



# Introducción a la arquitectura de la información e introducción al lenguaje SQL

Antonio López

Master in Data Science & Big Data Analytics

Durante esta sesión vamos a hablar de:

- Qué es la arquitectura de datos (o de la información) en una empresa
- Qué y cuáles son los modelos de bases de datos y sistemas de bases de datos más utilizados
- Cómo conectarnos a una base de datos y hacer consultas sencillas a través de SQL

## Índice

1. La arquitectura de datos en una empresa
2. Modelos de bases de datos
3. Sistemas de Gestión de Bases de Datos
4. Iniciación a SQL

# La Arquitectura de datos en una empresa

## *¿Qué es la arquitectura de datos?*

La arquitectura de datos es la **disciplina** que trata de crear un **ecosistema de datos** que dé respuesta a las **necesidades** de una compañía en el **corto, medio y largo plazo**, de una manera óptima en **costes y esfuerzo**.

Arquitectura de datos son los **modelos, políticas, reglas y estándares** que nos indican de qué manera tenemos que almacenar, organizar e integrar los datos que recoge una compañía con el objetivo de que sean aprovechables y útiles.



## *¿Qué es la arquitectura de datos?*

Cuando hablamos de un ecosistema nos referimos al **diseño de bases de datos** que le da soporte.

Cuando hablamos de una compañía hablamos de diseñar **el/los Data Lake y/o Data Warehouse**.

## *Bases de datos*

Una base de datos es cualquier colección de datos organizados para su almacenamiento, acceso y recuperación.

Las bases de datos **suelen** consistir en información dispuesta en filas, columnas y tablas, y organizada principalmente con el fin de facilitar la introducción y recopilación de distintos eventos.

## *Bases de datos operacionales vs Data Warehouse*

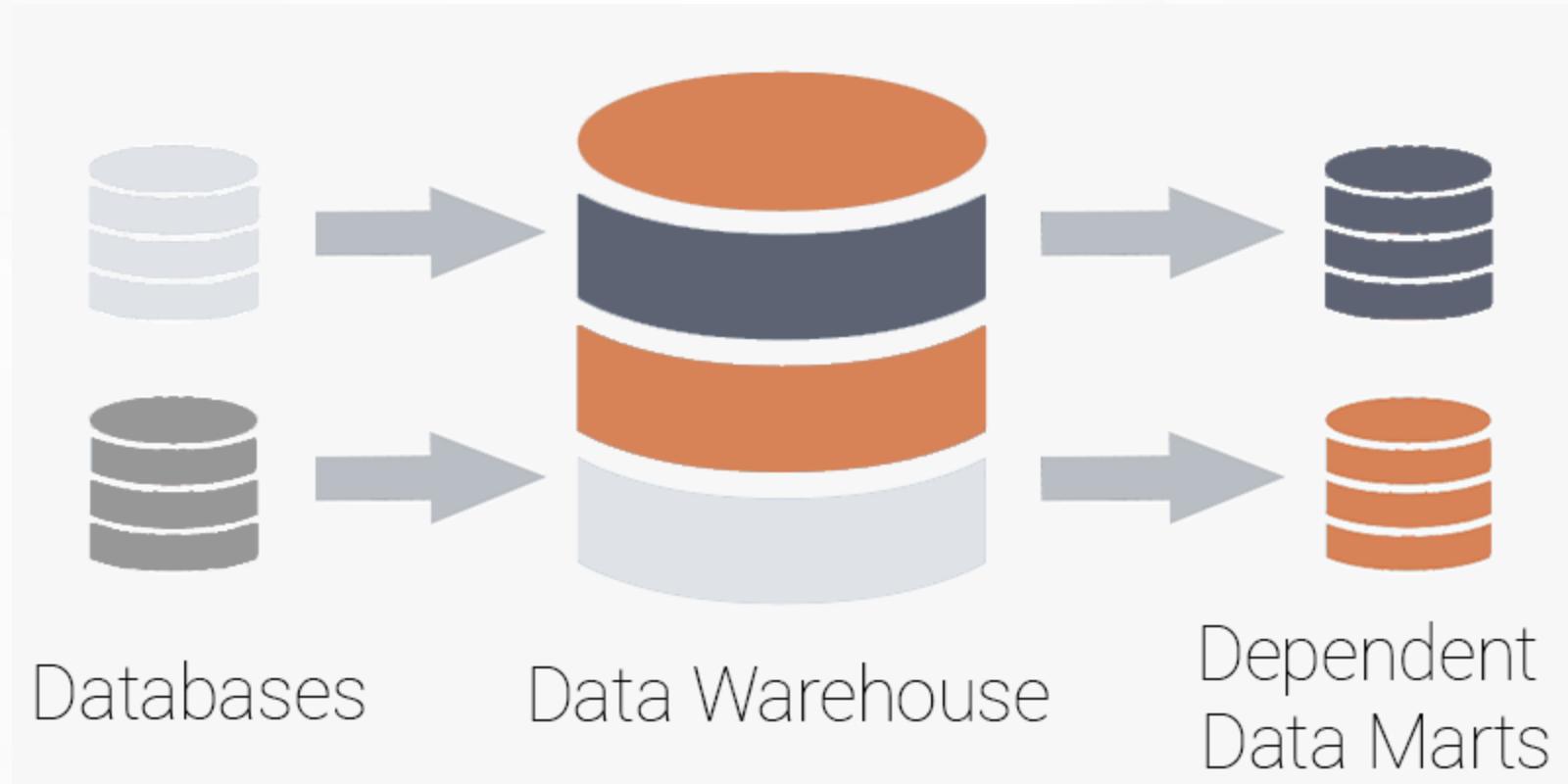
De las **bases de datos operacionales** de una empresa se **extraen datos**, especificando un conjunto de reglas para **transformarlos** y, a continuación, se **cargan** en un repositorio central para un fácil acceso y control.

Este repositorio central es un **Data Warehouse** y el proceso automatizado de extracción, transformación y carga se denomina **ETL**.

## *Data Warehouse y Data Lake*

Como ya hemos dicho, un **Data Warehouse (DW)** contiene los datos que son necesarios o útiles para una organización, es decir, es un repositorio de datos que van a ser transformados para generar información útil para el usuario. Es un conjunto de **Data Marts (DM)**.

Un **Data Lake** es un almacén de datos que todavía no tiene una finalidad definida, es un gran conjunto de datos en bruto. En cambio, en un Data Warehouse, los datos ya están estructurados y filtrados y han sido procesados para un propósito concreto.



# Perfiles en un equipo de Data



Data Architect

**Decide qué datos** son necesarios para la compañía.

Diseña la **estructura lógica** de los datos (qué tablas, vistas, etc), con mucha interlocución tanto con el negocio como con los DataScientist & Analysts para que el diseño les permita realizar su trabajo de la manera más ágil posible.



Data Engineer

Decide e implementa los **sistemas** que darán soporte al resto del equipo, es decir, qué **bases de datos** y qué **tecnologías** utilizar para conseguir los resultados deseados de la manera más **eficiente en tiempo y costes**.

Son los dueños y señores de las **ETLs**.

80% Python -20% SQL



Data Analyst

**Analiza** y extrae **insights** de los datos. Define y monitoriza **KPIs**, realiza **informes** y **análisis** que aporten conocimiento, crea **dashboards**.

Son los **responsables** de ayudar a la compañía a ser **data-driven**.

80% SQL - 20% Python



Data Scientist

Realiza los **análisis más complejos** donde los conocimientos de **estadística o matemática** son necesarios.

Utiliza técnicas de **machine learning & AI** para crear algoritmos que puedan dar respuestas a acontecimientos **futuros** o realizar tareas de manera **automática**.

70% Python - 30% SQL



ML Engineer

Es el último perfil en aparecer. Se enfoca en **desplegar, monitorizar y automatizar el proceso de creación de los modelos**.

Crea **pipelines** de **entrenamiento y despliegue** en las clouds.

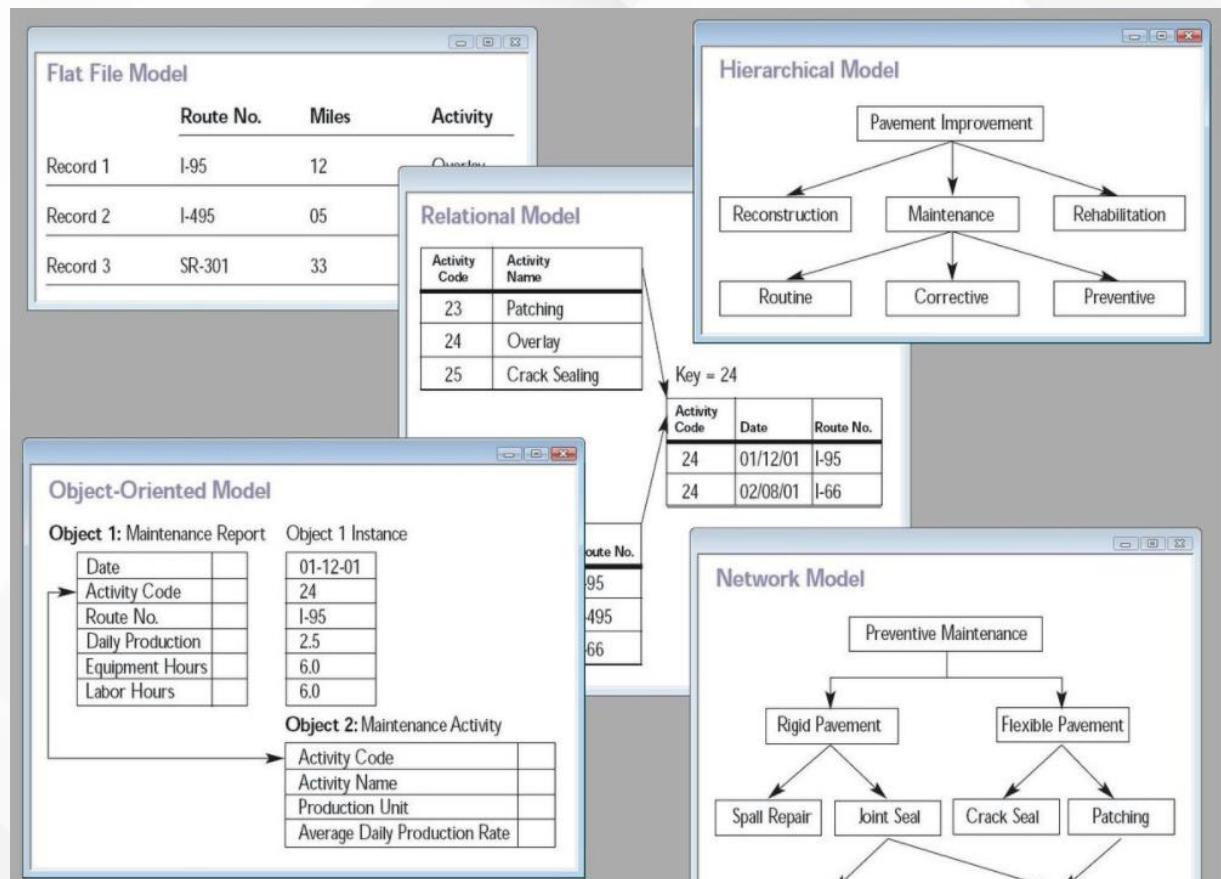
~100% Python

80%SQL - 20%Python

# Modelos de bases de datos

## Estructura lógica de los datos (modelos de bases de datos)

- Modelos antiguos
- **Modelos relacionales**
  - Tradicionales
  - **Modelo estrella y copo de nieve**
  - Post-relacionales
- Modelos no relacionales



# 1. Bases de datos relacionales

El principio de las bases de datos relacionales se basa en la organización de la información en trozos pequeños, que se relacionan entre ellos mediante la relación de identificadores.

Dentro de las bases de datos relacionales, vamos a ver dos tipos que se diferencian por el modelo de datos o estructura lógica de la que provienen.

Bases de datos relacionales tradicionales → esquema ER (entidad-relación)

Bases de datos relacionales dimensionales → esquema estrella o copo de nieve

*Empecemos por las bases de datos relacionales tradicionales*

## 1.1. Bases de datos relacionales tradicionales

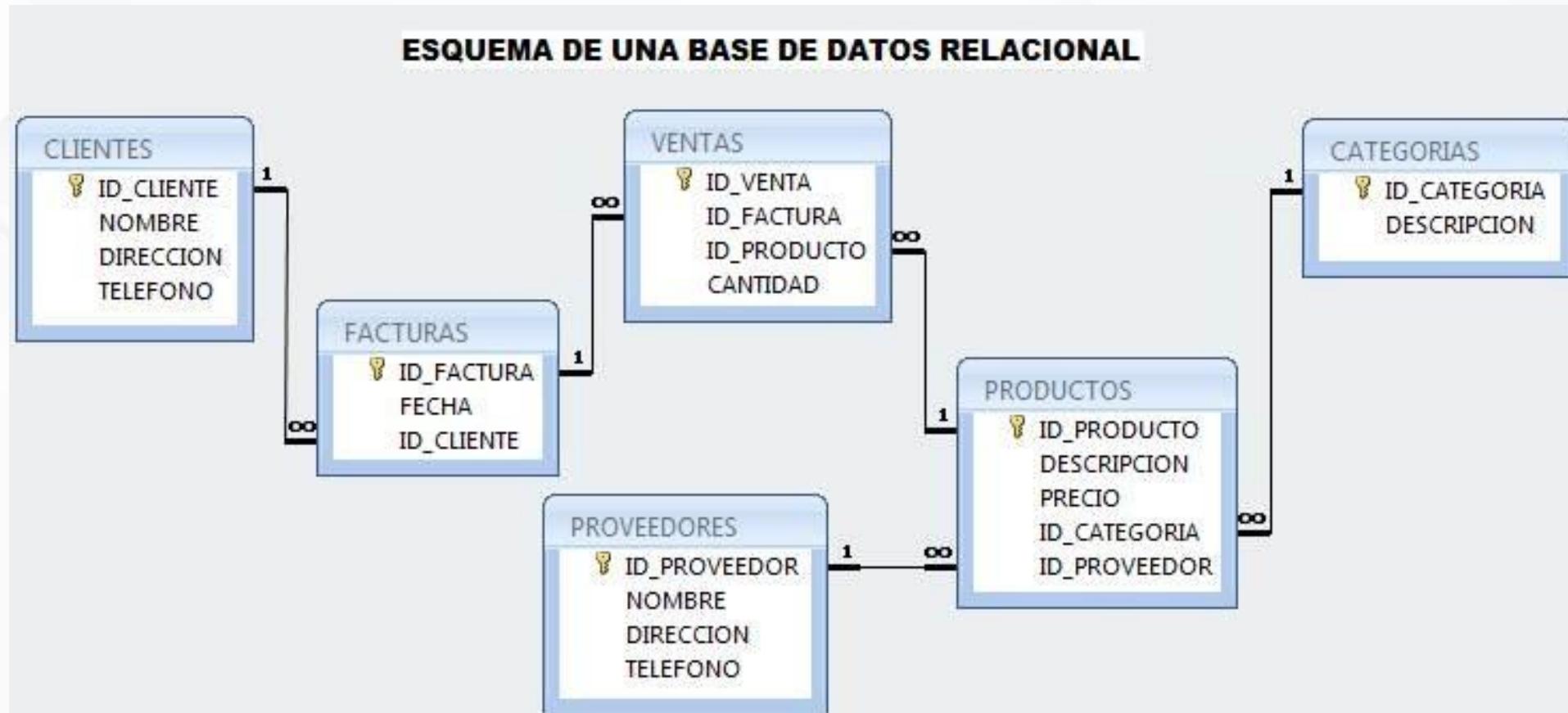
Cuando hablamos de bases de datos relacionales tradicionales hablamos de aquellas que implementamos a partir de un modelo ER (entidad-relación).

**PROS:** Estos modelos son fáciles de entender y están optimizados para el almacenamiento de información y las queries transaccionales.

**CONS:** Es difícil operar con ellos y hacer queries analíticas, además de que solo muestran el estado actual de los datos.

Enfoque OLTP (OnLine Transaction Processing)

Bases de datos operacionales



## Caso: Academia de Idiomas

fecha	cliente	idioma	nivel	suscripción	precio	descuento_%	precio final
25/06/2018	Pedro	Inglés	Intermedio	Mensual	7	0	7
25/06/2018	Pedro	Chino	Principiante	Mensual	9	0	9
01/07/2018	Aurelia	Francés	Avanzado	Anual	8	25	6
03/07/2018	Federico	Inglés	Intermedio	Trimestral	7	10	6.3

- No sabemos si el Pedro de la primera fila es el mismo cliente que el Pedro de la segunda fila o si son dos clientes con el mismo nombre.
- Si algún precio o descuento cambia, hay que modificarlo en todas las filas en las que aparece y, si no se hace correctamente, puede dar lugar a discrepancias. No tiene sentido que la información esté duplicada de esta manera.
- Si un cliente cambia su suscripción, habría que cambiar tanto el campo de suscripción como el precio. Y también puede dar lugar a discrepancias si no se hace correctamente.
- Al tener la columna de “precio final” se está duplicando información, ya que se puede calcular fácilmente con las columnas “precio” y “descuento\_%”. Esto también puede dar lugar a discrepancias.
- No hay manera de saber qué idiomas y niveles hay disponibles, ni cuál es su precio hasta que alguien lo contrate.

Teniendo en cuenta todos los potenciales problemas de almacenamiento que podemos encontrar (slide anterior), convendría tratar de generar una arquitectura sólida que no nos dé problemas a futuro.

Podríamos hacer un pequeño diagrama de una base de datos relacional para tratar de almacenar los datos de la forma más óptima.

Tras diseñar el diagrama debemos **normalizar** la base de datos para su implementación, pasando del plano lógico al plano físico.

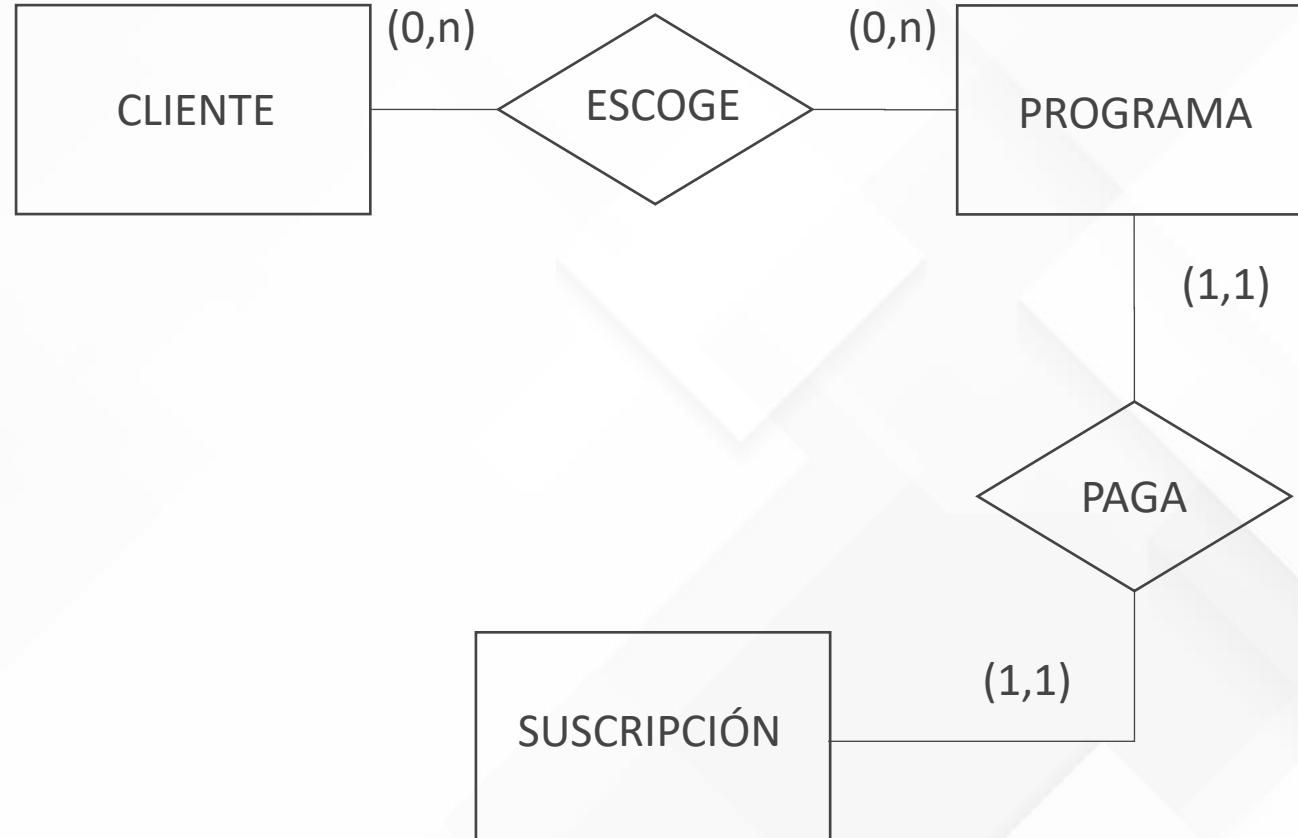
## *Normalización*

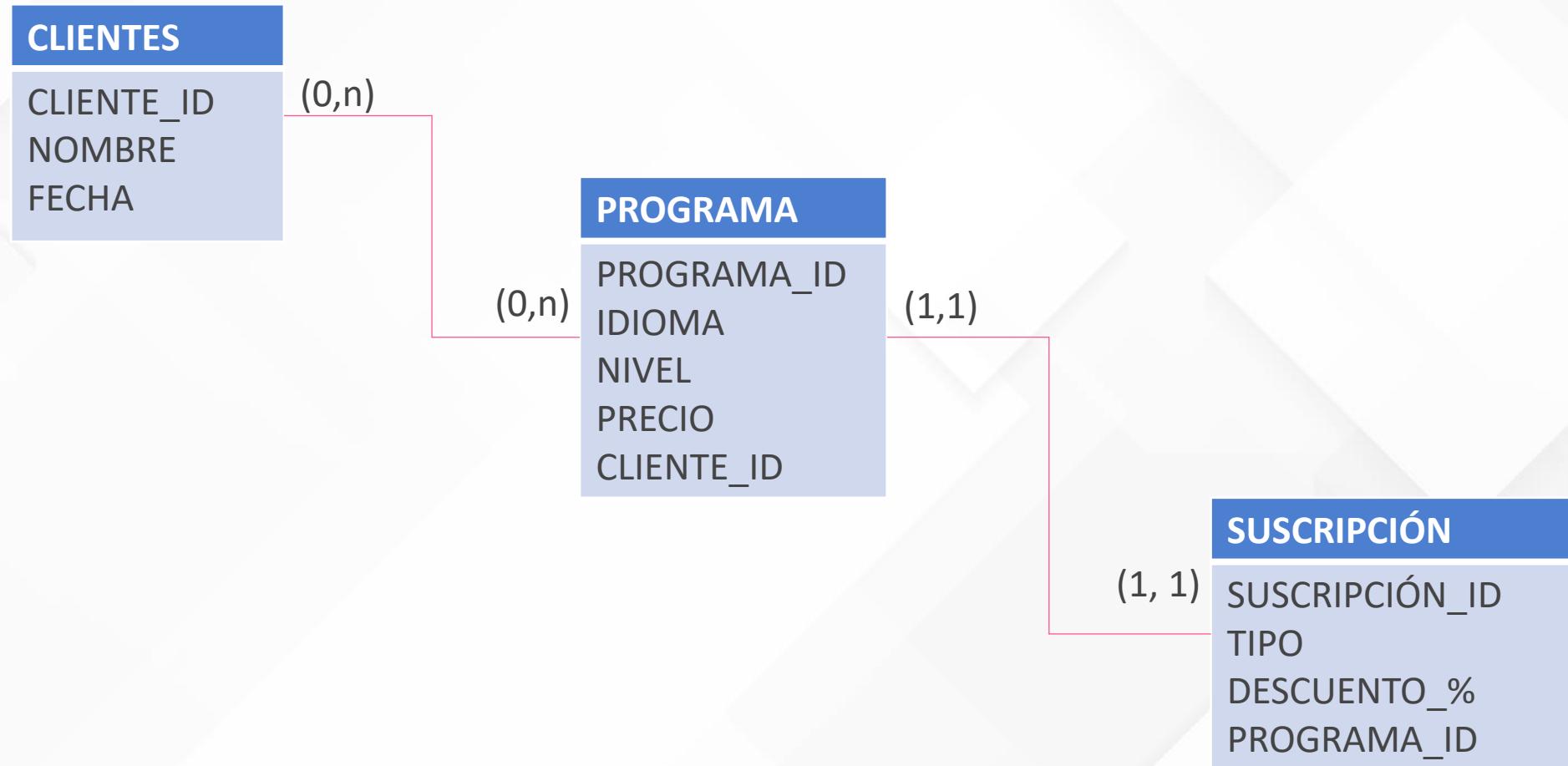
El proceso de normalización de una base de datos relacional consiste en aplicar una serie de reglas para evitar a futuro realizar queries, o consultas innecesariamente complejas. En otras palabras están enfocadas en eliminar redundancias e inconsistencias de dependencia en el diseño de las tablas.

Las bases de datos se normalizan para:

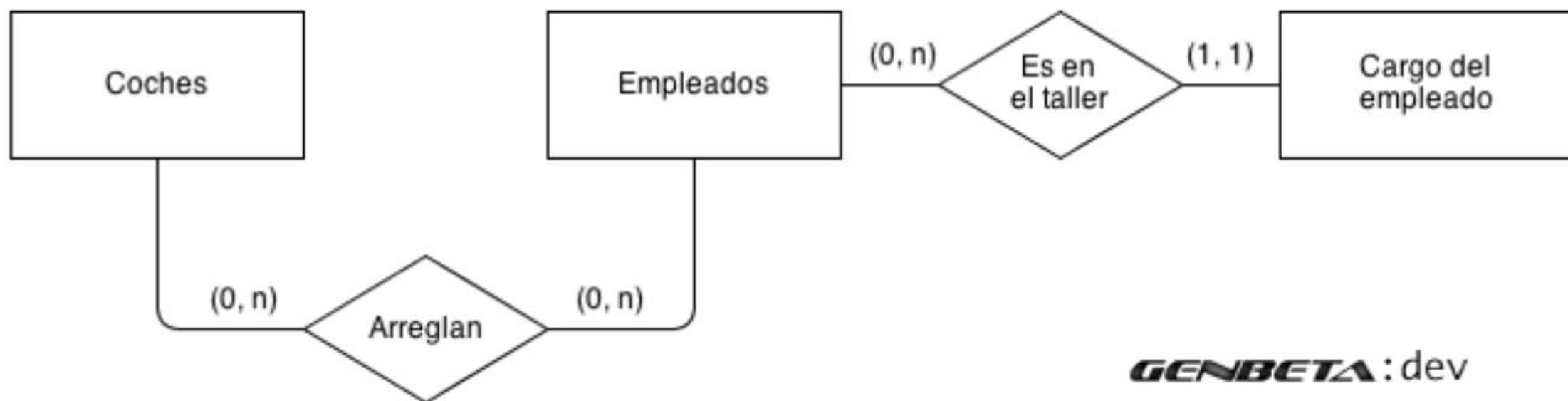
- Evitar la redundancia de datos
- Proteger la integridad de los datos
- Evitar problemas de actualización de los datos en las tablas

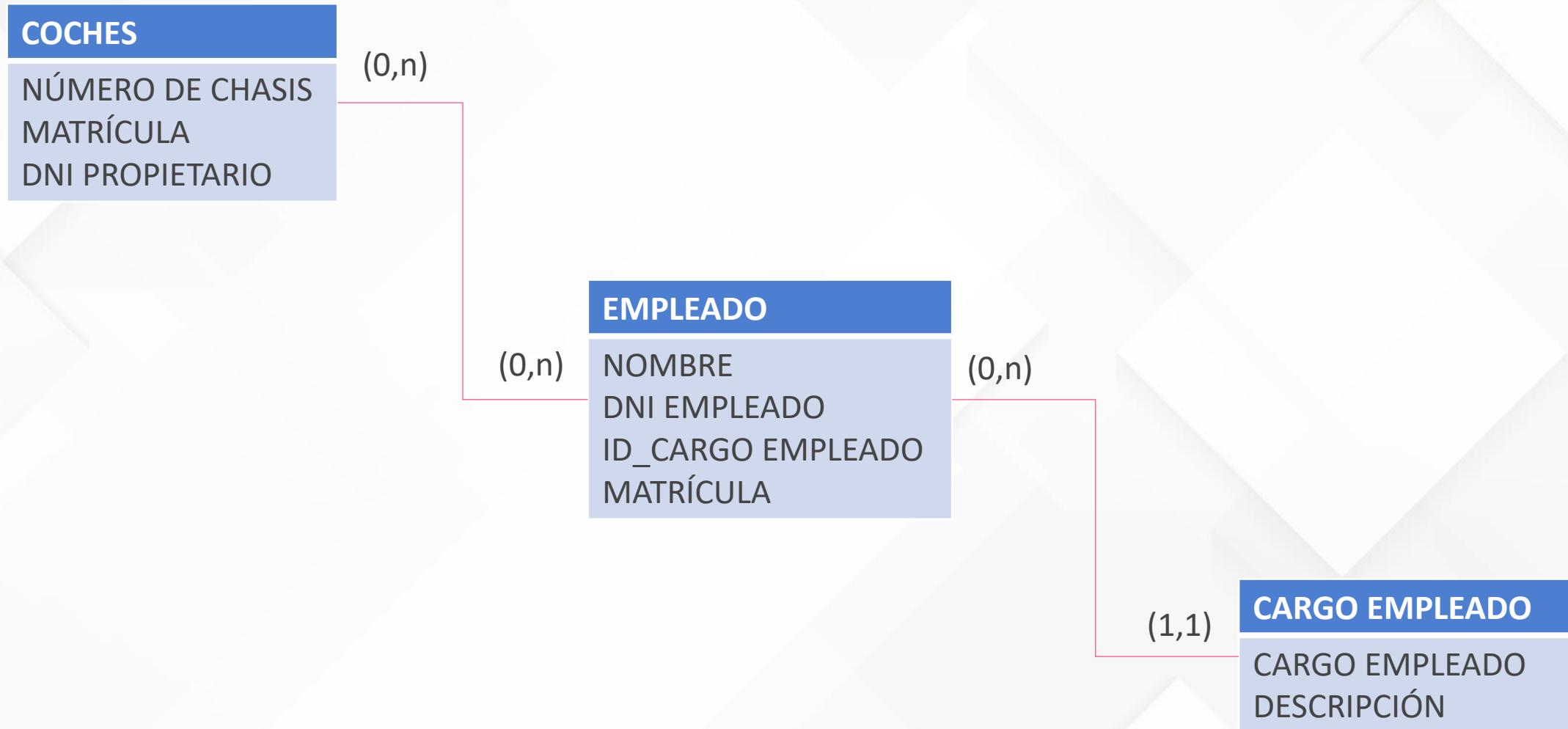
## DIAGRAMA ENTIDAD RELACIÓN





# Modelo entidad-relación





Entidad

## Coches

### Número de chasis

5tfem5f10ax007210

6hsen2j98as001982

5rgsb7a19js001982

Clave Primaria

### Matrícula

4817 BFK

8810 CLM

0019 GGL

### DNI del propietario

45338600L

02405068K

40588860J

## Empleados

### Nombre

Carlos Sánchez

Pepe Sánchez

Juan Sánchez

### DNI

45338600L

02405068K

40588860J

### Cargo

001

002

002

## Cargo del empleado

### ID del cargo

001

002

### Descripción

Jefe de taller

Mecánico

## *Elementos de las bases de datos relacionales tradicionales*

La estructura básica de datos del modelo relacional es la **relación (tabla)**, donde la información acerca de una determinada **entidad** (p. ej. "cliente") se almacena en **tuplas (filas)**, cada una con un conjunto de **atributos (columnas)**. Las columnas de cada tabla enumeran los distintos atributos de la entidad (el nombre del "cliente", dirección y número de teléfono, p. ej.), de modo que cada tupla de la relación "cliente" representa un cliente específico guardando los datos de ese cliente concreto.

- *Relaciones → Tablas*
- *Atributos → Columnas de una relación*
- *Dominios → Conjunto de valores que cada atributo puede tomar*

## 1.2. Bases de datos relacionales dimensionales

Cuando hablamos de bases de datos relacionales dimensionales hablamos de aquellas que implementamos a partir de un modelo estrella o copo de nieve.

Una base de datos relacional tradicional puede ser transformada a una base de datos relacional dimensional para crear el Data Warehouse.

Enfoque OLAP (OnLine Analytical Processing)

Data Warehouse

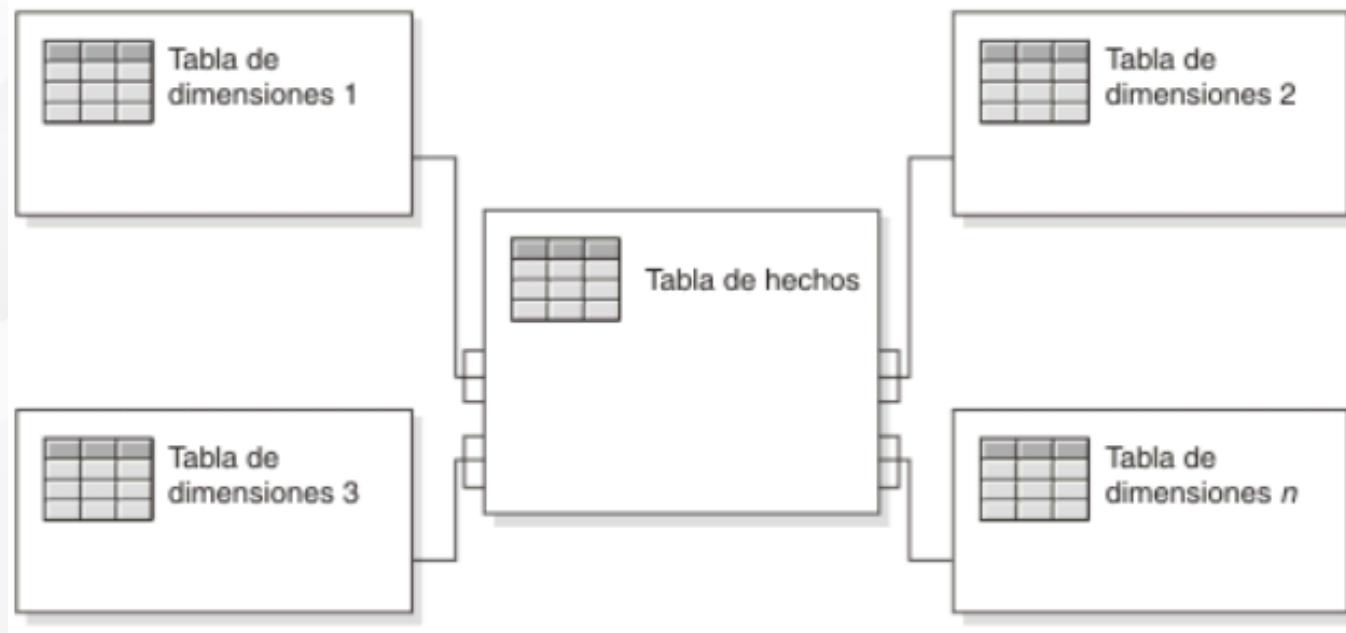
## *Esquema de estrella*

Un **esquema de estrella** es un tipo de esquema de base de datos relacional que consta de una sola tabla de **hechos** central rodeada de tablas de **dimensiones**.

Un esquema de estrella puede tener cualquier número de tablas de dimensiones. **Las ramas situadas al final de los enlaces que conectan las tablas indican una relación de muchos a uno entre la tabla de hechos y cada tabla de dimensiones.**

Figura 1. Esquema de estrella con una sola tabla de hechos con enlaces a varias tablas de dimensiones.

#### Esquema de estrella

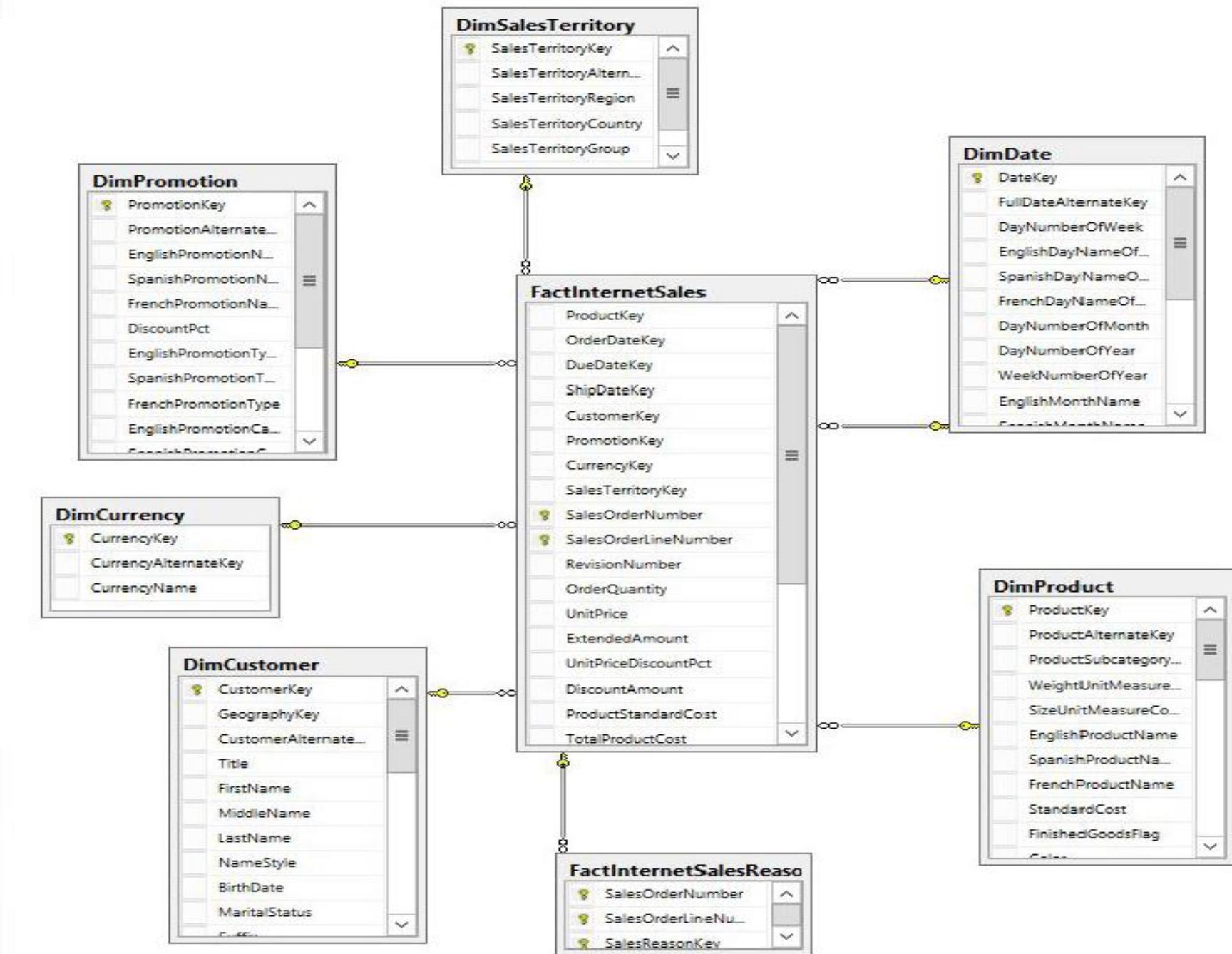


## *Esquema de estrella*

Esto quiere decir que la única tabla que tiene relación con otra es la de hechos, lo que significa que toda la información relacionada con una dimensión debe estar en una sola tabla.

**PROS:** Simple y fácil de usar. Mejor rendimiento en queries analíticas que el modelo ER.

**CONS:** Necesitas un equipo que traduzca en modelo ER en un modelo estrella a través de ETL's. Rígido y subóptimo en entornos big data.



## *Esquema de copo de nieve*

El **esquema de copo de nieve** consta de una tabla de hechos que está conectada a muchas tablas de dimensiones, que pueden estar conectadas a otras tablas de dimensiones a través de una relación de muchos a uno.

Un esquema de copo de nieve puede tener varias dimensiones y cada dimensión puede tener varios niveles.

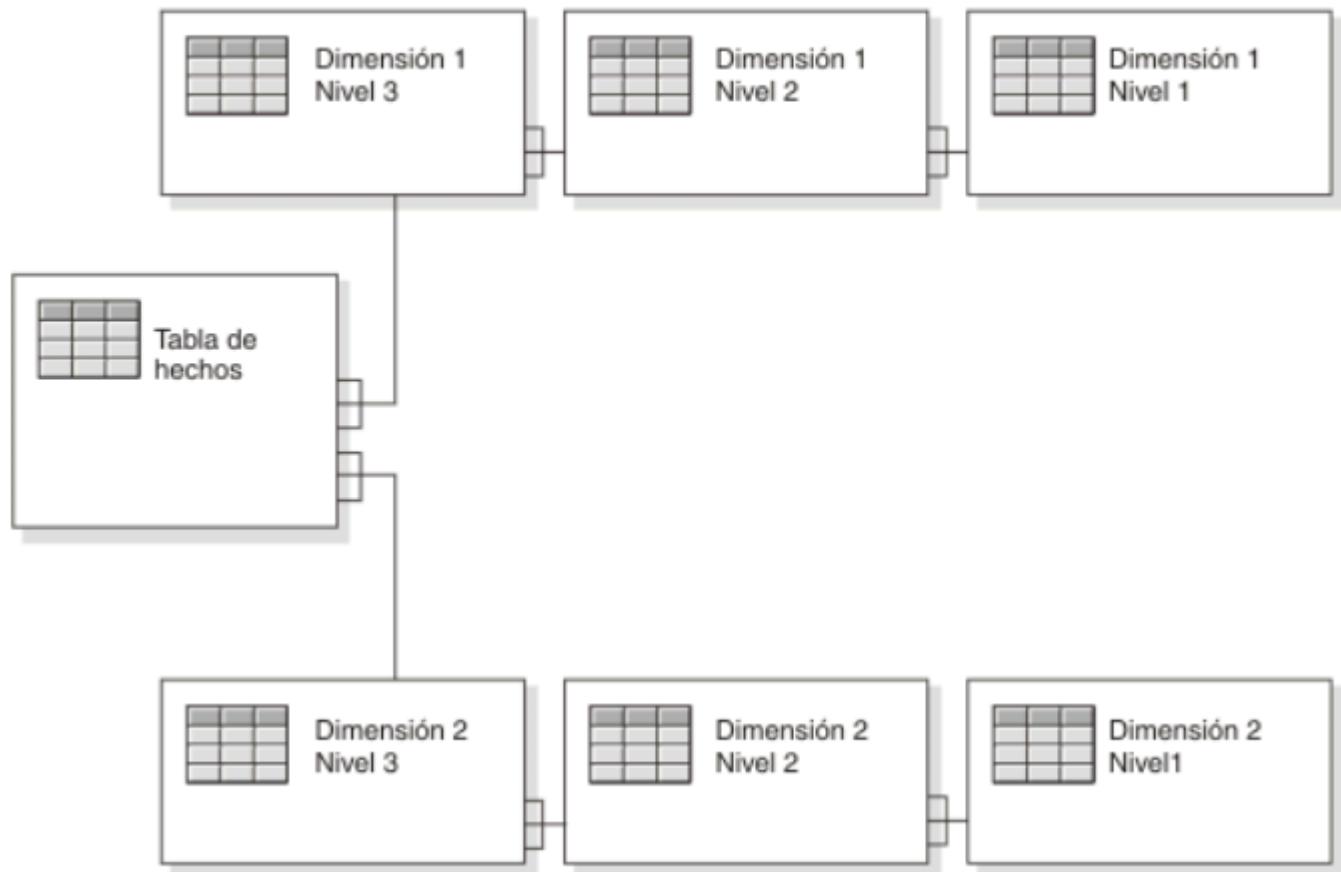
Se puede ver como una variación del esquema estrella.

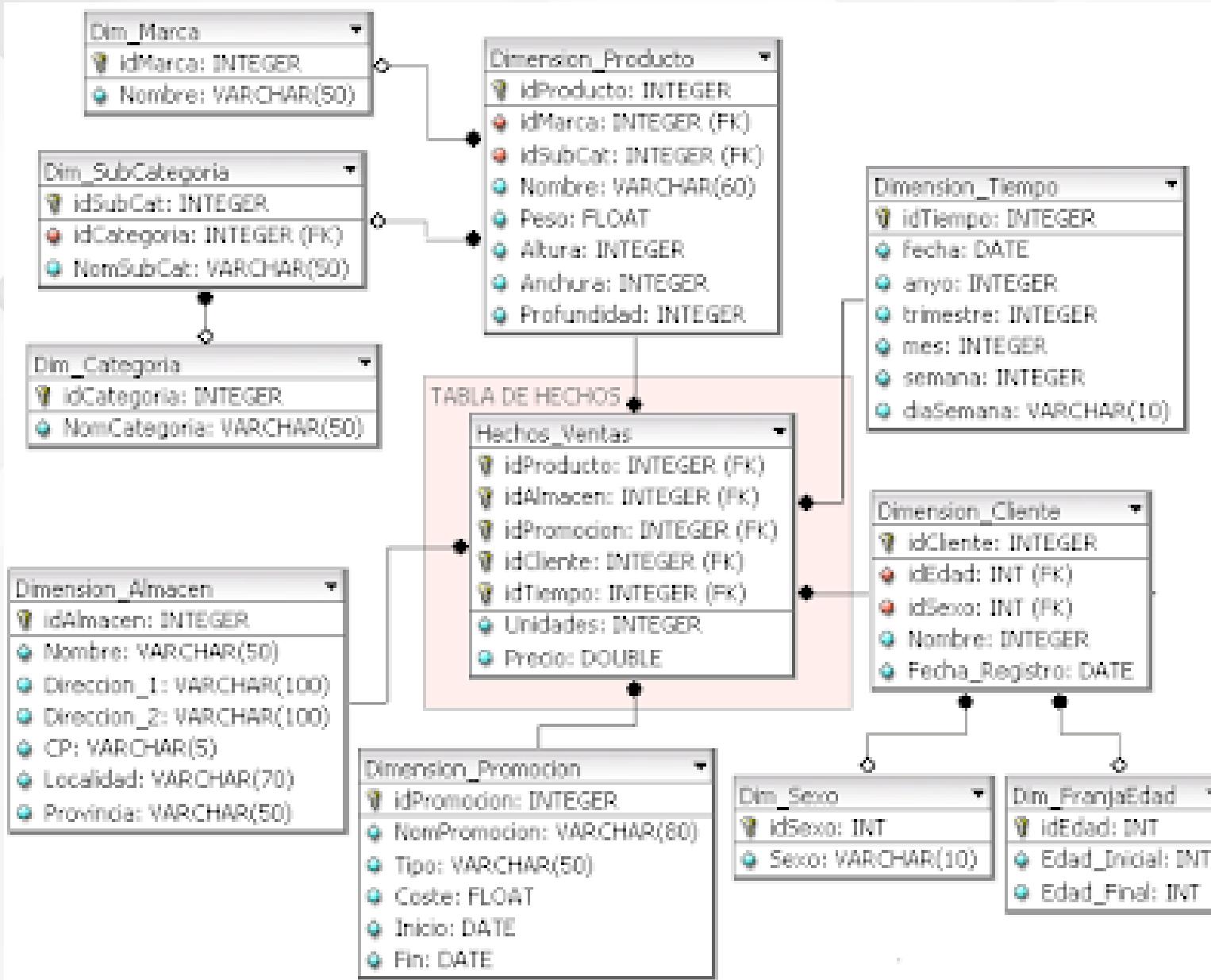
**PROS:** Menos redundancia de datos

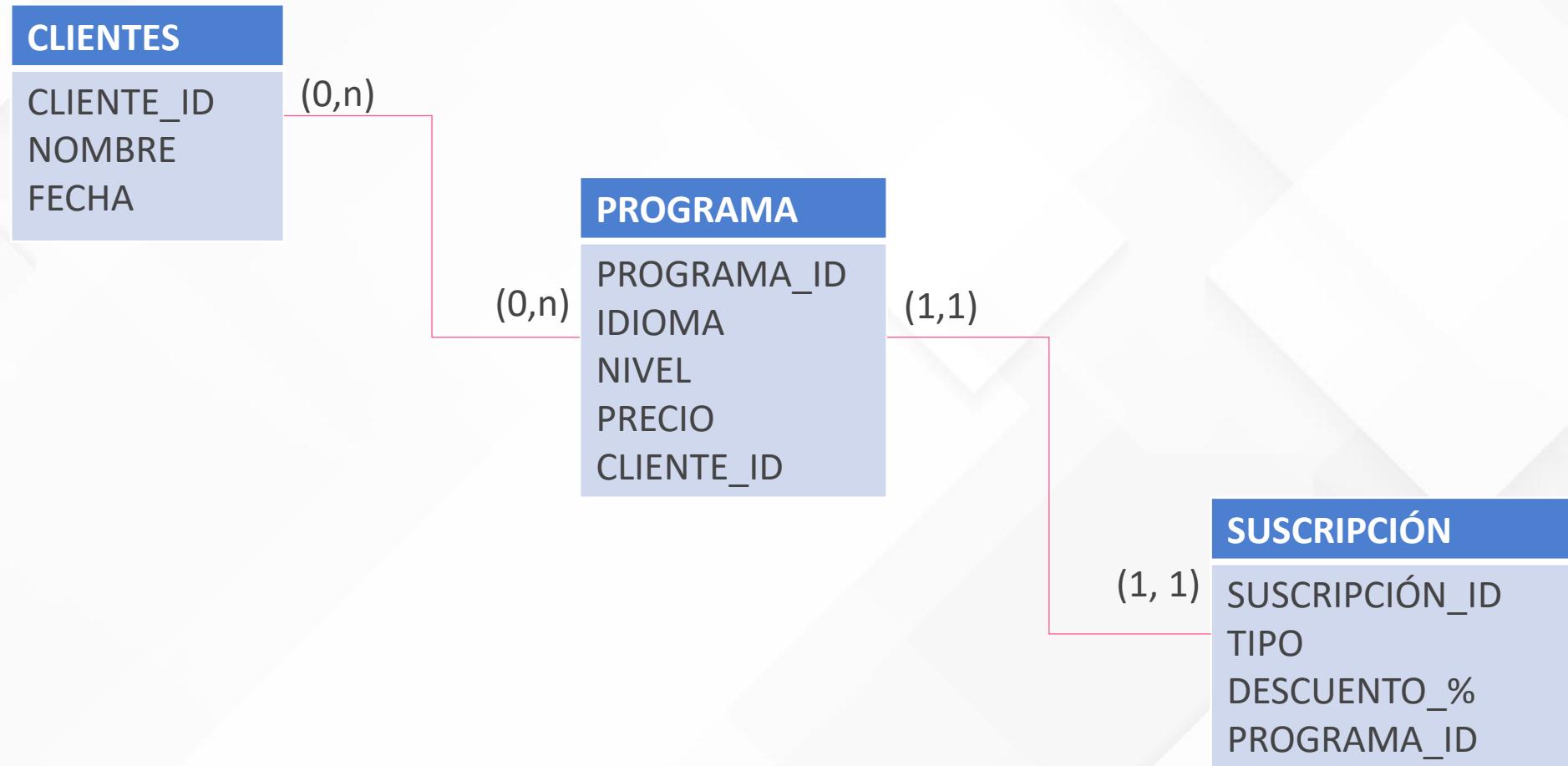
**CONS:** ETL's más complejas, más difícil mantenimiento.

*Figura 1. Esquema de copo de nieve con dos dimensiones y tres niveles cada una*

**Esquema de copo de nieve**







cliente_id	nombre_cliente
1	Pedro
2	Aurelia
3	Federico

id	cliente_id	programa_id	suscripcion_id
1	1	4	1
2	1	2	1
3	2	3	3
4	3	4	2

suscripcion_id	tipo	descuento_%
1	Mensual	0
2	Trimestral	10
3	Anual	25

programa_id	idioma	nivel	precio
1	alemán	principiante	7
2	chino	principiante	9
3	francés	avanzado	8
4	inglés	intermedio	7

## 1.3. Bases de datos no relacionales

student_id	age	score
1	12	77
2	12	68
3	11	75



```
[  
  {  
    "student_id":1,  
    "age":12,  
    "score":77  
  },  
  {  
    "student_id":2,  
    "age":12,  
    "score":68  
  },  
  {  
    "student_id":3,  
    "age":11,  
    "score":75  
  }  
]
```

# Sistemas de gestión de bases de datos (SGBD)

## *Implementación y diseño físico de bases de datos*

Podemos definir el diseño físico de una base de datos como un proceso que, a partir de su diseño lógico y de información sobre su uso esperado, creará una configuración física de la base de datos **adaptada al entorno** donde se alojará y que permita el almacenamiento y la explotación de los datos de la base de datos con un rendimiento adecuado.

## *Implementación y diseño físico de bases de datos*

El paso de un diseño lógico a uno físico requiere un conocimiento profundo del SGBD donde se vaya a implementar la base de datos.

SGBD relacionales más populares:

- Microsoft Access
- Microsoft SQL Server
- MySQL
- Oracle Database
- PostgreSQL
- MariaDB

## Conceptos clave a la hora de trabajar en un SGBD

- **Tablespace:** es un componente del SGBD que indica dónde se almacenarán los datos de la base de datos y en qué formato.
- **Índices o claves:** son estructuras de datos que permiten mejorar el tiempo de respuesta de las consultas sobre los datos de una tabla.
- **Vistas materializadas:** los resultados de las consultas sobre una o más tablas se pueden almacenar en vistas materializadas. A diferencia de las vistas convencionales, el conjunto de filas que conforma el contenido de una vista materializada se almacena físicamente en la base de datos.
- **Particiones:** permiten distribuir los datos de una tabla en distintos espacios físicos.

# Tipos de Índices

**Primary Key** Clave Principal de una tabla, es un valor único que no puede repetirse y generalmente es autonumérico.

**Unique Key** Clave Única, es un valor único que no puede repetirse y no es autonumérico.

**Key** Clave común, admite duplicados.

# Iniciación a SQL

# ¿Qué es SQL?

- SQL son las siglas de **S**tructured **Q**uery **L**anguage, y permite acceder y manipular bases de datos.
- Es un **estándar ISO** desde 1987
- Es un lenguaje que nos permite hablar con bases de datos para crear , destruir, actualizar o **seleccionar datos**.



# Utilizaremos BigQuery como nuestro campo de pruebas de SQL

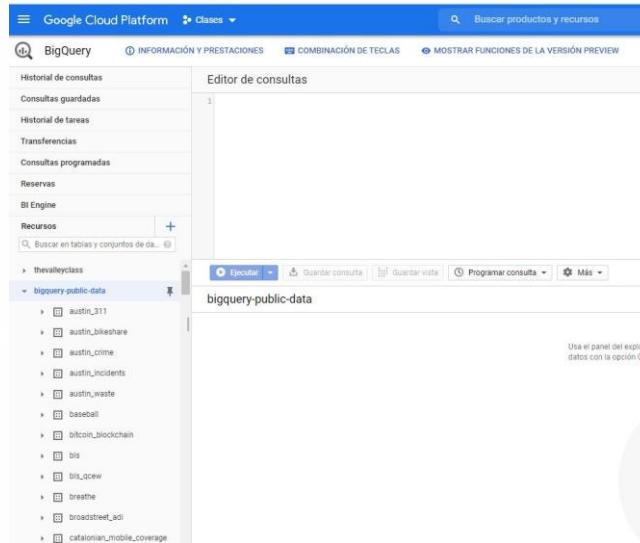
1 -> Acceder a <https://console.cloud.google.com/bigquery>

2-> Conseguir acceso al conjunto de datos de prueba de BigQuery

<https://console.cloud.google.com/marketplace/product/san-francisco-public-data/sf-bike-share>

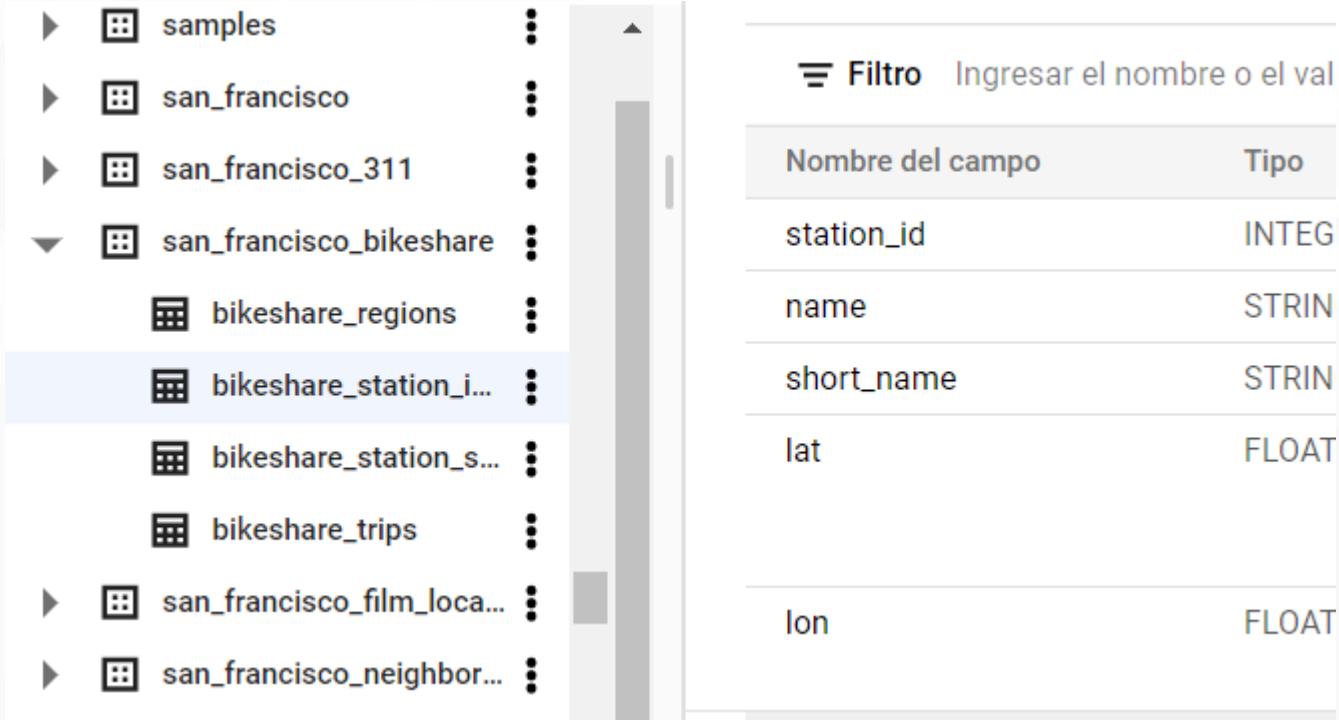
3 -> Dentro de bigquery-publicdata jugaremos con el conjunto de datos san\_francisco\_bikeshare.

1. Hacemos clic en VER CONJUNTO DE DATOS
2. En el Explorador hacemos clic en bigquery-public-data
3. Buscamos san\_francisco\_bikeshare



The screenshot shows the Google Cloud Platform BigQuery interface. On the left, there's a sidebar with options like Historial de consultas, Consultas guardadas, Historial de tareas, Transferencias, Consultas programadas, Reservas, BI Engine, and Recursos. Below that is a search bar for 'Consultas y conjuntos de datos'. The main area shows the 'bigquery-public-data' dataset expanded, revealing tables such as austin\_311, austin\_bikeshare, austin\_crime, austin\_incidents, austin\_waste, baseball, bitcoin\_blockchain, bix, bix\_qnew, breathe, broadstreet\_adt, and catalonian\_mobile\_coverage. To the right, there's a 'Recursos' sidebar with a search bar for 'Buscar en tablas y conjuntos de datos'. A red box highlights the 'san\_francisco\_bikeshare' table under the 'bigquery-public-data' section, which contains sub-tables: bikeshare\_regions, bikeshare\_station\_info, bikeshare\_station\_status, and bikeshare\_trips. Below these are other tables: san\_francisco\_film\_locations, san\_francisco\_sfdd\_service\_calls, and san\_francisco\_sfpd\_incidents.

Es una base de datos relacional formada por cuatro tablas, cada una con sus respectivos atributos (columnas)



The screenshot shows a database interface with a sidebar on the left listing tables:

- samples
- san\_francisco
- san\_francisco\_311
- san\_francisco\_bikeshare
  - bikeshare\_regions
  - bikeshare\_station\_i...
  - bikeshare\_station\_s...
  - bikeshare\_trips
- san\_francisco\_film\_loca...
- san\_francisco\_neighbor...

To the right, a detailed view of the 'san\_francisco\_bikeshare' table is shown:

Nombre del campo	Tipo
station_id	INTEG
name	STRIN
short_name	STRIN
lat	FLOAT
lon	FLOAT

 REDACTAR CONSULTA NUEVA

 Type a query to get started

Para empezar a hacer consultas clicamos

# Es un lenguaje sencillo

**select \* from tabla**

Siempre trabajaremos con la sentencia select. Todas nuestras consultas empezarán así

\* selecciona todas las columnas

después de seleccionar la columna debemos de poner la palabra “from” seguida de la tabla que queremos consultar

Esta consulta nos devuelve todas las columnas y filas de la tabla.

Podemos seleccionar solo las columnas que queramos

`select nombre, apellidos from clientes`

Normalmente no queremos todas las columnas de una tabla. Escribiremos los nombres de las columnas que queremos seleccionar separados por comas

# Recordar dejar limpio el código

```
select  
    nombre,  
    apellidos  
from  
    clientes
```

los espacios y saltos de línea no  
importan, solo tratan de hacer más  
entendible la consulta

Por alguna razón quieres saber en qué ciudades de California puedes encontrar estaciones de bicis.

Utiliza SELECT y FROM para obtener los nombres de las regiones donde hay estaciones de bicis

TABLA: `bigquery-public-data.san\_francisco\_bikeshare.bikeshare\_regions`

*\*Nota: el acento ` está al lado de la P en el teclado español, no es lo mismo que la comilla simple.*

```
1 SELECT name  
2 FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_regions`
```

# Where para filtrar filas

select

\*

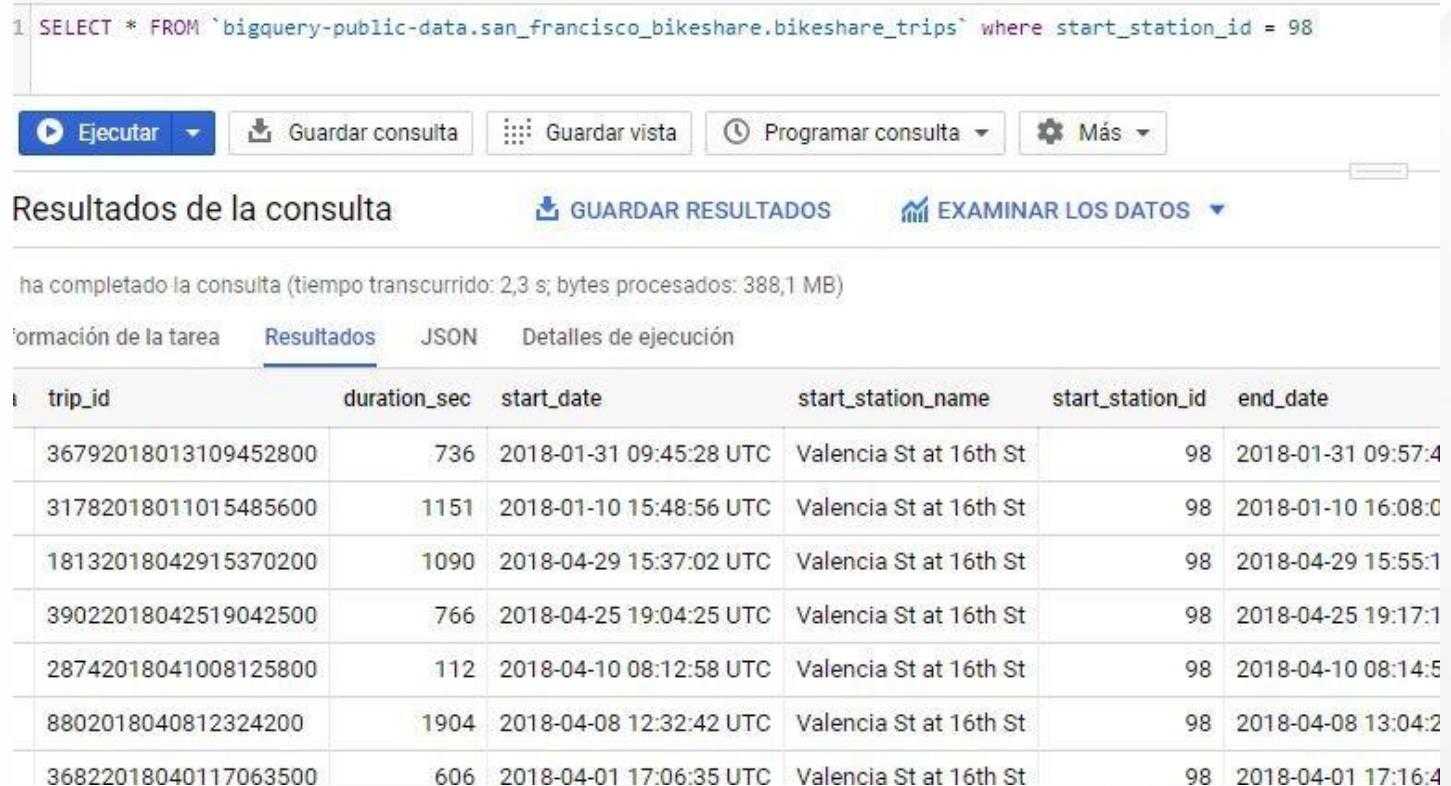
from

'nombre\_tabla'

where

start\_station\_id= 98

1 SELECT \* FROM `bigquery-public-data.san\_francisco\_bikeshare.bikeshare\_trips` where start\_station\_id = 98



Ejecutar Guardar consulta Guardar vista Programar consulta Más

Resultados de la consulta GUARDAR RESULTADOS EXAMINAR LOS DATOS

ha completado la consulta (tiempo transcurrido: 2,3 s; bytes procesados: 388,1 MB)

trip_id	duration_sec	start_date	start_station_name	start_station_id	end_date
36792018013109452800	736	2018-01-31 09:45:28 UTC	Valencia St at 16th St	98	2018-01-31 09:57:4
31782018011015485600	1151	2018-01-10 15:48:56 UTC	Valencia St at 16th St	98	2018-01-10 16:08:0
18132018042915370200	1090	2018-04-29 15:37:02 UTC	Valencia St at 16th St	98	2018-04-29 15:55:1
39022018042519042500	766	2018-04-25 19:04:25 UTC	Valencia St at 16th St	98	2018-04-25 19:17:1
28742018041008125800	112	2018-04-10 08:12:58 UTC	Valencia St at 16th St	98	2018-04-10 08:14:5
8802018040812324200	1904	2018-04-08 12:32:42 UTC	Valencia St at 16th St	98	2018-04-08 13:04:2
36822018040117063500	606	2018-04-01 17:06:35 UTC	Valencia St at 16th St	98	2018-04-01 17:16:4

# Operadores de comparacion para usar en filtros

Operador	Significado	Ejemplo
= (en algunos SQL, ==)	Igual a	a = 3
!=	Distinto de	a != 3
<	Menor que	a < 3
>	Mayor que	a > 3
<=	Menor igual	a >= 3
>=	Mayor igual	a <= 3

# Operadores de comparación para usar en filtros de tipo texto

**where** nombre = “Pepe”

**where** nombre like “%Jose%”

la cadena de texto “Nombre” contenga “Jose”

**where** nombre like “Maria%”

la cadena de texto “Nombre” empiece por “Maria”

**where** nombre like “Maria%” **and** edad>15

Los filtros se pueden combinar con paréntesis y AND y OR

```
where (nombre like "Maria%" and edad>15) or  
sexo="Hombre"
```

Por alguna razón quieres saber la fecha de inicio (start\_date) y el nombre de la estación de inicio (start\_station\_name) de aquellos viajes registrados que terminaron (end\_date) antes del '2014-06-11'.

Utiliza SELECT, FROM, WHERE y los operadores de comparación para obtener el resultado deseado

---

```
1 SELECT start_date, start_station_name
2 FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
3 WHERE end_date < '2014-06-11'
```

# ORDER BY

SQL permite la ordenación de resultados mediante la sentencia **ORDER BY**, que se sitúa al final de la consulta. Por defecto, ordena los resultados de menor a mayor, pero nosotros podemos indicarlo escribiendo **desc** o **asc** para ordenar de forma descendente o ascendente.

```
select nombre, edad order by edad
```

```
select nombre, edad order by edad asc
```

```
select nombre, edad order by edad desc
```

# Ordenando los resultados con ORDER BY

```
1 SELECT * FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips` ORDER BY start_date DESC LIMIT 1000
```

## Resultados de la consulta

[GUARDAR RESULTADOS](#)
[EXAMINAR LOS DATOS ▾](#)

Se ha completado la consulta (tiempo transcurrido: 2,8 s; bytes procesados: 388,1 MB)

[Información de la tarea](#)
[Resultados](#)
[JSON](#)
[Detalles de ejecución](#)

ila	trip_id	duration_sec	start_date	start_station_name	start_stati
	13342018043023584500	256	2018-04-30 23:58:45 UTC	Telegraph Ave at 19th St	
	792018043023574300	95	2018-04-30 23:57:43 UTC	Jersey St at Castro St	
	33562018043023565700	323	2018-04-30 23:56:57 UTC	Folsom St at 3rd St	
	37822018043023564600	275	2018-04-30 23:56:46 UTC	Rhode Island St at 17th St	
	30992018043023510700	477	2018-04-30 23:51:07 UTC	Haste St at Telegraph Ave	
	12752018043023504400	371	2018-04-30 23:50:44 UTC	Powell St BART Station (Market St at 5th St)	
	33682018043023504000	200	2018-04-30 23:50:40 UTC	Golden Gate Ave at Polk St	
	23622018043023495200	246	2018-04-30 23:49:52 UTC	S Park St at 3rd St	
	4482018043023473400	441	2018-04-30 23:47:34 UTC	Bancroft Way at Telegraph Ave	
0	29552018043023444000	125	2018-04-30 23:44:40 UTC	San Fernando St at 4th St	

# LIMIT para limitar el número de filas devueltas

Siempre podemos limitar el número de resultados devueltos añadiendo al final de la query “LIMIT X” donde X es el número de registros máximos que queremos

1 SELECT \* FROM `bigquery-public-data.san\_francisco\_bikeshare.bikeshare\_trips` ORDER BY start\_date DESC LIMIT 3

Resultados de la consulta [GUARDAR RESULTADOS](#) [EXAMINAR LOS DATOS](#)

Se ha completado la consulta (tiempo transcurrido: 2,2 s; bytes procesados: 388,1 MB)

Información de la tarea [Resultados](#) [JSON](#) [Detalles de ejecución](#)

Fila	trip_id	duration_sec	start_date	start_station_name	start_station_id	end_date	end_station_name
1	13342018043023584500	256	2018-04-30 23:58:45 UTC	Telegraph Ave at 19th St	183	2018-05-01 00:03:01 UTC	Bay Pl at Vernon St
2	792018043023574300	95	2018-04-30 23:57:43 UTC	Jersey St at Castro St	137	2018-04-30 23:59:18 UTC	Jersey St at Church St
3	33562018043023565700	323	2018-04-30 23:56:57 UTC	Folsom St at 3rd St	36	2018-05-01 00:02:20 UTC	Victoria Manalo Draves Parl

Por alguna razón quieres obtener los id (trip\_id) y la duración (duration\_sec) de los 10 viajes de menos tiempo de duración hechos desde 7th St at Brannan St hasta Spear St at Folsom St ordenados de menor a mayor tiempo.

Utiliza SELECT, FROM, WHERE, ORDER BY, LIMIT y los operadores lógicos y de comparación para conseguir el resultado deseado.

```
1 SELECT trip_id, duration_sec
2 FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
3 WHERE start_station_id = 79 and end_station_id = 24
4 ORDER BY duration_sec ASC
5 LIMIT 10
```

# GROUP BY + Funciones de agregación

Las funciones de agregación realizan cálculos sobre conjuntos de 1 a N filas, y se suelen agrupar con GROUP BY para agrupar filas con valores iguales en algunos campos.

```
SELECT start_station_id, sum(duration_sec) as total_secs
FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
GROUP BY 1
```

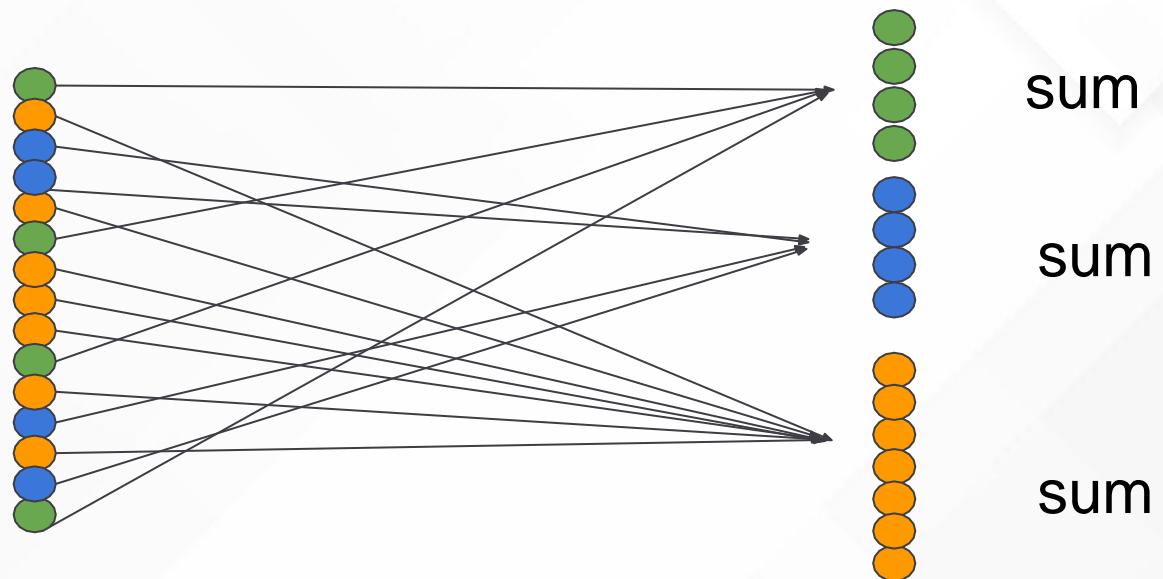
“GROUP BY 1” indica que se agrupará por el primer campo seleccionado, en este caso (`start_station_id`). También se puede poner el nombre del campo, pero cuando agrupamos por muchos campos empieza a ser complicado de mantener.

La función SUM(X) nos devuelve la suma de los valores.

Le ponemos el nombre que queramos a la columna resultado con “`as <nombre>`”

Fila	start_station_id	total_secs
1	4	12034279
2	33	8550395
3	13	7111307
4	10	9782134
5	32	4531904
6	80	4914987
7	84	7799177
8	8	15278854
9	0	11250450

GROUP BY separa las filas por el valor de las columnas que especifiquemos para posteriormente realizar los cálculos



# Funciones de agregación más comunes

SUM() -> Devuelve el valor total de sumar todas las filas

MAX() -> Devuelve el máximo valor

MIN() -> Devuelve el mínimo valor

AVG() -> Devuelve el valor medio

COUNT() -> Cuenta el número de filas. COUNT(DISTINCT X) Cuenta el número de valores distintos

# Podemos agrupar y calcular tantos campos como queramos

```
SELECT start_station_id, subscriber_type, sum(duration_sec) as total_secs, count(distinct trip_id ) as trips
FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
GROUP BY 1,2
ORDER BY 1,2
```

Fila	start_station_id	subscriber_type	total_secs	trips
1	2	Customer	3498726	845
2	2	Subscriber	7976987	12748
3	3	Customer	20102623	5723
4	3	Subscriber	9168991	14050
5	4	Customer	8755831	2447
6	4	Subscriber	3278448	6903

---

```
SELECT start_station_id, subscriber_type,
       sum(duration_sec) as total_secs,
       count(distinct trip_id ) as trips,
       sum(duration_sec)/count(distinct trip_id) as secs_per_trip
FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
GROUP BY 1,2
ORDER BY 1,2
```

Fila	start_station_id	subscriber_type	total_secs	trips	secs_per_trip
1	2	Customer	3498726	845	4140.504142011834
2	2	Subscriber	7976987	12748	625.7441951678695
3	3	Customer	20102623	5723	3512.602306482614
4	3	Subscriber	9168991	14050	652.5972241992882
5	4	Customer	8755831	2447	3578.190028606457
6	4	Subscriber	3278448	6903	474.93089960886573

Por alguna razón quieres obtener los id (start\_station\_id), la media de duración y el número de viajes de las 10 estaciones de inicio de viaje cuya media de duración de viaje sea más larga.

Utiliza SELECT, AVG(), COUNT(), FROM, GROUP BY, ORDER BY, LIMIT para conseguir el resultado deseado.

```
1 SELECT start_station_id,  
2     AVG(duration_sec) as average_duration,  
3     COUNT(trip_id) as number_of_trips  
4 FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`  
5 GROUP BY start_station_id  
6 ORDER BY average_duration DESC  
7 LIMIT 10
```

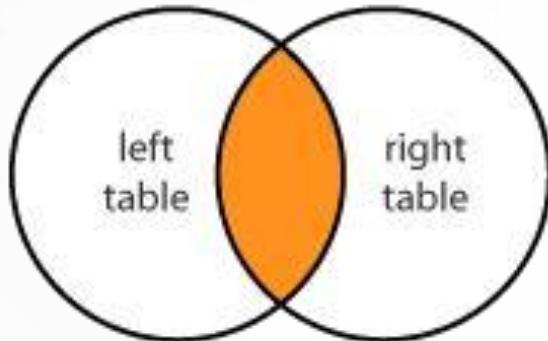
# Uniendo tablas con JOIN

```
SELECT t1.station_id,  
       t1.name as station_name,  
       t1.region_id,  
       t2.name as region_name  
FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_station_info` as t1  
LEFT JOIN `bigquery-public-data.san_francisco_bikeshare.bikeshare_regions` as t2  
ON t1.region_id = t2.region_id |
```

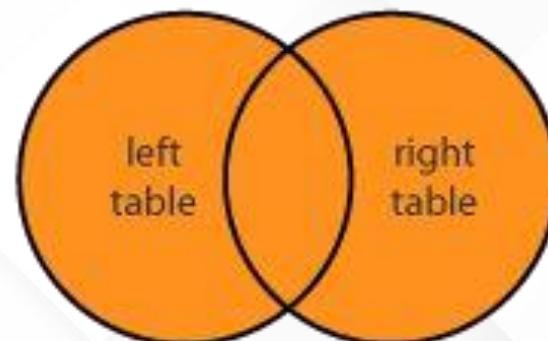
385	Woolsey St at Sacramento St	14	Berkeley
166	College Ave at Alcatraz Ave	14	Berkeley
251	California St at University Ave	14	Berkeley
254	Vine St at Shattuck Ave	14	Berkeley
256	Hearst Ave at Euclid Ave	14	Berkeley
257	Fifth St at Delaware St	14	Berkeley

# Tipos de JOINS

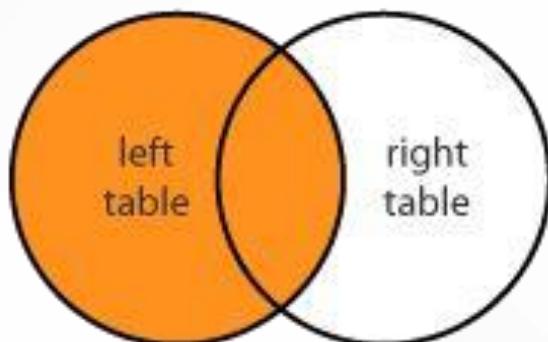
INNER JOIN



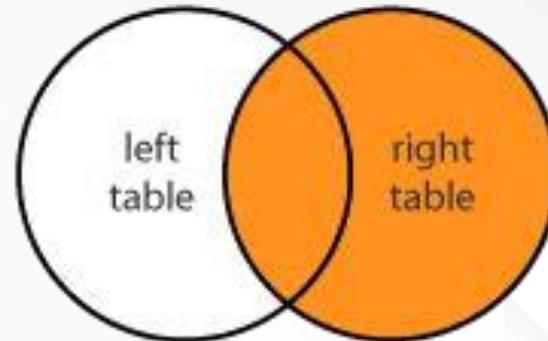
FULL JOIN



LEFT JOIN



RIGHT JOIN



## *Tipos de JOIN*

- INNER JOIN: Devuelve todas las filas cuando hay al menos una coincidencia en ambas tablas.
- LEFT JOIN: Devuelve todas las filas de la tabla de la izquierda, y las filas coincidentes de la tabla de la derecha.
- RIGHT JOIN: Devuelve todas las filas de la tabla de la derecha, y las filas coincidentes de la tabla de la izquierda.
- OUTER JOIN: Devuelve todas las filas de las dos tablas, la izquierda y la derecha. También se llama FULL JOIN.

Por alguna razónquieres obtener una tabla con el número de bicis disponibles en cada estación y el nombre de la estación correspondiente. En tu tabla soloquieres queaparezcan las 20 estaciones con mayor número de bicis disponibles.

Usa SELECT, FULL JOIN , ORDER BY Y LIMIT

---

```
1 SELECT t1.num_bikes_available as bikes_available,
2 t2.name as name,
3 FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_station_status` as t1
4 FULL JOIN `bigquery-public-data.san_francisco_bikeshare.bikeshare_station_info` as t2
5 ON t1.station_id = t2.station_id
6 ORDER BY bikes_available DESC
7 LIMIT 20
```

# ATENCIÓN

El JOIN es la operación más compleja en cuanto a procesamiento de todo SQL, ya que tiene que recorrer las dos tablas para encontrar las coincidencias.

Utilizar el JOIN con cabeza y tratar de mezclar solo la información necesaria.

# Ejercicios

# Ejercicio 1

Consigue las 5 estaciones desde las cuales han empezado más viajes distintos y las 5 estaciones de inicio que acumulan más segundos de viaje totales.

# Ejercicio 1 Soluciones

```
SELECT start_station_id, start_station_name, count(distinct trip_id)
as trips
FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
GROUP BY 1,2
ORDER BY 3 DESC
LIMIT 5
```

start_station_id	start_station_name	trips
70	San Francisco Caltrain (Townsend at 4th)	72683
69	San Francisco Caltrain 2 (330 Townsend)	56100
50	Harry Bridges Plaza (Ferry Building)	49062
60	Embarcadero at Sansome	41137
61	2nd at Townsend	39936

```
SELECT start_station_id, start_station_name, sum( duration_sec) as secs
FROM `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
GROUP BY 1,2
ORDER BY 3 DESC
LIMIT 5
```

Fila	start_station_id	start_station_name	secs
1	50	Harry Bridges Plaza (Ferry Building)	66941520
2	70	San Francisco Caltrain (Townsend at 4th)	58499154
3	60	Embarcadero at Sansome	54618230
4	69	San Francisco Caltrain 2 (330 Townsend)	38209059
5	66	South Van Ness at Market	36648693

# Ejercicio 2

Consigue el número de viajes realizados año a año ordenado de menor a mayor

En este ejercicio tendrás que utilizar la función EXTRACT. Con ella podrás extraer un elemento del dato temporal.

En este caso, si en tu consulta escribes *EXTRACT (YEAR FROM start\_date) AS year*, crearás una columna con solo el año de cada una de las start dates.

# Ejercicio 2 -Solucion

```
SELECT
    EXTRACT(YEAR
    FROM
        start_date) AS year,
    COUNT(DISTINCT trip_id) AS trips
FROM
    `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
GROUP BY 1
ORDER BY 1
```

Fila	year	trips
1	2013	100563
2	2014	326339
3	2015	346251
4	2016	210493
5	2017	519700
6	2018	444071

# Ejercicio 3

Cuales son los 10 trayectos (Estacion Inicio - Estacion Fin) más realizados?

# Ejercicio 3 - Solución

```

SELECT
    start_station_id, start_station_name, end_station_id, end_station_name, count(distinct trip_id) as trips
FROM
    `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
GROUP BY 1,2,3,4
ORDER BY 5 DESC
LIMIT 10

```

Fila	start_station_id	start_station_name	end_station_id	end_station_name	trips
1	50	Harry Bridges Plaza (Ferry Building)	60	Embarcadero at Sansome	9150
2	69	San Francisco Caltrain 2 (330 Townsend)	65	Townsend at 7th	8508
3	61	2nd at Townsend	50	Harry Bridges Plaza (Ferry Building)	7620
4	50	Harry Bridges Plaza (Ferry Building)	61	2nd at Townsend	6888
5	60	Embarcadero at Sansome	74	Steuart at Market	6874
6	65	Townsend at 7th	69	San Francisco Caltrain 2 (330 Townsend)	6836
7	51	Embarcadero at Folsom	70	San Francisco Caltrain (Townsend at 4th)	6351
8	70	San Francisco Caltrain (Townsend at 4th)	50	Harry Bridges Plaza (Ferry Building)	6215
9	74	Steuart at Market	61	2nd at Townsend	6039
10	74	Steuart at Market	70	San Francisco Caltrain (Townsend at 4th)	5959

# Ejercicio 4

Ordena las regiones por mayor a menor número de trips.

Pista: necesitarás JOINEAR hasta 3 tablas

# Ejercicio 4 - Solución

```
SELECT
    t3.region_id,
    t3.name,
    count(distinct t1.trip_id) as trips
FROM
    `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips` as t1
    LEFT JOIN `bigquery-public-data.san_francisco_bikeshare.bikeshare_station_info` as t2 ON t1.start_station_id = t2.station_id
    LEFT JOIN `bigquery-public-data.san_francisco_bikeshare.bikeshare_regions` AS t3 ON t2.region_id = t3.region_id
GROUP BY 1,2
ORDER BY 3 DESC
```

Fila	region_id	name	trips
1	3	San Francisco	1446671
2	null	null	265219
3	12	Oakland	144522
4	5	San Jose	43173
5	14	Berkeley	40619
6	13	Emeryville	7213

Se ha completado la consulta (tiempo transcurrido: 5,2 s; bytes procesados: 52,6 MB)

# Ejercicio 5

Ahora vamos a utilizar el dataset de Stackoverflow

Cuales son los usuarios que acumulan más favs en sus posts?

▼	stackoverflow
└	badges
└	comments
└	post_history
└	post_links
└	posts_answers
└	posts_moderator_nomination
└	posts_orphaned_tag_wiki
└	posts_privilege_wiki
└	posts_questions
└	posts_tag_wiki
└	posts_tag_wiki_excerpt
└	posts_wiki_placeholder
└	stackoverflow_posts
└	tags
└	users
└	votes

# Ejercicio 5 - Solución

```
SELECT owner_display_name, sum(favorite_count) as favs FROM `bigquery-public-data.stackoverflow.stackoverflow_posts` GROUP BY 1 ORDER BY 2 DESC
```

Fila	owner_display_name	favs
1	null	5911958
2	e-satis	6953
3	gropsedawk	6824
4	anon	5307
5	endpoint	5247
6	jelovirt	4835
7	buyutec	4712
8	Adam Davis	4568
9	Tim	3933
10	hmason	3761
11	matt	3489
12	Alex. S.	3429

# Ejercicio 6

Basándonos en la información de la tabla users, en la que tenemos la localización de cada usuario...

¿Cuáles son las localizaciones que acumulan más posts?

# Ejercicio 6 Solucion

```
SELECT
    t2.location,
    sum(t1.posts) as posts
FROM
    (SELECT owner_user_id, count(distinct id) as posts FROM `bigquery-public-data.stackoverflow.stackoverflow_posts` GROUP BY 1) as t1
    LEFT JOIN `bigquery-public-data.stackoverflow.users` as t2 on t1.owner_user_id=t2.id
GROUP BY 1
ORDER BY 2 DESC
```

Fila	location	posts
1	null	12113807
2	India	416006
3	London, United Kingdom	407872
4	United States	407567
5	United Kingdom	376742
6	Germany	350254
7	Bangalore, India	191103
8	Netherlands	190971
9	France	165465

Si tenéis alguna duda o si queréis profundizar en algún tema podéis poneros en contacto conmigo a través de:

- LinkedIn: <https://www.linkedin.com/in/antoniolopezrosell/>
- Email: [tonilopezrosell@gmail.com](mailto:tonilopezrosell@gmail.com)

El contenido de esta sesión está subido a un repositorio de mi cuenta de Github:

- <https://github.com/tonilopezrosell/datasciencemasterthevalley>