

# Introducción a las bases de datos

Exploratorio en Computación  
Invitado: Adrián Soto

¿Por qué bases de  
datos?

# Un día cualquiera (2 de abril)

10:30 am Despierto, reviso Telegram

11:30 am Supermercado, pago con tarjeta

12:00 pm Transporte público, utilizo tarjeta Bip!

12:30 pm Reviso EMOL

13:00 pm Reviso cuenta de BTC

13:30 pm Actualizo archivos de Drive

14:15 pm Aburrido, reviso Twitter

16:00 pm Subo una historia a Instagram

# Un día cualquiera (2 de abril)

- |          |   |
|----------|---|
| 10:30 am | Despierto, reviso <b>Telegram</b>       |
| 11:30 am | Supermercado, pago con <b>tarjeta</b>   |
| 12:00 pm | Transporte público, utilizo <b>Bip!</b> |
| 12:30 pm | Reviso <b>EMOL</b>                      |
| 13:00 pm | Reviso mi billetera de <b>BTC</b>       |
| 13:30 pm | Actualizo archivos de <b>Drive</b>      |
| 14:15 pm | Aburrido, reviso <b>Twitter</b>         |
| 16:00 pm | Subo una historia a <b>Instagræm</b>    |

...

...

Un día cualquiera (4 de abril)

# Un día cualquiera (4 de abril)

Todas las actividades involucraban una base de datos:

- Búsquedas en la web
- Redes sociales
- Métodos de pago

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

# Outline

- Por qué usar un sistemas de bases de datos
- Introducción a SQL
- NoSQL y sistemas de bases de Datos

# Sistemas de Bases de Datos

Sistema de gestión de bases de datos (Database Management System - **DBMS**)

- Programa que facilite el manejo de grandes volúmenes de datos

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

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

ID Película	Nombre Película
1	Interstellar
2	The Revenant
3	Harry Potter
4	The Wolf of Wall Street
...	...

ID Actor	Nombre Actor	ID Película	Nombre Película
1	Leonardo DiCaprio	1	Interstellar
2	Matthew McConaughey	2	The Revenant
3	Daniel Radcliffe	3	Harry Potter
4	Jessica Chastain	4	The Wolf of Wall Street
...	...	...	...

ID Actor	ID Película
1	2
1	4
2	1
3	3
...	...

# Por qué usar DBMS

# Por qué usar DBMS

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

# Cómo funciona un DBMS

# Cómo funciona un DBMS



# Cómo funciona un DBMS



¿Cuál es la mejor  
película de  
Christopher Nolan?

# Cómo funciona un DBMS



¿Cuál es la mejor  
película de  
Christopher Nolan?



# Cómo funciona un DBMS



¿Cuál es la mejor  
película de  
Christopher Nolan?



`SELECT movies,  
rating FROM  
Movies`

# Cómo funciona un DBMS



¿Cuál es la mejor  
película de  
Christopher Nolan?



# Cómo funciona un DBMS



¿Cuál es la mejor  
película de  
Christopher Nolan?



# Cómo funciona un DBMS



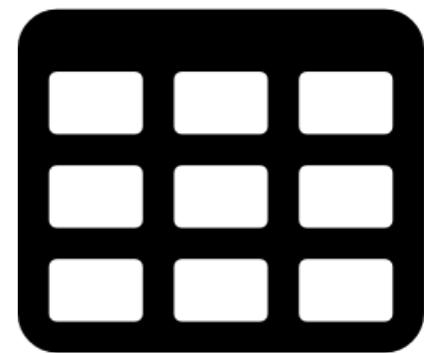
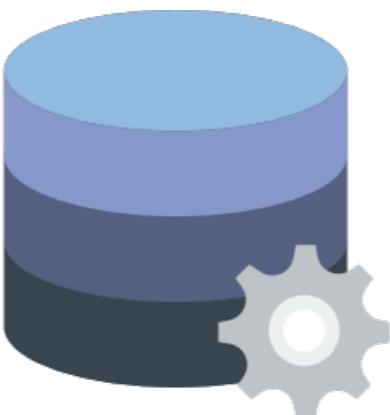
¿Cuál es la mejor  
película de  
Christopher Nolan?



# Cómo funciona un DBMS



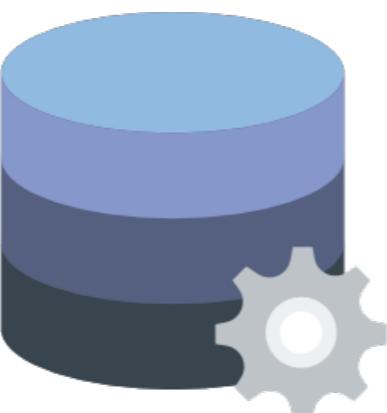
¿Cuál es la mejor  
película de  
Christopher Nolan?



# Cómo funciona un DBMS



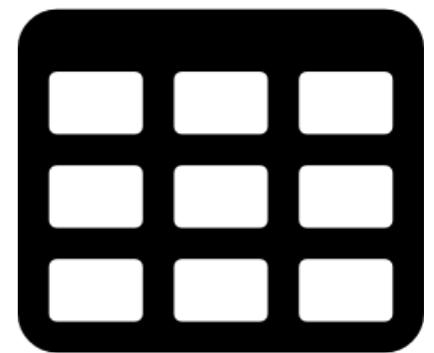
¿Cuál es la mejor  
película de  
Christopher Nolan?



# Cómo funciona un DBMS



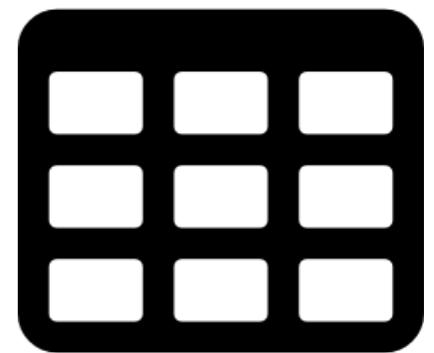
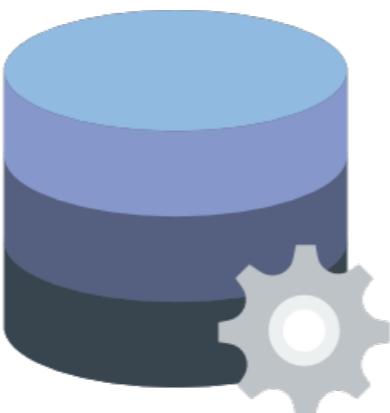
¿Cuál es la mejor  
película de  
Christopher Nolan?



# Cómo funciona un DBMS



¿Cuál es la mejor  
película de  
Christopher Nolan?



# Cómo funciona un DBMS



¿Cuál es la mejor  
película de  
Christopher Nolan?



# Cómo funciona un DBMS



¿Cuál es la mejor  
película de  
Christopher Nolan?



Interstellar







## 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 Oak, ¿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 DBMS...

# Por qué usar DBMS

# Por qué usar DBMS

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

# Implementación a la medida

Archivo libros.txt

L01 # Harry Quebert # Joël Dicker # \$12.500

L02 # Cien Años de Soledad # GGM # \$8.000

L03 # Origen # Dan Brown # \$15.000

L04 # La Divina Comedia # Dante # \$20.000

...

# Implementación a la medida

Archivo usuarios.txt

# Implementación a la medida

Archivo usuarios.txt

U01 # Jaime # XXXX 1234

U02 # Don N. # XXXX 2345

U03 # Profesor Oak # XXXX 3456

U04 # Tía Pame # XXXX 4567

...

# Implementación a la medida

Archivo compras.txt

# Implementación a la medida

Archivo compras.txt

L01 # U02

L01 # U04

L02 # U02

L04 # U01

...



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

# Implementación a la medida

Comprar un libro

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

Instrucciones para nuestro programa:

# 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 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:** ...

# Implementación a la medida

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

**Tercera tarea:** ...

Ahora además, la base de datos es utilizada por todos los usuarios de manera concurrente



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 consta de 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!

# Problemas

Cruce de datos entre dos tablas es aún más costoso:

- libros.txt y usuarios.txt pesan 16 mb
- Cada página de disco tiene 8 kb
- El archivo de compras tiene 1000 registros

Si usamos fuerza bruta, por cada par tardaremos 0.2 segundos, por lo que si queremos saber que usuario compró que libro podríamos tardar hasta 200 segundos!



Y luego aparecieron  
problemas que no tenía  
considerados!

# Problemas

¿Qué pasa con el manejo de buffer?

- Imaginemos que estudiantes.txt pesa 10 gb
- Tenemos un disco duro de 1 tb
- ¿Qué pasa si solamente tengo 8 gb de RAM?

# Problemas

Imaginemos que dos personas toman al mismo tiempo un curso para el que queda un cupo

- ¿Qué pasa con el control de concurrencia?



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



Exacto!

# Usando un DBMS

Un DBMS es un programa (por ejemplo PSQL, MySQL, MongoDB) que sirve para administrar datos

Los más clásicos son los motores SQL, que sirven para trabajar sobre un **modelo relacional**

# Usando un DBMS

Si usamos un DBMS:

- Tenemos un motor de consultas (encuentra la información, actualiza)
- Podemos optimizar consultas
- Podemos manejar transacciones (acceso concurrente)
- Almacenaje óptimo
- ...

# Usando un DBMS

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

# Ventajas de un DBMS

- La capa física está separada de la lógica:  
Si se cambia el almacenamiento en disco, la lógica de las consultas no cambia

# Ventajas de un DBMS

- Acceso eficiente a los datos
- Usamos un lenguaje declarativo:  
Declaramos la consulta sin decir como ejecutarla  
paso a paso
- Integridad de los datos (restricciones de integridad)

# Ventajas de un DBMS

- Acceso concurrente
- Recuperación ante caídas del sistema
- Velocidad y soporte para grandes cantidades de datos

# Desventajas de un DBMS

- Arquitectura cliente servidor puede no ser siempre la mejor
- 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



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



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)
- Columnas de las tablas (atributos con sus tipos)
- Filas de las tablas (tuplas) que contienen los datos

# Modelación

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

- Usuarios
- Productos
- Compras

# Diagrama E/R

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

# Diagrama E/R

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

Entidad

# Diagrama E/R

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

Entidad

Relación

# Entidades

En nuestro contexto, identificamos las entidades  
**Usuarios** y **Productos**

# Entidades

En nuestro contexto, identificamos las entidades  
**Usuarios** y **Productos**

```
Users(  
  uid: int, first_name: string,  
  last_name: string)
```

# Entidades

En nuestro contexto, identificamos las entidades  
**Usuarios** y **Productos**

```
Users(  
    uid: int, first_name: string,  
    last_name: string)
```

```
Products(  
    pid: int, name: string,  
    description: string, price: float,  
    stock: int)
```

# Entidades

```
Users(  
    uid: int, first_name: string,  
    last_name: string)
```

# Entidades

```
Users(  
    uid: int, first_name: string,  
    last_name: string)
```

Uid	First_name	Last_name
1	Ash	Ketchum
2	Arturo	Vidal
...	...	...

# Entidades

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

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

# Entidades

```
Products(  
    pid: int, name: string,  
    description: string, price: float,  
    stock: int)
```

# Entidades

Products(

```
pid: int, name: string,  
description: string, price: float,  
stock: int)
```

Pid	Name	Description	Price	Stock
1	Best of Bad Bunny	Disco del artista del momento	100	1
2	Figura Chikorita	Figura de Pokémon	50	4
3	Tablero de Ajedrez	Tablero de deporte clásico	240	10
4	Figura de Totodile	Figura de Pokémon	60	2
...	...	...	...	...

# Relaciones entre tablas

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

# Relaciones entre tablas

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

```
Orders(  
    pid: int, uid: int,  
    order_date: date, qty: int)
```

# Relaciones entre tablas

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

```
Orders(  
    pid: int, uid: int,  
    order_date: date, qty: int)
```

**Obs:** Las relaciones entre tablas también son tablas

# Relaciones

## Observación

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

# 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

# Entidades

```
Orders(  
    pid: int, uid: int,  
    order_date: date, qty: int)
```

# Entidades

```
Orders(  
    pid: int, uid: int,  
    order_date: date, qty: int)
```

Pid	Uid	Order_date	Qty
2	1	23/02/2018	2
1	2	27/03/2018	1
...	...	...	...

# Diagrama E/R

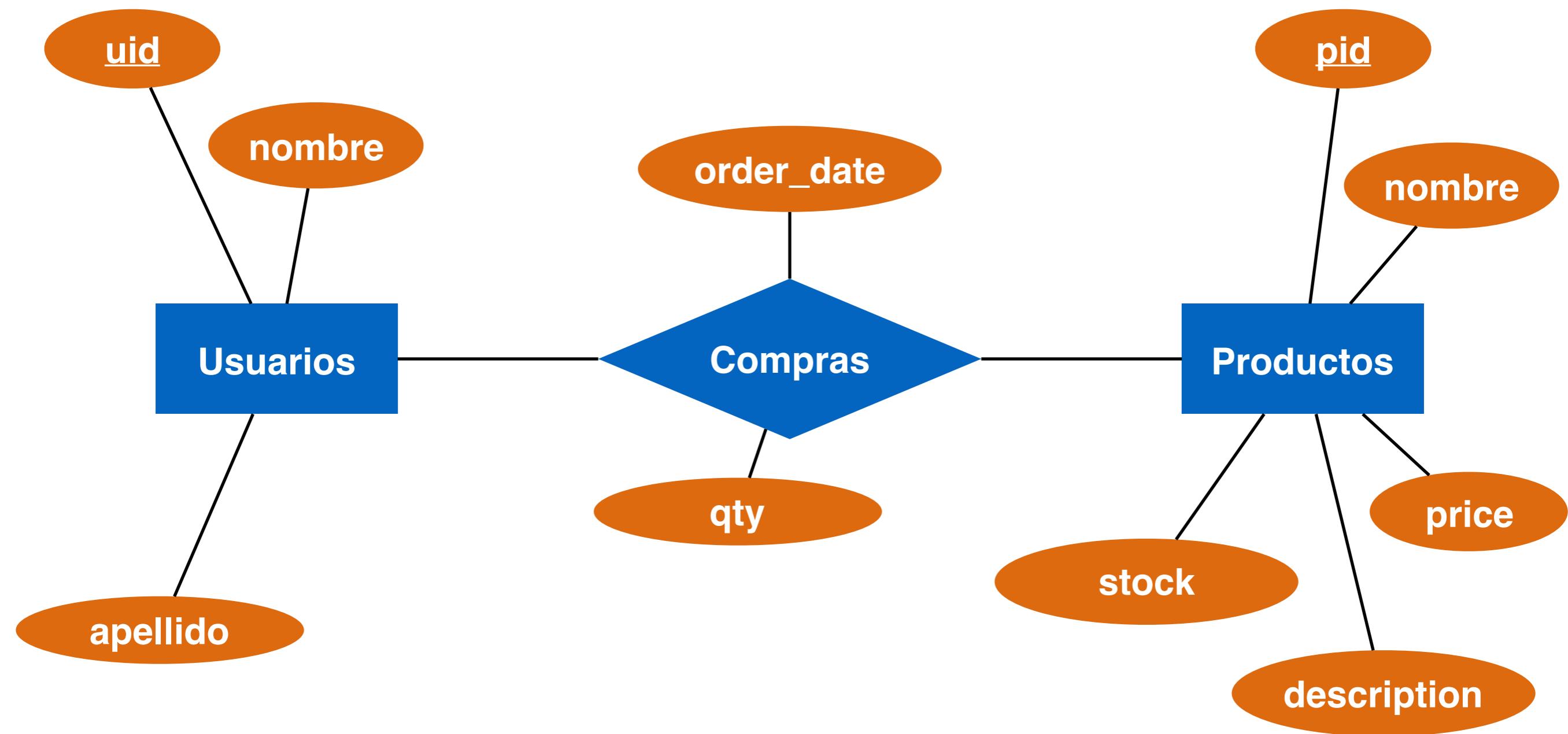
## Observaciones

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

# Diagrama E/R



# Esquema

En el ejemplo anterior, mostramos la declaración de las tablas

El conjunto de estas declaraciones es el esquema



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

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:

# 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
- 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

# 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:

# SQL

## Crear una tabla

Para crear una tabla:

```
CREATE TABLE Users(uid int,  
                  first_name varchar(100),  
                  last_name varchar(100),  
                  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:

# SQL

Insertar valor en una tabla

Para insertar valor en una tabla:

```
INSERT INTO Users(uid, first_name, last_name)  
VALUES(1, 'Don', 'N.')  
;
```

# SQL

Insertar valor en una tabla

Para insertar valor en una tabla:

```
INSERT INTO Users(uid, first_name, last_name)  
VALUES(1, 'Don', 'N.')
```

O bien:

# SQL

Insertar valor en una tabla

Para insertar valor en una tabla:

```
INSERT INTO Users(uid, first_name, last_name)  
VALUES(1, 'Don', 'N.')
```

O bien:

```
INSERT INTO Users  
VALUES(1, 'Don', 'N.')
```



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 Products
```



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

# SQL

## Proyección

Recordemos la tabla de productos:

```
Products(  
    pid: int, name: string,  
    description: string, price: float,  
    stock: int)
```

# SQL

## Proyección

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

```
SELECT name, price FROM Products
```

# SQL

## Proyección

En álgebra relacional lo escribimos como:

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



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 Products  
WHERE price < 1000
```

# SQL

## Selección

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

```
SELECT name, price  
FROM Products  
WHERE price < 1000
```

# SQL

## Selección

En álgebra relacional lo escribimos como:

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

O la versión con proyección

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

# SQL

## Selección

# SQL

## Selección

Las condiciones pueden ser:

# SQL

## Selección

Las condiciones pueden ser:

<, ≤, ≥, >, =, ≠

# SQL

## Selección

Las condiciones pueden ser:

<, ≤, ≥, >, =, ≠

Y se pueden combinar con:

# SQL

## Selección

Las condiciones pueden ser:

<, ≤, ≥, >, =, ≠

Y se pueden combinar con:

∧, ∨



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

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

$R_1.A$	$R_1.B$	$R_2.A$	$R_2.C$	$R_2.D$
$a_1$	$b_1$	$a_1$	$c_1$	$d_1$
$a_1$	$b_1$	$a_2$	$c_2$	$d_2$
$a_2$	$b_2$	$a_1$	$c_1$	$d_1$
$a_2$	$b_2$	$a_2$	$c_2$	$d_2$

# SQL

## Producto Cruz

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

```
SELECT * FROM Products, Orders
```

# SQL

Producto Cruz

En álgebra relacional lo escribimos como:

Products × Orders

# SQL

Join

Para la consulta debemos usar producto cruz y selección

# SQL

## Join

Para la consulta debemos usar producto cruz y selección

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

# 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:

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

# 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 Users, Orders
WHERE Users.uid = Orders.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

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

# SQL

Join

La consulta anterior en SQL se escribe:

# SQL

## Join

La consulta anterior en SQL se escribe:

```
SELECT Users.first_name, Products.name  
FROM Users, Orders, Products  
WHERE Users.uid = Orders.uid  
      AND Orders.pid = Products.pid
```

# SQL

## Otras funciones

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

Veamos ahora algunos ejemplos

# SQL

## Agregación

Para cada día, obtener el número de productos comprados

```
SELECT order_date, sum(qty)
FROM ORDERS
GROUP BY order_date
```

# SQL

Ordernar

Ordenar las compras por fecha:

```
SELECT *
FROM ORDERS
ORDER BY order_date
```

# SQL

Ordernar

Ordenar las compras por fecha:

```
SELECT *
FROM ORDERS
ORDER BY order_date
```

O en orden inverso:

# SQL

## Ordernar

Ordenar las compras por fecha:

```
SELECT *
FROM ORDERS
ORDER BY order_date
```

O en orden inverso:

```
SELECT *
FROM ORDERS
ORDER BY DESC order_date
```

# SQL

Agregación más compleja

Para cada usuario, obtener el dinero que ha gastado

```
SELECT Users.uid, Users.first_name,  
       SUM(Orders.qty*Products.price)  
FROM Users, Products, Orders  
WHERE Users.uid = Orders.uid  
      AND Orders.pid = Products.pid  
GROUP BY Users.uid, Users.first_name;
```

# 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 Users.uid, Users.first_name,  
       SUM(Orders.qty*Products.price)  
FROM Users, Products, Orders  
WHERE Users.uid = Orders.uid  
      AND Orders.pid = Products.pid  
GROUP BY Users.uid, Users.first_name;  
HAVING SUM(Orders.qty*Products.price) > 1000
```

# Consultas con Agregación

```
SELECT <S>
FROM R1, ..., Rn
WHERE <Condición 1>
GROUP BY a1, ..., ak
HAVING <Condición 2>
```

- S puede contener atributos de a<sub>1</sub>, ..., a<sub>k</sub> y/o agregados, pero ningún otro atributo (¿Por qué?)
- Condición 1 es una condición que usa atributos de R<sub>1</sub>, ..., R<sub>n</sub>
- Condición 2 es una condición de agregación de los atributos de R<sub>1</sub>, ..., R<sub>n</sub>

# Consultas con Agregación

## Evaluación

```
SELECT <S>
FROM R1, ..., Rn
WHERE <Condición 1>
GROUP BY a1, ..., ak
HAVING <Condición 2>
```

- Se computa el FROM - WHERE de R<sub>1</sub>, ..., R<sub>n</sub>
- Agrupar la tabla por los atributos de a<sub>1</sub>, ..., a<sub>k</sub>
- 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



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



Ejemplo en vivo





Doña V:

- Quiere comprar accesorios para mascotas
- Como el común de los mortales, no tiene por qué conocer el funcionamiento de la base de datos



Me gustaría poder comprar mis  
accesorios en una tienda en línea



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

# Un poco de historia

# Un poco de historia

Web 1.0: páginas estáticas

Web 2.0: interacciones con bases de datos

Web 3.0: web semántica

# SQL + Programación

# SQL + Programación

Es posible conectar un motor SQL a los lenguajes de programación (Python, Java, C#, etc)

# SQL + Programación

Es posible conectar un motor SQL a los lenguajes de programación (Python, Java, C#, etc)

Las aplicaciones que comúnmente usamos, se conectan a una base de datos (muchas de ellas, una base de datos SQL)

# SQL + Programación

# SQL + Programación

El usuario final no conoce como se administran ni  
almacenan los datos

# SQL + Programación

El usuario final no conoce como se administran ni  
almacenan los datos

Hay una **abstracción** para el usuario, quien se encarga  
de ocupar el programa



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?



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

# ORM

# ORM

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

# 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:

# ORM

## Ejemplo - Consultas

Obtener todos los músicos:

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

# ORM

## Ejemplo - Consultas

Obtener todos los músicos:

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

Obtener todos los músicos con nombre ‘James’:

# ORM

## Ejemplo - Consultas

Obtener todos los músicos:

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

Obtener todos los músicos con nombre 'James':

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

# 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:

# 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

# 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

# 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



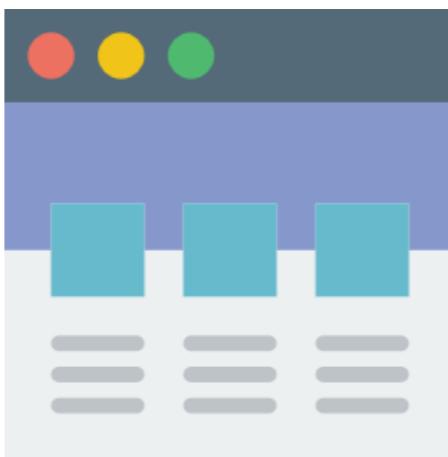


*“Es importante conocer cómo funciona SQL para construir aplicaciones de calidad”*



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

Profesor Oak, septiembre 2019



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



Don N., su aplicación está  
funcionando demasiado lenta,  
quizás le está faltando...

**Big Data** **Cloud Computing**  
**Blockchain** **Quantum Computing**  
**Buzzword** **Machine Learning** **Microservices**  
**Deep Learning** **Internet of Things**  
**Data Science**



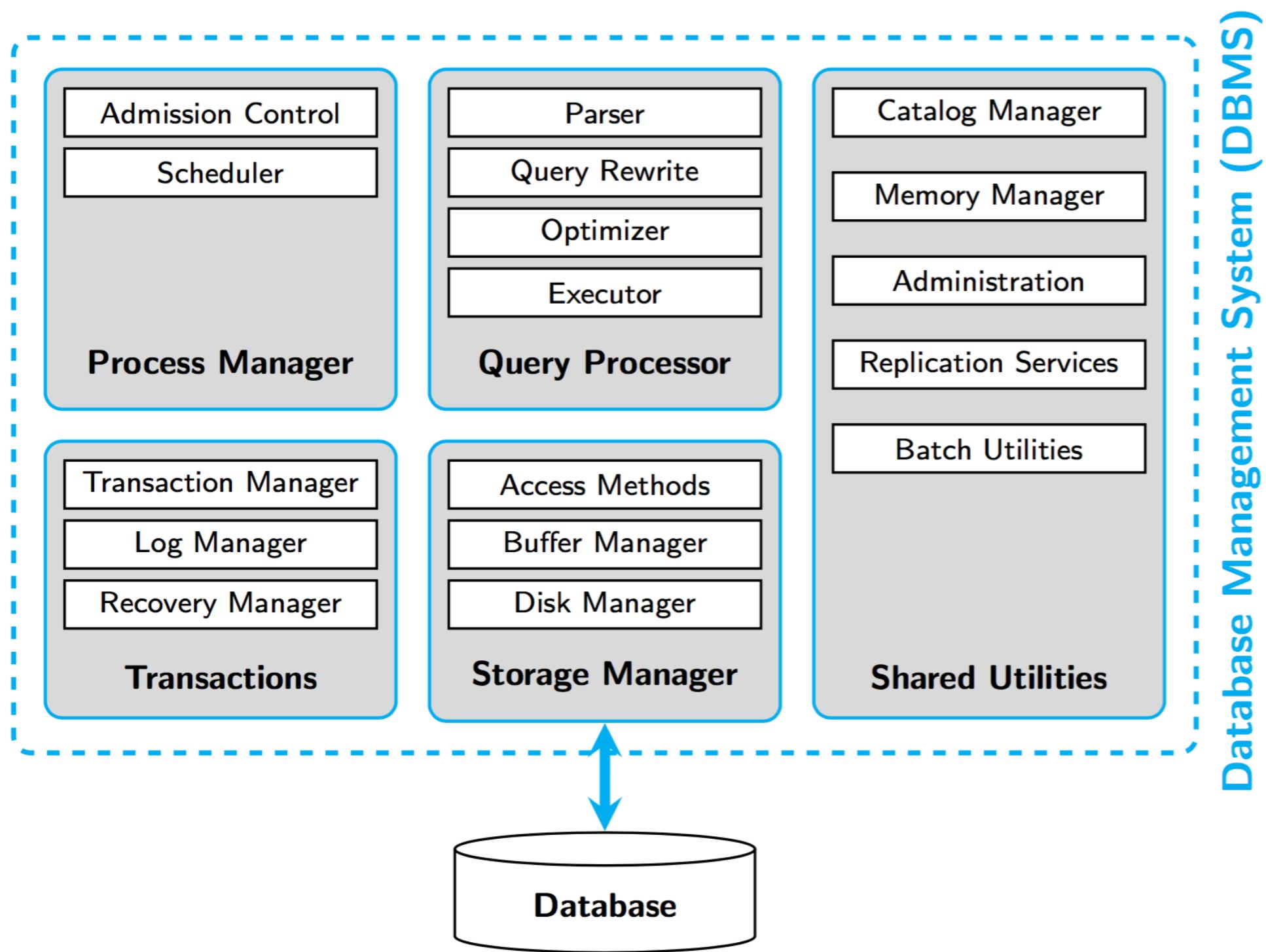


Profesor, me han contado que hay que usar NoSQL, porque funcionan mucho más rápido



A veces los problemas no son  
lo que parecen

# Cómo funciona una base de datos



# Componentes de un DBMS

- Índices (B+Trees, Hash Index)
- Transacciones (propiedades ACID)
- Optimizador de consultas
- ...

# ¿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
- ...

# ¿Es SQL suficiente?

Aplicaciones grandes, como una red social de millones de usuarios, o un laboratorio de millones de registros, necesita técnicas diferentes

Pero una aplicación típica puede funcionar perfecto con una base de datos SQL!

# ¿Es SQL suficiente?

Además, para funcionar más rápido hay que transar algunas cosas, como la **consistencia** de los datos

En general, para asegurar la integridad de los datos, es recomendable usar un sistema SQL



Y recuerde Don N., hay que  
tener cuidado con la  
publicidad

Rank			DBMS	Database Model	Score		
Apr 2018	Mar 2018	Apr 2017			Apr 2018	Mar 2018	Apr 2017
1.	1.	1.	Oracle 	Relational DBMS	1289.79	+0.18	-112.21
2.	2.	2.	MySQL 	Relational DBMS	1226.40	-2.46	-138.22
3.	3.	3.	Microsoft SQL Server 	Relational DBMS	1095.51	-9.28	-109.26
4.	4.	4.	PostgreSQL 	Relational DBMS	395.47	-3.88	+33.69
5.	5.	5.	MongoDB 	Document store	341.41	+0.89	+15.98
6.	6.	6.	DB2 	Relational DBMS	188.95	+2.28	+2.29
7.	7.	7.	Microsoft Access	Relational DBMS	132.22	+0.27	+4.04
8.	↑ 9.	↑ 11.	Elasticsearch 	Search engine	131.36	+2.81	+25.69
9.	↓ 8.	9.	Redis 	Key-value store	130.11	-1.12	+15.75
10.	10.	↓ 8.	Cassandra 	Wide column store	119.09	-4.40	-7.10
11.	11.	↓ 10.	SQLite 	Relational DBMS	115.99	+1.17	+2.19
12.	12.	12.	Teradata	Relational DBMS	73.68	+1.21	-2.88
13.	13.	↑ 17.	Splunk	Search engine	65.06	-0.61	+9.55
14.	↑ 15.	↑ 18.	MariaDB 	Relational DBMS	64.56	+1.45	+15.83
15.	↓ 14.	↓ 14.	Solr	Search engine	63.21	-1.60	-1.16
16.	16.	↓ 13.	SAP Adaptive Server 	Relational DBMS	61.63	-0.99	-5.83
17.	17.	↓ 15.	HBase 	Wide column store	59.69	-1.24	+1.22

Fuente: [http://db-engines.com/en/ranking]

# Tipos de DBMS

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

# Tipos de DBMS

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

# Tipos de DBMS

[PRODUCTS](#)[SOLUTIONS](#)[PARTNERS](#)[CUSTOMERS](#)[LEARN](#)[DEVELOPERS](#)

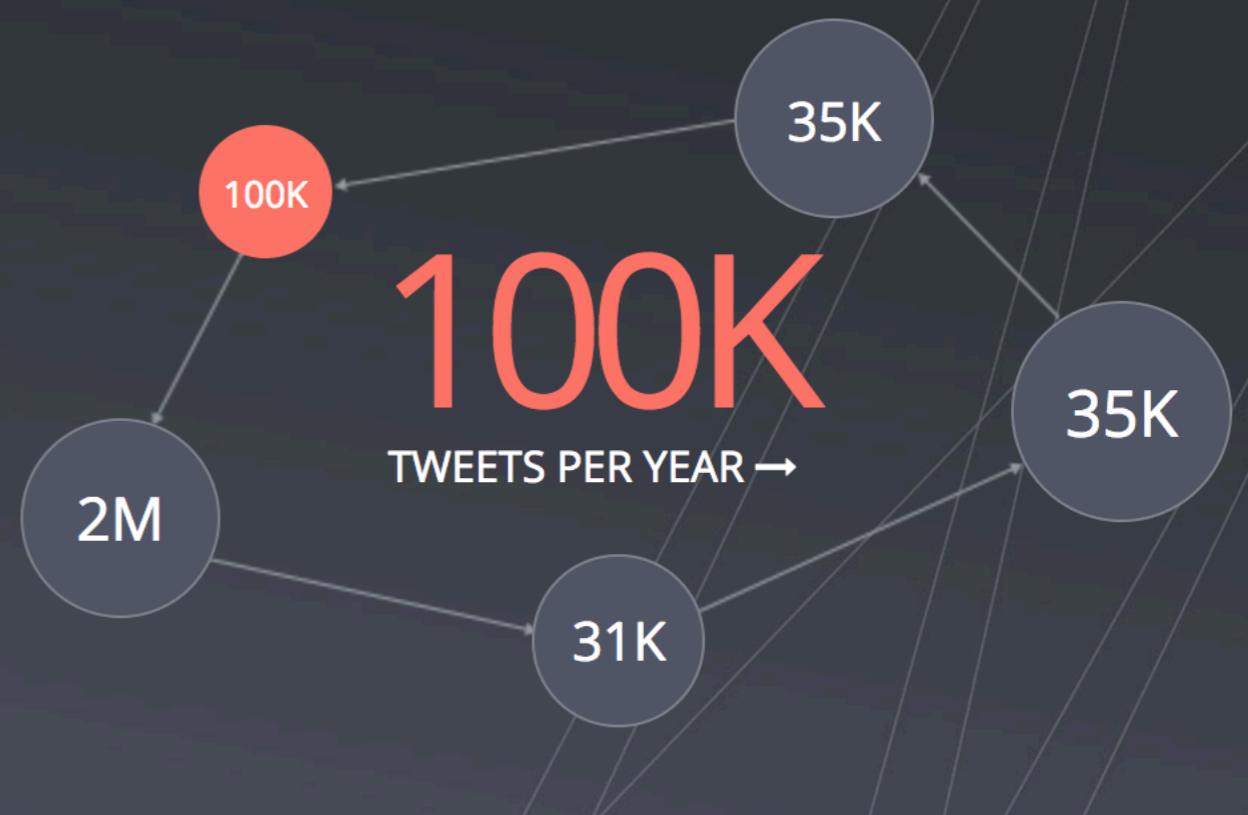
Search

(Neo4j)-[:LOVES]-(Developers)

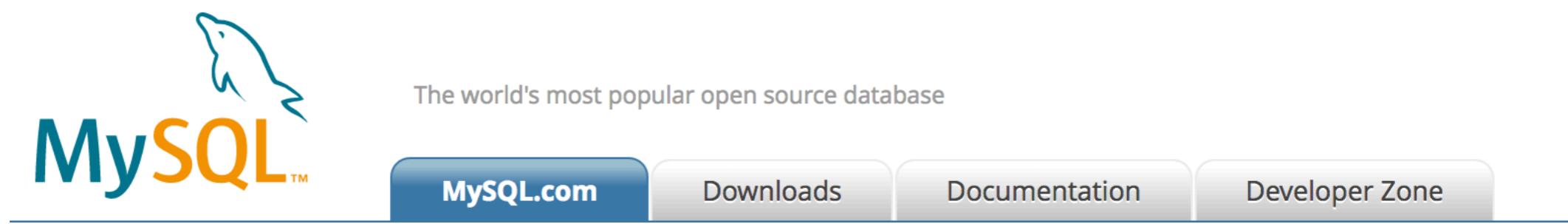
World's leading graph database, with native graph storage and processing.

Property graph model and Cypher query language makes it easy to understand.

Fast. Natural. Fun.



# Tipos de DBMS



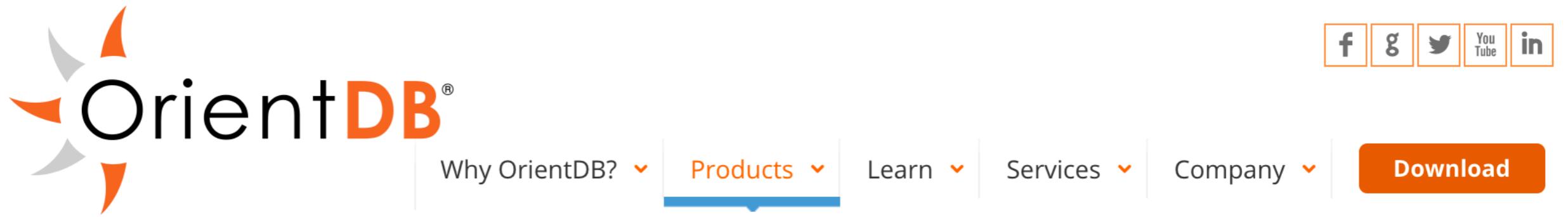
# Tipos de DBMS



[Home](#) [Download](#) [Documentation](#) [Community](#)

Manage massive amounts of data,  
fast, without losing sleep

# Tipos de DBMS



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

# Tipos de DBMS



SOLUTIONS CLOUD

## Do What You Could Never Do Before

What's been holding you back? MongoDB is the next-generation database that lets you create applications never before possible.

# Tipos de DBMS



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

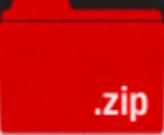
SQLite is a [self-contained](#), [high-reliability](#), [embedded](#), [full-featured](#), [public-domain](#), SQL database engine. SQLite is the [most used](#) database engine in the world. [More Info](#)

# Tipos de DBMS

[View Documentation](#)  [On Github](#) 

## TITAN

Distributed Graph Database

 .zip

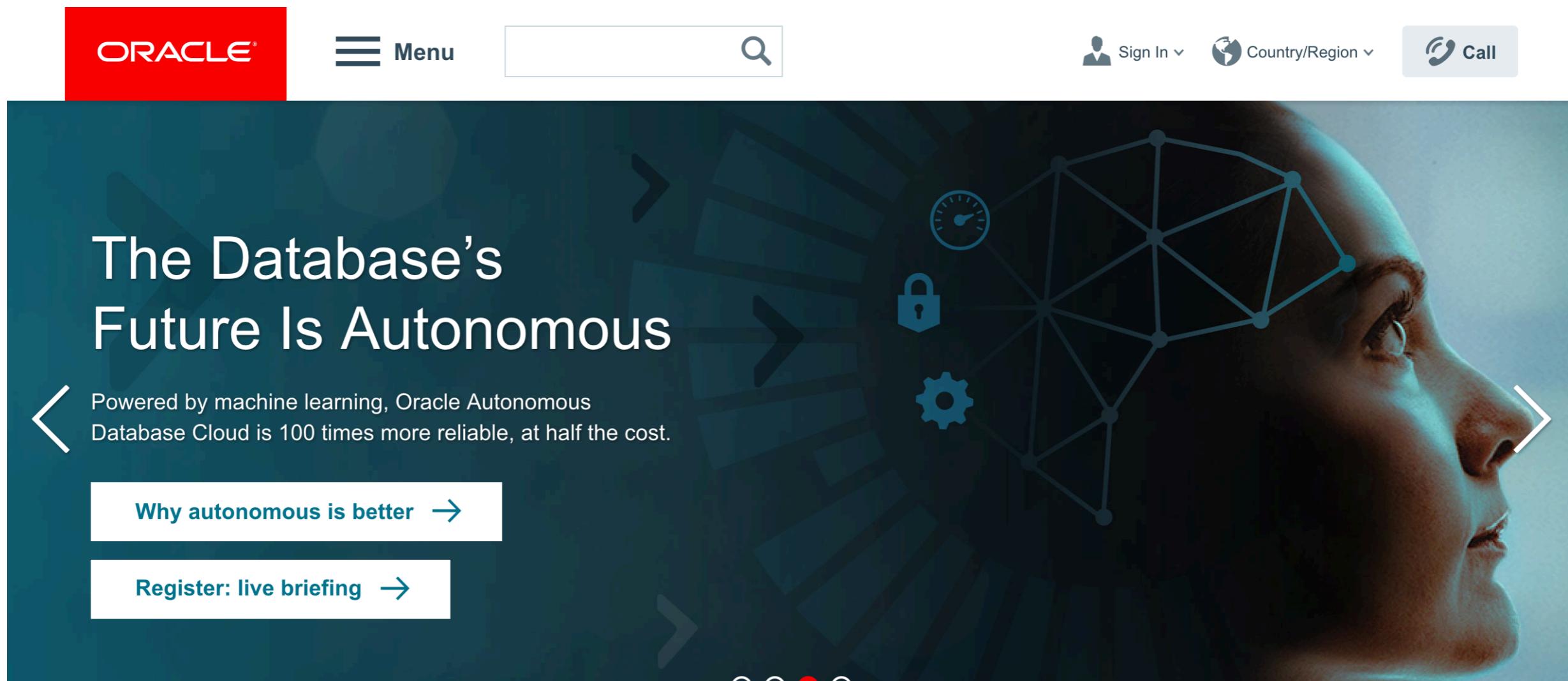
Titan is a scalable **graph database** optimized for storing and querying graphs containing hundreds of billions of vertices and edges distributed across a multi-machine cluster. Titan is a transactional database that can support **thousands of concurrent users** executing **complex graph traversals** in real time.

In addition, Titan provides the following features:

- Elastic and linear scalability for a growing data and user base.
- Data distribution and replication for performance and fault tolerance.
- Multi-datacenter high availability and hot backups.
- Support for **ACID** and **eventual consistency**.
- Support for various **storage backends**:
  - **Apache Cassandra**
  - **Apache HBase**
  - **Oracle BerkeleyDB**



# Tipos de DBMS



The image shows the Oracle Autonomous Database landing page. At the top, there is a red header bar with the "ORACLE" logo, a "Menu" icon, a search bar, and user account options for "Sign In", "Country/Region", and "Call". The main content area features a dark blue background with a profile of a woman's head on the right. Overlaid on the background are several white arrows pointing right, a network graph, a padlock icon, and a gear icon. The text "The Database's Future Is Autonomous" is prominently displayed in large white font. Below it, a callout states: "Powered by machine learning, Oracle Autonomous Database Cloud is 100 times more reliable, at half the cost." Two buttons are visible: "Why autonomous is better →" and "Register: live briefing →".

ORACLE

Menu

Sign In

Country/Region

Call

The Database's Future Is Autonomous

Powered by machine learning, Oracle Autonomous Database Cloud is 100 times more reliable, at half the cost.

Why autonomous is better →

Register: live briefing →

# Tipos de DBMS



## Welcome to Apache HBase™

Apache  HBase™ is the [Hadoop](#)  database, a distributed, scalable, big data store.

Use Apache HBase™ when you need random, realtime read/write access to your Big Data. This project's goal is the hosting of very large tables -- billions of rows X millions of columns -- atop clusters of commodity hardware. Apache HBase is an open-source, distributed, versioned, non-relational database modeled after Google's [Bigtable: A Distributed Storage System for Structured Data](#)  by Chang et al. Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

# Tipos de DBMS

THE EVOLUTION OF THE DATABASE | MARKLOGIC 9

## Your Data Isn't the Problem. It's Your Database.

MarkLogic is the world's best database for integrating data from silos, and the only NoSQL solution built for the enterprise.

PRODUCT    SOLUTIONS    LEARN    COMMUNITY    COMPANY    FREE TRIAL

# Resumen

# 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



Ahora es momento de unos  
anuncios publicitarios

# Datalab UC

Profesores



Marcelo Arenas



Domagoj Vrgoč



Juan L. Reutter



Cristian Riveros

# Datalab UC

Temas de Investigación

# Datalab UC

Temas de Investigación

- Bases de Datos
- Linked Data
- Teoría de computación
- Blockchain
- Data Science

# Introducción a las bases de datos

Exploratorio en Computación  
Invitado: Adrián Soto