

Bases de Datos

Clase 3: SQL

Hasta ahora

- Tenemos un lenguaje teórico para realizar consultas a relaciones
- Queremos un lenguaje de consultas para utilizar en la práctica

SQL

Structured Query Language

- Último estándar SQL99 (SQL3)
- Softwares implementan subconjunto del estándar
- Lenguaje declarativo

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

Cómo funciona un DBMS Relacional

DMBS Relacional

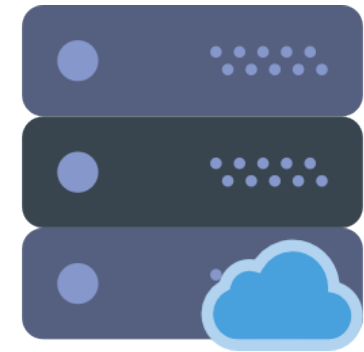
- Un DBMS relacional es un programa que se instala en un computador (**servidor**)
- Este programa se mantiene escuchando conexiones
- El usuario se conecta (**cliente**) al servidor y le entrega instrucciones a este programa

DMBS Relacional

DMBS Relacional



DMBS Relacional



DMBS Relacional



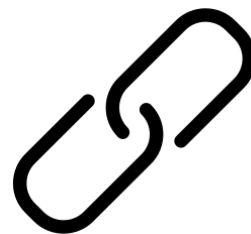
DMBS Relacional



DMBS Relacional



DMBS Relacional



DMBS Relacional

Obs. Muchas veces las consultas del cliente se realizan desde el mismo computador en donde se instaló el servidor del DBMS

Ejemplo

Aplicación Web sencilla

Para hacer *hosting* una aplicación web esta se expone en un servidor

Ese mismo servidor también puede tener corriendo un DBMS relacional

La app web va a realizar consultas al DBMS y ambas componentes estarán en el mismo computador

En este curso

Durante el curso vamos a usar dos sistemas:

- PSQL
- SQLite

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

SQLite3

Es una DBMS *self-contained* que se utiliza generalmente para desarrollo, pero **no** para aplicaciones en producción

Para utilizar SQLite no es necesario tener un cliente para conectarse

SQLite genera un archivo por cada base de datos, que se puede compartir entre usuarios

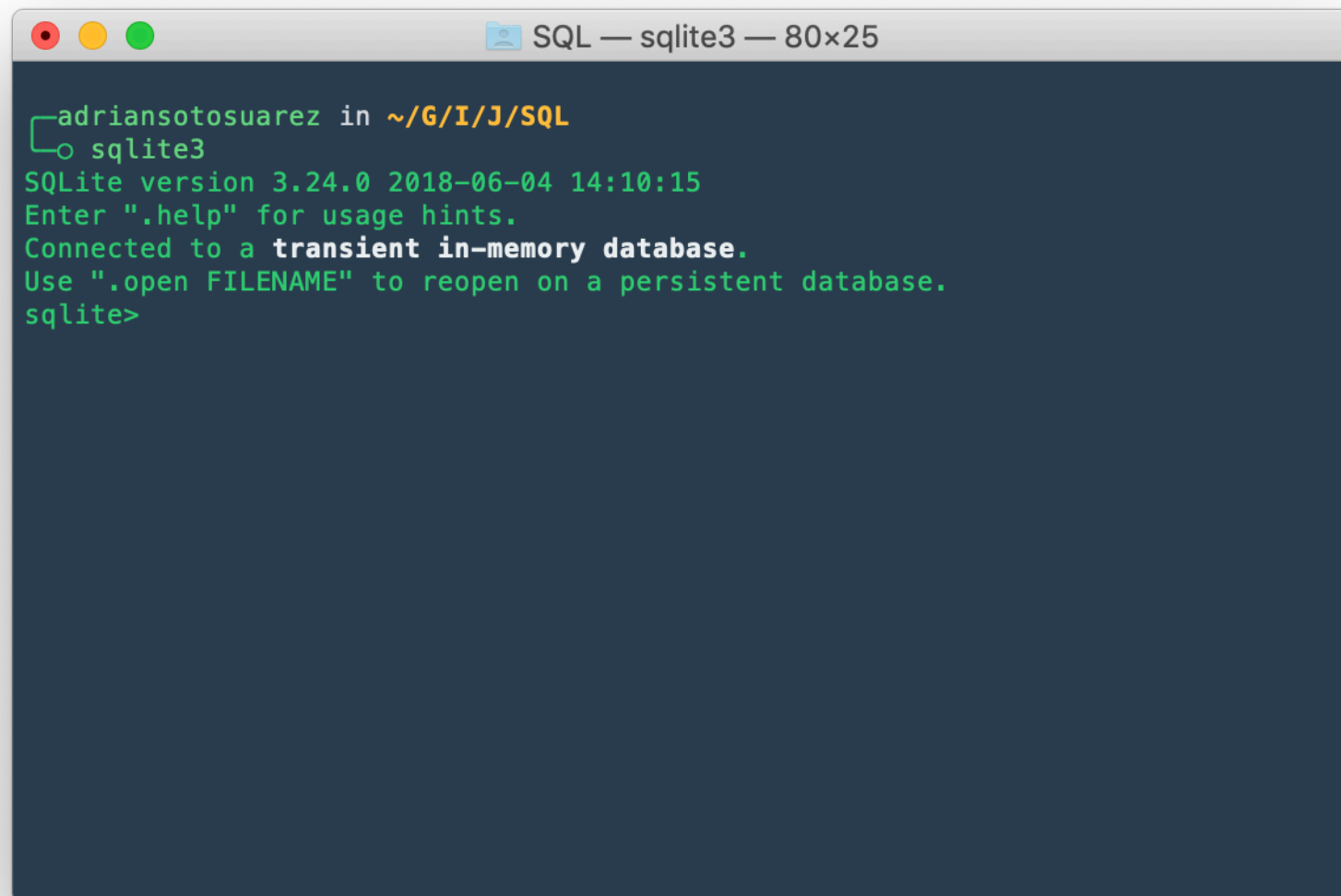
SQLite3

SQLite viene instalado en Linux y OSX y se puede instalar fácilmente en Windows

Se corre con el comando `sqlite3` en la terminal

SQLite3

Esto abre un cliente para hacer consultas:

A screenshot of a terminal window on a macOS system. The window title bar shows three colored window control buttons (red, yellow, green) on the left, followed by the text "SQL — sqlite3 — 80x25". The terminal content shows a user named "adriansotosuarez" in the directory "~/G/I/J/SQL" running the command "sqlite3". The output displays the SQLite version "3.24.0" with a timestamp "2018-06-04 14:10:15", a prompt to enter ".help" for usage hints, and a message stating it is connected to a "transient in-memory database". It also provides instructions on how to use ".open FILENAME" to connect to a persistent database. The prompt "sqlite>" is visible at the bottom of the output.

```
adriansotosuarez in ~/G/I/J/SQL
❯ sqlite3
SQLite version 3.24.0 2018-06-04 14:10:15
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

SQLite3

Pero en el curso, en general **no** utilizaremos el cliente

Vamos a trabajar en Jupyter y a utilizar la instalación de SQLite3 presente en Python

Pero hay que recordar que SQLite3 puede ser utilizado sin la necesidad de utilizar Python!

SQL

SQL

Crear Tablas

```
CREATE TABLE Cervezas (nombre char(30),  
                           tipo char(10),  
                           grados float,  
                           ciudad_origen char(30))
```

```
CREATE TABLE EnStock (nombre-cerveza char(30),  
                        cantidad int,  
                        precio-unitario float)
```


SQL

Tipos

- Caracteres (Strings)
 - `char(20)` - Largo fijo
 - `varchar(20)` - Largo variable
- Números
 - `int`, `smallint`, `float`, ...
- Tiempo y fecha
 - `time` - hora formato 24 hrs.
 - `date` - fecha, cuidado con los meses
 - `timestamp` - fecha + hora
- Otros, ver estándar

SQL

Modificar Tablas

Eliminar tabla:

```
DROP TABLE Cervezas
```

Eliminar atributo:

```
ALTER TABLE Cervezas DROP grados
```

Agregar atributo:

```
ALTER TABLE Cervezas ADD COLUMN grados
```

SQL

Insertar Datos

Sintaxis general:

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

Ejemplo:

```
INSERT INTO Cervezas(nombre, tipo, grado,  
ciudad_origen) VALUES ('Austral Lager', 'Lager',  
4.6, 'Punta Arenas')
```

Ejemplo abreviado (asume orden de creación):

```
INSERT INTO Cervezas VALUES ('Austral Lager',  
'Lager', 4.6, 'Punta Arenas')
```

SQL

Insertar Datos

Sintaxis con consulta:

```
INSERT INTO R VALUES (SELECT ... FROM ... WHERE ...)
```

SQL

Valores Default

Sintaxis con consulta:

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

Ejemplo:

```
CREATE TABLE Cervezas (nombre char(30),  
                        tipo char(10) DEFAULT 'Lager',  
                        grados float,  
                        ciudad-origen char(30))
```

SQL

Valores Default

Ejemplo:

```
INSERT INTO Cervezas VALUES  
('Austral Yagan', 'Ale', 5, 'Punta Arenas')
```

```
INSERT INTO Cervezas (nombre, grados, ciudad-origen)  
VALUES  
('Austral Lager', 4.6, 'Punta Arenas')
```

SQL

Valores Default

Ejemplo:

```
INSERT INTO Cervezas VALUES  
('Austral Yagan', 'Ale', 5, 'Punta Arenas')
```

```
INSERT INTO Cervezas (nombre, grados, ciudad-origen)  
VALUES  
('Austral Lager', 4.6, 'Punta Arenas')
```

nombre	tipo	grados	ciudad_origen
Austral Yagan	Ale	5.0	Pta. Arenas
Austral Lager	Lager	4.6	Pta. Arenas

SQL

Llaves

Sintaxis para crear llave:

```
CREATE TABLE Cervezas (nombre char(30) PRIMARY KEY,  
                           tipo char(10),  
                           grados float,  
                           ciudad-origen char(30))
```

O bien:

```
CREATE TABLE Peliculas (titulo varchar(30),  
                           año int,  
                           genero char(10),  
                           PRIMARY KEY (titulo,año))
```


Consultando con SQL

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

Forma básica

Las consultas en general se ven:

SQL

Forma básica

Las consultas en general se ven:

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

SQL

Esquema

Recordemos nuestro esquema de ejemplo:

SQL

Esquema

Recordemos nuestro esquema de ejemplo:

```
Películas(id, nombre, año, categoria, calificacion,  
director)
```

```
Actor(id, nombre, edad)
```

```
Actuo_en(id_actor, id_pelicula)
```

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'
```

SQL

Forma básica

Para las películas estrenadas desde el 2010:

SQL

Forma básica

Para las películas estrenadas desde el 2010:

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

SQL

Forma básica

Para las películas estrenadas desde el 2010:

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

En el **WHERE** aceptamos =, >, <, <=, >=, **AND**,
OR...

SQL

En General

La consulta:

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

Se traduce al álgebra relacional como:

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

SQL

Matching de patrones con LIKE

- s LIKE p:** string s es como p, donde p usa:
- % - Cualquier secuencia de caracteres
 - _ - Cualquier caracter (solamente uno)

SQL

Matching de patrones con LIKE

- s **LIKE** p: string s es como p, donde p usa:
- % - Cualquier secuencia de caracteres
 - _ - Cualquier caracter (solamente uno)

```
SELECT *  
FROM Peliculas  
WHERE director LIKE '%Nolan%'
```

SQL

Eliminando duplicados

Entregue todos los nombres distintos de las películas:

SQL

Eliminando duplicados

Entregue todos los nombres distintos de las películas:

```
SELECT DISTINCT nombre  
FROM Peliculas
```

SQL

Ordenando

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

SQL

Ordenando

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

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

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

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

SQL

Ordenando

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

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

SQL

Joins

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

SQL

Joins

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

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

SQL

Joins

Entregue todas las películas junto a los id 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 - Desambiguando atributos

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

Sirve cuando tenemos atributos en distintas tablas con el mismo nombre

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

SQL

Joins

Entregue todos los nombres de películas junto a los nombres de los actores que participaron en ella:

SQL

Joins

Entregue todos los nombres de películas junto a los nombres de los actores que participaron en ella:

```
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

Nuevo Esquema

Sea el siguiente nuevo esquema de ejemplo:

SQL

Nuevo Esquema

Sea el siguiente nuevo esquema de ejemplo:

`Persona(id, nombre, edad)`

`Amigo(id_persona1, id_persona2)`

SQL

Self Join

Entregue el nombre de todos los pares de amigos:

SQL

Self Join

Entregue el nombre de todos los pares de amigos:

```
SELECT P1.nombre, P2.nombre  
FROM Persona AS P1, Amigo, Persona AS P2  
WHERE P1.id = Amigo.id_persona1 AND P2.id =  
Amigo.id_persona2
```

SQL

Self Join

Entregue el nombre de todos los pares de amigos:

```
SELECT P1.nombre, P2.nombre  
FROM Persona AS P1, Amigo, Persona AS P2  
WHERE P1.id = Amigo.id_persona1 AND P2.id =  
Amigo.id_persona2
```

Observación: Usamos dos veces la tabla persona, renombrándola para poder hacer la distinción

SQL

Self Join

Entregue el nombre de todos los pares de amigos:

```
SELECT P1.nombre, P2.nombre  
FROM Persona P1, Amigo, Persona P2  
WHERE P1.id = Amigo.id_persona1 AND P2.id =  
Amigo.id_persona2
```

SQL

Self Join

Entregue el nombre de todos los pares de amigos:

```
SELECT P1.nombre, P2.nombre  
FROM Persona P1, Amigo, Persona P2  
WHERE P1.id = Amigo.id_persona1 AND P2.id =  
Amigo.id_persona2
```

Observación: Es posible quitar el AS

SQL

Union

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