



ANÁLISIS Y DISEÑO LÓGICO DE SISTEMAS

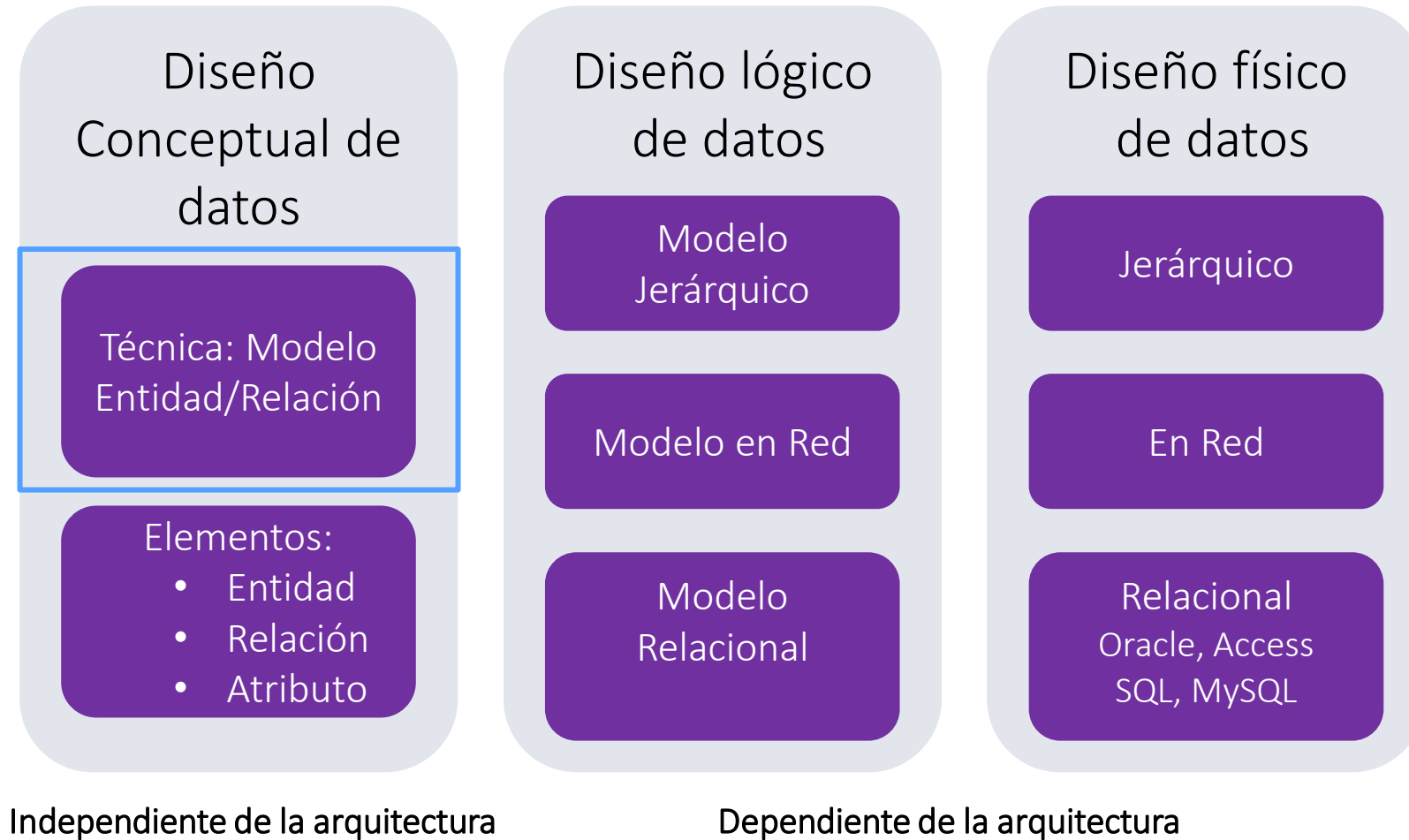
Octubre 2023

Luis E.Canales C.
lcanales@utalca.cl

Conectemos



Proceso de construcción de una BD



SQL



SueIdos(UserId, Nombre, Trabajo, Salario)

Componentes del Modelo Relacional



Diagram illustrating the components of a Relational Model:

- Tabla/Relación** (Table/Relation): Points to the entire table structure.
- Columnas/Atributos/Campos** (Columns/Attributes/Fields): Points to the header row of the table.
- Filas/Tuplas/Registros** (Rows/Tuples/Records): Points to the data rows of the table.

| Sueños | | | |
|---------------|--------|-----------|---------|
| UserID | Nombre | Trabajo | Salario |
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Sueños(UserId, Nombre, Trabajo, Salario)

Características del Modelo Relacional



- Establece una semántica
 - No hay tuplas duplicadas
- Los atributos son Typeados y estáticos
 - INTEGER, FLOAT, VARCHAR (n), DATETIME, ...
- Las tablas son planas

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Características del Modelo Relacional



- Establece una semántica
 - No hay tuplas duplicadas
- Los atributos son Typeados y estáticos
 - INTEGER, FLOAT, VARCHAR (n), DATETIME, ...
- Las tablas son planas

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

=

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 567 | Magda | Profesora | 900000 |
| 123 | Juan | Contador | 500000 |
| 789 | Diana | Profesora | 1000000 |
| 345 | Aline | Contador | 600000 |

Características del Modelo Relacional



- Establece una semántica
 - No hay tuplas duplicadas
- Los atributos son Typeados y estáticos
 - INTEGER, FLOAT, VARCHAR (n), DATETIME, ...
- Las tablas son planas

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |
| 789 | Diana | Profesora | 1000000 |

¡Viola la
semántica
establecida!

Características del Modelo Relacional



- Establece una semántica
- No hay tuplas duplicadas
- Los atributos son Typeados y estáticos
- INTEGER, FLOAT, VARCHAR (n), DATETIME, ...
- Las tablas son planas

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | Manzana |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

¡Viola el tipo de atributo, asumiendo que es entero!

Componentes del Modelo Relacional



- Pero, ¿cómo se almacenan estos datos REALMENTE?

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

No nos importa en este curso, eso se ve en estructuras de datos!
Independencia de datos físicos

Structured Query Language - SQL



- Muy bien, tengo datos y un esquema. ¿Cómo accedo a éstos?

Structured Query Language - SQL



SQL (por sus siglas en inglés Structured Query Language; en español lenguaje de consulta estructurada) es un lenguaje de dominio específico utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales. Una de sus principales características es el manejo del álgebra y el cálculo relacional para efectuar consultas con el fin de recuperar, de forma sencilla, información de bases de datos, así como realizar cambios en ellas.

Originalmente basado en el álgebra relacional y en el cálculo relacional, SQL consiste en un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de control de datos. El alcance de SQL incluye la inserción de datos, consultas, actualizaciones y borrado, la creación y modificación de esquemas y el control de acceso a los datos. También el SQL a veces se describe como un lenguaje declarativo, también incluye elementos procesales.

Structured Query Language - SQL



SQL fue uno de los primeros lenguajes comerciales para el modelo relacional de Edgar Frank Codd como se describió en su artículo de investigación de 1970 El modelo relacional de datos para grandes bancos de datos compartidos. A pesar de no adherirse totalmente al modelo relacional descrito por Codd, pasó a ser el lenguaje de base de datos más usado.

SQL pasó a ser el estándar del Instituto Nacional Estadounidense de Estándares (ANSI) en 1986 y de la Organización Internacional de Normalización (ISO) en 1987. Desde entonces, el estándar ha sido revisado para incluir más características. A pesar de la existencia de ambos estándares, la mayoría de los códigos SQL no son completamente portables entre sistemas de bases de datos diferentes sin ajustes.

Structured Query Language - SQL



Information Retrieval

P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. CODD

IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the “connection trap”).

Structured Query Language - SQL



1. Relational Model and Normal Form

1.1. INTRODUCTION

This paper is concerned with the application of elementary relation theory to systems which provide shared access to large banks of formatted data. Except for a paper by Childs [1], the principal application of relations to data systems has been to deductive question-answering systems. Levein and Maron [2] provide numerous references to work in this area.

In contrast, the problems treated here are those of *data independence*—the independence of application programs and terminal activities from growth in data types and changes in data representation—and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

lections of data (as opposed to individual items). Three of the principal kinds of data dependencies which still need to be removed are: ordering dependence, indexing dependence, and access path dependence. In some systems these dependencies are not clearly separable from one another.

1.2.1. *Ordering Dependence.* Elements of data in a data bank may be stored in a variety of ways, some involving no concern for ordering, some permitting each element to participate in one ordering only, others permitting each element to participate in several orderings. Let us consider those existing systems which either require or permit data elements to be stored in at least one total ordering which is closely associated with the hardware-determined ordering of addresses. For example, the records of a file concerning parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

SQL



- Structured Query Language
 - Uno de los muchos lenguajes para consultar datos relacionales
- El lenguaje más utilizado para consultar datos relacionales
 - Un lenguaje de programación declarativo

Nota

SQL es un lenguaje enorme!. Cubriremos los conceptos importantes aquí, pero usted tendrá que buscar los comandos que no cubrimos en clase.

En esta hoja se simplifica muy bien la gran mayoría de funcionalidades que ofrece SQL



@swapnakpanda

Categories

DDL : Data Definition Language
DQL : Data Query Language
DML : Data Manipulation Language
DCL : Data Control Language
TCL : Transaction Control Language

Commands

DDL

CREATE | DROP | ALTER | TRUNCATE
RENAME | COMMENT

DQL

SELECT

DML

INSERT | UPDATE | DELETE | LOCK
CALL | EXPLAIN PLAN

DCL

GRANT | REVOKE

TCL

COMMIT | ROLLBACK
SAVEPOINT | SET TRANSACTION

Operators

Arithmetic

+ - * / %

Bitwise

& | ^

Comparison

= < > <= >= != <> !=

Compound

+= -= *= /= %= &= |= ^=

Logical

AND | OR | NOT | ANY
SOME | ALL | BETWEEN
IN | EXISTS | LIKE
IS NULL | UNIQUE

Important Keywords

WHERE | DISTINCT | LIMIT
ORDER BY | DESC | ASC
AS | FROM | SET | VALUES
CASE | DEFAULT

SQL CHEATSHEET

- ✓ Genuine
- ✓ Authentic
- ✓ Quality

Database Objects

TABLE | VIEW | SYNONYM
SEQUENCE | INDEX | TRIGGER

Constraints

NOT NULL | UNIQUE
PRIMARY KEY | FOREIGN KEY
CHECK | DEFAULT

Aggregation Functions

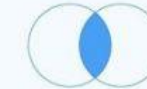
AVG | COUNT
MAX | MIN | SUM

Aggregation Keywords

GROUP BY | HAVING

Joins

INNER JOIN



LEFT [OUTER] JOIN



FULL [OUTER] JOIN



RIGHT [OUTER] JOIN

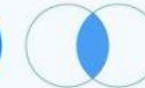


Set Operations

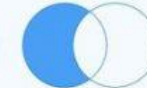
UNION
UNION ALL



INTERSECT



EXCEPT
MINUS



DDL Examples

Create a Table

```
CREATE TABLE Students(  
  rollno int PRIMARY KEY,  
  fname varchar(255) NOT NULL,  
  lname varchar(255)  
);
```

Adding a new column to the Table

```
ALTER TABLE Students  
ADD email varchar(255);
```

Modifying the data type of existing column

```
ALTER TABLE Students  
ALTER COLUMN lname varchar(512);
```

Removing an existing column from the Table

```
ALTER TABLE Students  
DROP COLUMN email;
```

Truncate (remove all data) a Table

```
TRUNCATE TABLE Students;
```

Drop a Table

```
DROP TABLE Students;
```

DQL Examples

Fetch all data from a Table

```
SELECT * FROM Students;
```

Filter data from a Table

```
SELECT * FROM Students  
WHERE rollno=1234;  
  
SELECT * FROM Students  
WHERE rollno>1234  
AND age < 15;
```

Fetch selected columns

```
SELECT fname, lname  
FROM Students  
WHERE rollno>1234  
AND age < 15;
```

Fetch maximum 10 rows

```
SELECT fname, lname  
FROM Students  
WHERE rollno>1234  
AND age < 15  
LIMIT 10;
```

Fetch count of records

```
SELECT count(*)  
FROM Students;
```

Fetch Maximum Age

```
SELECT max(age)  
FROM Students;
```

Fetch Minimum Age

```
SELECT min(age)  
FROM Students;
```

Fetch Sum of Age

```
SELECT sum(age)  
FROM Students;
```

Fetch Average Age

```
SELECT avg(age)  
FROM Students;
```

Fetch Average Age for each gender

```
SELECT avg(age)  
FROM Students  
GROUP BY gender;
```

Sort (order) fetched records

```
SELECT fname, lname  
FROM Students  
WHERE rollno>1234  
AND age < 15  
ORDER BY gender;
```

Sort in descending order

```
SELECT fname, lname  
FROM Students  
WHERE rollno>1234  
AND age < 15  
ORDER BY gender DESC;
```

Fetch from 2 Tables

```
SELECT fname, clsteacher  
FROM Students  
INNER JOIN Section  
ON Students.section  
= Section.id;
```

DML Examples

Insert data (rows) into a Table

```
INSERT INTO Students(rollno, fname, lname)  
VALUES (1234, 'Christiano', 'Ronaldo');
```

Update data (value of column) of a Table

```
UPDATE Students SET lname = 'Messi'  
WHERE rollno=1234;
```

Delete data (rows) from a Table

```
DELETE FROM Students WHERE rollno=1234;
```

Aggregate and, Filter

```
SELECT section, count(*) AS studentcount  
FROM Students  
GROUP BY section  
HAVING count(*) > 20;
```

Full Outer Join

```
SELECT fname, clsteacher  
FROM Students  
FULL JOIN Section  
ON Students.section = Section.id;
```

Hello World!



Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

```
SELECT S.Nombre, S.UserID  
FROM Sueldos AS S  
WHERE S.Trabajo = 'Contador';
```

SELECT

Qué tipo de datos
quiero

FROM

De dónde provienen
los datos

WHERE

Filtra los datos

Hello World!



Sueños

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

```
SELECT S.Nombre, S.UserID
FROM Sueños AS S
WHERE S.Trabajo = 'Contador';
```



| Nombre | UserID |
|--------|--------|
| Juan | 123 |
| Aline | 345 |

SQL

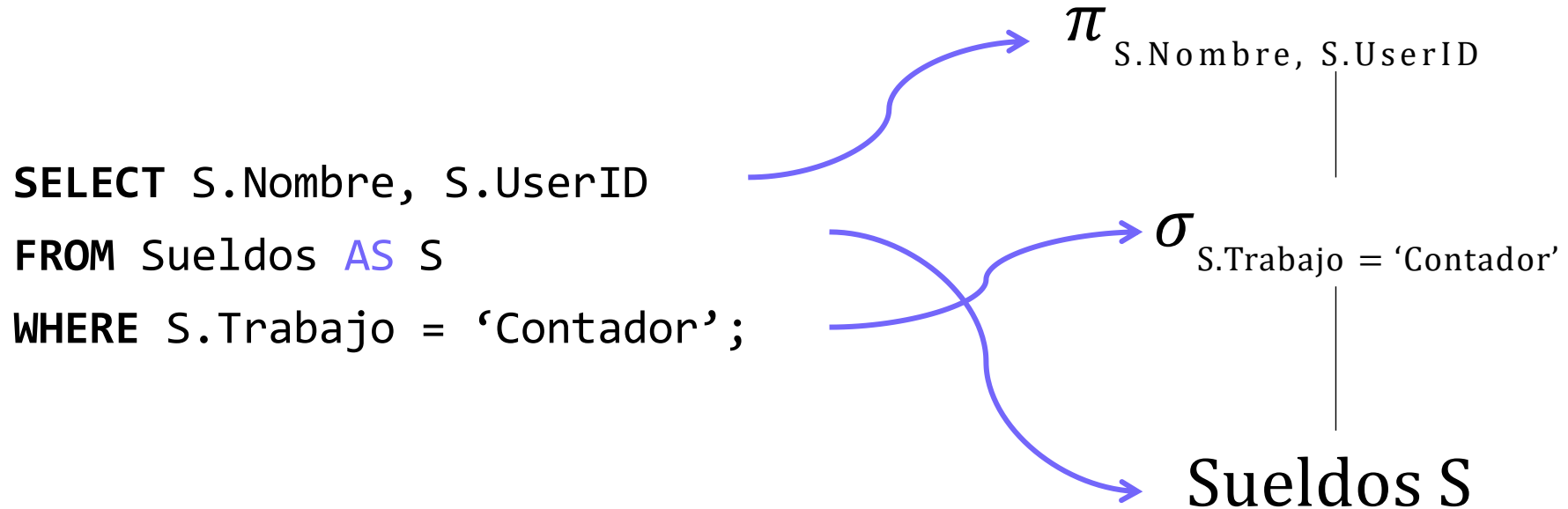


¿Cómo el
computador
entiende el
texto abstracto
de SQL?

Manejo interno de la Base de Datos



- El código tiene que reducirse a instrucciones en algún momento
- Los sistemas de gestión de bases de datos relacionales (RDBMS) utilizan álgebra relacional (AR)



Manejo interno de la Base de Datos



- El código tiene que reducirse a instrucciones en algún momento
- Los sistemas de gestión de bases de datos relacionales (RDBMS) utilizan álgebra relacional (AR)

$\pi_{S.Nombre, S.UserID}$

Para cada Semántica

$\sigma_{S.Trabajo = 'Contador'}$

`for` each fila in S:

`if` (fila.Trabajo == 'Contador'):

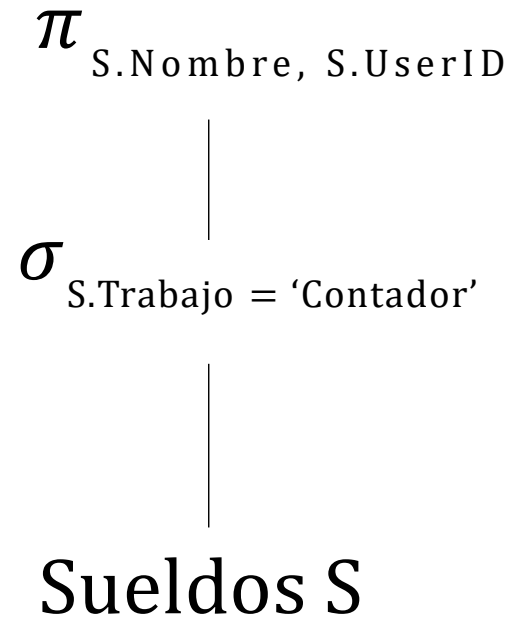
`output` (fila.Nombre, fila.UserID)

Sueldos S

Manejo interno de la Base de Datos



- El código tiene que reducirse a instrucciones en algún momento
- Los sistemas de gestión de bases de datos relacionales (RDBMS) utilizan álgebra relacional (AR)



Las tuplas "fluyen"
hacia el árbol RA
filtrado y modificado

Hello World!

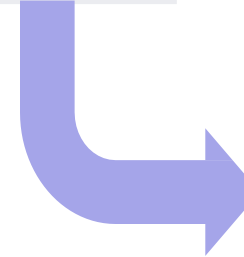


Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



```
SELECT S.Nombre, S.UserID  
FROM Sueldos AS S  
WHERE S.Trabajo = 'Contador';
```



| | |
|--------|--------|
| Nombre | UserID |
|--------|--------|

Hello World!

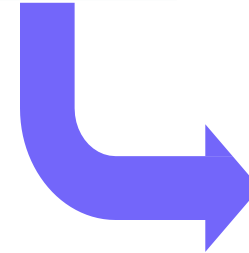


Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



```
SELECT S.Nombre, S.UserID  
FROM Sueldos AS S  
WHERE S.Trabajo = 'Contador';
```



| Nombre | UserID |
|--------|--------|
| Juan | 123 |

Hello World!

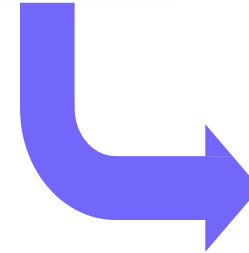


Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



```
SELECT S.Nombre, S.UserID  
FROM Sueldos AS S  
WHERE S.Trabajo = 'Contador';
```



| Nombre | UserID |
|--------|--------|
| Juan | 123 |

Hello World!

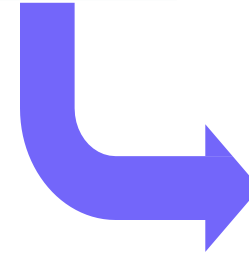


Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



```
SELECT S.Nombre, S.UserID
FROM Sueldos AS S
WHERE S.Trabajo = 'Contador';
```



| Nombre | UserID |
|--------|--------|
| Juan | 123 |
| Aline | 345 |

Hello World!

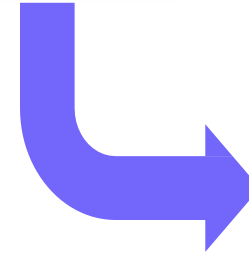


Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



```
SELECT S.Nombre, S.UserID
FROM Sueldos AS S
WHERE S.Trabajo = 'Contador';
```



| Nombre | UserID |
|--------|--------|
| Juan | 123 |
| Aline | 345 |

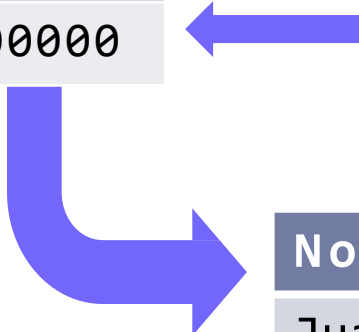
Hello World!



Sueldos

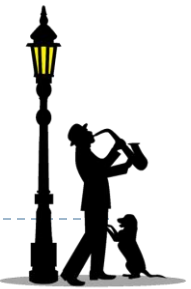
| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

```
SELECT S.Nombre, S.UserID
FROM Sueldos AS S
WHERE S.Trabajo = 'Contador';
```



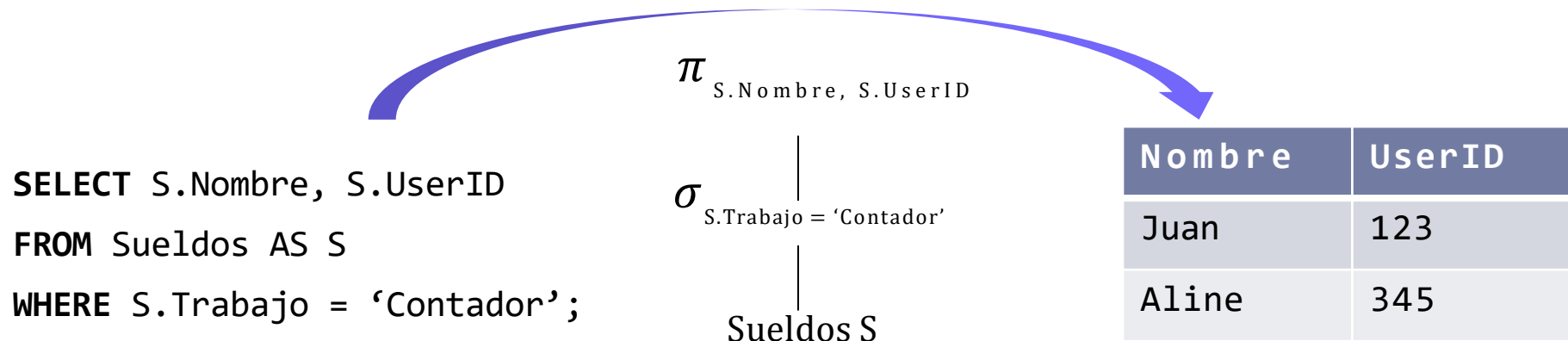
| Nombre | UserID |
|--------|--------|
| Juan | 123 |
| Aline | 345 |

En resumen!



- El concepto de modelo relacional
- Cómo funciona una consulta básica SELECT-FROM-WHERE
- Proceso de ejecución básico (AR) dentro de un RDBMS

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Sentencia: Crear Tabla

Sueldos(**Use**rId, Nombre, Trabajo, Salario)

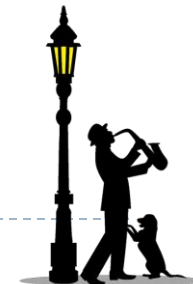


```
CREATE TABLE Sueldos (  
  UserID INT,  
  Nombre VARCHAR(100),  
  Trabajo VARCHAR(100),  
  Salario INT);
```

No distingue entre mayúsculas y minúsculas, pero es útil para facilitar la lectura.



Key



Key

Una clave es uno o más atributos que identifican de forma exclusiva una fila

Definitivamente no es
una clave

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Key



Key

Una clave es uno o más atributos que identifican de forma exclusiva una fila

Buen candidato para
una clave

Definitivamente no es
una clave

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Key



Key

Una clave es uno o más atributos que identifican de forma exclusiva una fila

Buen candidato para
una clave

Definitivamente no es
una clave

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Key



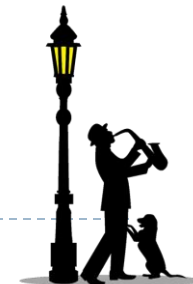
Key

Una clave es uno o más atributos que identifican de forma exclusiva una fila

Este es un buen candidato
para una clave?

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Key



Key

Una clave es uno o más atributos que identifican de forma exclusiva una fila

Este es un buen candidato
para una clave?

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |
| 913 | Pedro | Contador | 600000 |

Key



Key

Una clave es uno o más atributos que identifican de forma exclusiva una fila

Los datos provienen del mundo real
por lo que los modelos deben reflejarlo

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |
| 913 | Pedro | Contador | 600000 |

Key



Identificador
Único

```
CREATE TABLE Sueños (  
  UserID INT,  
  Nombre VARCHAR(100),  
  Trabajo VARCHAR(100),  
  Salario INT);
```

Sueños(UserId, Nombre, Trabajo, Salario)

Key



Identificador
Único

```
CREATE TABLE Sueños (  
  UserID INT PRIMARY KEY,  
  Nombre VARCHAR(100),  
  Trabajo VARCHAR(100),  
  Salario INT);
```

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |
| 123 | María | Profesora | 1200000 |

Sueños(UserId, Nombre, Trabajo, Salario)

Key



Identificador
Único

```
CREATE TABLE Sueños (  
  UserID INT,  
  Nombre VARCHAR(100),  
  Trabajo VARCHAR(100),  
  Salario INT);
```

Sueños(UserId, Nombre, Trabajo, Salario)

Key

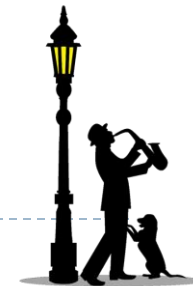


Identificador
Único

```
CREATE TABLE Sueldos (  
  { UserID INT,  
    Nombre VARCHAR(100),  
    Trabajo VARCHAR(100),  
    Salario INT,  
    PRIMARY KEY (UserID,  
    Nombre));
```

Sueldos(UserId, Nombre, Trabajo, Salario)

Foreign Keys



- Las bases de datos pueden contener múltiples tablas
- ¿Cómo capturamos las relaciones entre tablas?

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

Foreign Keys



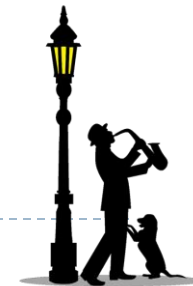
- Las bases de datos pueden contener múltiples tablas
- ¿Cómo capturamos las relaciones entre tablas?

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

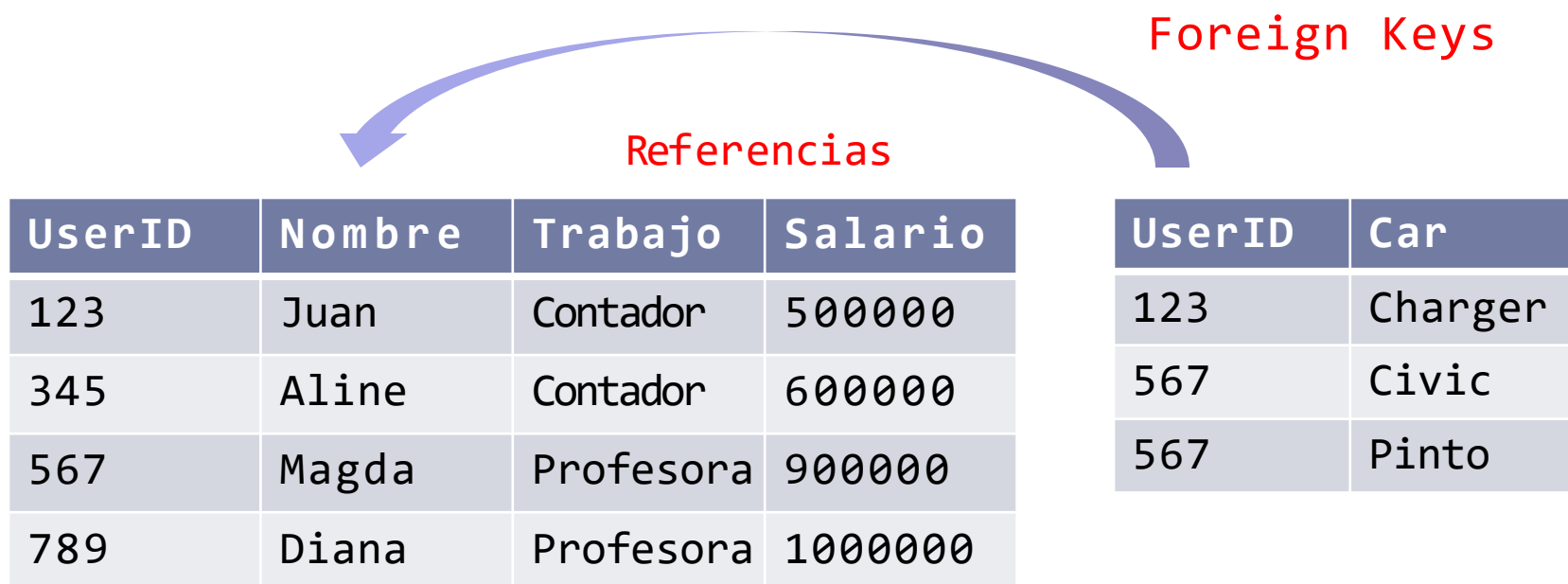
Foreign Keys

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

Foreign Keys



- Las bases de datos pueden contener múltiples tablas
- ¿Cómo capturamos las relaciones entre tablas?

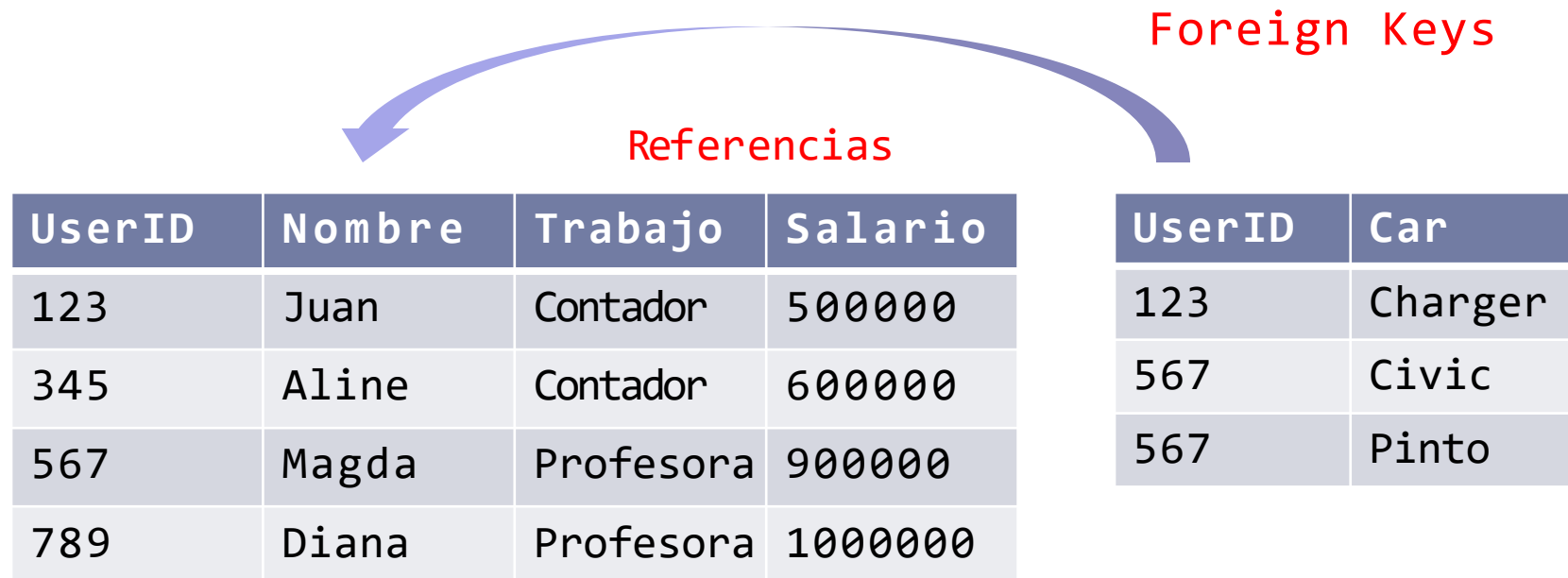


Foreign Keys

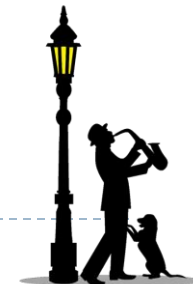


Foreign Key

Una clave foránea es uno o más atributos que identifican de forma exclusiva una fila en otra tabla



Foreign Keys



Foreign Key

Una clave foránea es uno o más atributos que identifican de forma exclusiva una fila en otra tabla

¿Esto es válido?

Referencias

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

NO

Foreign Keys



```
CREATE TABLE Sueños (  
  UserID INT PRIMARY KEY,  
  Nombre VARCHAR(100),  
  Trabajo VARCHAR(100),  
  Salario INT);
```

Sueños(UserId, Nombre, Trabajo, Salario)

```
CREATE TABLE Registros(  
  UserID INT,  
  Car VARCHAR(100));
```

Registros(UserId, Car)



Foreign Keys



```
CREATE TABLE Sueños (  
  UserID INT PRIMARY KEY,  
  Nombre VARCHAR(100),  
  Trabajo VARCHAR(100),  
  Salario INT);
```

Sueños(UserId, Nombre, Trabajo, Salario)

```
CREATE TABLE Registros(  
  UserID INT REFERENCES,  
  Car VARCHAR(100));
```

Registros(UserId, Car)



Joins



- Las claves foráneas pueden describir una relación entre tablas
- Los **Joins** pueden realizar combinaciones de datos

Inner Joins

- Pan y mantequilla de consultas SQL
- "Inner join" a menudo es intercambiable con solo "join"



Semántica de loops anidados



Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S JOIN Registros AS R
ON S.UserID = R.UserID;
```

¿Cómo obtenemos
algorítmicamente
nuestros
resultados?

| Nombre | Car |
|--------|---------|
| Juan | Charger |
| Magda | Civic |
| Magda | Pinto |

Semántica de loops anidados



Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Registros


| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S JOIN Registros AS R
ON S.UserID = R.UserID;
```

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

Semántica de loops anidados


Sueldos



| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



| Nombre | Car |
|--------|-----|
|--------|-----|

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```




Semántica de loops anidados

Sueldos



| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Registros



| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

| Nombre | Car |
|--------|---------|
| Juan | Charger |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

Semántica de loops anidados


Sueldos



| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |




| Nombre | Car |
|--------|---------|
| Juan | Charger |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

Semántica de loops anidados


Sueldos



| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



| Nombre | Car |
|--------|---------|
| Juan | Charger |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```


Semántica de loops anidados

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



| Nombre | Car |
|--------|---------|
| Juan | Charger |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

Semántica de loops anidados


Sueldos



| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



| Nombre | Car |
|--------|---------|
| Juan | Charger |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

Semántica de loops anidados

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



| Nombre | Car |
|--------|---------|
| Juan | Charger |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

Semántica de loops anidados

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



| Nombre | Car |
|--------|---------|
| Juan | Charger |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

Semántica de loops anidados

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



| Nombre | Car |
|--------|---------|
| Juan | Charger |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

Semántica de loops anidados

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



| Nombre | Car |
|--------|---------|
| Juan | Charger |
| Magda | Civic |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

Semántica de loops anidados

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



| Nombre | Car |
|--------|---------|
| Juan | Charger |
| Magda | Civic |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

Semántica de loops anidados

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



| Nombre | Car |
|--------|---------|
| Juan | Charger |
| Magda | Civic |
| Magda | Pinto |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```


Semántica de loops anidados

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

| Nombre | Car |
|--------|---------|
| Juan | Charger |
| Magda | Civic |
| Magda | Pinto |

Semántica de loops anidados

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

| Nombre | Car |
|--------|---------|
| Juan | Charger |
| Magda | Civic |
| Magda | Pinto |

Semántica de loops anidados


Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

| Nombre | Car |
|--------|---------|
| Juan | Charger |
| Magda | Civic |
| Magda | Pinto |

Semántica de loops anidados

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |



Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



| Nombre | Car |
|--------|---------|
| Juan | Charger |
| Magda | Civic |
| Magda | Pinto |

```
for each row1 in Sueldos:
  for each row2 in Registros:
    if (row1.UserID = row2.UserID):
      output (row1.Nombre, row2.Car)
```

Inner Joins

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

Explícito

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S JOIN Registros AS R
ON S.UserID = R.UserID;
```

Implícito

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S, Registros AS R
WHERE S.UserID = R.UserID;
```

Outer Joins

- Ahora queremos incluir a todas las tuplas, incluso si no coinciden.

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S LEFT OUTER JOIN Registros AS R
ON S.UserID = R.UserID;
```

Outer Joins

Sueldos

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

Registros

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

| Nombre | Car |
|--------|---------|
| Juan | Charger |
| Aline | NULL |
| Magda | Civic |
| Magda | Pinto |
| Diana | NULL |

NULL es un marcador de posición de valor. Dependiendo del contexto, puede significar desconocido, no aplicable, etc.

Outer Joins

- LEFT OUTER JOIN
 - Todas las filas de la tabla izquierda se conservan.
- RIGHT OUTER JOIN
 - Todas las filas de la tabla derecha se conservan.
- FULL OUTER JOIN
 - Todas las filas se conservan

Self Join

- Encuentra a todas las personas que conducen un Honda Civic



Self Join

- Encuentra a todas las personas que conducen un Honda Civic y un Ford Pinto



Self Join

- Encuentra a todas las personas que conducen un Honda Civic **y un Ford Pinto**

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S, Registros AS R
WHERE S.UserID = R.UserID AND
      R.Car = 'Civic';
```

Self Join

- Encuentra a todas las personas que conducen un Honda Civic y un Ford Pinto

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S, Registros AS R
WHERE S.UserID = R.UserID AND
        R.Car = 'Civic' AND
        R.Car = 'Pinto';
```

Funcionará ?

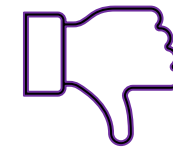


Self Join

- Encuentra a todas las personas que conducen un Honda Civic y un Ford Pinto

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |



```
SELECT S.Nombre, R.Car
FROM Sueldos AS S, Registros AS R
WHERE S.UserID = R.UserID AND
        R.Car = 'Civic' AND
        R.Car = 'Pinto';
```

Funcionará ?

No, devuelve el
conjunto vacío!

Self Join

- Encuentra a todas las personas que conducen un Honda Civic y un Ford Pinto

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |
| 789 | Pinto |

```
SELECT S.Nombre, R.Car
FROM Sueldos AS S, Registros AS R
WHERE S.UserID = R.UserID AND
      R.Car = 'Civic' AND
      R.Car = 'Pinto';
```

Discuta con su grupo
cómo resolvería esto.

Self Join

- Encuentra a todas las personas que conducen un Honda Civic y un Ford Pinto

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

```
SELECT S.Nombre, R1.Car, R2.Car
FROM Sueldos AS S, Registros AS R1, Registros AS R2
WHERE S.UserID = R1.UserID AND
      S.UserID = R2.UserID AND
      R1.Car = 'Civic' AND
      R2.Car = 'Pinto';
```

Self Join

- Encuentra a todas las personas que conducen un Honda Civic y un Ford Pinto

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

| UserID | Car |
|--------|---------|
| 123 | Charger |
| 567 | Civic |
| 567 | Pinto |

```
SELECT S.Nombre, R1.Car, R2.Car
FROM Sueldos AS S, Registros AS R1, Registros AS R2
WHERE S.UserID = R1.UserID AND
      S.UserID = R2.UserID AND
      R1.Car = 'Civic' AND
      R2.Car = 'Pinto';
```

Todos los pares de autos que una persona puede conducir

Un poco más de SQL

| UserID | Nombre | Trabajo | Salario |
|--------|--------|-----------|---------|
| 123 | Juan | Contador | 500000 |
| 345 | Aline | Contador | 600000 |
| 567 | Magda | Profesora | 900000 |
| 789 | Diana | Profesora | 1000000 |

- **ORDER BY**

- Ordena las tuplas resultantes por los atributos especificados (Ascendente predeterminado)

```
SELECT P.Nombre, P.UserID
FROM Sueldos AS S
WHERE S.Trabajo = 'Contador'
ORDER BY S.Salario, S.Nombre;
```

| Nombre | UserID |
|--------|--------|
| Juan | 123 |
| Aline | 345 |

- **DISTINCT**

- Deduplica los resultados de las tuplas

```
SELECT DISTINCT S.Trabajo
FROM Sueldos AS S
WHERE S.Salario > 450000;
```

| Trabajo |
|-----------|
| Contador |
| Profesora |