



DigitalHouse >
Coding School

DATA SCIENCE

UNIDAD 1
MÓDULO 1

Introducción a Bases de
Datos, SQL



Lucas Ceballos

IT Director @ Headway

Ing. Sistemas @ UTN



INTRODUCCIÓN

BASES DE DATOS Y SQL

- 1 **Características principales y utilidad de las bases de datos**
- 2 **Sintaxis básica de SQL: sentencias SELECT y JOIN**
- 3 **Esquema cliente/servidor. Conectarse a una base remota**
- 4 **Bases de datos en data science: performance, índices y plan de ejecución**

BASES DE DATOS



- Una **Base de Datos** es un conjunto de datos **relacionados y organizados** de tal manera que puedan responder a un **propósito específico**. Representan aspectos de la realidad (Dato != Información).
- Con el término “Base de Datos” también se hace referencia al software (**DBMS**) que permite administrarla, es decir, gestionar su tamaño, cargarle datos y consultarlos.
- Permiten **organizar** los datos con métodos eficientes para **obtener la información requerida** y crear reglas para asegurarse de que **los datos se almacenen de forma correcta y consistente**.
- El lenguaje que permite obtener datos de una Base de Datos se llama **SQL** (Structured Query Language).

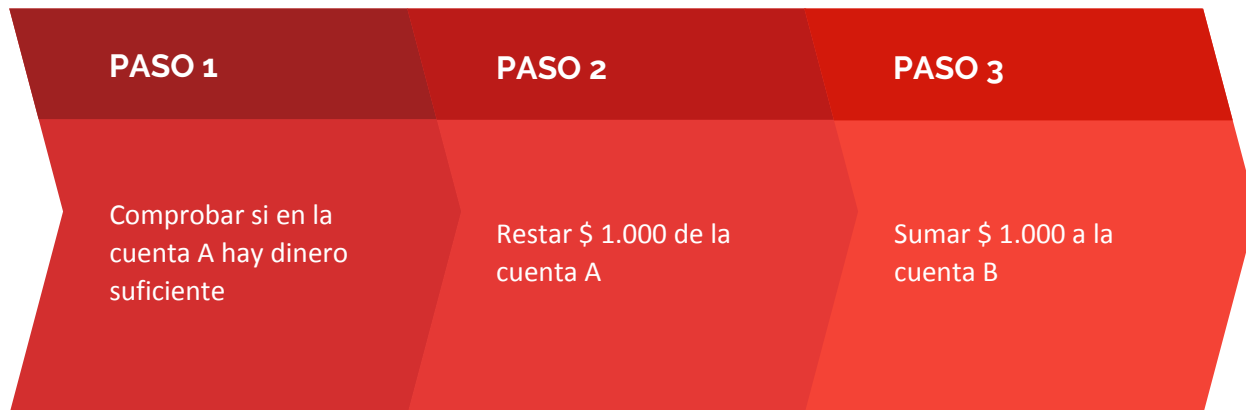
¿POR QUÉ UTILIZAMOS BASES DE DATOS EN LUGAR DE ARCHIVOS DE TEXTO AISLADOS?

PARADIGMA TRANSACCIONAL

- Por definición, las Transacciones deben ser **atómicas, consistentes, aisladas y durables**.
- Estas propiedades se las conocen como ACID, por sus siglas en inglés:
- **Atomicity:** Todas las modificaciones de una transacción deben ser ejecutadas o bien, ninguna de ella. Se ejecuta “todo o nada”.
- **Consistency:** La transacción comienza a ejecutarse en un estado válido (consistente) y finaliza en otro estado válido. Para eso, deben cumplirse todas las reglas de integridad que hayan sido definidas.
- **Isolation:** Dos o más transacciones *concurrentes* se ejecutan sin afectarse entre sí. Si ambas transacciones deben trabajar sobre el mismo dato una de ellas deberá esperar a que la otra termine.
- **Durability:** Asegura que una vez realizada la operación va a persistir (quedarán almacenados los cambios) por más que falle el sistema. Uso del Transaction Log del DBMS.

- Una **transacción** es un conjunto de operaciones, pertenecientes a una misma tarea, que se realizan sobre una base de datos y representan un cambio en los datos.

Ejemplo de una Transacción bancaria



- ¿Qué pasa si el proceso falla entre el paso 2 y el 3 (o sólo se ejecutan los pasos 1 y 2)?
- ¿Qué obtiene alguien que quiere consultar su Saldo entre los pasos mencionados?

BASE DE DATOS RELACIONAL



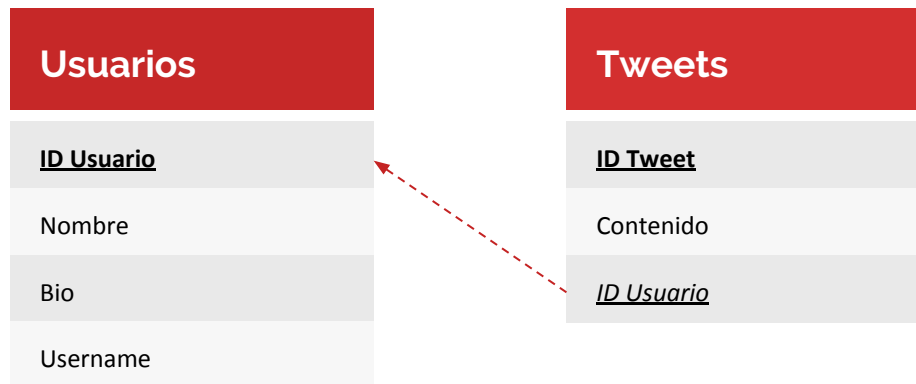
Son bases de datos basadas en el “modelo relacional” (Codd, 1970).

- La información se organiza en **tablas** compuestas de filas y columnas que están relacionadas de acuerdo a las entidades o conceptos que representan.
- A las tablas se le asocian reglas que son aplicadas cuando se intenta cargarles algún dato:
 - Cuántos campos tiene y cómo se llaman
 - De qué tipo son (textos, números enteros, fechas, etc.)
- Al menos una columna debe funcionar como **Clave Primaria**.
- Puede contener una **Clave Foránea** que haga referencia a la Clave Primaria de otra tabla.

EJEMPLO DE APLICACIÓN

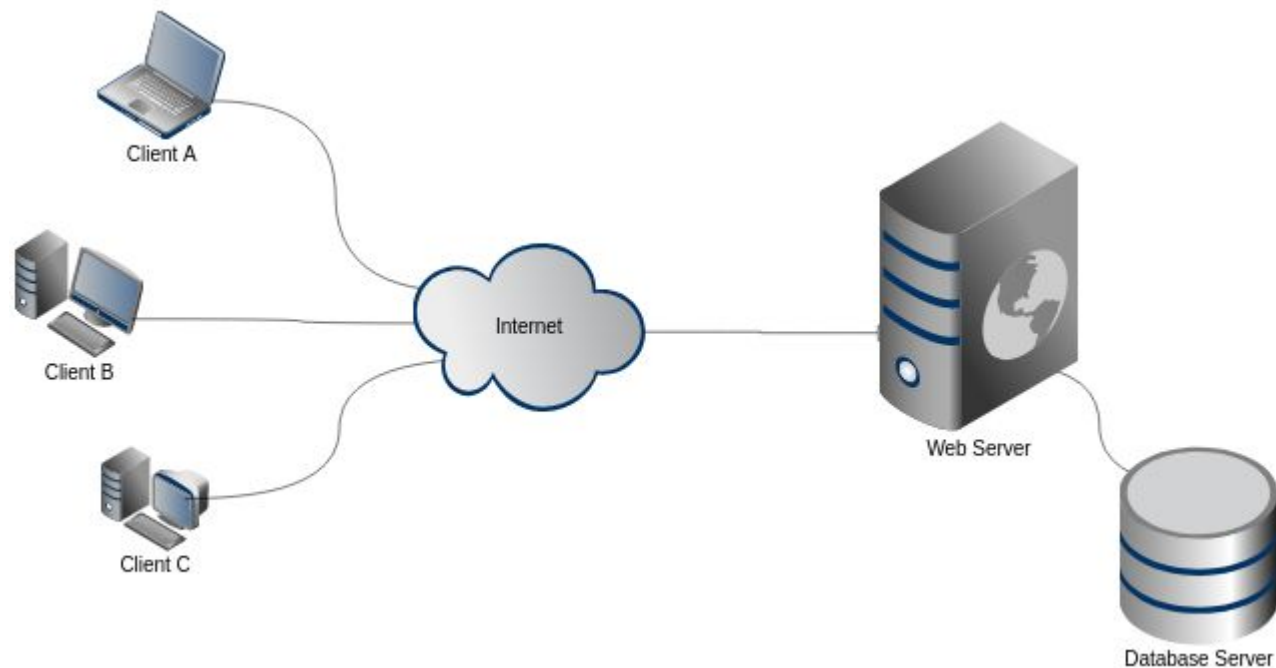
Twitter

- Las principales **entidades** o conceptos: Usuarios y Tweets. Cada una representada por una **tabla**.
- La tabla Usuarios tendrá el ID del usuario (Clave Primaria), su nombre y otros datos personales.
- La tabla Tweets tendrá el ID del Tweet (Clave Primaria), el texto o contenido del Tweet y el ID del usuario que lo escribió (Clave Foránea de la tabla de Usuarios).



BASE DE DATOS REMOTA

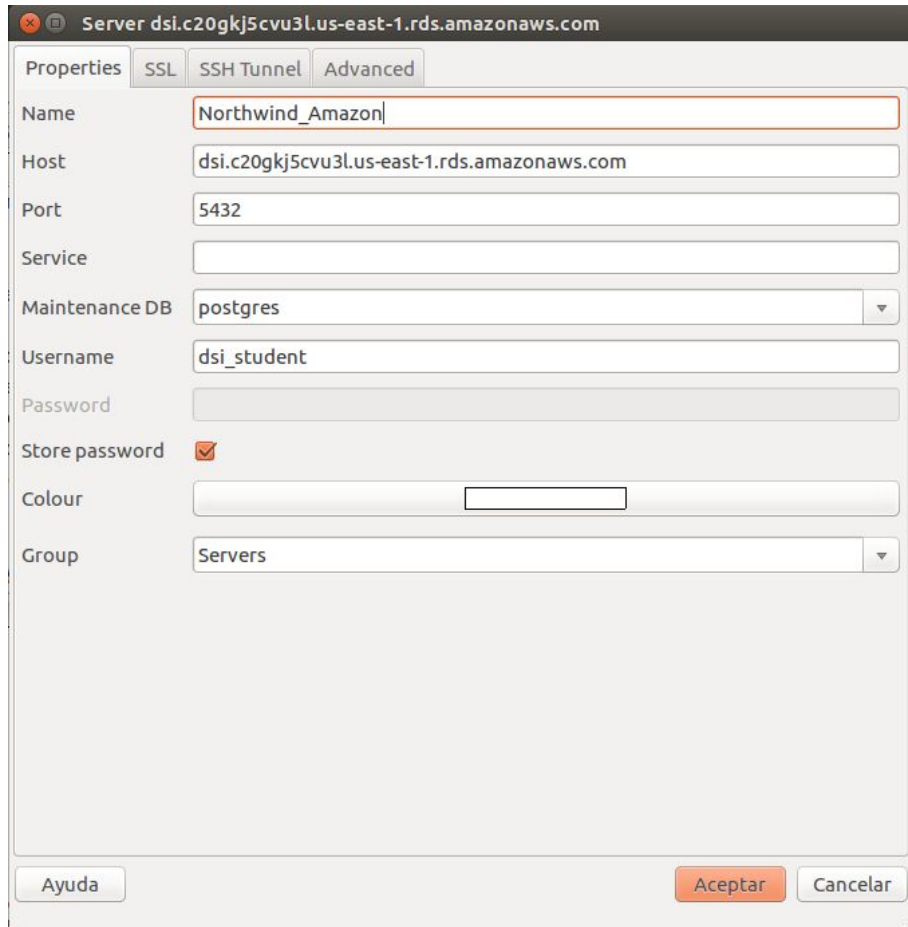




Ejemplo de una Base de Datos remota

- PostgreSQL (<https://www.postgresql.org/>) es un software que permite gestionar una Base de Datos Relacional, en una arquitectura cliente-servidor.
- Puede correr en diferentes plataformas, incluyendo Linux, UNIX y Windows.
- Soporta la mayoría de los tipos de datos SQL (http://www.w3schools.com/sql/sql_datatypes.asp), incluyendo BLOB, Imágenes, Sonidos, etc.
- Posee interfaces de programación nativa para C/C++, Java, .Net, Perl, Python, Ruby, ODBC y otros.
- Herramienta con GUI para administrar Bases de Datos PostgreSQL:
pgAdmin (<https://www.pgadmin.org/>)
Muchas otras herramientas (algunas open-source y otras comerciales) en:
<https://www.postgresql.org/download/products/1-administrationdevelopment-tools/>





The screenshot shows the 'Server Properties' dialog box in pgAdmin, titled 'Server dsi.c20gkj5cvu3l.us-east-1.rds.amazonaws.com'. The 'Properties' tab is selected. The fields are as follows:

- Name: Northwind_Amazon
- Host: dsi.c20gkj5cvu3l.us-east-1.rds.amazonaws.com
- Port: 5432
- Service: (empty)
- Maintenance DB: postgres
- Username: dsi_student
- Password: (empty)
- Store password: ☒
- Colour: (empty)
- Group: Servers

At the bottom, there are three buttons: 'Ayuda', 'Aceptar', and 'Cancelar'.

Conexión a una base de datos remota

- Utilizando pgAdmin nos conectamos a una base de datos remota alojada en Amazon.

- Trabajaremos sobre la Base de Datos **Northwind** (<https://northwinddatabase.codeplex.com/>)
- Se encuentra creada en la Base de Datos remota PostgreSQL presentada al inicio del módulo.
- Para resolver los ejercicios utilizaremos sólo las siguientes tablas de su Modelo de Datos:

Column Name	Condensed ...	Nul...
CustomerID	nchar(5)	No
CompanyName	nvarchar(40)	No
ContactName	nvarchar(30)	Yes
ContactTitle	nvarchar(30)	Yes
Address	nvarchar(60)	Yes
City	nvarchar(15)	Yes
Region	nvarchar(15)	Yes
PostalCode	nvarchar(10)	Yes
Country	nvarchar(15)	Yes
Phone	nvarchar(24)	Yes
Fax	nvarchar(24)	Yes

Column Name	Condensed Type	Nullable
OrderID	int	No
CustomerID	nchar(5)	Yes
EmployeeID	int	Yes
OrderDate	datetime	Yes
RequiredDate	datetime	Yes
ShippedDate	datetime	Yes
ShipVia	int	Yes
Freight	money	Yes
ShipName	nvarchar(40)	Yes
ShipAddress	nvarchar(60)	Yes
ShipCity	nvarchar(15)	Yes
ShipRegion	nvarchar(15)	Yes
ShipPostalCode	nvarchar(10)	Yes
ShipCountry	nvarchar(15)	Yes

Column Name	Condensed Type	Nullable
OrderID	int	No
ProductID	int	No
UnitPrice	money	No
Quantity	smallint	No
Discount	real	No

SINTAXIS SQL



SQL (Structured Query Language) es un lenguaje que permite administrar Bases de Datos relacionales.

- **DDL (Data Definition Language):** Permiten definir la estructura (e.g.: tablas) que almacene los datos:

CREATE TABLE MyTable (Campo1 Tipo1, Campo2 Tipo2);

ALTER TABLE MyTable ADD COLUMN Campo3 Tipo3;

DROP TABLE MyTable;

- **DML (Data Manipulation Language):** Permiten manipular los datos:

INSERT INTO MyTable (Campo1, Campo2) VALUES (Valor1, Valor2);

UPDATE MyTable SET Campo1 = Valor1, Campo2 = Valor2;

DELETE FROM MyTable;

SELECT * FROM MyTable;

Estructura (DML)

Datos (DDL)



DNI	NOMBRE	APELLIDO	EDAD	SEXO
12121212	Martín	Martinez	56	M
23232323	Carmen	Carter	40	F
34343434	Pablo	Ponce	23	M

DML



Lenguaje de manipulación de datos (DML)

Grupo de sentencias SQL utilizadas para realizar consultas y modificaciones sobre los **registros** almacenados dentro de cada una de las tablas.

Existen cuatro **tipos de operaciones**:

SELECT
INSERT
UPDATE
DELETE

SELECT



Selección de registros

Para hacer consultas sobre los registros de la tabla y seleccionar datos, se utiliza la sentencia **SELECT**. El resultado de la consulta es una nueva tabla que contiene la información solicitada.

Estructura:

```
SELECT column1, column2,..., columnN  
FROM table_name  
WHERE condition  
ORDER BY column1 ASC/DESC;
```

Las instrucciones detalladas en azul son optativas.

WHERE: permite especificar alguna condición para filtrar datos.

ORDER BY: permite ordenar los resultados obtenidos en forma ascendente (ASC) o descendente (DESC)

Se usa el * para informar a la consulta que quiero todas las columnas.

En los casos en los que se quiere ejecutar una sentencia `SELECT` que retorne solo los registros que cumplen alguna condición específica se utiliza el operador **WHERE**. El uso de la cláusula **WHERE** es opcional y puede no incluirse en caso de no ser necesario.

```
SELECT *  
FROM Productos  
WHERE idCategoria = 4;
```

```
SELECT Nombre  
FROM Productos  
WHERE Precio > 100;
```

En color **ROJO** se observan los operadores lógicos de comparación.

Operator	Description	Example
=	Equal to	Author = 'Alcott'
<>	Not equal to (many DBMSs accept != in addition to <>)	Dept <> 'Sales'
>	Greater than	Hire_Date > '2012-01-31'
<	Less than	Bonus < 50000.00
>=	Greater than or equal	Dependents >= 2
<=	Less than or equal	Rate <= 0.05
BETWEEN	Between an inclusive range	Cost BETWEEN 100.00 AND 500.00
LIKE	Match a character pattern	First_Name LIKE 'Will%'
IN	Equal to one of multiple possible values	DeptCode IN (101, 103, 209)
IS or IS NOT	Compare to null (missing data)	Address IS NOT NULL
IS NOT DISTINCT FROM	Is equal to value or both are nulls (missing data)	Debt IS NOT DISTINCT FROM - Receivables
AS	Used to change a field name when viewing results	SELECT employee AS 'department1'

Ordenar y Limitar Resultados

En caso de querer ordenar los resultados de una consulta se utilizará el operador **ORDER BY**.

Si se quiere limitar la cantidad de resultados se utilizará el operador **LIMIT**.

```
SELECT *  
FROM Productos  
WHERE idCategoria = 4  
ORDER BY SKU DESC;
```

```
SELECT Nombre  
FROM Productos  
WHERE Precio > 100  
LIMIT 10;
```

Ambos operadores son opcionales y pueden no ser incluidos en la consulta.

Agregacion



Contabilizar la cantidad de registros de una tabla

Se utiliza la operación **COUNT** aplicado a la columna que quiero contar.

```
SELECT COUNT(NombreDeColumna)  
FROM NombreDeLaTabla  
WHERE condición
```

```
SELECT COUNT(*)  
FROM NombreDeLaTabla  
WHERE condicion
```

Las instrucciones detalladas en azul son optativas.

Contabilizar la cantidad de registros de una tabla

```
SELECT COUNT(DNI)  
FROM Clientes  
WHERE Apellido="Lopez"
```

```
SELECT COUNT(*)  
FROM Clientes
```

Clientes
DNI (PK)
Nombre
Apellido

Operaciones numéricas

Suma de registros:

```
SELECT SUM(NombreDeColumna)
FROM NombreDeLaTabla
WHERE condición
```

Promedio de registros:

```
SELECT AVG(NombreDeColumna)
FROM NombreDeLaTabla
WHERE condición
```

Registro con valor máximo:

```
SELECT MAX(NombreDeColumna)
FROM NombreDeLaTabla
WHERE condición
```

Registro con valor mínimo:

```
SELECT MIN(NombreDeColumna)
FROM NombreDeLaTabla
WHERE condición
```


Operaciones numéricas

Ejemplos

```
SELECT SUM(Precio)  
FROM Productos
```

```
SELECT AVG(Precio)  
FROM Productos  
WHERE Precio > 10
```

```
SELECT MAX(Precio)  
FROM Productos  
WHERE Nombre LIKE "A%"
```

Productos
SKU (PK) (FK)
Nombre
Precio

Agrupación de registros

GROUP BY

El **GROUP BY** separa la tabla en diferentes subgrupos y a cada subgrupo le aplica la función de agregación que indicamos en el **SELECT**. Las columnas indicadas en el **GROUP BY** se denominan columnas de agrupación.

```
SELECT Funcion(ColumnaX), ColumnaY  
FROM NombreDeLaTabla  
GROUP BY ColumnaY
```

Agrupación de registros

Ejemplo:

```
SELECT DNI, SUM(Precio)
FROM Compras
GROUP BY DNI
```

Compras
SKU (PK) (FK)
DNI (FK)
Fecha
Precio

¿Cómo funciona el GROUP BY?

Como estamos agrupando por DNI, se crean grupos diferentes por cada DNI que exista en la tabla. En este ejemplo, existen dos grupos.

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
2	35.186.928	02/01/2017	60
3	33.241.677	03/01/2017	70
4	35.186.928	04/01/2017	40

Agrupación

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
2	35.186.928	02/01/2017	60
3	33.241.677	03/01/2017	70
4	35.186.928	04/01/2017	40

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
3	33.241.677	03/01/2017	70

SKU	DNI	Fecha	Precio
2	35.186.928	02/01/2017	60
4	35.186.928	04/01/2017	40

Aplicación de la función de agregación

Sobre cada **grupo**, se aplica la función de agregación que se indicó en el SELECT.
En este caso, se aplica la función suma sobre la columna precio

SUM			
SKU	DNI	Fecha	Precio
2	35.186.928	02/01/2017	60
4	35.186.928	04/01/2017	40

SUM			
SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
3	33.241.677	03/01/2017	70

Resultado

El resultado de la consulta, es una tabla que contiene el resultado de cada grupo.

DNI	SUM(PRECIO)
35.186.928	100
33.241.677	120

Agrupación de registros con condiciones

HAVING permite eliminar los grupos que no cumplan con la condición indicada. La condición será una función de agregación, ya que se aplica al grupo entero. Es similar al **WHERE**, pero se aplica al grupo entero y no a cada fila.

```
SELECT Funcion(ColumnaX), ColumnaY  
FROM NombreDeLaTabla  
GROUP BY ColumnaY  
HAVING condicion
```


Agrupación de registros

Ejemplo:

```
SELECT DNI, SUM(Precio)
FROM Compras
GROUP BY DNI
HAVING SUM(Precio) > 110
```

Compras
SKU (PK) (FK)
DNI (FK)
Fecha
Precio

¿Cómo funciona el **GROUP BY** con **HAVING**?

Como estamos agrupando por DNI, se crean grupos diferentes por cada DNI que exista en la tabla. En este ejemplo, existen dos grupos.

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
2	35.186.928	02/01/2017	60
3	33.241.677	03/01/2017	70
4	35.186.928	04/01/2017	40

Agrupación

SKU	DNI	Fecha	Precio
1	33.241.67 7	01/01/201 7	50
2	35.186.92 8	02/01/201 7	60
3	33.241.67 7	03/01/201 7	70
4	35.186.92 8	04/01/201 7	40

SKU	DNI	F	4	35.186.928	04/01/2017	40	Fecha	Precio
2	35.186.928	02/01/2017		60	1	33.241.677	01/01/2017	50
4	35.186.928	04/01/2017		40	3	33.241.677	03/01/2017	70

Aplicación de la función de Agregación

Sobre cada **grupo**, se aplica la función de agregación que se indicó en el SELECT. En este caso, se aplica la función suma sobre la columna precio.

SUM			
SKU	DNI	Fecha	Precio
2	35.186.928	02/01/2017	60
4	35.186.928	04/01/2017	40

SUM			
SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
3	33.241.677	03/01/2017	70

Aplicación de la condición del HAVING

Se eliminan los grupos que no cumplan la condición indicada. En este ejemplo, se eliminan todos los grupos cuya suma es menor a 110.

DNI	SUM(PRECIO)
35.186.928	100
33.241.677	120

Resultado

El resultado de la consulta, es una tabla que contiene el resultado de cada grupo que cumple la condición.

DNI	SUM(PRECIO)
33.241.677	120

Agrupación de registros

Ejemplo Avanzado:

```
SELECT DNI, SUM(Precio)
FROM Compras
WHERE Fecha > "02/01/2017"
GROUP BY DNI
HAVING SUM(Precio) >= 70
```

Compras
SKU (PK) (FK)
DNI (FK)
Fecha
Precio

¿Cómo funciona el **GROUP BY** con **HAVING** y con **WHERE**?

Lo primero que se ejecuta en esta consulta es la cláusula **WHERE**. Se eliminan todas las filas que no cumplan la condición.

SKU	DNI	Fecha	Precio
1	33.241.677	01/01/2017	50
2	35.186.928	02/01/2017	60
3	33.241.677	03/01/2017	70
4	35.186.928	04/01/2017	40

Agrupación

Luego, se continúa con la agrupación. Como estamos agrupando por DNI, se crean grupos diferentes por cada DNI que exista en la tabla. En este ejemplo, existen dos grupos.

		SKU	DNI	Fecha	Precio			
		3	33.241.677	03/01/2017	70			
		4	35.186.928	04/01/2017	40			
SKU	DNI	Fecha	Precio		SKU	DNI	Fecha	Precio
4	35.186.928	04/01/2017	40		3	33.241.677	03/01/2017	70

Aplicación de la función de Agregación

Sobre cada **grupo**, se aplica la función de agregación que se indicó en el SELECT.
En este caso, se aplica la función suma sobre la columna precio

SUM				SUM			
SKU	DNI	Fecha	Precio	SKU	DNI	Fecha	Precio
4	35.186.928	04/01/2017	40	3	33.241.677	03/01/2017	70

Aplicación de la condición del HAVING

Se eliminan los grupos que no cumplan la condición indicada. En este ejemplo, se eliminan todos los grupos cuya suma es menor a 70.

DNI	SUM(PRECIO)
35.186.928	40
33.241.677	70

Resultado

El resultado de la consulta, es una tabla que contiene el resultado de cada grupo que cumple la condición.

DNI	SUM(PRECIO)
33.241.677	70

PRÁCTICA GUIADA 1

Sintaxis básica de SQL

JOINS SQL

- Imaginemos que contamos con un dataset que contiene las siguientes tablas: Personas y Logros
- Veamos el contenido de las tablas:

```
SELECT * FROM Personas;  
SELECT * FROM Logros;
```

Personas

PersonalID	Apellido
1	Fleming
2	Eratóstenes
3	Newton
4	Fernandez

Logros

LogroID	PersonalID	Logro
1	1	Descubrimiento de la Penicilina
2	2	Cálculo del perímetro terrestre
3	3	Ley de gravitación universal
4	3	Desarrollo del Cálculo
5		Cura del cancer

INNER

JOIN

- El tipo más común de combinación es el: INNER JOIN (join simple). El INNER JOIN devuelve todas las filas de ambas tablas donde se cumple la condición de combinación.
- Observen en las tablas de Personas y Logros que la columna PersonalID en la tabla Logros hace referencia a PersonalID en la tabla Personas.

Personas

PersonalID	Apellido
1	Fleming

Logros

LogroID	PersonalID	Logro
1	1	Descubrimiento de la Penicilina

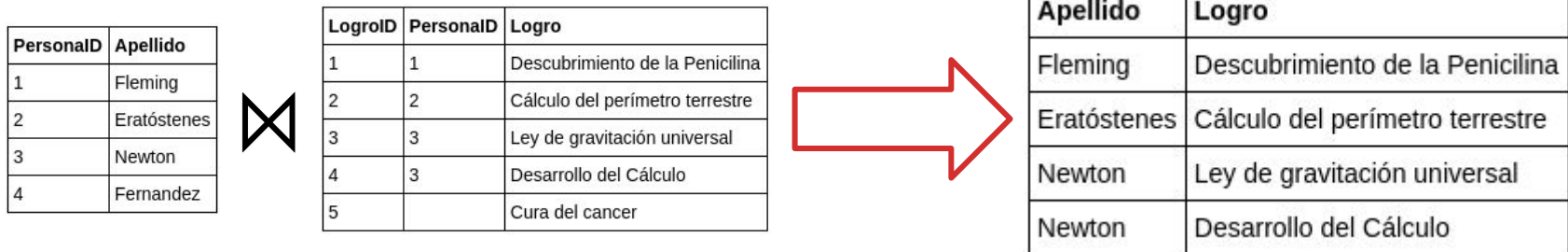
- La relación entre las dos tablas es la columna PersonalID. Por lo tanto, podemos combinar las dos tablas para obtener una tabla como la siguiente:
- Donde la información contenida en las dos tablas está combinada en una sola tabla, usando la clave común PersonalID.

Apellido	Logro
Fleming	Descubrimiento de la Penicilina
Eratóstenes	Cálculo del perímetro terrestre
Newton	Ley de gravitación universal
Newton	Desarrollo del Cálculo

- A partir de las tablas anteriores, para producir la tabla combinada, donde Apellido provenga de la tabla Personas y Logro provenga de la tabla Logros, se puede utilizar la siguiente sentencia SQL:

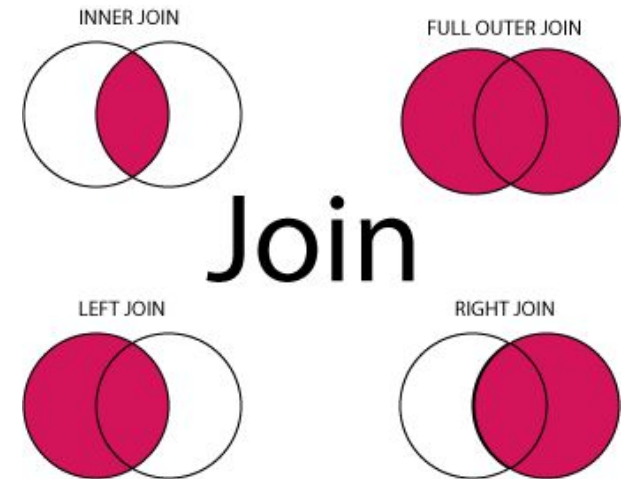
```
SELECT Personas.Apellido, Logros.Logro
FROM Personas
INNER JOIN Logros
ON Personas.PersonaID = Logros.PersonaID;
```

- El INNER JOIN hace la intersección de dos tablas, excluyendo los registros donde PersonaID es null en cualquiera de las dos tablas.



- Hay varios tipos de operaciones de JOIN.
 - **INNER JOIN:** Retorna todos los registros donde haya al menos una coincidencia en ambas tablas
 - **LEFT JOIN:** Retorna todos los registros de la tabla izquierda, y los registros que coincidan de la tabla derecha
 - **RIGHT JOIN:** Retorna todos los registros de la tabla derecha, y los registros que coincidan de la tabla izquierda
 - **FULL OUTER JOIN:** Retorna todos los registros de ambas tablas aunque no tengan correspondencia

- Es mucho más fácil entender los JOIN como operaciones de intersección de conjuntos.
- Existe una teoría matemáticamente sólida detrás de estas operaciones llamada Álgebra Relacional.



LEFT JOIN

El LEFT JOIN retorna todas las filas de la tabla izquierda (tabla1), con las filas coincidentes en la tabla derecha (tabla2).

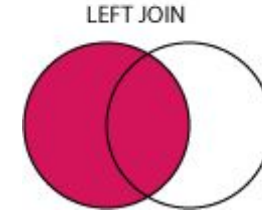
El resultado es NULL en el lado derecho cuando no hay coincidencia.

Sintaxis del Left Join:

```
SELECT nombres_columnas
FROM tabla1
LEFT JOIN tabla2
ON tabla1.columna_relacion=tabla2.columna_relacion;
```

Ejemplo para Personas y Logros:

```
SELECT Personas.Apellido, Logros.Logro
FROM Personas
LEFT JOIN Logros
ON Personas.PersonaID = Logros.PersonaID;
```



Apellido	Logro
Fleming	Descubrimiento de la Penicilina
Eratóstenes	Cálculo del perímetro terrestre
Newton	Ley de gravitación universal
Newton	Desarrollo del Cálculo
Fernandez	

RIGHT JOIN

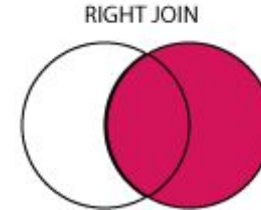
De forma similar, el RIGHT JOIN devuelve todas las filas de la tabla derecha (tabla2), con las filas coincidentes en la tabla de la izquierda (tabla1). El resultado es NULL del lado izquierdo cuando no hay coincidencia.

Sintaxis del Right Join:

```
SELECT nombres_columnas
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.columna_relacion=tabla2.columna_relacion;
```

Ejemplo para Personas y Logros:

```
SELECT Personas.Apellido, Logros.Logro
FROM Personas
RIGHT JOIN Logros
ON Personas.PersonaID = Logros.PersonaID;
```



Apellido	Logro
Fleming	Descubrimiento de la Penicilina
Eratóstenes	Cálculo del perímetro terrestre
Newton	Ley de gravitación universal
Newton	Desarrollo del Cálculo
	Cura del cancer

FULL OUTER JOIN

El FULL OUTER JOIN retorna todas las filas de la tabla de la izquierda (tabla1) y de la tabla de la derecha (tabla2).
El FULL OUTER JOIN combina el resultado de LEFT y RIGHT JOIN. En este caso podemos tener valores NULL de ambos lados.

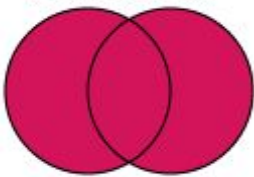
Sintaxis del FULL OUTER JOIN:

```
SELECT nombres_columnas
FROM tabla1
FULL OUTER JOIN tabla2
ON tabla1.columna_relacion=tabla2.columna_relacion;
```

Ejemplo para Personas y Logros:

```
SELECT Personas.Apellido, Logros.Logro
FROM Personas
FULL OUTER JOIN Logros
ON Personas.PersonaID = Logros.PersonaID;
```

FULL OUTER JOIN



Apellido	Logro
Fleming	Descubrimiento de la Penicilina
Eratóstenes	Cálculo del perímetro terrestre
Newton	Ley de gravitación universal
Newton	Desarrollo del Cálculo
	Cura del cancer
Fernandez	

PRÁCTICA GUIADA 2

Joins SQL

ASPECTOS DE PERFORMANCE



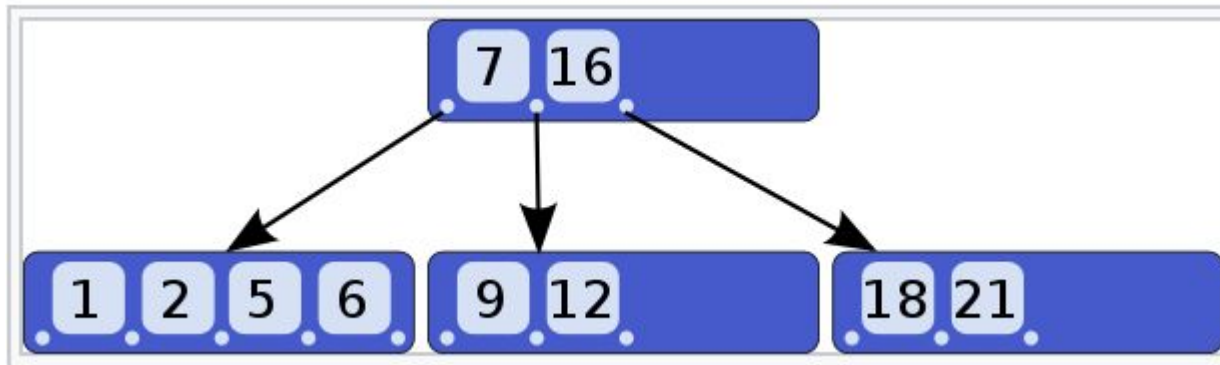
Para los siguientes ejercicios, vamos a crear una base de datos northwind pero esta vez local para tener permisos tanto de acceso como de modificación.

Ahora cada alumno trabajará sobre su propia base. Ya no consultamos todos la misma información.

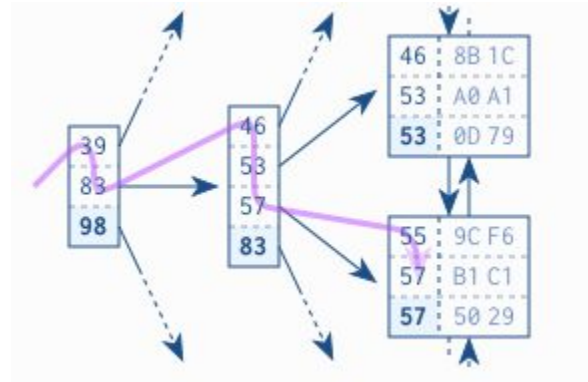
Para crear la DB, conectarse a localhost desde pgAdmin (por default el usuario y contraseña es postgresql), crear una base de datos vacía y ejecutar el archivo SQL que se encuentra [aquí](#)

- Los índices se utilizan para aumentar la performance de las bases de datos.
- Un índice permite consultar determinadas filas específicas de una base de datos de forma más rápida.
- Sin embargo, los índices agregan trabajo a la hora de almacenar nuevos datos, razón por la cual, hay que usarlos con precaución.
- Existen varios tipos de índices con distintos algoritmos que se utilizan para optimizar distintos tipos de consultas. Vamos a desarrollar
 - B-tree
 - Hash

- Es el índice por default de PostgreSQL
- Este tipo de índice construye un árbol de manera “equilibrada”, lo cual permite acceder a la ubicación en memoria de cualquier elemento en el mismo número de etapas.
- El algoritmo consiste en construir un grafo donde siempre hay una cantidad de nodos hijos que respeta un rango prefijado.

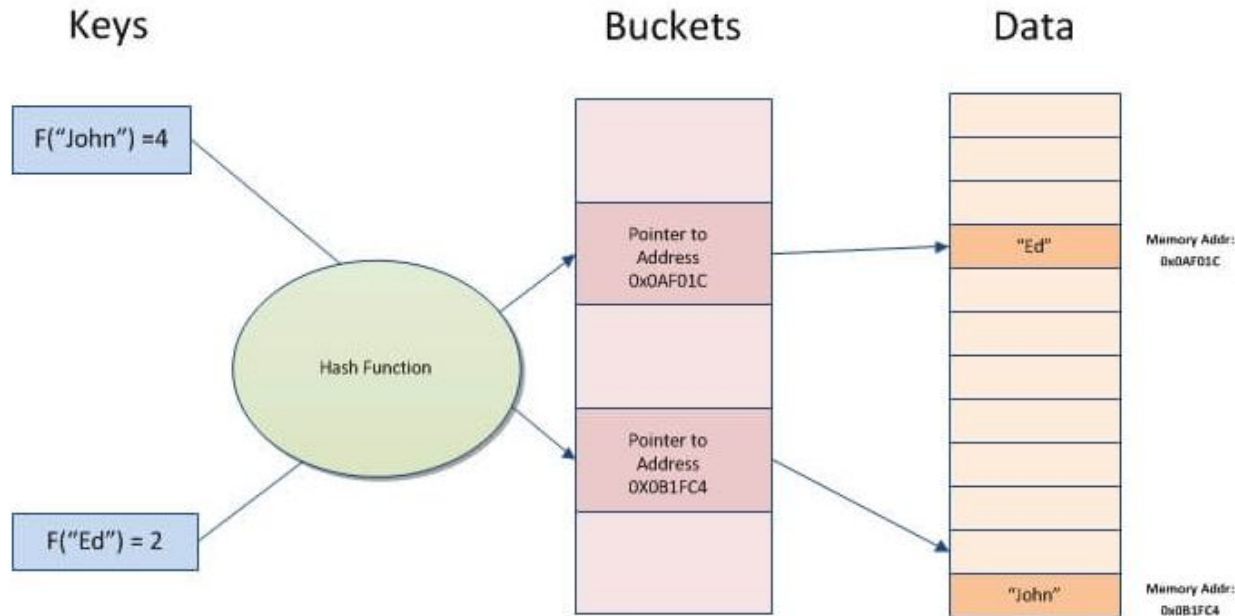


- Para hacer una búsqueda en el grafo se espera a encontrar un elemento por lo menos mayor que el número buscado para pasar al siguiente nivel.
- Este es un ejemplo del recorrido de una búsqueda para encontrar el número 57:



- Escalabilidad logarítmica: la cantidad de registros que puede manejar un índice crece exponencialmente con la profundidad del árbol..
- Este tipo de índice optimiza las operaciones SQL de tipo "<", ">", ">=", "<=", "=", las derivadas de estas como BETWEEN e IN y las comparaciones IS NULL e IS

- Los índices de tipo hash aplican una función matemática al contenido para crear un nuevo valor o "hash". Ideal para texto, el hash es una representación más corta del texto original, útil para indexar.
- Útiles para operaciones de "=" ó "<>" pero no comparaciones



- Las bases de datos PostgreSQL dan información al usuario sobre la forma en la que realizan las consultas.
- Para identificar la ejecución de una consulta, se puede utilizar EXPLAIN o EXPLAIN ANALYSE. Mientras EXPLAIN hace una estimación de los pasos a realizar, EXPLAIN ANALYSE efectivamente ejecuta la consulta.
- Alguna de la información relevante que veremos en el EXPLAIN:
 - Tiempo estimado de ejecución
 - Tiempo real de ejecución (EXPLAIN ANALYSE)
 - Cantidad de filas devueltas
 - Tablas examinadas
 - Si las búsquedas son secuenciales o sobre índices

PRÁCTICA GUIADA 3

Query Explain

CONCLUSIONES

- Las Bases de Datos relacionales son las más utilizadas.
- Organizan sus datos en tablas, con filas y columnas.
- Aprendimos la sintaxis básica de SQL
- En adelante, vamos a tener en cuenta los aspectos de performance

INTRODUCCIÓN

BASES DE DATOS Y SQL