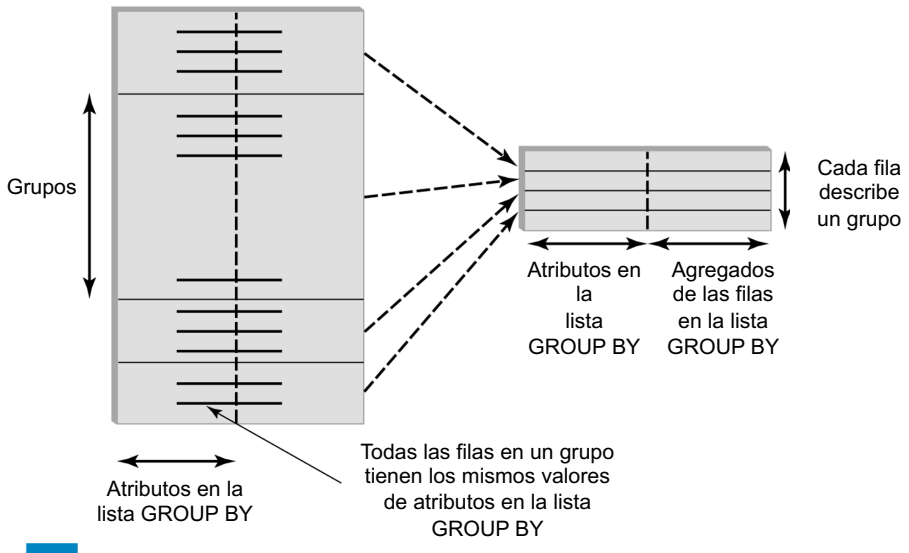
- Para ello se usa GROUP BY seguido de una lista con los atributos a agrupar

## GROUP BY

```
SELECT A1, SUM(A2)
FROM R1
GROUP BY A1;
```

- Muy importante!! SELECT tiene dos clases de términos
  (1) Agregaciones
  (2) Atributos que aparecen en la orden GROUP BY
- Cuando la orden SELECT tiene agregados SOLO aquellos atributos que se mencionan en GROUP BY pueden aparecer no agregados.


## GROUP BY - Semántica




## Ejemplo GROUP BY

- *Listado de actores ordenado por numero de veces que son estrellas (solo para aquellos con mas de 10 estrellatos).*
- Empezamos por: ¿de dónde salen los datos?

```
SELECT actor_id, pelicula_id
 FROM reparto
 WHERE ord=1
```


## Ejemplo GROUP BY

- *Listado de actores ordenado por numero de veces que son estrellas (solo para aquellos con mas de 10 estrellatos)*
- Seguimos por: ¿qué nos interesa?

```
SELECT actor_id, count(pelicula_id) AS stars
   FROM reparto
   WHERE ord=1
   GROUP BY actor_id
```

## Ejemplo GROUP BY

- *Listado de actores ordenado por numero de veces que son estrellas (solo para aquellos con mas de 10 estrellatos)*
- Y al final

```
SELECT nombre, stars
FROM actor, (SELECT actor_id, count(pelicula_id) AS stars
   FROM reparto
   WHERE ord=1
   GROUP BY actor_id) AS ranking
WHERE id=actor_id and stars > 10
ORDER BY stars;
```


## Ejemplo GROUP BY

- *Listado de actores ordenado por numero de veces que son estrellas (solo para aquellos con mas de 10 estrellatos)*
- Usando VIEW

```
CREATE VIEW ranking AS
   SELECT actor_id, count(pelicula_id) AS stars
   FROM reparto
   WHERE ord=1
   GROUP BY actor_id;

SELECT nombre, stars
FROM actor, ranking
WHERE id=actor_id and stars > 10
ORDER BY stars;
```

## HAVING

- Permite seleccionar grupos en base a alguna propiedad del grupo

```
SELECT x, sum(y)
FROM test1
GROUP BY x
HAVING sum(y) > 3;
```


## Ejemplo HAVING

- La misma consulta anterior
- Empezamos por actores > 10 estrellatos

```
SELECT …
   FROM reparto
   WHERE ord=1
   GROUP BY actor_id
   having count(pelicula_id)>10;
```

## Ejemplo HAVING

- La misma consulta anterior
- Extraemos la información deseada (el nombre)

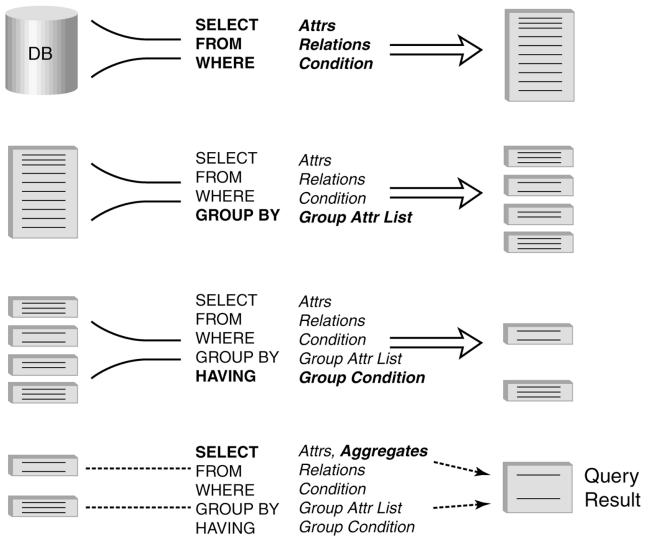
```
SELECT nombre, count(pelicula_id) AS stars
   FROM reparto, actor
   WHERE ord=1 and reparto.actor_id=actor.id
   GROUP BY nombre
   having count(pelicula_id)>10;
```


## Sintaxis de una consulta SQL

SELECT } obligatorios
FROM }

Si aparecen deben hacerlo en este orden {
WHERE
GROUP BY
HAVING
ORDER BY

## Procesamiento de una consulta con funciones de agregación





## Procesamiento de una Consulta

1. Crear el producto descrito en `FROM`
2. Aplicar las restricciones descritas en `WHERE`
3. Si no hay “group-by”, proyectar la relación (2) tal y como describa `SELECT`. Fin.
4. En caso contrario agrupar las tuplas por valores tal y como este especificado por `GROUP-BY`
5. Aplíquese `HAVING`
6. Aplíquese `SELECT`. Fin.