Тема 4. Язык SQL: расширенные возможности.

Цель занятия:

Ознакомиться с расширенными возможностями языка SQL, включая работу с агрегирующими функциями, вложенными запросами, группировкой данных, фильтрацией сгруппированных данных и использованием псевдонимов (alias) для упрощения запросов.

Учебные вопросы:

1. Агрегирующие функции.
2. Вложенные запросы.
3. Оператор GROUP BY для группировки данных.
4. Оператор HAVING для фильтрации сгруппированных данных.
5. Псевдонимы (Alias).

1. Агрегирующие функции.

Агрегирующие функции в SQL — это функции, которые выполняют вычисления над множеством значений из одного или нескольких столбцов и возвращают одно итоговое значение.

Агрегирующие функции предназначены для обработки и анализа больших объемов данных, чтобы получить сводные результаты. Они позволяют эффективно выполнять операции над группами данных, предоставляя возможность быстро извлекать ключевую информацию, такую как общие суммы, количество записей, минимальные или максимальные значения и средние показатели.

Существует 5 таких функций:

- **COUNT():** Подсчет количества строк.
- **SUM():** Суммирование значений.
- **AVG():** Вычисление среднего значения.
- **MIN():** Поиск минимального значения.
- **MAX():** Поиск максимального значения.

1. COUNT(): Подсчитывает количество строк в наборе данных.

Примеры: Подсчет количества сотрудников в таблице Employees

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

Количество завершенных заказов в таблице Orders:

```
SELECT COUNT(*) FROM Orders WHERE Status = 'Completed';
```

2. SUM(): Вычисляет сумму значений в указанном столбце.

Примеры: Сумма всех зарплат сотрудников.

```
SELECT SUM(Salary) FROM Employees;
```

Запрос вычисляет общую сумму заказов клиента с идентификатором 101:

```
SELECT SUM(OrderAmount) FROM Orders WHERE CustomerID = 101;
```

3. AVG(): Вычисляет среднее значение для выбранного столбца.

Примеры: Средняя зарплата сотрудников:

```
SELECT AVG(Salary) FROM Employees;
```

Запрос возвращает средний возраст сотрудников в отделе кадров:

```
SELECT AVG(Age) FROM Employees WHERE Department = 'HR';
```

4. MIN(): Определяет минимальное значение в столбце.

Пример: Минимальная зарплата среди сотрудников:

```
SELECT MIN(Salary) FROM Employees;
```

5. MAX(): Определяет максимальное значение в столбце.

Примеры: Максимальная зарплата среди сотрудников:

```sql
SELECT MAX(Salary) FROM Employees;
```

Запрос возвращает минимальную и максимальную зарплаты среди сотрудников IT-отдела:

```sql
SELECT MIN(Salary), MAX(Salary) FROM Employees WHERE Department = 'IT';
```

2. Вложенные запросы.

Вложенные запросы (subqueries) в SQL — это запросы, которые находятся внутри других запросов.

Они могут быть использованы в различных частях основного запроса, таких как SELECT, FROM, WHERE, HAVING, и могут возвращать одиночные значения, строки или целые таблицы данных.

Вложенные запросы позволяют выполнять более сложные и гибкие операции с данными.

1. Вложенные запросы в SELECT

Используются для вычисления значений в столбцах основного запроса.

Пример, вложенный запрос находит название отдела для каждого сотрудника:

```
SELECT EmployeeID,
       (SELECT DepartmentName
        FROM Departments
        WHERE Employees.DepartmentID = Departments.DepartmentID) AS DepartmentName
FROM Employees;
```

2. Вложенные запросы в WHERE

Используются для фильтрации данных в основном запросе на основе результатов подзапроса.

Пример, запрос находит сотрудников, работающих в отделе кадров:

```
SELECT EmployeeID, FirstName, LastName
FROM Employees
WHERE DepartmentID = (SELECT DepartmentID
                      FROM Departments
                      WHERE DepartmentName = 'HR');
```

3. Вложенные запросы в FROM

Используются для создания временных таблиц, которые затем могут быть использованы в основном запросе.

Пример, вложенный запрос создает таблицу зарплат для IT-отдела, на основе которой вычисляется средняя зарплата:

```
SELECT AVG(Salary)
FROM (SELECT Salary
      FROM Employees
      WHERE DepartmentID = 1) AS ITDepartmentSalaries;
```

4. Вложенные запросы в HAVING

Используются для фильтрации сгруппированных данных, основанных на агрегированных значениях.

Пример, запрос находит отделы, где количество сотрудников больше среднего по всем отделам:

```
SELECT DepartmentID, COUNT(EmployeeID) AS NumberOfEmployees
FROM Employees
GROUP BY DepartmentID
HAVING COUNT(EmployeeID) > (SELECT AVG(EmployeeCount)
                            FROM (SELECT COUNT(EmployeeID) AS EmployeeCount
                                  FROM Employees
                                  GROUP BY DepartmentID) AS DepartmentEmployeeCounts);
```

Примеры вложенных запросов:

Нахождение максимальной зарплаты в HR – отделе:

```
SELECT FirstName, LastName, Salary
FROM Employees
WHERE Salary = (SELECT MAX(Salary)
                FROM Employees
                WHERE DepartmentID = (SELECT DepartmentID
                                      FROM Departments
                                      WHERE DepartmentName = 'HR'));
```

Основной запрос выбирает столбцы FirstName, LastName и Salary из таблицы Employees

вложенный запрос вычисляет максимальную зарплату среди всех сотрудников, работающих в отделе кадров.

этот запрос выбирает идентификатор отдела (DepartmentID) из таблицы Departments.

3. Оператор GROUP BY.

Оператор GROUP BY используется для группировки строк в результирующем наборе данных по одному или нескольким столбцам.

Применяется совместно с агрегирующими функциями (например, COUNT, SUM, AVG, MAX, MIN), чтобы выполнять вычисления для каждой группы отдельно.

Синтаксис:

```
SELECT column1, column2, AGGREGATE_FUNCTION(column3)
FROM table_name
WHERE condition
GROUP BY column1, column2
HAVING aggregate_condition;
```

column1, column2: Столбцы, по которым будут группироваться строки.

AGGREGATE_FUNCTION(column3): Агрегирующая функция, выполняющая операцию над значениями столбца column3 для каждой группы.

HAVING aggregate_condition: Опциональная часть, которая позволяет фильтровать группы на основании агрегированных значений.

Группировочные столбцы должны обязательно присутствовать в SELECT!

Пример 1: Группировка по одному столбцу

Предположим, у нас есть таблица Sales, содержащая данные о продажах:

SalesID	ProductID	Quantity	SaleDate
1	101	5	2023-01-01
2	102	3	2023-01-02
3	101	2	2023-01-03
4	103	7	2023-01-04
5	102	6	2023-01-05

Чтобы узнать общее количество проданных единиц для каждого продукта, можно использовать следующий запрос:

```
SELECT ProductID, SUM(Quantity) AS TotalQuantity
FROM Sales
GROUP BY ProductID;
```

Результат:

ProductID	TotalQuantity
101	7
102	9
103	7

Этот запрос группирует данные по ProductID и вычисляет сумму количества проданных единиц (SUM(Quantity)) для каждого продукта.

Пример 2: Группировка по нескольким столбцам.

Предположим, что в таблицу Sales добавили еще один столбец Region, и нужно узнать количество проданных единиц для каждого продукта в каждом регионе:

```
SELECT ProductID, Region, SUM(Quantity) AS TotalQuantity
FROM Sales
GROUP BY ProductID, Region;
```

Таблица Sales:

SalesID	ProductID	Quantity	SaleDate	Region
1	101	5	2023-01-01	North
2	102	3	2023-01-02	South
3	101	2	2023-01-03	North
4	103	7	2023-01-04	West
5	102	6	2023-01-05	South
6	101	4	2023-01-06	East

Результат запроса:

ProductID	Region	TotalQuantity
101	North	7
101	East	4
102	South	9
103	West	7

4. Оператор HAVING.

Оператор **HAVING** работает аналогично **WHERE**, но в отличие от него, применяется не к отдельным строкам, а к группам строк, сформированным **после группировки**. HAVING.

Синтаксис:

```
SELECT column1, column2, AGGREGATE_FUNCTION(column3)
FROM table_name
WHERE condition
GROUP BY column1, column2
HAVING aggregate_condition;
```

AGGREGATE_FUNCTION(column3): Агрегирующая функция (например, SUM, COUNT), которая применяется к сгруппированным данным.

HAVING aggregate_condition: Условие, накладываемое на результат агрегирующей функции.

Пример 1: Фильтрация групп по агрегированным данным

Предположим, у нас есть таблица Orders, содержащая информацию о заказах:

OrderID	CustomerID	Amount	OrderDate
1	101	150	2023-01-01
2	102	200	2023-01-02
3	101	250	2023-01-03
4	103	300	2023-01-04
5	102	400	2023-01-05

Задача: найти клиентов, у которых общая сумма заказов превышает 300.

Запрос:

```
SELECT CustomerID, SUM(Amount) AS TotalAmount
FROM Orders
GROUP BY CustomerID
HAVING SUM(Amount) > 300;
```

Результат:

CustomerID	TotalAmount
101	400
102	600

Этот запрос группирует заказы по CustomerID, вычисляет общую сумму заказов (SUM(Amount)) для каждого клиента, а затем фильтрует только те группы, где общая сумма заказов превышает 300.

5. Псевдонимы (Alias).

Псевдонимы (Alias) в SQL используются для временного переименования столбцов или таблиц в запросах.

Это позволяет упростить работу с длинными именами, улучшить читаемость запросов или избежать конфликтов имен при объединении таблиц.

1. Псевдонимы для столбцов:

Псевдонимы для столбцов используются для переименования столбцов в результирующем наборе данных. Это может быть полезно, когда вы хотите, чтобы столбец имел более понятное или краткое имя.

Синтаксис:

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

column_name: имя столбца в таблице.

AS – ключевое слово.

alias_name: имя, которое вы хотите присвоить этому столбцу в выводе.

Пример 1: Псевдоним для столбца

```
SELECT FirstName AS Name, LastName AS Surname
FROM Employees;
```

В результате выполнения этого запроса, столбец FirstName будет отображаться как Name, а LastName — как Surname.

Результат:

Name	Surname
John	Doe
Alice	Smith

2. Псевдонимы для таблиц:

Псевдонимы для таблиц используются для упрощения работы с длинными именами таблиц или для сокращения запросов при объединении (JOIN) нескольких таблиц.

Синтаксис:

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

table_name: имя таблицы.

alias_name: псевдоним для таблицы.

Пример 2: Псевдоним для таблицы

```sql
SELECT e.FirstName, e.LastName, d.DepartmentName
FROM Employees AS e
JOIN Departments AS d ON e.DepartmentID = d.DepartmentID;
```

В этом примере Employees обозначена как **e**, а Departments как **d**. Это позволяет использовать сокращенные имена в запросе.

3. Использование псевдонимов без ключевого слова AS:

В SQL допускается использование псевдонимов без ключевого слова AS. Это может сделать код чуть короче, но иногда ухудшает его читаемость.

Пример 3: Псевдонимы без AS

```
SELECT FirstName Name, LastName Surname
FROM Employees e;
```

Этот запрос даст тот же результат, что и предыдущие примеры с AS.

4. Псевдонимы и вычисляемые поля:

Псевдонимы часто используются для названия вычисляемых полей.

Пример 4: Псевдонимы для вычисляемых столбцов

```
SELECT FirstName, LastName, (Salary * 12) AS AnnualSalary
FROM Employees;
```

Здесь AnnualSalary — это псевдоним для столбца, который рассчитывает годовую зарплату на основе столбца Salary.

Результат:

FirstName	LastName	AnnualSalary
John	Doe	72000
Alice	Smith	84000

Ключевые моменты:

- Псевдонимы улучшают читаемость SQL-запросов, особенно в случаях длинных имен столбцов или таблиц.
- AS не является обязательным, но его использование повышает ясность кода.
- Псевдонимы полезны при объединении таблиц, создании вычисляемых полей и форматировании вывода данных.

Контрольные вопросы:

- Что такое агрегирующие функции в SQL и для чего они используются?
- Перечислите основные агрегирующие функции и расскажите о каждой из них.
- Как использовать оператор GROUP BY для группировки данных?
- Чем отличается оператор WHERE от оператора HAVING?
- Как используются вложенные запросы в SQL? Приведите пример.
- В чем преимущество использования псевдонимов (alias) в SQL?
- Можно ли использовать псевдонимы без ключевого слова AS? Приведите пример.
- Объясните, как составить запрос с вложенным запросом в секции FROM.
- Какой синтаксис применяется для фильтрации групп с помощью HAVING?
- Почему важно использовать псевдонимы при объединении таблиц?

Практика:

https://sql-academy.org/ru/guide