

STRUCTURED QUERY LANGUAGE

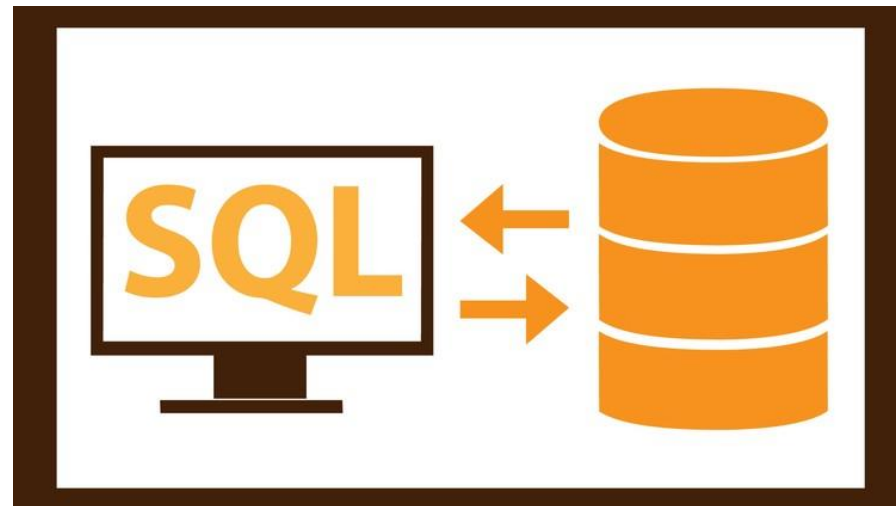
Christian Camilo Urcuqui López, MSc



STRUCTURED QUERY LANGUAGE (SQL)

Es un lenguaje diseñado para administrar, y recuperar información de sistemas de gestión de **base de datos relacionales**.

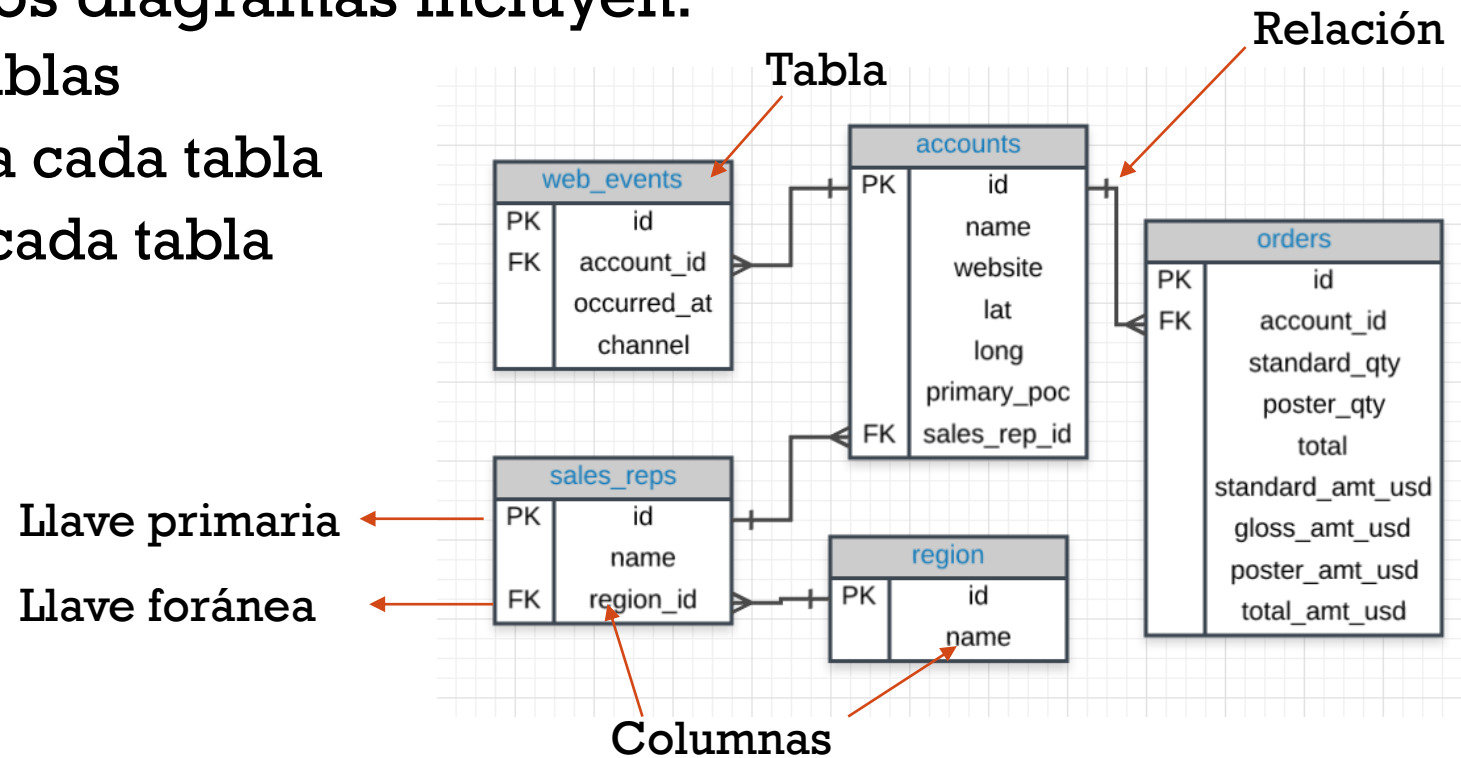
Basado en el álgebra relacional y en el cálculo relacional, SQL es un lenguaje de definición, manipulación, consultas y control de datos.



BASE DE DATOS RELACIONALES

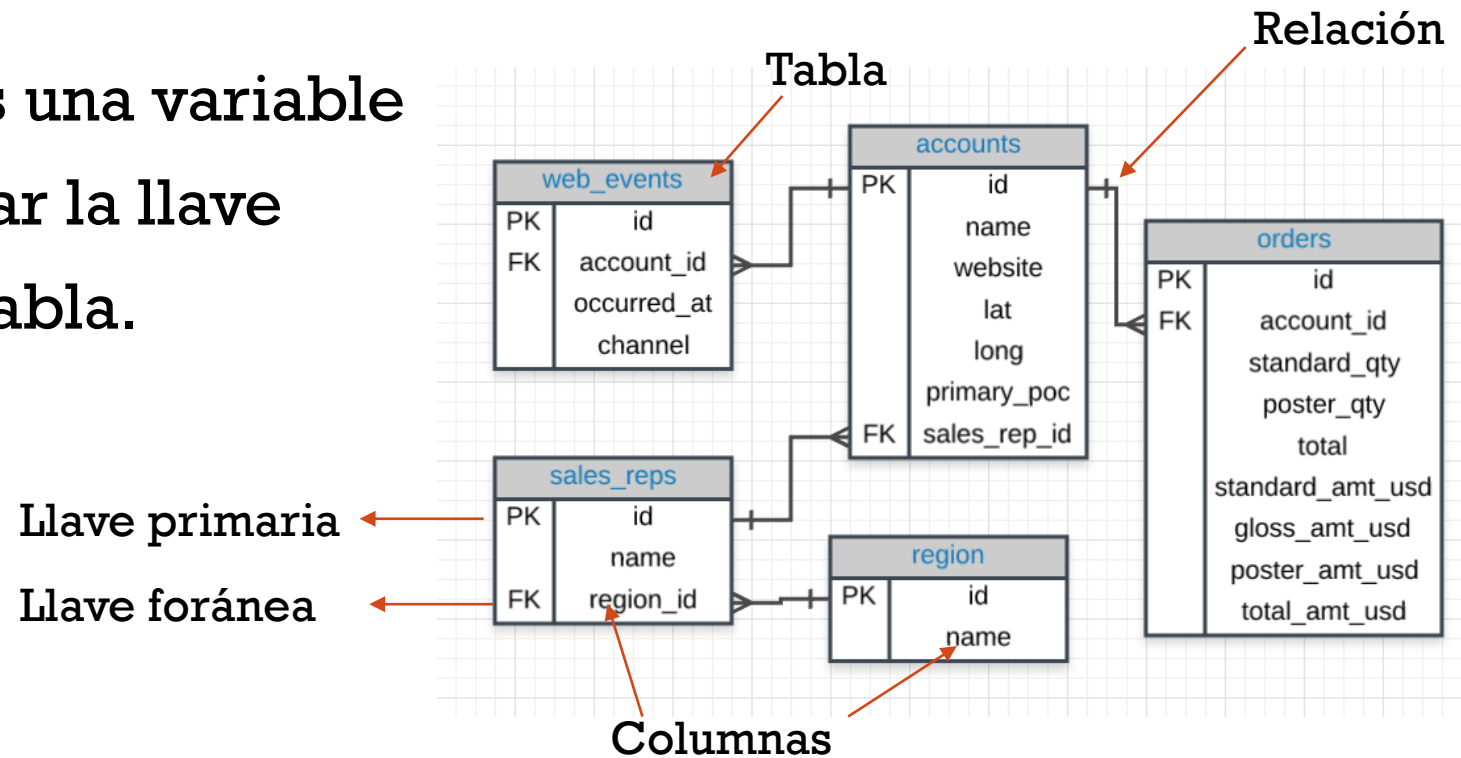
- Los diagramas de entidades de relaciones (Entity Relationship Diagrams, ERD) nos permiten ver la estructura de una base de datos relacional. Los diagramas incluyen:

- Nombres de las tablas
- Las columnas para cada tabla
- La relación entre cada tabla



BASE DE DATOS RELACIONALES

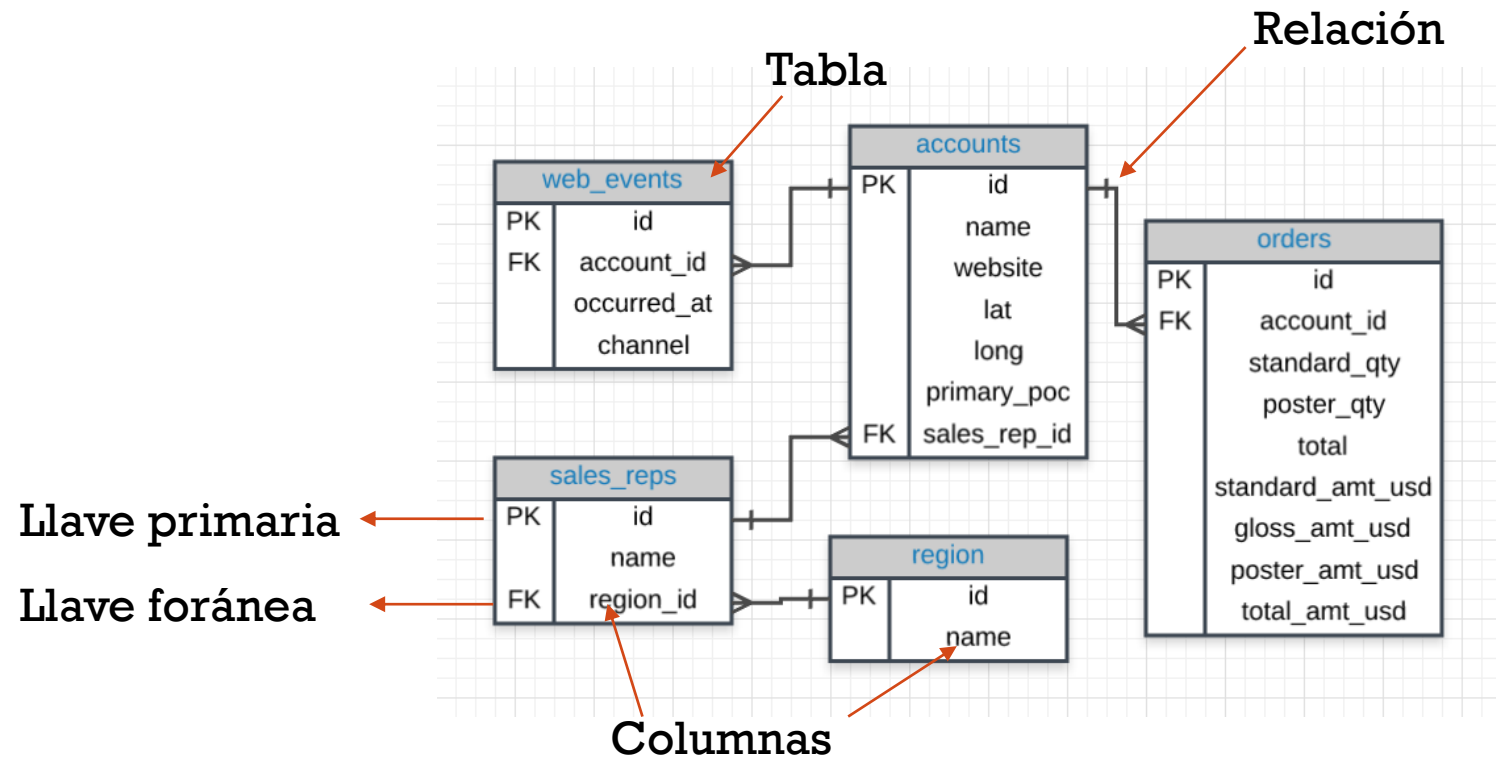
- La llave primaria (Primary Key) es la variable cuyos valores son únicos e irrepetibles en la tabla, es decir, identifican a cada registro.
- La llave foránea es una variable que permite asociar la llave primaria de otra tabla.



BASE DE DATOS RELACIONALES

Las relaciones entre tablas pueden ser de tres tipos:

- Uno a uno
- Uno a muchos
- Muchos a muchos



BASE DE DATOS RELACIONALES

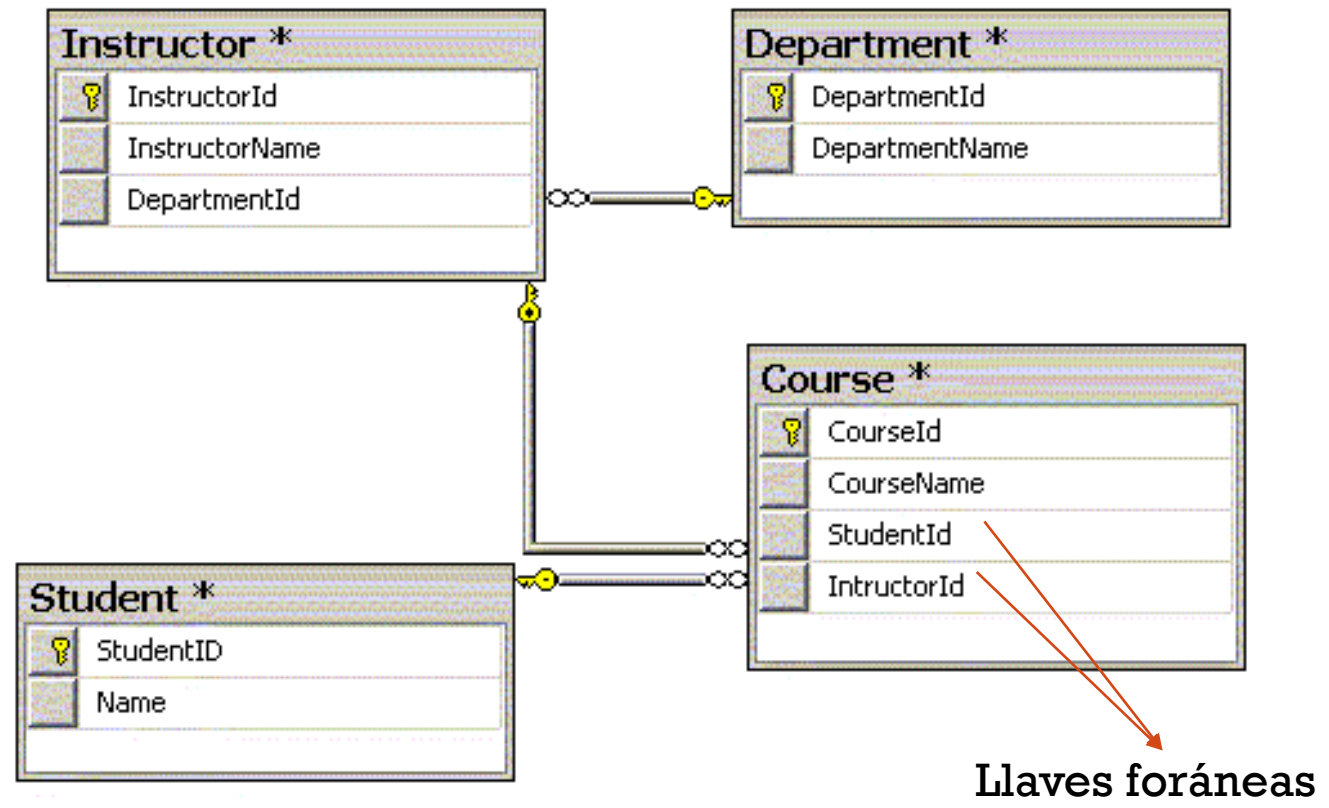
Suponga el siguiente caso, la universidad Icesi requiere desarrollar un sistema de base de datos que permita almacenar la siguiente información:

- Datos de los departamentos: *nombre*
- Datos de los profesores asociados a un departamento: *nombre y cédula*.
- Datos de los estudiantes: *nombre*.
- Datos de los cursos que ha dictado cada profesor y que han visto los estudiantes: *nombre del curso*.

BASE DE DATOS RELACIONALES

Suponga el siguiente caso, la universidad Icesi requiere desarrollar un sistema de base de datos que permita almacenar la siguiente información:

- Datos de los departamentos: *nombre*
- Datos de los profesores asociados a un departamento: *nombre y cédula*.
- Datos de los estudiantes: *nombre*.
- Datos de los cursos que ha dictado cada profesor y que han visto los estudiantes: *nombre del curso*.



BASE DE DATOS RELACIONALES

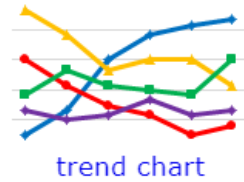
id	name	website	lat	long	primary_poc	sales_rep_id
1001	Walmart	www.walmart.com	40.2384956	-75.1032974	Tamara Tuma	321500
1011	Exxon Mobil	www.exxonmobile.com	41.1691563	-73.8493737	Sung Shields	321510
1021	Apple	www.apple.com	42.2904481	-78.0840094	Jodee Lupo	321520
1031	Berkshire Hathaway	www.berkshirehathaway.com	40.9490213	-75.2849983	Serafina Banda	321530
1031	McKesson	www.mckesson.com	42.2170934	-75.2849982	Angela Crusoe	321540
1051	UnitedHealth Group	www.unitedhealthgroup.com	40.0879254	-73.7676353	Bavanna Gayman	321550
1061	CVS Health	www.cvshealth.com	41.4677952	-76.7101814	Anabel Haskell	321560
1071	General Motors	www.gm.com	40.8055176	-76.7101814	Barrie Omeara	321560
1081	Ford Motor	www.ford.com	41.1139402	-75.8542245	Kym Hagerman	321570
1091	AT&T	www.att.com	42.4974627	-74.9027122	Jamel Mosqueda	321580

Stores
id
name
website
lat
long
primary_poc
sales_rep_id

DB-Engines Ranking

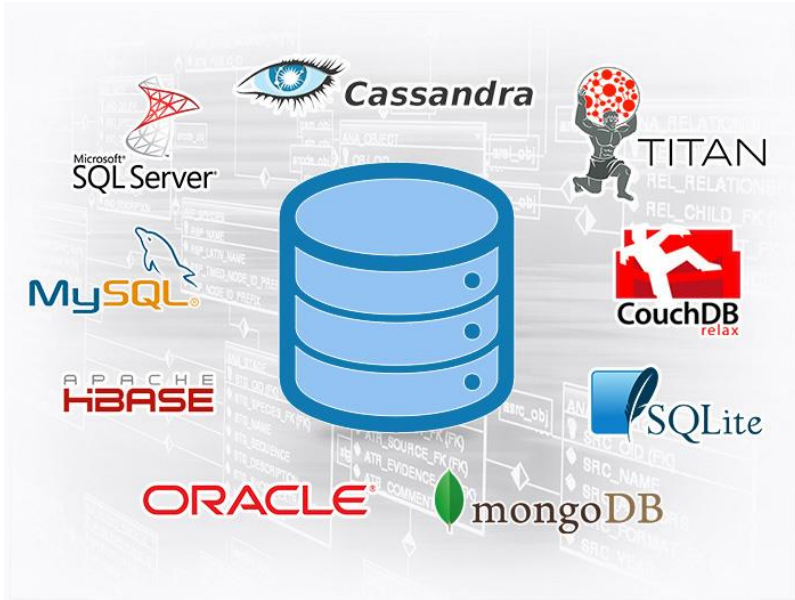
The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the [method](#) of calculating the scores.



345 systems in ranking, April 2019

Rank			DBMS	Database Model	Score		
Apr 2019	Mar 2019	Apr 2018			Apr 2019	Mar 2019	Apr 2018
1.	1.	1.	Oracle +	Relational, Multi-model i	1279.94	+0.80	-9.85
2.	2.	2.	MySQL +	Relational, Multi-model i	1215.14	+16.89	-11.26
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model i	1059.96	+12.11	-35.55
4.	4.	4.	PostgreSQL +	Relational, Multi-model i	478.72	+8.91	+83.25
5.	5.	5.	MongoDB +	Document	401.98	+0.64	+60.57
6.	6.	6.	IBM Db2 +	Relational, Multi-model i	176.05	-1.15	-12.89
7.	↑ 8.	↑ 9.	Redis +	Key-value, Multi-model i	146.38	+0.25	+16.27
8.	↑ 9.	8.	Elasticsearch +	Search engine, Multi-model i	146.00	+3.21	+14.64
9.	↓ 7.	↓ 7.	Microsoft Access	Relational	144.65	-1.55	+12.43
10.	10.	↑ 11.	SQLite +	Relational	124.21	-0.66	+8.23
11.	11.	↓ 10.	Cassandra +	Wide column	123.61	+0.81	+4.52
12.	12.	↑ 14.	MariaDB +	Relational, Multi-model i	85.23	+0.92	+20.67
13.	13.	13.	Splunk	Search engine	83.09	-0.01	+18.03
14.	14.	↓ 12.	Teradata +	Relational	75.35	+0.13	+1.67
15.	15.	↑ 18.	Hive +	Relational	74.71	+1.71	+17.31
16.	16.	↓ 15.	Solr	Search engine	60.22	+0.21	-2.99
17.	17.	17.	HBase	Wide column	58.66	-0.13	-1.03



BASE DE DATOS RELACIONALES

ORACLE

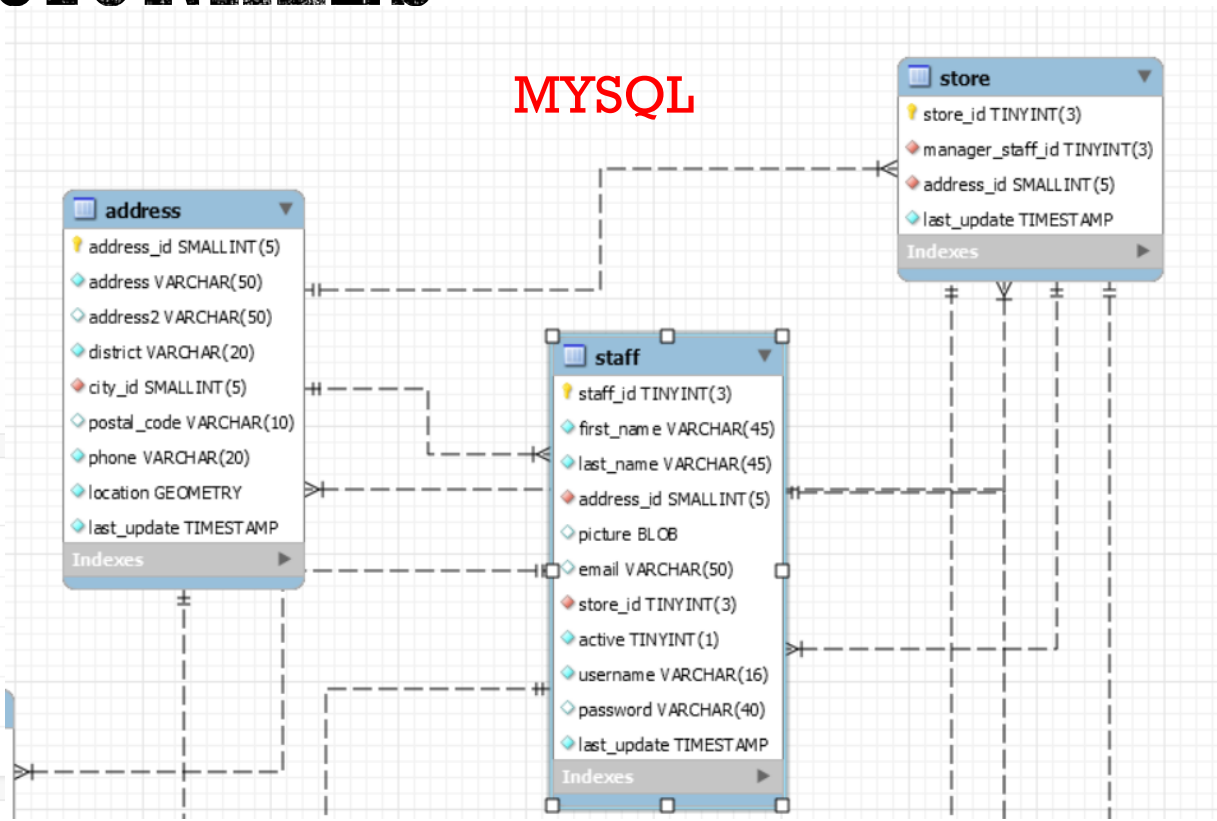
PRODUCT_DIM

Show All Table Attributes Columns Indexes Triggers Constraints

Table Attributes	
Table Name	PRODUCT_DIM
Status	VALID
Temporary	No
Nested	No
Owner	AV

Columns							
#	Column	Type	Length	Precision	Scale	Nullable	
1	DEPARTMENT_ID	NUMBER	22			Yes	
2	DEPARTMENT_NAME	VARCHAR2	100			Yes	Byte
3	CATEGORY_ID	NUMBER	22			Yes	
4	CATEGORY_NAME	VARCHAR2	100			Yes	Byte

MYSQL



SENTENCIAS

- **CREATE**

- Nos permite crear una nueva tabla en nuestra base de datos

- **DROP TABLE**

- Remueve una tabla de la base de datos

- **SELECT**

- Nos permite seleccionar y leer datos. Son conocidas como *queries*.

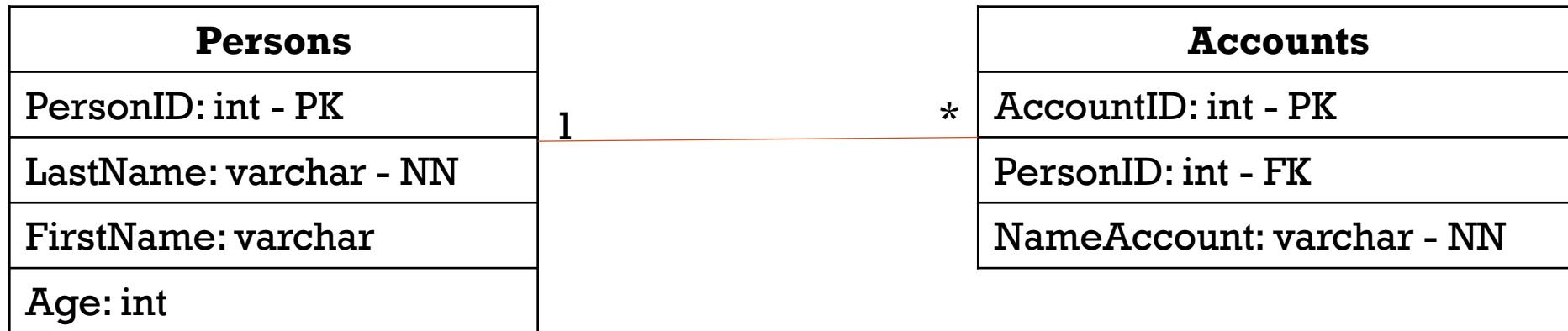
CREAR Y BORRAR BASES DE DATOS

```
CREATE TABLE Persons (  
    PersonID int NOT NULL PRIMARY KEY,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
);
```

Persons
PersonID: int - PK
LastName: varchar - NN
FirstName: varchar
Age: int

https://www.w3schools.com/sql/trysql.asp?filename=try_sql_create_table

CREAR Y BORRAR BASES DE DATOS

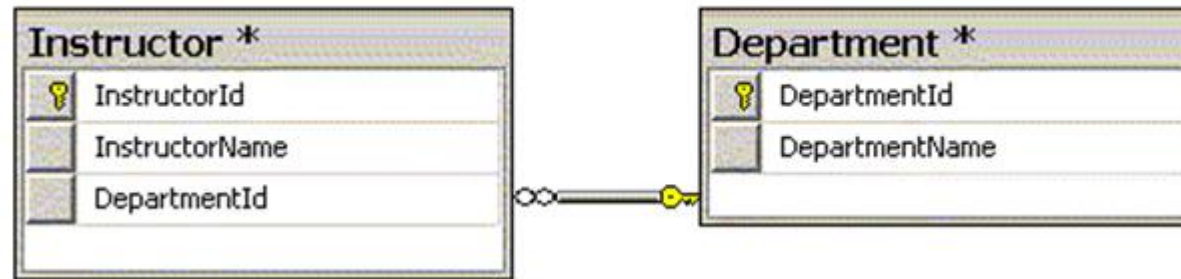


https://www.w3schools.com/sql/trysql.asp?filename=trysql_create_table

```
CREATE TABLE Accounts (  
  AccountID int,  
  NameAccount varchar(255) NOT NULL,  
  PersonID int NOT NULL,  
  PRIMARY KEY (AccountID),  
  FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);
```

EJERCICIOS

Construya la siguiente bases de datos



https://www.w3schools.com/sql/trysql.asp?filename=try_sql_create_table

INSERTAR DATOS

INSERT INTO

Es una sentencia que nos permite ingresar nuevos datos a una tabla. Existen dos formas de agregar información, estas son:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Si vamos a adicionar datos a todas las columnas, no es necesario especificar los nombres de las columnas en una query SQL.

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

EJERCICIO

Ingresa los siguientes valores a un primer registro a las tablas de Instructor y Department.

Department

DepartmentId: 102

DepartmentName: "ingeniería"

Instructor

InstructorId: 6178

InstructorName: "Camilo"

DepartmentId: 102

SELECT

- **SELECT**

- Indica que columna o las columnas que queremos obtener información.

- **FROM**

- Especifica la tabla o las tablas las que deseamos procesar.

- Si queremos obtener todas las variables (columnas) de una o más tablas utilizamos *, por ejemplo:

Select * FROM orders

Obtenemos todos los registros de la tabla *orders*

EJERCICIOS

- Desarrollemos los primeros dos ejercicios de la siguiente página web.

https://www.w3schools.com/sql/exercise.asp?filename=exercise_select1

SINTAXIS Y SEMÁNTICA

- Las queries de SQL no son case-sensitive, es decir, las sentencias pueden estar escritas tanto en mayúsculas y en minúsculas.

```
SELECT account_id  
FROM orders
```

=

```
select account_id  
from orders
```

=

```
SeLeCt AcCoUnt_id  
FrOm oRdErS
```

- Sin embargo, las palabras reservadas de SQL (por ejemplo, SELECT, FROM, entre otras) se suelen escribir en mayúsculas y lo que no hace parte del lenguaje en minúscula.

SINTAXIS Y SEMÁNTICA

- Es común no utilizar espacios en los nombres de las columnas y es por ello que se utiliza guion bajo “_” como conector entre las palabras del nombre.

Por ejemplo el nombre “account id” se escribe “account_id”

- Puede utilizar los espacios de línea siempre y cuando no altere la sintaxis del lenguaje SQL.

SELECT account_id FROM orders



SELECT account_id
FROM orders



SEL
ECT account_id
FR
OM orders



SINTAXIS Y SEMÁNTICA

- Las mejores practicas de programación recomiendan siempre terminar una query con punto y coma “;” con el fin de independizar cada sentencia y así mismo ejecutar múltiples queries.

```
SELECT account_id  
FROM orders;
```

CLAUSULAS EN QUERIES

Para casos con grandes volúmenes de información la aplicación de queries puede ser un poco compleja, si se desease obtener una muestra de estos datos podríamos utilizar un conjunto de clausulas con el fin de optimalizar nuestras búsquedas.

Nota:

No todos los sistemas de base de datos soportan la clausula **SELECT TOP**. MySQL implementa **LIMIT** para seleccionar un número limitado de registros, mientras que ORACLE utiliza **ROWNUM**.

EJERCICIOS

- https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_top&ss=-1
- https://www.w3schools.com/sql/trymysql.asp?filename=trysql_select_limit

CLAUSULAS EN QUERIES

ORDER BY

- Es una clausula que nos permite ordenar los resultados de una consulta de cualquier columna.
- Esta sentencia siempre va luego de el **SELECT** y el **FROM**, pero antes que el **LIMIT**.

```
SELECT column1, column2, ...  
FROM table_name  
ORDER BY column1, column2, ... ASC|DESC;
```

- Preste atención al ASC y al DESC que definen la forma de serán ordenados los datos

EJERCICIOS

- Desarrolle el query que permita obtener los 10 primeros registros de employees ordenados por su LastName.

https://www.w3schools.com/sql/trysql.asp?filename=trysql_select_orderby

- Implemente un query que permita obtener los últimos 5 registros de OrderDetails ordenados por su Quantity.
- Implemente un query que permita obtener los 10 primeros dígitos de Orders ordenados de forma ascendente por OrderDate y ShipperID.

CLAUSULAS EN QUERIES

WHERE

- Es una sentencia que nos permite obtener subconjuntos de datos a partir de una o varias condiciones, es decir, realizar un filtro en la información.
- Las condiciones están compuestas por operadores lógicos entre los que podemos encontrar:

>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
=	Igual
!=	Diferencia

CLAUSULAS EN QUERIES

WHERE

```
SELECT *  
FROM Customers  
WHERE Country='Mexico';
```

Obtenemos todos los registros de la tabla Customer que tengan como valor en la variable Country igual a Mexico

EJERCICIOS

- Desarrolle un query que permita obtener los datos de las variables CustomerName y el ContactName de la tabla Customers siempre y cuando la ciudad sea 'London'.
- Desarrolle un query que permita obtener las ordenes cuya cantidad sea mayor o igual a 10.
- Obtenga las 5 primeros registros de las ordenes cuyo cliente sea igual a 20.

CLAUSULAS EN QUERIES

WHERE con datos categóricos

- Para este tipo de datos podemos utilizar los operadores lógicos = y != (vistos previamente).
- Pero, también podríamos utilizar los siguientes operadores junto con WHERE:

LIKE, es un operador utilizado para buscar un patrón específico.

NOT, representa la condición NOT TRUE.

IN, es un operador que compara un valor en múltiples variables o en el resultado de otra query.

CLAUSULAS EN QUERIES

LIKE

El operador LIKE es usado en la clausula WHERE para buscar un patrón específico en una columna. A continuación, se presentan los comodines utilizados con LIKE:

- % - el porcentaje representa el cero, uno, o múltiples caracteres
- _ - representa a un solo carácter.

```
SELECT column1, column2, ...  
FROM table_name  
WHERE columnN LIKE pattern;
```

CLAUSULAS EN QUERIES

LIKE

WHERE CustomerName LIKE 'a%'	Encuentra todos los valores que inician con “a”
WHERE CustomerName LIKE '%a'	Encuentra a todos los valores que terminan con “a”
WHERE CustomerName LIKE '%or%'	Retorna todos los valores que tienen “or” en cualquier posición
WHERE CustomerName LIKE '_r%'	Retorna todos los valores donde “r” esta en la segunda posición
WHERE CustomerName LIKE 'a_%_ %'	Encuentra todos los valores que inician con “a” y como mínimo tienen tres caracteres
WHERE ContactName LIKE 'a%o'	Retorna los valores que inician con “a” y terminan con “o”

EJERCICIOS

- Retorne todos los registros de clientes cuyos nombres comiencen con “a” y terminen con “n”
- Retorne los nombres de los productos que tengan la palabra “Queso”

CLAUSULAS EN QUERIES

IN

Es un operador que compara un valor en múltiples variables o en el resultado de otra query.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

CLAUSULAS EN QUERIES

Retorne los registros de los clientes que se encuentran en los países de cada proveedor.

```
SELECT * FROM Customers  
WHERE Country IN  
(SELECT Country FROM Suppliers);
```

EJERCICIO

- Retorne todos los clientes que se encuentran en “Germany”, “London” y “Berlin”

OPERACIONES ARITMÉTICAS

Es posible aplicar operaciones aritméticas a los resultados de las queries, por ejemplo:

```
SELECT Price, Price *2 FROM [Products]
```

La anterior operación nos dará un nombre de una columna que puede ser cambiada a través de la asignación de alias, es decir...

```
SELECT Price, Price *2 AS Double_price FROM [Products]
```

OPERACIONES ARITMÉTICAS

Algunas de las operaciones aritméticas que podemos aplicar son:

- * (Multiplicación)
- + (Adición)
- - (substracción)
- / (división)

OPERACIONES LÓGICAS

Algunas de las operaciones lógicas que podemos aplicar son:

LIKE, es un operador utilizado para buscar un patrón específico.

NOT, representa la condición NOT TRUE.

IN, es un operador que compara un valor en múltiples variables o en el resultado de otra query

AND & BETWEEN, permite combinar operaciones donde su resultado debe ser TRUE

OR, nos permite realizar operaciones donde al menos uno de estos sea TRUE

OPERACIONES LÓGICAS

AND & BETWEEN, permite combinar operaciones donde su resultado debe ser TRUE

OR, nos permite realizar operaciones donde al menos uno de estos sea TRUE

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

OPERACIONES LÓGICAS

AND

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 AND condition2 AND condition3 ...;
```

OR

```
SELECT column1, column2, ...  
FROM table_name  
WHERE condition1 OR condition2 OR condition3 ...;
```

NOT

```
SELECT column1, column2, ...  
FROM table_name  
WHERE NOT condition;
```


OPERACIONES LÓGICAS

Obtenga los registros de los clientes donde el país sea Germany y las ciudades de este sean Berlin o München

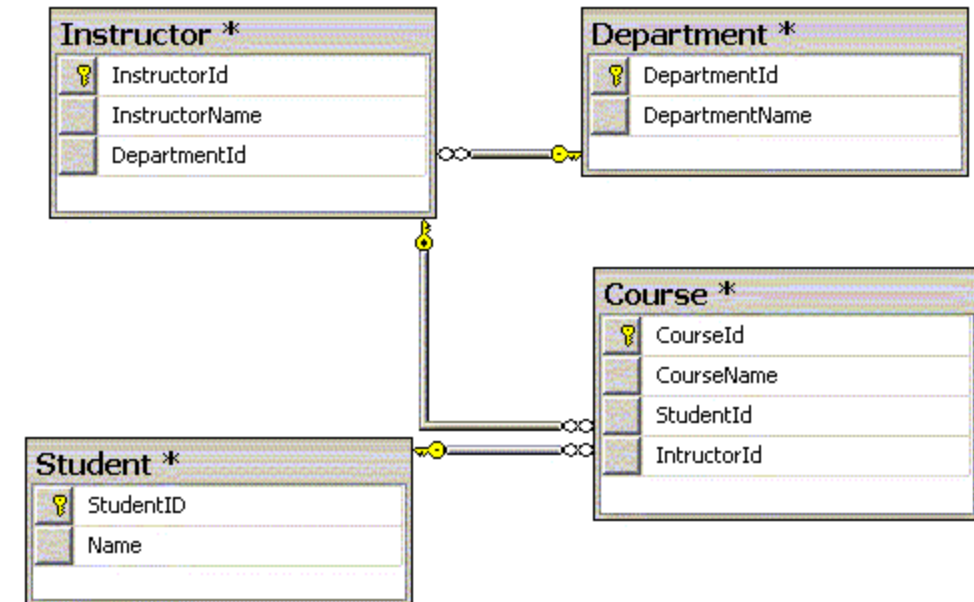
```
SELECT * FROM Customers  
WHERE  
Country='Germany'  
AND  
(City='Berlin' OR City='München');
```

JOINS

Una clausula JOIN nos permite combinar filas de una o más tablas a partir de una relación entre estas.

Para aplicar estas operaciones es importante conocer como acceder desde un conjunto de datos a otro, por ejemplo:

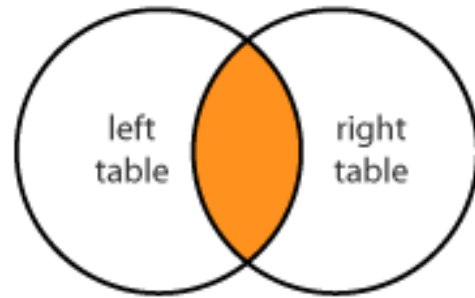
¿Cómo es posible conocer cuantos estudiantes han visto clases de un departamento?



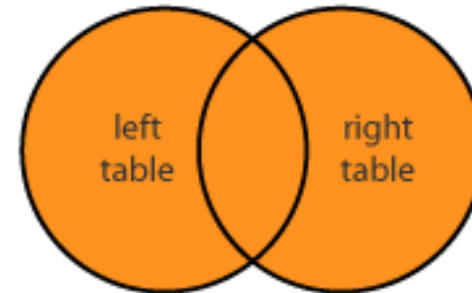
JOINS

- Siempre debe existir una variable que permita hacer de enlace entre dos conjuntos

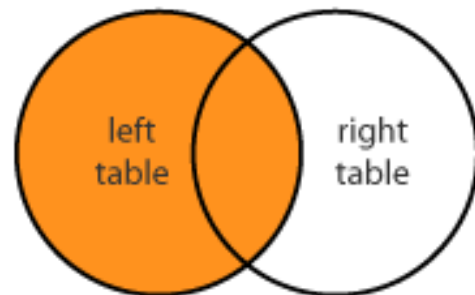
INNER JOIN



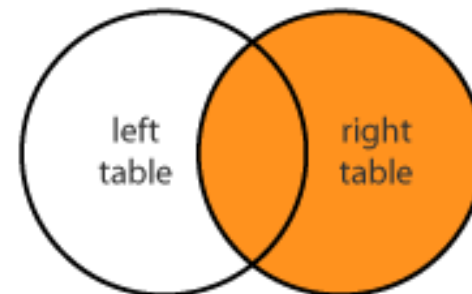
FULL JOIN



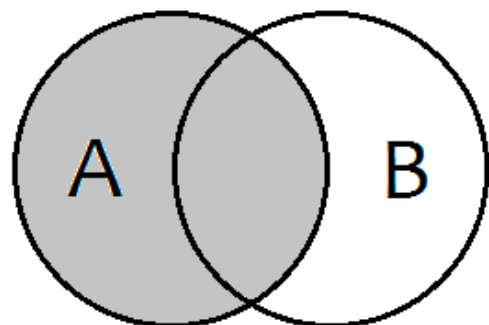
LEFT JOIN



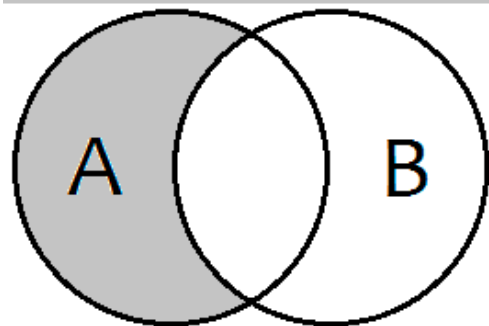
RIGHT JOIN



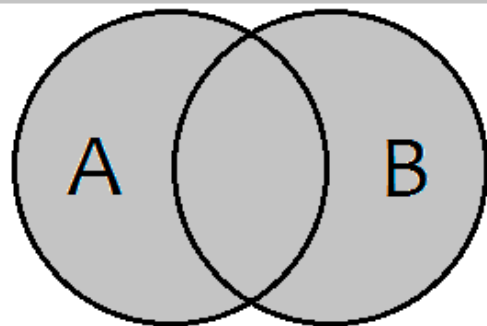
SQL JOINS



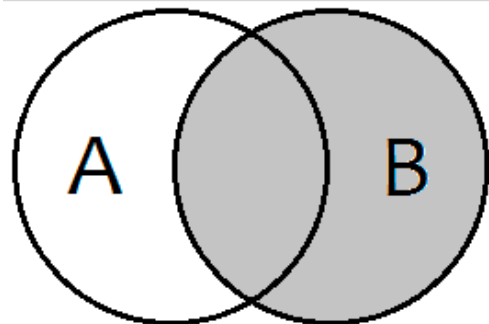
```
SELECT *  
FROM TableA a  
LEFT JOIN TableB b  
ON a.Key = b.Key
```



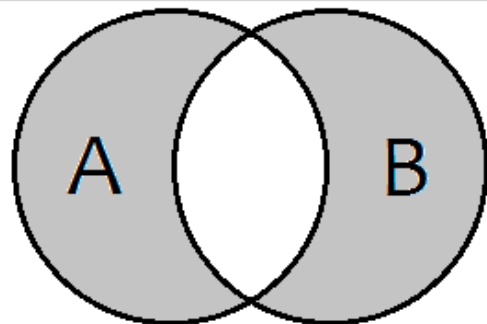
```
SELECT *  
FROM TableA a  
LEFT JOIN TableB b  
ON a.Key = b.Key  
WHERE b.Key IS NULL
```



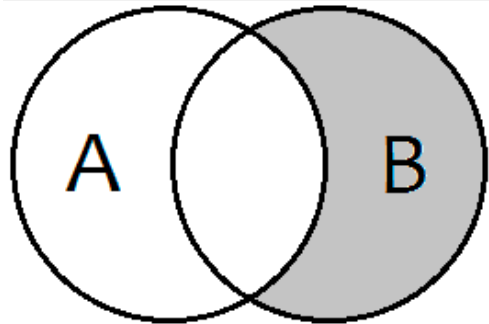
```
SELECT *  
FROM TableA a  
FULL OUTER JOIN TableB b  
ON a.Key = b.Key
```



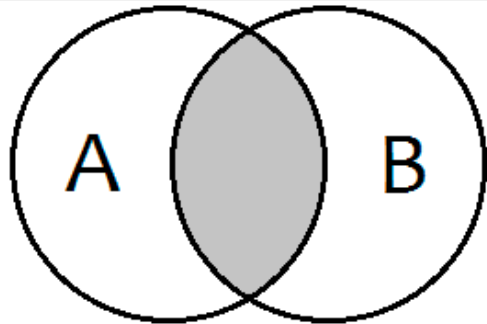
```
SELECT *  
FROM TableA a  
RIGHT JOIN TableB b  
ON a.Key = b.Key
```



```
SELECT *  
FROM TableA a  
FULL OUTER JOIN TableB b  
ON a.Key = b.Key  
WHERE a.Key IS NULL  
OR b.Key IS NULL
```

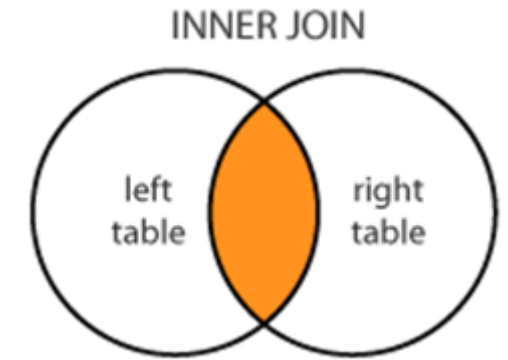


```
SELECT *  
FROM TableA a  
RIGHT JOIN TableB b  
ON a.Key = b.Key  
WHERE a.Key IS NULL
```



```
SELECT *  
FROM TableA a  
INNER JOIN TableB b  
ON a.Key = b.Key
```

INNER JOIN



- Retorna los registros que son iguales en ambas tablas

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Customer
CustomerID: int - PK
CustomerName: varchar

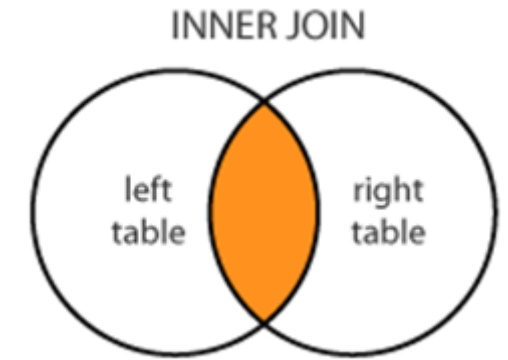
1

*

Orders
OrderID: int - PK
CustomerID: int - FK
OrderDate: varchar - NN

CustomerID	CustomerName
1	Alfreds Futterkiste
2	Ana Trujillo Emparedados y helados
3	Antonio Moreno Taquería

INNER JOIN



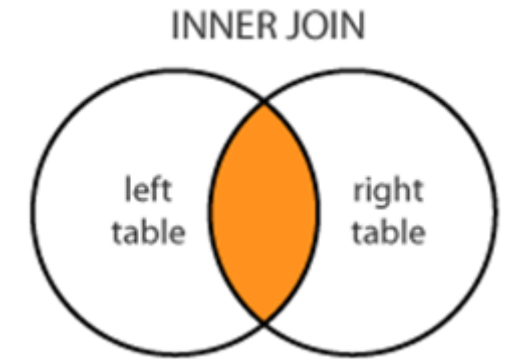
- Retorna los registros que son iguales en ambas tablas

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

```
SELECT Orders.OrderID, Customers.CustomerName,  
Orders.OrderDate  
FROM Orders  
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

CustomerID	CustomerName
1	Alfreds Futterkiste
2	Ana Trujillo Emparedados y helados
3	Antonio Moreno Taquería

INNER JOIN



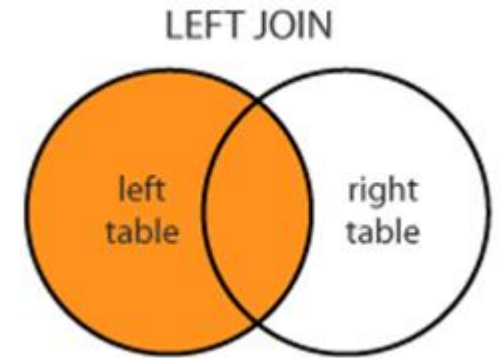
- Retorna los registros que son iguales en ambas tablas

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CustomerID	CustomerName
1	Alfreds Futterkiste
2	Ana Trujillo Emparedados y helados
3	Antonio Moreno Taquería

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

LEFT JOIN



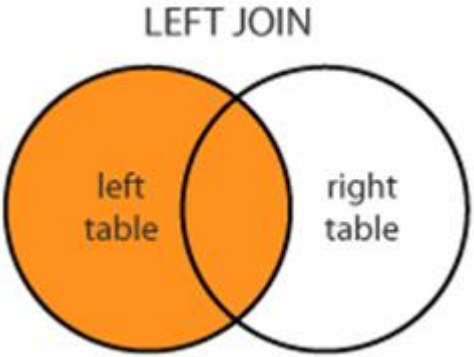
- Retorna los registros que son iguales en ambas tablas y los de la izquierda

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

CustomerID	CustomerName
1	Alfreds Futterkiste
2	Ana Trujillo Emparedados y helados
3	Antonio Moreno Taquería

```
SELECT *  
FROM Orders  
LEFT JOIN Customers ON Orders.CustomerID =  
Customers.CustomerID;
```


LEFT JOIN



- Retorna los registros que son iguales en ambas tablas y los de la izquierda

(196,5)

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2
10251	84	3	1996-07-08	1

Number of Records: 196

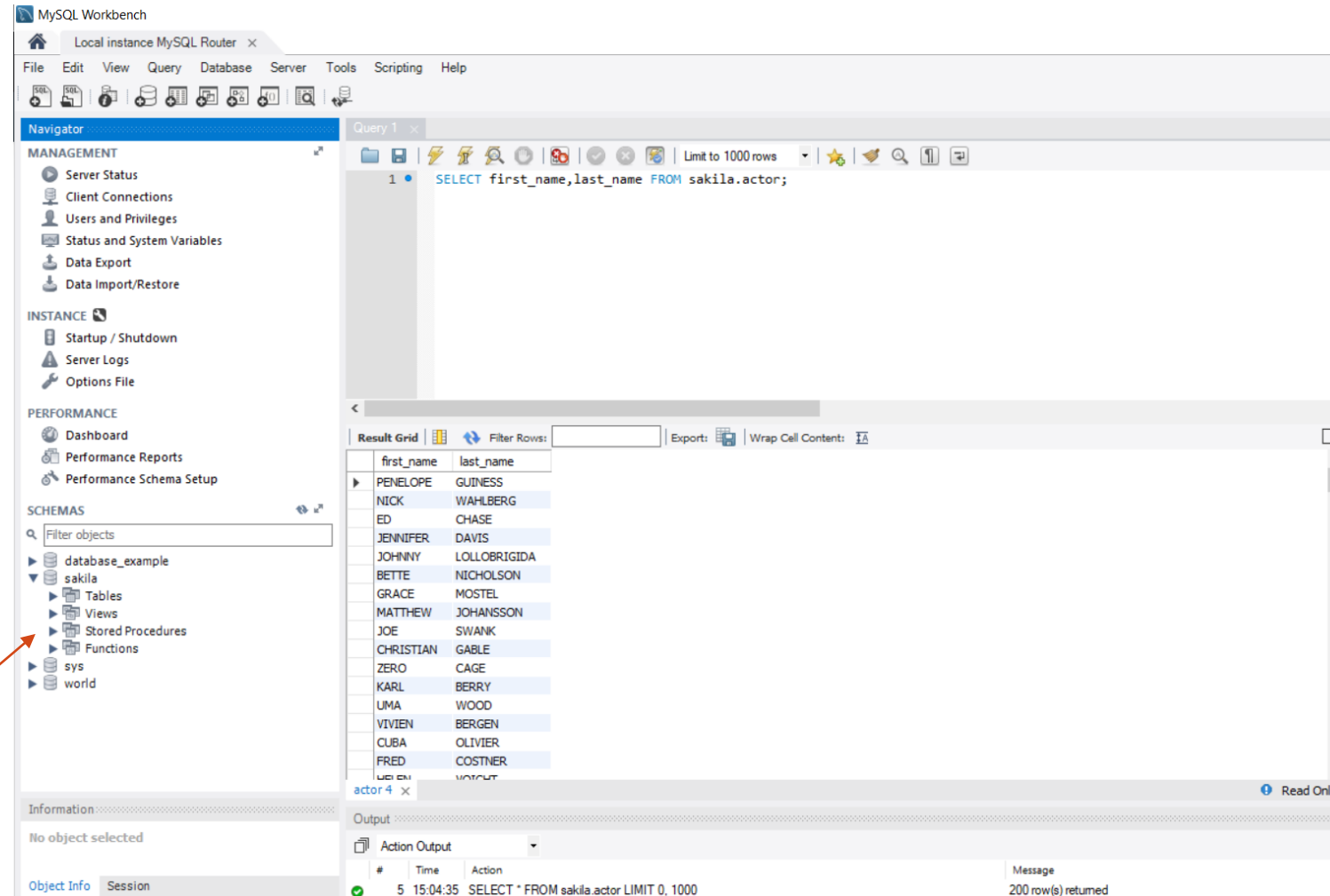
OrderID	CustomerID	EmployeeID	OrderDate	ShipperID	CustomerName	ContactName	Address	City	PostalCode	Country
10248	90	5	1996-07-04	3	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
10249	81	6	1996-07-05	1	Tradição Hipermercados	Anabela Domingues	Av. Inês de Castro, 414	São Paulo	05634-030	Brazil
10250	34	4	1996-07-08	2	Hanari Carnes	Mario Pontes	Rua do Paço, 67	Rio de Janeiro	05454-876	Brazil

(91,7)

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

SCHEMA

- Un esquema es una colección de objetos de base de datos asociados a un usuario.
- Una base de datos puede tener múltiples schema con el mismo nombre pero para distintos usuarios.



The screenshot shows the MySQL Workbench interface. On the left, the 'Navigator' pane displays the 'SCHEMAS' section with a list of databases: database_example, sakila, sys, and world. An orange arrow points to the 'sakila' database. The main query editor shows a query: `SELECT first_name, last_name FROM sakila.actor;`. The 'Result Grid' displays the following data:

first_name	last_name
PENELOPE	GUINNESS
NICK	WAHLBERG
ED	CHASE
JENNIFER	DAVIS
JOHNNY	LOLLOBRIGIDA
BETTE	NICHOLSON
GRACE	MOSTEL
MATTHEW	JOHANSSON
JOE	SWANK
CHRISTIAN	GABLE
ZERO	CAGE
KARL	BERRY
UMA	WOOD
VIVIEN	BERGEN
CUBA	OLIVIER
FRED	COSTNER

The bottom status bar indicates that 200 row(s) were returned.

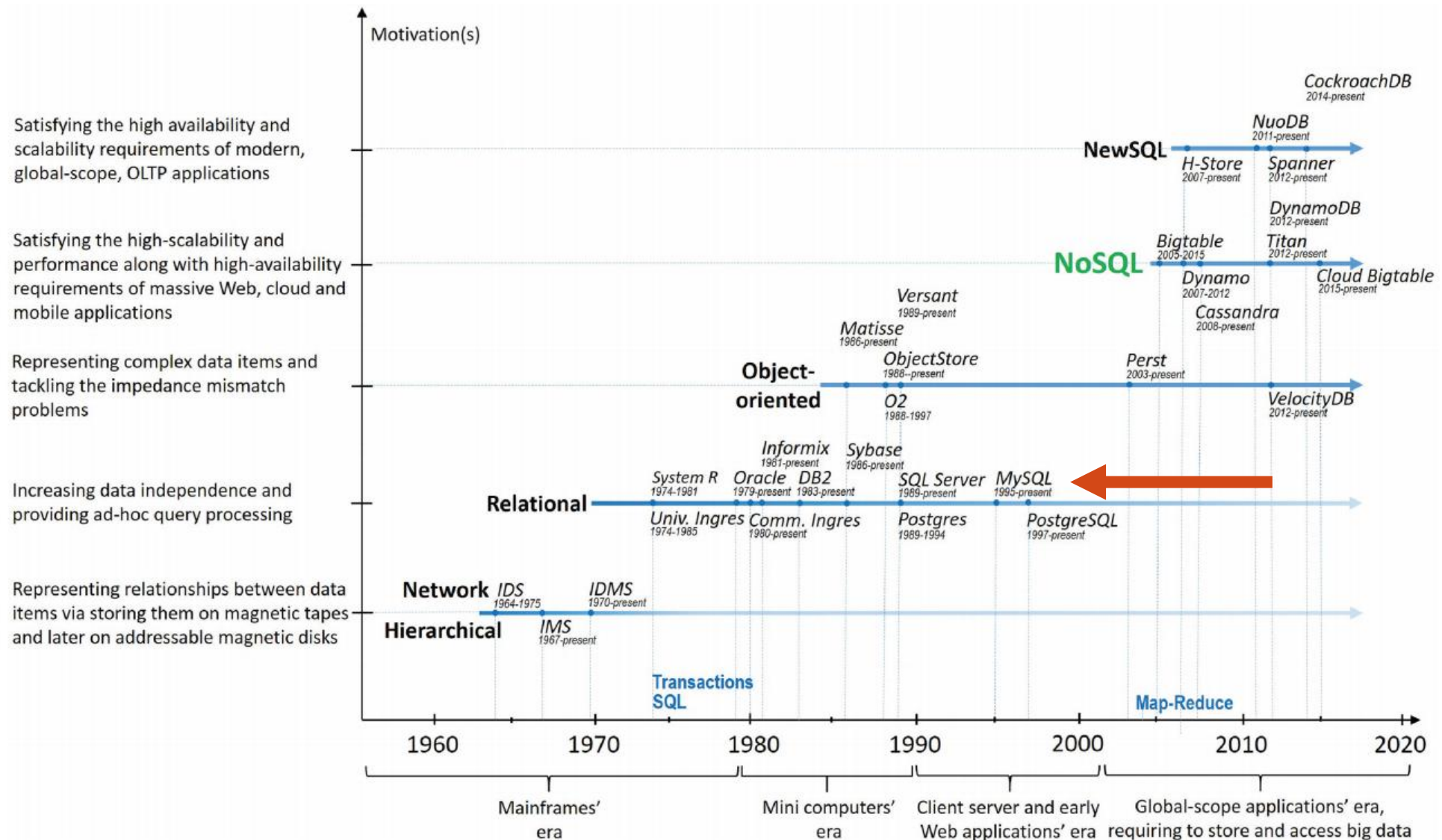


Fig. 1. The continuous development of major database technologies and some corresponding database systems.

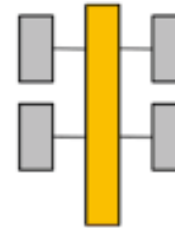
Davoudian, A., Chen, L., & Liu, M. (2018). A survey on NoSQL stores. *ACM Computing Surveys (CSUR)*, 51(2), 40.

SQL Database

Relational



Analytical (OLAP)



NoSQL Database

Column-Family



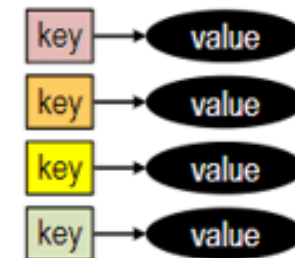
Graph



Document



Key-Value



BIBLIOGRAFÍA

- <https://www.guru99.com/nosql-tutorial.html>
- <https://www.lkeydata.com/es/sql/sintaxis-sql.php>
- <https://www.dofactory.com/Images/sql-joins.png>
- http://3.bp.blogspot.com/-af_pTzs8ryo/VOC-k2gVZMI/AAAAAAAAAI1Q/EcUe-cMU9yg/s1600/join.png