SQL

Funciones de agregación y cálculo

CertiDevs

Índice de contenidos

1. Funciones de agregación en SQL

Las **funciones de agregación** son útiles para realizar cálculos en un conjunto de filas y devolver un único valor.

En SQL, algunas de las funciones de agregación más comunes incluyen `COUNT`, `SUM`, `AVG`, `MIN` y `MAX`. En esta sección, exploraremos cómo utilizar estas funciones en consultas SQL con ejemplos avanzados.

1.1. Función COUNT

La función `COUNT` se utiliza para contar el número de filas en una tabla o el número de filas que coinciden con una condición específica. La sintaxis básica es:

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

Por ejemplo, para contar el número total de empleados en la tabla employees:

```
SELECT COUNT(*)
FROM employees;
```

Para contar el número de empleados con un salario mayor a 50000 en la tabla employees:

```
SELECT COUNT(*)
FROM employees
WHERE salary > 50000;
```

1.2. Función SUM

La función `SUM` se utiliza para calcular la suma de los valores de una columna numérica. La sintaxis básica es:

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

Por ejemplo, para calcular la suma de los salarios de todos los empleados en la tabla employees:

```
SELECT SUM(salary)
FROM employees;
```

1.3. Función AVG

La función `AVG` se utiliza para calcular el promedio de los valores de una columna numérica. La sintaxis básica es:

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

Por ejemplo, para calcular el salario promedio de todos los empleados en la tabla employees:

```
SELECT AVG(salary)
FROM employees;
```

1.4. Funciones MIN y MAX

Las funciones `MIN` y `MAX` se utilizan para encontrar el valor mínimo y máximo de una columna, respectivamente. La sintaxis básica es:

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

SELECT MAX(column)
FROM table_name
WHERE condition;
```

Por ejemplo, para encontrar el salario mínimo y máximo de todos los empleados en la tabla employees:

```
SELECT MIN(salary), MAX(salary)
FROM employees;
```

1.5. Cláusula GROUP BY

La cláusula `GROUP BY` se utiliza junto con funciones de agregación para agrupar los resultados por una o más columnas. La sintaxis básica es:

```
SELECT column1, column2, ..., aggregate_function(column)
FROM table_name
WHERE condition
GROUP BY column1, column2, ...;
```

Por ejemplo, para calcular el salario promedio de los empleados en cada departamento de la tabla employees:

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

1.6. Cláusula HAVING

La cláusula `HAVING` se utiliza junto con la cláusula `GROUP BY` para filtrar los resultados de las funciones de agregación basándose en una condición. La sintaxis básica es:

```
SELECT column1, column2, ..., aggregate_function(column)
FROM table_name
WHERE condition
GROUP BY column1, column2, ...
HAVING condition;
```

Por ejemplo, para calcular el salario promedio de los empleados en cada departamento de la tabla employees, pero solo mostrar los departamentos con un salario promedio superior a 50000:

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 50000;
```

1.7. Agrupar por múltiples columnas

Puede utilizar la cláusula `GROUP BY` con múltiples columnas para agrupar los resultados en función de la combinación de valores de esas columnas. Por ejemplo, para calcular el salario promedio de los empleados en cada departamento y cada puesto en la tabla employees:

```
SELECT department_id, job_title, AVG(salary)
FROM employees
GROUP BY department_id, job_title;
```

1.8. GROUP BY con ORDER BY

La cláusula `ORDER BY` se puede utilizar junto con `GROUP BY` para ordenar los resultados de una consulta de agregación. Por ejemplo, para calcular el salario promedio de los empleados en cada departamento de la tabla employees y ordenar los resultados por el salario promedio de mayor a menor:

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
ORDER BY AVG(salary) DESC;
```

1.8.1. Ejemplo 1:

Ejemplo: Número de empleados por departamento con al menos 5 empleados.

En este ejemplo, usaremos la cláusula GROUP BY para contar el número de empleados en cada departamento de la tabla employees, pero solo mostraremos los departamentos que tienen al menos 5 empleados usando la cláusula HAVING:

```
SELECT department_id, COUNT()
FROM employees
GROUP BY department_id
HAVING COUNT() >= 5;
```

1.9. Ejemplo 2:

Ejemplo 2: Salario mínimo, máximo y promedio por departamento con al menos 3 empleados.

Aquí, calcularemos el salario mínimo, máximo y promedio de los empleados en cada departamento de la tabla employees, pero solo mostraremos los departamentos que tienen al menos 3 empleados:

```
SELECT department_id, MIN(salary) AS min_salary, MAX(salary) AS max_salary, AVG(
salary) AS avg_salary
FROM employees
GROUP BY department_id
HAVING COUNT(*) >= 3;
```

1.10. Ejemplo 3:

Ejemplo 3: Departamentos con una diferencia salarial superior a 30000

En este ejemplo, encontraremos los departamentos de la tabla employees donde la diferencia entre el salario máximo y mínimo es superior a 30000:

```
SELECT department_id, MAX(salary) - MIN(salary) AS salary_difference
FROM employees
GROUP BY department_id
HAVING salary_difference > 30000;
```

1.11. Ejemplo 4:

Ejemplo 4: Número de empleados por puesto con un salario promedio superior a 70000

Este ejemplo muestra cómo contar el número de empleados por puesto en la tabla employees y solo mostrar los puestos que tienen un salario promedio superior a 70000:

```
SELECT job_title, COUNT(*) AS num_employees, AVG(salary) AS avg_salary
FROM employees
GROUP BY job_title
HAVING avg_salary > 70000;
```

2. Funciones matemáticas

Las **funciones matemáticas** en SQL permiten realizar cálculos y manipulaciones numéricas en los datos de una tabla. A continuación, se presentan algunas funciones matemáticas comunes en SQL y ejemplos de cómo utilizarlas en consultas.

2.1. ABS (valor)

La función `ABS` devuelve el valor absoluto de un número.

Ejemplo: Calcular el valor absoluto de la diferencia de salario entre dos empleados con IDs 101 y 102:

```
SELECT ABS((SELECT salary FROM employees WHERE employee_id = 101) - (SELECT salary
FROM employees WHERE employee_id = 102)) AS salary_difference;
```

2.2. ROUND (valor, [decimales])

La función `ROUND` redondea un número decimal al número entero más cercano o a la cantidad especificada de decimales.

Ejemplo: Redondear el salario promedio de los empleados al entero más cercano:

```
SELECT ROUND(AVG(salary), 2) AS rounded_avg_salary
FROM employees;
```

2.3. CEIL (valor) y FLOOR (valor)

Las funciones `CEIL` y `FLOOR` redondean un número hacia arriba (al entero más pequeño pero igual o mayor) y hacia abajo (al entero más grande pero igual o menor) respectivamente.

Ejemplo: Calcular el entero más cercano hacia arriba y hacia abajo del salario promedio de los

empleados:

```
SELECT CEIL(AVG(salary)) AS ceiling_avg_salary, FLOOR(AVG(salary)) AS floor_avg_salary
FROM employees;
```

2.4. POWER (base, exponente)

La función `POWER` devuelve el resultado de elevar una base a un exponente dado.

Ejemplo: Calcular el cuadrado y el cubo del salario de un empleado con ID 101:

```
SELECT employee_id, salary, POWER(salary, 2) AS squared_salary, POWER(salary, 3) AS
cubed_salary
FROM employees
WHERE employee_id = 101;
```

2.5. SQRT (valor)

La función `SQRT` devuelve la raíz cuadrada de un número.

Ejemplo: Calcular la raíz cuadrada del salario de cada empleado:

```
SELECT employee_id, salary, SQRT(salary) AS sqrt_salary
FROM employees;
```

3. Funciones de cadena

Las **funciones de cadena** en SQL permiten manipular y transformar datos de tipo cadena en una base de datos. A continuación, se presentan algunas funciones de cadena comunes en SQL y ejemplos de cómo utilizarlas en consultas.

3.1. CONCAT (cadena1, cadena2, ...)

La función `CONCAT` combina dos o más cadenas en una sola cadena.

Ejemplo: Concatenar el nombre y apellido de los empleados con un espacio en medio:

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM employees;
```

3.2. SUBSTRING (cadena, inicio, [longitud])

La función `SUBSTRING` devuelve una subcadena de una cadena, comenzando en la posición especificada y con la longitud opcional.

Ejemplo: Extraer los primeros 3 caracteres del nombre de cada empleado:

```
SELECT first_name, SUBSTRING(first_name, 1, 3) AS name_prefix
FROM employees;
```

3.3. LOWER (cadena) y UPPER (cadena)

Las funciones `LOWER` y `UPPER` convierten una cadena a minúsculas y mayúsculas respectivamente.

Ejemplo: Convertir el nombre y apellido de los empleados a mayúsculas:

```
SELECT UPPER(first_name) AS upper_first_name, UPPER(last_name) AS upper_last_name
FROM employees;
```

3.4. LENGTH (cadena)

La función `LENGTH` devuelve la longitud de una cadena en caracteres.

Ejemplo: Calcular la longitud del nombre y apellido de los empleados:

```
SELECT first_name, last_name, LENGTH(first_name) AS name_length, LENGTH(last_name) AS
last_name_length
FROM employees;
```

3.5. REPLACE (cadena, cadena_buscada, cadena_reemplazo)

La función `REPLACE` reemplaza todas las apariciones de una cadena buscada con otra cadena de reemplazo en una cadena original.

Ejemplo: Reemplazar todas las apariciones de 'John' en el nombre de los empleados con 'Jonathan':

```
SELECT first_name, REPLACE(first_name, 'John', 'Jonathan') AS modified_first_name
FROM employees
WHERE first_name LIKE 'John%';
```

4. Funciones de fecha y hora

Las funciones de fecha y hora en SQL permiten manipular y transformar datos de tipo fecha y hora en una base de datos. A continuación, se presentan algunas funciones de fecha y hora comunes en SQL y ejemplos de cómo utilizarlas en consultas.

4.1. CURRENT_DATE, CURRENT_TIME y CURRENT_TIMESTAMP

Las funciones `CURRENT_DATE`, `CURRENT_TIME` y `CURRENT_TIMESTAMP` devuelven la fecha actual, la hora actual y la fecha y hora actuales, respectivamente.

Ejemplo: Obtener la fecha y hora actual del servidor:

```
SELECT CURRENT_DATE AS today, CURRENT_TIME AS now, CURRENT_TIMESTAMP AS timestamp;
```

4.2. DATE_ADD(fecha, INTERVAL valor unidad)

La función `DATE_ADD` suma un intervalo de tiempo a una fecha. La unidad puede ser `SECOND`, `MINUTE`, `HOUR`, `DAY`, `WEEK`, `MONTH` o `YEAR`.

Ejemplo: Calcular la fecha de nacimiento de los empleados más 30 días:

```
SELECT birthdate, DATE_ADD(birthdate, INTERVAL 30 DAY) AS birthdate_plus_30_days
FROM employees;
```

4.3. DATEDIFF (unidad, fecha1, fecha2)

La función `DATEDIFF` devuelve la diferencia entre dos fechas en la unidad especificada. La unidad puede ser SECOND, MINUTE, HOUR, DAY, WEEK, MONTH o YEAR.

Ejemplo: Calcular la diferencia en días entre la fecha de contratación y la fecha de nacimiento de los empleados:

```
SELECT birthdate, hire_date, DATEDIFF(DAY, birthdate, hire_date) AS days_difference
FROM employees;
```

4.4. EXTRACT (unidad FROM fecha)

La función `EXTRACT` devuelve una parte de una fecha como un número entero. La unidad puede ser SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, YEAR, etc.

Ejemplo: Extraer el año, mes y día de la fecha de nacimiento de los empleados:

```
SELECT birthdate,
EXTRACT(YEAR FROM birthdate) AS birth_year,
EXTRACT(MONTH FROM birthdate) AS birth_month,
EXTRACT(DAY FROM birthdate) AS birth_day
FROM employees;
```

4.5. DATE_FORMAT (fecha, formato)

La función `DATE_FORMAT` convierte una fecha en una cadena formateada según el formato especificado.

Ejemplo: Formatear la fecha de nacimiento de los empleados como "YYYY-MM-DD":

```
SELECT birthdate, DATE_FORMAT(birthdate, '%Y-%m-%d') AS formatted_birthdate
FROM employees;
```

5. Funciones de conversión de tipos de datos

Las funciones de conversión de tipos de datos en SQL permiten cambiar el tipo de datos de una expresión o valor. A continuación, se presentan algunas funciones de conversión de tipos de datos comunes en SQL y ejemplos de cómo utilizarlas en consultas.

5.1. CAST (expresión AS tipo_dato)

La función `CAST` convierte una expresión al tipo de datos especificado.

Ejemplo: Convertir un número decimal a entero:

```
SELECT price, CAST(price AS INTEGER) AS price_integer
FROM products;
```

5.2. CONVERT (tipo_dato, expresión)

La función `CONVERT` es similar a `CAST` y también convierte una expresión al tipo de datos especificado.

Ejemplo: Convertir una fecha en una cadena de caracteres formateada:

```
SELECT birthdate, CONVERT(VARCHAR(10), birthdate, 23) AS formatted_birthdate
FROM employees;
```

5.3. PARSE (expresión AS tipo_dato)

La función `PARSE` es específica de SQL Server y convierte una cadena de caracteres en el tipo de datos especificado, utilizando la cultura especificada.

Ejemplo: Convertir una cadena de caracteres en fecha:

```
SELECT date_string, PARSE(date_string AS DATE USING 'en-US') AS parsed_date
FROM date_strings;
```