Introducción a las bases de datos

Exploratorio en Computación
Invitado: Valentina Álvarez

¿Por qué bases de datos?

Un día cualquiera (13 de abril)

8:30 am	Despierto, reviso Telegram
10:00 am	Clases, a través de Zoom
12:00 pm	Supermercado, pago con Tarjeta
12:30 pm	Busco productos en Tiendas en línea
13:30 pm	Actualizo archivos de Drive
16:00 pm	Subo una historia a Instagræm
18:00 pm	Veo una serie en Netflix

Un día cualquiera (13 de abril)

8:30 am	Despierto, reviso **Telegram**
10:00 am	Clases, a través de **Zoom**
12:00 pm	Supermercado, pago con **Tarjeta**
12:30 pm	Busco productos en **Tiendas en línea**
13:30 pm	Actualizo archivos de **Drive**
16:00 pm	Subo una historia a **Instagræm**
18:00 pm	Veo una serie en **Netflix**

Un día cualquiera (13 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

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

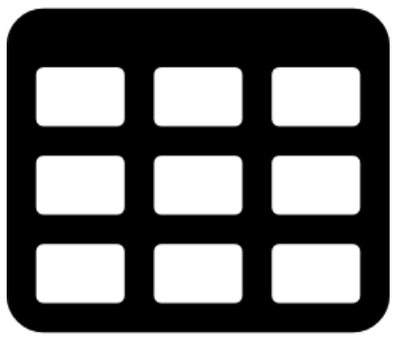
ID Actor	Nombre Actor
1	Leonardo DiCaprio
2	Matthew McConaughey
3	Daniel Radcliffe
4	Jessica Chastain
…	…

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

ID Actor	ID Película
1	2
1	4
2	1
3	3
…	…

Cómo funciona un DBMS

¿Cuál es la mejor película de Christopher Nolan?



Interstellar
~~The Dark Knight~~

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 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

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

U01 # Conito # XXXX 1234
U02 # Don N. # XXXX 2345
U03 # Profesor A # XXXX 3456
U04 # Tía Pame # XXXX 4567
U05 # Frambuesa # XXXX 5678
…

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

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: …

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!

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?

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

Ventajas de un DBMS

- Acceso eficiente a los datos

Ventajas de un DBMS

- 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?

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

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:

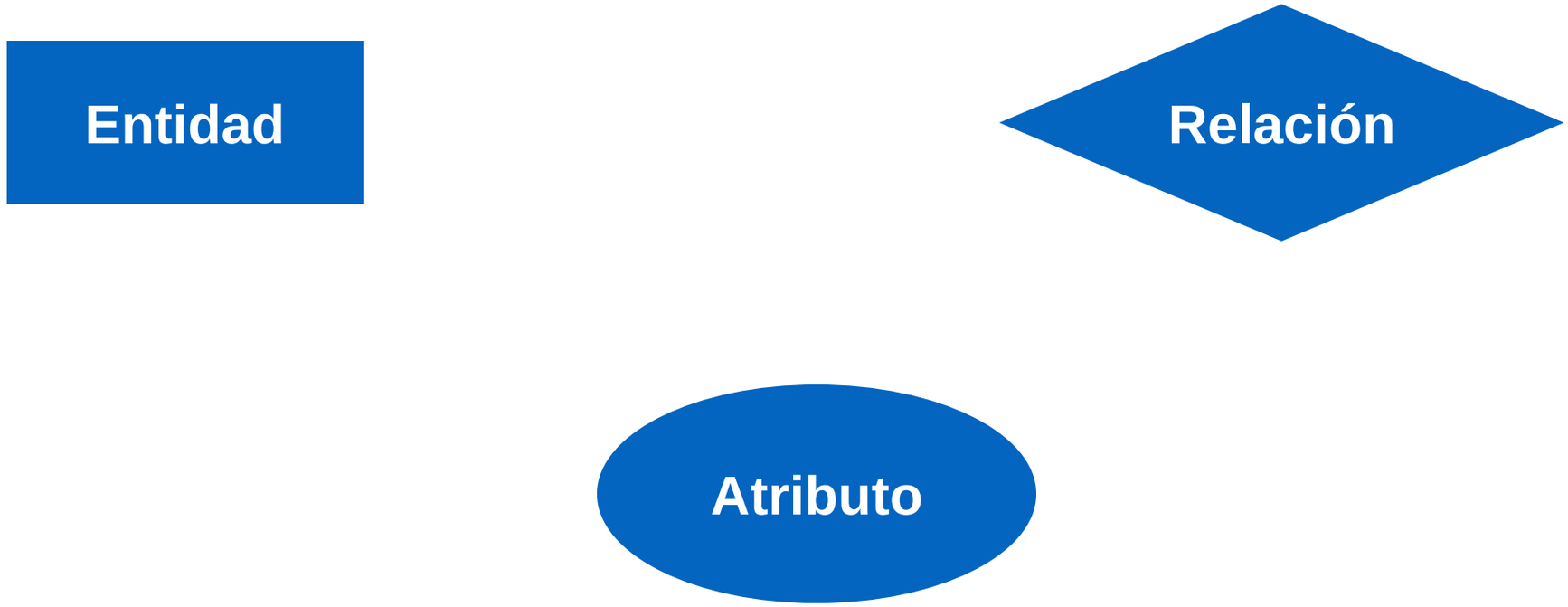
- Usuarios
- Productos
- Compras

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



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

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

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

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

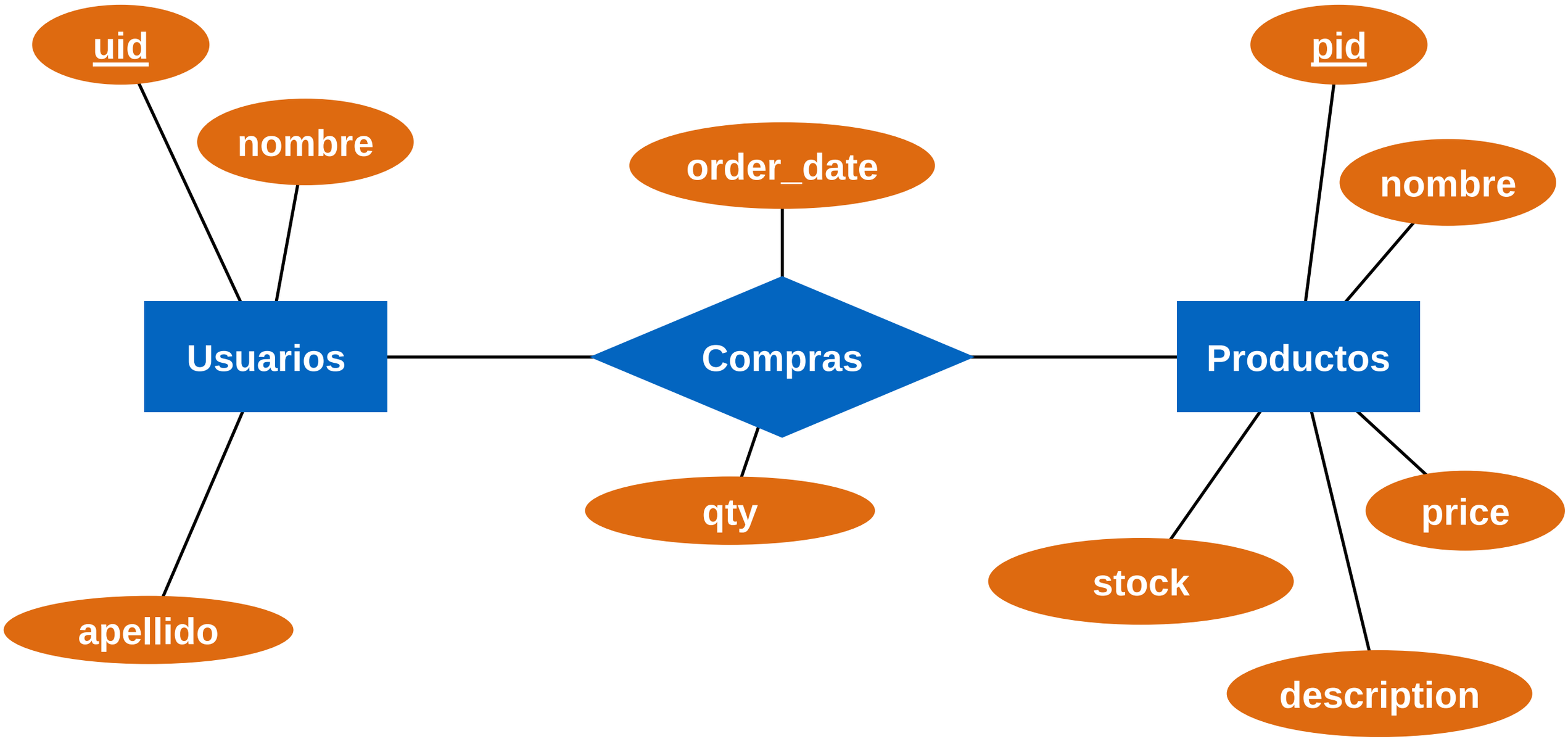
Observaciones

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


uid
nombre
order_date
pid
nombre
Usuarios
Compras
Productos
qty
apellido
stock
price
description

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

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

- El DBMS transforma la consulta SQL en 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:

```
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

R_1	A	B
	a_1	b_1
	a_2	b_2

$\times$

R_2	A	C	D
	a_1	c_1	d_1
	a_2	c_2	d_2

$=$

$R_1 \times R_2$

$R_1.A$	$R_1.B$	$R_2.A$	$R_2.C$	$R_2.D$
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 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

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

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:

$$\text{Users} \bowtie \text{Orders}$$

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

Ordernar

Ordenar las compras por fecha:

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

O en orden inverso:

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

SQL

Ordernar

```
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

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 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 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	2650
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	2195
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	30

SQL

Agregación más compleja

```
SELECT

FROM Usuarios, Productos, Compras

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

GROUP 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	2650
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	2195
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	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 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, …, R_n

WHERE <Condición 1>

GROUP BY a_1, …, a_k

HAVING <Condición 2>

- S puede contener atributos de a_1, …, a_k y/o agregados, pero ningún otro atributo
- Condición 1 es una condición que usa atributos de R_1, …, R_n
- Condición 2 es una condición de agregación de los atributos de R_1, …, R_n

Consultas con Agregación

Evaluación

SELECT <S>

FROM R_1, …, R_n

WHERE <Condición 1>

GROUP BY a_1, …, a_k

HAVING <Condición 2>

- Se computa el FROM - WHERE de R_1, …, R_n
- Agrupar la tabla por los atributos de a_1, …, 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