Тема 14. Выборка из одной таблицы. Фильтрация. Сортировка. Агрегирующие функции.

Цель занятия:

Ознакомиться с концепциями работы с несколькими таблицами в базах данных, а также операциями объединения таблиц.

Учебные вопросы:

1. DML-запросы. SELECT-запросы, выборки из одной таблицы.

2. Агрегирующие функции

3. Вложенные запросы

4. Оператор GROUP BY для группировки данных

5. Оператор HAVING для фильтрации сгруппированных данных

6. Alias

1. DML-запросы. SELECT-запросы, выборки из одной таблицы.

DML (Data Manipulation Language) — это подмножество SQL, которое включает команды для работы с данными в базах данных.

В MySQL DML-запросы используются для добавления, обновления, удаления и выборки данных из таблиц.

Далее рассмотрим основные команды DML и примеры их использования.

1. INSERT — Вставка данных в таблицу

Команда INSERT используется для добавления новых строк в таблицу.

Пример:

```
INSERT INTO Employees (FirstName, LastName, Age, Department)
VALUES ('John', 'Doe', 30, 'HR');
```

INSERT INTO Employees: Указывает, что вы добавляете новую запись в таблицу Employees.

(FirstName, LastName, Age, Department): Это список столбцов, в которые будут добавлены значения.

VALUES ('John', 'Doe', 30, 'HR'): Указывает значения, которые будут вставлены в соответствующие столбцы.

2. **UPDATE** — Обновление данных в таблице

Команда UPDATE используется для изменения существующих данных в таблице.

Пример:

```sql
UPDATE Employees
SET Age = 31
WHERE FirstName = 'John' AND LastName = 'Doe';
```

Этот запрос обновляет возраст сотрудника с именем John Doe на 31 год.

3. **DELETE** — Удаление данных из таблицы

Команда DELETE используется для удаления строк из таблицы.

Пример:

```
DELETE FROM Employees
WHERE LastName = 'Doe';
```

Этот запрос удаляет всех сотрудников с фамилией Doe из таблицы Employees.

4. SELECT — Выборка данных из таблицы

Команда SELECT используется для извлечения данных из таблицы в MySQL. Она позволяет выбирать все или определённые столбцы, фильтровать записи, сортировать и агрегировать данные.

Пример. Простая выборка всех данных Выбирает все записи из таблицы.:

```
SELECT * FROM Students;
```

Выборка определённых столбцов. Извлекает только указанные поля.

Пример:

```
SELECT id, name, email FROM Students;
```

Этот запрос возвращает только id, name и email из Students.

Общий вид структуры SELECT-запроса:

```
SELECT [DISTINCT] столбцы
FROM таблица
[JOIN другая_таблица ON условие_связи]
[WHERE условие_фильтрации]
[GROUP BY столбец_или_столбцы]
[HAVING условие_для_группы]
[ORDER BY столбец_или_столбцы [ASC | DESC]]
[LIMIT количество_строк [OFFSET смещение]];
```

Компоненты запроса:

SELECT: Обязательная часть запроса, указывающая, какие столбцы или выражения нужно выбрать.

Опционально можно использовать ключевое слово DISTINCT, чтобы удалить дубликаты из результата.

FROM: Обязательная часть, указывающая таблицу (или таблицы), из которых нужно выбирать данные.

Компоненты запроса:

JOIN: Опциональная часть, позволяющая объединять данные из нескольких таблиц на основе заданного условия.

Виды JOIN: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN.

WHERE: Опциональная часть, задающая условия фильтрации строк, которые должны быть включены в результат.

GROUP BY: Опциональная часть, группирующая строки, имеющие одинаковые значения в указанных столбцах.

Обычно используется с агрегатными функциями (например, COUNT, SUM, AVG).

HAVING: Опциональная часть, задающая условие фильтрации для групп, образованных с помощью GROUP BY.

ORDER BY: Опциональная часть, определяющая порядок сортировки результирующих строк.

Можно указать сортировку по возрастанию (ASC, по умолчанию) или по убыванию (DESC).

LIMIT: Опциональная часть, ограничивающая количество возвращаемых строк.

Параметр OFFSET позволяет пропустить указанное количество строк перед выборкой.

Пример полного SELECT-запроса:

```sql
SELECT DISTINCT FirstName, LastName, COUNT(*) AS NumOrders
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID
WHERE Orders.OrderDate >= '2024-01-01'
GROUP BY FirstName, LastName
HAVING COUNT(*) > 5
ORDER BY NumOrders DESC
LIMIT 10 OFFSET 5;
```

Этот запрос выбирает уникальные имена и фамилии клиентов, которые сделали более 5 заказов с 1 января 2024 года, сортирует результаты по количеству заказов в убывающем порядке и возвращает 10 строк, начиная с 6-й строки.

Пример с фильтрацией:

```
SELECT * FROM Students WHERE age > 18;
```

WHERE — условие, ограничивающее выборку.

Арифметические операции в SELECT-запросах.

в SELECT-запросах можно использовать различные арифметические операторы для данных, хранящихся в таблицах:

- + – сложение;
- - – вычитание;
- / – деление;
- * – умножение;
- % – взятие остатка от деления.

Пример:

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

Этот запрос выбирает имя и зарплату сотрудника, а также вычисляет годовую зарплату, умножая месячную зарплату на 12.

```
SELECT FirstName, Salary, Salary + 1000 AS IncreasedSalary
FROM Employees;
```

Этот запрос выбирает имя и зарплату сотрудника, а также вычисляет зарплату с увеличением на 1000 единиц.

Оператор WHERE и фильтрация по условиям.

Оператор WHERE в SQL используется для фильтрации строк в результате запроса, чтобы вернуть только те записи, которые соответствуют заданным условиям.

Он позволяет ограничить выборку данных на основе различных критериев.

В качестве условий можно использовать:

- сравнения (=, >, <, >=, <=, !=);
- оператор IN;
- оператор BETWEEN;
- оператор LIKE.

С условиями можно применять логические операторы and, or и not:

- Оператор AND отображает запись, если оба операнда истинны;
- Оператор OR отображает запись, если хотя бы один операнд истинен;
- Оператор NOT инвертирует исходный операнд.

Примеры:

```
SELECT * FROM Employees
WHERE Department = 'HR';
```

Этот запрос выбирает все строки из таблицы Employees, где столбец Department равен 'HR’.

```
SELECT * FROM Employees
WHERE Age > 30 AND Department = 'HR';
```

Этот запрос выбирает все строки, где возраст больше 30 и отдел равен 'HR'.

```sql
SELECT title, release_year
FROM film
WHERE release_year >= 2000;
```

Этот запрос выбирает данные из таблицы film, отображая названия и года выпуска фильмов, которые были выпущены в 2000 году или позже.

```sql
SELECT first_name, last_name, active
FROM staff
WHERE NOT active = true;
```

Этот запрос выбирает данные из таблицы staff, отображая имена, фамилии и статус активности сотрудников, которые не активны.

Оператор **IN** в SQL используется для проверки, находится ли значение в заданном списке значений.

Он позволяет упростить запросы, которые требуют проверки на соответствие нескольким возможным значениям.

Синтаксис оператора IN

```
SELECT столбцы
FROM таблица
WHERE столбец IN (значение1, значение2, ..., значениеN);
```

Пример:

```sql
SELECT first_name, last_name, department
FROM staff
WHERE department IN ('HR', 'IT', 'Finance');
```

Этот запрос выбирает имена, фамилии и отделы сотрудников, которые работают в отделах HR, IT или Finance

Использование с подзапросом :

```sql
SELECT title, release_year
FROM film
WHERE film_id IN (
    SELECT film_id
    FROM rentals
    WHERE rental_date >= '2024-01-01'
);
```

Этот запрос выбирает названия и года выпуска фильмов, которые были арендованы начиная с 1 января 2024 года. Подзапрос возвращает идентификаторы фильмов, соответствующих условию.

Использование с отрицанием:

```sql
SELECT first_name, last_name
FROM staff
WHERE department NOT IN ('HR', 'IT');
```

Этот запрос выбирает имена и фамилии сотрудников, которые не работают в отделах HR или IT.

Оператор **BETWEEN** в SQL используется для фильтрации данных по диапазону значений.

Он позволяет выбрать строки, значения в которых находятся в заданном диапазоне, включая граничные значения.

Этот оператор может применяться как к числовым, так и к дата-временным данным.

Синтаксис оператора BETWEEN

```
SELECT столбцы
FROM таблица
WHERE столбец BETWEEN значение1 AND значение2;
```

Фильтрация по числовому диапазону:

```sql
SELECT first_name, salary
FROM employees
WHERE salary BETWEEN 30000 AND 50000;
```

Этот запрос выбирает имена и зарплаты сотрудников, чьи зарплаты находятся в диапазоне от 30,000 до 50,000 включительно.

Фильтрация по дате:

```sql
SELECT order_id, order_date
FROM orders
WHERE order_date BETWEEN '2024-01-01' AND '2024-12-31';
```

Этот запрос выбирает идентификаторы заказов и даты заказов, которые были сделаны в течение 2024 года, включая оба крайних дня.

Фильтрация с отрицанием:

```sql
SELECT first_name, last_name
FROM employees
WHERE salary NOT BETWEEN 30000 AND 50000;
```

Этот запрос выбирает имена и фамилии сотрудников, чьи зарплаты не находятся в диапазоне от 30,000 до 50,000.

Оператор LIKE в SQL используется для поиска строк, которые соответствуют определенному шаблону.

Это полезно для фильтрации данных, когда нужно найти строки, содержащие определенные подстроки или соответствующие определенным условиям.

Синтаксис оператора LIKE:

```
SELECT столбцы
FROM таблица
WHERE столбец LIKE шаблон;
```

Специальные символы шаблона в операторе LIKE:

%: Заменяет ноль или более символов.

_: Заменяет ровно один символ.

Поиск по начальной части строки:

```sql
SELECT first_name, last_name
FROM employees
WHERE first_name LIKE 'A%';
```

Этот запрос выбирает имена и фамилии сотрудников, чьи имена начинаются с буквы 'A'. Символ % заменяет любые символы после 'A'.

Поиск по содержимому строки:

```sql
SELECT email
FROM users
WHERE email LIKE '%@example.com';
```

Этот запрос выбирает электронные адреса пользователей, которые заканчиваются на '@example.com’.

Поиск по части строки с фиксированной длиной:

```sql
SELECT product_name
FROM products
WHERE product_code LIKE 'AB_123';
```

Этот запрос выбирает названия продуктов, где код продукта начинается с 'AB', за которым следует любой один символ (обозначенный _), затем '123'.

Использование NOT LIKE:

```sql
SELECT first_name, last_name
FROM employees
WHERE first_name NOT LIKE 'A%';
```

Этот запрос выбирает имена и фамилии сотрудников, чьи имена не начинаются с буквы 'A'.

Поиск по строкам, содержащим определенный набор символов:

```
SELECT address
FROM contacts
WHERE address LIKE '%Main St%';
```

Этот запрос выбирает данные из таблицы film, отображая названия и описания фильмов, в которых в поле description содержится слово "Scientist".

Сортировка данных при помощи ORDER BY.

Оператор ORDER BY в SQL используется для сортировки результатов запроса по одному или нескольким столбцам.

Он позволяет упорядочить строки в результирующем наборе данных в порядке возрастания или убывания.

Синтаксис оператора ORDER BY:

```
SELECT столбцы
FROM таблица
ORDER BY столбец1 [ASC|DESC], столбец2 [ASC|DESC], ...;
```

столбцы: Список столбцов, которые должны быть выбраны.

таблица: Имя таблицы, из которой будут извлечены данные.

столбец1, столбец2, ...: Столбцы, по которым будет производиться сортировка.

ASC: Сортировка по возрастанию (по умолчанию).

DESC: Сортировка по убыванию.

Сортировка по одному столбцу:

```sql
SELECT first_name, last_name
FROM employees
ORDER BY last_name ASC;
```

Этот запрос выбирает имена и фамилии сотрудников и сортирует их по фамилии в порядке возрастания (по умолчанию ASC).

Сортировка по нескольким столбцам:

```sql
SELECT first_name, last_name, hire_date
FROM employees
ORDER BY hire_date DESC, last_name ASC;
```

Этот запрос выбирает имена, фамилии и даты найма сотрудников. Сначала сортирует по дате найма в порядке убывания, а затем по фамилии в порядке возрастания для сотрудников, нанятых в один и тот же день.

Сортировка числовых значений:

```sql
SELECT product_name, price
FROM products
ORDER BY price DESC;
```

Этот запрос выбирает названия продуктов и их цены, сортируя их по цене в порядке убывания.

Оператор LIMIT.

Оператор LIMIT в SQL используется для ограничения числа строк, возвращаемых запросом. Это полезно для оптимизации запросов и управления объемом данных, возвращаемых клиенту. Особенно актуально это при работе с большими таблицами или при реализации постраничного отображения данных.

Синтаксис оператора LIMIT:

```
SELECT столбцы
FROM таблица
ORDER BY столбец
LIMIT количество_строк;
```

Возвращение первых N строк:

```sql
SELECT first_name, last_name
FROM employees
ORDER BY hire_date DESC
LIMIT 5;
```

Этот запрос выбирает имена и фамилии пяти самых последних нанятых сотрудников, упорядоченных по дате найма в порядке убывания.

Возвращение строк с определенного номера:

```sql
SELECT first_name, last_name
FROM employees
ORDER BY hire_date DESC
LIMIT 10 OFFSET 5;
```

Этот запрос выбирает 10 сотрудников, начиная с 6-го (пропустив первые 5), упорядоченных по дате найма в порядке убывания.

2. Агрегирующие функции

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

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';
```

3. 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.

4. 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

5. 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.

6. 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.

Домашнее задание:

Задание 1: Агрегирующие функции

Создайте таблицу sales с полями:

- id (INT, PRIMARY KEY, AUTO_INCREMENT)
- product_name (VARCHAR)
- quantity (INT)
- price (DECIMAL)
- sale_date (DATE)

Заполните таблицу данными (минимум 10 записей).

Напишите запросы:

- Найдите общее количество проданных товаров.
- Найдите среднюю цену товара.
- Найдите максимальную и минимальную цену товара.
- Найдите общую сумму продаж за все время.

2. Дана таблица Students со следующей структурой:

- StudentID (INT) — уникальный идентификатор студента
- Name (VARCHAR) — имя студента
- Age (INT) — возраст студента
- City (VARCHAR) — город проживания
- Score (DECIMAL) — средний балл

Напишите запрос, который выведет все данные из таблицы Students.
Выведите только имена и возраст студентов.
Найдите всех студентов старше 20 лет.
Найдите студентов, проживающих в городе "Moscow".
Выведите студентов, чей средний балл больше или равен 4.5.
Отсортируйте студентов по возрастанию их среднего балла.
Выведите первые 3 записи из таблицы.
Найдите студентов, чьи имена начинаются на букву "A".
Выведите студентов, чей возраст находится в диапазоне от 18 до 25 лет.

3. Дана таблица "Employees" со следующей структурой:

- EmployeeID (INT)
- FirstName (VARCHAR)
- LastName (VARCHAR)
- Age (INT)
- Department (VARCHAR)
- Salary (DECIMAL)
- HireDate (DATE)

Необходимо написать следующие SELECT-запросы:

- Вывести полный список всех сотрудников, отсортированный по фамилии.
- Найти всех сотрудников старше 30 лет из отдела "HR".
- Вывести информацию о сотрудниках с зарплатой в диапазоне от 30000 до 50000.
- Найти всех сотрудников, чьи фамилии начинаются на букву "A".
- Вывести топ-5 самых высокооплачиваемых сотрудников.
- Найти среднюю зарплату по каждому отделу.
- Вывести список сотрудников, нанятых в 2023 году.
- Найти всех сотрудников с именем "John" или "Jane".
- Вывести список сотрудников, чьи фамилии содержат подстроку "ов".

Задание 4: Группировка данных

Используя таблицу sales, напишите запросы:

- Сгруппируйте данные по товарам (product_name) и найдите общее количество проданных единиц для каждого товара.
- Сгруппируйте данные по дате (sale_date) и найдите общую сумму продаж за каждый день.
- Найдите среднее количество проданных товаров в день.

Список литературы:

1. Руководство по MySQL.

Материалы лекций:

https://github.com/ShViktor72/Education2025