

# Bases de Datos

Clase 5: SQL

# Hasta ahora

- Tenemos un lenguaje teórico para realizar consultas a relaciones
- Queremos un programa con tablas y un lenguaje de consultas para utilizar en la práctica.
- Conocemos los conceptos de:
  - Esquema
  - Instancia
  - Relación
  - Atributo
  - Tupla
  - llave

¿Qué es una RDBMS?

**Relational**

**Data**

**Base**

**Management**

**System**



¿Cómo funciona un  
RDBMS?

# DBMS Relacional

- Un DBMS relacional es un programa que se instala en un computador (**servidor**)
- Este programa se mantiene escuchando conexiones
- El usuario (generalmente otro programa) se conecta (**cliente**) al programa y le puede entregar instrucciones.

# DBMS Relacional



# DBMS Relacional



<https://www.imdb.com/title/tt6723592/>



# DBMS Relacional



# DBMS Relacional



# DBMS Relacional



# DBMS Relacional



```
SELECT name, score  
FROM titles  
WHERE title.id='tt6723592'
```

# DBMS Relacional



name	score
Tenet	7.4

# DBMS Relacional



name	score
Tenet	7.4

¿Qué es SQL?

# SQL

Structured Query Language

- Usado para todas las comunicaciones con bases de datos relacionales.
- Aplicaciones web, locales, móviles, análisis de datos, etc.
- Hasta algunas arquitecturas serverless lo requieren o usan [lenguajes basados en SQL](#).



# SQL

Structured Query Language

- Softwares implementan “subconjunto” del estándar (cada uno tiene diferencias sutiles)
- Lenguaje declarativo

# Declarativo vs. Procedural

- SQL es declarativo, decimos lo que queremos, pero sin dar detalles de cómo lo computamos
- El DBMS transforma la consulta SQL en un algoritmo ejecutado sobre un lenguaje procedural
- Un lenguaje como Java es procedural: para hacer algo debemos indicar paso a paso el procedimiento

# SQL

Structured Query Language

- DDL: Lenguaje de definición de datos
  - Crear y modificar tablas, atributos y llaves
- DML: Lenguaje de manipulación de datos
  - Consultar una o más tablas
  - Insertar, eliminar, modificar tuplas

# En este curso

Durante el curso vamos a usar un motor RDBMS:

- PostgreSQL (PSQL)



# PSQL



PSQL es un sistema relacional *open source*

Tiene varias funcionalidades avanzadas, como por ejemplo el uso de procedimientos almacenados o el almacenamiento de JSON

# PSQL



PSQL va a ser usado en el proyecto y parte de la cátedra

Cada grupo tendrá acceso a un servidor en el que dispondrán de una base de datos a la que su aplicación se va a conectar

Los datos son almacenados en múltiples archivos en carpetas ocultas del computador

Continuemos con SQL

# SQL

## Tipos de datos

- Caracteres (Strings)
  - `char(20)` - Largo fijo
  - `varchar(20)` - Largo variable
- Números
  - `int`, `smallint`, `float`, ...
- Tiempo y fecha
  - `time` - hora formato 24 hrs.
  - `date` - fecha
  - `timestamp` - fecha + hora
- ¡Y varios otros! Dependen del RDBMS que se esté usando.



# SQL

Creando un esquema

Consideremos de ejemplo:

*Peliculas(id, nombre, año, categoria, calificacion, director)*

*Actores(id, nombre, edad)*

*Actuo\_en(id\_actor, id\_pelicula)*

Cómo se llama esto?

id	nombre	año	categoria	calificacion	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografia	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Crear Tablas

**CREATE TABLE** Peliculas(

id **int**,

nombre **varchar**(30),

año **int**,

categoria **varchar**(30),

calificacion **float**,

director **varchar**(30)

)

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Crear Tablas

id	nombre	edad
1	Leonardo DiCaprio	41
2	Matthew McConaughey	46
3	Daniel Radcliffe	27
4	Jessica Chastain	39

CREATE TABLE Peliculas(

id int,

nombre varchar(30),

año int,

categoría varchar(30),

calificación float,

director varchar(30)

)

CREATE TABLE Actores(

id int,

nombre varchar(30),

edad int

)

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Crear Tablas

id	nombre	edad
1	Leonardo DiCaprio	41
2	Matthew McConaughey	46
3	Daniel Radcliffe	27
4	Jessica Chastain	39

CREATE TABLE Peliculas(

id int,

nombre varchar(30),

año int,

categoría varchar(30),

calificación float,

director varchar(30)

)

CREATE TABLE Actores(

id int,

nombre varchar(30),

edad int

)

CREATE TABLE Actuo\_en(

id\_actor int,

id\_pelicula int,

)

id_actor	id_pelicula
1	2
2	1
4	1
3	3
1	5

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Crear Tablas con llaves

id	nombre	edad
1	Leonardo DiCaprio	41
2	Matthew McConaughey	46
3	Daniel Radcliffe	27
4	Jessica Chastain	39

CREATE TABLE Peliculas(

id int PRIMARY KEY,  
 nombre varchar(30),  
 año int,  
 categoría varchar(30),  
 calificación float,  
 director varchar(30)

)

CREATE TABLE Actores(

id int PRIMARY KEY,  
 nombre varchar(30),  
 edad int

)

id_actor	id_pelicula
1	2
2	1
4	1
3	3
1	5

CREATE TABLE Actuo\_en(

id\_actor int,  
 id\_pelicula int,  
 PRIMARY KEY (id\_pelicula, id\_actor)

)

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

SQL  
Valores  
Default

## Sintaxis general:

CREATE TABLE <Nombre> (...<atr> tipo DEFAULT <valor>...)

## Ejemplo:

```
CREATE TABLE Peliculas(
  id int PRIMARY KEY,
  nombre varchar(30),
  año int,
  categoria varchar(30) DEFAULT 'Acción',
  calificacion float DEFAULT 0,
  director varchar(30)
)
```

# SQL

## Modificar Tablas

Eliminar tabla:

`DROP TABLE Peliculas`

Eliminar atributo:

`ALTER TABLE Peliculas DROP COLUMN director`

Agregar atributo:

`ALTER TABLE Peliculas ADD COLUMN productor varchar(30)`

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Insertar Datos

Sintaxis general:

```
INSERT INTO R(at_1, ..., at_n) VALUES (v_1, ..., v_n)
```

Ejemplo:

```
INSERT INTO Peliculas(id, nombre, año, categoría, calificación, director) VALUES
(321351, 'V for Vendetta', 2005, 'Action', 8.2, 'James McTeigue')
```

Ejemplo abreviado (asume orden de creación):

```
INSERT INTO Peliculas VALUES (321351, 'V for Vendetta', 2005, 'Action',
8.2, 'James McTeigue')
```



# Llaves Foráneas en SQL

## Inserciones con llaves foráneas

¿Qué pasa en este caso?

```
CREATE TABLE R(a int, b int, PRIMARY KEY(a));  
CREATE TABLE S(a int, c int, FOREIGN KEY(a) REFERENCES R, ...);  
INSERT INTO R VALUES(1, 1);  
INSERT INTO S VALUES(1, 2);
```

Todo bien hasta ahora...

# Llaves Foráneas en SQL

## Inserciones con llaves foráneas

¿Qué pasa en este caso?

```
CREATE TABLE R(a int, b int, PRIMARY KEY(a));  
CREATE TABLE S(a int, c int, FOREIGN KEY(a) REFERENCES R, ...);  
INSERT INTO R VALUES(1, 1);  
INSERT INTO S VALUES(1, 2);  
INSERT INTO S VALUES(2, 3);
```



**ERROR!**

La base de datos no permite que se agreguen filas en que la llave foránea no está en la tabla referenciada!

Consultando con SQL

# SQL

Forma básica

Las consultas en general se ven:

# SQL

Forma básica

Las consultas en general se ven:

**SELECT** atributos

**FROM** relaciones

**WHERE** condiciones

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Forma básica

Para ver todo de una tabla (en este caso película):

```
SELECT * FROM Peliculas
```

Para ver nombre y calificación de todas las películas dirigidas por Nolan:

```
SELECT nombre, calificacion
FROM Peliculas
WHERE director = 'C. Nolan'
```

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Forma básica

Para las películas estrenadas desde el 2010:

```
SELECT *
FROM Peliculas
WHERE año >= 2010
```

El **WHERE** permite =, <>, !=, >, <, <=, >=, AND, OR, NOT, IN, BETWEEN, etc...

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Forma básica

El **WHERE** permite =, <>, !=, >, <, <=, >=, AND, OR, NOT, IN, BETWEEN, etc...

Películas estrenadas entre 1971 y 1978:

```
SELECT *
FROM Peliculas
WHERE año BETWEEN 1971 AND 1978
```

Películas estrenadas en 1971, 1973 y 2001:

```
SELECT *
FROM Peliculas
WHERE año IN (1971, 1973, 2001)
```



# SQL

En General

La consulta:

```
SELECT a_1, ..., a_n  
FROM T_1, ..., T_m  
WHERE <condicion>
```

Se traduce al álgebra relacional como:

$$\pi_{a_1, \dots, a_n}(\sigma_{condiciones}(T_1 \times \dots \times T_m))$$

# Update

Para actualizar valores de una tabla:

UPDATE Peliculas

SET calificacion = 0

WHERE nombre = 'Sharknado 6'

id	nombre	año	categoria	calificacion	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Sharknado 6	2010	Terror	8.8	A. Ferrante



id	nombre	año	categoria	calificacion	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Sharknado 6	2010	Terror	0	A. Ferrante

# Update

Forma general

UPDATE R

SET <Nuevos valores>

WHERE <Condición sobre R>

<Nuevos valores>  $\rightarrow$  (atributo<sub>1</sub> = nuevoValor<sub>1</sub>, ..., atributo<sub>n</sub> = nuevoValor<sub>n</sub>)

# Delete

Para borrar filas que cumplan una condición:

**DELETE FROM R**

**WHERE** <Condición sobre R>

**HICE UN DELETE**



**Y OLVIDÉ PONER EL  
WHERE**

[HTTP://CODERFACTS.COM](http://CODERFACTS.COM)

[https://www.youtube.com/watch?v=i\\_cVJglz\\_Cs](https://www.youtube.com/watch?v=i_cVJglz_Cs)

¿Qué pasa si se nos olvida el `WHERE` en un `UPDATE` o `DELETE FROM`?

`UPDATE Users`

`SET password = 'contraseña'`

`DELETE FROM Tweets`

... ¿O si se nos pasa un `;` entre medio?

`UPDATE Users`

`SET password = 'contraseña';`

`WHERE email = 'some@email.com'`

`DELETE FROM Tweets;`

`WHERE user_name = 'realDonaldTrump'`

¿Qué pasa si se nos olvida el **WHERE** en un **UPDATE** o **DELETE FROM**?  
... ¿O si se nos pasa un ';' entre medio?

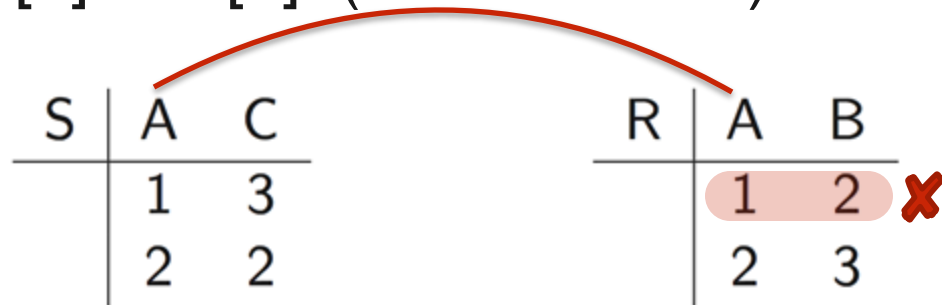
**Se borran / actualizan todas las filas!!**



# Llaves Foráneas en SQL

Eliminar con llaves foráneas

Tenemos  $S[a] \subseteq R[a]$  (llave foránea)



S	A	C
	1	3
	2	2

R	A	B
	1	2
	2	3

¿Qué ocurre al eliminar (1, 2) en **R**?

**ERROR!**

La base de datos no permite eliminar llaves foráneas usadas en otras tablas.



# Llaves Foráneas en SQL

Eliminar con llaves foráneas

¿Qué ocurre al eliminar (1, 2) en **R**?



Tenemos las siguientes opciones:

- No permitir eliminación
- Propagar la eliminación y también borrar (1,3) de S
- Mantener la tupla en **S** pero dejar en la llave foránea el valor en null.

# Llaves Foráneas en SQL

Eliminar con llaves foráneas

¿Qué ocurre al eliminar (1, 2) en **R**?

S	A	C
	1	3
	2	2

R	A	B
	1	2
	2	3

Opción 1: no permitir la eliminación. **Default en SQL!**

```
CREATE TABLE R(a int, b int, PRIMARY KEY(a))
```

```
CREATE TABLE S(a int, c int, FOREIGN KEY(a) REFERENCES R, ...)
```

**Respuesta: obtenemos error**

# Llaves Foráneas en SQL

Eliminar con llaves foráneas

¿Qué ocurre al eliminar (1, 2) en **R**?

S	A	C
	1	3
	2	2

R	A	B
	1	2
	2	3

Opción 2: Propagar la eliminación. (**Cascada de eliminaciones**)

```
CREATE TABLE R(a int, b int, PRIMARY KEY(a))
```

```
CREATE TABLE S(a int, c int,
```

```
FOREIGN KEY(a) REFERENCES R ON DELETE CASCADE, ...)
```

**Respuesta: se elimina también (1, 3) en S**

# Llaves Foráneas en SQL

Eliminar con llaves foráneas

¿Qué ocurre al eliminar (1, 2) en **R**?

S	A	C
	1	3
	2	2

R	A	B
	1	2
	2	3

Opción 3: **Dejar en nulo**

```
CREATE TABLE R(a int, b int, PRIMARY KEY(a))
```

```
CREATE TABLE S(a int, c int,
```

```
FOREIGN KEY(a) REFERENCES R ON DELETE SET NULL, ...)
```

**Respuesta: la tupla (1, 3) en S ahora es (null, 3)**

# Producto cruz

Si pedimos datos de más de una tabla la base de datos va hacer un producto cruz y entregará  **$n \times m$**  filas.

**SELECT \***

**FROM Peliculas, Actuo\_en**

id	nombre	año	categoria	calificacion	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

id_actor	id_pelicula
1	2
2	1
4	1
3	3
1	5

id	nombre	año	categoria	calificacion	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Joins

id_actor	id_pelicula
1	2
2	1
4	1
3	3
1	5

Podemos hacer un join agregando un **WHERE**

Por ejemplo, para obtener todas las películas junto a los ids de los actores que participaron en ella:

**SELECT \***

**FROM** Peliculas, Actuo\_en

**WHERE** id = id\_pelicula

**Observación:** id es atributo de Peliculas, mientras que id\_pelicula es atributo de Actuo\_en

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Joins

—

id_actor	id_pelicula
1	2
2	1
4	1
3	3
1	5

Desambiguando atributos

Entregue todas las películas junto a los id de los actores que participaron en ella:

**SELECT \***

**FROM** Peliculas, Actuo\_en

**WHERE** Peliculas.id = Actuo\_en.id\_pelicula

Sirve cuando tenemos atributos en distintas tablas con el mismo nombre y para agregarle claridad a la consulta.

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Joins

id	nombre	edad
1	Leonardo DiCaprio	41
2	Matthew McConaughey	46
3	Daniel Radcliffe	27
4	Jessica Chastain	39

id_actor	id_pelicula
1	2
2	1
4	1
3	3
1	5

¿Y si queremos los nombres de los actores en vez de los ids?

```
SELECT Peliculas.nombre, Actores.nombre
FROM Peliculas, Actuo_en, Actores
WHERE Peliculas.id = Actuo_en.id_pelicula
AND Actores.id = Actuo_en.id_actor
```



id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Alias

id	nombre	edad
1	Leonardo DiCaprio	41
2	Matthew McConaughey	46
3	Daniel Radcliffe	27
4	Jessica Chastain	39

id_actor	id_pelicula
1	2
2	1
4	1
3	3
1	5

Podemos acortar la consulta anterior:

```
SELECT p.nombre, a.nombre
FROM Peliculas as p, Actuo_en as ae, Actores as a
WHERE p.id = ae.id_pelicula
AND a.id = ae.id_actor
```

Ese tipo de alias no es muy recomendable

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Alias

Podemos hacer operaciones y nombrar la columna:

```
SELECT (nombre || ' dirigida por ' || director) as credits, año
FROM Peliculas
```

credits	año
Interstellar dirigida por C.Nolan	2014

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Ordenando

Entregue el nombre y la calificación de todas las películas (orden ascendente):

```
SELECT nombre, calificacion
FROM Peliculas
ORDER BY nombre, calificacion
```

El i-ésimo atributo del ORDER BY resuelve un empate en el atributo i-1

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

## Ordenando

Entregue el nombre y la calificación de todas las películas (orden descendente):

**SELECT** nombre, calificacion

**FROM** Peliculas

**ORDER BY** nombre **DESC**, calificacion

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

SQL  
Union

id	nombre	edad
1	Leonardo DiCaprio	41
2	Matthew McConaughey	46
3	Daniel Radcliffe	27
4	Jessica Chastain	39

Entregue el nombre de todos actores y directores:

```
SELECT nombre
FROM Actores
UNION
SELECT director
FROM Peliculas
```

# SQL

## Operadores de conjuntos

- **EXCEPT**: diferencia del álgebra
- **UNION**: unión del álgebra
- **INTERSECT**: intersección del álgebra
- **UNION ALL**: unión que admite duplicados

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

# SQL

Matching de patrones con LIKE

s **LIKE** p: string s es como p, donde p es un patrón definido mediante:

- **%** Cualquier secuencia de caracteres
- **\_** Cualquier caracter (solamente uno)

**SELECT \***

**FROM Peliculas**

**WHERE name LIKE '%Harry Potter%'**

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	SciFi	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	Harry Potter	2011	Fantasia	8.1	D. Yates
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh
5	Inception	2010	Adventure	8.8	C. Nolan

**SQL**  
Eliminando  
duplicados

Entregue todos los nombres distintos de las películas:

```
SELECT DISTINCT nombre
FROM Peliculas
```

OJO: **DISTINCT** es un operador en sí mismo.



# Agregación

¿Qué hace esta consulta?

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

También podemos usar SUM, MIN, MAX, COUNT,  
etc.

# Agregación

¿Qué diferencia estas consultas?

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

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

**COUNT(\*)** cuenta los nulos. Más de eso en un rato

Ojo: En ambas se cuentan los duplicados

# Ejemplo

Compra(producto, fecha, precio, cantidad)

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

SELECT COUNT(\*)  
FROM Compra

**5**

SELECT COUNT(cantidad)  
FROM Compra

**3**

# Ejemplo

Compra(producto, fecha, precio, cantidad)

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

**SELECT** DISTINCT **COUNT**(cantidad)  
**FROM** Compra

**3**

**SELECT** **COUNT**(DISTINCT cantidad)  
**FROM** Compra

**2**

# GROUP BY

```
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

```

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

```

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

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

```

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

## 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

```

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

3) Agregar por grupo y ejecutar la proyección

producto	ventaTotal
tomates	1200
zapallo	2800

# HAVING

## G

Misma consulta, pero sólo queremos 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
- 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

# Consultas Anidadas

- Como ya habíamos visto con las operaciones de conjuntos, una consulta puede estar constituida por operaciones entre consultas.
- Pero esa no es la única forma, SQL nos ofrece mucho más.

# Consultas Anidadas

Como condición

Consideremos este esquema:

*Bandas(nombre, vocalista, ...)*

*Estudiantes\_UC(nombre, ...)*

*Toco\_en(nombre\_banda, nombre\_festival)*

Obtengamos todas las bandas cuyo vocalista sea un estudiante UC y que hayan tocado en lollapalooza

# Consultas Anidadas

Como condición

```
SELECT Bandas.nombre  
FROM Bandas, Estudiantes_UC  
WHERE Bandas.vocalista = Estudiantes_UC.nombre  
AND Bandas.nombre IN (  
    SELECT Toco_en.nombre_banda  
    FROM Toco_en  
    WHERE Toco_en.nombre_festival = 'Lollapalooza'  
)
```

Comprobamos que Bandas.nombre esté dentro del listado de bandas que han tocado en Lollapalooza.

# Consultas Anidadas

Como condición

Si la sub consulta retorna un escalar podemos usar los operadores condicionales típicos. Si no podemos hacerlo agregando un operador adicional.

- $s \text{ IN } R$
- $s > \text{ALL } R$  (no disponible en SQLite)
- $s > \text{ANY } R$  (no disponible en SQLite)
- $\text{EXISTS } R$



# Consultas Anidadas

ALL, ANY

*Cervezas(nombre, precio, ...)*

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

Cervezas que no son 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**<sup>1</sup>. Una consulta con **ALL** no es monótona. Una consulta con **ANY** lo es.

<sup>1</sup>**Una consulta es "monótona" si agregar más datos nunca reduce el conjunto de resultados que produce.**

# Sub Consultas Relacionadas

Nombres de películas que se repiten en años diferentes

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

La sub consulta depende de la externa!

# Consultas Anidadas

Como Joins

El nombre de cada actor junto con el total de películas en las que ha actuado.

```
SELECT Actores.nombre, agg.cuenta
FROM Actores, (
    SELECT id_actor as id, COUNT(*) as cuenta
    FROM Actuo_en
    GROUP BY id_actor
) as agg
WHERE Actores.id = agg.id
```

# Consultas Anidadas

Como Joins

El nombre de cada actor junto con el año de la primera película en la que actuó.

```
SELECT Actores.nombre, agg.año
FROM Actores, (
    SELECT Actuo_en.id_actor as id, MIN(Peliculas.año) as año
    FROM Actuo_en, Peliculas
    WHERE Actuo_en.id_pelicula = Peliculas.id
    GROUP BY id_actor
) as agg
WHERE Actores.id = agg.id
```

¿Qué pasa si queremos el nombre de la película además del año?

# Información Incompleta

- En una base de datos real, muy seguido no tendremos los datos para llenar todas las columnas al agregar una fila.
- También puede ser que por la lógica del problema, que un campo esté vacío tenga una semántica relevante para la aplicación.
- Con SQL podemos modelar la falta de información mediante nulos (**NULL**).
- Los nulos en las tablas generan ciertos comportamientos extraños que es bueno tener en cuenta al trabajar con ellos. Los discutiremos en esta clase.

# Información Incompleta

## Peliculas

nombre	director	actor
Django sin cadenas	Tarantino	Di Caprio
Django sin cadenas	Tarantino	Waltz
null	Tarantino	Thurman
null	Tarantino	null
El Hobbit	Jackson	McKellen
Señor de los Anillos	null	McKellen

¿Qué significan los nulos en este caso?



# Nulos

## Significado

En el caso anterior puede significar que no se dispone de la información, pero existe!

En general los nulos pueden significar:

- Valor existe, pero no tengo la información
- Valor no existe (si premios = null la información no existe)
- Ni siquiera sé si el valor existe o no

# Nulos

Consultando con nulos

Sea la relación **R**(a, b), las consultas:

- `SELECT * FROM R`
- `SELECT * FROM R WHERE R.b = 3 OR R.b <> 3`

¿Son lo mismo?

Si `R.b` es nulo, ni `R.b = 3` o `R.b <> 3` evalúan a verdadero

# Nulos

Consultando con nulos

La consulta:

```
SELECT * FROM R
```

Equivale a la unión de:

- ```
SELECT * FROM R WHERE R.b = 3
```
- ```
SELECT * FROM R WHERE R.b <> 3
```
- ```
SELECT * FROM R WHERE R.b IS NULL
```

Para ver si un elemento es nulo usamos **IS NULL**

Para ver si un elemento no es nulo usamos **IS NOT NULL**

# Nulos

## Operaciones con nulos

Si algún argumento de una operación aritmética es nulo, el resultado es nulo

# Nulos

## Operaciones con nulos

Sean las siguientes instancias de **R** y **S**:

| R | A    | S | B |
|---|------|---|---|
|   | 1    |   | 2 |
|   | null |   | 3 |

La consulta **SELECT R.A + S.B FROM R, S** retorna:

| Respuesta | $R.A + S.B$ |
|-----------|-------------|
|           | 3           |
|           | 4           |
|           | null        |
|           | null        |

# Nulos

## Operaciones con nulos

Sean las siguientes instancias de **R** y **S**:

| R | A    | S | B |
|---|------|---|---|
|   | 1    |   | 2 |
|   | null |   | 3 |

Tenemos que  $R.A = S.B$  vale:

- FALSE cuando  $R.A = 1$  y  $S.B = 2$
- UNKNOWN cuando  $R.A = \text{null}$  y  $S.B = 2$

# Lógica de tres valores

SQL usa lógica de tres valores:

| x       | NOT x   |
|---------|---------|
| true    | false   |
| false   | true    |
| unknown | unknown |

# Nulos

Ejemplo

| R | A | S | A |
|---|---|---|---|
|   | 1 |   | 1 |
|   | 2 |   | 2 |
|   |   |   | 3 |
|   |   |   | 4 |

```
SELECT S.A FROM S  
WHERE S.A NOT IN (SELECT R.A FROM R)
```

Resultado: {3, 4}



# Nulos

Ejemplo

| R | A    | S | A |
|---|------|---|---|
|   | 1    |   | 1 |
|   | 2    |   | 2 |
|   | null |   | 3 |
|   |      |   | 4 |

```
SELECT S.A FROM S  
WHERE S.A NOT IN (SELECT R.A FROM R)
```

Resultado: tabla vacía!

# Lógica de tres valores

El resultado puede ser contraintuitivo pero es correcto

Veamos la evaluación de `3 NOT IN (SELECT R.A FROM R)`

```
3  NOT IN  {1, 2, null}
    =      NOT (3 IN {1, 2, null})
    =      NOT (3 = 1 OR 3 = 2 OR 3 = null)
    =      NOT (false OR false OR unknown)
    =      NOT (unknown)
    =      unknown
```

# Lógica de tres valores

Además 1 NOT IN (SELECT R.A FROM R) es FALSE (Lo mismo para 2)

Para el valor 4 aplicamos el razonamiento anterior.

Resultado: tabla vacía

# Nulos

Agregación

| A    |
|------|
| 1    |
| null |

`SELECT COUNT(*) FROM R` vale 2

`SELECT COUNT(R.A) FROM R` vale 1

# Nulos

## Agregación

`SELECT SUM(R.A) FROM R` retorna 1 si  $R.A = \{1, \text{null}\}$

Para funciones de agregación:

- Ignore todos los nulos
- Compute el valor de la agregación

La única excepción es `COUNT(*)`

# Nulos

Al crear una tabla o agregar una columna podemos incluir una restricción para que no permita NULLs

```
CREATE TABLE <nombre> (...  
    <atributo> <tipo> NOT NULL,  
    ...)
```

# Inner Joins

Usualmente hacemos JOINS, especificando en la sentencia **FROM** de la consulta las tablas que queremos usar y en el **WHERE** las condiciones.

```
SELECT *  
FROM Peliculas, Actuo_en  
WHERE id = id_pelicula
```

Pero SQL tiene una sintaxis mucho más clara para expresar un join normal: **INNER JOIN** (alias **JOIN**)

# Inner Joins

Estas 3 consultas son equivalentes:

```
SELECT *  
FROM Peliculas, Actuo_en  
WHERE id = id_pelicula
```

```
SELECT *  
FROM Peliculas JOIN Actuo_en  
ON id = id_pelicula
```

```
SELECT *  
FROM Peliculas INNER JOIN Actuo_en  
ON id = id_pelicula
```



# Outer Joins

Consideremos estas tablas:

| Estudios |           | Películas  |         |
|----------|-----------|------------|---------|
| Nombre   | Titulo    | Titulo     | Ingreso |
| Warner   | Argo      | Argo       | 136     |
| Warner   | El Origen | El Origen  | 292     |
| MGM      | El Hobbit | El Artista | 44      |

Escribamos una consulta que liste los ingresos totales de cada estudio.

# Outer Joins

```
SELECT Estudio.nombre, SUM(Pelicula.ingreso)
FROM Estudio JOIN Pelicula
ON Estudio.titulo = Pelicula.titulo
GROUP BY Estudio.nombre
```

- Resultado: (Warner, 428)

¿Cuál es el problema de esta consulta?

El estudio MGM, cuyas películas no tenemos información va a desaparecer por no tener contraparte en el **JOIN**.

# Outer Joins

Lo solucionamos con un Outer Join izquierdo, que mantiene las tuplas sin pareja de la primera tabla:

```
SELECT Estudio.nombre, SUM(Pelicula.ingreso)
FROM Estudio LEFT OUTER JOIN Pelicula
ON Estudio.titulo = Pelicula.titulo
GROUP BY Estudio.nombre
```

- Resultado: (Warner, 428), (MGM, null)

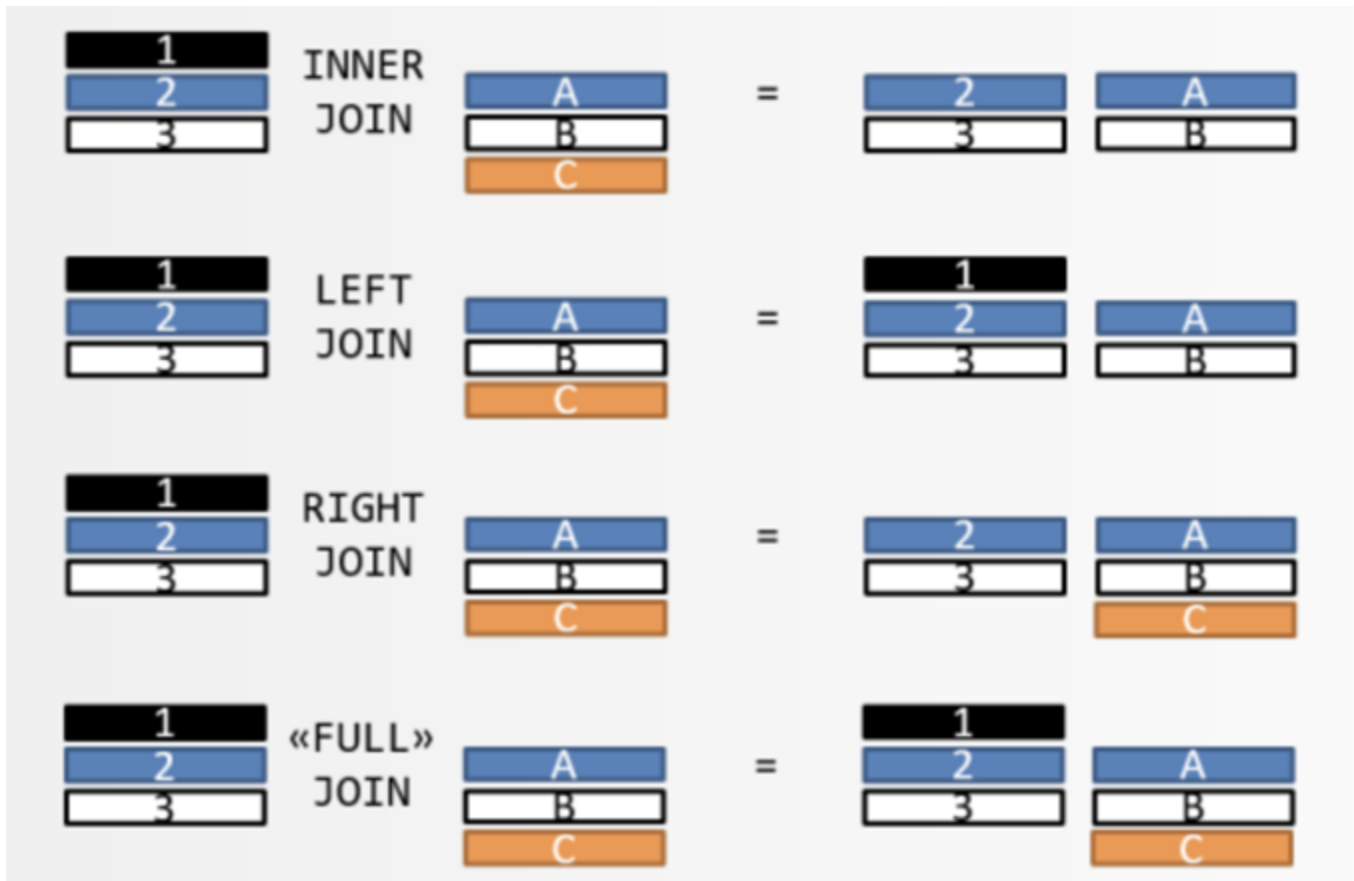
# Left Outer Join

```
SELECT *  
FROM Estudio LEFT OUTER JOIN Pelicula  
ON Estudio.titulo = Pelicula.titulo
```

| nombre | titulo    | titulo    | ingreso |
|--------|-----------|-----------|---------|
| Warner | Argo      | Argo      | 136     |
| Warner | El Origen | El Origen | 292     |
| MGM    | El Hobbit | null      | null    |

# Outer Joins

- **R LEFT OUTER JOIN S**: mantenemos las tuplas de **R** que no tienen correspondencia.
- **R RIGHT OUTER JOIN S**: mantenemos las tuplas de **S** que no tienen correspondencia.
- **R FULL OUTER JOIN S**: mantenemos las tuplas de **R** y **S** que no tienen correspondencia



# Redundancia en SQL

Recordemos esta consulta:

```
SELECT Bandas.nombre  
FROM Bandas, Estudiantes_UC  
WHERE Bandas.vocalista = Estudiantes_UC.nombre  
AND Bandas.nombre IN (  
    SELECT Toco_en.nombre_banda  
    FROM Toco_en  
    WHERE Toco_en.nombre_festival = 'Lollapalooza'  
)
```

```
SELECT Bandas.nombre
FROM Bandas, Estudiantes_UC
WHERE Bandas.vocalista = Estudiantes_UC.nombre
AND Bandas.nombre IN (
    SELECT Toco_en.nombre_banda
    FROM Toco_en
    WHERE Toco_en.nombre_festival = 'Lollapalooza'
)
```

```
SELECT DISTINCT Bandas.nombre
FROM Bandas, Estudiantes_UC, Toco_en
WHERE Bandas.vocalista = Estudiantes_UC.nombre
AND Banda.nombre = Toco_en.nombre_banda
AND Toco_en.nombre_festival = 'Lollapalooza'
```

```
SELECT Bandas.nombre
FROM Bandas, Estudiantes_UC
WHERE Bandas.vocalista = Estudiantes_UC.nombre
INTERSECT
SELECT Toco_en.nombre_banda
FROM Toco_en
WHERE Toco_en.nombre_festival = 'Lollapalooza'
```

Son todas equivalentes!



# Redundancia en SQL

¿Cómo saber cuál usar?

- Dada la naturaleza declarativa de SQL, es muy difícil predecir cómo diferentes formas de hacer una consulta puede tener un mejor rendimiento al ser ejecutadas por el RDBMS.
- Una aplicación real necesita hacer muchas consultas junto con código procedural para realizar su tarea. Cada consulta implica conectarse con la base de datos, y las conexiones tienen un *overhead* de tiempo adicional.
- Por eso en la práctica generalmente optimizamos el [número de consultas](#) a hacer, más que cómo están escritas esas consultas.

¿Dudas?