

# Base de Datos I

---

Introducción a SQL - DDL

# Herramientas

- MySQL

- Es un Sistema de Gestión de Bases de Datos (SGBD)
- Tiene una versión gratuita y otra paga
- Hay que descargar el [Server](#) y el [Workbench \(IDE\)](#)



- SQLFiddle (opcional)

- Es una web que genera una Base de datos para que podamos trabajar directamente
- Se puede acceder en [sqlfiddle.com](http://sqlfiddle.com)

# Conceptos

- SQL (Structured Query Language)
  - DDL (Data Definition Language)
    - Se utiliza para crear y modificar la estructura de la Base de datos
    - Algunas palabras reservadas son: CREATE, ALTER, DROP, TRUNCATE
  - DML (Data Manipulation Language)
    - Son sentencias utilizadas para la manipulación (crear, eliminar, modificar, consultar) de los datos de una base de datos.
    - Algunas palabras reservadas: SELECT, INSERT, UPDATE, DELETE,

# Tipos de datos

---

# Tipos de datos

- INT
  - Ocupa 4 bytes
  - Puede alojar un número del -2147483648 al 2147483647
  - Opción UNSIGNED
  - Otras variantes: TINYINT, SMALLINT, MEDIUMINT, BIGINT

# Variantes Integer

MySQL 8.0 Reference Manual / ... / Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT

version 8.0 ▼

## 11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT

MySQL supports the SQL standard integer types `INTEGER` (or `INT`) and `SMALLINT`. As an extension to the standard, MySQL also supports the integer types `TINYINT`, `MEDIUMINT`, and `BIGINT`. The following table shows the required storage and range for each integer type.

**Table 11.1 Required Storage and Range for Integer Types Supported by MySQL**

Type	Storage (Bytes)	Minimum Value Signed	Minimum Value Unsigned	Maximum Value Signed	Maximum Value Unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	$-2^{63}$	0	$2^{63}-1$	$2^{64}-1$

# Tipos de datos

- **DECIMAL(M,D)**
  - Longitud no requerida. M=Longitud total, incluyendo decimales (Default 10), D=Decimales (Default 0).
  - Es equivalente a NUMERIC
  - Otras variantes: FLOAT, DOUBLE

# Tipos de datos

- BOOL, BOOLEAN
  - Es equivalente a TINYINT



# Tipos de datos

- DATE
  - Formato Default: 'yyyy-mm-dd'
  - La base de datos almacena la fecha en un formato desconocido por nosotros
  - Para enviar y obtener fechas a la base de datos utilizamos una cadena de caracteres ej 'yyyy-mm-dd', 'yyyymmdd', 'yyyy/mm/dd'

# Tipos de datos

- TIME
  - Formato Default: 'hh:mi:ss'

# Tipos de datos

- DATETIME
  - Formato Default: 'yyyy-mm-dd hh:mi:ss'

# Tipos de datos

- CHAR(M)
  - Soporta una cantidad fija de caracteres
  - Longitud no requerida (Default 1)
  - Completa con espacios hasta la longitud
  - Hasta 255 caracteres

# Tipos de datos

- **VARCHAR(M)**
  - Sirve para guardar una cadena de caracteres variable
  - Longitud requerida
  - Longitud + 1 byte (cuando longitud es  $\leq 255$ ) o 2 bytes ( $>255$ )
  - Hasta 65532 caracteres

# Tipos de datos

- TEXT
  - Otras variantes: TINYTEXT, MEDIUMTEXT, LONGTEXT

# Tipos de datos

- BLOB
  - Binary Large Object
  - Nos sirve para guardar archivos de cualquier tipo en la base de datos
  - Otras variantes: TINYBLOB, MEDIUMBLOB, LONGBLOB

# Sentencias DDL

---



# Nomenclatura

PALABRA RESERVADA OPCION1 | OPCION2 [OPCIONAL] <nombre>;

# Creación de una base de datos

```
CREATE DATABASE | SCHEMA [IF NOT EXISTS] <NOMBRE>;
```

```
DROP DATABASE | SCHEMA [IF EXISTS] <NOMBRE>;
```

```
USE <NOMBRE>;
```



```
CREATE SCHEMA IF NOT EXISTS universidad;  
DROP DATABASE universidad;  
USE universidad;
```

# Creación de tablas

CREATE TABLE <TABLA>


(<CAMPO\_1> <TIPO\_DATO\_1> [RESTRICCIONES\_CAMPO\_1],  
<CAMPO\_2> <TIPO\_DATO\_2> [RESTRICCIONES\_CAMPO\_2],  
...  
<CAMPO\_N> <TIPO\_DATO\_N> [RESTRICCIONES\_CAMPO\_N],  
  
[RESTRICCIONES\_TABLA]);



```
CREATE TABLE alumnos (legajo SMALLINT,  
    nombre VARCHAR(40),  
    apellido VARCHAR(40),  
    fecha_nacimiento DATE);
```

# Restricciones de campo

- NOT NULL
- PRIMARY KEY (Claves primarias simples)
- UNIQUE (Claves únicas simples)
- AUTO\_INCREMENT (Tipos numéricos. Debe ser PK. Sólo puede haber una)
- DEFAULT <VALOR>




```
CREATE TABLE alumno (legajo SMALLINT PRIMARY KEY AUTO_INCREMENT,  
                        nombre VARCHAR(40) NOT NULL,  
                        apellido VARCHAR(40) NOT NULL,  
                        fecha_nacimiento DATE);
```

# Restricciones de tabla

- [CONSTRAINT <NOMBRE>] PRIMARY KEY(<LISTA\_CAMPOS>)
- [CONSTRAINT <NOMBRE>] FOREIGN KEY(<LISTA\_CAMPOS>) REFERENCES <TABLA\_REF>(<LISTA\_CAMPOS\_REF>)
- [CONSTRAINT <NOMBRE>] UNIQUE(<LISTA\_CAMPOS>)
- [CONSTRAINT <NOMBRE>] INDEX(<LISTA\_CAMPOS>)





```
CREATE TABLE tipo_documento (tipo CHAR(3) PRIMARY KEY,  
                                descripcion VARCHAR(50));  
  
CREATE TABLE alumno (tipo_documento CHAR(3),  
                        numero_documento VARCHAR(15),  
                        nombre VARCHAR(40),  
                        apellido VARCHAR(40) NOT NULL,  
                        fecha_nacimiento DATE,  
                        CONSTRAINT alumno_pk PRIMARY KEY (tipo_documento, numero_documento),  
                        FOREIGN KEY (tipo_documento) REFERENCES tipo_documento(tipo));
```

# Borrado de tablas

`DROP TABLE <TABLA>;`



A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The text `DROP TABLE alumno;` is displayed in a light purple font.

```
DROP TABLE alumno;
```

# Modificación de tablas

- `ALTER TABLE <TABLA> ADD COLUMN <CAMPO> <TIPO_DATO> [RESTRICCIONES_CAMPO];`
- `ALTER TABLE <TABLA> DROP COLUMN <CAMPO>;`
- `ALTER TABLE <TABLA> CHANGE <NOMBRE_CAMPO_VIEJO> <NOMBRE_CAMPO_NUEVO> <TIPO_DATO> [RESTRICCIONES_CAMPO];`



```
ALTER TABLE alumno ADD COLUMN domicilio VARCHAR(100) NOT NULL;
```

```
ALTER TABLE alumno DROP COLUMN nombre;
```

```
ALTER TABLE alumno CHANGE apellido nombre_apellido VARCHAR(100) NOT NULL;
```

# Modificación de restricciones

- `ALTER TABLE <TABLA> ADD PRIMARY KEY(<LISTA_CAMPOS>);`
- `ALTER TABLE <TABLA> DROP PRIMARY KEY;`
- `ALTER TABLE <TABLA> ADD [CONSTRAINT <NOMBRE>]  
UNIQUE|INDEX(<LISTA_CAMPOS>);`
- `ALTER TABLE <TABLA> DROP INDEX <NOMBRE>;` (Borra Unique e Index)
- `ALTER TABLE <TABLA> ADD [CONSTRAINT <NOMBRE>] FOREIGN  
KEY(<LISTA_CAMPOS>) REFERENCES <TABLA_REF>(<LISTA_CAMPOS_REF>);`
- `ALTER TABLE <TABLA> DROP FOREIGN KEY <NOMBRE>;`



```
ALTER TABLE alumno ADD PRIMARY KEY (tipo_documento, numero_documento);  
  
ALTER TABLE alumno ADD FOREIGN KEY (tipo_documento) REFERENCES tipo_documento(tipo);
```

# Base de Datos I

---

SQL - SELECT Simple

# Modelo de datos

**Empleado**

<u>nro</u>	nombre	<u>cod_esp</u>	<u>nro_jefe</u>	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

**Trabaja**

<u>nro_emp</u>	<u>cod_area</u>
1000	A1
1000	A2
1001	A1
1002	A2

**Area**

<u>cod_area</u>	descripcion
A1	Area 1
A2	Area 2

**Especialidad**

<u>cod_esp</u>	descripcion
1	Gerente
2	Operario



# Sintaxis

```
SELECT [DISTINCT] * | <lista_campos>  
FROM <lista_tablas>  
[WHERE <condicion>]  
[GROUP BY <lista_campos_agrupamiento>]  
[HAVING <condicion_post_agrupamiento>]  
[ORDER BY <lista_campos_orden>];
```

# Ejercicio 1

“Listar el nombre de todos los empleados”

FROM Empleado

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

# Ejercicio 1

“Listar el nombre de todos los empleados”

```
SELECT nombre  
FROM Empleado;
```

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

# Ejercicio 1

“Listar el nombre de todos los empleados”

```
SELECT nombre  
FROM Empleado;
```

## Resultado

nombre
Juan
Pedro
Daniel

# Ejercicio 2

“Listar el nombre y número de todos los empleados”

FROM Empleado

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

## Ejercicio 2

“Listar el nombre y número de todos los empleados”

```
SELECT nombre, nro  
FROM Empleado;
```

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

# Ejercicio 2

“Listar el nombre y número de todos los empleados”

```
SELECT nombre, nro  
FROM Empleado;
```

## Resultado

nombre	nro
Juan	1000
Pedro	1001
Daniel	1002

# Lista de campos (SELECT)

<campo1> [ [AS] Alias1 ], <campo2> [ [AS] Alias2 ], ... , <campoN> [ [AS] AliasN ]



## Ejercicio 2

“Listar el nombre y número de todos los empleados”

```
SELECT nombre, nro AS numero  
FROM Empleado;
```

### Resultado

nombre	numero
Juan	1000
Pedro	1001
Daniel	1002

# Ejercicio 3

“Listar todos los datos de todos los empleados”

FROM Empleado

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

# Ejercicio 3

“Listar todos los datos de todos los empleados”

```
SELECT nro, nombre, cod_esp, nro_jefe, sueldo, f_ingreso  
FROM Empleado;
```

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

# Ejercicio 3

“Listar todos los datos de todos los empleados”

```
SELECT *  
FROM Empleado;
```

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

# Ejercicio 3

“Listar todos los datos de todos los empleados”

```
SELECT *  
FROM Empleado;
```

## Resultado

nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

# Ejercicio 4

“Listar el nombre de los empleados, ordenados por sueldo”

FROM Empleado

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

# Ejercicio 4

“Listar el nombre de los empleados, ordenados por sueldo”

FROM Empleado  
ORDER BY sueldo

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1002	Daniel	2	1000	2000	1/10/2009
1001	Pedro	2	1000	5000	1/5/2008
1000	Juan	1		10000	1/1/2000

# Ejercicio 4

“Listar el nombre de los empleados, ordenados por sueldo”

```
SELECT nombre  
FROM Empleado  
ORDER BY sueldo;
```

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1002	Daniel	2	1000	2000	1/10/2009
1001	Pedro	2	1000	5000	1/5/2008
1000	Juan	1		10000	1/1/2000



# Ejercicio 4

“Listar el nombre de los empleados, ordenados por sueldo”

```
SELECT nombre  
FROM Empleado  
ORDER BY sueldo;
```

## Resultado

nombre
Daniel
Pedro
Juan

## Ejercicio 5

“Listar el nombre de los empleados ordenados por sueldo y, para mismo sueldo, ordenar por antigüedad, mostrando a los más nuevos primero”

# Lista de campos (ORDER BY)

<campo1> [ ASC | DESC ], <campo2> [ ASC | DESC ], ... , <campoN> [ ASC | DESC ]

## Ejercicio 5

“Listar el nombre de los empleados ordenados por sueldo y, para mismo sueldo, ordenar por antigüedad, mostrando a los más nuevos primero”

```
SELECT nombre  
FROM Empleado  
ORDER BY sueldo, f_ingreso DESC;
```

## Ejercicio 6

“Listar número y nombre de empleados con sueldo mayor a \$3000”

# Condición (WHERE)

- Comparaciones

- Literal  $\Leftrightarrow$  Variable (Campo)
- Variable (Campo)  $\Leftrightarrow$  Variable (Campo)
- ~~Literal  $\Leftrightarrow$  Literal~~

- Comparadores

- >
- <
- >=
- <=
- =
- <> o !=

# Condición (WHERE)

- Conectores
  - AND
  - OR
- Precedencia
  - Implícita: AND sobre OR
  - Explícita: dada por ( )

# Ejercicio 6

“Listar número y nombre de empleados con sueldo mayor a \$3000”

FROM Empleado

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009



# Ejercicio 6

“Listar número y nombre de empleados con sueldo mayor a \$3000”

FROM Empleado

WHERE sueldo > 3000

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

# Ejercicio 6

“Listar número y nombre de empleados con sueldo mayor a \$3000”

FROM Empleado

WHERE sueldo > 3000

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

# Ejercicio 6



“Listar número y nombre de empleados con sueldo mayor a \$3000”

FROM Empleado

WHERE sueldo > 3000

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009



# Ejercicio 6

“Listar número y nombre de empleados con sueldo mayor a \$3000”

FROM Empleado

WHERE sueldo > 3000

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008

# Ejercicio 6

“Listar número y nombre de empleados con sueldo mayor a \$3000”

```
SELECT nro, nombre  
FROM Empleado  
WHERE sueldo > 3000;
```

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008

# Ejercicio 6

“Listar número y nombre de empleados con sueldo mayor a \$3000”

```
SELECT nro, nombre  
FROM Empleado  
WHERE sueldo > 3000;
```

## Resultado

nro	nombre
1000	Juan
1001	Pedro

## Ejercicio 7

“Listar número y nombre de empleados con sueldo menor a \$8000 cuyo nombre sea Pedro”

```
SELECT nro, nombre  
FROM Empleado  
WHERE sueldo < 8000  
AND nombre = 'Pedro';
```

## Ejercicio 8

“Listar nombre de empleados que ingresaron en el año 2008”

```
SELECT nombre  
FROM Empleado  
WHERE f_ingreso >= '20080101'  
AND f_ingreso <= '20081231';
```



# Operador BETWEEN

<campo> [NOT] BETWEEN <valor\_desde> AND <valor\_hasta>

- Compara un intervalo de valores
- Incluye los extremos (intervalo cerrado)

## Ejercicio 8

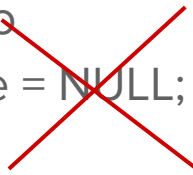
“Listar nombre de empleados que ingresaron en el año 2008”

```
SELECT nombre  
FROM Empleado  
WHERE f_ingreso BETWEEN '20080101' AND '20081231';
```

## Ejercicio 9

“Listar nombre de empleados sin jefe”

```
SELECT nombre  
FROM Empleado  
WHERE nro_jefe = NULL;
```



# Operador IS NULL

<campo> IS [NOT] NULL

## Ejercicio 9

“Listar nombre de empleados sin jefe”

```
SELECT nombre  
FROM Empleado  
WHERE nro_jefe IS NULL;
```

## Ejercicio 10

“Listar nombre y sueldo de empleados cuyo nombre comienza con la letra A”

# Operador LIKE

<campo> [NOT] LIKE <patron>

- El patrón es una cadena de caracteres que usa comodines
- Comodines
  - % (ninguno, uno o muchos caracteres)
  - \_ (uno y solo un caracter)
- Ejemplos de patrones comunes
  - 'A%' (Comienza con A) => Ana, Alejandro, A
  - '%s' (Termina con s) => Luis, Ines, s
  - '%ana%' (Contiene ana) => Banana, anastasia, ana
- Otros ejemplos
  - '\_s\_' => Asa, Osa, Isa
  - 'A\_C%' => ABCdef, ACCd, AbC

## Ejercicio 10

“Listar nombre y sueldo de empleados cuyo nombre comienza con la letra A”

```
SELECT nombre, sueldo  
FROM Empleado  
WHERE nombre LIKE 'A%';
```



# Ejercicio 11

“Listar nombre y sueldo de los empleados de número 1, 2, 5, 7 y 9”

```
SELECT nombre, sueldo  
FROM Empleado  
WHERE nro = 1  
OR nro = 2  
OR nro = 5  
OR nro = 7  
OR nro = 9;
```

# Operador IN

<campo> [NOT] IN (<lista\_valores>)

# Ejercicio 11

“Listar nombre y sueldo de los empleados de número 1, 2, 5, 7 y 9”

```
SELECT nombre, sueldo  
FROM Empleado  
WHERE nro IN (1, 2, 5, 7, 9);
```

# Ejercicio 12

“Listar el número de aquellos empleados que sean jefe”

FROM Empleado

**Tabla Temporal**

Empleado					
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

# Ejercicio 12

“Listar el número de aquellos empleados que sean jefe”

```
SELECT nro_jefe  
FROM Empleado  
WHERE nro_jefe IS NOT NULL;
```

## Resultado

nro_jefe
1000
1000

# Ejercicio 12

“Listar el número de aquellos empleados que sean jefe”

```
SELECT DISTINCT nro_jefe  
FROM Empleado  
WHERE nro_jefe IS NOT NULL;
```

## Resultado

nro_jefe
1000

# Base de Datos I

---

SQL - INSERT

# Sintaxis INSERT

INSERT INTO <tabla>

[ ( <lista\_campos> ) ]

VALUES ( <lista\_valores> );



# Ejemplo INSERT

“Insertar al empleado de número 5 llamado Héctor, quien ingresó el 01/01/2020 con un sueldo de \$5000 y especialidad de código 2”

```
INSERT INTO Empleado  
( nro, nombre, cod_esp, f_ingreso, sueldo )  
VALUES ( 5, 'Hector', 2, '20200101', 5000 );
```

# Sintaxis INSERT de múltiples registros

INSERT INTO <tabla>

[ ( <lista\_campos> ) ]

VALUES ( <lista\_valores1> ), ( <lista\_valores2> ), ... , ( <lista\_valoresN> );

# Base de Datos I

---

SQL - Práctica 1

# Ejercicio 1

“Listar los números de artículos cuyo precio se encuentre entre \$100 y \$1000 y su descripción comience con la letra A”

```
SELECT cod_art  
FROM articulo  
WHERE precio BETWEEN 100 AND 1000  
      AND descripcion LIKE 'A%';
```

## Ejercicio 2

“Listar todos los datos de todos los proveedores”

```
SELECT *  
FROM proveedor;
```

## Ejercicio 3

“Listar la descripción de los materiales de código 1, 3, 6, 9 y 18”

```
SELECT descripcion  
FROM material  
WHERE cod_mat IN (1,3,6,9,18);
```

## Ejercicio 4

“Listar código y nombre de proveedores de la calle Suipacha, que hayan sido dados de alta en el año 2001”

```
SELECT cod_prov, nombre  
FROM proveedor  
WHERE domicilio LIKE '%Suipacha%'  
      AND fecha_alta >= '2001-01-01'  
      AND fecha_alta < '2002-01-01';
```

# Base de Datos I

---

SQL - SELECT Multitabla



# Modelo de datos

**Empleado**

<u>nro</u>	nombre	<u>cod_esp</u>	<u>nro_jefe</u>	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

**Trabaja**

<u>nro_emp</u>	<u>cod_area</u>
1000	A1
1000	A2
1001	A1
1002	A2

**Area**

<u>cod_area</u>	descripcion
A1	Area 1
A2	Area 2

**Especialidad**

<u>cod_esp</u>	descripcion
1	Gerente
2	Operario

# Ejercicio 1

“Listar el nombre y descripción de especialidad de todos los empleados”

# FROM Empleado, Especialidad

## Tabla Temporal

[illegible]

# Ejercicio 1

“Listar el nombre y descripción de especialidad de todos los empleados”

FROM Empleado,  
Especialidad

**Tabla Temporal**

Producto  
Cartesiano

Empleado						Especialidad	
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso	cod_esp	descripcion
1000	Juan	1		10000	1/1/2000	1	Gerente
1001	Pedro	2	1000	5000	1/5/2008	2	Operario
1002	Daniel	2	1000	2000	1/10/2009	1	Gerente
1000	Juan	1		10000	1/1/2000	2	Operario
1001	Pedro	2	1000	5000	1/5/2008	1	Gerente
1002	Daniel	2	1000	2000	1/10/2009	2	Operario

# Junta implícita

Producto cartesiano en FROM +  
Condición de junta en WHERE

# Ejercicio 1

FROM Empleado,

Especialidad

WHERE cod\_esp = cod\_esp

**Tabla Temporal**

**Producto  
Cartesiano**

Empleado						Especialidad	
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso	cod_esp	descripcion
1000	Juan	1		10000	1/1/2000	1	Gerente
1001	Pedro	2	1000	5000	1/5/2008	2	Operario
1002	Daniel	2	1000	2000	1/10/2009	1	Gerente
1000	Juan	1		10000	1/1/2000	2	Operario
1001	Pedro	2	1000	5000	1/5/2008	1	Gerente
1002	Daniel	2	1000	2000	1/10/2009	2	Operario

# Ejercicio 1

```
SELECT Empleado.nombre, Especialidad.descripcion  
FROM Empleado,  
      Especialidad  
WHERE Empleado.cod_esp = Especialidad.cod_esp
```

**Tabla Temporal**

Producto  
Cartesiano

Empleado						Especialidad	
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso	cod_esp	descripcion
1000	Juan	1		10000	1/1/2000	1	Gerente
1001	Pedro	2	1000	5000	1/5/2008	2	Operario
1002	Daniel	2	1000	2000	1/10/2009	1	Gerente
1000	Juan	1		10000	1/1/2000	2	Operario
1001	Pedro	2	1000	5000	1/5/2008	1	Gerente
1002	Daniel	2	1000	2000	1/10/2009	2	Operario

# Lista de tablas en FROM

<tabla1> [ [AS] Alias1 ], <tabla2> [ [AS] Alias1 ], ... , <tablaN> [ [AS] AliasN ]

# Ejercicio 1

“Listar el nombre y descripción de especialidad de todos los empleados”

```
SELECT EM.nombre, ES.descripcion  
FROM Empleado EM,  
      Especialidad ES  
WHERE EM.cod_esp = ES.cod_esp;
```



# Junta explícita (JOIN)

Tipos de junta explícita más comunes:

- [ INNER ] JOIN
- LEFT [ OUTER ] JOIN

# INNER JOIN

<tabla1> [ INNER ] JOIN <tabla2> ON <condicion\_junta>

Resulta una tabla formada por la combinación de registros de tabla1 y tabla2 que cumplan con la condición de junta

# Ejercicio 1 (con junta explícita)

“Listar el nombre y descripción de especialidad de todos los empleados”

```
SELECT EM.nombre, ES.descripcion  
FROM Empleado EM JOIN  
    Especialidad ES ON M.cod_esp = ES.cod_esp;
```

## Ejercicio 2 (junta implícita)

“Listar el nombre de aquellos empleados que trabajan en el área de código A1”

```
SELECT EM.nombre  
FROM Empleado EM,  
      Trabaja T  
WHERE EM.nro = T.nro_emp  
AND T.cod_area = 'A1';
```

## Ejercicio 2 (junta explícita)

“Listar el nombre de aquellos empleados que trabajan en el área de código A1”

```
SELECT EM.nombre  
FROM Empleado EM JOIN  
      Trabaja T ON EM.nro = T.nro_emp  
WHERE T.cod_area = 'A1';
```

## Ejercicio 3

“Listar el nombre de todos los empleados junto al nombre de su jefe”

# Ejercicio 3

FROM Empleado EM,  
Empleado J

**Tabla Temporal**

Empleado (EM)					Empleado (J)				
nro	nombre	cod_esp	nro_jefe	...	nro	nombre	cod_esp	nro_jefe	...
1000	Juan	1			1000	Juan	1		
1001	Pedro	2	1000		1000	Juan	1		
1002	Daniel	2	1000		1000	Juan	1		
1000	Juan	1			1001	Pedro	2	1000	
1001	Pedro	2	1000		1001	Pedro	2	1000	
1002	Daniel	2	1000		1001	Pedro	2	1000	
1000	Juan	1			1002	Daniel	2	1000	
1001	Pedro	2	1000		1002	Daniel	2	1000	
1002	Daniel	2	1000		1002	Daniel	2	1000	

## Ejercicio 3

“Listar el nombre de todos los empleados junto al nombre de su jefe”

```
SELECT EM.nombre, J.nombre jefe  
FROM Empleado EM JOIN  
    Empleado J ON EM.nro_jefe = J.nro;
```



## Ejercicio 4

“Listar el nombre de todos sus empleados. Indicar además el nombre de su jefe (si es que tiene)”

# LEFT OUTER JOIN

```
<tabla1> LEFT [ OUTER ] JOIN <tabla2> ON <condicion_junta>
```

Resulta una tabla formada por la combinación de registros de tabla1 y tabla2 que cumplan con la condición de junta.

Si existe algún registro de tabla1 (izquierda) para el cual no se encontrara combinación alguna, se agrega el registro en la tabla resultante y se completa con valores nulos en los campos correspondientes a tabla2.

## Ejercicio 4

“Listar el nombre de todos los empleados. Indicar además el nombre de su jefe (si es que tiene)”

FROM Empleado EM LEFT JOIN

Empleado J ON EM.nro\_jefe = J.nro

**Tabla Temporal**

Empleado (EM)					Empleado (J)				
nro	nombre	cod_esp	nro_jefe	...	nro	nombre	cod_esp	nro_jefe	...
1001	Pedro	2	1000		1000	Juan	1		
1002	Daniel	2	1000		1000	Juan	1		
1000	Juan	1							

## Ejercicio 4

“Listar el nombre de todos los empleados. Indicar además el nombre de su jefe (si es que tiene)”

```
SELECT EM.nombre, J.nombre jefe  
FROM Empleado EM LEFT JOIN  
    Empleado J ON EM.nro_jefe = J.nro;
```

## Ejercicio 5

“Listar el nombre de los empleados que trabajan en el área de descripción Area 1 y que cobran más de \$5000”

```
SELECT EM.nombre  
FROM Empleado EM JOIN  
      Trabaja T ON EM.nro = T.nro_emp JOIN  
      Area A ON A.cod_area = T.cod_area  
WHERE A.descripcion = 'Area 1'  
AND EM.sueldo > 5000;
```

# Ejercicio 5

- Problema 1: qué sucede si hay 2 áreas con misma descripción?
  - Solución: uso de DISTINCT
- Problema 2: si usamos DISTINCT, qué sucede si dos empleados se llaman igual?
  - Solución: traer además un campo para diferenciar los registros (ej: nro. de empleado), aunque esto no está pedido en el problema

# Otras variantes de junta explícita

- RIGHT [ OUTER ] JOIN
  - Igual al LEFT JOIN pero se invierte el orden de las tablas
- NATURAL JOIN
  - No requiere condición de junta (ON) y combina mediante los campos de mismo nombre

# Base de Datos I

---

SQL - Práctica SELECT Multitablas



## Ejercicio 5

“Listar nombre de todos los proveedores y de su ciudad”

```
SELECT p.nombre, c.nombre as "Ciudad"  
FROM proveedor p LEFT JOIN ciudad c ON p.cod_ciu = c.cod_ciu;
```

## Ejercicio 6

“Listar los nombres de los proveedores de la ciudad de La Plata”

```
SELECT p.nombre  
FROM proveedor p JOIN ciudad c ON p.cod_ciu = c.cod_ciu  
WHERE c.nombre = 'La Plata';
```

## Ejercicio 7

“Listar los números de almacenes que almacenan el artículo de descripción A”

```
SELECT DISTINCT tie.nro  
  FROM contiene tie JOIN articulo art on (tie.cod_art = art.cod_art)  
 WHERE art.descripcion = 'A';
```

## Ejercicio 8

“Listar los materiales (código y descripción) provistos por proveedores de la ciudad de Rosario”

```
SELECT DISTINCT m.cod_mat, m.descripcion
  FROM material m JOIN provisto_por pp ON m.cod_mat = pp.cod_Mat
    JOIN proveedor p ON pp.cod_prov = p.cod_prov
    JOIN ciudad c ON p.cod_ciu = c.cod_ciu
WHERE c.nombre = 'Rosario';
```

## Ejercicio 9

“Listar los nombres de los proveedores que proveen materiales para artículos ubicados en almacenes que Martín Gómez tiene a su cargo”

```
SELECT DISTINCT pro.nombre
FROM proveedor pro JOIN provisto_por pp ON pro.cod_prov = pp.cod_prov
                JOIN compuesto_por cp ON cp.cod_mat = pp.cod_mat
                JOIN contiene tie ON tie.cod_art = cp.cod_art
                JOIN almacen alm ON alm.nro = tie.nro
WHERE alm.responsable = 'Martín Gómez';
```

# Base de Datos I

---

SQL - SELECT Agregación

# Modelo de datos

**Empleado**

<u>nro</u>	nombre	<u>cod_esp</u>	<u>nro_jefe</u>	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

**Trabaja**

<u>nro_emp</u>	<u>cod_area</u>
1000	A1
1000	A2
1001	A1
1002	A2

**Area**

<u>cod_area</u>	descripcion
A1	Area 1
A2	Area 2

**Especialidad**

<u>cod_esp</u>	descripcion
1	Gerente
2	Operario

# Ejercicio 1

“Indicar la cantidad de empleados de la empresa”



# Funciones de agregación

- COUNT (\*)
  - COUNT (<campo>)
  - COUNT (DISTINCT <campo>)
- MAX(<campo>)
- MIN(<campo>)
- SUM(<campo>)
- AVG(<campo>)

# Proceso de agrupamiento

Temporal



Agrupamiento

Temporal Agrupada



- Un registro por **grupo**
- Aplicar funciones de agregación

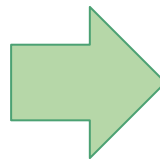
# Ejercicio 1

“Indicar la cantidad de empleados de la empresa”

```
SELECT COUNT(*)  
FROM Empleado;
```

Temporal

nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009



Temporal Agrupada

COUNT(*)
3

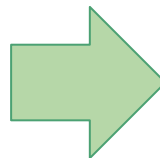
## Ejercicio 2

“Indicar la cantidad de empleados y sueldo máximo de la empresa”

```
SELECT COUNT(*), MAX(sueldo)
FROM Empleado;
```

Temporal

nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009



Temporal Agrupada

COUNT(*)	MAX(sueldo)
3	10000

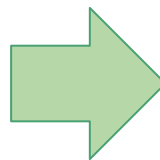
# Ejercicio 3

“Cuántos empleados ganan más de \$3000?”

```
SELECT COUNT(*)  
FROM Empleado  
WHERE sueldo > 3000;
```

Temporal

nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008



Temporal Agrupada

COUNT(*)
2

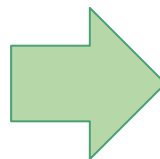
# Ejercicio 4

“Indicar el sueldo mínimo de los empleados por cada código de especialidad”

```
SELECT cod_esp, MIN(sueldo)
FROM Empleado;
```

Temporal

nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009



Temporal Agrupada

cod_esp	MIN(sueldo)
?	2000

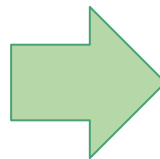
## Ejercicio 4

“Indicar el sueldo mínimo de los empleados por cada código de especialidad”

FROM Empleado  
GROUP BY cod\_esp;

Temporal

nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009



Temporal Agrupada

cod_esp	
1	
2	

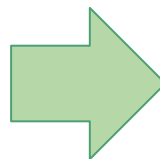
## Ejercicio 4

“Indicar el sueldo mínimo de los empleados por cada código de especialidad”

```
SELECT cod_esp, MIN(sueldo)
FROM Empleado
GROUP BY cod_esp;
```

Temporal

nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009



Temporal Agrupada

cod_esp	MIN(sueldo)
1	10000
2	2000



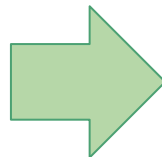
# Ejercicio 5

“Indicar el sueldo mínimo de los empleados por cada código de especialidad, sólo para aquellas especialidades cuyo mínimo sea mayor a 3000”

```
SELECT cod_esp, MIN(sueldo)
FROM Empleado
WHERE MIN(sueldo) > 3000
GROUP BY cod_esp;
```

Temporal

nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009



Temporal Agrupada

cod_esp	MIN(sueldo)
1	10000
2	2000

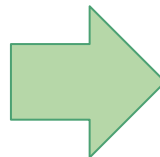
# Ejercicio 5

“Indicar el sueldo mínimo de los empleados por cada código de especialidad, sólo para aquellas especialidades cuyo mínimo sea mayor a 3000”

```
SELECT cod_esp, MIN(sueldo)
FROM Empleado
GROUP BY cod_esp
HAVING MIN(sueldo) > 3000;
```

Temporal

nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009



Temporal Agrupada

cod_esp	MIN(sueldo)
1	10000
2	2000

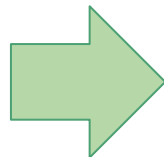
# Ejercicio 5

“Indicar el sueldo mínimo de los empleados por cada código de especialidad, sólo para aquellas especialidades cuyo mínimo sea mayor a 3000”

```
SELECT cod_esp, MIN(sueldo)
FROM Empleado
GROUP BY cod_esp
HAVING MIN(sueldo) > 3000;
```

Temporal

nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009



Temporal Agrupada

cod_esp	MIN(sueldo)
1	10000

## Ejercicio 6

“Indicar el sueldo mínimo de los empleados por cada código de especialidad, sólo para aquellas especialidades con más de 5 empleados”

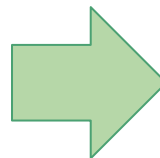
```
SELECT cod_esp, MIN(sueldo)
FROM Empleado
GROUP BY cod_esp
HAVING COUNT(*) > 5;
```

# Ejercicio 7

“Indicar cantidad de empleados por cada descripción de especialidad”

```
SELECT ES.descripcion, COUNT(*)  
FROM Empleado EM JOIN  
    Especialidad ES ON EM.cod_esp = ES.cod_esp  
GROUP BY ES.descripcion;
```

Empleado			Temporal			Especialidad	
nro	nombre	cod_esp	nro_jefe	suelo	f_ingreso	cod_esp	descripcion
1000	Juan	1		10000	1/1/2000	1	Gerente
1001	Pedro	2	1000	5000	1/5/2008	2	Operario
1002	Daniel	2	1000	2000	1/10/2009	2	Operario



Temporal Agrupada	
descripcion	COUNT(*)
Gerente	1
Operario	2

# Ejercicio 7

Problemas:

- Que sucede si dos o más especialidades tienen la misma descripción?

# Ejercicio 7

“Indicar cantidad de empleados por cada descripción de especialidad”

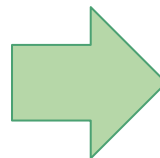
```
SELECT ES.descripcion, COUNT(*)
```

```
FROM Empleado EM JOIN
```

```
    Especialidad ES ON EM.cod_esp = ES.cod_esp
```

```
GROUP BY ES.cod_esp;
```

Empleado			Temporal			Especialidad	
nro	nombre	cod_esp	nro_jefe	suelo	f_ingreso	cod_esp	descripcion
1000	Juan	1		10000	1/1/2000	1	Gerente
1001	Pedro	2	1000	5000	1/5/2008	2	Operario
1002	Daniel	2	1000	2000	1/10/2009	2	Operario



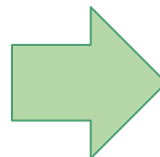
Temporal Agrupada	
cod_esp	COUNT(*)
1	1
2	2

# Ejercicio 7

“Indicar cantidad de empleados por cada descripción de especialidad”

```
SELECT ES.descripcion, COUNT(*)  
FROM Empleado EM JOIN  
    Especialidad ES ON EM.cod_esp = ES.cod_esp  
GROUP BY ES.cod_esp, ES.descripcion;
```

Empleado			Temporal			Especialidad	
nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso	cod_esp	descripcion
1000	Juan	1		10000	1/1/2000	1	Gerente
1001	Pedro	2	1000	5000	1/5/2008	2	Operario
1002	Daniel	2	1000	2000	1/10/2009	2	Operario



Temporal Agrupada		
cod_esp	descripcion	COUNT(*)
1	Gerente	1
2	Operario	2



# Ejercicio 7

Solución:

- Normalmente agruparemos por campos que sean claves
- Sólo podremos devolver en la cláusula `SELECT` aquellos campos por los cuales hemos agrupado, o funciones de agregación

## Ejercicio 8

“Indicar cantidad de empleados a cargo de cada número de jefe”

```
SELECT nro_jefe, COUNT(*)  
FROM Empleado  
WHERE nro_jefe IS NOT NULL  
GROUP BY nro_jefe;
```

## Ejercicio 9

“Indicar cantidad de empleados a cargo de cada jefe, mostrando su nombre”

```
SELECT J.nombre, COUNT(*)  
FROM Empleado EM JOIN  
      Empleado J ON EM.nro_jefe = J.nro  
GROUP BY J.nro, J.nombre;
```

# Ejercicio 10

“Cuántos jefes hay?”

Empleado

nro	nombre	cod_esp	nro_jefe	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009
1003	Alejandro	2	1001	1500	1/1/2010
1004	Rafael	2	1003	1500	1/1/2010
1005	Leandro	2	1001	1500	1/1/2010
1006	Marcos	2	1001	1500	1/1/2010

# Variantes de COUNT

- COUNT (\*): cantidad de registros del grupo
- COUNT (<campo>): cantidad de registros del grupo, donde el campo indicado no sea NULL
- COUNT (DISTINCT <campo>): cantidad de registros **únicos** del grupo, donde el campo indicado no sea NULL (no cuenta repetidos)

# Ejercicio 10

“Cuántos jefes hay?”

```
SELECT COUNT(DISTINCT nro_jefe)  
FROM Empleado;
```

# Base de Datos I

---

SQL - Práctica Funciones de Agregación

## Ejercicio 12

“Indicar la cantidad de proveedores que comienzan con la letra F”

```
SELECT count(*) as 'cantidad de proveedores'  
FROM proveedor  
WHERE nombre LIKE 'F%';
```



## Ejercicio 13

“Listar el promedio de precios de los artículos por cada almacén (nombre)”

```
SELECT alm.nombre as 'almacen',  
       AVG(art.precio) as 'precio promedio'  
FROM almacen alm JOIN contiene c ON alm.nro = c.nro  
                JOIN Articulo art ON art.cod_art = c.cod_art  
GROUP BY alm.nro, alm.nombre;
```

## Ejercicio 14

“Listar la descripción de artículos compuestos por al menos 2 materiales”

```
SELECT a.descripcion
      FROM articulo a JOIN compuesto_por c ON a.cod_art = c.cod_art
GROUP BY a.cod_art,a.descripcion
HAVING COUNT(*) >= 2;
```

# Ejercicio 15

“Listar cantidad de materiales que provee cada proveedor (código, nombre y domicilio)”

```
SELECT p.cod_prov,  
       p.nombre,  
       p.domicilio,  
       COUNT(pp.cod_mat) AS 'cantidad de materiales'  
FROM proveedor p LEFT JOIN provisto_por pp ON p.cod_prov = pp.cod_prov  
GROUP BY p.cod_prov, p.nombre, p.domicilio;
```

## Ejercicio 16

“Cuál es el precio máximo de los artículos que proveen los proveedores de la ciudad de Zárate”

```
SELECT max(a.precio)
FROM proveedor p JOIN provisto_por pp ON p.cod_prov = pp.cod_prov
              JOIN compuesto_por cp ON pp.cod_mat = cp.cod_mat
              JOIN articulo a ON cp.cod_art = a.cod_art
              JOIN ciudad c ON c.cod_ciu = p.cod_ciu
WHERE c.nombre = 'Zarate';
```

## Ejercicio 17

“Listar los nombres de aquellos proveedores que no proveen ningún material”

```
SELECT p.nombre  
FROM proveedor p LEFT JOIN provisto_por pp ON p.cod_prov = pp.cod_prov  
WHERE pp.cod_mat IS NULL;
```

```
SELECT p.nombre  
FROM proveedor p LEFT JOIN provisto_por pp ON p.cod_prov = pp.cod_prov  
GROUP BY p.cod_prov, p.nombre  
HAVING COUNT(pp.cod_mat) = 0;
```

\*resolver este mismo ejercicio de otra forma la siguiente clase

# Base de Datos I

---

SQL - INSERT / UPDATE / DELETE

# Sintaxis INSERT

INSERT INTO <tabla>

[ ( <lista\_campos> ) ]

VALUES ( <lista\_valores> );

# Ejemplo INSERT

“Insertar al empleado de número 5 llamado Héctor, quien ingresó el 01/01/2020 con un sueldo de \$5000 y especialidad de código 2”

```
INSERT INTO Empleado  
( nro, nombre, cod_esp, f_ingreso, sueldo )  
VALUES ( 5, 'Hector', 2, '20200101', 5000 );
```



# Sintaxis INSERT de múltiples registros

INSERT INTO <tabla>

[ ( <lista\_campos> ) ]

VALUES ( <lista\_valores1> ), ( <lista\_valores2> ), ... , ( <lista\_valoresN> );

# Sintaxis UPDATE

```
UPDATE <tabla>  
SET <campo1> = <valor1>,  
    <campo2> = <valor2>,  
    ...  
    <campoN> = <valorN>,  
[ WHERE <condicion> ];
```

# Ejemplo UPDATE

“Al empleado de número 5 deberá modificarse su especialidad a la de código 3”

```
UPDATE Empleado  
SET cod_esp = 3  
WHERE nro = 5;
```

# Ejemplo UPDATE

“Incrementar \$1000 el sueldo de todos los empleados”

UPDATE Empleado

SET sueldo = sueldo + 1000;

# Sintaxis DELETE

DELETE FROM <tabla>  
[ WHERE <condicion> ];



\*Ref: [https://www.youtube.com/watch?v=i\\_cVJglz\\_Cs](https://www.youtube.com/watch?v=i_cVJglz_Cs)

# Ejemplo DELETE

“Eliminar todos los empleados cuyo nombre comienza con A”

```
DELETE FROM Empleado  
WHERE nombre LIKE 'A%';
```

# Ejemplo DELETE

“Eliminar todos los empleados”

```
DELETE FROM Empleado;
```

# Sintaxis TRUNCATE

TRUNCATE [TABLE] <tabla>;



# Ejemplo TRUNCATE

“Eliminar todos los empleados”

```
TRUNCATE TABLE Empleado;
```

# Base de Datos I

---

SQL - Subconsultas / Conjuntos

# Modelo de datos

**Empleado**

<u>nro</u>	nombre	<u>cod_esp</u>	<u>nro_jefe</u>	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

**Trabaja**

<u>nro_emp</u>	<u>cod_area</u>
1000	A1
1000	A2
1001	A1
1002	A2

**Area**

<u>cod_area</u>	descripcion
A1	Area 1
A2	Area 2

**Especialidad**

<u>cod_esp</u>	descripcion
1	Gerente
2	Operario

# Ejercicio 1

“Listar los nombres de los empleados que trabajan en algún área que termina con la letra S”

```
SELECT EM.nombre  
FROM Empleado EM JOIN  
      Trabaja T ON T.nro_emp = EM.nro JOIN  
      Area A ON T.cod_area = A.cod_area  
WHERE A.descripcion LIKE '%S';
```

*Problema: qué sucede si un empleado trabaja en 2 o más áreas que terminan con la letra S?*

# Ejercicio 1

“Listar los nombres de los empleados que trabajan en algún área que termina con la letra S”

```
SELECT DISTINCT EM.nombre  
FROM Empleado EM JOIN  
    Trabaja T ON T.nro_emp = EM.nro JOIN  
    Area A ON T.cod_area = A.cod_area  
WHERE A.descripcion LIKE '%S';
```

*Problema: qué sucede si 2 o más empleados que se llaman igual trabajan en áreas que terminan con la letra S?*

# Ejercicio 1

“Listar los nombres de los empleados que trabajan en algún área que termina con la letra S”

```
SELECT EM.nombre  
FROM Empleado EM JOIN  
      Trabaja T ON T.nro_emp = EM.nro JOIN  
      Area A ON T.cod_area = A.cod_area  
WHERE A.descripcion LIKE '%S'  
GROUP BY EM.nro, EM.nombre;
```

# Ejercicio 1

“Listar los nombres de los empleados que trabajan en algún área que termina con la letra S”

```
SELECT EM.nombre  
FROM Empleado EM  
WHERE EM.nro IN ( <nros_empleados_de_areas_terminan_s> );
```

# Ejercicio 1.1

“Listar los números de los empleados que trabajan en algún área que termina con la letra S”

```
SELECT T.nro_emp  
FROM Trabaja T JOIN  
      Area A ON T.cod_area = A.cod_area  
WHERE A.descripcion LIKE '%S';
```

<u>nro</u>
1005
1006
1007



# Ejercicio 1

“Listar los nombres de los empleados que trabajan en algún área que termina con la letra S”

```
SELECT EM.nombre  
FROM Empleado EM  
WHERE EM.nro IN (
```

<u>nro</u>
1005
1006
1007

```
);
```

# Ejercicio 1

“Listar los nombres de los empleados que trabajan en algún área que termina con la letra S”

```
SELECT EM.nombre  
FROM Empleado EM  
WHERE EM.nro IN ( SELECT T.nro_emp  
                  FROM Trabaja T JOIN  
                  Area A ON T.cod_area = A.cod_area  
                  WHERE A.descripcion LIKE '%S' );
```

# IN + Subconsulta

- Debe devolver un solo campo
- El tipo de dato del campo a devolver debe ser compatible con el tipo del campo que se está comparando
- Puede retornar 0, 1 o muchos registros

# Predicado EXISTS

EXISTS ( <subconsulta> )

- Verdadero: la subconsulta retorna algún registro
- Falso: la subconsulta no retorna registro alguno (tabla vacía)
- Puede negarse la lógica mediante NOT EXISTS

# Ejercicio 1 (EXISTS)

“Listar los nombres de los empleados que trabajan en algún área que termina con la letra S”

```
SELECT EM.nombre
FROM Empleado EM
WHERE EXISTS ( SELECT T.nro_emp
                FROM Trabaja T JOIN
                Area A ON T.cod_area = A.cod_area
                WHERE A.descripcion LIKE '%S'
                AND EM.nro = T.nro_emp );
```

# Ejercicio 1 (EXISTS)

“Listar los nombres de los empleados que trabajan en algún área que termina con la letra S”

```
SELECT EM.nombre
FROM Empleado EM
WHERE EXISTS ( SELECT 1
                FROM Trabaja T JOIN
                  Area A ON T.cod_area = A.cod_area
                WHERE A.descripcion LIKE '%S'
                AND EM.nro = T.nro_emp );
```

## Ejercicio 2

“Listar los nombres de los empleados que ganan el sueldo máximo”

## Ejercicio 2

“Listar los nombres de los empleados que ganan el sueldo máximo”



## Ejercicio 2

“Listar los nombres de los empleados que ganan el sueldo máximo”

```
SELECT EM.nombre  
FROM Empleado EM  
WHERE sueldo = <sueldo_maximo>;
```

## Ejercicio 2.1

“Cuál es el sueldo máximo?”

```
SELECT MAX(EM2.sueldo)  
FROM Empleado EM2;
```

<u>max</u>
10000

## Ejercicio 2

“Listar los nombres de los empleados que ganan el sueldo máximo”

SELECT EM.nombre

FROM Empleado EM

WHERE sueldo = 

<u>max</u>
10000

 ;

## Ejercicio 2

“Listar los nombres de los empleados que ganan el sueldo máximo”

```
SELECT EM.nombre  
FROM Empleado EM  
WHERE sueldo = ( SELECT MAX(EM2.sueldo)  
                  FROM Empleado EM2 ) ;
```

# Subconsulta como valor

- Debe devolver un solo campo
- El tipo de dato del campo a devolver debe ser compatible con el tipo del campo que se está comparando
- Debe retornar si o si un único registro

## Ejercicio 3

“Indicar la descripción de aquellas áreas sin empleados asignados”

```
SELECT A.descripcion  
FROM Area A  
WHERE NOT EXISTS ( SELECT 1  
                    FROM Trabaja T  
                    WHERE T.cod_area = A.cod_area);
```

## Ejercicio 3

“Indicar la descripción de aquellas áreas sin empleados asignados”

```
SELECT A.descripcion  
FROM Area A  
WHERE A.cod_area NOT IN ( SELECT T.cod_area  
                           FROM Trabaja T);
```

## Ejercicio 4 (División)

“Listar el nombre de los empleados que trabajan en todas las áreas de la empresa”

```
SELECT EM.nombre
FROM Empleado EM JOIN
      Trabaja T ON T.nro_emp = EM.nro
GROUP BY EM.nro, EM.nombre
HAVING COUNT(*) = ( SELECT COUNT(*)
                    FROM Area);
```



## Ejercicio 4 (División)

“Listar el nombre de los empleados que trabajan en todas las áreas de la empresa”

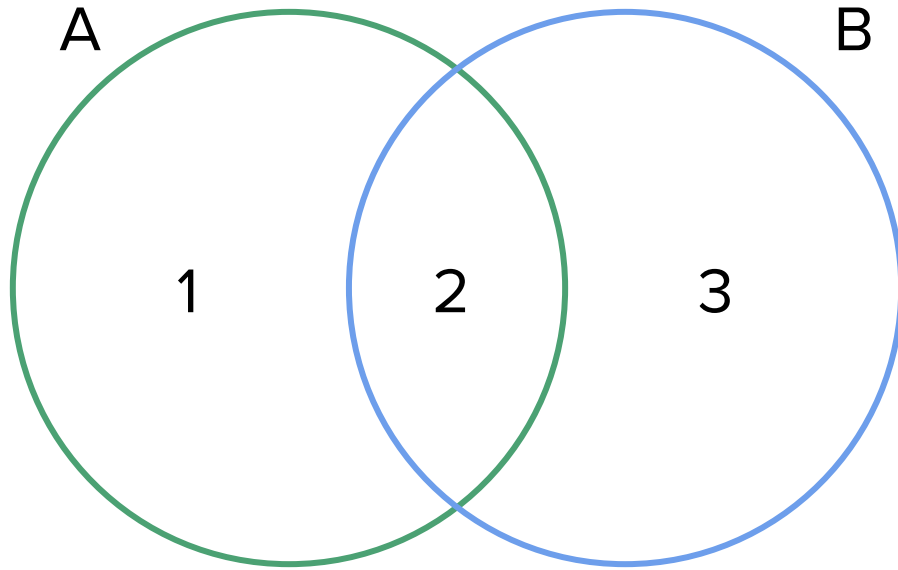
```
SELECT EM.nombre
FROM Empleado EM
WHERE NOT EXISTS ( SELECT 1
                    FROM Area A
                    WHERE NOT EXISTS ( SELECT 1
                                      FROM Trabaja T
                                      WHERE T.nro_emp = EM.nro
                                      AND T.cod_area = A.cod_area)
                  );
```

## Ejercicio 4 (División)

“Listar el nombre de los **empleados** que trabajan en todas las **áreas** de la empresa”

```
SELECT EM.nombre
FROM Empleado EM
WHERE NOT EXISTS ( SELECT 1
                    FROM Area A
                    WHERE NOT EXISTS ( SELECT 1
                                      FROM Trabaja T
                                      WHERE T.nro_emp = EM.nro
                                      AND T.cod_area = A.cod_area)
                  );
```

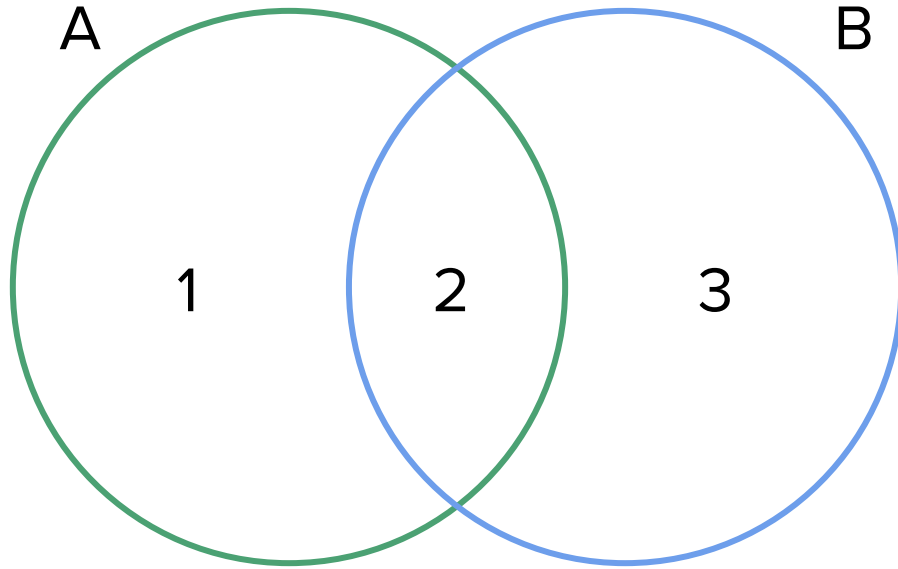
# Operaciones de conjuntos



$$A = \{ 1, 2 \}$$

$$B = \{ 2, 3 \}$$

# UNION

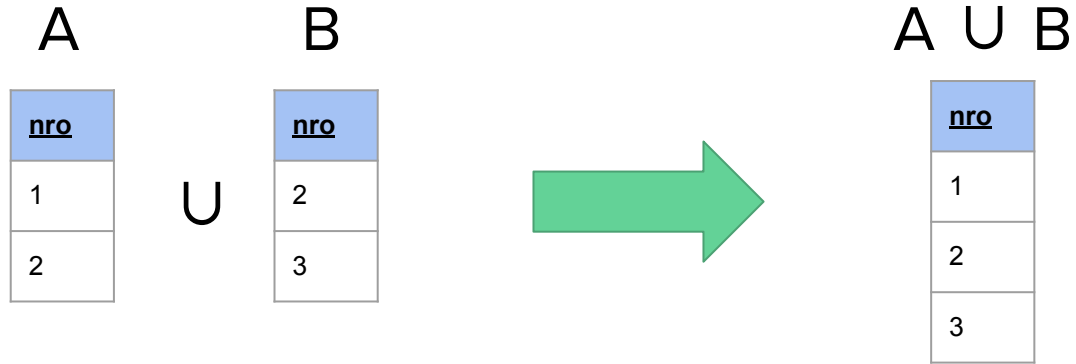


$$A = \{ 1, 2 \}$$

$$B = \{ 2, 3 \}$$

$$A \cup B = \{ 1, 2, 3 \}$$

# UNION



# UNION

```
SELECT A.nro  
FROM A  
UNION  
SELECT B.nro  
FROM B;
```



A U B

<u>nro</u>
1
2
3

# UNION ALL

```
SELECT A.nro  
FROM A  
UNION ALL  
SELECT B.nro  
FROM B;
```



$A \underline{\cup} B$

<u>nro</u>
1
2
2
3

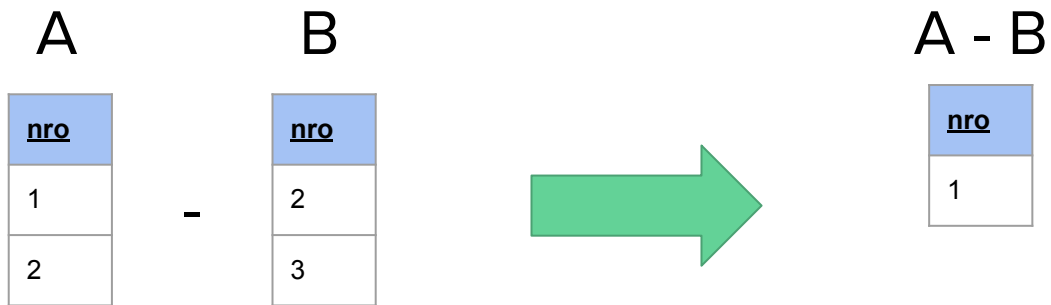
# INTERSECTS



*No soportado por MySQL*



# MINUS



*No soportado por MySQL*

# Base de Datos I

---

SQL - Práctica Subconsultas

## Ejercicio 17

“Listar los nombres de aquellos proveedores que no proveen ningún material”

```
SELECT p.nombre  
  FROM proveedor p  
 WHERE not exists (SELECT 1  
                   FROM provisto_por pp  
                   WHERE pp.cod_prov = p.cod_prov)
```

## Ejercicio 18

“Listar los códigos de los materiales que provea el proveedor 10 y no los provea el proveedor 15”

```
SELECT m.cod_mat
FROM material m
WHERE EXISTS (SELECT 1
              FROM provisto_por pp
              WHERE m.cod_mat = pp.cod_mat
                  AND pp.cod_prov = 10)
AND NOT EXISTS (SELECT 1
               FROM provisto_por pp2
               WHERE m.cod_mat = pp2.cod_mat
                   AND pp2.cod_prov = 15);
```



# Ejercicio 19

“Listar número y nombre de almacenes que contienen los artículos de descripción A y los de descripción B (ambos)”

```
SELECT a.nro,a.nombre
FROM almacen a
WHERE exists (SELECT 1
              FROM contiene c JOIN articulo art ON c.cod_art = art.cod_art
              WHERE a.nro = c.nro
              AND art.descripcion = 'A')
AND exists (SELECT 1
            FROM contiene c JOIN articulo art2 ON c.cod_art = art2.cod_art
            WHERE a.nro = c.nro
            AND art2.descripcion = 'B');
```

## Ejercicio 20

“Listar la descripción de artículos compuestos por todos los materiales.”

[illegible]

# Ejercicio 21

“Hallar los códigos y nombres de los proveedores que proveen al menos un material que se usa en algún artículo cuyo precio es mayor a \$100”

```
SELECT p.cod_prov, p.nombre
FROM proveedor p
WHERE exists (SELECT 1
              FROM provisto_por pp JOIN compuesto_por cp ON pp.cod_mat = cp.cod_mat
              JOIN articulo a ON cp.cod_art = a.cod_art
              WHERE a.precio > 100
              AND pp.cod_prov = p.cod_prov);
```



## Ejercicio 22

“Listar la descripción de los artículos de mayor precio”

```
SELECT a.descripcion  
FROM articulo a  
WHERE precio = (SELECT max(precio) FROM articulo b);
```

# Base de Datos I

---

SQL - Vistas / Transacciones

# Vistas

---

# Modelo de datos

**Empleado**

<u>nro</u>	nombre	<u>cod_esp</u>	<u>nro_jefe</u>	sueldo	f_ingreso
1000	Juan	1		10000	1/1/2000
1001	Pedro	2	1000	5000	1/5/2008
1002	Daniel	2	1000	2000	1/10/2009

**Trabaja**

<u>nro_emp</u>	<u>cod_area</u>
1000	A1
1000	A2
1001	A1
1002	A2

**Area**

<u>cod_area</u>	descripcion
A1	Area 1
A2	Area 2

**Especialidad**

<u>cod_esp</u>	descripcion
1	Gerente
2	Operario

# Sintaxis VIEW

```
CREATE [OR REPLACE] VIEW <nombre_vista> AS  
<SELECT...>;
```

# Ejemplo 1

```
CREATE VIEW EmpleadoSinJefe AS  
SELECT *  
FROM Empleado  
WHERE nro_jefe IS NULL;
```

```
SELECT nombre  
FROM EmpleadoSinJefe;
```

## Ejemplo 2

```
CREATE VIEW EmpleadoConSuEspecialidad AS  
SELECT EM.nombre, EM.sueldo, ES.descripcion especialidad  
FROM Empleado EM JOIN  
    Especialidad ES ON EM.cod_esp = ES.cod_esp;
```

```
SELECT nombre, especialidad  
FROM EmpleadoConSuEspecialidad;
```

```
SELECT f_ingreso  
FROM EmpleadoConSuEspecialidad;
```

# Características de vistas

- El objeto se crea por única vez (o se actualiza)
- El resultado se evalúa en cada uso
  - Algunas bases de datos permiten almacenar temporalmente el resultado (cache)
- Se utiliza únicamente en operaciones de lectura
  - Algunas bases de datos permiten sobrescribir el comportamiento de escritura
- Se ven únicamente los resultados que la consulta de la vista indica
  - Filas
  - Columnas (alias)
- Usos posibles
  - Simplificar consultas de uso común
  - Protección/ocultamiento de datos (filas/columnas)



# Transacciones

---

# Modelo de datos

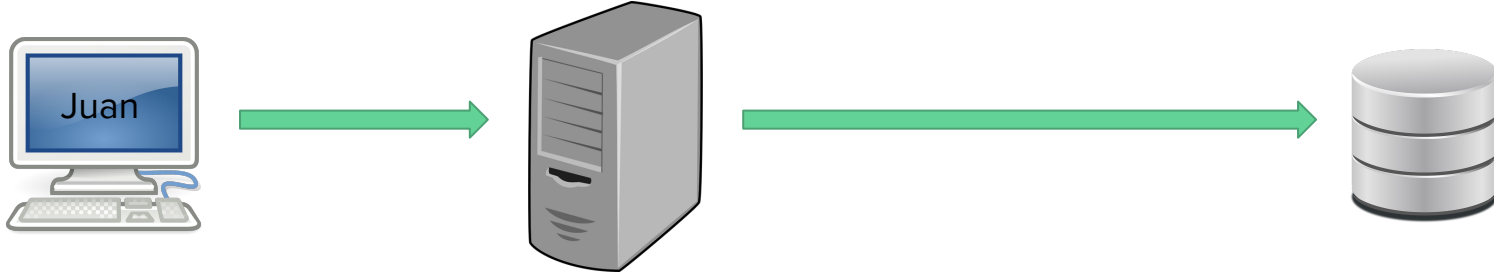
Cuenta

<u>nro</u>	titular	saldo
1000	Juan	1000
1001	Pedro	0
1002	Daniel	500

Transferencia

<u>id</u>	fecha	<u>nro_orig</u>	<u>nro_dest</u>	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500

# Transferencia bancaria



“Transferir \$200 a Pedro”

**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	1000
1001	Pedro	0
1002	Daniel	500

**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500

# Transferencia bancaria



“Transferir \$200 a Pedro”



```
UPDATE Cuenta  
SET saldo = saldo - 200  
WHERE nro = 1000;
```



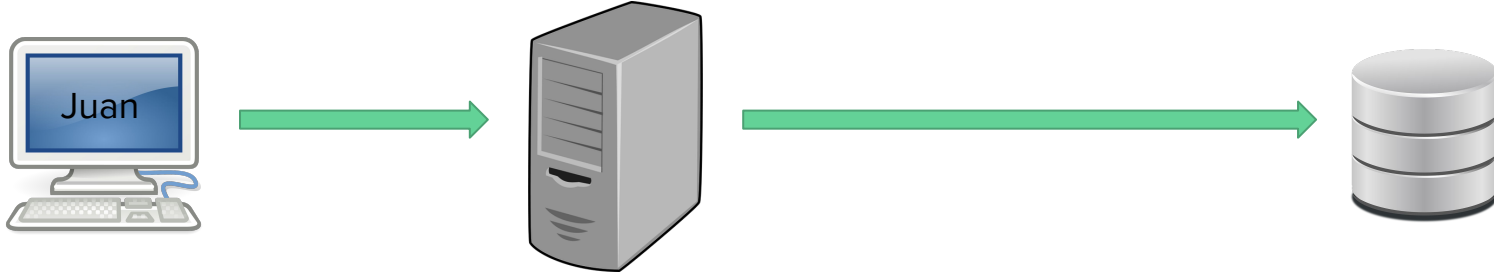
**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	800
1001	Pedro	0
1002	Daniel	500

**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500

# Transferencia bancaria



“Transferir \$200 a Pedro”

```
UPDATE Cuenta
SET saldo = saldo - 200
WHERE nro = 1000;
```

```
UPDATE Cuenta
SET saldo = saldo + 200
WHERE nro = 1001;
```

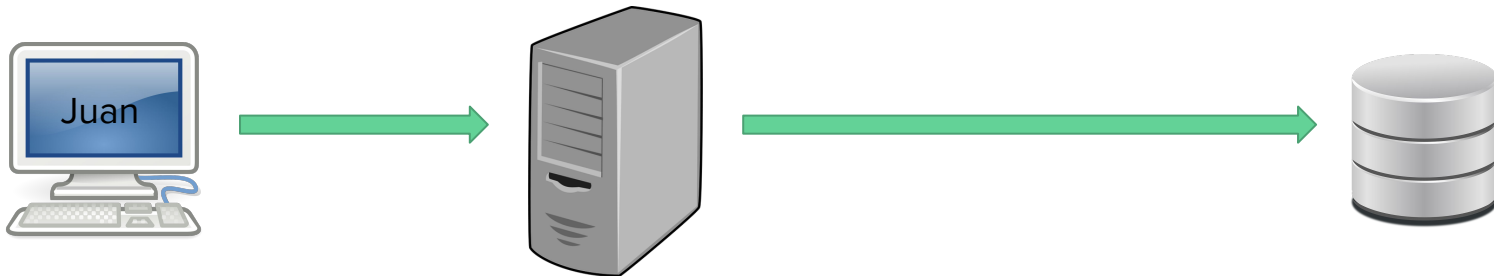
**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	800
1001	Pedro	<b>200</b>
1002	Daniel	500

**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500

# Transferencia bancaria



“Transferir \$200 a Pedro”

```
UPDATE Cuenta
SET saldo = saldo - 200
WHERE nro = 1000;
```

```
UPDATE Cuenta
SET saldo = saldo + 200
WHERE nro = 1001;
```

```
INSERT INTO Transferencia
(id, fecha, nro_orig, nro_dest, importe)
VALUES ( 3, '20200105', 1000, 1001, 200);
```

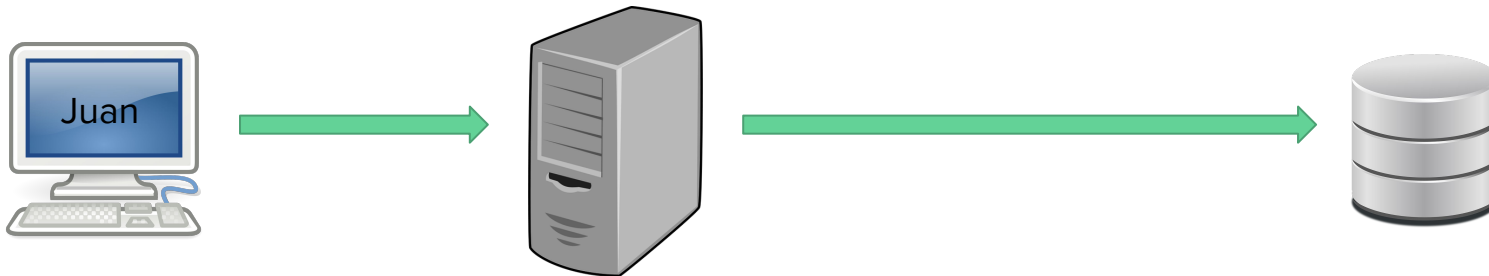
**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	800
1001	Pedro	200
1002	Daniel	500

**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200

# Transferencia bancaria



“Transferir \$200 a Pedro”



```
UPDATE Cuenta
SET saldo = saldo - 200
WHERE nro = 1000;
```

```
UPDATE Cuenta
SET saldo = saldo + 200
WHERE nro = 1001;
```

```
INSERT INTO Transferencia
(id, fecha, nro_orig, nro_dest, importe)
VALUES ( 3, '20200105', 1000, 1001, 200);
```

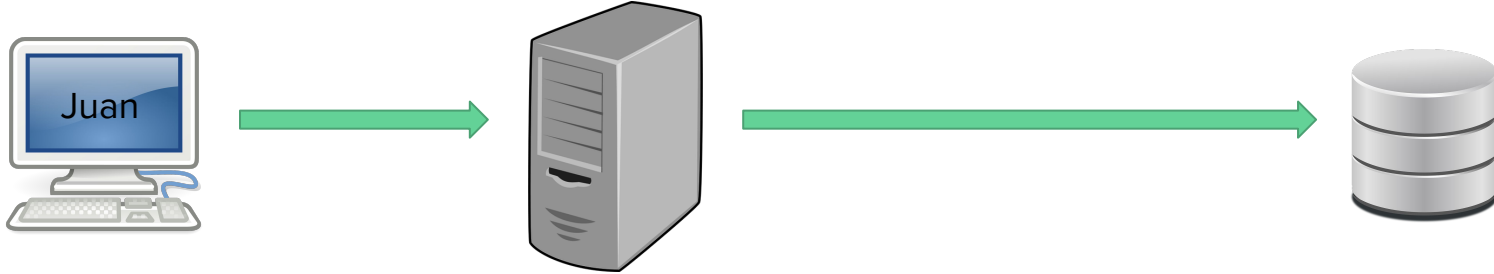
**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	800
1001	Pedro	200
1002	Daniel	500

**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200

# Transferencia bancaria 2



“Transferir \$200 a Daniel”

**Cuenta**

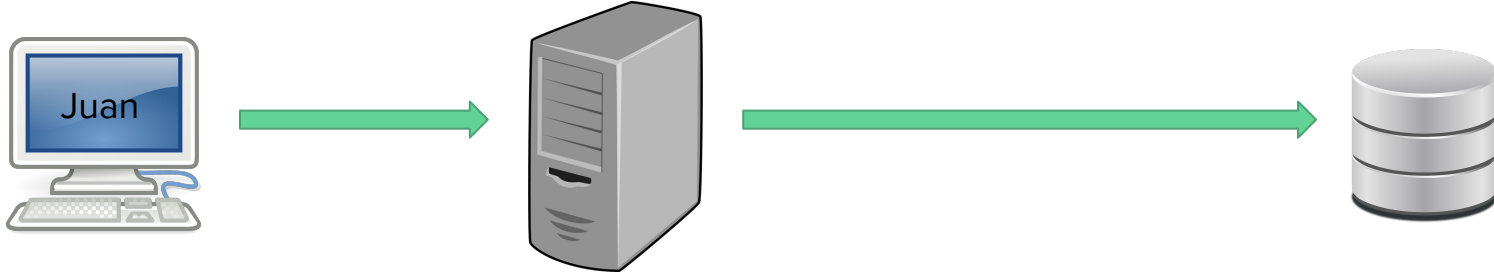
<u>nro</u>	titular	saldo
1000	Juan	800
1001	Pedro	200
1002	Daniel	500

**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200



# Transferencia bancaria 2



“Transferir \$200 a Daniel”

```
UPDATE Cuenta
SET saldo = saldo - 200
WHERE nro = 1000;
```

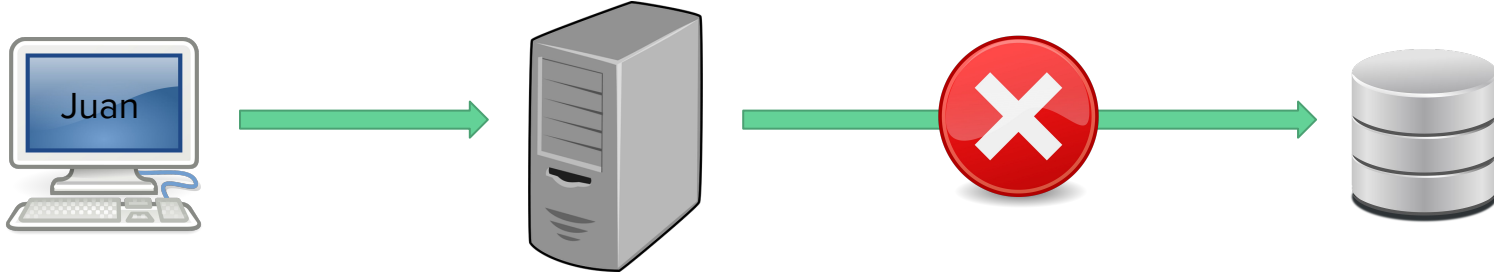
**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	600
1001	Pedro	200
1002	Daniel	500

**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200

# Transferencia bancaria 2



“Transferir \$200 a Daniel”

```
UPDATE Cuenta
SET saldo = saldo - 200
WHERE nro = 1000;
```

```
UPDATE Cuenta
SET saldo = saldo + 200
WHERE nro = 1002;
```

**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	600
1001	Pedro	200
1002	Daniel	500

**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200

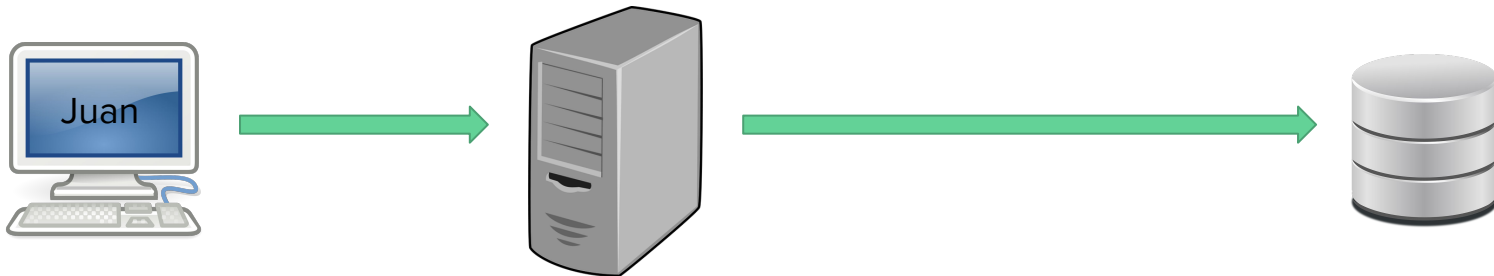
# Transacción

“Conjunto de operaciones que debe ejecutarse en forma atómica en la base de datos”

Se define según las siguientes directivas:

- BEGIN TRANSACTION
- COMMIT TRANSACTION
- ROLLBACK TRANSACTION

# Transacción satisfactoria



“Transferir \$200 a Daniel”



**BEGIN TRANSACTION**

```
UPDATE Cuenta
SET saldo = saldo - 200
WHERE nro = 1000;
```

```
UPDATE Cuenta
SET saldo = saldo + 200
WHERE nro = 1002;
```

```
INSERT INTO Transferencia
(id, fecha, nro_orig, nro_dest, importe)
VALUES ( 4, '20200105', 1000, 1002, 200);
```

**COMMIT TRANSACTION**

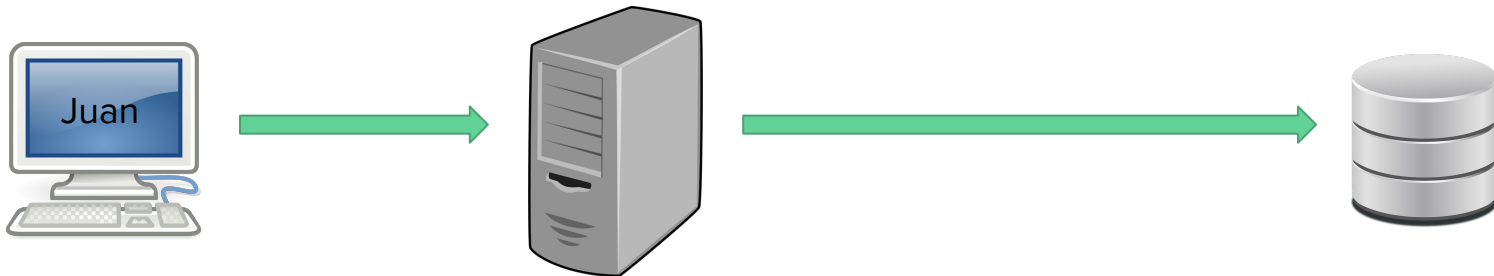
**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	800
1001	Pedro	200
1002	Daniel	500

**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200

# Transacción satisfactoria



“Transferir \$200 a Daniel”

## BEGIN TRANSACTION

UPDATE Cuenta  
SET saldo = saldo - 200  
WHERE nro = 1000;

UPDATE Cuenta  
SET saldo = saldo + 200  
WHERE nro = 1002;

INSERT INTO Transferencia  
( id, fecha, nro\_orig, nro\_dest, importe)  
VALUES ( 4, '20200105', 1000, 1002, 200);

## COMMIT TRANSACTION

Cuenta

nro	titular	saldo
1000	Juan	<del>800</del> 600
1001	Pedro	200
1002	Daniel	500

Transferencia

id	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200

# Transacción satisfactoria



“Transferir \$200 a Daniel”

## BEGIN TRANSACTION

```
UPDATE Cuenta  
SET saldo = saldo - 200  
WHERE nro = 1000;
```

➡ 

```
UPDATE Cuenta  
SET saldo = saldo + 200  
WHERE nro = 1002;
```

```
INSERT INTO Transferencia  
( id, fecha, nro_orig, nro_dest, importe)  
VALUES ( 4, '20200105', 1000, 1002, 200);
```

## COMMIT TRANSACTION

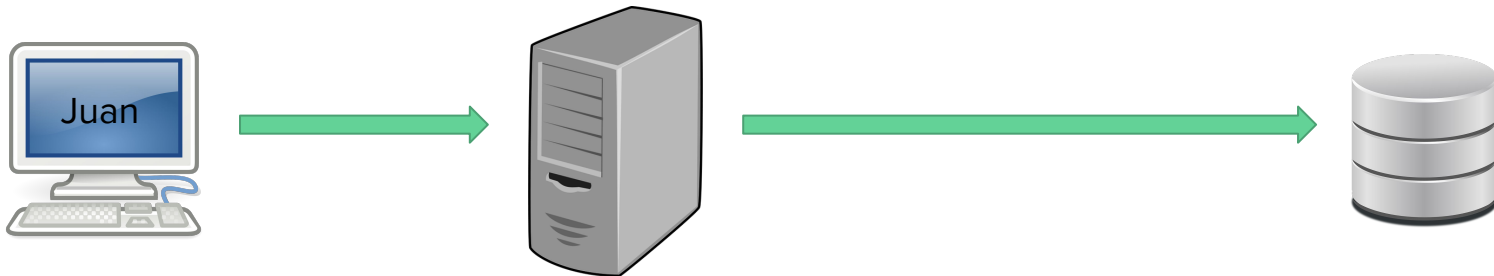
Cuenta

nro	titular	saldo
1000	Juan	<del>800</del> 600
1001	Pedro	200
1002	Daniel	<del>500</del> 700

Transferencia

id	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200

# Transacción satisfactoria



“Transferir \$200 a Daniel”

## BEGIN TRANSACTION

```
UPDATE Cuenta
SET saldo = saldo - 200
WHERE nro = 1000;
```

```
UPDATE Cuenta
SET saldo = saldo + 200
WHERE nro = 1002;
```

➡ INSERT INTO Transferencia  
( id, fecha, nro\_orig, nro\_dest, importe)  
VALUES ( 4, '20200105', 1000, 1002, 200);

## COMMIT TRANSACTION

### Cuenta

nro	titular	saldo
1000	Juan	<del>800</del> 600
1001	Pedro	200
1002	Daniel	<del>500</del> 700

### Transferencia

id	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200
4	5/1/2020	1000	1002	200

# Transacción satisfactoria



“Transferir \$200 a Daniel”

**BEGIN TRANSACTION**

```
UPDATE Cuenta  
SET saldo = saldo - 200  
WHERE nro = 1000;
```

```
UPDATE Cuenta  
SET saldo = saldo + 200  
WHERE nro = 1002;
```

```
INSERT INTO Transferencia  
( id, fecha, nro_orig, nro_dest, importe)  
VALUES ( 4, '20200105', 1000, 1002, 200);
```



**COMMIT TRANSACTION**

**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	600
1001	Pedro	200
1002	Daniel	700

**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200
4	5/1/2020	1000	1002	200



# Transacción satisfactoria



“Transferir \$200 a Daniel”

## BEGIN TRANSACTION

```
UPDATE Cuenta  
SET saldo = saldo - 200  
WHERE nro = 1000;
```

```
UPDATE Cuenta  
SET saldo = saldo + 200  
WHERE nro = 1002;
```

```
INSERT INTO Transferencia  
( id, fecha, nro_orig, nro_dest, importe)  
VALUES ( 4, '20200105', 1000, 1002, 200);
```



## COMMIT TRANSACTION

### Cuenta

<u>nro</u>	titular	saldo
1000	Juan	600
1001	Pedro	200
1002	Daniel	700

### Transferencia

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200
4	5/1/2020	1000	1002	200

# Transacción abortada explícitamente



“Transferir \$100 a Pedro”



**BEGIN TRANSACTION**

```
UPDATE Cuenta  
SET saldo = saldo - 100  
WHERE nro = 1000;
```

```
UPDATE Cuenta  
SET saldo = saldo + 100  
WHERE nro = 1001;
```

```
INSERT INTO Transferencia  
( id, fecha, nro_orig, nro_dest, importe)  
VALUES ( 5, '20200105', 1000, 1001, 100);
```

**COMMIT TRANSACTION**

**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	600
1001	Pedro	200
1002	Daniel	700

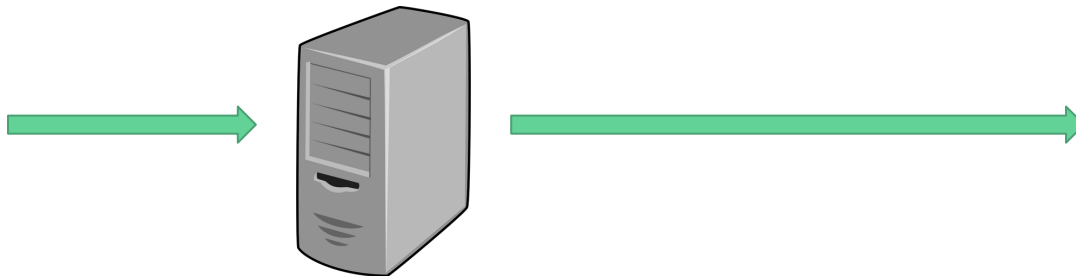
**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200
4	5/1/2020	1000	1002	200

# Transacción abortada explícitamente



“Transferir \$100 a Pedro”



**BEGIN TRANSACTION**

➡ UPDATE Cuenta  
SET saldo = saldo - 100  
WHERE nro = 1000;

UPDATE Cuenta  
SET saldo = saldo + 100  
WHERE nro = 1001;

INSERT INTO Transferencia  
( id, fecha, nro\_orig, nro\_dest, importe)  
VALUES ( 5, '20200105', 1000, 1001, 100);

**COMMIT TRANSACTION**

**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	<del>600</del> 500
1001	Pedro	200
1002	Daniel	700

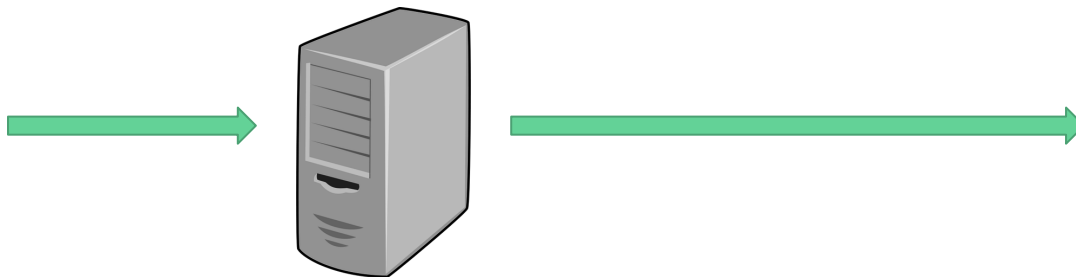
**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200
4	5/1/2020	1000	1002	200

# Transacción abortada explícitamente



“Transferir \$100 a Pedro”



BEGIN TRANSACTION

```
UPDATE Cuenta  
SET saldo = saldo - 100  
WHERE nro = 1000;
```

→ UPDATE Cuenta  
SET saldo = saldo + 100  
WHERE nro = 1001;

```
INSERT INTO Transferencia  
( id, fecha, nro_orig, nro_dest, importe)  
VALUES ( 5, '20200105', 1000, 1001, 100);
```

COMMIT TRANSACTION

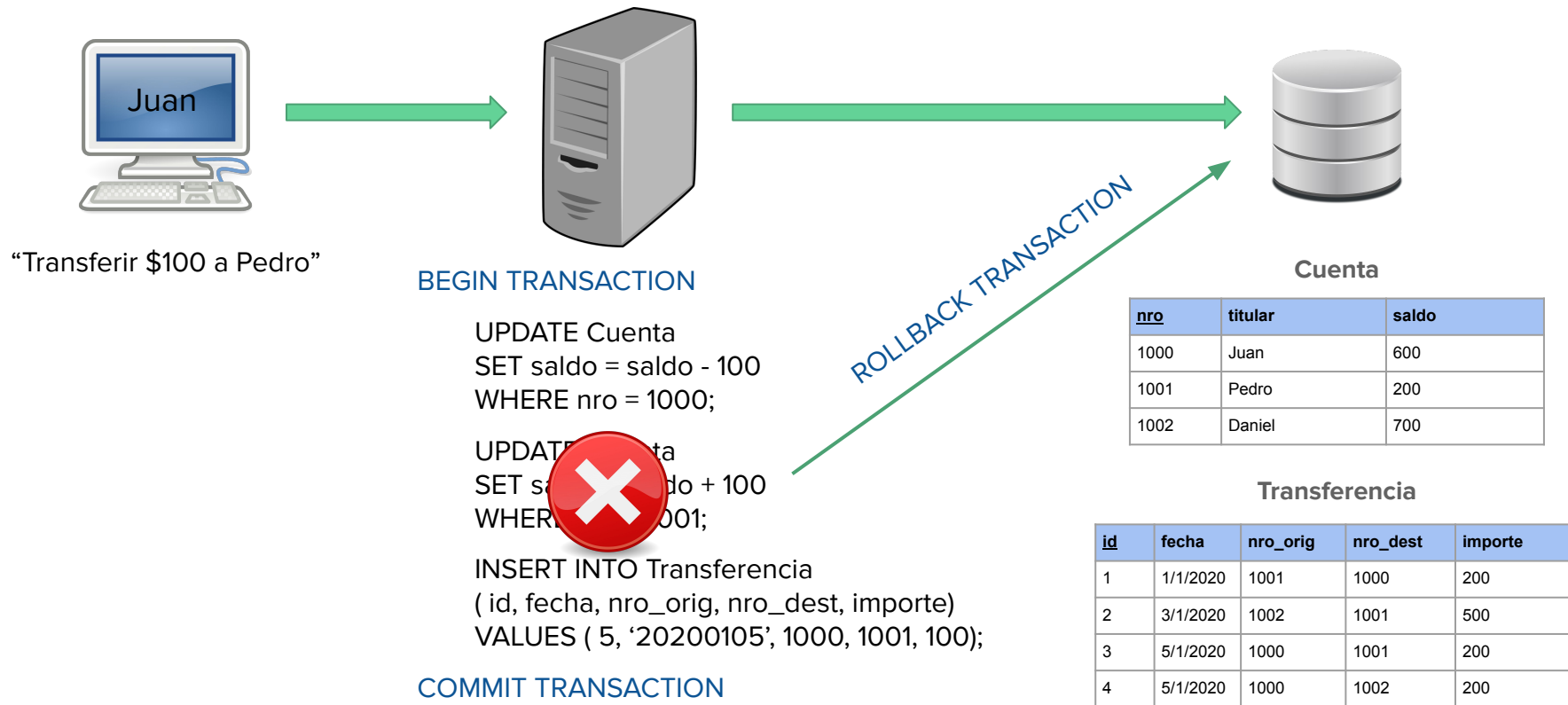
Cuenta

nro	titular	saldo
1000	Juan	<del>600</del> 500
1001	Pedro	200
1002	Daniel	700

Transferencia

id	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200
4	5/1/2020	1000	1002	200

# Transacción abortada explícitamente



# Transacción abortada implícitamente (Timeout)



“Transferir \$100 a Pedro”



**BEGIN TRANSACTION**

```
UPDATE Cuenta
SET saldo = saldo - 100
WHERE nro = 1000;
```

```
UPDATE Cuenta
SET saldo = saldo + 100
WHERE nro = 1001;
```

```
INSERT INTO Transferencia
(id, fecha, nro_orig, nro_dest, importe)
VALUES ( 5, '20200105', 1000, 1001, 100);
```

**COMMIT TRANSACTION**

**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	600
1001	Pedro	200
1002	Daniel	700

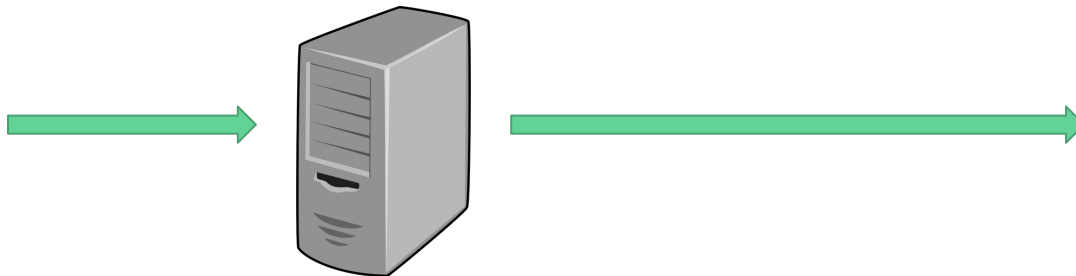
**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200
4	5/1/2020	1000	1002	200

# Transacción abortada implícitamente (Timeout)



“Transferir \$100 a Pedro”



**BEGIN TRANSACTION**

➡ UPDATE Cuenta  
SET saldo = saldo - 100  
WHERE nro = 1000;  
  
UPDATE Cuenta  
SET saldo = saldo + 100  
WHERE nro = 1001;  
  
INSERT INTO Transferencia  
( id, fecha, nro\_orig, nro\_dest, importe)  
VALUES ( 5, '20200105', 1000, 1001, 100);

**COMMIT TRANSACTION**

**Cuenta**

<u>nro</u>	titular	saldo
1000	Juan	<del>600</del> 500
1001	Pedro	200
1002	Daniel	700

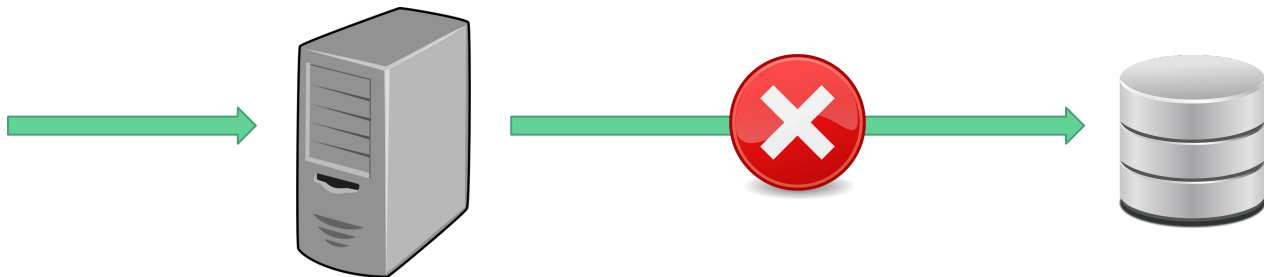
**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200
4	5/1/2020	1000	1002	200

# Transacción abortada implícitamente (Timeout)



“Transferir \$100 a Pedro”



**BEGIN TRANSACTION**

```
UPDATE Cuenta  
SET saldo = saldo - 100  
WHERE nro = 1000;
```



```
UPDATE Cuenta  
SET saldo = saldo + 100  
WHERE nro = 1001;
```

```
INSERT INTO Transferencia  
( id, fecha, nro_orig, nro_dest, importe)  
VALUES ( 5, '20200105', 1000, 1001, 100);
```

**COMMIT TRANSACTION**

**Cuenta**

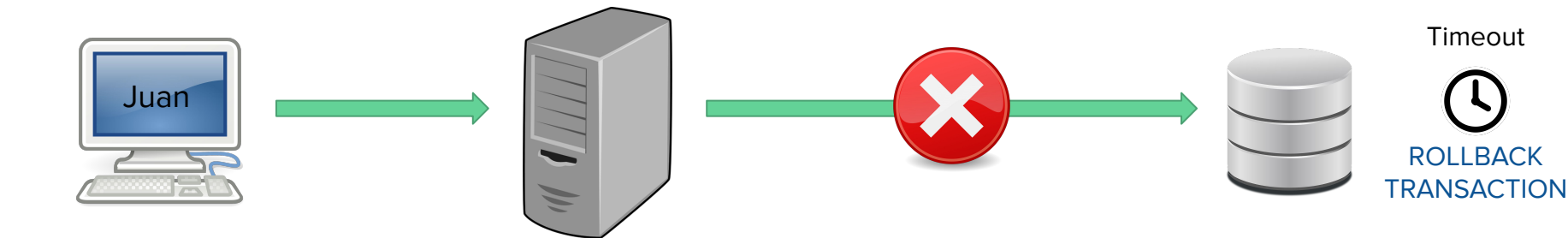
<u>nro</u>	titular	saldo
1000	Juan	<del>600</del> 500
1001	Pedro	200
1002	Daniel	700

**Transferencia**

<u>id</u>	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200
4	5/1/2020	1000	1002	200



# Transacción abortada implícitamente (Timeout)



```
UPDATE Cuenta  
SET saldo = saldo - 100  
WHERE nro = 1000;
```

```
UPDATE Cuenta  
SET saldo = saldo + 100  
WHERE nro = 1001;
```

```
INSERT INTO Transferencia  
( id, fecha, nro_orig, nro_dest, importe)  
VALUES ( 5, '20200105', 1000, 1001, 100);
```

COMMIT TRANSACTION

Cuenta

nro	titular	saldo
1000	Juan	600
1001	Pedro	200
1002	Daniel	700

Transferencia

id	fecha	nro_orig	nro_dest	importe
1	1/1/2020	1001	1000	200
2	3/1/2020	1002	1001	500
3	5/1/2020	1000	1001	200
4	5/1/2020	1000	1002	200

# Otras operaciones

- BEGIN TRANSACTION implícito
- COMMIT TRANSACTION automático