

Introducción a las bases de datos

Exploratorio en Computación
Invitado: Valentina Álvarez

¿Por qué bases de
datos?



Don N:

- Emisor y regulador de la moneda NebCoin
- Va a abrir su propia tienda: NebStore
- No está seguro de cómo manejar sus datos
- Le encanta usar el lenguaje de programación Python



Profesor A:

- Experto en Bases de Datos
- Tiene mucha experiencia en el tema
- Gran amigo de Don N.



Profesor A, ¿por qué no hago
mi propio programa en Python
para manejar datos?



Ok Don N., déjame contarte
una historia



Cuando era joven quise hacer
una aplicación web para
vender libros



Y no quise usar un sistema de
bases de datos...

Implementación a la medida

Supongamos que queremos hacer una aplicación web para vender libros en línea. Queremos guardar:

- Libros
- Usuarios
- Compras

Tenemos un archivo para cada registro

- libros.txt
- usuarios.txt
- compras.txt

Implementación a la medida

Archivo libros.txt

```
L01 # Harry Quebert # Joël Dicker # $12.500 # 3
L02 # Cien Años de Soledad # GGM # $8.000 # 2
L03 # Origen # Dan Brown # $15.000 # 10
L04 # La Divina Comedia # Dante # $20.000 # 1
...
```

Implementación a la medida

Archivo usuarios.txt

```
U01 # Conito # XXXX 1234 # conito@uc.cl  
U02 # Don N. # XXXX 2345 # DonN@uc.cl  
U03 # Profesor A # XXXX 3456 # ProfesorA@uc.cl  
U04 # Tía Pame # XXXX 4567 # TiaPame@uc.cl  
U05 # Frambuesa # XXXX 5678 # Frambuesa@uc.cl  
...
```

Implementación a la medida

Archivo compras.txt

L01 # U02 # 10/09/2021

L01 # U04 # 14/08/2021

L02 # U02 # 07/09/2021

L04 # U01 # 27/05/2021

...



Necesitaba permitir que los
usuarios compraran libros

Implementación a la medida

Comprar un libro

Primera tarea: La tía Pame desea comprar un libro sobre los gonfoterios

Instrucciones para nuestro programa:

read usuarios.txt

find Tía Pame

read libros.txt

find Gonfoterios

update inventario

update compras.txt



Pero después aparecieron más
funcionalidades

Implementación a la medida

Segunda tarea: Un usuario quiere ver la lista de todos los libros.

Tercera tarea: Un usuario quiere ver la lista de libros que ha comprado.

...



Y descubrí que hacer todo de manera eficiente no era fácil!

Problemas

Búsqueda de datos costosa. Supongamos que:

- libros.txt pesa 16 mb
- Cada página de disco tiene 8 kb
- Archivo ocupa 2000 páginas en disco
- Cada búsqueda en disco dura toma al menos 0.1 ms (mínimo).

Buscar un libro (recorrer el archivo entero) toma al menos 0.2 segundos!



Y luego aparecieron problemas
que no tenía considerados!

Problemas

Imaginemos que dos personas compran al mismo tiempo un libro para el que queda uno solo en stock.

- ¿Qué pasa con el control de concurrencia?



Y más problemas aún !

Problemas

Imaginemos que el computador donde está corriendo la aplicación se apaga.

- ¿Qué pasa con la recuperación de fallas?



Hmm, quizás programar todo
desde 0 no sea la mejor
alternativa



Exacto!

Sistemas de Bases de Datos

Sistema de gestión de bases de datos
Database Management System
DBMS

Sistemas de Bases de Datos

Programa (por ejemplo PSQL, MySQL, MongoDB) que
sirve para administrar datos

Sistemas de Bases de Datos

- Almacenar datos (insertar)
- Encontrar datos (búsquedas y consultas)
- Modificar datos (update)
- Asegurar la consistencia de los datos
- Seguridad y privacidad de los datos

Sistemas de Bases de Datos

Los usuarios finales solamente se encargan de diseñar la estructura de los datos y las consultas

Cómo funciona un DBMS



¿Cuál es la mejor película de Christopher Nolan?



~~Interstellar~~
The Dark Knight



Desventajas de un DBMS

- Es una implementación en un lenguaje de bajo nivel (generalmente C) de las ventajas mencionadas:
 - Siempre podemos hacer una consulta más rápidamente sin DBMS
 - Pero el DBMS ofrece una gran cantidad de soluciones ya fabricadas

Donde sea que trabajen,
tendrán que interactuar con
Bases de Datos



Ok Profesor, usaré un sistema de bases de datos, pero ¿por dónde empiezo?



Primero, tenemos que modelar
el problema

Modelo Relacional

Nota: durante estas dos clases se va a mostrar un resumen de lo que se puede hacer con un motor relacional, pero para aprender esto bien se necesitan al menos 8 clases!

Modelo Relacional

El modelo de las bases de datos relacionales se basa en:

- Tablas (relaciones/*relation*)
- Columnas de las tablas (atributos con sus tipos)
- Filas de las tablas (tuplas) que contienen los datos
- Tablas se relacionan entre ellas (relaciones/*relationship*)

Modelo Relacional

Relación

ID Actor	Nombre Actor
1	Leonardo DiCaprio
2	Matthew McConaughey
3	Daniel Radcliffe
4	Jessica Chastain
...	...

Modelo Relacional

Atributos

ID Actor	Nombre Actor
1	Leonardo DiCaprio
2	Matthew McConaughey
3	Daniel Radcliffe
4	Jessica Chastain
...	...

Modelo Relacional

Tuplas

ID Actor	Nombre Actor
1	Leonardo DiCaprio
2	Matthew McConaughey
3	Daniel Radcliffe
4	Jessica Chastain

...

...

Modelación

Don N. necesita una base de datos para manejar los datos de su tienda. Habrá información de:

- Usuarios
- Productos
- Compras

Entidades

Las columnas de la tabla las llamamos **atributos**

Cada **atributo** tiene un dominio determinado (string, int, float, etc)

Entidades

En nuestro contexto, identificamos las entidades Usuarios y Productos

Usuarios(
uid: int,
nombre: string,
edad: int)

Productos(
pid: int,
nombre: string,
precio: float,
stock: int)

Entidades

Usuarios(uid: int, nombre: string, edad: int)

uid	nombre	edad
1	Juan	33
2	Constanza	20
3	Francisca	15

Entidades

Productos(pid: int, nombre: string, precio: float, stock: int)

pid	nombre	precio	stock
1	Computador	1000	3
2	Silla Gamer	500	5
3	Disco Bad Bunny	60	2
4	Plant Based Burger	15	100
5	Figura Iron Man	35	10

Relaciones entre tablas

Necesitamos relacionar qué usuario compró qué producto, almacenando la fecha y la cantidad en la que compró

Compras(
 pid: int,
 uid: int,
 fecha: date,
 cantidad: int)

Obs: Las relaciones entre tablas también son tablas

Relaciones entre tablas

Compras(pid: int, uid: int, fecha: date, cantidad: int)

pid	uid	fecha	Cantidad
1	1	'2020-10-12'	1
2	1	'2020-10-12'	3
3	2	'2020-10-10'	2
4	1	'2020-09-18'	10
4	2	'2020-09-10'	5
4	3	'2020-08-21'	2
1	2	'2020-04-26'	2

Relaciones

Observación

En la programación orientada a objetos estábamos acostumbrados a que, por ejemplo, un usuario tuviese una lista de compras

En el modelo relacional se utilizan tablas intermedias para relacionar las distintas entidades, ya que no se acostumbra a tener atributos de tipo lista

Diagrama E/R

Diagrama utilizado para expresar el modelo de una base de datos relacional

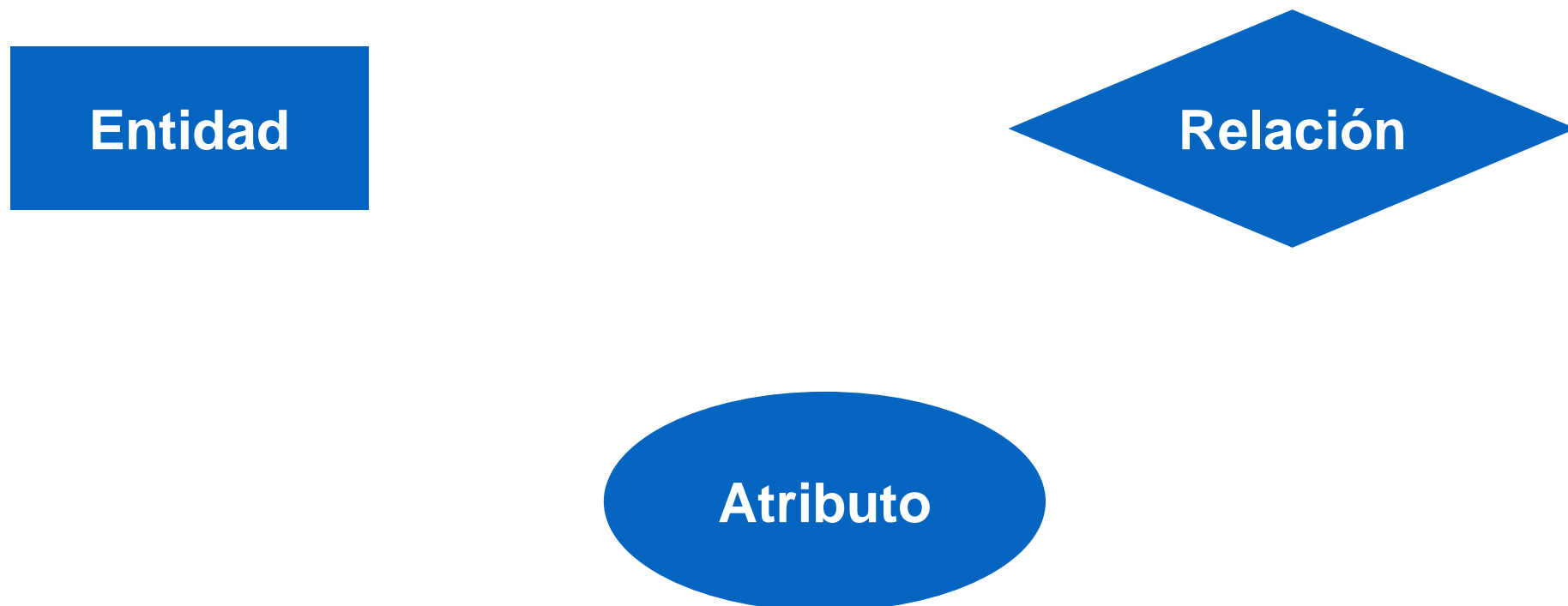


Diagrama E/R

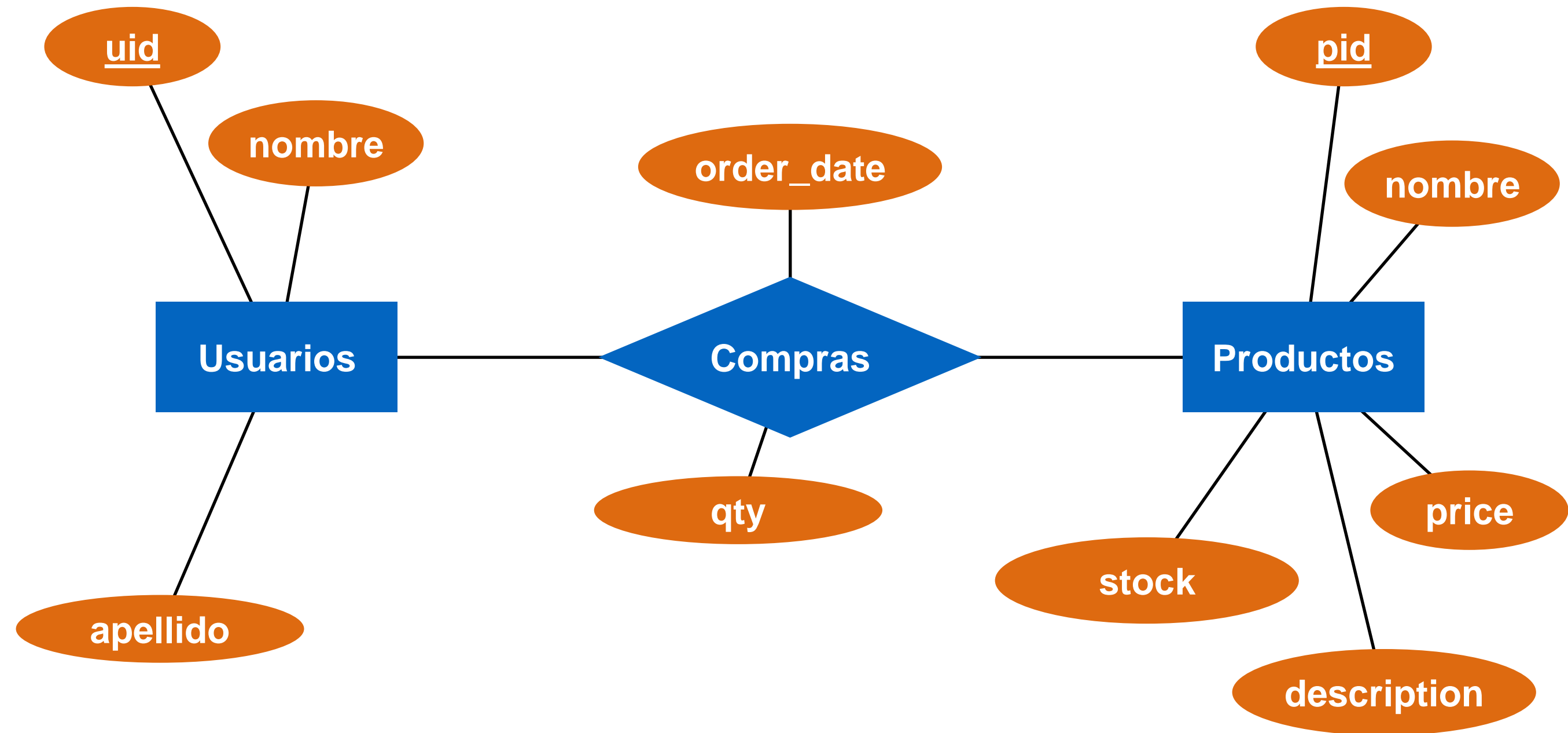


Diagrama E/R

Observaciones

No podemos tener dos usuarios o productos con el mismo identificador

Los campos uid y pid que están presentes en Compras deben existir en las otras tablas

Estas son restricciones de integridad que se pueden incorporar en el modelo

Esquema

En el ejemplo anterior, mostramos la declaración de las tablas

Esquema

En el ejemplo anterior, mostramos la declaración de las tablas

El conjunto de estas declaraciones es el esquema

Usuarios(uid: int, nombre: string, edad: int)

Productos(pid: int, nombre: string, precio: float, stock: int)

Compras(pid: int, uid: int, fecha: date, cantidad: int)



Perfecto! Pero utilizando
Python hubiese podido
extraer cierta información



Podría filtrar por precio, nombre de cliente, saber cuál es el cliente que gastó más NebCoins en un año



Puedo hacer estas cosas ahora?
Quizás a esto le está faltando Data
Science...



Don N., recuerde que los sistemas de bases de datos proveen lenguajes de consulta

SQL

Structured Query Language

SQL es el lenguaje de consulta de las bases de datos relacionales

Un conocimiento sólido de SQL es algo que hay que tener para ser *data scientist*

Revisar, entre otros [<https://www.kdnuggets.com/2014/12/data-science-skills-most-demand.html>]

SQL

Forma básica

Las consultas en general se ven:

SELECT atributos

FROM relaciones

WHERE condiciones / selecciones

Declarativo vs. Procedural

- SQL es declarativo, decimos lo que queremos, pero sin dar detalles de cómo lo computamos
- Un lenguaje como Java es procedural: para hacer algo debemos indicar paso a paso el procedimiento

Álgebra relacional

- Para entender bien SQL, debemos entender el álgebra relacional
- El álgebra relacional nos permite hacer operaciones sobre tablas
- SQL puede expresar todas las consultas que se pueden hacer en álgebra relacional (y más)

Álgebra relacional de selección, proyección y unión

- Lenguaje teórico
- Posee un conjunto de operadores que como input toman tablas, y como output devuelven tablas

$$\pi, \sigma, \cup, \times$$

- En el curso Bases de Datos se aprende álgebra relacional en detalle



Primero debemos crear y llenar
nuestras tablas

SQL

Crear una tabla

Para crear una tabla:

```
CREATE TABLE Usuarios(  
    uid INT,  
    nombre VARCHAR(100),  
    edad INT,  
    PRIMARY KEY(uid)  
);
```

SQL

Crear una tabla

- Se declara el nombre de la tabla, los atributos y sus tipos
- La llave primaria señala el conjunto de atributos que tiene que ser necesariamente distinto para cada fila de la tabla

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

Insertar valor en una tabla

Para insertar valor en una tabla:

```
INSERT INTO Usuarios(uid, nombre, edad)  
VALUES(1, 'Juan', 33)
```

O bien:

```
INSERT INTO Usuarios  
VALUES(1, 'Juan', 33)
```

Usuarios

uid	nombre	edad
1	Juan	33
2	Constanza	20
3	Francisca	15

Productos

pid	nombre	precio	stock
1	Computador	1000	3
2	Silla Gamer	500	5
3	Disco Bad Bunny	60	2
4	Plant Based Burger	15	100
5	Figura Iron Man	35	10

Compras

pid	uid	fecha	Cantidad
1	1	'2020-10-12'	1
2	1	'2020-10-12'	3
3	2	'2020-10-10'	2
4	1	'2020-09-18'	10
4	2	'2020-09-10'	5
4	3	'2020-08-21'	2
1	2	'2020-04-26'	2



Entonces, Don N., ¿qué
consultas desea hacer a la Base
de Datos?



Me gustaría obtener un listado con
todos los productos

SQL

Extraer todo de una tabla

Para extraer todo de la tabla productos:

```
SELECT * FROM Productos
```

SQL

Extraer todo de una tabla

Para extraer todo de la tabla productos:

```
SELECT * FROM Productos
```

pid	nombre	precio	stock
1	Computador	1000	3
2	Silla Gamer	500	5
3	Disco Bad Bunny	60	2
4	Plant Based Burger	15	100
5	Figura Iron Man	35	10



Pensándolo bien, me gustaría obtener
sólo el nombre y precio de los
productos

SQL

Proyección

Si queremos sólo los atributos nombre y precio, debemos declararlo en el **SELECT**:

```
SELECT nombre, precio
```

```
FROM Productos
```

SQL

Proyección

Si queremos sólo los atributos nombre y precio, debemos declararlo en el **SELECT**:

```
SELECT nombre, precio
```

```
FROM Productos
```

pid	nombre	precio	stock
1	Computador	1000	3
2	Silla Gamer	500	5
3	Disco Bad Bunny	60	2
4	Plant Based Burger	15	100
5	Figura Iron Man	35	10

SQL

Proyección

Si queremos sólo los atributos nombre y precio, debemos declararlo en el **SELECT**:

```
SELECT nombre, precio
```

```
FROM Productos
```

nombre	precio
Computador	1000
Silla Gamer	500
Disco Bad Bunny	60
Plant Based Burger	15
Figura Iron Man	35

SQL

Proyección

En álgebra relacional lo escribimos como:

$$\pi_{\text{name,price}}(\text{Products})$$



Muy bien! Y si ahora pregunto por
todos los productos con precio
menor a 1000?

SQL

Selección

Si queremos sólo los productos con precios menor a 1000, lo declaramos en el **WHERE**:

```
SELECT *
```

```
FROM Productos
```

```
WHERE precio < 1000
```

SQL

Selección

Si queremos sólo los productos con precios menor a 1000, lo declaramos en el **WHERE**:

```
SELECT *
```

```
FROM Productos
```

```
WHERE precio < 1000
```

pid	nombre	precio	stock
1	Computador	1000	3
2	Silla Gamer	500	5
3	Disco Bad Bunny	60	2
4	Plant Based Burger	15	100
5	Figura Iron Man	35	10

SQL

Selección

Si queremos sólo los productos con precios menor a 1000, lo declaramos en el **WHERE**:

```
SELECT *
```

```
FROM Productos
```

```
WHERE precio < 1000
```

pid	nombre	precio	stock
1	Computador	1000	3
2	Silla Gamer	500	5
3	Disco Bad Bunny	60	2
4	Plant Based Burger	15	100
5	Figura Iron Man	35	10

SQL

Selección

Si queremos sólo los productos con precios menor a 1000, lo declaramos en el **WHERE**:

```
SELECT *
```

```
FROM Productos
```

```
WHERE precio < 1000
```

pid	nombre	precio	stock
2	Silla Gamer	500	5
3	Disco Bad Bunny	60	2
4	Plant Based Burger	15	100
5	Figura Iron Man	35	10

SQL

Selección

Si queremos proyectar sólo el nombre y precio de los productos con precios menor a 1000:

```
SELECT nombre, precio
```

```
FROM Productos
```

```
WHERE precio < 1000
```

pid	nombre	precio	stock
2	Silla Gamer	500	5
3	Disco Bad Bunny	60	2
4	Plant Based Burger	15	100
5	Figura Iron Man	35	10

SQL

Selección

Si queremos proyectar sólo el nombre y precio de los productos con precios menor a 1000:

```
SELECT nombre, precio  
FROM Productos  
WHERE precio < 1000
```

nombre	precio
Silla Gamer	500
Disco Bad Bunny	60
Plant Based Burger	15
Figura Iron Man	35

SQL

Selección

En álgebra relacional lo escribimos como:

$$\sigma_{\text{price} < 1000}(\text{Products})$$

O la versión con proyección

$$\pi_{\text{name, price}}(\sigma_{\text{price} < 1000}(\text{Products}))$$

SQL

Selección

Las condiciones pueden ser:

$<, \leq, \geq, >, =, \neq$

Y se pueden combinar con:

\wedge, \vee

SQL

Selección

Las condiciones pueden ser:

<, <=, >=, >, <>

Y se pueden combinar con:

AND, OR



Ahora me gustaría cruzar datos de
tablas distintas



Por ejemplo, saber el nombre de los productos que cada usuario ha comprado

Producto Cruz

Nos falta cruzar información entre tablas

El operador \times permite hacer el producto cartesiano de dos relaciones

$$\begin{array}{c|cc} R_1 & A & B \\ \hline & a_1 & b_1 \\ & a_2 & b_2 \end{array} \times \begin{array}{c|ccc} R_2 & A & C & D \\ \hline & a_1 & c_1 & d_1 \\ & a_2 & c_2 & d_2 \end{array} = \begin{array}{c|ccccc} & R_1.A & R_1.B & R_2.A & R_2.C & R_2.D \\ \hline & a_1 & b_1 & a_1 & c_1 & d_1 \\ & a_1 & b_1 & a_2 & c_2 & d_2 \\ & a_2 & b_2 & a_1 & c_1 & d_1 \\ & a_2 & b_2 & a_2 & c_2 & d_2 \end{array}$$

SQL

Producto Cruz

Si queremos hacer el producto cruz entre dos tablas, las declaramos en el **FROM**:

```
SELECT *
```

```
FROM Productos, Compras
```

pid	nombre	precio	stock
1	Computador	1000	3
2	Silla Gamer	500	5
3	Disco Bad Bunny	60	2
4	Plant Based Burger	15	100
5	Figura Iron Man	35	10

pid	uid	fecha	Cantidad
1	1	'2020-10-12'	1
2	1	'2020-10-12'	3
3	2	'2020-10-10'	2
4	1	'2020-09-18'	10
4	2	'2020-09-10'	5
4	3	'2020-08-21'	2
1	2	'2020-04-26'	2

SQL

Producto Cruz

En álgebra relacional lo escribimos como:

$\text{Products} \times \text{Orders}$

SQL

Join

Para la consulta debemos usar producto cruz y selección

Vamos a utilizar el “operador” Join \bowtie . En realidad no es un operador, pues es definible con selección y producto cruz:

$$R_1 \bowtie_{\text{condición}} R_2 = \sigma_{\text{condición}}(R_1 \times R_2)$$

SQL

Join

Parte 1: para cada usuario, obtener el id de los productos que ha comprado

$$\sigma_{\text{Users.uid} = \text{Orders.uid}}(\text{Users} \times \text{Orders})$$

Esto es equivalente a:

$$\text{Users} \bowtie_{\text{Users.uid} = \text{Orders.uid}} \text{Orders}$$

uid	nombre	edad
1	Juan	33
2	Constanza	20
3	Francisca	15

pid	uid	fecha	Cantidad
1	1	'2020-10-12'	1
2	1	'2020-10-12'	3
3	2	'2020-10-10'	2
4	1	'2020-09-18'	10
4	2	'2020-09-10'	5
4	3	'2020-08-21'	2
1	2	'2020-04-26'	2

Users ⋈_{Users.uid = Orders.uid} Orders

uid	nombre	edad	pid	uid	fecha	Cantidad
1	Juan	33	1	1	'2020-10-12'	1
1	Juan	33	2	1	'2020-10-12'	3
2	Constanza	20	3	2	'2020-10-10'	2
1	Juan	33	4	1	'2020-09-18'	10
2	Constanza	20	4	2	'2020-09-10'	5
3	Francisca	15	4	3	'2020-08-21'	2
2	Constanza	20	1	2	'2020-04-26'	2

SQL

Natural Join

Cuando los atributos en ambas relaciones tienen el mismo nombre, es posible no indicar la condición:

La consulta anterior se puede escribir como:

Users \bowtie Orders

SQL

Join

Y en SQL se escribe:

```
SELECT *
```

```
FROM Usuarios, Compras
```

```
WHERE Usuarios.uid = Compras.uid
```

SQL

Join

Parte 2: para cada usuario, obtener el nombre de los productos que ha comprado, y retornar sólo el nombre del producto y del usuario

$$\pi_{\text{Users.first_name, Products.name}}(\text{Users} \bowtie \text{Orders} \bowtie \text{Products})$$

SQL

Join

La consulta anterior en SQL se escribe:

```
SELECT Usuarios.nombre, Productos.nombre  
FROM Usuarios, Compras, Productos  
WHERE Usuarios.uid = Compras.uid  
      AND Compras.pid = Productos.pid
```

SQL

Join

SELECT *

FROM Usuarios, Compras, Productos

WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid

uid	nombre	edad	pid	fecha	Cantidad	nombre_1	precio	stock
1	Juan	33	1	'2020-10-12'	1	Computador	1000	3
1	Juan	33	2	'2020-10-12'	3	Silla Gamer	500	5
2	Constanza	20	3	'2020-10-10'	2	Disco Bad Bunny	60	2
1	Juan	33	4	'2020-09-18'	10	Plant Based Burger	15	100
2	Constanza	20	4	'2020-09-10'	5	Plant Based Burger	15	100
3	Francisca	15	4	'2020-08-21'	2	Plant Based Burger	15	100
2	Constanza	20	1	'2020-04-26'	2	Computador	1000	3

SQL

Join

```
SELECT Usuarios.nombre, Productos.nombre  
FROM Usuarios, Compras, Productos  
WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid
```

nombre	nombre_1
Juan	Computador
Juan	Silla Gamer
Constanza	Disco Bad Bunny
Juan	Plant Based Burger
Constanza	Plant Based Burger
Francisca	Plant Based Burger
Constanza	Computador

SQL

Otras funciones

En SQL también podemos hacer agregación, ordenar, hacer consultas anidadas, entre otras cosas

Veamos ahora algunos ejemplos

SQL

Ordenar

Ordenar las compras por fecha:

```
SELECT *  
FROM Compras  
ORDER BY fecha
```

O en orden inverso:

```
SELECT *  
FROM Compras  
ORDER BY DESC fecha
```

SQL

Ordenar

```
SELECT *  
FROM Compras  
ORDER BY fecha
```

pid	uid	fecha	Cantidad
1	2	'2020-04-26'	2
4	3	'2020-08-21'	2
4	2	'2020-09-10'	5
4	1	'2020-09-18'	10
3	2	'2020-10-10'	2
1	1	'2020-10-12'	1
2	1	'2020-10-12'	3

SQL

Agregación

Para cada día, obtener el número de productos comprados

```
SELECT fecha, SUM(cantidad)
FROM Compras
GROUP BY fecha
```

SQL

Agregación

```
SELECT fecha, SUM(cantidad)
```

```
FROM Compras
```

```
GROUP BY fecha
```

pid	uid	fecha	Cantidad
1	1	'2020-10-12'	1
2	1	'2020-10-12'	3
3	2	'2020-10-10'	2
4	1	'2020-09-18'	10
4	2	'2020-09-10'	5
4	3	'2020-08-21'	2
1	2	'2020-04-26'	2

SQL

Agregación

```
SELECT fecha, SUM(cantidad)
```

```
FROM Compras
```

```
GROUP BY fecha
```

fecha	SUM(cantidad)
'2020-10-12'	4
'2020-10-10'	2
'2020-09-18'	10
'2020-09-10'	5
'2020-08-21'	2
'2020-04-26'	2

SQL

Agregación más compleja

Para cada usuario, obtener el dinero que ha gastado

SELECT

FROM

WHERE

GROUP BY

SQL

Agregación más compleja

SELECT

FROM Usuarios, Productos, Compras

WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid

GROUP BY

uid	nombre	edad	pid	fecha	Cantidad	nombre_1	precio	stock
1	Juan	33	1	'2020-10-12'	1	Computador	1000	3
1	Juan	33	2	'2020-10-12'	3	Silla Gamer	500	5
2	Constanza	20	3	'2020-10-10'	2	Disco Bad Bunny	60	2
1	Juan	33	4	'2020-09-18'	10	Plant Based Burger	15	100
2	Constanza	20	4	'2020-09-10'	5	Plant Based Burger	15	100
3	Francisca	15	4	'2020-08-21'	2	Plant Based Burger	15	100
2	Constanza	20	1	'2020-04-26'	2	Computador	1000	3

SQL

Agregación más compleja

SELECT

FROM Usuarios, Productos, Compras

WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid

GROUP BY Usuarios.uid, Usuarios.nombre;

uid	nombre	edad	pid	fecha	Cantidad	nombre_1	precio	stock
1	Juan	33	1	'2020-10-12'	1	Computador	1000	3
1	Juan	33	2	'2020-10-12'	3	Silla Gamer	500	5
1	Juan	33	4	'2020-09-18'	10	Plant Based Burger	15	100
2	Constanza	20	3	'2020-10-10'	2	Disco Bad Bunny	60	2
2	Constanza	20	4	'2020-09-10'	5	Plant Based Burger	15	100
2	Constanza	20	1	'2020-04-26'	2	Computador	1000	3
3	Francisca	15	4	'2020-08-21'	2	Plant Based Burger	15	100

SQL

Agregación más compleja

SELECT

FROM Usuarios, Productos, Compras

WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid

GROUP BY Usuarios.uid, Usuarios.nombre;

Dinero Gastado

uid	nombre	edad	pid	fecha	Cantidad	nombre_1	precio	stock
1	Juan	33	1	'2020-10-12'	1	Computador	1000	3
1	Juan	33	2	'2020-10-12'	3	Silla Gamer	500	5
1	Juan	33	4	'2020-09-18'	10	Plant Based Burger	15	100
2	Constanza	20	3	'2020-10-10'	2	Disco Bad Bunny	60	2
2	Constanza	20	4	'2020-09-10'	5	Plant Based Burger	15	100
2	Constanza	20	1	'2020-04-26'	2	Computador	1000	3
3	Francisca	15	4	'2020-08-21'	2	Plant Based Burger	15	100

SQL

Agregación más compleja

```
SELECT Usuarios.uid, Usuarios.nombre, SUM(Compras.cantidad*Productos.precio)
```

```
FROM Usuarios, Productos, Compras
```

```
WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid
```

```
GROUP BY Usuarios.uid, Usuarios.nombre;
```

uid	nombre	edad	pid	fecha	Cantidad	nombre_1	precio	stock
1	Juan	33	1	'2020-10-12'	1	Computador	1000	3
1	Juan	33	2	'2020-10-12'	3	Silla Gamer	500	5
1	Juan	33	4	'2020-09-18'	10	Plant Based Burger	15	100
2	Constanza	20	3	'2020-10-10'	2	Disco Bad Bunny	60	2
2	Constanza	20	4	'2020-09-10'	5	Plant Based Burger	15	100
2	Constanza	20	1	'2020-04-26'	2	Computador	1000	3
3	Francisca	15	4	'2020-08-21'	2	Plant Based Burger	15	100

SQL

Agregación más compleja

```
SELECT Usuarios.uid, Usuarios.nombre, SUM(Compras.cantidad*Productos.precio)
FROM Usuarios, Productos, Compras
WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid
GROUP BY Usuarios.uid, Usuarios.nombre;
```

uid	nombre	SUM(Compras.cantidad*Productos.precio)
1	Juan	2650
2	Constanza	2195
3	Francisca	30

SQL

Agregación más compleja

Para cada usuario, filtrar los que han gastado más de 1000 NebCoins

SQL

Agregación más compleja

Para cada usuario, filtrar los que han gastado más de 1000 NebCoins

SELECT

FROM

WHERE

GROUP BY

HAVING

SQL

Agregación más compleja

SELECT

FROM Usuarios, Productos, Compras

WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid

GROUP BY

HAVING

uid	nombre	edad	pid	fecha	Cantidad	nombre_1	precio	stock
1	Juan	33	1	'2020-10-12'	1	Computador	1000	3
1	Juan	33	2	'2020-10-12'	3	Silla Gamer	500	5
2	Constanza	20	3	'2020-10-10'	2	Disco Bad Bunny	60	2
1	Juan	33	4	'2020-09-18'	10	Plant Based Burger	15	100
2	Constanza	20	4	'2020-09-10'	5	Plant Based Burger	15	100
3	Francisca	15	4	'2020-08-21'	2	Plant Based Burger	15	100
2	Constanza	20	1	'2020-04-26'	2	Computador	1000	3

SQL

Agregación más compleja

SELECT

FROM Usuarios, Productos, Compras

WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid

GROUP BY Usuarios.uid, Usuarios.nombre

HAVING

uid	nombre	edad	pid	fecha	Cantidad	nombre_1	precio	stock
1	Juan	33	1	'2020-10-12'	1	Computador	1000	3
1	Juan	33	2	'2020-10-12'	3	Silla Gamer	500	5
1	Juan	33	4	'2020-09-18'	10	Plant Based Burger	15	100
2	Constanza	20	3	'2020-10-10'	2	Disco Bad Bunny	60	2
2	Constanza	20	4	'2020-09-10'	5	Plant Based Burger	15	100
2	Constanza	20	1	'2020-04-26'	2	Computador	1000	3
3	Francisca	15	4	'2020-08-21'	2	Plant Based Burger	15	100

SQL

Agregación más compleja

SELECT

FROM Usuarios, Productos, Compras

WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid

GROUP BY Usuarios.uid, Usuarios.nombre

HAVING

¿Cuánto han gastado?

uid	nombre	edad	pid	fecha	Cantidad	nombre_1	precio	stock
1	Juan	33	1	'2020-10-12'	1	Computador	1000	3
1	Juan	33	2	'2020-10-12'	3	Silla Gamer	500	5
1	Juan	33	4	'2020-09-18'	10	Plant Based Burger	15	100
2	Constanza	20	3	'2020-10-10'	2	Disco Bad Bunny	60	2
2	Constanza	20	4	'2020-09-10'	5	Plant Based Burger	15	100
2	Constanza	20	1	'2020-04-26'	2	Computador	1000	3
3	Francisca	15	4	'2020-08-21'	2	Plant Based Burger	15	100

2650

2195

30

SQL

Agregación más compleja

SELECT

FROM Usuarios, Productos, Compras

WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid

GROUP BY Usuarios.uid, Usuarios.nombre

HAVING SUM(Compras.cantidad* Productos.precio) > 1000

uid	nombre	edad	pid	fecha	Cantidad	nombre_1	precio	stock
1	Juan	33	1	'2020-10-12'	1	Computador	1000	3
1	Juan	33	2	'2020-10-12'	3	Silla Gamer	500	5
1	Juan	33	4	'2020-09-18'	10	Plant Based Burger	15	100
2	Constanza	20	3	'2020-10-10'	2	Disco Bad Bunny	60	2
2	Constanza	20	4	'2020-09-10'	5	Plant Based Burger	15	100
2	Constanza	20	1	'2020-04-26'	2	Computador	1000	3
3	Francisca	15	4	'2020-08-21'	2	Plant Based Burger	15	100

2650

2195

30

SQL

Agregación más compleja

```
SELECT SELECT Usuarios.uid, Usuarios.nombre, SUM(Compras.cantidad*Productos.precio)
FROM Usuarios, Productos, Compras
WHERE Usuarios.uid = Compras.uid AND Compras.pid = Productos.pid
GROUP BY Usuarios.uid, Usuarios.nombre
HAVING SUM(Compras.cantidad* Productos.precio) > 1000
```

uid	nombre	SUM(Compras.cantidad*Productos.precio)
1	Juan	2650
2	Constanza	2195

Consultas con Agregación

SELECT <S>

FROM R_1, \dots, R_n

WHERE <Condición 1>

GROUP BY a_1, \dots, a_k

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

SQL

Otras funciones

Entre otras cosas, podemos:

- Contar (**COUNT**)
- Sumar (**SUM**)
- Sacar promedio (**AVG**)
- Obtener mínimo (**MIN**)
- Obtener máximo (**MAX**)



Esto es muy interesante! Estoy ansioso por empezar a usar el sistema de bases de datos



Si hubiese hecho mi programa en Python, desarrollaría una aplicación web que usara los datos



Don N., por supuesto que una aplicación web puede conectarse a la base de datos relacional



Don N., me contaron que al usar un
framework web ya no hay que saber
SQL



Entonces lo que aprendí no me va a servir de nada?

ORM

Los frameworks web tienen librerías para abstraerse de la base de datos

Un ORM (Object-Relational Mapping) es una técnica para tratar a los datos de un sistema como objetos de un lenguaje de programación

ORM

Ejemplo - Modelos

```
from django.db import models

class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)

class Album(models.Model):
    artist = models.ForeignKey(Musician, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    release_date = models.DateField()
    num_stars = models.IntegerField()
```

ORM

Ejemplo - Consultas

Obtener todos los músicos:

```
>>> Musician.objects.all()
```

Obtener todos los músicos con nombre 'James':

```
>>> Musician.objects.filter(first_name='James')
```

Obtener todos los álbumes del artista con id 1:

```
>>> Musician.objects.get(id=1).album_set.all()
```

ORM

Un ORM permite abstraerse de un sistema de bases de datos en particular

No es tan flexible como utilizar SQL, y no depende del desarrollador cómo se traducen las consultas

Nosotros instalamos la base de datos, pero el ORM se encarga de utilizarla



Es posible hacer una aplicación web
sin saber SQL y sólo saber usar el
ORM?



Sí, pero también es posible hacer una aplicación web en un editor en línea sin saber cómo funcionan

[Funciones](#)[Explora](#)[Suscripciones](#) [Plantillas](#)[Ayuda](#)[Entrar](#)

Todo empieza con tu increíble página web

Wix combina la belleza con la más avanzada tecnología para crear tu increíble página web. Es fácil y gratis.

[Empieza ya](#)



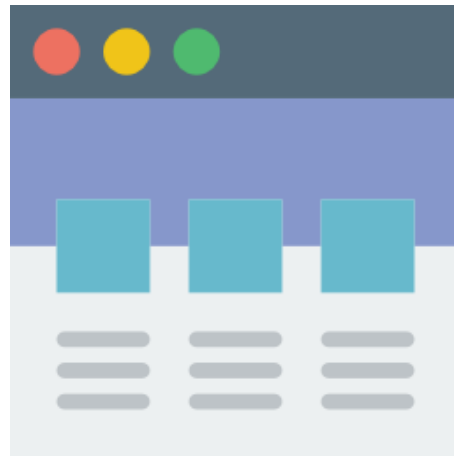
O también es posible armar un puente sin saber cómo hacerlo





“Es importante conocer cómo funciona SQL para construir aplicaciones de calidad”

Profesor A, abril 2021



Y así, Don N. finalizó la
aplicación web de su NebStore,
pero...



La aplicación está funcionando demasiado lenta, quizás le está faltando...

Big Data

Cloud Computing

Blockchain

Buzzword

Quantum Computing

Microservices

Deep Learning

Internet of Things

Data Science





Profesor, me han contado que
hay que usar NoSQL, porque
funcionan mucho más rápido

¿Es SQL suficiente?

Un sistema puede funcionar lento porque:

- La base de datos está mal indexada
- Los datos están mal modelados
- Hay una cantidad de datos demasiado grande
- ...

Tipos de DBMS

- Relacional
- Key - Value
- Column Store
- Graph Databases
- Document Store
- RDF
- In Memory Databases
- ...

Tipos de DBMS

- Relacional



- SQLite
- PostgreSQL

Tipos de DBMS

- Documentos  - MongoDB

```
1  {  
2    _id: "5cf0029caff5056591b0ce7d",  
3    firstname: 'Jane',  
4    lastname: 'Wu',  
5    address: {  
6      street: '1 Circle Rd',  
7      city: 'Los Angeles',  
8      state: 'CA',  
9      zip: '90404'  
10   }  
11 }
```



Tipos de DBMS

- Documentos  - MongoDB

```
db.usuarios.find()
```

```
db.usuarios.find({uid: 2}, {}).pretty()
```

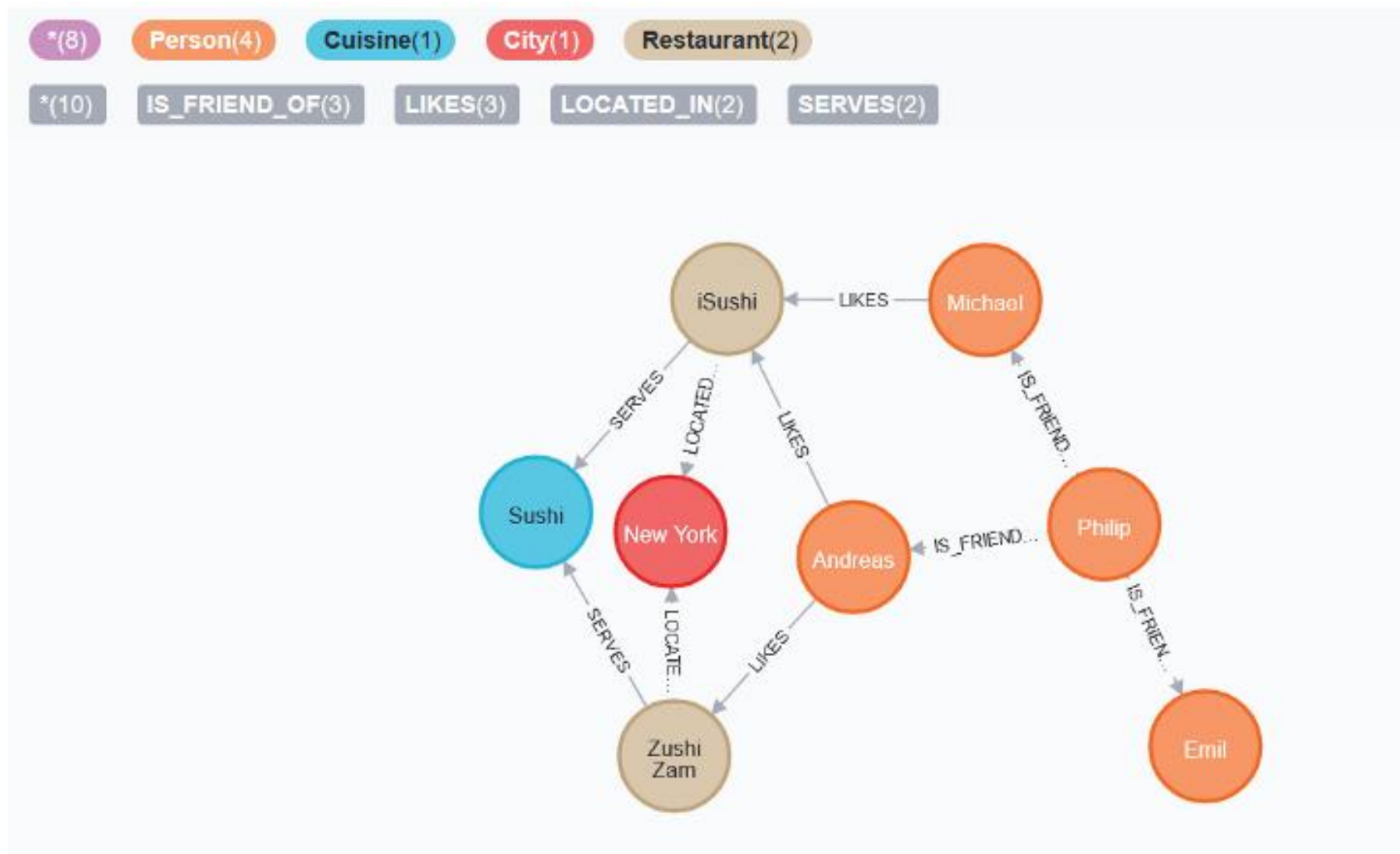
```
db.productos.find({price: {$gt: 300}}, {}).pretty()
```

Tipos de DBMS

- Grafos



- Neo4j



Tipos de DBMS

- Grafos



- Neo4j

```
MATCH (p:Person {name: 'Jennifer'})-[:LIKES]-(r:Restaurant)
RETURN r
```


Tipos de DBMS

- Hay que usar cada sistema para lo que fue pensado
- No hay dejarse engañar por publicidad

Tipos de DBMS



Why OrientDB? ▾

Products ▾

Learn ▾

Services ▾

Company ▾

Download

OrientDB - The World's First Distributed Multi-Model NoSQL Database with a Graph Database Engine

Tipos de DBMS



[Home](#) [About](#) [Documentation](#) [Download](#) [License](#) [Support](#) [Purchase](#)

SQLite is a [self-contained](#), [high-reliability](#), [embedded](#), [full-featured](#), [public-domain](#), SQL database engine. SQLite is the [most used](#) database engine in the world. [More Info](#)

Resumen

- Todos necesitamos manejar datos
- Salvo que queramos programar algoritmos más allá de nuestro problema, nos conviene usar un DBMS
- Hay que saber identificar el caso de uso de cada DBMS

Links de interés

- Video de 20 min de SQL Básico:

https://www.youtube.com/watch?v=K8bgmWoty0E&t=644s&ab_channel=Adri%C3%A1nSotoSu%C3%A1rez

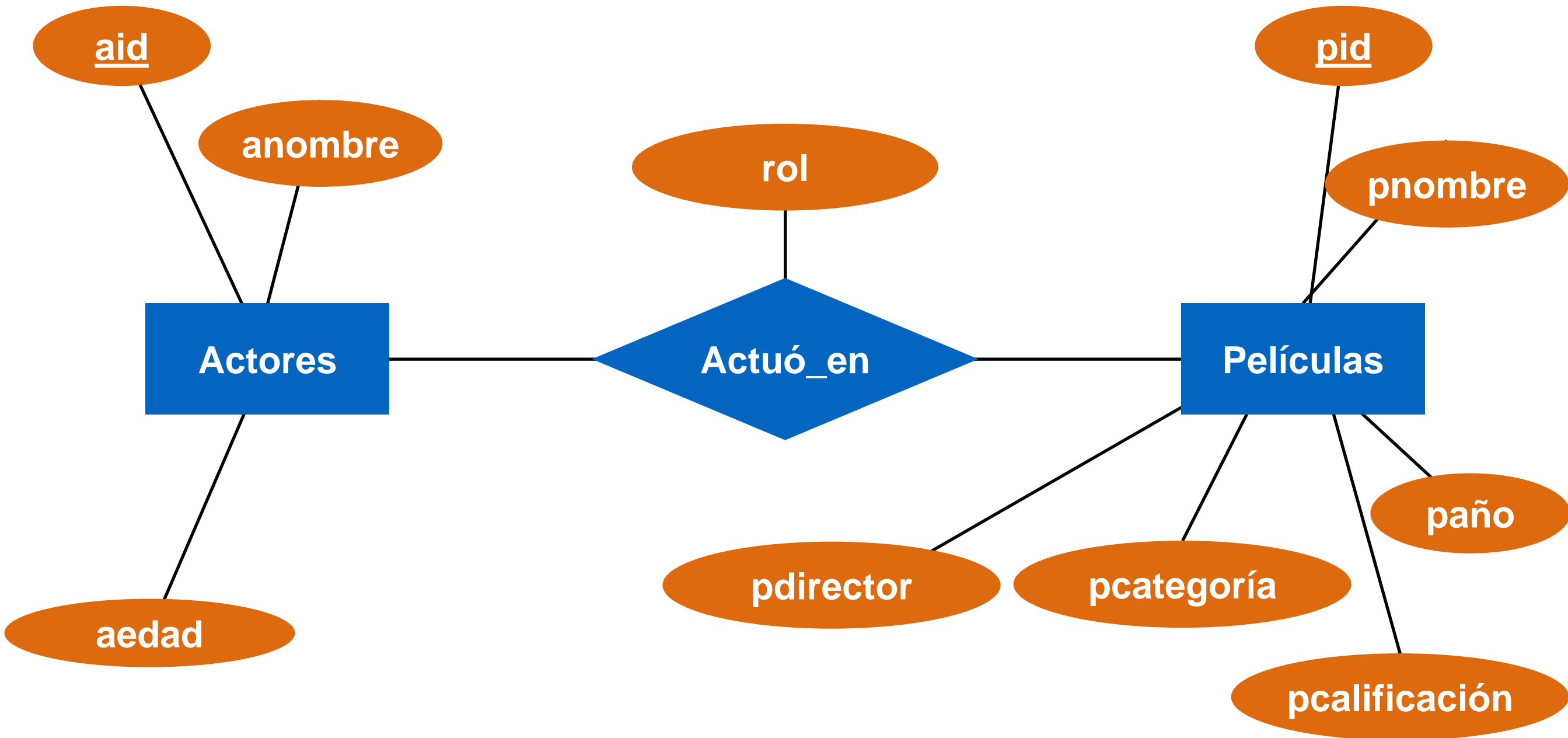
- Video de 15 min de Agregación en SQL:

https://www.youtube.com/watch?v=n-Ky1go7Alk&t=7s&ab_channel=Adri%C3%A1nSotoSu%C3%A1rez



Te parece ver un ejemplo en vivo
para ver que cosas se pueden
hacer?

Diagrama E/R Ejemplo en vivo





Ejemplo en vivo

Introducción a las bases de datos

Exploratorio en Computación
Invitado: Valentina Álvarez