

Sentencias DML en SQL (INSERT, SELECT, UPDATE y DELETE)



diego.coder26 · Follow

9 min read · Aug 9, 2023



Share

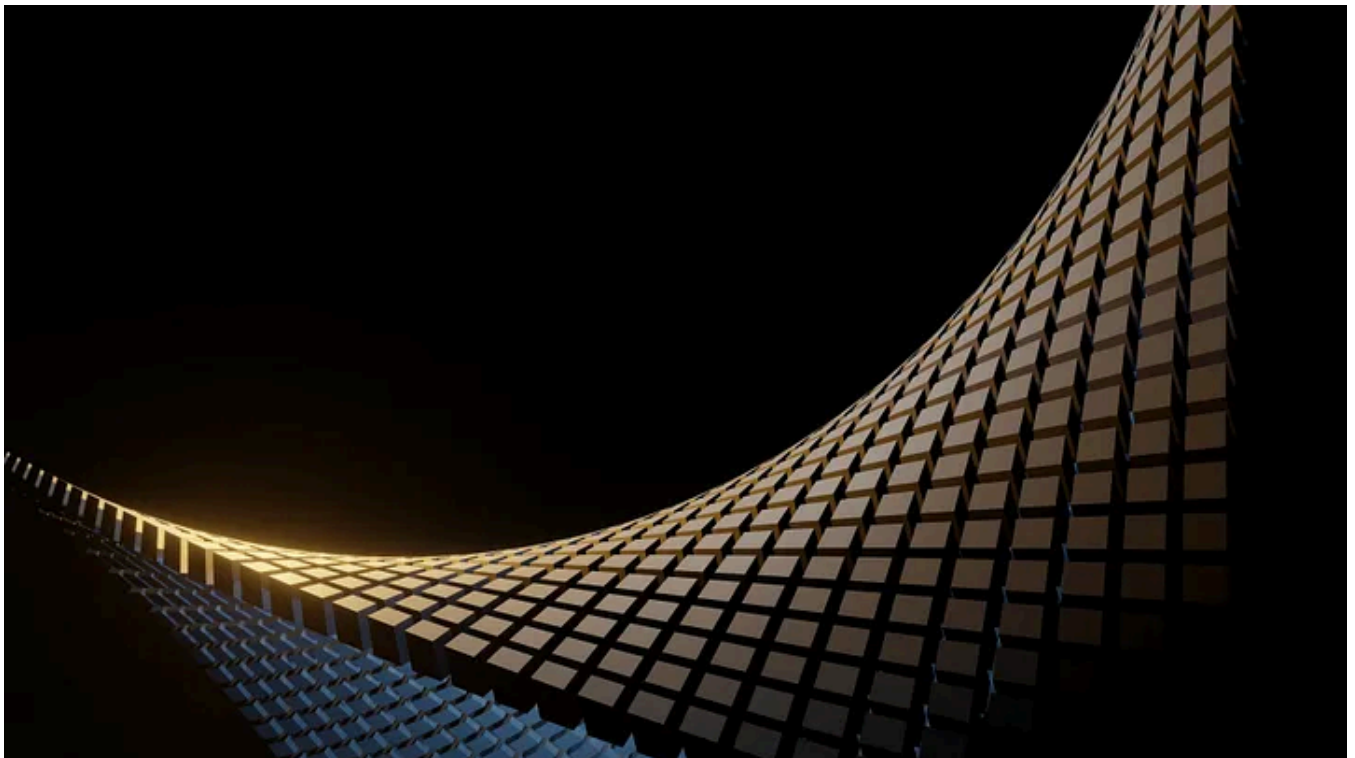


Photo by [Shubham Dhage](#) on [Unsplash](#)

Artículo básico sobre SQL. Sentencias DML, casos de uso y ejemplos (INSERT, SELECT, UPDATE y DELETE).

Recomendaciones

Antes de empezar a estudiar sentencias SQL DML, se recomienda contar con algunos conocimientos previos antes de leer este artículo, por lo cual, dejo la invitación a revisar los siguientes enlaces de apoyo opcionales:

--	--

Sentencias DDL en SQL (CREATE, ALTER Y DROP)

Artículo básico sobre SQL. Sentencias DDL, casos de uso y ejemplos (CREATE, ALTER y DROP).

medium.com

Introducción a PostgreSQL

Artículo básico de PostgreSQL. Composición, conceptos y estructura de referencia.

medium.com

Sentencias DML en SQL

Las sentencias DML (Data Manipulation Language) son un conjunto de instrucciones SQL utilizadas para manipular los datos almacenados en una base de datos. Estas sentencias permiten realizar operaciones como la inserción, actualización, eliminación y recuperación de datos.

Algunas de las sentencias DDL incluyen:

1. INSERT.
2. SELECT.
3. UPDATE.
4. DELETE.

A continuación revisaremos cada una de estas sentencias con algunos ejemplos básicos.

NOTA 🤖: En el contexto de SQL y las bases de datos, los términos “sentencia” y “comando” a menudo se utilizan indistintamente para referirse a una instrucción SQL específica que realiza una acción en la base de datos.

Sandbox

Antes de comenzar con los ejemplos, ejecutaremos las siguientes sentencias DML del artículo anterior para poder efectuar los ejemplos de manipulación.

```
CREATE DATABASE employees;

CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    age SMALLINT,
    salary DECIMAL(10, 2),
    contractor_date DATE,
    email VARCHAR(100),
    active BOOLEAN
);
```

1- INSERT

El comando `INSERT` en SQL se utiliza para insertar nuevos registros en una tabla de una base de datos. Esta sentencia permite agregar datos a la tabla, creando nuevas filas con valores específicos en las columnas correspondientes.

La sintaxis básica de un comando `INSERT` en SQL es la siguiente:

```
INSERT INTO [tabla] (columna1, columna2, columna3) VALUES (valor1, valor2, valor3);
```

- `nombre_tabla` : Es el nombre de la tabla en la que deseas insertar datos.
- `columna1, columna2, columna3, ...` : Son los nombres de las columnas a las que deseas insertar valores.
- `valor1, valor2, valor3, ...` : Son los valores que deseas insertar en las columnas correspondientes.

Ejemplo de cómo se usaría la sintaxis:

- Insertar valores en la tabla empleados

```
INSERT INTO employees (name, age, salary, contractor_date, email, active)
VALUES ('Ana Martínez', 35, 60000.00, '2023-11-15', 'ana@example.com', true),
       ('Juan López', 42, 70000.00, '2023-10-20', 'juan@example.com', false),
       ('María Rodríguez', 28, 55000.00, '2023-09-05', 'maria@example.com', true);
```

```
( 'Luisa García', 31, 52000.00, '2023-12-03', 'luisa@example.com', false )  
( 'Pedro Sánchez', 39, 68000.00, '2023-07-25', 'pedro@example.com', true )  
( 'Marta Fernández', 33, 62000.00, '2023-06-14', 'marta@example.com', false )  
( 'Carlos Pérez', 45, 75000.00, '2023-05-10', 'carlos@example.com', true )  
( 'Sofía Gómez', 27, 51000.00, '2023-04-22', 'sofia@example.com', false ),  
( 'Javier Ruiz', 37, 64000.00, '2023-03-18', 'javier@example.com', true ),  
( 'Laura López', 30, 53000.00, '2023-02-09', 'laura@example.com', false );
```

Hemos omitido la columna `id`, ya que es una columna `SERIAL` y se generará automáticamente. A su vez, debemos asegurarnos de que los valores coincidan con los tipos de datos especificados en la definición de la tabla. Los valores numéricos no requieren comillas, pero las fechas y los valores de texto deben ir entre comillas simples. El valor `true` para la columna `active` se interpreta como un valor booleano (Revisar documentación de SGBD utilizada).

2- SELECT

El comando `SELECT` es una sentencia SQL fundamental que se utiliza para recuperar datos de una o más tablas en una base de datos. Su función principal es realizar consultas y extraer información específica de la base de datos en función de las condiciones y criterios que se determinen. El comando `SELECT` es esencial para obtener información de la base de datos y presentarla de manera legible y útil para los usuarios.

La sintaxis básica de un comando `SELECT` en SQL es la siguiente:

```
SELECT [columna1, columna2, ...] FROM [tabla];
```

- `columnas` : Son las columnas específicas de las que deseas recuperar datos. Puedes seleccionar todas las columnas usando `*`.
- `tabla` : Es la tabla desde la cual deseas recuperar los datos.

Ejemplos de cómo se usaría la sintaxis:

- Obtener todos los registros de la tabla empleados (*)

```
SELECT * FROM employees;
```

En este caso hemos utilizado (*) como comodín para la obtención de todos los registros. En muchos casos, es recomendable especificar las columnas que necesitas en lugar de seleccionar todas con * , ya que esto puede reducir el tiempo y los recursos necesarios para ejecutar la consulta.

- Obtener todos los registros de la tabla empleados columna por columna

```
SELECT id, name, age, salary, contractor_date, email, active  
FROM employees;
```

Cláusulas en una sentencia SELECT

Como se mencionó anteriormente, la sentencia SELECT tiene la capacidad de extraer datos de acuerdo a condiciones y criterios que se determinen. Estas condiciones se realizan mediante cláusulas que se agregan adicionalmente a la sentencia.

A continuación el listado de cláusulas más utilizadas con la sentencia SELECT

- WHERE.
- GROUP BY.
- HAVING..
- ORDER BY.
- LIMIT.
- OFFSET.
- DISTINCT.
- JOIN.

A continuación algunos ejemplos con cada cláusula, descartando “JOIN” debido a que se abordará en otro artículo. 🐼

* WHERE

La cláusula `WHERE` se utiliza para filtrar filas basadas en una condición especificada. Esta cláusula permite restringir el conjunto de resultados devueltos por una consulta SQL a solo aquellos que cumplan con ciertas condiciones. A su vez, también podemos concatenar múltiples condiciones a la cláusula `WHERE` utilizando los operadores lógicos de `AND` y `OR`.

Por ejemplo, para recuperar todos los registros de la tabla `employees` donde el estado de la columna `active` sea verdadero lo podemos hacer de la siguiente manera utilizando `WHERE`.

Open in app ↗

Sign up

Sign in



Search



Otro caso, para seleccionar todos los empleados que tienen más de 30 años y un salario superior a \$50000 debemos utilizar `AND` de la siguiente manera:

```
SELECT *  
FROM employees  
WHERE age > 30 AND salary > 50000;
```

En el caso de que necesitemos cumplir con al menos una condición debemos utilizar `OR`. Por ejemplo si necesitamos seleccionar todos los empleados que tienen menos de 25 años o que fueron contratados después del 1 de enero de 2020 lo podemos hacer de la siguiente manera:

```
SELECT *  
FROM employees  
WHERE age < 25 OR contractor_date > '2020-01-01';
```

* GROUP BY

La cláusula `GROUP BY` en SQL se utiliza para agrupar filas que tienen los mismos valores en una o más columnas y aplicar funciones de agregación, como `SUM()`,

COUNT() , AVG() , MAX() , MIN() , etc., a los grupos resultantes.

Para calcular la suma total de salarios por edad, utilizaremos GROUP BY con la función de agregación SUM() de la siguiente manera:

```
SELECT age, SUM(salary) AS total_salary_by_age
FROM employees
GROUP BY age;
```

Para contar el número de empleados por edad, utilizaremos GROUP BY con la función de agregación COUNT() de la siguiente manera:

```
SELECT age, COUNT(*) AS qty_employees;
FROM employees
GROUP BY age;
```

Para calcular el salario promedio por edad, utilizaremos GROUP BY con la función de agregación AVG() de la siguiente manera:

```
SELECT age, AVG(salary) AS salary_avg
FROM employees
GROUP BY age;
```

Para encontrar la edad máxima y el salario máximo, utilizaremos GROUP BY con la función de agregación MAX() de la siguiente manera:

```
SELECT MAX(age) AS age_max, MAX(salary) AS salary_max
FROM employees;
```

Por último, para encontrar la edad mínima y el salario mínimo, utilizaremos `GROUP BY` con la función de agregación `MIN()` de la siguiente manera:

```
SELECT MIN(age) AS age_min, MIN(salary) AS salary_min
FROM employees;
```

* HAVING

La cláusula `HAVING` en SQL se utiliza para filtrar los resultados de una consulta basada en condiciones de agregación. A diferencia de la cláusula `WHERE`, que se utiliza para filtrar filas antes de que se realice la agregación, la cláusula `HAVING` se aplica después de que se han agrupado los datos y se han aplicado las funciones de agregación.

Por ejemplo, para contar los empleados por edad y filtrar grupos con más de 5 empleados, lo podemos hacer de la siguiente manera:

```
SELECT age, COUNT(*) AS cantidad_empleados
FROM employees
GROUP BY age
HAVING COUNT(*) > 5;
```

Este ejemplo primero cuenta el número de empleados por edad y luego filtra los grupos de edad que tienen más de 5 empleados.

* ORDER BY

La cláusula `ORDER BY` en SQL se utiliza para ordenar los resultados de una consulta basada en el valor de una o más columnas en orden ascendente o descendente. Es una de las cláusulas más comunes en SQL y es útil cuando necesitas que los resultados de una consulta estén ordenados de una manera específica.

Por ejemplo, utilizaremos la cláusula `ORDER BY` para ordenar los resultados por la columna `salary` en orden descendente o ascendente según sea el caso:

```
SELECT salary
FROM employees
```



```
ORDER BY salary DESC;  
  
SELECT salary  
FROM employees  
ORDER BY salary ASC;
```

* LIMIT

La cláusula `LIMIT` en SQL se utiliza para limitar el número de filas que se devuelven en el resultado de una consulta. Es una cláusula muy útil cuando solo deseas ver una cantidad específica de filas en lugar de todas las que cumplen con los criterios de la consulta.

Por ejemplo, para obtener los primeros 3 empleados con el salary más alto lo podemos hacer de esta manera:

```
SELECT * FROM employees  
ORDER BY salary DESC  
LIMIT 3;
```

* OFFSET

La cláusula `OFFSET` en SQL se utiliza junto con la cláusula `LIMIT` para desplazar el punto de inicio desde donde se devolverán las filas en el resultado de una consulta. En otras palabras, `OFFSET` se utiliza para omitir un número específico de filas al principio del conjunto de resultados. Esta cláusula es útil cuando deseas paginar los resultados de una consulta SQL.

Por ejemplo, para obtener los empleados a partir del tercer empleado (es decir, omitir los primeros dos) lo podemos hacer de la siguiente manera:

```
SELECT *  
FROM employees  
OFFSET 2  
LIMIT 10;
```

* DISTINCT

La palabra clave `DISTINCT` en SQL se utiliza para eliminar los duplicados de los resultados de una consulta. Cuando se utiliza la palabra clave `DISTINCT` en una consulta, se garantiza que solo se devolverán valores únicos en el conjunto de resultados, eliminando cualquier duplicado que pueda haber.

Por ejemplo, para obtener una lista de correos electrónicos únicos de los empleados, lo podemos hacer de la siguiente manera:

```
SELECT DISTINCT email
FROM employees;
```

3- UPDATE

El comando `UPDATE` en SQL se utiliza para modificar o actualizar los registros existentes en una tabla. Esta sentencia permite cambiar los valores de una o más columnas en las filas de la tabla que cumplan con una condición específica. Es útil cuando necesitas actualizar información en la base de datos, como corregir errores, cambiar valores obsoletos o realizar actualizaciones masivas.

La sintaxis básica de un comando `UPDATE` en SQL es la siguiente:

```
UPDATE nombre_tabla
SET columna1 = nuevo_valor1, columna2 = nuevo_valor2, ...
WHERE condicion;
```

- `nombre_tabla` : Es el nombre de la tabla en la que deseas realizar la actualización.
- `columna1` , `columna2` , etc.: Son las columnas que deseas actualizar.
- `nuevo_valor1` , `nuevo_valor2` , etc.: Son los nuevos valores que deseas asignar a las columnas correspondientes.
- `condicion` : Es una expresión que determina qué filas serán actualizadas. Si no se proporciona una condición, todos los registros en la tabla se actualizarán.

Ejemplo de cómo se usaría la sintaxis:

- Actualizar los todos valores de todos salarios de los empleados en un 10%

```
UPDATE employees
SET salary = salary * 1.10;
```

- Actualizar el salario de un empleado específico cuyo nombre es “Pedro Sánchez”

```
UPDATE employees
SET salary = 60000
WHERE name = 'Pedro Sánchez';
```

- Actualizar el salario, la edad y el estado de activo de un empleado cuyo nombre es “Marta Fernández”

```
UPDATE employees
SET salary = 70000,
    age = 40,
    active = false
WHERE name = 'Marta Fernández';
```

4- DELETE

El comando `DELETE` en SQL se utiliza para eliminar registros de una tabla en una base de datos. Esta sentencia permite eliminar una o varias filas que cumplan con una condición específica. El comando `DELETE` es útil cuando necesitas eliminar datos que ya no son necesarios o que son incorrectos en la base de datos.

La sintaxis básica de un comando `DELETE` en SQL es la siguiente:

```
DELETE FROM nombre_tabla
WHERE condicion;
```

- `nombre_tabla` : Es el nombre de la tabla de la que deseas eliminar registros.

- **condicion** : Es una expresión que determina qué filas serán eliminadas. Si no se proporciona una condición, todos los registros en la tabla serán eliminados.

Ejemplo de cómo se usaría la sintaxis:

- Eliminar el empleado con el ID 3 en nuestra tabla

```
DELETE FROM employees  
WHERE id = 3;
```

Es importante tener cuidado al utilizar el comando `DELETE`, ya que eliminará permanentemente los registros de la tabla. Asegúrate de proporcionar una condición precisa para evitar eliminar datos incorrectos y considera realizar copias de respaldo antes de realizar operaciones de eliminación masiva. A su vez, en muchos SGBD el comando `DELETE` por defecto para ejecutarlo es necesario contar con la cláusula `WHERE` de lo contrario no te dejará realizar la ejecución.

Código fuente

GitHub - dcortesnet/SQL-fundamentals: Fundamentals of the SQL language. Repository with examples...

Fundamentals of the SQL language. Repository with examples and basic exercises of queries, create tables, update, and...

github.com

Libros para aprender SQL

- “SQL For Dummies” por Allen G. Taylor (Principiante).
- “Learning SQL” por Alan Beaulieu (Principiante — intermedio).
- “SQL Performance Explained” por Markus Winand (Avanzado).

Gracias por haber llegado hasta aquí, si encuentras esto útil, no olvides aplaudir 🙌
. Suscríbete para obtener más contenido 🔔.

Si necesita ayuda adicional, comuníquese conmigo 🙋.

@diego.coder | Linktree

Systems analyst focused on programming with more than 5+ years of experience.

linktr.ee

Muchas gracias por leer, agradezco su tiempo.

Database

Relational Databases

Sql

Sql Dml



Follow

Written by diego.coder26

413 Followers

Hello, my name is Diego! 🙋, I am Software Engineer that writes about computer science, mathematics, and personal growth. 👉 🤖 linktr.ee/diego.coder

More from diego.coder26



 diego.coder26

Instalar múltiples versiones de Node.js en Windows con NVM (Node version Manager)

Artículo básico sobre la instalación de la herramienta NVM. Comandos y ejemplos.

4 min read · Dec 30, 2023



9



1



 diego.coder26

Presente simple y presente continuo (reglas gramaticales en inglés)

Artículo básico sobre el presente simple y continuo. Conceptos, verbo to be, reglas y auxiliares.

4 min read · Jul 24, 2022



8



diego.coder26

Las mejores APIs públicas para practicar programación

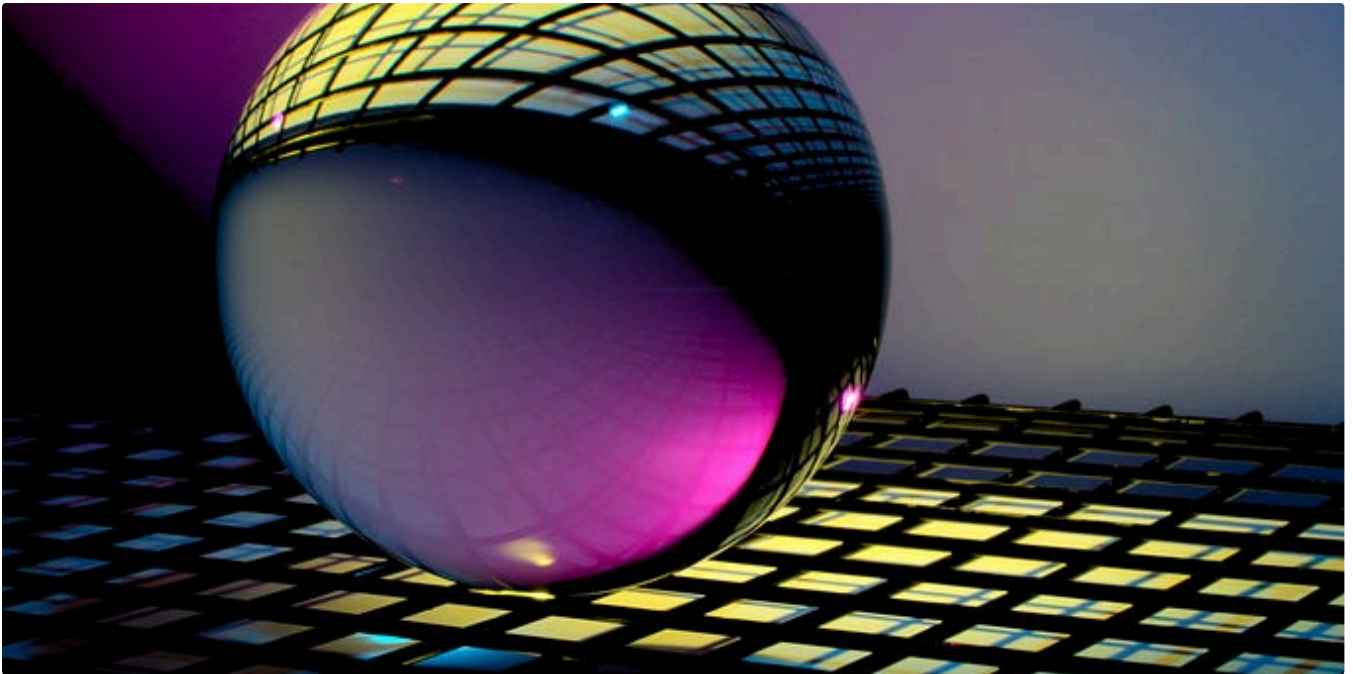
Artículo sobre las mejores APIs para practicar y modelar nuestros desarrollos.

4 min read · Jan 10, 2024



18





 diego.coder26

Introducción a las “Clean Architectures”

Artículo básico sobre arquitecturas limpias. Definición, características, capas, variantes y consideraciones.

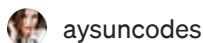
8 min read · Aug 1, 2023

 32 



See all from diego.coder26

Recommended from Medium



MySQL stands as an indispensable tool for developers, database administrators, and anyone involved in the management of structured data. As...

3 min read · Feb 12, 2024



Çağatay Uncu in Devops Türkiye

Lazy Loading and Eager Loading: Data Loading Methods and Performance Comparisons

In software development, Lazy Loading and Eager Loading are two prominent strategies that affect an application's performance based on how...

2 min read · Dec 22, 2023

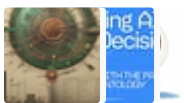


Lists



ChatGPT

21 stories · 671 saves



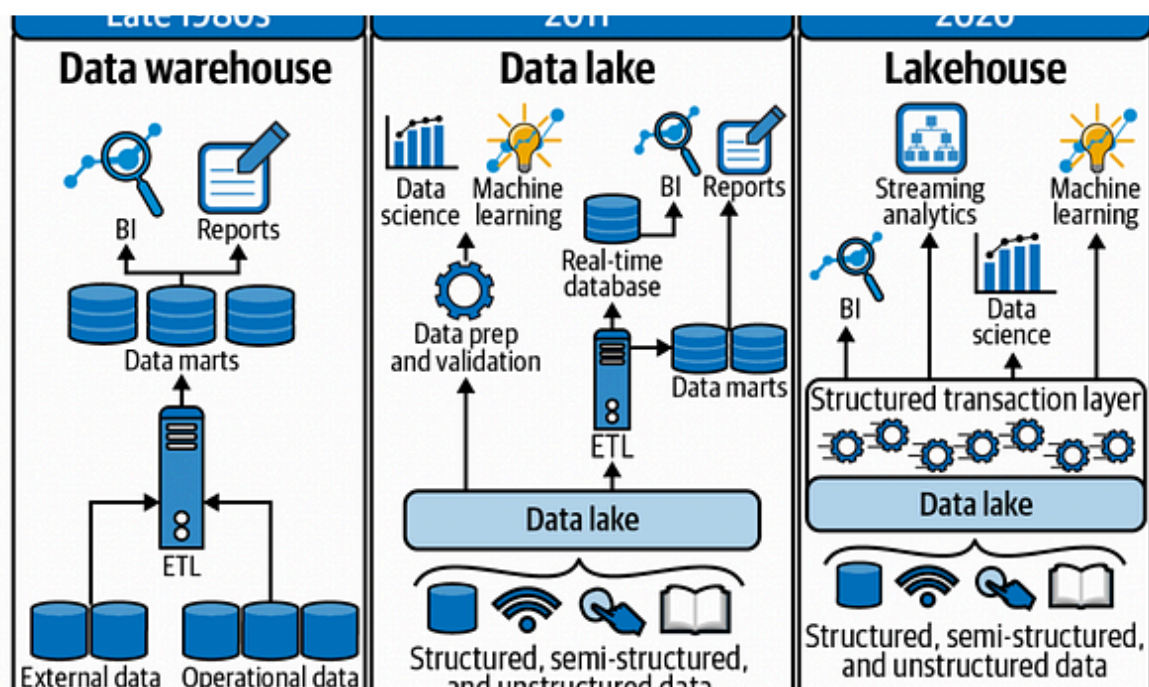
data science and AI

40 stories · 175 saves



Natural Language Processing

1498 stories · 1018 saves



Ihor Lukianov

A Brief History of Data Management—From Relational Databases to Data Lakehouses

How we evolved to modern data management approaches and what should we know as Data Engineers

5 min read · 6 days ago



176



2



A.C. Green	33	6-9	225	Oregon State	USA	1985	1	23
Aaron McKie	24	6-5	209	Temple	USA	1994	1	17
Aaron Williams	25	6-9	225	Xavier	USA	Undrafted	Undrafted	Undrafted
Acie Earl	27	6-11	240	Iowa	USA	1993	1	19
Adam Keefe	27	6-9	241	Stanford	USA	1992	1	10
Adrian Caldwell	30	6-8	265	Lamar	USA	Undrafted	Undrafted	Undrafted
Alan Henderson	24	6-9	235	Indiana	USA	1995	1	16
Aleksandar Djordjevic	29	6-2	198	None	Serbia	Undrafted	Undrafted	Undrafted
Allan Houston	26	6-6	200	Tennessee	USA	1993	1	11



Chayan Shrang Raj

SQL Window Functions and CTEs

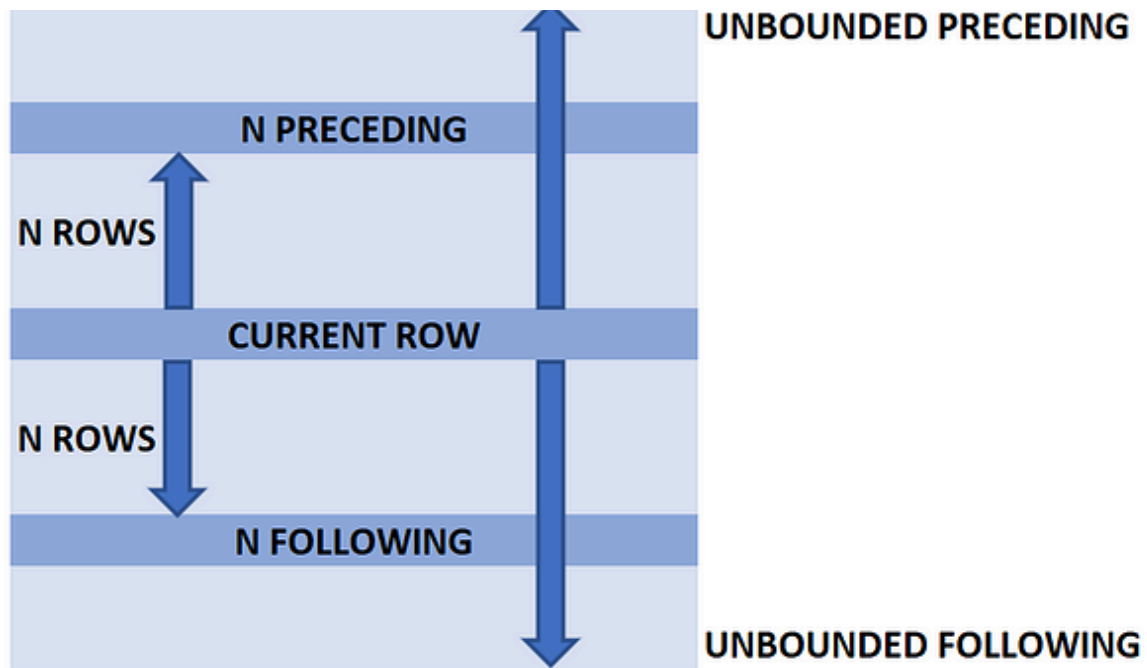
Window functions are significantly different from other available SQL functions and keep a special place in a Data Engineer's heart. It is...


8 min read · May 8, 2024



112





 Ritu Santra

Window Functions—ROWS Clause

2 min read · Jan 26, 2024

 54 



 Rishabh Agarwal

Views in Relational Databases

Simply put, Views are stored queries that when invoked produce some results!

4 min read · Dec 20, 2023

 22





See more recommendations