SQL INTERVIEW QUESTIONS

Подборка вопросов по SQL для подготовки к техническим собеседованиям

40

вопросов

Вопрос 1

Что такое SQL и для чего он используется?

> *SQL (Structured Query Language) — это стандартный язык для взаимодействия с реляционными базами данных. Он используется для определения, управления и извлечения данных из баз данных. С помощью SQL можно выполнять следующие операции:*

- Создание новых баз данных и таблиц
- Вставка новых данных в таблицы
- Чтение данных с использованием запросов
- Обновление существующих данных
- Удаление данных
- Управление доступом и разрешениями

Вопрос 2

Объясните различия между DDL, DML и DCL в SQL?

SQL-команды разделяются на три основные категории:

DDL (Data Definition Language) — язык определения данных:

- Используется для определения структуры базы данных (схемы).
- Основные команды: `CREATE`, `ALTER`, `DROP`, `TRUNCATE`, `RENAME`.

DML (Data Manipulation Language) — язык манипулирования данными:

- Используется для работы с данными внутри таблиц.
- Основные команды: `SELECT`, `INSERT`, `UPDATE`, `DELETE`.

DCL (Data Control Language) — язык управления доступом:

- Используется для управления правами доступа к базе данных.
- Основные команды: `GRANT`, `REVOKE`.

Вопрос 3

Что такое первичный ключ и внешний ключ?

Первичный ключ (PRIMARY KEY):

- Уникальный идентификатор записи в таблице.
- Не допускает дубликатов и `NULL` значений.
- Может состоять из одного или нескольких столбцов (составной ключ).

```
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(100),
    age INT
);
```

Внешний ключ (FOREIGN KEY):

- Столбец или набор столбцов, которые ссылаются на первичный ключ другой таблицы.
- Обеспечивает ссылочную целостность между таблицами.
- Позволяет связать записи из разных таблиц.

```
CREATE TABLE enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Вопрос 4

Объясните понятие нормализации и ее преимущества

> *Нормализация — это процесс организации структуры базы данных с целью уменьшения избыточности данных и обеспечения их целостности.*

Основные цели нормализации:

- Устранение избыточности: предотвращает дублирование данных.
- Повышение целостности данных: минимизирует вероятность неконсистентности данных.
- Облегчение поддержки и обновления: делает структуру базы данных более гибкой и понятной.

Основные нормальные формы

- Первая нормальная форма (1НФ): все столбцы содержат атомарные (неделимые) значения.
- Вторая нормальная форма (2НФ): удовлетворяет 1НФ и все неключевые столбцы зависят от всего первичного ключа.
- Третья нормальная форма (3НФ): удовлетворяет 2НФ и нет транзитивных зависимостей между неключевыми столбцами.

Пример

До нормализации:

StudentID	StudentName	CourseID	CourseName
1	John	101	Math
1	John	102	Music

После нормализации:

Таблица Students

StudentID	StudentName
1	John

Таблица Courses

CourseID	CourseName
101	Math
102	Music

Таблица Enrollments

StudentID	CourseID
1	101
1	102

Вопрос 5

Что такое JOIN и какие виды JOIN вы знаете?

> *JOIN — это операция в SQL, которая позволяет объединить строки из двух или более таблиц на основе связанных между ними столбцов.*

Виды JOIN:

- `INNER JOIN`:

Возвращает записи, у которых есть соответствующие записи в обеих таблицах.

- `LEFT JOIN (или LEFT OUTER JOIN)`:

Возвращает все записи из левой таблицы и соответствующие записи из правой таблицы. Если соответствия нет, возвращает `NULL` для правой таблицы.

- `RIGHT JOIN (или RIGHT OUTER JOIN)`:

Возвращает все записи из правой таблицы и соответствующие записи из левой таблицы. Если соответствия нет, возвращает `NULL` для левой таблицы.

- `FULL OUTER JOIN`:

Возвращает все записи, когда есть соответствие в одной из таблиц.

- `CROSS JOIN`:

Выполняет декартово произведение двух таблиц, объединяя каждую строку первой таблицы с каждой строкой второй таблицы.

Вопрос 6

Что такое подзапрос (subquery) и когда он используется?

> *Подзапрос — это SQL-запрос, вложенный внутри другого запроса. Он используется для выполнения операций, результат которых необходим для основного запроса.*

Случаи использования подзапросов:

- **Фильтрация данных**: Использование результатов подзапроса в условиях `WHERE` или `HAVING`.
- **Выборка данных**: Использование подзапроса в списке выбранных столбцов.
- **Создание виртуальных таблиц**: Использование подзапроса в операторе `FROM`.

Примеры:

1. Подзапрос в `WHERE`:

```
SELECT name
FROM employees
WHERE department_id = (SELECT id FROM departments WHERE name = 'IT');
```

2. Подзапрос в `FROM`:

```
SELECT sub.department, COUNT(*)
FROM (
    SELECT department_id AS department
    FROM employees
) sub
GROUP BY sub.department;
```

Вопрос 7

Как удалить дубликаты в результате SQL-запроса?

Использовать ключевое слово `DISTINCT` в операторе `SELECT`, чтобы вернуть только уникальные записи.

Пример:

```
SELECT DISTINCT position
FROM employees;
```

Вопрос 8

Объясните разницу между WHERE и HAVING

WHERE

- Фильтрует строки **до** группировки данных.
- Не может использовать агрегатные функции (`SUM()` , `COUNT()` , `AVG()` , и т.д.).
- Применяется к отдельным записям таблицы.

Пример:

```
SELECT department_id, COUNT()
FROM employees
WHERE salary > 50000
GROUP BY department_id;
```

HAVING

- *Фильтрует группы строк после группировки данных.*
- *Может использовать агрегатные функции.*
- *Применяется к результатам* `GROUP BY` .

Пример:

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

Вопрос 9

Что такое индекс и как он влияет на производительность?

> ***Индекс*** *— это специальная структура данных, которая улучшает скорость операций выборки данных из таблицы за счет создания указателей на данные.*

Индексы ускоряют операции чтения, но могут замедлять операции записи (вставка, обновление, удаление), так как индексы нужно обновлять при изменении данных.

Преимущества индексов:

- Быстрое выполнение операций `SELECT`.
- Улучшение производительности при сортировке и поиске данных.

Недостатки индексов:

- Дополнительное пространство на диске.
- Замедление операций `INSERT`, `UPDATE`, `DELETE`.

Пример создания индекса:

```
-- Создание индекса на столбце name таблицы employees
CREATE INDEX idx_employees_name ON employees(name);
```

Вопрос 10

Каковы основные различия между DELETE и TRUNCATE?

DELETE:

- Удаляет выбранные записи из таблицы.
- Можно использовать условие `WHERE` для удаления конкретных записей.
- Операция записывается в журнал транзакций построчно.
- Триггеры `ON DELETE` срабатывают.
- Медленнее по сравнению с `TRUNCATE`.

Пример:

```
DELETE FROM employees WHERE salary < 30000;
```

TRUNCATE:

- Удаляет все записи из таблицы без возможности восстановления через `ROLLBACK` (в большинстве СУБД).
- Нельзя использовать `WHERE`.
- Операция быстрее, так как не логируется построчно.
- Сбрасывает идентификаторы (если используется автоинкремент).
- Триггеры не срабатывают.

Пример:

```
TRUNCATE TABLE employees;
```

Вопрос 11

Что такое транзакция и какие свойства транзакции (ACID)?

> ***Транзакция*** *— это последовательность операций, выполняемых как единое логическое действие, которое должно быть полностью выполнено или полностью отменено.*

Свойства транзакций (ACID):

Атомарность (Atomicity):

- Транзакция выполняется полностью или не выполняется вовсе.
- Если происходит сбой, все изменения отменяются.

Согласованность (Consistency):

- Транзакция переводит базу данных из одного согласованного состояния в другое.
- Все правила и ограничения базы данных соблюдаются.

Изоляция (Isolation):

- Результаты транзакции невидимы для других транзакций до ее завершения.
- Предотвращает взаимное влияние параллельных транзакций.

Долговечность (Durability):

- После успешного завершения транзакции ее результаты сохраняются даже при сбоях системы.
- Изменения записываются на постоянное хранилище.

Вопрос 12

Что такое триггеры в SQL?

> ***Триггер*** *— это хранимая процедура, которая автоматически выполняется при наступлении определенного события в базе данных, такого как* `INSERT`, `UPDATE` *или* `DELETE` *на определенной таблице.*

Типы триггеров:

- DML триггеры: реагируют на операции `INSERT`, `UPDATE`, `DELETE`.
- DDL триггеры: реагируют на операции `CREATE`, `ALTER`, `DROP`.
- Триггеры уровня строки или оператора.

Преимущества триггеров:

- Автоматизация проверок и ограничений.
- Логирование изменений.
- Поддержание целостности данных.

Пример создания триггера:

```
CREATE TRIGGER trg_after_insert_employee
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
    INSERT INTO audit_log (employee_id, action, action_time)
    VALUES (NEW.id, 'INSERT', NOW());
END;
```

Вопрос 13

Объясните, что такое VIEW и его преимущества

> ***VIEW (представление)*** *— это виртуальная таблица, основанная на результате SQL-запроса. Представление не хранит данные самостоятельно, а предоставляет определенный способ просмотра данных из одной или нескольких таблиц.*

Преимущества VIEW:

- **Упрощение сложных запросов**: позволяет сохранить сложный запрос и использовать его как простую таблицу.
- **Безопасность**: предоставляет пользователям доступ только к определенным данным, скрывая остальные.
- **Обновляемость**: в некоторых случаях можно обновлять данные через представление.
- **Поддержание целостности данных**: может содержать данные из нескольких таблиц, объединенные определенным образом.

Пример создания VIEW:

```
CREATE VIEW employee_details AS
SELECT e.id, e.name, d.name AS department, e.salary
FROM employees e
JOIN departments d ON e.department_id = d.id;
```

Использование VIEW:

```
SELECT * FROM employee_details WHERE salary > 50000;
```

Вопрос 14

Как использовать оператор LIKE и для чего он применяется?

Оператор `LIKE` используется в условиях `WHERE` для поиска строк, соответствующих определенному шаблону. В шаблонах используются подстановочные символы:

- `%` — соответствует любой последовательности символов (включая нулевую длину).
- `_` — соответствует одному любому символу.

Примеры использования:

1. Поиск строк, начинающихся с «A»:

```
SELECT  FROM employees WHERE name LIKE 'A%';
```

2. Поиск строк, оканчивающихся на «e»:

```
SELECT  FROM employees WHERE name LIKE '%e';
```

3. Поиск строк, где второй символ — «a»:

```
SELECT * FROM employees WHERE name LIKE '_a%';
```

Вопрос 15

Что такое агрегатные функции? Приведите примеры

Агрегатные функции выполняют вычисления над набором значений и возвращают одно значение. Они часто используются в сочетании с оператором `GROUP BY`.

Основные агрегатные функции

- `COUNT()` — подсчитывает количество строк.
- `SUM()` — вычисляет сумму значений.
- `AVG()` — вычисляет среднее значение.
- `MAX()` — находит максимальное значение.
- `MIN()` — находит минимальное значение.

Примеры использования

1. Подсчет количества сотрудников:

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

2. Средняя зарплата по отделам:

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

3. Максимальная зарплата в компании:

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

4. Суммарные продажи за месяц:

```
SELECT SUM(amount) FROM sales WHERE sale_date BETWEEN '2023-01-01' AND '2023-01-31';
```

Вопрос 16

Объясните разницу между UNION и UNION ALL

UNION:

- Объединяет результаты двух или более `SELECT`-запросов.
- Удаляет дубликаты из объединенного результата.

Синтаксис

```
SELECT column_list FROM table1
UNION
SELECT column_list FROM table2;
```

UNION ALL:

- Объединяет результаты двух или более `SELECT`-запросов.
- Сохраняет дубликаты в объединенном результате.
- Быстрее, так как не выполняет дополнительную операцию по удалению дубликатов.

Синтаксис

```
SELECT column_list FROM table1
UNION ALL
SELECT column_list FROM table2;
```

Вопрос 17

Что такое хранимая процедура и как она отличается от функции?

Хранимая процедура (Stored Procedure):

- Набор SQL-команд, сохраненных на сервере для повторного использования.
- Может выполнять операции `SELECT`, `INSERT`, `UPDATE`, `DELETE`.
- Может возвращать несколько наборов результатов или ничего не возвращать.
- Может иметь входные и выходные параметры.
- Не может быть вызвана внутри SQL-запроса.

Пример хранимой процедуры:

```
CREATE PROCEDURE GetEmployeeByID(IN emp_id INT)
BEGIN
    SELECT * FROM employees WHERE id = emp_id;
END;
```

Вызов процедуры:

```
CALL GetEmployeeByID(1);
```

Функция (Function):

- Возвращает одно значение (скалярная функция) или таблицу (табличная функция).
- Может использоваться в SQL-выражениях (например, в SELECT или WHERE).
- Должна возвращать значение.
- Обычно используется для вычислений и возвращает детерминированный результат.

Пример функции:

```
CREATE FUNCTION GetEmployeeSalary(emp_id INT) RETURNS DECIMAL(10,2)
BEGIN
    DECLARE salary DECIMAL(10,2);
    SELECT e.salary INTO salary FROM employees e WHERE e.id = emp_id;
    RETURN salary;
END;
```

Использование функции:

```
SELECT name, GetEmployeeSalary(id) FROM employees;
```

Вопрос 18

Как оптимизировать производительность SQL-запросов?

Использовать индексы:

- Создавайте индексы на столбцах, часто используемых в условиях `WHERE`, `JOIN` и `ORDER BY`.
- Избегайте избыточных индексов.

**Избегать `SELECT`*:

- **Выбирайте только необходимые столбцы.**
- **Уменьшает объем передаваемых данных.**

Оптимизировать условия `JOIN` и `WHERE`:

- **Используйте равенство (=) вместо неравенства, где возможно.**
- **Избегайте функций и вычислений над индексируемыми столбцами в условиях.**

Использовать ограничения результатов (`LIMIT`):

- **Ограничивайте количество возвращаемых строк, если не нужны все данные.**

Избегать подзапросов, где возможны соединения:

- **Заменяйте коррелированные подзапросы на `JOIN` или `EXISTS`.**

Кеширование часто используемых данных:

- **Используйте материализованные представления или кеширование на уровне приложения.**

Профилирование и анализ запросов:**

- Используйте инструменты (`EXPLAIN`, `EXPLAIN PLAN`) для анализа плана выполнения запросов.

Вопрос 19

Что такое ограничения (constraints) и какие виды существуют?

Ограничения обеспечивают целостность и надежность данных в таблице, определяя правила для данных в столбцах.

Виды ограничений:

NOT NULL:

- Запрещает хранение `NULL` значений в столбце.

Пример:

```
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);
```

UNIQUE:

- Обеспечивает уникальность значений в столбце или группе столбцов.

Пример:

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);
```

PRIMARY KEY:

- Комбинация `NOT NULL` и `UNIQUE`.
- Идентифицирует каждую запись в таблице.

FOREIGN KEY:

- Обеспечивает ссылочную целостность между таблицами.
- Значение должно соответствовать существующему значению первичного ключа в связанной таблице.

Пример:

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    user_id INT,
    FOREIGN KEY (user_id) REFERENCES users(user_id)
);
```

CHECK:

- Определяет условие, которому должны соответствовать значения в столбце.

Пример:

```
CREATE TABLE employees (
    id INT PRIMARY KEY,
    age INT CHECK (age >= 18)
);
```

DEFAULT:

- Устанавливает значение по умолчанию для столбца, если значение не указано при вставке.

```
CREATE TABLE tasks (
    task_id INT PRIMARY KEY,
    status VARCHAR(20) DEFAULT 'Pending'
);
```

Вопрос 20

Что такое SQL-инъекция и как от нее защититься?

> ***SQL-инъекция*** *— это метод атаки на базу данных, при котором злоумышленник вставляет вредоносный SQL-код через вводимые данные, позволяя выполнять несанкционированные SQL-запросы.*

Последствия SQL-инъекции:

- Кража данных.
- Удаление или изменение данных.
- Получение административного доступа.

Способы защиты от SQL-инъекций:

1. Параметризированные запросы (Prepared Statements):

- Использование параметров вместо конкатенации строк.
- СУБД автоматически экранирует специальные символы.

Пример (на языке Java с использованием JDBC)

```
String sql = "SELECT * FROM users WHERE username = ? AND password = ?";
PreparedStatement stmt = connection.prepareStatement(sql);
stmt.setString(1, username);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
```

2. Использование ORM (Object-Relational Mapping):

- Библиотеки ORM часто включают механизмы защиты от SQL-инъекций.

3. Проверка и фильтрация вводимых данных:

- Проверять данные на соответствие ожидаемому формату.
- Использовать валидацию на стороне сервера и клиента.

4. Ограничение прав доступа:

- Предоставлять минимально необходимые права пользователям базы данных.
- Ограничить доступ к системным таблицам и операциям.

5. Использование хранимых процедур:

- Логика работы с данными инкапсулирована в процедуре.

- Пользователи имеют доступ только к процедурам, а не к таблицам напрямую.

Вопрос 21

Что такое реляционная база данных?

> ***Реляционная база данных*** *— это база данных, основанная на реляционной модели данных. В такой базе данных данные хранятся в таблицах, и отношения между данными определяются с помощью ключей.*

Основные характеристики:

- Таблицы (отношения): данные организованы в таблицы, состоящие из строк и столбцов.
- Строки (записи): каждая строка представляет отдельную запись.
- Столбцы (атрибуты): каждый столбец содержит данные определенного типа.
- Ключи: используются для идентификации записей и установления связей между таблицами.

Преимущества реляционных баз данных:

- Гибкость: легко добавлять новые таблицы и столбцы.
- Целостность данных: использование ограничений для поддержания целостности.
- SQL: стандартный язык для управления данными.

Вопрос 22

Объясните разницу между INNER JOIN и OUTER JOIN

INNER JOIN:

- Возвращает только те записи, у которых есть соответствующие записи в обеих объединяемых таблицах.
- Если соответствия нет, запись не включается в результат.

Пример:

```
SELECT
FROM employees e
INNER JOIN departments d ON e.department_id = d.id;
```

OUTER JOIN:

- *Возвращает записи, которые имеют соответствия, а также записи из одной таблицы, для которых соответствий нет.*

Виды OUTER JOIN:

LEFT OUTER JOIN (LEFT JOIN):

- *Возвращает все записи из левой таблицы и соответствующие записи из правой таблицы.*
- *Если соответствия нет, столбцы правой таблицы будут* `NULL`.

Пример:

```
SELECT
FROM employees e
LEFT JOIN departments d ON e.department_id = d.id;
```

RIGHT OUTER JOIN (RIGHT JOIN):

- Возвращает все записи из правой таблицы и соответствующие записи из левой таблицы.
- Если соответствия нет, столбцы левой таблицы будут `NULL`.

Пример:

```
SELECT
FROM employees e
RIGHT JOIN departments d ON e.department_id = d.id;
```

FULL OUTER JOIN (FULL JOIN):

- *Возвращает все записи, когда есть соответствие в одной из таблиц.*
- *Если соответствия нет, соответствующие столбцы будут* `NULL`.

```
SELECT
FROM employees e
FULL OUTER JOIN departments d ON e.department_id = d.id;
```

Вопрос 23

Что такое NULL и как с ним работать в SQL?

> ***NULL*** *— это специальное значение в SQL, обозначающее отсутствие данных или неизвестное значение.*

Особенности NULL:

- `NULL` не эквивалентно пустой строке или нулю.
- Операции с `NULL` возвращают `NULL`.
- Сравнение `NULL` = `NULL` возвращает `FALSE`.

Работа с NULL

Для проверки на `NULL` используется оператор `IS NULL` или `IS NOT NULL`.

```
-- Поиск записей с неизвестной датой рождения
SELECT  FROM employees WHERE birth_date IS NULL;

-- Поиск записей с известной датой рождения
SELECT  FROM employees WHERE birth_date IS NOT NULL;
```

Функции для работы с NULL

COALESCE Возвращает первый элемент списка не равный `NULL`

```
COALESCE(val1[, val2, ...., val_n])
```

ISNULL Возвращает 1 или 0 в зависимости равно ли выражение `NULL`

```
ISNULL(value)
```

IFNULL Возвращает значение, переданное 1-ым аргументом, если оно не равно `NULL`. В противном случае, возвращает значение переданное вторым аргументом.

```
    IFNULL(value, alternative_value)
```

Вопрос 24

Как использовать оператор CASE в SQL?

Оператор `CASE` используется для реализации условной логики в SQL-запросах. Он позволяет возвращать значения на основе условий, подобно оператору `IF-ELSE`.

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE default_result
END
```

Примеры

- Присвоение категорий на основе зарплаты:

```
    SELECT name,
        salary,
        CASE
            WHEN salary >= 80000 THEN 'Высокая'
            WHEN salary >= 50000 THEN 'Средняя'
            ELSE 'Низкая'
        END AS salary_category
    FROM employees;
```

Вопрос 25

Объясните транзакционные команды COMMIT и ROLLBACK.

COMMIT

- Фиксирует текущую транзакцию.
- Все изменения, сделанные в транзакции, становятся постоянными и видимыми для других пользователей.
- После COMMIT отменить изменения нельзя.

```
BEGIN TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE account_id = 1;
UPDATE accounts SET balance = balance + 100 WHERE account_id = 2;

COMMIT;
```

ROLLBACK

- Отменяет текущую транзакцию.
- Все изменения, сделанные в транзакции, откатываются.
- База данных возвращается в состояние до начала транзакции.

```
BEGIN TRANSACTION;

DELETE FROM orders WHERE order_date < '2022-01-01';

-- Если передумали
ROLLBACK;
```

Использование в управлении транзакциями:

- `BEGIN TRANSACTION` или `START TRANSACTION`: начало транзакции.
- `COMMIT`: фиксация транзакции
- `ROLLBACK`: откат транзакции.

Вопрос 26

Объясните различия между CHAR и VARCHAR

CHAR(n):

- Хранит строки фиксированной длины `n`.
- Если введенная строка короче `n`, она дополняется пробелами до длины `n`.
- Используется для хранения данных одинаковой длины (например, коды стран, почтовые индексы).

VARCHAR(n):

- Хранит строки переменной длины до `n` символов.
- Фактически занимает столько места, сколько символов в строке плюс небольшой служебный объем для хранения длины.
- Используется для хранения строковых данных переменной длины.

Основные различия:

Память и производительность:

- `CHAR` всегда занимает фиксированное количество памяти.
- `VARCHAR` более эффективен с точки зрения использования памяти, но может быть немного медленнее при доступе.

Применение:

- `CHAR` подходит для данных с предсказуемой длиной.
- `VARCHAR` подходит для данных переменной длины.

Пример:

```
CREATE TABLE products (
    code CHAR(10),     -- Код товара фиксированной длины
    name VARCHAR(100)  -- Название товара переменной длины
);
```

Вставка данных:

```
INSERT INTO products (code, name)
VALUES ('A123', 'Ноутбук Lenovo');
```

В столбце `code` значение будет дополненно пробелами до 10 символов.

Вопрос 27

Что такое временная таблица в SQL?

> ***Временная таблица*** *— это таблица, которая существует только в рамках текущей сессии или соединения и автоматически удаляется после завершения сессии или когда соединение закрывается.*

Создание временной таблицы

```
CREATE TEMPORARY TABLE TempTable (
    id INT,
    name VARCHAR(100)
);
```

Использование временной таблицы

```
-- Вставка данных во временную таблицу
INSERT INTO #TempTable (id, name)
VALUES (1, 'Иван'), (2, 'Петр');

-- Выборка данных из временной таблицы
SELECT * FROM #TempTable;

-- Временная таблица будет автоматически удалена после завершения сессии
```

Применение временной таблицы

- Хранение промежуточных результатов в сложных запросах.
- Обработка больших объемов данных в пакетных операциях.
- Избегание конфликтов при одновременной работе нескольких пользователей.

Вопрос 28

Что такое оконные функции в SQL?

> ***Оконные функции*** *— это функции, которые выполняют вычисления по набору строк (окну), связанных с текущей строкой, и возвращают результат для каждой строки без группировки данных.*

Основные оконные функции:

Агрегатные функции:

- `SUM` — подсчитывает общую сумму значений
- `COUNT` — считает общее количество записей в колонке
- `AVG` — рассчитывает среднее арифметическое
- `MAX` — находит наибольшее значение
- `MIN` — определяет наименьшее значение

Ранжирующие функции:

- `ROW_NUMBER` : присваивает последовательный номер строке в пределах окна
- `RANK` : присваивает ранг строке в пределах окна с пропусками при совпадении значений
- `DENSE_RANK` : присваивает ранг строке без пропусков

Функции смещения:

- `LAG` : возвращает значение из предыдущей строки
- `LEAD` : возвращает значение из следующей строки
- `FIRST_VALUE` : возвращает первое значение в окне
- `LAST_VALUE` : возвращает последнее значение в окне

Подробное объяснение работы оконных функций есть в нашем курс.

Вопрос 29

Объясните понятие CTE (Common Table Expression)

> ***CTE (Common Table Expression)*** *— это временный именованный результат набора, определенный в SQL-запросе с помощью ключевого слова* `WITH`.

Он улучшает читаемость и структуру сложных запросов.

Синтаксис

```
WITH CTEName (column1, column2, ...)
AS (
    -- Ваш запрос
    SELECT ...
)
SELECT * FROM CTEName;
```

Преимущества CTE

- Улучшает читаемость кода.
- Позволяет разбивать сложные запросы на логические части.
- Поддерживает рекурсивные запросы (рекурсивные CTE).

Подробное объяснение работы CTE в нашем курс.

Вопрос 30

Как удалить таблицу вместе с ее данными?

Для этого используется команда `DROP TABLE`, которая удаляет таблицу и все ее данные из базы данных.

Синтаксис

```
DROP TABLE table_name;
```

Особенности:

- Все данные, индексы, триггеры и разрешения, связанные с таблицей, удаляются.
- Действие необратимо, если не настроено резервное копирование или механизмы восстановления.

Вопрос 31

Что такое FOREIGN KEY и как он обеспечивает целостность данных?

> ***FOREIGN KEY*** *(внешний ключ) — это ограничение, которое устанавливает связь между столбцом или набором столбцов в одной таблице и столбцом или столбцами в другой таблице (обычно первичным ключом).*

Он обеспечивает ссылочную целостность, гарантируя, что значения в столбце внешнего ключа соответствуют существующим значениям в связанной таблице.

Как обеспечивается целостность данных:

- **Ограничивается вставка некорректных данных**

Невозможно вставить значение в столбец внешнего ключа, если такого значения нет в связанной таблице.

- **Ограничивается удаление связанных записей**

Невозможно удалить запись из родительской таблицы, если на нее ссылаются записи в дочерней таблице, без дополнительных действий.

Пример внешнего ключа:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

Вопрос 32

Как добавить новый столбец в существующую таблицу?

Для этого используется команда `ALTER TABLE` с оператором `ADD` для добавления нового столбца в таблицу.

Синтаксис

```
ALTER TABLE table_name
ADD column_name data_type [constraints];
```

Пример

Допустим, у нас есть таблица `employees`, и мы хотим добавить столбец `email` типа `VARCHAR(255)`.

```
ALTER TABLE employees
ADD email VARCHAR(255);
```

Добавление столбца с ограничением `NOT NULL` и значением по умолчанию:

```
ALTER TABLE employees
ADD date_of_birth DATE NOT NULL DEFAULT '1900-01-01';
```

Особенность При добавлении столбца с ограничением `NOT NULL`, если в таблице уже есть данные, необходимо указать значение по умолчанию, иначе будет ошибка.

Вопрос 33

Что такое коррелированный подзапрос?

> ***Коррелированный подзапрос*** *— это подзапрос, который зависит от внешнего запроса. Он выполняется для каждой строки внешнего запроса, используя значения из этой строки.*

Особенности

- Подзапрос ссылается на столбцы из внешнего запроса.
- Может быть менее эффективным из-за множественного выполнения.

Пример

Есть таблица `employees` и `departments`.

```
SELECT e.name, e.salary
FROM employees e
WHERE e.salary > (
    SELECT AVG(salary)
    FROM employees
    WHERE department_id = e.department_id
);
```

- Здесь подзапрос вычисляет среднюю зарплату для отдела каждого сотрудника
- Основной запрос выбирает сотрудников, чья зарплата выше средней по отделу

Больше информации о коррелированных подзапросах в нашем курс.

Вопрос 34

Объясните разницу между DELETE, TRUNCATE и DROP

DELETE

- Удаляет выбранные записи из таблицы.
- Можно использовать условие `WHERE` для удаления конкретных записей.
- Операция записывается в журнал транзакций построчно.
- Триггеры `ON DELETE` срабатывают.

Синтаксис

```
DELETE FROM table_name [WHERE condition];
```

TRUNCATE

- Удаляет все данные из таблицы без возможности восстановления.
- Нельзя использовать `WHERE`.
- Быстрее, так как не логирует удаление каждой строки.
- Сбрасывает идентификаторы автоинкремента.
- Триггеры не срабатывают.

Синтаксис

```
TRUNCATE TABLE table_name;
```

DROP

- Удаляет всю таблицу вместе с данными, структурой, индексами, ограничениями.
- Действие необратимо.

Синтаксис

```
DROP TABLE table_name;
```

Выбор между командами

- `DELETE` используется, когда нужно удалить определенные записи.

- `TRUNCATE` используется для быстрого удаления всех данных из таблицы, сохраняя ее структуру.
- `DROP` используется для полного удаления таблицы из базы данных.

Вопрос 35

Что такое Self-Join и когда он используется?

> ***Self-Join*** *— это тип соединения в SQL, когда таблица объединяется сама с собой. Это полезно, когда нужно сравнить строки одной таблицы между собой или обработать иерархические данные.*

Когда используется Self-Join:

- **Иерархические структуры**

Например, в таблице сотрудников, где каждый сотрудник может иметь менеджера, также являющегося сотрудником в той же таблице.

- **Сравнение записей**

Для нахождения дубликатов или сравнения значений в разных строках одной таблицы.

Пример

employeeId	name	managerId
1	John	
2	Michail	1
3	Alisa	1
4	Max	2

Чтобы получить список сотрудников и их менеджеров:

```
SELECT e.name AS Employee, m.name AS Manager
FROM Employee e
LEFT JOIN Employee m ON e.managerId = m.employeeId;
```

В этом запросе мы соединяем таблицу `Employee` с самой собой, чтобы сопоставить каждого сотрудника с его менеджером.

Вопрос 36

Как выполнить резервное копирование и восстановление базы данных?

Резервное копирование и восстановление базы данных — это критически важные операции для обеспечения сохранности данных и возможности их восстановления в случае сбоя или потери.

Как выполнить резервное копирование базы данных:

Методы резервного копирования зависят от используемой системы управления базами данных (СУБД).

Ниже приведены примеры для некоторых популярных СУБД.

MySQL

Резервное копирование с помощью утилиты `mysqldump`:

```
mysqldump -u username -p mydatabase > backup.sql
```

- `username` — имя пользователя базы данных.
- `mydatabase` — имя базы данных, которую нужно скопировать.
- `backup.sql` — файл, куда будет сохранена резервная копия.

PostgreSQL

Резервное копирование с помощью утилиты `pg_dump`:

```
pg_dump -U username mydatabase > backup.sql
```

- `-U username` — имя пользователя базы данных.
- `mydatabase` — имя базы данных.
- `backup.sql` — выходной файл резервной копии.

Как восстановить базу данных из резервной копии:

MySQL

Восстановление базы данных из файла `backup.sql:`

```
mysql -u username -p mydatabase < backup.sql
```

PostgreSQL

Восстановление базы данных с помощью утилиты `psql`:

```
psql -U username mydatabase < backup.sql
```

Рекомендации

- **Права доступа**

Убедитесь, что у вас есть необходимые права для выполнения операций резервного копирования и восстановления.

- **Регулярность**

Настройте регулярное автоматическое резервное копирование для минимизации риска потери данных.

- **Хранение копий**

Сохраняйте резервные копии в безопасном и надежном месте, предпочтительно вне основного сервера.

- **Тестирование**

Периодически проверяйте резервные копии путем восстановления на тестовом сервере, чтобы убедиться в их целостности и работоспособности.

Вопрос 37

Как реализовать отношения многие-ко-многим в SQL?

Отношения **многие-ко-многим** в реляционных базах данных возникают, когда одна запись в первой таблице может соответствовать нескольким записям во второй таблице, и наоборот.

В SQL такие отношения реализуются с помощью промежуточной таблицы (также известной как таблица связей или соединительная таблица), которая связывает две основные таблицы посредством внешних ключей.

Как реализовать отношения многие-ко-многим в SQL

- **Создайте две основные таблицы**, которые нужно связать.
- **Создайте промежуточную таблицу**, содержащую внешние ключи на первичные ключи обеих основных таблиц.
- **Определите внешние ключи и составной первичный ключ** в промежуточной таблице для обеспечения ссылочной целостности и уникальности пар связей.

Пример реализации

Представим сценарий с таблицами `Student` (Студенты) и `Course` (Курсы), где один студент может записаться на несколько курсов, и один курс может быть пройден несколькими студентами.

- Создание таблицы студентов:

```
CREATE TABLE Student (
    StudentID INT PRIMARY KEY,
    Name VARCHAR(100)
);
```

- Создание таблицы курсов:

```
CREATE TABLE Course (
    CourseID INT PRIMARY KEY,
    Title VARCHAR(100)
);
```

- Создание промежуточной таблицы для установления связи многие-ко-многим:

```
CREATE TABLE StudentCourse (
    StudentID INT,
    CourseID INT,
    PRIMARY KEY (StudentID, CourseID),
    FOREIGN KEY (StudentID) REFERENCES Student(StudentID),
    FOREIGN KEY (CourseID) REFERENCES Course(CourseID)
);
```

В результате:

- **Промежуточная таблица** `StudentCourse` содержит пары `StudentID` и `CourseID`, представляющие связи между студентами и курсами.
- **Составной первичный ключ** (`StudentID` , `CourseID`) гарантирует, что каждая пара уникальна, предотвращая дублирование связей.
- **Внешние ключи** обеспечивают целостность данных, ссылаясь на соответствующие записи в таблицах `Student` и `Course` .

Вопрос 38

Как работают команды REVOKE и GRANT?

Команды `GRANT` и `REVOKE` в SQL используются для управления правами доступа пользователей к объектам базы данных. Они позволяют предоставлять или отзывать определенные привилегии у пользователей или ролей, о беспечивая безопасность и контроль над тем, кто и какие действия может выполнять в базе данных.

GRANT

Команда `GRANT` предоставляет пользователям или ролям определенные привилегии на объекты базы данных.

Синтаксис:

```
GRANT privileges ON object TO user [WITH GRANT OPTION];
```

- `privileges`: действия, которые разрешено выполнять (например, SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES).
- `object`: база данных, таблица, представление, процедура и т.д.
- `user`: имя пользователя или роли, которой предоставляются права.
- `WITH GRANT OPTION` (опционально): позволяет получателю привилегий передавать их другим пользователям.

Пример: Предоставить пользователю `user1` право выбора данных из таблицы `employees` можно следующим способом:

```
GRANT SELECT ON employees TO user1;
```

REVOKE

Команда `REVOKE` отзывает ранее предоставленные привилегии у пользователей или ролей.

Синтаксис:

```
REVOKE privileges ON object FROM user;
```

Пример: Отозвать у пользователя `user1` право выбора данных из таблицы `employees`:

```
REVOKE SELECT ON employees FROM user1;
```

Вопрос 39

Объясните использование функций хеширования в SQL

Хеш-функции в SQL используются для преобразования входных данных произвольной длины в фиксированную строку определенной длины. Это преобразование называется **хешированием**, и оно широко применяется для обеспечения безопасности, целостности данных и оптимизации операций сравнения и поиска.

Основные случаи использования хеш-функций в SQL:

1. Хранение паролей

- Вместо хранения паролей в открытом виде, их хранят в виде хешей. Это повышает безопасность, поскольку даже при компрометации базы данных злоумышленники не смогут восстановить исходные пароли.
- Используются криптографические хеш-функции, такие как `SHA-256`, `SHA-512` и другие.

2. Проверка целостности данных

Хеш-функции позволяют определить, были ли данные изменены. Создавая хеш от исходных данных и сравнивая его с текущим хешем, можно обнаружить изменения.

3. Индексирование и оптимизация поиск

Хеш-значения могут использоваться для быстрого сравнения больших объемов данных или создания индексов для ускорения поиска.

4. Создание уникальных идентификаторов:

Хеширование помогает генерировать уникальные идентификаторы для записей на основе их содержимого.

Пример использования хеш-функций в SQL:

```
-- Создаем таблицу пользователей
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Username VARCHAR(50) NOT NULL,
    PasswordHash VARBINARY(512) NOT NULL
);

-- При добавлении нового пользователя хешируем пароль
INSERT INTO Users (UserID, Username, PasswordHash)
VALUES (1, 'user1', HASHBYTES('SHA2_512', 'password123'));

-- Проверяем соответствие введенного пароля хешу в базе данных
SELECT UserID FROM Users
WHERE Username = 'user1' AND PasswordHash = HASHBYTES('SHA2_512', 'password123');
```

Рекомендации и особенности

Выбор хеш-функции

- Отдавайте предпочтение более современным и стойким алгоритмам, таким как `SHA-256` или `SHA-512`.
- Избегайте использования устаревших функций, таких как `MD5` или `SHA1`, из-за известных уязвимостей.

Безопасность хранения

Храните хеши в бинарном формате (`VARBINARY`) вместо строкового (`VARCHAR`) для сохранения точности и экономии места.

Необратимость хеша

Помните, что хеширование является односторонней функцией; исходные данные невозможно восстановить из хеша.

Вопрос 40

Как работает SQL-триггер?

Что такое SQL-триггер

> ***SQL-триггер*** *— это объект базы данных, представляющий собой специальный тип хранимой процедуры, которая автоматически выполняется при наступлении определенного события в базе данных. Эти события могут включать операции* `INSERT`, `UPDATE` *или* `DELETE` *на таблицах или представлениях.*

Как работает SQL-триггер

Триггер активируется автоматически в ответ на заданное событие (например, добавление новой записи в таблицу).

Типы триггеров

- `BEFORE` : Срабатывает до выполнения операции.

Используется для проверки или изменения данных перед их сохранением.

- `AFTER` : Срабатывает после выполнения операции.

Часто используется для ведения логов или обновления связанных таблиц.

- `INSTEAD OF` : Заменяет стандартное поведение операции.

Используется в случаях, когда необходимо переопределить действие по умолчанию, например, при работе с представлениями.

Внутри триггера можно обращаться к старым и новым значениям данных через специальные псевдотаблицы: `OLD` и `NEW`.

Пример использования триггера

Есть таблица `Employees`, и нужно автоматически сохранять историю изменений зарплат сотрудников в таблицу `SalaryHistory` при обновлении данных.

Создание таблиц:

```
-- Таблица сотрудников
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    Name VARCHAR(100),
    Salary DECIMAL(10, 2)
);

-- Таблица истории зарплат
CREATE TABLE SalaryHistory (
    EmployeeID INT,
    OldSalary DECIMAL(10, 2),
    NewSalary DECIMAL(10, 2),
    ChangeDate DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

Создание триггера:

```
DELIMITER $$

CREATE TRIGGER trg_AfterSalaryUpdate
AFTER UPDATE ON Employees
FOR EACH ROW
BEGIN
    IF OLD.Salary <> NEW.Salary THEN
        INSERT INTO SalaryHistory (EmployeeID, OldSalary, NewSalary)
        VALUES (NEW.EmployeeID, OLD.Salary, NEW.Salary);
    END IF;
END$$

DELIMITER ;
```

Когда использовать триггеры

- **Ведение журналов и аудита**:

Автоматическое логирование изменений данных для отслеживания действий пользователей.

- **Поддержание целостности данных**:

Обеспечение сложных бизнес-правил, не реализуемых с помощью ограничений.

- **Синхронизация данных**:

Автоматическое обновление или модификация связанных таблиц при изменении данных.

- **Вычисление значений**:

Автоматический расчет и обновление агрегированных или производных данных.