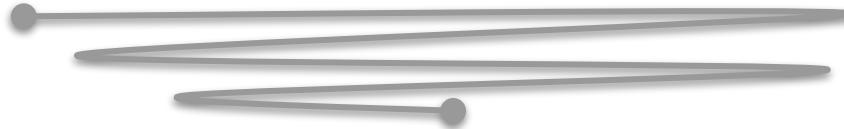


# *Laboratorio de Datos*



*AR - SQL*

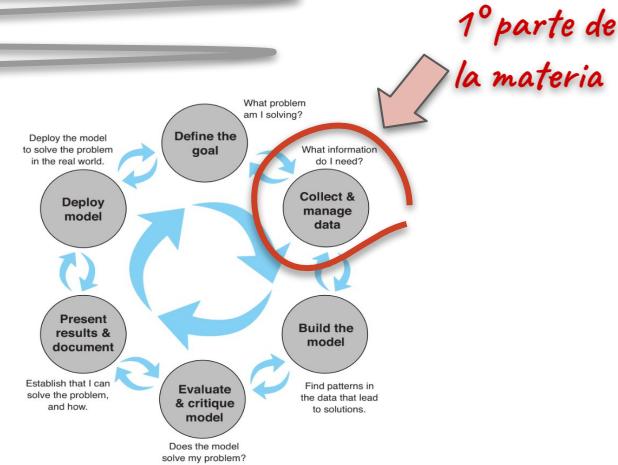


# Recorrido de la materia (hasta ahora)

- ✓ Lenguaje de programación para trabajar en nuestros proyectos



- ✓ Etapas de un proyecto de Ciencias de Datos



- ✓ Modelado de Datos



- ✓ Representación de los Datos

Materia

Código	Nombre
1	Laboratorio de Datos
2	Análisis II
3	Álgebra Lineal

Unidad

Código Materia	Título	Descripción
1	Administración de datos	Obtención y Manejo de los datos
1	Modelos Explicativos	Construcción de modelos explicativos
1	Modelos Predictivos	Construcción de modelos predictivos
2	Integrales sobre curvas y	Integrales en múltiples variables
2	Ecuaciones Diferenciales	Solución de ecuaciones diferenciales

## Tarea Procesamiento de Datos - Preguntas



Responder las siguientes preguntas (escribir las respuestas a modo de comentario dentro del script)

1. ¿Cómo afectó a la programación de la función cuando cambiaron levemente la matriz de empleado?
  - a. En el caso en que le agregaron más filas
  - b. En el caso en que le alteraron el orden de las columnas
2. ¿Y cuando a empleado le cambiaron la forma de representar las matrices (de lista de filas a lista de columnas)?
3. ¿Cuál es la ventaja, desde el punto de vista del usuario de la función, disponer de ella y no escribir directamente el código de la consulta dentro de su programa?

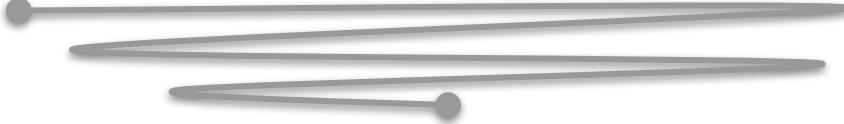
## *Tarea Procesamiento de Datos - Cierre*



### *I. Modificaciones en estructura de datos*

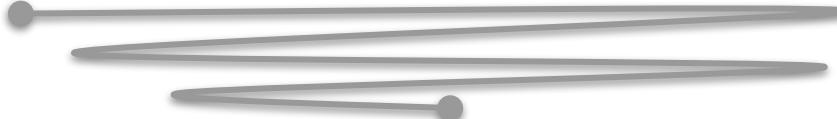
*-> Cambios en programación de las consultas*

## Tarea Procesamiento de Datos - Reflexión



- ✓ Necesitamos “algo” que nos permita acceder a los datos independientemente de cómo se encuentran almacenados físicamente (matriz como lista de filas, matriz como lista de columnas, orden en que se encuentran almacenadas las columnas, etc.)
- ✓ Nos vendría bien un lenguaje que nos permita expresar el acceso a los datos, independientemente de su implementación
- ✓ El modelo relacional tiene un lenguaje desarrollado para ello llamado SQL y se basa en el Álgebra Relacional (AR)

# Álgebra Relacional



## Empecemos con ...

## Álgebra Relacional

Verano - 2024

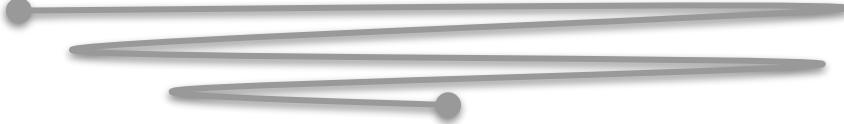


## *SQL - Introducción*

Pasemos a ...



## SQL - Introducción



- ✓ SQL = Structured Query Language
- ✓ Lenguaje universalmente más usado para bases de datos relacionales
- ✓ Lenguaje declarativo de alto nivel
- ✓ Desarrollado por IBM (1974-1977)
- ✓ Se convirtió en un standard definido por:
  - ANSI (American National Standards Institute)
  - ISO (International Standards Organization)
- ✓ El estándar actual es el SQL:1999 (aunque muchas DBMS no lo implementaron por completo aún. Existen revisiones del 2003 y 2006)

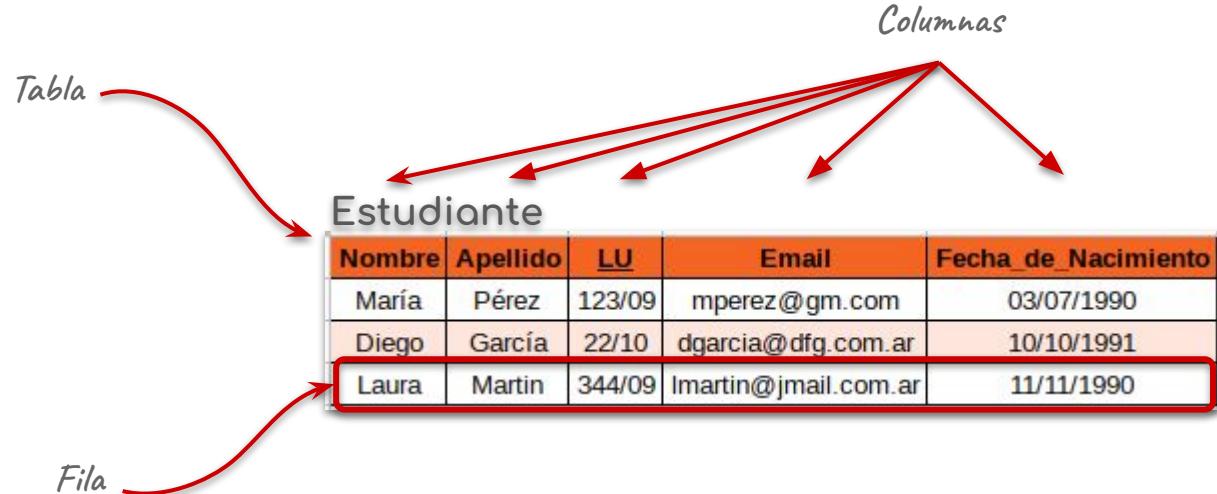
# SQL - Introducción

- ✓ Sentencias del SQL se dividen en:
    - Sentencias DDL** (Data Definition Language). Permiten crear/modificar/borrar estructuras de datos (Ej. Tablas). Acá también es donde se definen las restricciones (claves, foreign keys, etc.).  
Ej. de comandos: CREATE/ALTER/DROP TABLE.
    - Sentencias DML** (Data Manipulation Language). Posibilitan manipular datos.  
Basado en Álgebra Relacional.  
Ej. de comandos: SELECT/INSERT/UPDATE/DELETE. Si no se cumple la restricción avisa.
  - ✓ También provee sentencias para:
    - Definir permisos (control de acceso de usuarios)
    - Manejo de transacciones
    - Otros
- Nos vamos a enfocar en DML.  
¡En particular en SELECT!

# SQL - Introducción

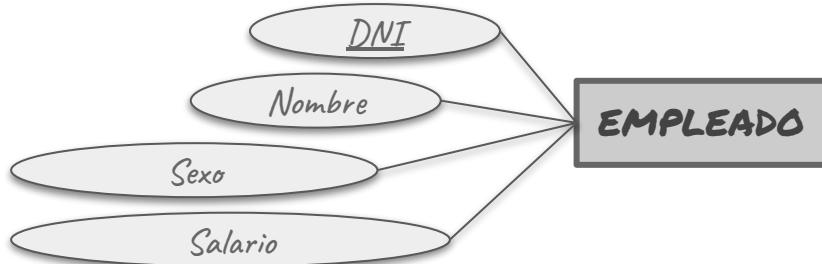
## ✓ Términos (Modelo Relacional -> Álgebra Relacional)

- Tabla  Relación
- Fila  Tupla
- Columna  Atributo



# AR <-> SQL

1. Contamos con el siguiente DER

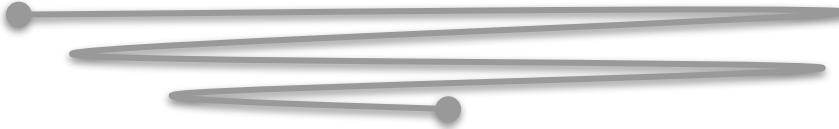


2. Lo mapeamos a la siguiente estructura del Modelo Relacional: **EMPLEADO(DNI, Nombre, Sexo, Salario)**

3. La tabla contiene los siguientes datos:

EMPLEADO			
DNI	Nombre	Sexo	Salario
20222333	Diego	M	\$20.000,00
33456234	Laura	F	\$25.000,00
45432345	Marina	F	\$10.000,00

*AR <-> SQL*

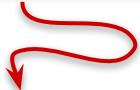


*AR - Proyección <-> SQL - SELECT*

# *AR - Proyección <-> SQL - SELECT*

Consigna: Listar DNI y Salario de EMPLEADO

EMPLEADO			
DNI	Nombre	Sexo	Salario
20222333	Diego	M	\$20.000,00
33456234	Laura	F	\$25.000,00
45432345	Marina	F	\$10.000,00



DNI	Salario
20222333	\$20.000,00
33456234	\$25.000,00
45432345	\$10.000,00

*Álgebra Relacional*

$\pi_{DNI, Salario}(EMPLEADO)$

*SQL*

```
SELECT DISTINCT DNI, Salario  
FROM empleado;
```

# SQL - SELECT <-> Python

```
1 # -*- coding: utf-8 -*-
2 """
3     Materia: Laboratorio de datos - FCEyN - UBA
4     Clase : Clase SQL. Script clase.
5     Autor : Pablo Turjanski
6     Fecha : 2023-03-07
7 """
8
9 # Importamos bibliotecas
10 import pandas as pd
11 from inline_sql import sql, sql_val
12
```

- } *Información sobre el programa*
- } *Bibliotecas.*
- Pandas. Para dataframes
  - inline\_sql: Para lenguaje sql

# SQL - SELECT <-> Python

Función para generar el dataframe empleado

```
559 # =====
560 # FUNCIONES PARA LA GENERACIÓN DE DATAFRAMES
561 #
562 def get_empleado():
563     # Genera el dataframe "empleado" que contiene las siguientes columnas
564     # (en el orden mencionado):
565     # 1. DNI
566     # 2. Nombre
567     # 3. Sexo
568     # 4. Salario
569
570     # ... Creamos el dataframe vacío (sólo con los nombres de sus columnas)
571     empleado = pd.DataFrame(columns = ['DNI', 'Nombre', 'Sexo', 'Salario'])
572     # ... Agregamos cada una de las filas al DataFrame
573     empleado = pd.concat([empleado,pd.DataFrame([
574         {'DNI' : 20222333, 'Nombre' : 'Diego', 'Sexo' : 'M', 'Salario' : 20000.0},
575         {'DNI' : 33456234, 'Nombre' : 'Laura', 'Sexo' : 'F', 'Salario' : 25000.0},
576         {'DNI' : 45432345, 'Nombre' : 'Marina', 'Sexo' : 'F', 'Salario' : 10000.0}
577     ]))
578
579     return empleado
580
```

Definición de estructura de dataframe (vacío)

} Se insertan 3 filas nuevas

Usamos la función

```
13 def main():
14
15     print()
16     print("# =====")
17     print("# Creamos los datasets que vamos a utilizar en este programa")
18     print("# =====")
19
20     # Ejercicios AR-PROJECT, SELECT, RENAME
21     empleado = get_empleado()
22     # Ejercicios AR-UNION, INTERSECTION, MINUS
```

Llamamos a la función y el resultado lo guardamos en un dataframe

# SQL - SELECT <-> Python

Vemos el contenido del  
dataframe original

La Consulta se define  
como un string

```
45 print()  
46 print("# =====")  
47 print("# Ejemplo inicial")  
48 print("# =====")  
49  
50 print(empleado)  
51  
52 consultaSQL = """  
53     SELECT DISTINCT DNI, Salario  
54     FROM empleado;  
55 """  
56  
57  
58 dataframeResultado = sql^ consultaSQL  
59  
60 print(dataframeResultado)  
61
```

Se imprime el resultado

Se ejecuta la consulta  
y se almacena en un  
dataframe

# SQL - SELECT <-> Python

```
007  
008 # =====  
009 # DEFINICIÓN DE FUNCIÓN DE IMPRESIÓN EN PANTALLA  
010 # =====  
011 # Imprime en pantalla en un formato ordenado:  
012 # 1. Consigna  
013 # 2. Cada dataframe de la lista de dataframes de entrada  
014 # 3. Query  
015 # 4. Dataframe de salida  
016 def imprimirEjercicio(consigna, listaDeDataframesDeEntrada, consultaSQL, dataframeResultadoDeConsultaSQL):  
017     print("# -----")  
018     print("# Consigna: ", consigna)  
019     print("# -----")  
020     print()  
021     for i in range(len(listaDeDataframesDeEntrada)):  
022         print("# Entrada {},".format(i),sep='')  
023         print("# -----")  
024         print(listaDeDataframesDeEntrada[i])  
025         print()  
026     print("# SQL:")  
027     print("# ----")  
028     print(consultaSQL)  
029     print()  
030     print("# Salida:")  
031     print("# -----")  
032     print(dataframeResultadoDeConsultaSQL)  
033     print()  
034     print("# -----")  
035     print("# -----")  
036     print()  
037     print()
```

Creamos una función a la que le pasamos:

1. una consigna (string)
2. una lista de dataframes de entrada (dataframe/s)
3. una consulta SQL (string)
4. el resultado de la consulta SQL (dataframe)

Y como resultado imprime en pantalla:

1. la consigna
2. cada dataframe de entrada (original)
3. la consulta SQL (string)
4. el resultado de la consulta SQL

# SQL - SELECT <-> Python

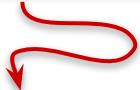
```
62
63     print()
64     print("# =====")
65     print("# Ejercicios AR-PROJECT <-> SELECT")
66     print("# =====")
67
68     consigna    = "a.- Listar DNI y Salario de empleados "
69     consultaSQL = """
70             SELECT DISTINCT DNI, Salario
71                 FROM empleado;
72             """
73
74     imprimirEjercicio(consigna, [empleado], consultaSQL, sql^consultaSQL)
```

*Uso de la función que creamos*

# *AR - Proyección <-> SQL - SELECT*

Consigna: Listar Sexo de EMPLEADO

EMPLEADO			
DNI	Nombre	Sexo	Salario
20222333	Diego	M	\$20.000,00
33456234	Laura	F	\$25.000,00
45432345	Marina	F	\$10.000,00



Sexo
M
F

*Álgebra Relacional*

$\pi_{Sexo}(EMPLEADO)$

*SQL*

```
SELECT DISTINCT Sexo  
FROM empleado;
```

# SQL - SELECT <-> Python

```
74
75      # -----
76      consigna    = "b.- Listar Sexo de empleados "
77      consultaSQL = """
78          SELECT DISTINCT Sexo
79          FROM empleado;
80      """
81      imprimirEjercicio(consigna, [empleado], consultaSQL, sql^consultaSQL)
```

# AR - Proyección <-> SQL - SELECT

Consigna: Listar Sexo de EMPLEADO

EMPLEADO			
DNI	Nombre	Sexo	Salario
20222333	Diego	M	\$20.000,00
33456234	Laura	F	\$25.000,00
45432345	Marina	F	\$10.000,00



Sexo
M
F

Álgebra Relacional

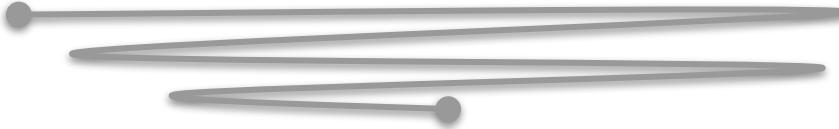
$\pi_{Sexo}(EMPLEADO)$

¿Qué sucede si removemos  
DISTINCT?

SQL

```
SELECT DISTINCT Sexo  
FROM empleado;
```

*AR <-> SQL*



*AR - Selección <-> SQL - WHERE*

# AR - Selección <-> SQL - WHERE

Consigna: Listar de EMPLEADO sólo aquellos cuyo sexo es femenino

EMPLEADO

DNI	Nombre	Sexo	Salario
20222333	Diego	M	\$20.000,00
33456234	Laura	F	\$25.000,00
45432345	Marina	F	\$10.000,00

Álgebra Relacional

$$\sigma_{Sexo=F}(EMPLEADO)$$


DNI	Nombre	Sexo	Salario
33456234	Laura	F	\$25.000,00
45432345	Marina	F	\$10.000,00

SQL

```
SELECT DISTINCT DNI, Nombre, Sexo, Salario  
FROM empleado  
WHERE Sexo='F';
```

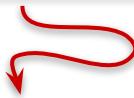
# AR - Selección <-> SQL - WHERE

Consigna: Listar de EMPLEADO aquellos cuyo sexo es femenino y su salario es mayor a \$15.000

EMPLEADO

DNI	Nombre	Sexo	Salario
20222333	Diego	M	\$20.000,00
33456234	Laura	F	\$25.000,00
45432345	Marina	F	\$10.000,00

Álgebra Relacional

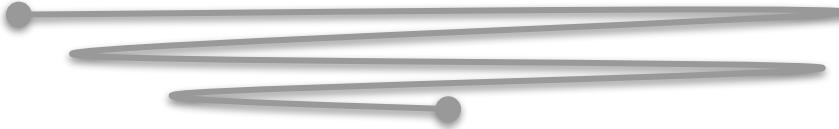
$$\sigma_{\text{Sexo} = F \text{ AND } \text{Salario} > \$15.000}(\text{EMPLEADO})$$


DNI	Nombre	Sexo	Salario
33456234	Laura	F	\$25.000,00

SQL

```
SELECT DISTINCT DNI, Nombre, Sexo, Salario  
FROM empleado  
WHERE Sexo='F' AND Salario>15000;
```

*AR <-> SQL*

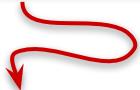


*AR - Renombre <-> SQL - AS*

# AR - Renombre $\leftrightarrow$ SQL - AS

Consigna: Listar DNI y Salario de EMPLEADO, y renombrarlos como id e Ingreso

EMPLEADO			
DNI	Nombre	Sexo	Salario
20222333	Diego	M	\$20.000,00
33456234	Laura	F	\$25.000,00
45432345	Marina	F	\$10.000,00



id	Ingreso
20222333	\$20.000,00
33456234	\$25.000,00
45432345	\$10.000,00

## Álgebra Relacional

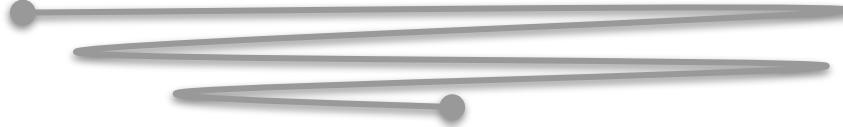
$EMPLEADO(id, Ingreso) \leftarrow \pi_{DNI, Salario}(EMPLEADO)$

T

## SQL

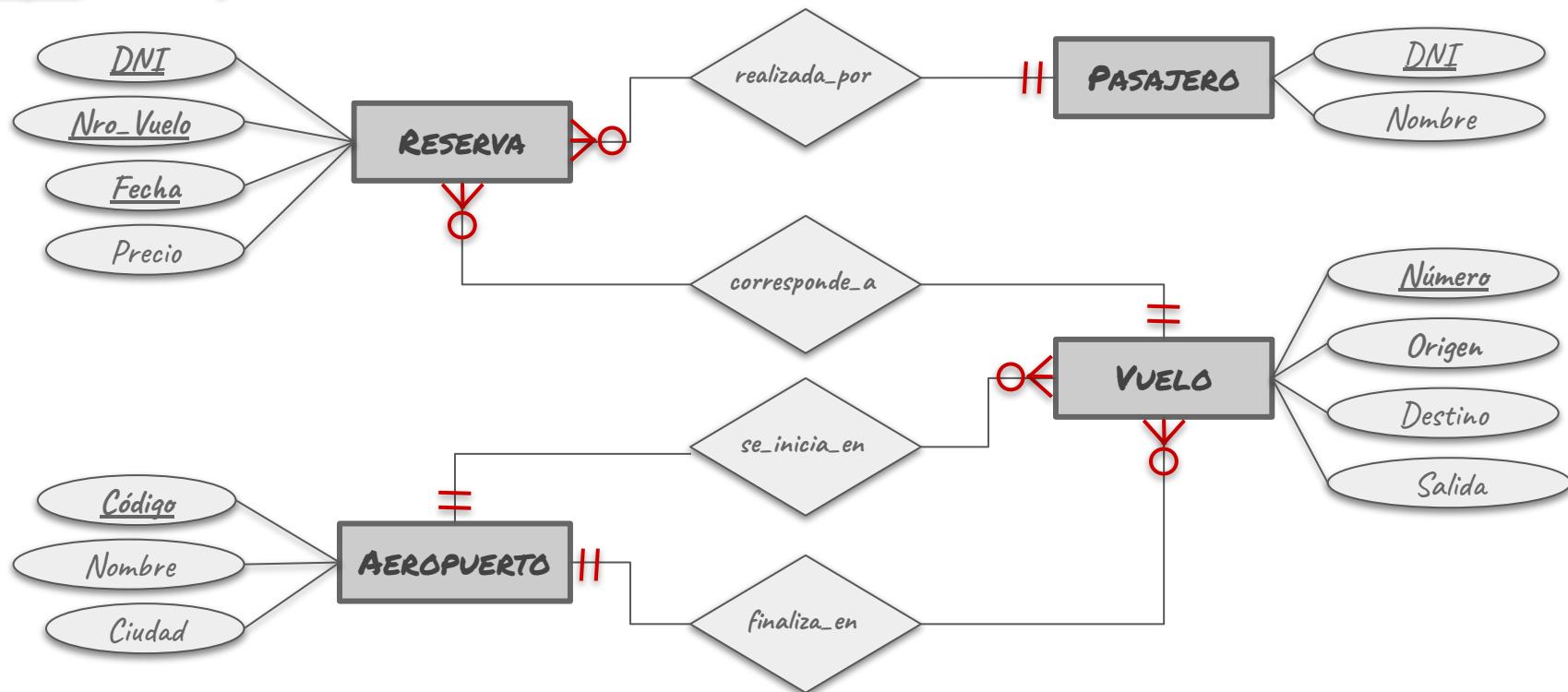
`SELECT DISTINCT DNI AS id, Salario AS Ingreso  
FROM empleado;`

# *SQL - Ejercicio 1*



# SQL - Ejercicio 1

Consigna: Sea el siguiente DER



# SQL - Ejercicio 1

VUELO

Número	Origen	Destino	Salida
345	MAD	CDG	12:30
321	MAD	ORY	19:05
165	LHR	CDG	09:55
903	CDG	LHR	14:40
447	CDG	LHR	17:00

AEROPUERTO

Código	Nombre	Ciudad
MAD	Barajas	Madrid
LGW	Gatwick	Londres
LHR	Heathrow	Londres
ORY	Orly	París
CDG	Charles de Gaulle	París

PASAJERO

DNI	Nombre
123	María
456	Pedro
789	Isabel

RESERVA

DNI	Nro_Vuelo	Fecha	Precio
789	165	07-01-11	210
123	345	20-12-10	170
789	321	15-12-10	250
456	345	03-11-10	190

1 Retornar Código y Nombre de los aeropuertos de Londres

2 ¿Qué retorna

```
SELECT DISTINCT Ciudad AS City
FROM aeropuerto
WHERE Codigo='ORY' OR Codigo='CDG' ;
```

3 Obtener los números de vuelo que van desde CDG hacia LHR

4 Obtener los números de vuelo que van desde CDG hacia LHR o viceversa

5 Devolver las fechas de reservas cuyos precios son mayores a \$200

# AR <-> SQL

1. Contamos con el siguiente DER, donde cada entidad representa a los alumnos que cursan cada una de esas materias.



2. Lo mapeamos a la siguiente estructura del Modelo Relacional:

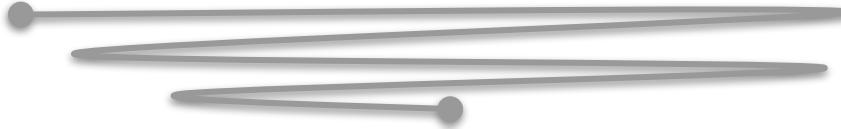
$\text{ALUMNOS\_BD}(\underline{\text{id}}, \text{Nombre})$

$\text{ALUMNOS\_TLENG}(\underline{\text{id}}, \text{Nombre})$

3. Las tablas contienen los siguientes datos:

ALUMNOS_BD		ALUMNOS_TLENG	
<b>id</b>	<b>Nombre</b>	<b>id</b>	<b>Nombre</b>
1	Diego	2	Laura
2	Laura	4	Alejandro
3	Marina		

*AR <-> SQL*



*AR - Unión <-> SQL - UNION*

# AR - Unión <-> SQL - UNION

Consigna: Listar a los alumnos que cursan BDs o TLENG

ALUMNOS\_BD

id	Nombre
1	Diego
2	Laura
3	Marina

ALUMNOS\_TLENG

id	Nombre
2	Laura
4	Alejandro

Álgebra Relacional

$ALUMNOS\_BD \cup ALUMNOS\_TLENG$

id	Nombre
1	Diego
2	Laura
3	Marina
4	Alejandro

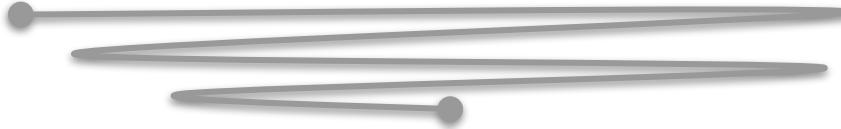
¿Qué sucede si cambiamos  
por UNION ALL?

Todas las columnas  
de la tabla

SQL

```
SELECT DISTINCT *
FROM alumnos_BD
UNION
SELECT DISTINCT *
FROM alumnos_TLeng;
```

*AR <-> SQL*



*AR - Intersección <-> SQL - INTERSECT*

# AR - Intersección <-> SQL - INTERSECT

Consigna: Listar a los alumnos que cursan simultáneamente BDs y TLENG

ALUMNOS\_BD

id	Nombre
1	Diego
2	Laura
3	Marina

ALUMNOS\_TLENG

id	Nombre
2	Laura
4	Alejandro

Álgebra Relacional

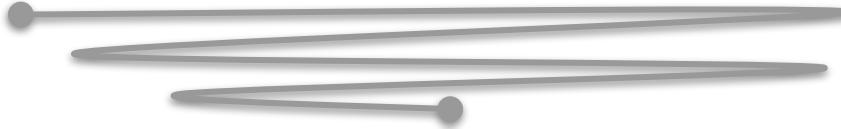
$ALUMNOS\_BD \cap ALUMNOS\_TLENG$

id	Nombre
2	Laura

SQL

```
SELECT DISTINCT *
FROM alumnos_BD
INTERSECT
SELECT DISTINCT *
FROM alumnos_TLeng;
```

*AR <-> SQL*



*AR - Minus <-> SQL - EXCEPT*

# AR - Minus <-> SQL - EXCEPT

Consigna: Listar a los alumnos que cursan BDs y no cursan TLENG

ALUMNOS\_BD

id	Nombre
1	Diego
2	Laura
3	Marina

ALUMNOS\_TLENG

id	Nombre
2	Laura
4	Alejandro

Álgebra Relacional

ALUMNOS\_BD – ALUMNOS\_TLENG

id	Nombre
1	Diego
3	Marina

SQL

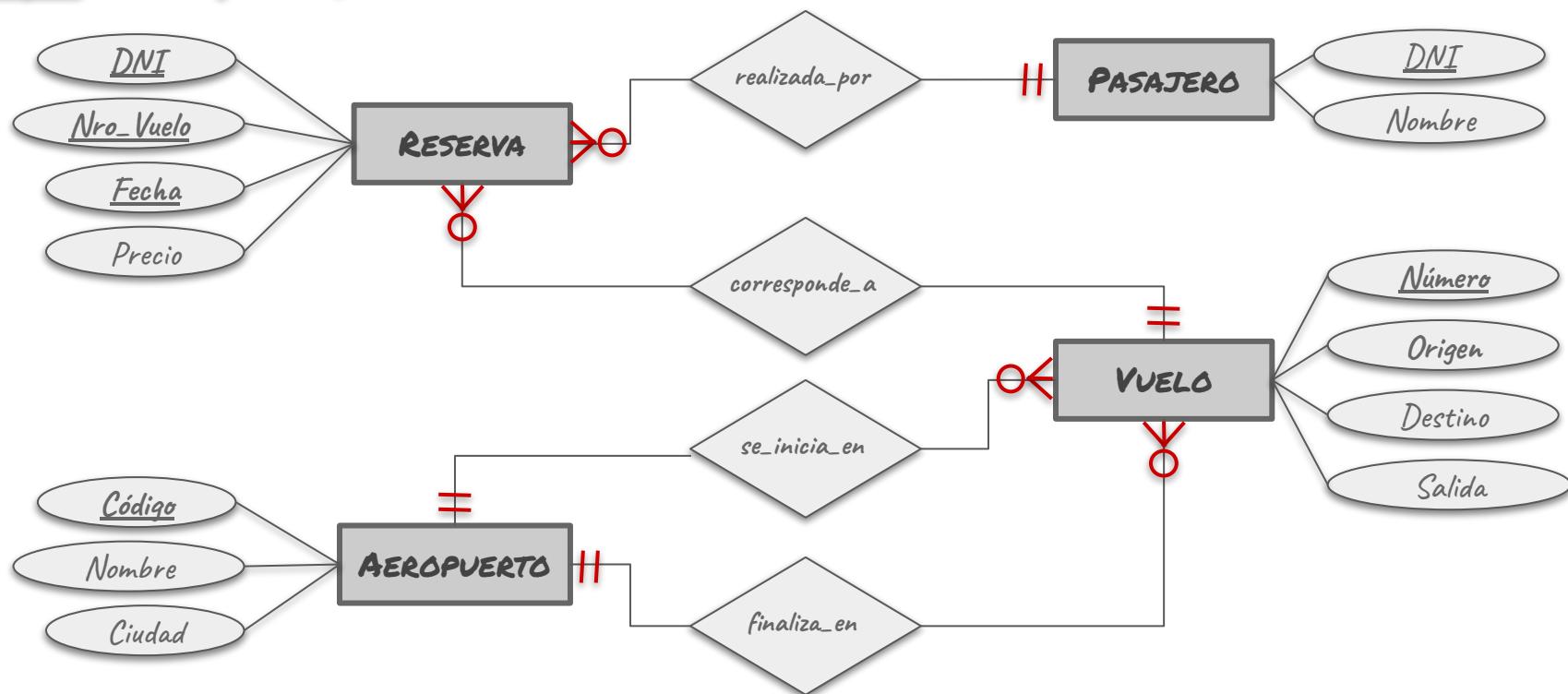
```
SELECT DISTINCT *
FROM alumnos_BD
EXCEPT
SELECT DISTINCT *
FROM alumnos_TLeng;
```

# *SQL - Ejercicio 2*



# SQL - Ejercicio 2

Consigna: Sea el siguiente DER



# SQL - Ejercicio 2

VUELO

Número	Origen	Destino	Salida
345	MAD	CDG	12:30
321	MAD	ORY	19:05
165	LHR	CDG	09:55
903	CDG	LHR	14:40
447	CDG	LHR	17:00

AEROPUERTO

Código	Nombre	Ciudad
MAD	Barajas	Madrid
LGW	Gatwick	Londres
LHR	Heathrow	Londres
ORY	Orly	París
CDG	Charles de Gaulle	París

PASAJERO

DNI	Nombre
123	María
456	Pedro
789	Isabel

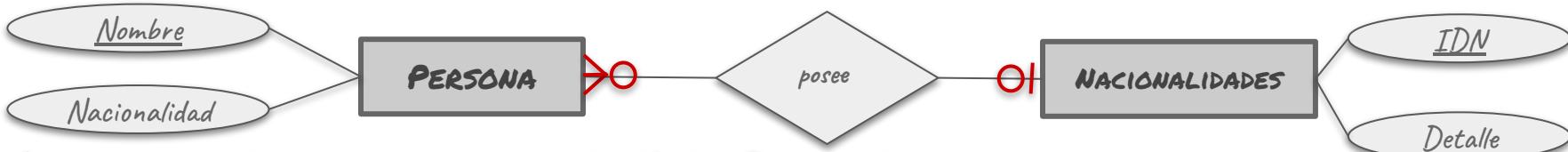
RESERVA

DNI	Nro_Vuelo	Fecha	Precio
789	165	07-01-11	210
123	345	20-12-10	170
789	321	15-12-10	250
456	345	03-11-10	190

- 1 Devolver los número de vuelo que tienen reservas generadas (utilizar  $\cap$ )
- 2 Devolver los número de vuelo que aún no tienen reservas
- 3 Retornar los códigos de aeropuerto de los que parten o arriban los vuelos

# AR -> SQL

1. Contamos con el siguiente DER, donde se representa las nacionalidades de un grupo de personas.



2. Lo mapeamos a la siguiente estructura del Modelo Relacional:

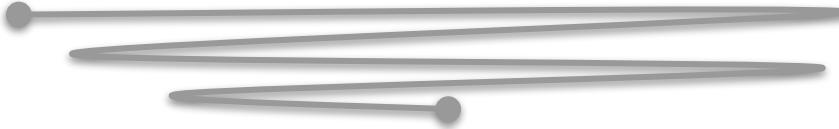
**PERSONA**(Nombre, Nacionalidad)

**NACIONALIDADES**(IDN, Detalle)

3. A modo de ejemplo, las tablas pueden contener:

PERSONA		NACIONALIDADES	
Nombre	Nacionalidad	IDN	Detalle
Diego	AR	AR	Argentina
Laura	BR	BR	Brasilera
Marina	AR	CH	Chilena

*AR <-> SQL*



*AR - Producto Cartesiano <-> SQL - CROSS JOIN*

# AR - Producto Cartesiano <-> SQL - CROSS JOIN

Consigna: Listar el producto cartesiano entre las tablas PERSONA y NACIONALIDADES

PERSONA	
Nombre	Nacionalidad
Diego	AR
Laura	BR
Marina	AR

NACIONALIDADES	
IDN	Detalle
AR	Argentina
BR	Brasilera
CH	Chilena

Nombre	Nacionalidad	IDN	Detalle
Diego	AR	AR	Argentina
Diego	AR	BR	Brasilera
Diego	AR	CH	Chilena
Laura	BR	AR	Argentina
Laura	BR	BR	Brasilera
Laura	BR	CH	Chilena
Marina	AR	AR	Argentina
Marina	AR	BR	Brasilera
Marina	AR	CH	Chilena

Álgebra Relacional

PERSONA X NACIONALIDADES

SQL

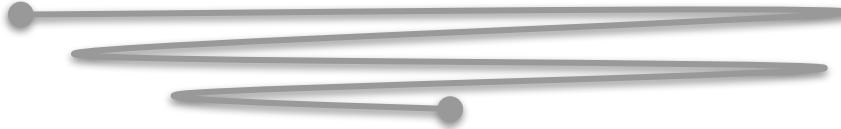
```
SELECT DISTINCT *
FROM PERSONA,NACIONALIDADES;
```

Otra manera ...

SQL

```
SELECT DISTINCT *
FROM PERSONA
CROSS JOIN NACIONALIDADES;
```

*AR <-> SQL*



*AR - Inner Join <-> SQL - INNER JOIN*

*Sin embargo, la cláusula ON permite que el RDBM pueda optimizar la operación, ya que el JOIN es una operación computacionalmente muy costosa*

## *AR - Inner Join <-> SQL - INNER JOIN*

Consigna: Vincular las tablas PERSONA y NACIONALIDADES a través de un INNER JOIN

PERSONA		NACIONALIDADES	
Nombre	Nacionalidad	IDN	Detalle
Diego	BR	AR	Argentina
Laura	NULL	BR	Brasilera
Marina	AR	CH	Chilena
Santiago	UY		



Nombre	Nacionalidad	IDN	Detalle
Diego	BR	BR	Brasilera
Marina	AR	AR	Argentina

### Álgebra Relacional

$\text{PERSONA} \bowtie_{\text{Nacionalidad}=\text{IDN}} \text{NACIONALIDADES}$

### SQL

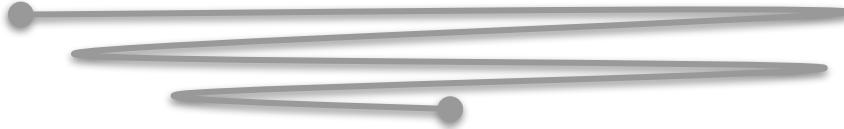
```
SELECT DISTINCT *
FROM PERSONA, NACIONALIDADES
WHERE Nacionalidad=IDN;
```

### SQL

```
SELECT DISTINCT *
FROM PERSONA
INNER JOIN NACIONALIDADES
ON Nacionalidad=IDN;
```

*Otra manera ...*

*AR <-> SQL*



*AR - Left Outer Join <-> SQL - LEFT OUTER JOIN*

# AR - Left Outer Join <-> SQL - LEFT OUTER JOIN

Consigna: Vincular las tablas PERSONA y NACIONALIDADES a través de un LEFT OUTER JOIN

PERSONA		NACIONALIDADES	
Nombre	Nacionalidad	IDN	Detalle
Diego	BR	AR	Argentina
Laura	NULL	BR	Brasilera
Marina	AR	CH	Chilena
Santiago	UY		



Nombre	Nacionalidad	IDN	Detalle
Diego	BR	BR	Brasilera
Laura	NULL	NULL	NULL
Marina	AR	AR	Argentina
Santiago	UY	NULL	NULL

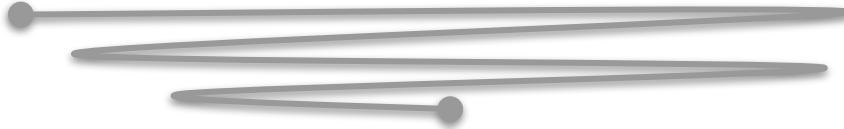
## Álgebra Relacional

PERSONA  $\bowtie_{Nacionalidad=IDN}$  NACIONALIDADES

## SQL

```
SELECT DISTINCT *
FROM PERSONA
LEFT OUTER JOIN NACIONALIDADES
ON Nacionalidad=IDN;
```

*SQL*



*¿Mismos nombres en diferentes tablas?*

# SQL - ¿Mismos nombres?

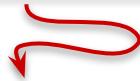
Consigna: Vincular las tablas Se\_inscribe\_en y Materia. Mostrar sólo LU y Nombre de materia

Se\_inscribe\_en

LU	Codigo_materia
123/09	1
22/10	1
22/10	2
344/09	1

Materia

Codigo_materia	Nombre
1	Laboratorio de Datos
2	Análisis II
3	Probabilidad



LU	Nombre
123/09	Laboratorio de Datos
22/10	Laboratorio de Datos
22/10	Análisis II
344/09	Laboratorio de Datos

SQL

```
SELECT DISTINCT LU, Nombre  
FROM Se_inscribe_en  
INNER JOIN Materia  
ON ...;
```

Qué iría acá ...

# SQL - ¿Mismos nombres?

Consigna: Vincular las tablas Se\_inscribe\_en y Materia. Mostrar sólo LU y Nombre de materia

Se\_inscribe\_en

LU	Codigo_materia
123/09	1
22/10	1
22/10	2
344/09	1

Materia

Codigo_materia	Nombre
1	Laboratorio de Datos
2	Análisis II
3	Probabilidad



LU	Nombre
123/09	Laboratorio de Datos
22/10	Laboratorio de Datos
22/10	Análisis II
344/09	Laboratorio de Datos

SQL

```
SELECT DISTINCT LU, Nombre  
FROM Se_inscribe_en  
INNER JOIN Materia  
ON Codigo_materia=Codigo_materia;
```

¿De cuál de las 2 tablas?

# SQL - ¿Mismos nombres?

Consigna: Vincular las tablas Se\_inscribe\_en y Materia. Mostrar sólo LU y Nombre de materia

Se\_inscribe\_en

LU	Codigo_materia
123/09	1
22/10	1
22/10	2
344/09	1

Materia

Codigo_materia	Nombre
1	Laboratorio de Datos
2	Análisis II
3	Probabilidad

SQL

```
SELECT DISTINCT LU, Nombre  
FROM Se_inscribe_en  
INNER JOIN Materia  
ON Se_inscribe_en.Codigo_materia=  
Materia.Codigo_materia;
```

LU	Nombre
123/09	Laboratorio de Datos
22/10	Laboratorio de Datos
22/10	Análisis II
344/09	Laboratorio de Datos

Así se indica de qué tabla

# SQL - ¿Mismos nombres?

Consigna: Vincular las tablas Se\_inscribe\_en y Materia. Mostrar sólo LU y Nombre de materia

Se\_inscribe\_en

LU	Codigo_materia
123/09	1
22/10	1
22/10	2
344/09	1

Materia

Codigo_materia	Nombre
1	Laboratorio de Datos
2	Análisis II
3	Probabilidad



LU	Nombre
123/09	Laboratorio de Datos
22/10	Laboratorio de Datos
22/10	Análisis II
344/09	Laboratorio de Datos

SQL

```
SELECT DISTINCT i.LU, m.Nombre  
FROM Se_inscribe_en AS i  
INNER JOIN Materia AS m  
ON i.Codigo_materia=  
m.Codigo_materia;
```



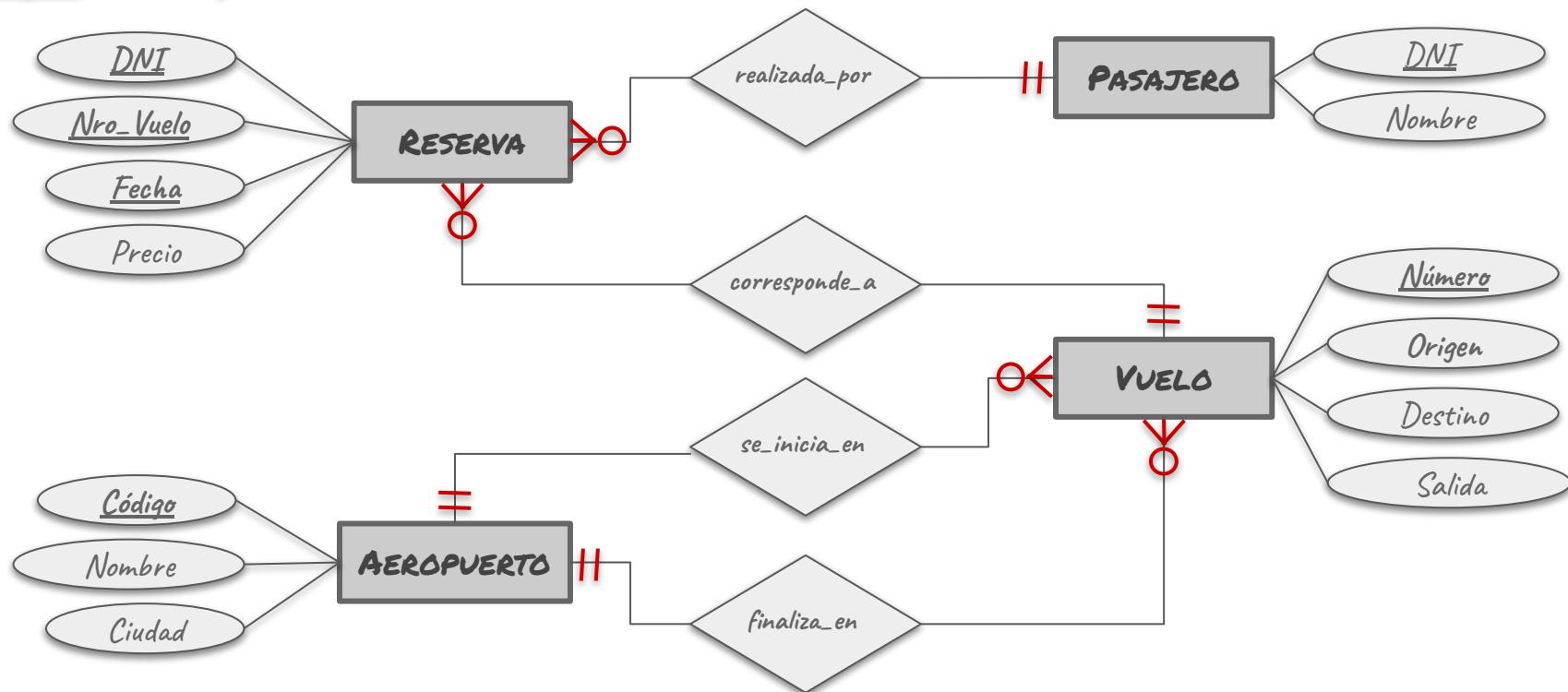
Y se puede acortar  
renombrando a las tablas

# *SQL - Ejercicio 3*



# SQL - Ejercicio 3

Consigna: Sea el siguiente DER



# SQL - Ejercicio 3

VUELO

Número	Origen	Destino	Salida
345	MAD	CDG	12:30
321	MAD	ORY	19:05
165	LHR	CDG	09:55
903	CDG	LHR	14:40
447	CDG	LHR	17:00

AEROPUERTO

Código	Nombre	Ciudad
MAD	Barajas	Madrid
LGW	Gatwick	Londres
LHR	Heathrow	Londres
ORY	Orly	París
CDG	Charles de Gaulle	París

PASAJERO

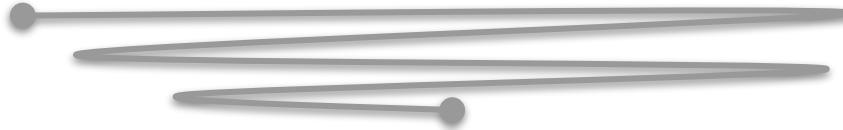
DNI	Nombre
123	María
456	Pedro
789	Isabel

RESERVA

DNI	Nro_Vuelo	Fecha	Precio
789	165	07-01-11	210
123	345	20-12-10	170
789	321	15-12-10	250
456	345	03-11-10	190

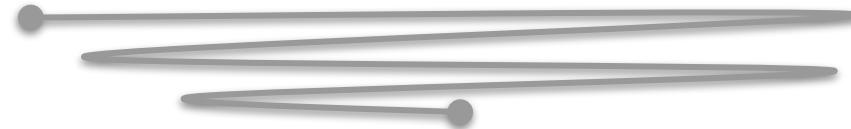
- 1 Devolver el nombre de la ciudad de partida del vuelo número 165
- 2 Retornar el nombre de las personas que realizaron reservas a un valor menor a \$200
- 3 Obtener Nombre, Fecha y Destino del Viaje de todos los pasajeros que vuelan desde Madrid

*SQL*



*Join de varias tablas en simultáneo*

# SQL - Join de varias tablas en simultáneo



VUELO

Número	Origen	Destino	Salida
345	MAD	CDG	12:30
321	MAD	ORY	19:05
165	LHR	CDG	09:55
903	CDG	LHR	14:40
447	CDG	LHR	17:00

AEROPUERTO

Código	Nombre	Ciudad
MAD	Barajas	Madrid
LGW	Gatwick	Londres
LHR	Heathrow	Londres
ORY	Orly	París
CDG	Charles de Gaulle	París

PASAJERO

DNI	Nombre
123	María
456	Pedro
789	Isabel

RESERVA

DNI	Nro_Vuelo	Fecha	Precio
789	165	07-01-11	210
123	345	20-12-10	170
789	321	15-12-10	250
456	345	03-11-10	190

Consigna: Vincular las tablas Reserva, Pasajero y Vuelo en un solo SELECT. Mostrar sólo Fecha de reserva, hora de salida del vuelo y nombre de pasajero.

SQL

```
SELECT DISTINCT ...  
...;
```



## SQL - Join de varias tablas en simultáneo

VUELO

Número	Origen	Destino	Salida
345	MAD	CDG	12:30
321	MAD	ORY	19:05
165	LHR	CDG	09:55
903	CDG	LHR	14:40
447	CDG	LHR	17:00

AEROPUERTO

Código	Nombre	Ciudad
MAD	Barajas	Madrid
LGW	Gatwick	Londres
LHR	Heathrow	Londres
ORY	Orly	París
CDG	Charles de Gaulle	París

PASAJERO

DNI	Nombre
123	María
456	Pedro
789	Isabel

RESERVA

DNI	Nro_Vuelo	Fecha	Precio
789	165	07-01-11	210
123	345	20-12-10	170
789	321	15-12-10	250
456	345	03-11-10	190

Consigna: Vincular las tablas Reserva, Pasajero y Vuelo en un solo SELECT. Mostrar sólo Fecha de reserva, hora de salida del vuelo y nombre de pasajero.

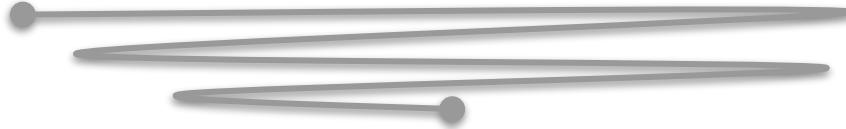
SQL

```
SELECT DISTINCT r.Fecha, v.Salida, p.Nombre  
FROM reserva AS r, pasajero AS p, vuelo AS v  
WHERE r.DNI=p.DNI AND r.Nro_Vuelo=v.Número;
```

Recordar que si NO se usa la cláusula JOIN junto con ON es probable que el RDBM no pueda optimizar la operación.

¿En este caso no optimizado, cuántas tuplas intermedias puede llegar a generar?

*SQL*



*Tuplas espúreas*

# SQL - Tuplas espúreas

1. Contamos con la siguiente tabla

EmpleadoRolProyecto

empleado	rol	proyecto
Ana	Diseñador/a	YPF
Ana	Programador/a	BNA
Ana	Diseñador/a	BNA
Bruno	Diseñador/a	BNA

2. Dividimos los datos en dos tablas  
(eliminando los duplicados), dejando  
una columna en común para poder  
juntarlas luego (con un JOIN)

EmpleadoRol

empleado	rol
Ana	Diseñador/a
Ana	Programador/a
Bruno	Diseñador/a

RolProyecto

rol	proyecto
Diseñador/a	YPF
Programador/a	BNA
Diseñador/a	BNA

# SQL - Tuplas espúreas

Consigna: Vincular (JOIN) EmpleadoRol y RolProyecto para obtener la tabla original EmpleadoRolProyecto

EmpleadoRol

empleado	rol
Ana	Diseñador/a
Ana	Programador/a
Bruno	Diseñador/a

RolProyecto

rol	proyecto
Diseñador/a	YPF
Programador/a	BNA
Diseñador/a	BNA

SQL

```
SELECT DISTINCT ...  
FROM ...  
INNER JOIN ...  
ON ...;
```



EmpleadoRolProyecto

empleado	rol	proyecto
Ana	Diseñador/a	YPF
Ana	Programador/a	BNA
Ana	Diseñador/a	BNA
Bruno	Diseñador/a	BNA



# SQL - Tuplas espúreas

Consigna: Vincular (JOIN) EmpleadoRol y RolProyecto para obtener la tabla original EmpleadoRolProyecto

EmpleadoRol

empleado	rol
Ana	Diseñador/a
Ana	Programador/a
Bruno	Diseñador/a

RolProyecto

rol	proyecto
Diseñador/a	YPF
Programador/a	BNA
Diseñador/a	BNA

SQL

```
SELECT DISTINCT er.empleado, er.rol, rp.proyecto  
FROM EmpleadoRol AS er  
INNER JOIN RolProyecto rp  
ON er.rol=rp.rol;
```

EmpleadoRolProyecto

empleado	rol	proyecto
Ana	Diseñador/a	YPF
Ana	Programador/a	BNA
Ana	Diseñador/a	BNA
Bruno	Diseñador/a	BNA

empleado	rol	rol	proyecto
Ana	Diseñador/a	Diseñador/a	YPF
Ana	Diseñador/a	Programador/a	BNA
Ana	Diseñador/a	Diseñador/a	BNA
Ana	Programador/a	Diseñador/a	YPF
Ana	Programador/a	Programador/a	BNA
Ana	Programador/a	Diseñador/a	BNA
Bruno	Diseñador/a	Diseñador/a	YPF
Bruno	Diseñador/a	Programador/a	BNA
Bruno	Diseñador/a	Diseñador/a	BNA



# SQL - Tuplas espúreas

Consigna: Vincular (JOIN) EmpleadoRol y RolProyecto para obtener la tabla original EmpleadoRolProyecto

EmpleadoRol

empleado	rol
Ana	Diseñador/a
Ana	Programador/a
Bruno	Diseñador/a

RolProyecto

rol	proyecto
Diseñador/a	YPF
Programador/a	BNA
Diseñador/a	BNA

SQL

```
SELECT DISTINCT er.empleado, er.rol, rp.proyecto  
FROM EmpleadoRol AS er  
INNER JOIN RolProyecto rp  
ON er.rol=rp.rol;
```

EmpleadoRolProyecto

empleado	rol	proyecto
Ana	Diseñador/a	YPF
Ana	Programador/a	BNA
Ana	Diseñador/a	BNA
Bruno	Diseñador/a	BNA

tupla espúrea

empleado	rol	proyecto
Ana	Diseñador/a	YPF
Ana	Programador/a	BNA
Ana	Diseñador/a	BNA
Bruno	Diseñador/a	BNA
Bruno	Diseñador/a	YPF



Error muy común y difícil de detectar

en particular cuando trabajamos con muchos datos

# SQL - Tuplas espúreas

Consigna: Vincular (JOIN) EmpleadoRol y RolProyecto para obtener la tabla original EmpleadoRolProyecto

EmpleadoRol

empleado	rol
Ana	Diseñador/a
Ana	Programador/a
Bruno	Diseñador/a

RolProyecto

rol	proyecto
Diseñador/a	YPF
Programador/a	BNA
Diseñador/a	BNA

SQL

```
SELECT DISTINCT er.empleado, er.rol, rp.proyecto  
FROM EmpleadoRol AS er  
INNER JOIN RolProyecto rp  
ON er.rol=rp.rol;
```

EmpleadoRolProyecto

empleado	rol	proyecto
Ana	Diseñador/a	YPF
Ana	Programador/a	BNA
Ana	Diseñador/a	BNA
Bruno	Diseñador/a	BNA

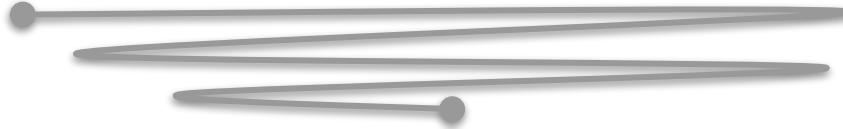
tupla espúrea

empleado	rol	proyecto
Ana	Diseñador/a	YPF
Ana	Programador/a	BNA
Ana	Diseñador/a	BNA
Bruno	Diseñador/a	BNA
Bruno	Diseñador/a	YPF



Para que no suceda, la condición de JOIN SIEMPRE debe ser superclave de una de las tablas que participa del JOIN

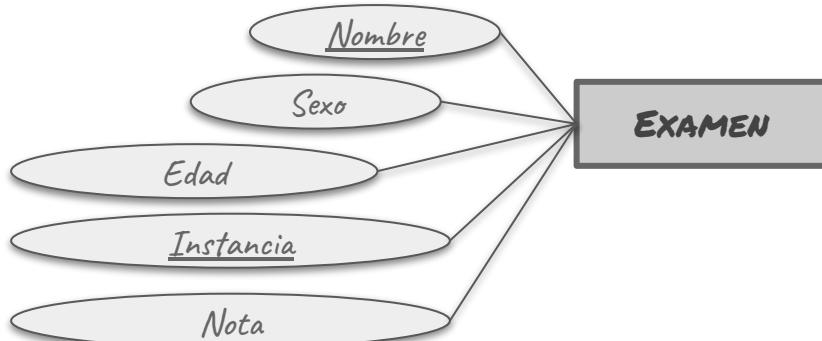
*SQL*



## *Funciones de agregación*

# SQL - Funciones de agregación

1. Contamos con el siguiente DER



# SQL - Funciones de agregación

Consigna: Usando sólo SELECT contar cuántos exámenes fueron rendidos (en total)

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

SELECT ...

CantidadExamenes  
16



# SQL - Funciones de agregación

Consigna: Usando sólo SELECT contar cuántos exámenes fueron rendidos (en total)

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT count(*) AS cantidadExamenes  
FROM examen;
```

CantidadExamenes

16

# SQL - Funciones de agregación

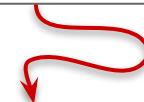
Consigna: Usando sólo SELECT contar cuántos exámenes fueron rendidos en cada Instancia

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

SELECT ...



Instancia	Asistieron
Parcial-01	6
Recuperatorio-01	3
Parcial-02	5
Recuperatorio-02	2

# SQL - Funciones de agregación

Consigna: Usando sólo SELECT contar cuántos exámenes fueron rendidos en cada Instancia

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT Instancia, COUNT(*) AS Asistieron  
FROM Examen  
GROUP BY Instancia;
```



Instancia	Asistieron
Parcial-01	6
Recuperatorio-01	3
Parcial-02	5
Recuperatorio-02	2

# SQL - Funciones de agregación

Consigna: Usando sólo SELECT contar cuántos exámenes fueron rendidos en cada Instancia

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT Instancia, COUNT(*) AS Asistieron  
FROM Examen  
GROUP BY Instancia;
```

Instancia	Asistieron
Parcial-01	6
Recuperatorio-01	3
Parcial-02	5
Recuperatorio-02	2

# SQL - Funciones de agregación

Consigna: Usando sólo SELECT contar cuántos exámenes fueron rendidos en cada Instancia

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT Instancia, COUNT(*) AS Asistieron  
FROM Examen  
GROUP BY Instancia;
```

6 } 3 } 5 } 2 }

Instancia	Asistieron
Parcial-01	6
Recuperatorio-01	3
Parcial-02	5
Recuperatorio-02	2

# SQL - Funciones de agregación

Consigna: Usando sólo SELECT contar cuántos exámenes fueron rendidos en cada Instancia (ordenado x instancia)

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT Instancia, COUNT(*) AS Asistieron  
FROM Examen  
GROUP BY Instancia;
```

Instancia	Asistieron
Parcial-01	6
Recuperatorio-01	3
Parcial-02	5
Recuperatorio-02	2

# SQL - Funciones de agregación

Consigna: Usando sólo SELECT contar cuántos exámenes fueron rendidos en cada Instancia (ordenado x instancia)

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT Instancia, COUNT(*) AS Asistieron  
FROM Examen  
GROUP BY Instancia  
ORDER BY Instancia ASC;
```

Instancia	Asistieron
Parcial-01	6
Parcial-02	5
Recuperatorio-01	3
Recuperatorio-02	2

Se puede explicitar si es ascendente o descendente:

*ORDER BY Instancia ASC*

*ORDER BY Instancia DESC*

# SQL - Funciones de agregación

Consigna: Ídem ejercicio anterior, pero mostrar sólo las instancias a las que asistieron menos de 4 Estudiantes.

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT ... ;
```

6  
3  
5  
2

Instancia	Asistieron
Recuperatorio-01	3
Recuperatorio-02	2

# SQL - Funciones de agregación

Consigna: Ídem ejercicio anterior, pero mostrar sólo las instancias a las que asistieron menos de 4 Estudiantes.

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

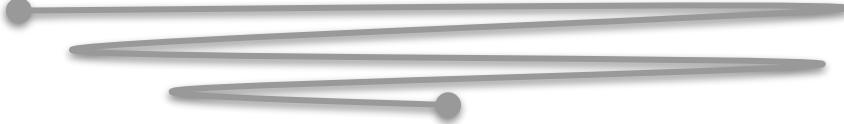
SQL

```
SELECT Instancia, COUNT(*) AS Asistieron  
FROM Examen  
GROUP BY Instancia  
HAVING Asistieron < 4  
ORDER BY Instancia;
```

Instancia	Asistieron
Recuperatorio-01	3
Recuperatorio-02	2

**HAVING** permite poner condiciones a los valores resultantes de las funciones de grupo

## *SQL - Funciones de agregación*



- *COUNT(). Cantidad de elementos en el grupo.*
- *MAX(). Máximo registrado en el grupo.*
- *MIN(). Mínimo registrado en el grupo.*
- *SUM(). Sumatoria correspondiente al grupo.*
- *AVG(). Promedio correspondiente al grupo.*

# SQL - Funciones de agregación

Consigna: Mostrar el promedio de edad de los estudiantes en cada instancia de examen

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT . . . ;
```

Instancia	PromedioEdad
Parcial-01	21,83
Parcial-02	21,8
Recuperatorio-01	19,67
Recuperatorio-02	21,00

# SQL - Funciones de agregación

Consigna: Mostrar el promedio de edad de los estudiantes en cada instancia de examen

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT Instancia,  
       AVG(Edad) AS PromedioEdad  
FROM Examen  
GROUP BY Instancia  
ORDER BY Instancia;
```

Instancia	PromedioEdad
Parcial-01	21,83
Parcial-02	21,8
Recuperatorio-01	19,67
Recuperatorio-02	21,00

*SQL*



*LIKE*

# SQL - LIKE

Consigna: Mostrar cuál fue el promedio de notas en cada instancia de examen, sólo para instancias de parcial.

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

SELECT . . . ;



Instancia	NotaPromedio
Parcial-01	5,17
Parcial-02	6,8

# SQL - LIKE

Consigna: Mostrar cuál fue el promedio de notas en cada instancia de examen, sólo para instancias de parcial.

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT Instancia,  
       AVG(Nota) AS NotaPromedio  
FROM Examen  
GROUP BY Instancia  
HAVING instancia='Parcial-01' OR  
       instancia='Parcial-02'  
ORDER BY Instancia;
```

Instancia	NotaPromedio
Parcial-01	5,17
Parcial-02	6,8

SQL permite usar comodines a través de LIKE

# SQL - LIKE

Consigna: Mostrar cuál fue el promedio de notas en cada instancia de examen, sólo para instancias de parcial.

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT Instancia,  
       AVG(Nota) AS PromedioNota  
FROM Examen  
GROUP BY Instancia  
HAVING instancia='Parcial%'  
ORDER BY Instancia;
```

Instancia	NotaPromedio
Parcial-01	5,17
Parcial-02	6,8

SQL permite usar comodines a través de LIKE

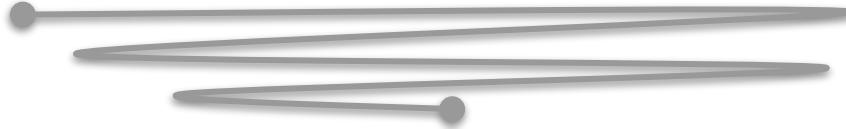
# SQL - LIKE

SQL

```
SELECT columna1, columna2, ..., columnaN
FROM tabla
...
(WHERE) HAVING columnai LIKE patrón;
```

Condición	Devuelve verdadero si el valor de la columna, ...
WHERE/HAVING columna <sub>i</sub> LIKE 'a%'	comienza con "a"
WHERE/HAVING columna <sub>i</sub> LIKE '%a'	finaliza con "a"
WHERE/HAVING columna <sub>i</sub> LIKE '%or%'	contiene el valor "or" en cualquier posición de la cadena
WHERE/HAVING columna <sub>i</sub> LIKE '_r%'	contiene "r" en la segunda posición de la cadena
WHERE/HAVING columna <sub>i</sub> LIKE 'a_%'	comienza con "a" y al menos posee 2 caracteres
WHERE/HAVING columna <sub>i</sub> LIKE 'a__%'	comienza con "a" y al menos posee 3 caracteres
WHERE/HAVING columna <sub>i</sub> LIKE 'a%o'	comienza con "a" y finaliza con "o"

*SQL*



*Eligiendo ...*

# SQL - Eligiendo

Consigna: Listar a cada alumno que rindió el Parcial-01 y decir si aprobó o no (se aprueba con nota  $\geq 4$ )

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

SELECT . . . ;



Nombre	Nota	Estado
Ana	10	APROBÓ
Bruno	2	NO APROBÓ
Camila	3	NO APROBÓ
Diego	1	NO APROBÓ
Eva	7	APROBÓ
Francisco	8	APROBÓ

# SQL - Eligiendo

Consigna: Listar a cada alumno que rindió el Parcial-01 y decir si aprobó o no (se aprueba con nota  $\geq 4$ )

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT Nombre,  
       Nota,  
       CASE WHEN Nota >= 4  
             THEN 'APROBÓ'  
             ELSE 'NO APROBÓ'  
         END AS Estado  
  
FROM Examen  
  
WHERE Instancia='Parcial-01'  
  
ORDER BY Nombre;
```

Nombre	Nota	Estado
Ana	10	APROBÓ
Bruno	2	NO APROBÓ
Camila	3	NO APROBÓ
Diego	1	NO APROBÓ
Eva	7	APROBÓ
Francisco	8	APROBÓ

# SQL - Eligiendo

Consigna: Modificar la consulta anterior para que informe cuántos estudiantes aprobaron/reprobaron en cada instancia.

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

SELECT . . . ;



Instancia	Estado	Cantidad
Parcial-01	APROBÓ	3
Parcial-01	NO APROBÓ	3
Parcial-02	APROBÓ	4
Parcial-02	NO APROBÓ	1
Recuperatorio-01	APROBÓ	2
Recuperatorio-01	NO APROBÓ	1
Recuperatorio-02	APROBÓ	2

# SQL - Eligiendo

Consigna: Modificar la consulta anterior para que informe cuántos estudiantes aprobaron/reprobaron en cada instancia.

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT Instancia,  
       CASE WHEN Nota>=4  
              THEN 'APROBÓ'  
              ELSE 'NO APROBÓ'  
         END AS Estado,  
       COUNT(*) as Cantidad  
FROM Examen  
GROUP BY Instancia, Estado  
ORDER BY Instancia, Estado;
```

Instancia	Estado	Cantidad
Parcial-01	APROBÓ	3
Parcial-01	NO APROBÓ	3
Parcial-02	APROBÓ	4
Parcial-02	NO APROBÓ	1
Recuperatorio-01	APROBÓ	2
Recuperatorio-01	NO APROBÓ	1
Recuperatorio-02	APROBÓ	2

*SQL*



*Subqueries ...*

# SQL - Subqueries

Consigna: Listar los alumnos que en cada instancia obtuvieron una nota mayor al promedio de dicha instancia

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

5,17  
5,33  
6,80  
7,00

Nombre	Instancia	Nota
Ana	Parcial-01	10
Francisco	Parcial-01	8
Eva	Parcial-01	7
Ana	Parcial-02	10
Eva	Parcial-02	9
Bruno	Parcial-02	7
Camila	Parcial-02	7
Camila	Recuperatorio-01	7
Bruno	Recuperatorio-01	6
Diego	Recuperatorio-02	9

SQL

SELECT ... ;



# SQL - Subqueries

Consigna: Listar los alumnos que en cada instancia obtuvieron una nota mayor al promedio de dicha instancia

## Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

51.5  
53.5  
63.5  
9.80  
2.00

Nombre	Instancia	Nota
Ana	Parcial-01	10
Francisco	Parcial-01	8
Eva	Parcial-01	7
Ana	Parcial-02	10
Eva	Parcial-02	9
Bruno	Parcial-02	7
Camila	Parcial-02	7
Camila	Recuperatorio-01	7
Bruno	Recuperatorio-01	6
Diego	Recuperatorio-02	9

## SQL

```

SELECT e1.Nombre, e1.Instancia, e1.Nota
FROM Examen AS e1
WHERE e1.Nota > (
    SELECT AVG(e2.Nota)
    FROM Examen AS e2
    WHERE e2.Instancia = e1.Instancia
)
ORDER BY Instancia ASC, Nota DESC;
  
```

Es responsabilidad del programador asegurar que el subquery devolverá una sola fila.  
 ¡Si el subquery devuelve 0 o + de 1 fila, da error!

# SQL - Subqueries

Consigna: Listar los alumnos que en cada instancia obtuvieron una nota mayor al promedio de dicha instancia

## Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

51.5  
53.5  
6.80  
7.00

Nombre	Instancia	Nota
Ana	Parcial-01	10
Francisco	Parcial-01	8
Eva	Parcial-01	7
Ana	Parcial-02	10
Eva	Parcial-02	9
Bruno	Parcial-02	7
Camila	Parcial-02	7
Camila	Recuperatorio-01	7
Bruno	Recuperatorio-01	6
Diego	Recuperatorio-02	9

## SQL

```

SELECT e1.Nombre, e1.Instancia, e1.Nota
FROM Examen AS e1
WHERE e1.Nota > (
    SELECT AVG(e2.Nota)
    FROM Examen AS e2
    WHERE e2.Instancia = e1.Instancia
)
ORDER BY Instancia ASC, Nota DESC;

```

Operadores Single-rows

para subqueries que retornan una sola fila:

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

# SQL - Subqueries

Nombre	Instancia	Nota
Ana	Parcial-01	10
Ana	Parcial-02	10
Camila	Recuperatorio-01	7
Diego	Recuperatorio-02	9

Consigna: Listar los alumnos que en cada instancia obtuvieron la mayor nota de dicha instancia

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

SELECT ... ;



# SQL - Subqueries

Nombre	Instancia	Nota
Ana	Parcial-01	10
Ana	Parcial-02	10
Camila	Recuperatorio-01	7
Diego	Recuperatorio-02	9

Consigna: Listar los alumnos que en cada instancia obtuvieron la mayor nota de dicha instancia

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

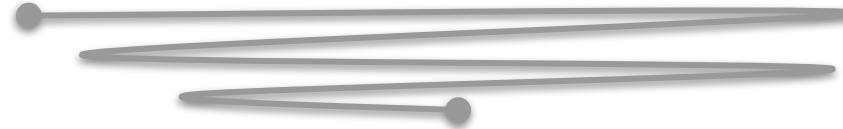
```

SELECT e1.Nombre, e1.Instancia, e1.Nota
FROM Examen AS e1
WHERE e1.Nota >= ALL (
    SELECT e2.Nota
    FROM Examen AS e2
    WHERE e2.Instancia = e1.Instancia
)
ORDER BY e1.Instancia ASC, e1.Nombre ASC;

```

Operadores Multiple-rows  
para subqueries que retornan varias filas:  
*IN, ANY, ALL, EXISTS*

# SQL - Subqueries operadores multiple-rows



OPERADOR	SIGNIFICADO
IN	Retorna TRUE si está incluido en los valores retornados por el subquery
ANY	Retorna TRUE si la comparación es TRUE para al menos un valor retornado por el subquery
ALL	Retorna TRUE si la comparación es TRUE para todos los valores retornados por el subquery
EXISTS	Retorna TRUE si el subquery devuelve al menos una fila. FALSE si devuelve 0 filas

# SQL - Subqueries

Nombre	Instancia	Nota
Ana	Parcial-01	10
Ana	Parcial-02	10
Eva	Parcial-01	7
Eva	Parcial-02	9

Consigna: Listar el nombre, instancia y nota sólo de los estudiantes que no rindieron ningún Recuperatorio

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

SELECT . . . ;



# SQL - Subqueries

Nombre	Instancia	Nota
Ana	Parcial-01	10
Ana	Parcial-02	10
Eva	Parcial-01	7
Eva	Parcial-02	9

Consigna: Listar el nombre, instancia y nota sólo de los estudiantes que no rindieron ningún Recuperatorio

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

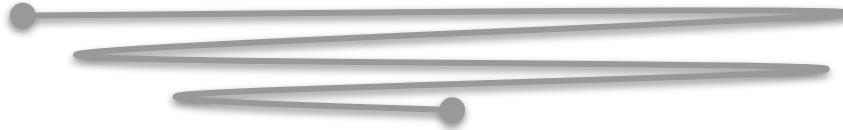
SQL

```

SELECT e1.Nombre, e1.Instancia, e1.Nota
FROM Examen AS e1
WHERE NOT EXISTS (
    SELECT *
    FROM Examen AS e2
    WHERE e2.Nombre = e1.Nombre AND
        e2.Instancia LIKE 'Recuperatorio%'
)
ORDER BY e1.Nombre ASC, e1.Instancia ASC;

```

*SQL*



*Integrando variables de Python*

# SQL - Integrando variables de Python

Consigna: Mostrar Nombre, Instancia y Nota de los alumnos cuya Nota supera el umbral indicado en la variable de Python `umbralNota`

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

Nombre	Instancia	Nota
Ana	Parcial-01	10
Francisco	Parcial-01	8
Ana	Parcial-02	10
Eva	Parcial-02	9
Diego	Recuperatorio-02	9

Ejemplo con `umbralNota = 7`

SQL

`SELECT ... ;`



# SQL - Integrando variables de Python

Consigna: Mostrar Nombre, Instancia y Nota de los alumnos cuya Nota supera el umbral indicado en la variable de Python `umbralNota`

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

Nombre	Instancia	Nota
Ana	Parcial-01	10
Francisco	Parcial-01	8
Ana	Parcial-02	10
Eva	Parcial-02	9
Diego	Recuperatorio-02	9

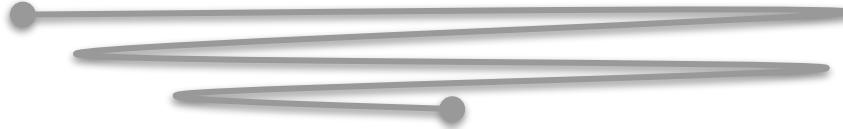
Ejemplo con `umbralNota = 7`

SQL

```
SELECT Nombre, Instancia, Nota
FROM Examen
WHERE Nota > $umbralNota;
```

Se agrega el símbolo \$ delante de la variable

*SQL*



*Manejo de valores NULL*

# SQL - Manejo de NULLs

Consigna: Listar todas las tuplas de Examen03 cuyas Notas son menores a 9

Examen03

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	NULL
Francisco	M	22	Parcial-01	8

SQL

```
SELECT ...;
```



# SQL - Manejo de NULLs

Consigna: Listar todas las tuplas de Examen03 cuyas Notas son menores a 9

Examen03

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	NULL
Francisco	M	22	Parcial-01	8

SQL

```
SELECT *
FROM Examen03
WHERE Nota < 9;
```

Nombre	Sexo	Edad	Instancia	Nota
Francisco	M	22	Parcial-01	8

¿Qué pasó con  
Nota == NULL?

# SQL - Manejo de NULLs

Consigna: Listar todas las tuplas de Examen03 cuyas Notas son **mayores o iguales** a 9

Examen03

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	NULL
Francisco	M	22	Parcial-01	8

SQL

```
SELECT ...;
```



# SQL - Manejo de NULLs

Consigna: Listar todas las tuplas de Examen03 cuyas Notas son **mayores o iguales** a 9

Examen03

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	NULL
Francisco	M	22	Parcial-01	8

SQL

```
SELECT *
FROM Examen03
WHERE Nota >= 9;
```

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10

¿Qué pasó con  
Nota == NULL?

# SQL - Manejo de NULLs

Consigna: Listar el **UNION** de todas las tuplas de Examen03 cuyas Notas son **menores** a 9 y las que son **mayores o iguales** a 9

Examen03

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	NULL
Francisco	M	22	Parcial-01	8

SQL

```
SELECT ...;
```



# SQL - Manejo de NULLs

Consigna: Listar el **UNION** de todas las tuplas de Examen03 cuyas Notas son **menores** a 9 y las que son **mayores o iguales** a 9

Examen03

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	NULL
Francisco	M	22	Parcial-01	8

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Francisco	M	22	Parcial-01	8

SQL

```
SELECT *
FROM Examen03
WHERE Nota < 9
UNION
SELECT *
FROM Examen03
WHERE Nota >= 9;
```



¡No aparece el  
NULL!

# SQL - Manejo de NULLs

Consigna: Obtener el promedio de notas

Examen03

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	NULL
Francisco	M	22	Parcial-01	8

SQL

```
SELECT AVG(Nota) AS NotaPromedio  
FROM Examen03;
```



$$\text{Wavy arrow} \rightarrow (10+8)/2 = 9$$

SQL

```
SELECT ...;
```



$$\text{Wavy arrow} \rightarrow (10+0+8)/3 = 6$$

# SQL - Manejo de NULLs

Consigna: Obtener el promedio de notas

Examen03

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	NULL
Francisco	M	22	Parcial-01	8

SQL

```
SELECT AVG(Nota) AS NotaPromedio  
FROM Examen03;
```



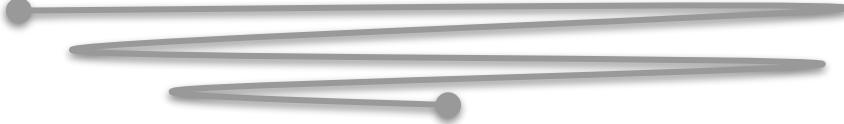
$$\text{Wavy arrow} \rightarrow (10+8)/2 = 9$$

SQL

```
SELECT AVG(CASE WHEN Nota IS NULL THEN 0 ELSE Nota END) AS NotaPromedio  
FROM Examen03;
```

$$\text{Wavy arrow} \rightarrow (10+0+8)/3 = 6$$

# SQL - Manejo de NULLs



- ✓ La presencia de NULL genera algunas complicaciones
  - ¿“Nota > 9” - TRUE o FALSE cuando Nota es NULL?
  - $(NULL > 0)$  es NULL
  - $(NULL + 0)$  es NULL
  - $(NULL = 0)$  es NULL
  - $(NULL \text{ AND } TRUE)$  es NULL
- ✓ Hay que ser cuidadoso con la cláusula WHERE
- ✓ En SQL el WHERE elimina toda fila que NO es evaluada como TRUE (es decir, condiciones que son evaluadas como FALSE o NULL se descartan)
- ✓ Surge la necesidad de una “3-valued logic” (TRUE, FALSE y NULL)
- ✓ Operador especial para controlar si un valor es nulo (IS NULL o IS NOT NULL)

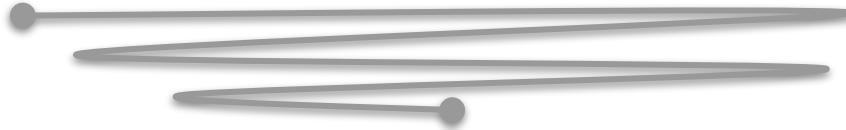
# *SQL - Lógica trivaluada*



AND	T	F	NULL
T	T	F	NULL
F	F	F	F
NULL	NULL	F	NULL

OR	T	F	NULL
T	T	T	T
F	T	F	NULL
NULL	T	NULL	NULL

*SQL*



*Mayúsculas/Minúsculas*

# SQL - Reemplazos

Consigna: Transformar todos los caracteres de las descripciones de los roles a mayúscula

EmpleadoRol

empleado	rol
Ana	Diseñador/a
Ana	Programador/a
Bruno	Diseñador/a



EmpleadoRol

empleado	rol
Ana	DISEÑADOR/A
Ana	PROGRAMADOR/A
Bruno	DISEÑADOR/A

SQL

```
SELECT ...;
```



# SQL - Reemplazos

Consigna: Transformar todos los caracteres de las descripciones de los roles a mayúscula

EmpleadoRol

empleado	rol
Ana	Diseñador/a
Ana	Programador/a
Bruno	Diseñador/a



EmpleadoRol

empleado	rol
Ana	DISEÑADOR/A
Ana	PROGRAMADOR/A
Bruno	DISEÑADOR/A

SQL

```
SELECT empleado, UPPER(rol) AS rol  
FROM EmpleadoRol;
```

# SQL - Reemplazos

Consigna: Transformar todos los caracteres de las descripciones de los roles a minúscula

EmpleadoRol

empleado	rol
Ana	Diseñador/a
Ana	Programador/a
Bruno	Diseñador/a



EmpleadoRol

empleado	rol
Ana	diseñador/a
Ana	programador/a
Bruno	diseñador/a

SQL

```
SELECT ...;
```



# SQL - Reemplazos

Consigna: Transformar todos los caracteres de las descripciones de los roles a mayúscula

EmpleadoRol

empleado	rol
Ana	Diseñador/a
Ana	Programador/a
Bruno	Diseñador/a



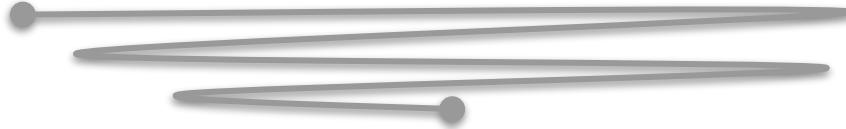
EmpleadoRol

empleado	rol
Ana	diseñador/a
Ana	programador/a
Bruno	diseñador/a

SQL

```
SELECT empleado, LOWER(rol) AS rol  
FROM EmpleadoRol;
```

*SQL*



*Reemplazos*

# SQL - Reemplazos

Consigna: En la descripción de los roles de los empleados reemplazar las ñ por ni

EmpleadoRol

empleado	rol
Ana	Diseñador/a
Ana	Programador/a
Bruno	Diseñador/a



EmpleadoRol

empleado	rol
Ana	Disenador/a
Ana	Programador/a
Bruno	Disenador/a

SQL

```
SELECT ...;
```



## SQL - Reemplazos

Consigna: En la descripción de los roles de los empleados reemplazar las ñ por ni

EmpleadoRol

empleado	rol
Ana	Diseñador/a
Ana	Programador/a
Bruno	Diseñador/a



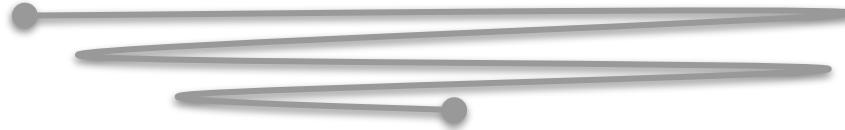
EmpleadoRol

empleado	rol
Ana	Disenador/a
Ana	Programador/a
Bruno	Disenador/a

SQL

```
SELECT empleado, REPLACE(rol,'ñ','ni') AS rol  
FROM EmpleadoRol;
```

*SQL*



*Desafío*

# SQL - Desafío 01

Consigna: Mostrar para cada estudiante las siguientes columnas con sus datos: Nombre, Sexo, Edad, Nota-Parcial-01, Nota-Parcial-02, Recuperatorio-01 y , Recuperatorio-02

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

SELECT ... ;

SELECT ... ;

SELECT ... ;

SELECT ... ;



Más de 1 paso



Nombre	Sexo	Edad	Parcial_01	Parcial_02	Recuperatorio_01	Recuperatorio_02
Ana	F	20	10	10	NULL	NULL
Bruno	M	19	2	7	6	NULL
Camila	F	20	3	7	7	NULL
Diego	M	20	1	1	3	9
Eva	F	30	7	9	NULL	NULL
Francisco	M	22	8	NULL	NULL	5

# SQL - Desafío 02

Consigna: Agregar al ejercicio anterior la columna Estado, que informa si el alumno aprobó la cursada (APROBÓ/NO APROBÓ). Se aprueba con 4.

Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

SQL

```
SELECT ...;  
SELECT ...;  
SELECT ...;  
SELECT ...;
```



Más de 1 paso



Nombre	Sexo	Edad	Parcial_01	Parcial_02	Recuperatorio_01	Recuperatorio_02	Estado
Ana	F	20	10	10	NULL	NULL	APROBÓ
Bruno	M	19	2	7	6	NULL	APROBÓ
Camila	F	20	3	7	7	NULL	APROBÓ
Diego	M	20	1	1	3	9	NO APROBÓ
Eva	F	30	7	9	NULL	NULL	APROBÓ
Francisco	M	22	8	NULL	NULL	5	APROBÓ

# SQL - Desafío 03

Consigna: Generar la tabla Examen a partir de la tabla obtenida en el desafío anterior

Nombre	Sexo	Edad	Parcial_01	Parcial_02	Recuperatorio_01	Recuperatorio_02	Estado
Ana	F	20	10	10	NULL	NULL	APROBÓ
Bruno	M	19	2	7	6	NULL	APROBÓ
Camila	F	20	3	7	7	NULL	APROBÓ
Diego	M	20	1	1	3	9	NO APROBÓ
Eva	F	30	7	9	NULL	NULL	APROBÓ
Francisco	M	22	8	NULL	NULL	5	APROBÓ

SQL

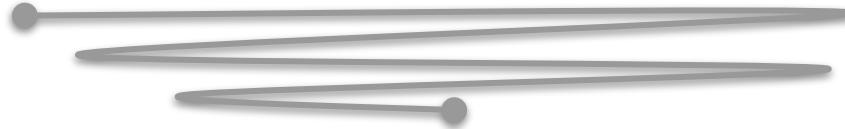
```
SELECT . . . ;
```



## Examen

Nombre	Sexo	Edad	Instancia	Nota
Ana	F	20	Parcial-01	10
Bruno	M	19	Parcial-01	2
Camila	F	20	Parcial-01	3
Diego	M	20	Parcial-01	1
Eva	F	30	Parcial-01	7
Francisco	M	22	Parcial-01	8
Bruno	M	19	Recuperatorio-01	6
Camila	F	20	Recuperatorio-01	7
Diego	M	20	Recuperatorio-01	3
Ana	F	20	Parcial-02	10
Bruno	M	19	Parcial-02	7
Camila	F	20	Parcial-02	7
Diego	M	20	Parcial-02	1
Eva	F	30	Parcial-02	9
Diego	M	20	Recuperatorio-02	9
Francisco	M	22	Recuperatorio-02	5

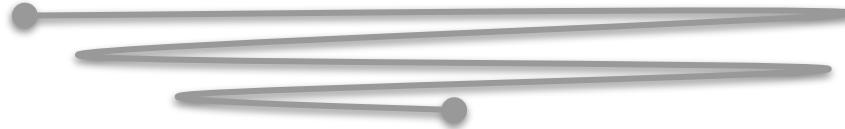
# *SQL - Resumen (no exhaustivo)*



*SQL*

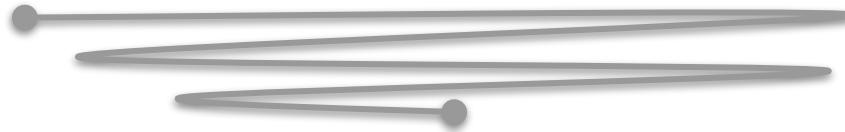
```
SELECT [ALL/DISTINCT] lista_campos  
FROM tabla [AS alias tabla] [,....]  
[WHERE condición]  
[GROUP BY lista_columnas]  
[HAVING condición]  
[ORDER BY nombre_columna [ASC/DESC] [,....];
```

## Cierre



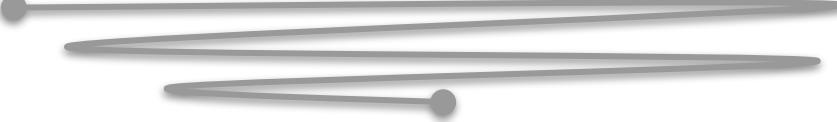
1. Álgebra Relacional. Provee fundamento formal a las operaciones asociadas al modelo relacional
2. SQL. Nos permite acceder a los datos independientemente de cómo se encuentran almacenados físicamente

## *Tareas para la próxima clase*



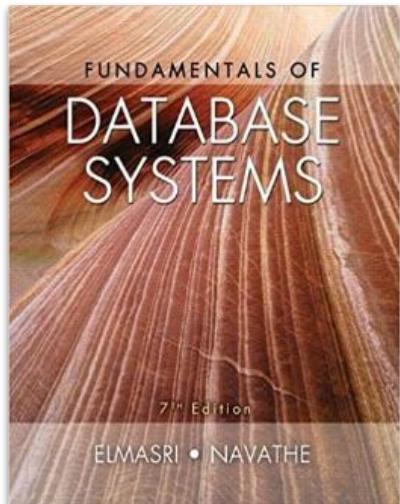
- 1. Resolver la guía de ejercicios de “SQL”*

## *Tareas para el Jueves 15/02/2024*

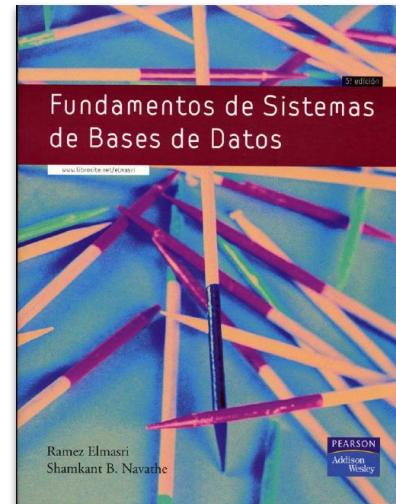


- 1. Traer leidos los siguientes documentos (ver sitio Campus)*
  - a. Ley 25.326 - “Protección de Datos Personales”*
  - b. Proyecto de nueva Ley de “Protección de Datos Personales” (2023)*

# Bibliografía



Elmasri/Navathe, *Fundamentals of Database Systems*,  
7th. Ed., Pearson, 2016.



Elmasri/Navathe, *Fundamentos de Sistemas de Bases de Datos*,  
5ta Ed., Pearson, 2007.

(Aviso. Difieren un poco en la notación)