# PROGRAMACIÓN WEB FULL STACK

NIVEL 2

Módulo 4: Bases de Datos

Silicon Misiones

# Bases de Datos Relacional

Son una colección de elementos de datos organizados en un conjunto de tablas formalmente descritas, desde donde se puede acceder a los datos o volver a montarlos de muchas maneras diferentes sin tener que reorganizar las tablas de la base. La interfaz estándar de programa de usuario y aplicación a una base de datos relacional, es el Lenguaje de Consultas Estructuradas (SQL). Los comando SQL se utilizan tanto para consultas interactivas como para obtener información de una base de datos relacional y la recopilación de datos para informes.

Las bases de datos relacionales se basan en la organización de la información en partes pequeñas que se integran mediante identificadores; a diferencia de las bases de datos no relacionales que, como su nombre lo indica, no tienen un identificador que sirva para relacionar dos o más conjuntos de datos. Además son más robustas, es decir, tienen mayor capacidad de almacenamiento, y son menos vulnerables ante fallas, estas son sus principales características.

# Bases de datos no relacionales

Están diseñadas específicamente para modelos de datos específicos y tienen esquemas flexibles para crear aplicaciones modernas. Son ampliamente reconocidas porque son fáciles de desarrollar, tanto en funcionalidad como en rendimiento a escala. Usan una variedad de modelos de datos, que incluyen documentos, gráficos, clave-valor, en-memoria y búsqueda.

Las bases de datos no relacionales (NoSQL) son las que, a diferencia de las relacionales, no tienen un identificador que sirva de relación entre un conjunto de datos y otros. Como veremos, la información se organiza normalmente mediante documentos y es muy útil cuando no tenemos un esquema exacto de lo que se va a almacenar.

Con relación a formatos, la información de una base de datos puede ser almacenada en tablas o documentos. Cuando los datos son organizados en un archivo de Excel, es en formato tabla, pero cuando simplemente son datos escritos como cartas, fórmulas o recetas, son datos en formato documento. Esto aplica para los dos tipos de bases de datos.

# DB relacional vs No relacional

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

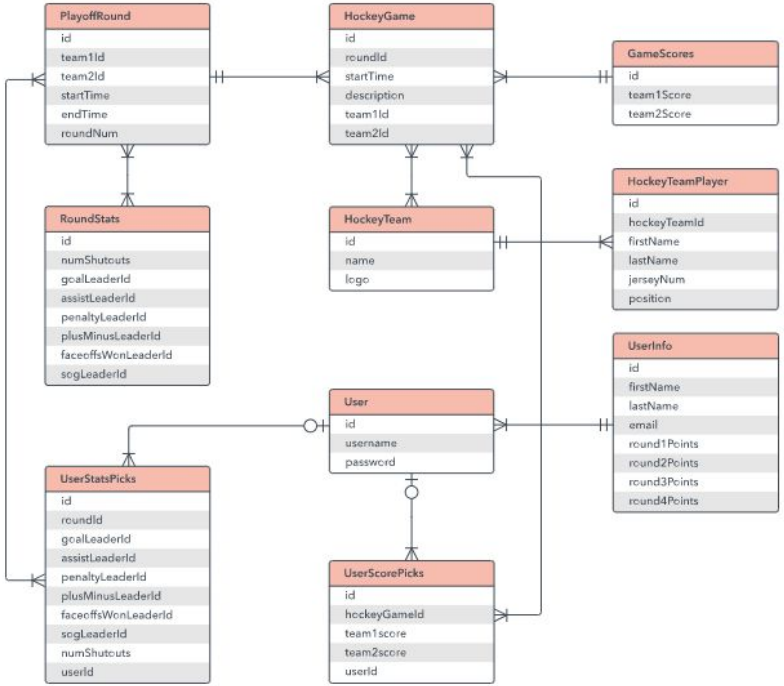



```
[
 {
  "student_id":1,
  "age":12,
  "score":77
 },
 {
  "student_id":2,
  "age":12,
  "score":68
 },
 {
  "student_id":3,
  "age":11,
  "score":75
 }
]
```

# Diagrama Entidad Relacion (DER)

Un diagrama entidad-relación, también conocido como modelo entidad relación o ERD, es un tipo de diagrama de flujo que ilustra cómo las "entidades", como personas, objetos o conceptos, se relacionan entre sí dentro de un sistema. Los diagramas ER se usan a menudo para diseñar o depurar bases de datos relacionales en los campos de ingeniería de software, sistemas de información empresarial, educación e investigación. También conocidos como los ERD o modelos ER, emplean un conjunto definido de símbolos, tales como rectángulos, diamantes, óvalos y líneas de conexión para representar la interconexión de entidades, relaciones y sus atributos. Son un reflejo de la estructura gramatical y emplean entidades como sustantivos y relaciones como verbos.

# Diagrama Entidad Relacion (DER)



# Bases de datos

## Componentes DER

Silicon Misiones

# DER Componentes

Los diagramas ER se componen de entidades, relaciones y atributos. También representan la cardinalidad, que define las relaciones en términos de números.

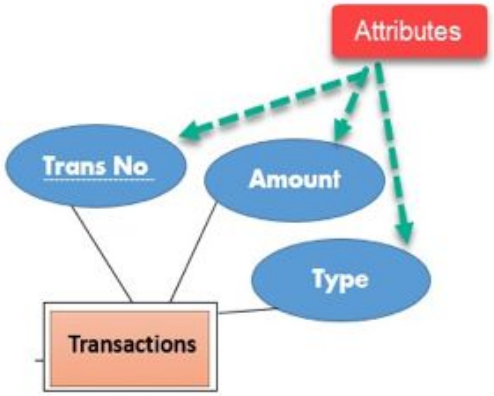
Bibliografía: Click aquí

# Entidad

Algo que se puede definir, como una persona, objeto, concepto u evento, que puede tener datos almacenados acerca de este. Piensa en las entidades como si fueran sustantivos. Por ejemplo: un cliente, estudiante, auto o producto. Por lo general se muestran como un rectángulo.

# Atributos

Los atributos se utilizan para explicar una entidad definiendo todo tipo de características relacionadas con ella. En la mayoría de los casos, es una sola propiedad la que explica una entidad, permitiéndonos conocer el tipo de datos almacenados en ella. En un Diagrama de Entidad-Relación, un atributo es representado por una elipse y está vinculado directamente a una entidad.

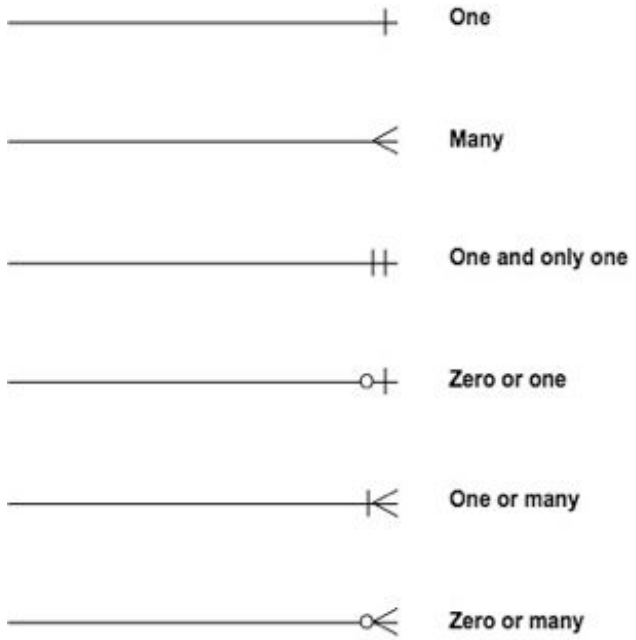


# Relación

Una relación define cómo dos o más entidades están conectadas entre sí. Se utilizan para identificar cuáles son las entidades que dependen de otras en una base de datos. Por ejemplo, consideramos que Tom y Química son dos entidades que están vinculadas con una relación que representa que Tom estudia Química.

# Cardinalidad

La cardinalidad es un componente vital en cualquier Diagrama de Entidad-Relación que explora el vínculo del atributo en la relación. Es decir, nos ayuda a explorar cómo dos o más entidades están vinculadas entre sí. Puede haber diferentes tipos de cardinalidad en un diagrama ER.



# Símbolos del diagrama entidad-relación

- Rectángulo: Se utiliza para una entidad o un tipo de entidad

- Doble rectángulo: Representa una entidad débil

- Diamante: Representa la relación entre diferentes entidades

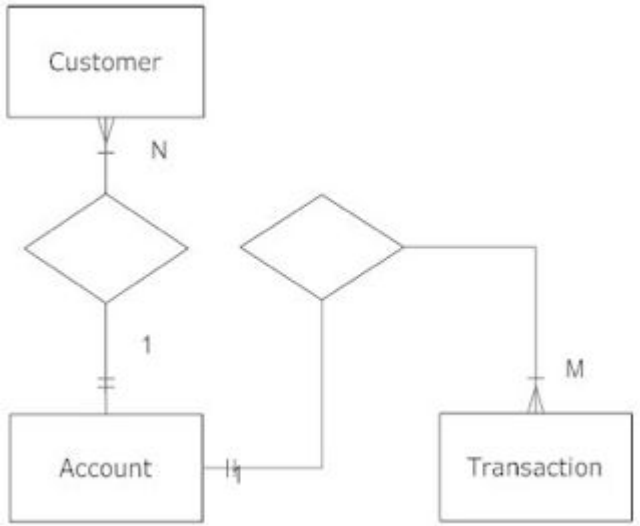
- Doble diamante: Representa una relación débil

- Elipse doble: Define un atributo multivalor

- Elipse con una línea: Se utiliza para un atributo de clave primaria

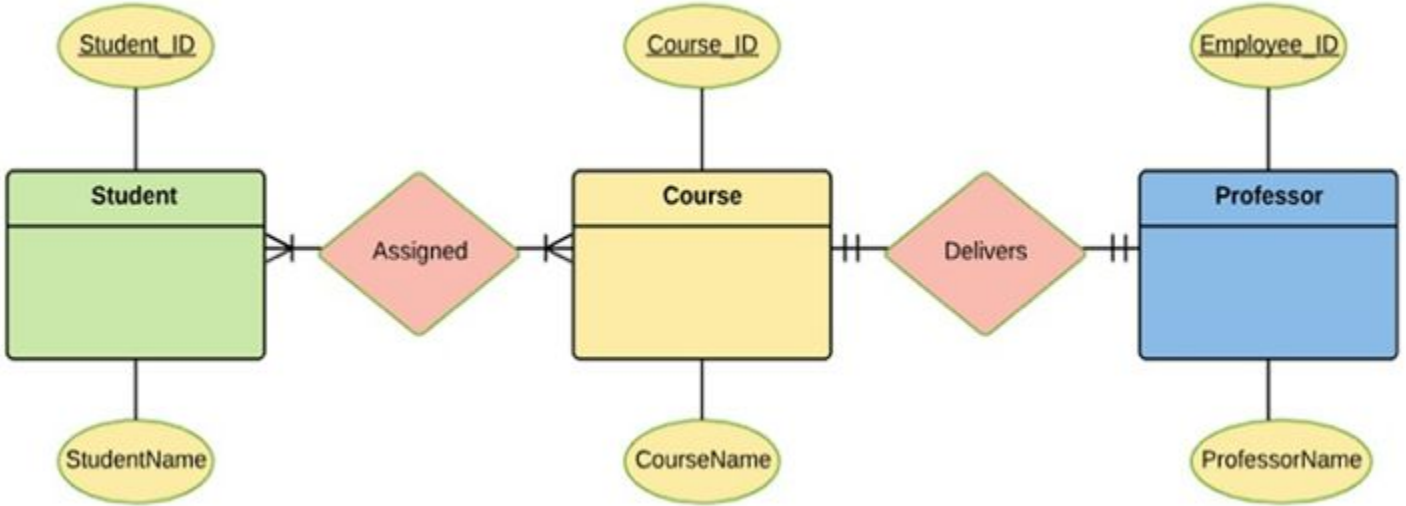
# Ej Base de datos de transacciones de cuentas

Empecemos con el caso más sencillo de un modelo ER. Este diagrama ER tiene tres entidades principales: cliente, cuenta y transacciones. En el caso de los titulares de cuentas conjuntas, más de un cliente puede tener una misma cuenta. Además, desde cada cuenta se pueden realizar múltiples transacciones.

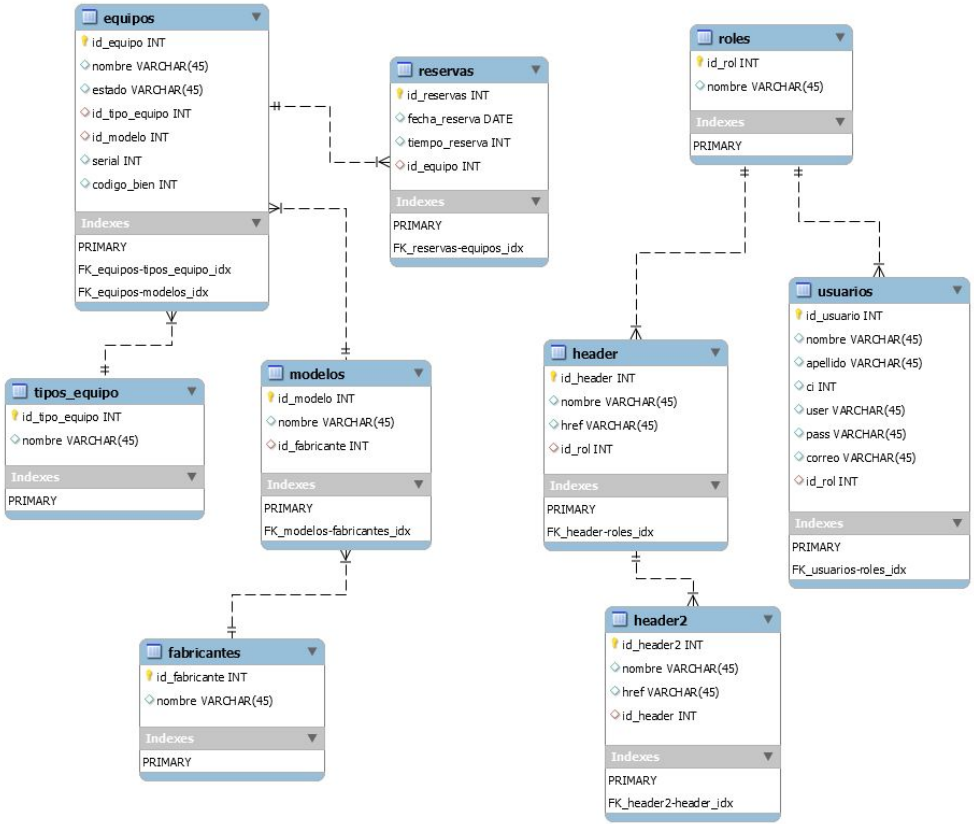


# Ej Base de datos de cursos para estudiantes

Aquí, Estudiante, Curso y Profesor son tres conjuntos de entidades diferentes. Cada conjunto tiene una clave única - id de estudiante, id de curso e id de profesor, además de sus atributos individuales. Como se puede ver, múltiples estudiantes pueden ser asignados a múltiples cursos. Sin embargo, los cursos individuales son impartidos por profesores exclusivos (cardinalidad uno a uno).



# Diagrama de Bases de Datos



# Bases de datos

# SQL

Bibliografía

Click aquí

Silicon Misiones

# Estructura mínima de almacenamiento

**Tabla:** Objeto de almacenamiento perteneciente a una BD. Es una estructura en forma de cuadrante donde se almacenan registros o filas de datos. Cada tabla tiene un nombre único en la BD.

**Registro:** Cada una de las filas de una tabla, esta compuesto por campos o atributos.

**Campo:** Cada uno de los “cajoncitos” de un registro donde se guardan los datos. Cada campo tiene un nombre único para la tabla de la cual forma parte, además es de un tipo (naturaleza) determinado, por tanto no podemos guardar limones en el cajón de las naranjas, en términos informáticos y a modo de ejemplo, no encontraremos un dato alfanumérico (letras y números) en un campo diseñado para guardar datos numéricos. Dedicaremos una lección a los tipos de datos más adelante.

# Ej tabla Empleados

| ID_EMPLEADO | NOMBRE | APELLIDOS | F_NACIMIENTO | SEXO | CARGO | SALARIO |
|---|---|---|---|---|---|---|
| 1 | Carlos | Jiménez Clarín | 1985-05-03 | H | Mozo | 1500 |
| 2 | Elena | Rubio Cuestas | 1978-09-25 | M | Secretaria | 1300 |
| 3 | José | Calvo Sisman | 1990-11-12 | H | Mozo | 1400 |
| 4 | Margarita | Rodríguez Garcés | 1992-05-16 | M | Secretaria | 1325.5 |

Cada registro o fila de datos contiene información de un empleado. En el ejemplo observamos que la tabla tiene un diseño de siete campos y que almacena cuatro registros. El nombre de cada campo viene dado por la fila de encabezado. El dato que contiene el campo ID_EMPLEADO identifica cada registro, pero por ahora no le demos importancia a esto.

Los registros de una tabla tienen en común sus atributos, no el dato en sí, que lo más probable es que difiera de un registro a otro, pero sí el hecho de que todos ellos poseen esos atributos. Por lo tanto las tablas de una BD guardan información de individuos o unidades de una misma naturaleza con una serie de atributos en común.

# Tipos de datos

Al igual que en el lenguaje JS visto en módulo 1, para SQL existen distintos tipos de datos.

Se destacan:

- Cadena (cadena de texto o alfanumérica)
- Número entero (sin decimales)
- Número decimal (parte entera + parte decimal)
- Fecha

# Crear Base de Datos

Para crear bases de datos se usa la declaración CREATE DATABASE

**Sintaxis**:

```
CREATE DATABASE databasename;
```

**Ejemplo:**

```
CREATE DATABASE silicon;
```

Para listar bases de datos se usa la declaración SHOW DATABASES

**Sintaxis**:

```
SHOW DATABASES;
```

Para eliminar bases de datos se usa la declaración DROP DATABASE

**Sintaxis**:

```
DROP DATABASE databasename;
```

**Ejemplo:**

```
DROP DATABASE silicon;
```

# Crear Tablas

Para crear tablas se usa la declaración CREATE TABLE

**Sintaxis**:

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

**Ejemplo:**

```
CREATE TABLE Persons (
      PersonID int ,
      LastName varchar(255),
      FirstName varchar(255),
      Address varchar(255),
      City varchar(255)
);
```

# Eliminar Tablas

Para eliminar tablas se usa la declaración:

**Sintaxis**:

```
DROP TABLE table_name;
```

**Ejemplo:**

```
DROP TABLE alumnos;
```

# Truncar Tablas

Para los datos que contiene una tabla se usa la declaración:

**Sintaxis**:

```
TRUNCATE TABLE table_name;
```

**Ejemplo:**

```
TRUNCATE TABLE alumnos;
```

# Restricciones SQL (Constraints)

Las constraints se usan para especificar reglas de restricciones a las tablas y columnas de una DB.
Estas se pueden especificar al crear una tabla o al modificarla.

**Sintaxis:**

```
CREATE TABLE table_name (
      column1 datatype constraint,
      column2 datatype constraint,
      column3 datatype constraint,
      ....
);
```

# Restricciones SQL

Las restricciones más comunes son:

- **NOT NULL**- Asegura que una columna no puede tener un valor NULL
- **UNIQUE**- Asegura que todos los valores en una columna sean diferentes
- **PRIMARY KEY**- Una combinación de a NOT NULL y UNIQUE. Identifica de forma única cada fila en una tabla
- **FOREIGN KEY** - Previene acciones que destruirían enlaces entre tablas
- **CHECK**- Asegura que los valores en una columna satisfagan una condición específica
- **DEFAULT**- Establece un valor predeterminado para una columna si no se especifica ningún valor
- **CREATE INDEX**- Se utiliza para crear y recuperar datos de la base de datos muy rápidamente

# Restricciones NOT NULL

**NOT NULL**- Asegura que una columna no puede tener un valor NULL

```
CREATE TABLE Persona (
      ID int NOT NULL,
      Nombre varchar(255) NOT NULL,
      Apellido varchar(255) NOT NULL,
      Edad int
);
```

# Restricciones UNIQUE

**UNIQUE**- Asegura que todos los valores en una columna sean diferentes

```
CREATE TABLE Persona (
      ID int NOT NULL UNIQUE,
      Nombre varchar(255) NOT NULL,
      Apellido varchar(255) NOT NULL,
      Edad int
);
```

# Restricciones PRIMARY KEY

**PRIMARY KEY**- Una combinación de a NOT NULL y UNIQUE. Identifica de forma única cada fila en una tabla

```
CREATE TABLE Persona (
      ID int NOT NULL,
      Nombre varchar(255) NOT NULL,
      Apellido varchar(255) NOT NULL,
      Edad int,
      PRIMARY KEY (ID)
);
```

# Restricciones FOREIGN KEY

Previene acciones que destruirían enlaces entre tablas.
Las tablas con Foreign Key (FK) son llamadas tablas hijo y las tablas con una Primary Key (PK) son llamadas tablas padre.

PK

| PersonaID | Nombre | Apellido | edad |
|---|---|---|---|
| **1** | Miguel | Zapata | 30 |
| **2** | Lorena | Perez | 23 |
| **3** | Sofia | Lopez | 20 |

PK FK

| OrdenID | OrdenNumber | PersonaID |
|---|---|---|
| 1 | 77895 | **3** |
| 2 | 44678 | **3** |
| 3 | 22456 | **2** |
| 4 | 24562 | **1** |

# Restricciones FOREIGN KEY

```
CREATE TABLE personas (
      personaId int AUTO_INCREMENT,
      nombre varchar(255) NOT NULL,
      apellido varchar(255) NOT NULL,
      edad int,
      PRIMARY KEY (personaId)
);
```

```
CREATE TABLE ordenes (
      ordenID int AUTO_INCREMENT,
      ordenDescripcion varchar(255),
      personaId int,
    PRIMARY KEY (ordenID),
    FOREIGN KEY (personaId) REFERENCES personas(personaId)
);
```

| personaId | nombre | apellido | edad |
|---|---|---|---|
| **1** | Miguel | Zapata | 30 |
| **2** | Lorena | Perez | 23 |
| **3** | Sofia | Lopez | 20 |

| ordenID | ordenDescripcion | personaId |
|---|---|---|
| **1** | Cerveza coronoa 725 | **3** |
| **2** | Pizza Napolitana | **3** |
| **3** | 6 Emp jamon y queso | **2** |

# Auto Increment

Es un número único que se genera automáticamente cuando un registro es insertado en una tabla.

```
CREATE TABLE personas (
      ID int NOT NULL AUTO_INCREMENT,
      nombre varchar(255) NOT NULL,
      apellido varchar(255) NOT NULL,
      edad int,
      PRIMARY KEY (ID)
);
```

# Insertar datos

Para cargar datos en una tabla se usa la declaración INSERT INTO

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

```
INSERT INTO personas (nombre, apellido) VALUES ('Miguel','Cichanowski');
```

## Vimos como:

- Crear/Eliminar una DB
- Crear/Eliminar una tabla
- Insertar datos en una tabla

## Ahora nos pasamos a cómo Consultar datos, para luego ver:

- Actualización de datos
- Eliminación de datos

Silicon Misiones

## Opcionalmente:

- **Pueden descargarse DBs de pruebas con datos cargados:**

https://dev.mysql.com/doc/index-other.html

Silicon Misiones

# Consultas SQL

Abordemos las consultas SQL con un caso práctico. Sobre la tabla EMPLEADOS se plantea la siguiente cuestión:
¿Qué empleados tienen un salario mayor a 1350?

| ID_EMPLEADO | NOMBRE | APELLIDOS | F_NACIMIENTO | SEXO | CARGO | SALARIO |
|---|---|---|---|---|---|---|
| 1 | Carlos | Jiménez Clarín | 1985-05-03 | H | Mozo | 1500 |
| 2 | Elena | Rubio Cuestas | 1978-09-25 | M | Secretaria | 1300 |
| 3 | José | Calvo Sisman | 1990-11-12 | H | Mozo | 1400 |
| 4 | Margarita | Rodríguez Garcés | 1992-05-16 | M | Secretaria | 1325.5 |

Vamos pues a construir la consulta que nos permita responder a esta cuestión.

# Consultas SQL

**Preguntas de Construcción**
Para construir una consulta SQL debemos hacernos como mínimo tres preguntas:

Primero hemos de preguntarnos**: ¿qué datos nos están pidiendo?**
En este caso, el nombre y los apellidos de los empleados.

Lo siguiente que nos preguntamos es: **¿dónde están esos datos?**
Obviamente están en la tabla empleados.

Y por último: **¿qué requisitos deben cumplir los registros?**
En este caso, que el sueldo del empleado sea superior a 1350.

# Consultas SQL

Selecciónname el NOMBRE y los APELLIDOS
del archivo EMPLEADOS
cuyo SALARIO sea mayor a 1350

El ayudante sirve la petición y se la entrega para que finalmente usted se la facilite a su jefe. ¿Que papel ocuparían hoy en una empresa moderna? El jefe sigue siendo el jefe, eso está claro, pero usted ha pasado a ser el informático, y su ayudante el SGBD.

Sintaxis SQL
En SQL la forma de operar es parecida, esta información se obtiene mediante la siguiente consulta:

```
select NOMBRE , APELLIDOS
  from EMPLEADOS
 where SALARIO > 1350
```



| NOMBRE | APELLIDOS |
|---|---|
| Carlos | Jiménez Clarín |
| José | Calvo Sisman |

| ID_EMPLEADO | NOMBRE | APELLIDOS | F_NACIMIENTO | SEXO | CARGO | SALARIO |
|---|---|---|---|---|---|---|
| 1 | Carlos | Jiménez Clarín | 1985-05-03 | H | Mozo | 1500 |
| 2 | Elena | Rubio Cuestas | 1978-09-25 | M | Secretaria | 1300 |
| 3 | José | Calvo Sisman | 1990-11-12 | H | Mozo | 1400 |
| 4 | Margarita | Rodríguez Garcés | 1992-05-16 | M | Secretaria | 1325.5 |

```
select NOMBRE , APELLIDOS
  from EMPLEADOS
 where SALARIO > 1350
```



| NOMBRE | APELLIDOS |
|---|---|
| Carlos | Jiménez Clarín |
| José | Calvo Sisman |

Cláusula **SELECT**: Donde indicamos los campos de la tabla que queremos obtener, separados por comas. Responde a la pregunta: ¿Qué datos nos piden?
Cláusula **FROM**: Donde indicamos en que tabla se encuentran estos campos. Responde a la pregunta: ¿Dónde están los datos?
Cláusula **WHERE**: Donde establecemos la condición que han de cumplir los registros de la tabla que serán seleccionados. Responde a la pregunta: ¿Qué requisitos deben cumplir lo registros? Es de hecho donde se establece el filtro de registros, es decir, que registros serán considerados para mostrar sus datos y cuales no.

# Select Where Operadores

La cláusula Where como vimos es usada para filtrar resultados.
Los operadores de comparación son:

| Operator | Description |
|---|---|
| = | Igual |
| > | Mayor |
| < | Menor |
| >= | Mayor o igual |
| <= | Menor o igual |
| <> | Distinto |
| BETWEEN | Entre un rango |
| LIKE | Busca por un patron |
| IN | En un conjunto de datos |

# Select DISTINCT

Para recuperar únicamente datos distintos

Sintaxis:

```
SELECT DISTINCT column1, column2, ...  FROM table_name;
```

Ejemplo:

```
SELECT DISTINCT nombre FROM personas;
```

# Select Where operadores

Para combinar operadores en una misma consulta se usan:

- **AND**: para solicitar que cumpla con ambas condiciones
- **OR**: para que cumpla con alguna de las condiciones
- **NOT**: Para negar una condicion

Sintaxis:

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

Ejemplo:

```
SELECT * FROM personas WHERE nombre = 'Miguel' AND apellido = 'Cicha';
```

# Select Order by

Usado para ordenar los resultados

Sintaxis:

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

Ejemplo:

```
SELECT * FROM personas ORDER BY nombre;
```

## Ahora si veamos:

- Actualización de datos
- Eliminación de datos

Silicon Misiones

# UPDATE

Usado para actualizar datos

Sintaxis:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Ejemplo:

```
UPDATE personas SET nombre = 'Pedro' WHERE ID=1;
```

# DELETE

Usado para eliminar datos

Sintaxis:

```
DELETE FROM table_name WHERE condition;
```

Ejemplo:

```
DELETE FROM personas WHERE ID=1;
```

# Volvemos a las consultas SELECT

**Hay mucho mas por ver…**

Silicon Misiones

# SELECT LIMIT

Usado para recuperar solo los primeros n resultados de una consulta.
Sintaxis:

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

Ejemplo:

```
SELECT * FROM personas LIMIT 2;
```

# SELECT MIN y MAX

Usado para recuperar minimos y maximos de una consulta.
Sintaxis:

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

Ejemplo:

```
SELECT MIN(edad) FROM personas;
```

# SELECT COUNT, AVG, SUM

**COUNT**: usado para contar cantidad de filas

```
SELECT COUNT(column_name) FROM table_name WHERE condition;
```

**AVG**: usado para sacar promedio

```
SELECT AVG(column_name) FROM table_name WHERE condition;
```

**SUM**: usado para sumar promedio

```
SELECT SUM(column_name) FROM table_name WHERE condition;
```

# LIKE

Usado en el WHERE para buscar mediante un patrón:

- se usa simbolo %, para determinar cualquier caracter
- se usa simbolo _ para representar 1 caracter

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

```
SELECT * FROM personas WHERE nombre LIKE 'M%';
```

# IN

Usado en el WHERE para buscar entre un conjunto de variables

```
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);
```

Ejemplo:

```
SELECT * FROM personas WHERE nombre IN ('Miguel','Lorena');
```

# BETWEEN

Usado en el WHERE para buscar entre un rango de valores. Numeros, texto, fecha.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Ejemplo:

```
SELECT * FROM personas WHERE edad BETWEEN 1 AND 18
```

# Aliases

Usado para asignarle un alias a una columna/s resultantes de una query.

```
SELECT column_name AS alias_name
FROM table_name;
```

Ejemplo:

```
SELECT personaId, CONCAT(nombre,' ',apellido) AS nombreCompleto FROM
personas;
```

# JOIN

Usado para combinar columna/s provenientes de mas de una tabla.

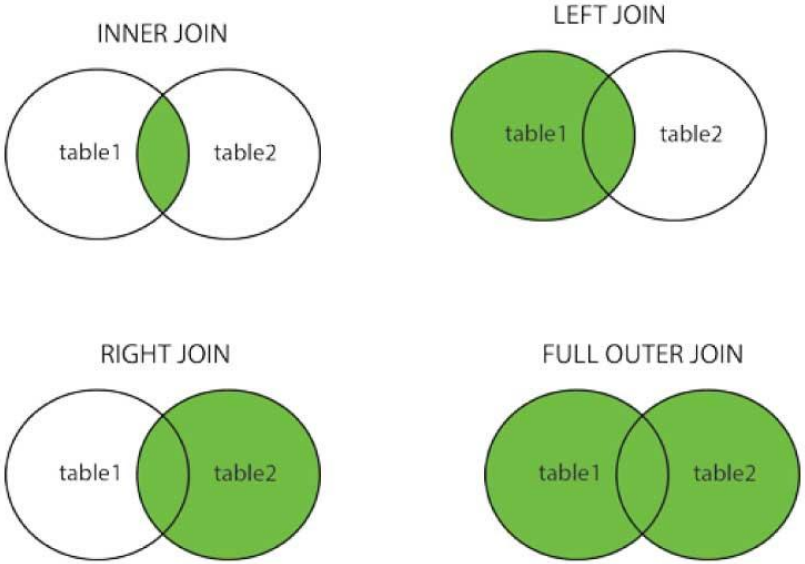
| CustomerID | CustomerName | ContactName | Country |
|---|---|---|---|
| 1 | Alfreds Futterkiste | Maria Anders | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mexico |

| 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;
```

| 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 |

# JOIN


INNER JOIN
table1
table2
LEFT JOIN
table1
table2
RIGHT JOIN
table1
table2
FULL OUTER JOIN
table1
table2