



ESCUELA DE INGENIERÍA
FACULTAD DE INGENIERÍA

EDUCACIÓN
PROFESIONAL

Gestión de datos SQL I

Maximiliano Arancibia y Matías Toro
Educación Profesional - Escuela de Ingeniería

El uso de apuntes de clases estará reservado para finalidades académicas. La reproducción total o parcial de los mismos por cualquier medio, así como su difusión y distribución a terceras personas no está permitida, salvo con autorización del autor.

- Tenemos una serie de operadores para realizar consultas a relaciones
- Queremos un programa con tablas y un lenguaje de consultas para utilizar en la práctica.



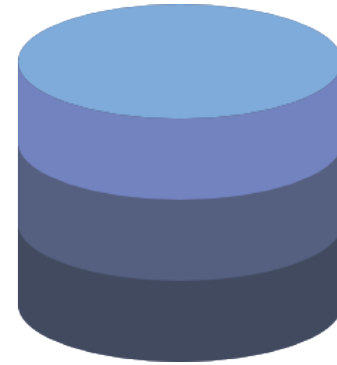
Relational

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



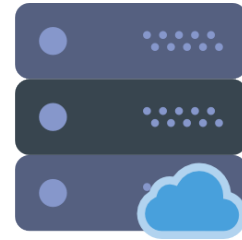
DBMS relacional



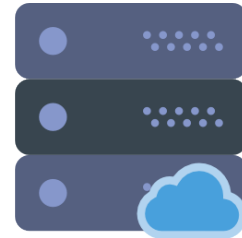
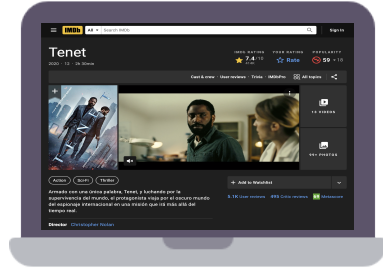
<https://www.imdb.com/title/tt6723592/>



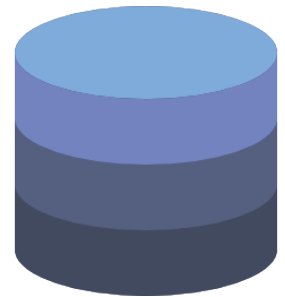
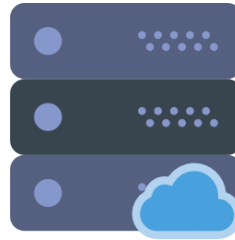
DBMS relacional



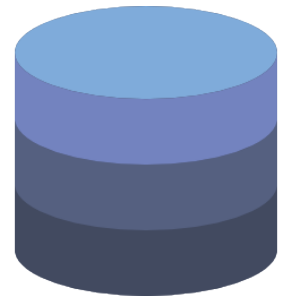
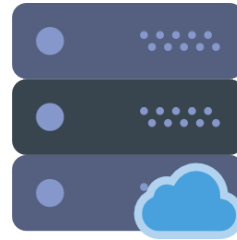
DBMS relacional



DBMS relacional



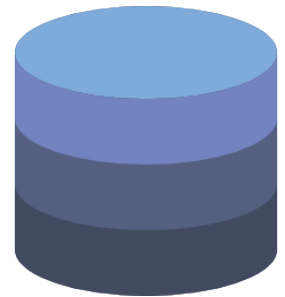
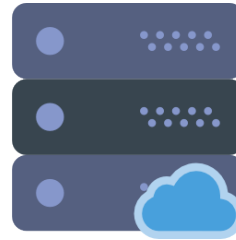
DBMS relacional



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



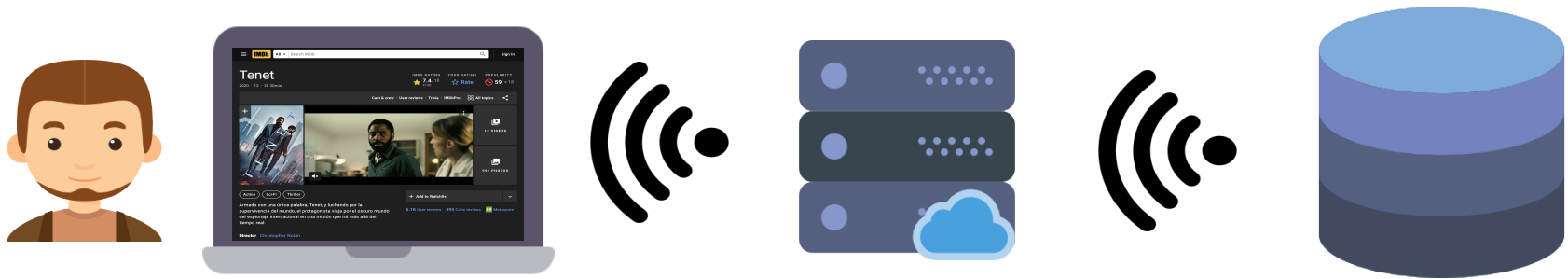
DBMS relacional



name	score
Tenet	7.4



DBMS relacional



name	score
Tenet	7.4



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.



¿Por qué queremos aprender SQL?

Los DBMS relacional persisten la información en disco duro, y son tolerantes a fallos.

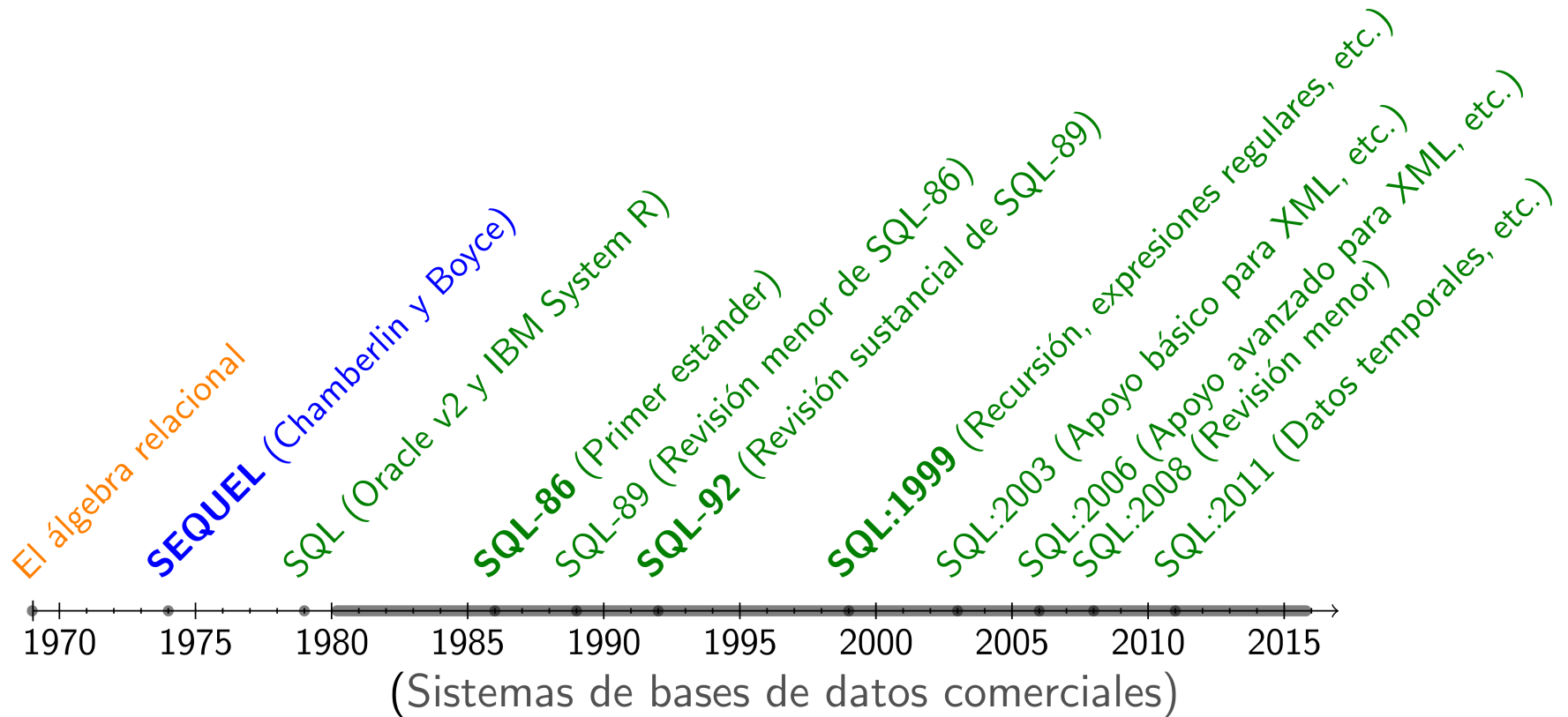
La mayoría de los DBMS en el mundo usa SQL. Saber SQL implica saber trabajar con la mayoría de las bases de datos.



La evolución de SQL



SQL



Sistemas de bases de datos (con SQL)

□ include secondary database models

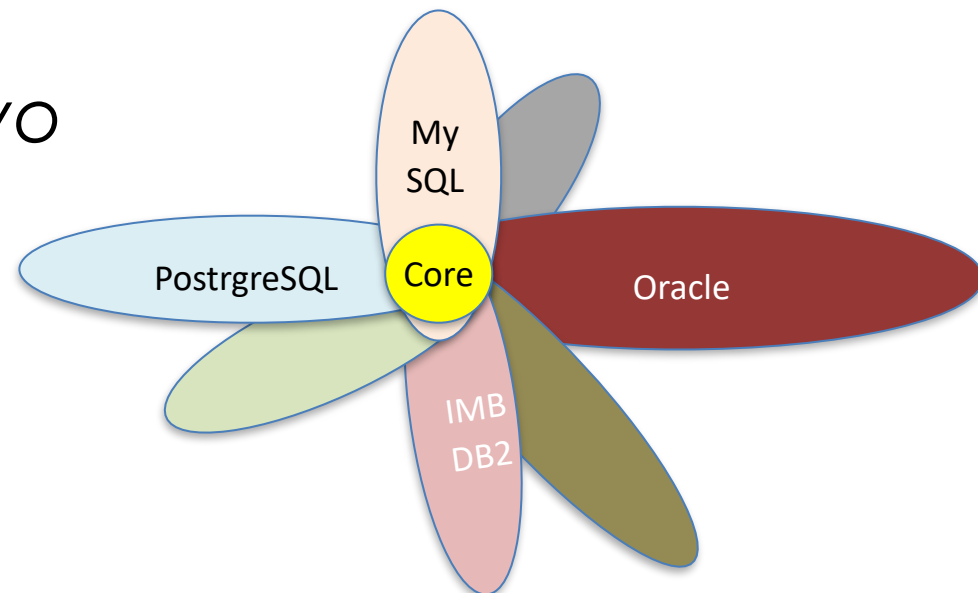
159 systems in ranking, May 2022

Rank			DBMS	Database Model	Score		
May 2022	Apr 2022	May 2021			May 2022	Apr 2022	May 2021
1.	1.	1.	Oracle +	Relational, Multi-model ⓘ	1262.82	+8.00	-7.12
2.	2.	2.	MySQL +	Relational, Multi-model ⓘ	1202.10	-2.06	-34.28
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model ⓘ	941.20	+2.74	-51.46
4.	4.	4.	PostgreSQL + ⓘ	Relational, Multi-model ⓘ	615.29	+0.83	+56.04
5.	5.	5.	IBM Db2	Relational, Multi-model ⓘ	160.32	-0.13	-6.34
6.	6.	↑ 7.	Microsoft Access	Relational	143.44	+0.66	+28.04
7.	7.	↓ 6.	SQLite +	Relational	134.73	+1.94	+8.04
8.	8.	8.	MariaDB +	Relational, Multi-model ⓘ	111.13	+0.81	+14.44
9.	9.	↑ 16.	Snowflake +	Relational	93.51	+4.06	+63.46
10.	10.	10.	Microsoft Azure SQL Database	Relational, Multi-model ⓘ	85.33	-0.45	+14.88
11.	11.	↓ 9.	Hive +	Relational	81.61	+0.18	+5.42
12.	12.	↓ 11.	Teradata +	Relational, Multi-model ⓘ	68.39	+0.82	-1.59
13.	13.	↓ 12.	SAP HANA +	Relational, Multi-model ⓘ	55.09	-0.71	+2.33
14.	14.	14.	FileMaker	Relational	52.27	-0.64	+5.55
15.	↑ 16.	15.	Google BigQuery +	Relational	48.61	+0.63	+10.98
16.			Databricks	Multi-model ⓘ	47.85		
17.	↓ 15.	↓ 13.	SAP Adaptive Server	Relational, Multi-model ⓘ	47.78	-0.58	-2.19
18.	18.	18.	Amazon Redshift +	Relational	25.94	-0.24	+3.43
19.	↓ 17.	↓ 17.	Firebird	Relational	25.72	-0.92	+1.29
20.	↓ 19.	↓ 19.	Informix	Relational, Multi-model ⓘ	23.37	-0.13	+0.96



SQL: Structured Query Language

- Último estándar SQL99 (SQL3)
- 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 **R** es **procedural**: para hacer algo debemos **indicar paso a paso el procedimiento**



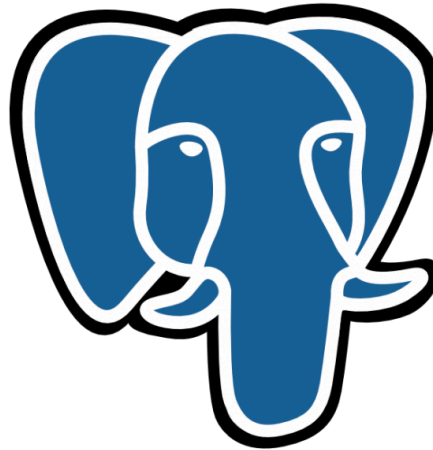
SQL: Structured Query Language

- **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



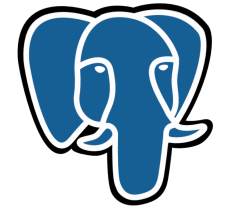
En este curso

Durante el curso vamos a usar un motor RDBMS:



PostgreSQL





PSQL es un sistema **relacional** *open source*

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

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



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: DDL - Creando un Esquema

Consideremos el siguiente esquema:

```
Películas(id: int, nombre: string, año: int,  
          categoría: string, calificación: float,  
          director: string)
```

```
Actor(id: int, nombre: string, edad: int)
```

```
Actuó_en(id_actor: int, id_película: int)
```



SQL: DDL - Crear Tablas

Películas(id: int, nombre: string, año: int,
categoría: string, calificación: float, director: string)

```
CREATE TABLE Peliculas (  
    id int,  
    nombre varchar(30),  
    anho int,  
    categoria varchar(30),  
    calificacion float,  
    director varchar(30)  
)
```



SQL: DDL - Crear Tablas

Actor(id: int, nombre: string, edad: int)

```
CREATE TABLE Actores (  
    id int,  
    nombre varchar(30),  
    edad int,  
)
```



SQL: DDL - Crear Tablas

Actuó_en(id_actor: **int**, id_película: **int**)

```
CREATE TABLE Actuó_en (  
    id_actor int,  
    id_película int  
)
```



SQL: DDL - Crear Tablas

Películas(id: int, nombre: string, año: int, categoría: string, calificación: float)

Actor(id: int, nombre: string, edad: int)

Actuó_en(id_actor: int, id_película: int)

```
CREATE TABLE Peliculas (  
    id int,  
    nombre varchar(30),  
    anho int,  
    categoria varchar(30),  
    calificacion float,  
    director varchar(30)  
)
```

```
CREATE TABLE Actores (  
    id int,  
    nombre varchar(30),  
    edad int,  
)
```

```
CREATE TABLE Actuo_en (  
    id_actor int,  
    id_pelicula int  
)
```

¿Algún problema?



SQL: DDL - Crear Tablas (con llaves)

Películas(id: int, nombre: string, año: int, categoría: string, calificación: float)

Actor(id: int, nombre: string, edad: int)

Actuó_en(id_actor: int, id_película: int)

```
CREATE TABLE Peliculas (  
    id int PRIMARY KEY,  
    nombre varchar(30),  
    anho int,  
    categoria varchar(30),  
    calificacion float,  
    director varchar(30)  
)
```

```
CREATE TABLE Actores (  
    id int PRIMARY KEY,  
    nombre varchar(30),  
    edad int,  
)
```

```
CREATE TABLE Actuo_en (  
    id_actor int,  
    id_pelicula int,  
    PRIMARY KEY (id actor, id pelicula)  
)
```




SQL: DDL - Crear Tablas (con llaves)

Películas(id: int, nombre: string, año: int, categoría: string, calificación: float)

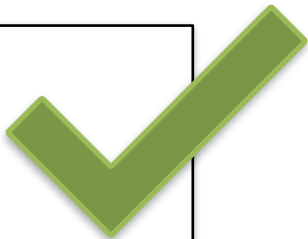
Actor(id: int, nombre: string, edad: int)

Actuó_en(id_actor: int, id_película: int)

```
CREATE TABLE Actuó_en(  
    id_actor int PRIMARY KEY,  
    id_película int PRIMARY KEY  
)
```



```
CREATE TABLE Actuó_en(  
    id_actor int,  
    id_película int,  
    PRIMARY KEY (id_actor, id_película)  
)
```



SQL: DDL - Valores Default

Sintaxis general:

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

Ejemplo:

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



SQL: DDL - 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);
```



Consultando con SQL



SQL: Forma básica

Las consultas en general se ven:

```
SELECT atributos  
FROM relaciones  
WHERE condiciones
```



SQL: Forma básica

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

```
SELECT * FROM  
Peliculas
```

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	Fantasía	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	The Imitation Game	2014	Biografía	8.1	M. Tyldum
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh



SQL: Forma básica - proyectando datos

Para ver nombre y calificación de todas las películas:

```
SELECT nombre, calificacion  
FROM Peliculas
```

nombre	calificacion
Interstellar	8.6
The Revenant	8.1
The Imitation Game	8.1
The Theory of Everything	7.7



SQL: Forma básica

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

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

nombre	calificacion
Interstellar	8.6



SQL: Forma básica

Nota: las comillas simples denotan un string (palabra); las dobles denotan el nombre de una tabla o atributo.

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



Error: no existe la columna C. Nolan



SQL: Forma básica

Para las películas estrenadas desde el 2010:

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

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



SQL: Forma básica

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

Películas dirigidas por Nolan **no en el 2020**:

```
SELECT *  
FROM Peliculas  
WHERE director = 'C. Nolan' AND  
anho <> 2020
```

```
SELECT *  
FROM Peliculas  
WHERE director = 'C. Nolan' AND  
NOT anho = 2020
```



SQL: Forma básica

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

Películas dirigidas por Nolan o Tarantino:

```
SELECT *  
FROM Peliculas  
WHERE director = 'C. Nolan' OR  
director = 'Q. Tarantino'
```



SQL: Forma básica

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

Películas estrenadas entre 1971 y 1978:

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



SQL: Forma básica

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

Películas estrenadas en 1971, 1973 y 2001:

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



SQL: Forma básica

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

Películas por Nolan o Tarantino, estrenadas en 1971, 1973 y 2001:

```
SELECT *  
FROM Peliculas  
WHERE (director = 'C. Nolan' OR  
director = 'Q. Tarantino') AND  
anho IN (1971, 1973, 2001)
```



SQL: Forma básica - Duplicados

Para ver categorías de todas las películas:

```
SELECT categoria  
FROM Peliculas
```

categoria
Fantasía
Drama
Biografía
Biografía

¿Algún problema?



SQL: Forma básica - distinto

Para ver categorías de todas las películas:

```
SELECT DISTINCT categoria  
FROM Peliculas
```

categoria
Fantasía
Drama
Biografía

¿Que piensan ustedes?
¿Son útiles los duplicados?



SQL: Forma general

Las consultas en general se ven:

```
SELECT atributos  
FROM relaciones  
WHERE condiciones
```



SQL: 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	categoría	calificación	director
1	Interstellar	2014	Fantasía	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	The Imitation Game	2014	Biografía	8.1	M. Tyldum
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh

id_actor	id_pelicula
1	2
2	1
...	...



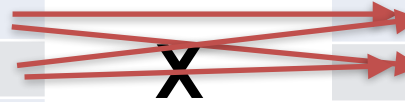
SQL: Producto cruz

SELECT *

FROM Peliculas, Actuo_en

id	nombre	año	categoría	calificación	director
1	Interstellar	2014	Fantasia	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	The Imitation Game	2014	Biografía	8.1	M. Tyldum
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh

id_actor	id_pelicula
1	2
2	1
...	...



=

id	nombre	año	categoría	calificación	director	id_actor	id_pelicula
1	Interstellar	2014	Fantasia	8.6	C. Nolan	1	2
1	Interstellar	2014	Fantasia	8.6	C. Nolan	2	1
2	The Revenant	2015	Drama	8.1	A. Iñárritu	1	2
2	The Revenant	2015	Drama	8.1	A. Iñárritu	2	1
3	The Imitation Game	2014	Biografía	8.1	M. Tyldum	1	2
3	The Imitation Game	2014	Biografía	8.1	M. Tyldum	2	1
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh	2	1
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh	1	2
...



SQL: Joins

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`



SQL: Joins

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

id	nombre	anho	categoria	calificacion	director
1	Interstellar	2014	Fantasia	8.6	C. Nolan
2	The Revenant	2015	Drama	8.1	A. Iñárritu
3	The Imitation Game	2014	Biografía	8.1	M. Tyldum
4	The Theory of Everything	2014	Biografía	7.7	J. Marsh

id_actor	id_pelicula
1	2
2	1
...	...

X

=

id	nombre	anho	categoria	calificacion	director	id_actor	id_pelicula
1	Interstellar	2014	Fantasia	8.6	C. Nolan	1	2
2	The Revenant	2015	Drama	8.1	A. Iñárritu	1	2
...



SQL: Joins

```
SELECT *  
FROM Peliculas, Actuo_en  
WHERE id = id_pelicula  
AND director = 'C. Nolan'
```

Es equivalente a:

```
SELECT *  
FROM Peliculas JOIN Actuo_en  
ON id = id_pelicula  
WHERE director = 'C. Nolan'
```



SQL: Joins - 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.



SQL: Joins

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

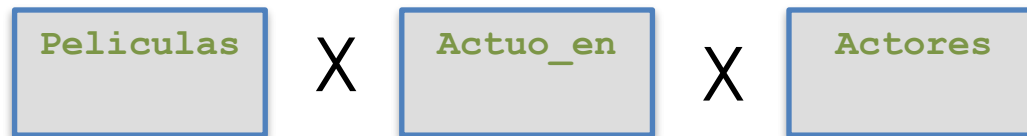
```
SELECT Peliculas.nombre, Actores.nombre  
FROM Peliculas, Actuo_en, Actores  
WHERE Peliculas.id = Actuo_en.id_pelicula  
AND Actores.id = Actuo_en.id_actor
```



SQL: Joins

¿Qué pasa si no ponemos la condición del join?

```
SELECT Peliculas.nombre, Actores.nombre  
FROM Peliculas, Actuo_en, Actores
```



!La consulta podría tomar mucho tiempo y recursos!



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



SQL: Alias

```
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
```

Es equivalente a:

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



SQL: Alias

Podemos hacer operaciones y nombrar la columna:

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

credits	anho
V for Vendetta dirigida por James McTeigue	2005
Dunkirk dirigida por C. Nolan	2017



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



SQL: Ordenando

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

nombre	calificacion
Batman	8.2
Batman	8.4
Batman	9
The Theory of Everything	7.7



SQL: Ordenando

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

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

Entrega nombres y calificaciones de películas ordenadas por nombre descendiente. Si hay empates ordena por calificación ascendente.



SQL: Ordenando

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

nombre	calificacion
The Theory of Everything	7.7
Batman	8.2
Batman	8.4
Batman	9



SQL: Ordenando

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

nombre	calificacion
The Theory of Everything	7.7
Batman	9
Batman	8.4
Batman	8.2



SQL: Union (distinta)

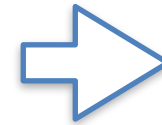
Entregue el nombre de todos actores y directores:

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

nombre
Leonardo DiCaprio
Matthew McConaughey
Daniel Radcliffe
Jessica Chastain
Q. Tarantino

UNION

director
C. Nolan
A. Iñárritu
M. Tyldum
Q. Tarantino
J. Marsh



nombre
Leonardo DiCaprio
Matthew McConaughey
Daniel Radcliffe
Jessica Chastain
Q. Tarantino
C. Nolan
A. Iñárritu
M. Tyldum
J. Marsh



SQL: Union (bruta)

Entregue el nombre de todos actores y directores:

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

nombre
Leonardo DiCaprio
Matthew McConaughey
Daniel Radcliffe
Jessica Chastain
Q. Tarantino

UNION ALL



director
C. Nolan
A. Iñárritu
M. Tyldum
Q. Tarantino
J. Marsh

nombre
Leonardo DiCaprio
Matthew McConaughey
Daniel Radcliffe
Jessica Chastain
Q. Tarantino
C. Nolan
A. Iñárritu
M. Tyldum
Q. Tarantino
J. Marsh



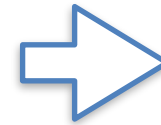
SQL: Diferencia

Entregue el nombre de todos actores que no son directores:

```
SELECT nombre  
FROM Actores  
EXCEPT  
SELECT director  
FROM Peliculas
```

nombre
Leonardo DiCaprio
Matthew McConaughey
Daniel Radcliffe
Jessica Chastain
Q. Tarantino

director
C. Nolan
A. Iñárritu
M. Tyldum
Q. Tarantino
J. Marsh



nombre
Leonardo DiCaprio
Matthew McConaughey
Daniel Radcliffe
Jessica Chastain



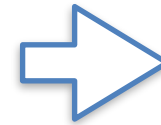
SQL: Intersección

Entregue el nombre de todos actores que son directores:

```
SELECT nombre
FROM Actores
INTERSECT
SELECT director
FROM Peliculas
```

nombre
Leonardo DiCaprio
Matthew McConaughey
Daniel Radcliffe
Jessica Chastain
Q. Tarantino

INTERSECT



director
C. Nolan
A. Iñárritu
M. Tyldum
Q. Tarantino
J. Marsh

nombre
Q. Tarantino



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 nombre
FROM Peliculas
WHERE nombre LIKE
'%Spiderman%'
```

nombre
Spiderman
Spiderman 2
Spiderman 3
The Amazing Spiderman
The Amazing Spiderman 2
Spiderman: Homecoming
Spiderman: Far From Home
Spiderman: No Way Home



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 nombre  
FROM Peliculas  
WHERE nombre LIKE  
'Spiderman%'
```

nombre
Spiderman
Spiderman 2
Spiderman 3
Spiderman: Homecoming
Spiderman: Far From Home
Spiderman: No Way Home



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 nombre
FROM Peliculas
WHERE nombre LIKE
'%Spiderman _'
```

nombre
Spiderman 2
Spiderman 3
The Amazing Spiderman 2



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 nombre  
FROM Peliculas  
WHERE nombre LIKE  
'Spiderman _'
```

nombre
Spiderman 2
Spiderman 3



SQL: Insertar Datos

Sintaxis general:

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

Ejemplo:

```
INSERT INTO Peliculas  
  (id, nombre, año, categoria, calificacion, 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')
```



SQL: Actualizar datos

Para actualizar valores de una tabla:

```
UPDATE Peliculas  
SET calificación = 0  
WHERE name = 'Sharknado 6'
```



SQL: Actualizar datos

Forma general:

```
UPDATE R  
SET <nuevos valores>  
WHERE <condición sobre R>
```

<nuevos valores> → (atributo1 = nuevoValor1, ..., atributon = nuevoValorn)



SQL: Eliminación de datos

Para borrar filas que cumplan una condición:

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



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

```
UPDATE Users  
SET password = 'contraseña'
```

```
DELETE FROM Tweets
```



El ';' indica el fin de una instrucción.

```
SELECT categoria  
FROM Peliculas  
WHERE director = 'C. Nolan'
```

!No ejecuta nada!

```
SELECT categoria  
FROM Peliculas  
WHERE director = 'C. Nolan';
```

Ahora sí

¿Qué pasa si se nos pasa un ';' entre medio?

```
UPDATE Users  
SET password = 'contraseña';  
WHERE email = 'some@email.com'
```

```
DELETE FROM Tweets;  
WHERE email = 'realDonaldTrump'
```

¿Qué pasa si se nos pasa un ';' entre medio?

!!Se borran / actualizan todas las filas!!



¿Preguntas?

