***Основные понятия теории баз данных: база данных, система управления базами данных, основные требования к информации в БД, модели данных, логическая схема БД, основная терминология реляционных баз данных.***

1. База данных: База данных (БД) - это организованная коллекция структурированных данных, которая обеспечивает эффективное хранение, управление и извлечение информации. БД представляет собой центральное хранилище данных для приложений.

2. Система управления базами данных (СУБД): Система управления базами данных (СУБД) - это программное обеспечение, которое позволяет создавать, управлять и обрабатывать базы данных. Оно предоставляет набор инструментов и функций для создания схемы базы данных, добавления данных, выполнения запросов и обеспечения безопасности данных.

3. Основные требования к информации в БД: К основным требованиям к информации в базе данных относятся:

- Полнота: БД должна содержать все необходимые данные для удовлетворения потребностей приложений.

- Точность: Данные в БД должны быть точными и соответствовать реальному миру.

- Единообразие: Данные должны быть структурированы и организованы по единым правилам.

- Надежность: БД должна быть надежной и обеспечивать сохранность данных при сбоях или ситуациях аварийного завершения работы.

4. Модели данных: Модель данных определяет способ представления данных в базе данных. Некоторые из наиболее распространенных моделей данных включают:

- Иерархическая модель: данные организованы в виде иерархии, состоящей из родительских и дочерних элементов.

- Сетевая модель: данные представлены в виде графа, где каждый узел может иметь несколько связей.

- Реляционная модель: данные представлены в виде таблиц (реляций), состоящих из строк (кортежей) и столбцов (атрибутов).

- Объектно-ориентированная модель: данные представлены в виде объектов с атрибутами и методами.

5. Логическая схема базы данных: Логическая схема базы данных определяет структуру данных, используемую в БД, включая таблицы, поля, связи между таблицами и ограничения целостности. Логическая схема определяет логическую организацию данных

без учета конкретной физической реализации.

6. Основная терминология реляционных баз данных:

- Таблица (реляция): Структура данных, представляющая собой двумерную сетку с рядами (кортежами) и столбцами (атрибутами).

- Ключ: Уникальный идентификатор для каждого кортежа в таблице. Ключ может быть составным или простым.

- Связь: Отношение между двумя таблицами, которое использует значения ключевых полей для установления связи между ними.

- Запрос: Команда для извлечения, изменения или добавления данных в базу данных.

- Нормализация: Процесс организации данных в базе данных, чтобы устранить избыточность и обеспечить эффективность и целостность данных.

Это основные понятия, связанные с теорией баз данных. Если у вас есть дополнительные вопросы, не стесняйтесь задавать их.

***Язык SQL.***

Язык SQL (Structured Query Language) - это язык программирования, используемый для управления и манипулирования данными в реляционных базах данных. Он предоставляет набор команд и операторов, которые позволяют создавать, изменять, извлекать и удалять данные в базе данных.

Вот некоторые основные команды и операторы SQL:

1. Команда SELECT: Используется для извлечения данных из базы данных.

2. Команда INSERT: Используется для добавления новых данных в таблицу.

3. Команда UPDATE: Используется для изменения существующих данных

4. Команда DELETE: Используется для удаления данных из таблицы.

5. Оператор WHERE: Используется для задания условия фильтрации при выполнении запроса.

6. Оператор JOIN: Используется для объединения данных из двух или более таблиц на основе связей между ними.

7. Оператор ORDER BY: Используется для сортировки результатов запроса по заданному столбцу.

8. Оператор GROUP BY: Используется для группировки результатов запроса по заданному столбцу.

Это лишь некоторые основные команды и операторы SQL. SQL также предоставляет множество других возможностей, включая агрегатные функции (например, SUM, AVG), подзапросы (subqueries), представления (views) и многое другое.

***Системные базы данных: master, msdb, model, tempdb.***

В Microsoft SQL Server есть несколько системных баз данных, которые выполняют важные функции для управления сервером и его объектами. Вот они:

1. master: База данных "master" содержит системную информацию, необходимую для управления SQL Server. В ней хранятся метаданные о других базах данных, информация о системных настройках, логины пользователей, информация о поддерживаемых функциях и многое другое. При запуске SQL Server сначала загружается база данных "master".

2. msdb: База данных "msdb" используется для управления и поддержки заданий (jobs), операций резервного копирования и восстановления, отправки почты, поддержки службы SQL Server Agent и других системных функций. В "msdb" хранятся информация о выполненных заданиях, расписаниях, истории резервного копирования и других операциях.

3. model: База данных "model" служит основой для создания новых баз данных. Когда создается новая база данных, SQL Server использует "model" в качестве шаблона для определения структуры, начальных настроек и объектов новой базы данных. Если вы изменяете "model", эти изменения будут отражаться во всех новых базах данных.

4. tempdb: База данных "tempdb" используется для временных объектов и временных данных во время выполнения операций в SQL Server. В "tempdb" хранятся временные таблицы, переменные таблицы, временные процедуры и другие временные объекты, которые создаются в процессе выполнения запросов. "tempdb" также используется для временного хранения промежуточных результатов при выполнении операций сортировки и объединения данных.

Эти системные базы данных важны для функционирования и управления SQL Server. Они предоставляют необходимую системную информацию и поддерживают различные системные операции и функции.

***Структура файла базы данных в SQL Server. Файловые группы.***

Структура файла базы данных в SQL Server состоит из файловых групп и файлов базы данных. Файловые группы представляют собой логическую организацию файлов базы данных, а файлы базы данных - это физические файлы, в которых хранятся данные.

Вот некоторая информация о файловых группах и их роли в структуре файла базы данных в SQL Server:

1. Файловая группа (Filegroup): Файловая группа - это логический контейнер, в котором хранятся файлы базы данных. Каждая база данных может содержать одну или несколько файловых групп. Файлы внутри одной файловой группы распределены по различным физическим дисковым устройствам, что обеспечивает возможность распределения данных и повышения производительности.

2. Файл базы данных (Database File): Файл базы данных представляет собой физический файл, который хранит данные базы данных. В SQL Server данные базы данных могут быть разделены на несколько файлов для распределения нагрузки, увеличения производительности и обеспечения отказоустойчивости.

3. Primary Filegroup: Primary Filegroup - это основная файловая группа, которая создается по умолчанию при создании новой базы данных. Она содержит системные таблицы и объекты базы данных. Primary Filegroup также может содержать пользовательские данные, если не были созданы дополнительные файловые группы.

4. User-defined Filegroup: Пользовательская файловая группа - это дополнительная файловая группа, которая может быть создана пользователем. Она позволяет организовать данные по своему усмотрению и логически разделить их. Например, можно создать файловую группу для хранения отдельных таблиц или индексов.

Использование файловых групп позволяет более гибко управлять распределением данных, улучшать производительность и обеспечивать отказоустойчивость в SQL Server. Каждая файловая группа может содержать один или несколько файлов базы данных, которые физически хранят данные на диске.

***Нормализация таблиц базы данных. Нормальные формы таблиц.***

Нормализация таблиц в базе данных - это процесс организации данных в соответствии с набором правил, называемых нормальными формами. Цель нормализации состоит в устранении избыточности данных, обеспечении целостности и минимизации аномалий при внесении изменений в данные.

Вот основные нормальные формы таблиц в базе данных:

1. Первая нормальная форма (1NF): Таблица находится в 1NF, если все ее атрибуты содержат только атомарные значения и не допускают повторений. Другими словами, каждая ячейка таблицы должна содержать только одно значение, а не список значений.

2. Вторая нормальная форма (2NF): Таблица находится в 2NF, если она находится в 1NF и каждый неключевой атрибут зависит от полного ключа таблицы. Это означает, что атрибуты, которые не являются ключевыми, не должны зависеть от части составного ключа.

3. Третья нормальная форма (3NF): Таблица находится в 3NF, если она находится в 2NF и каждый неключевой атрибут не зависит от других неключевых атрибутов. То есть атрибуты, которые не являются ключевыми, должны быть непосредственно зависимыми только от ключевых полей, а не от других атрибутов.

4. Четвертая нормальная форма (4NF): Таблица находится в 4NF, если она находится в 3NF и не имеет многозначных зависимостей, то есть случаев, когда один неключевой атрибут зависит от другого неключевого атрибута.

5. Пятая нормальная форма (5NF): Таблица находится в 5NF, также известной как нормальная форма проекции-соединения или форма Бойса-Кодда, если она находится в 4NF и не имеет зависимостей между неключевыми атрибутами через другие неключевые атрибуты.

Существуют также дополнительные нормальные формы, такие как шестая (6NF) и седьмая нормальные формы (7NF), но они редко используются в практике разработки баз данных и обычно применяются в специфических ситуациях.

Нормализация таблиц помогает улучшить структуру базы данных, сделать ее более гибкой, э

ффективной и устойчивой к аномалиям данных. Однако нормализация также может привести к увеличению числа таблиц и соединений, поэтому при проектировании базы данных необходимо найти правильный баланс между нормализацией и производительностью.

***Таблицы и их типы данных Microsoft SQL Server.***

Microsoft SQL Server предоставляет различные типы данных, которые могут быть использованы при создании таблиц. Вот некоторые наиболее часто используемые типы данных в SQL Server:

1. Целочисленные типы данных:

- INT: 32-битное целое число.

- BIGINT: 64-битное целое число.

- SMALLINT: 16-битное целое число.

- TINYINT: 8-битное беззнаковое целое число.

2. Десятичные и числовые типы данных:

- DECIMAL(p, s): Число с фиксированной точностью и масштабом, где p указывает общее количество цифр, а s - количество цифр после десятичной точки.

- NUMERIC(p, s): Аналогично DECIMAL.

- FLOAT: Плавающая точка с плавающей запятой числа двойной точности.

3. Типы данных для хранения символьных строк:

- VARCHAR(n): Строка переменной длины, где n указывает максимальное количество символов.

- CHAR(n): Строка фиксированной длины, где n указывает количество символов.

- TEXT: Большой объем текстовых данных.

4. Типы данных для хранения даты и времени:

- DATE: Дата без времени.

- TIME: Время без даты.

- DATETIME: Дата и время с точностью до миллисекунд.

- SMALLDATETIME: Дата и время с точностью до минуты.

5. Типы данных для хранения бинарных данных:

- BINARY(n): Бинарные данные фиксированной длины, где n указывает количество байт.

- VARBINARY(n): Бинарные данные переменной длины, где n указывает максимальное количество байт.

- IMAGE: Большой объем бинарных данных.

Это только некоторые из наиболее распространенных типов данных, доступных в SQL Server. Кроме того, SQL Server также предлагает типы данных для XML, JSON, географических данных и других специализированных типов данных.

***Подзапросы. Конструкции in, exists, all, any, some.***

Подзапросы - это запросы, которые включаются внутри другого запроса. Они позволяют выполнить вложенный запрос для получения набора данных, который затем используется внешним запросом.

В Microsoft SQL Server поддерживаются различные конструкции для работы с подзапросами. Ниже приведены некоторые из них, которые часто используются вместе с операторами IN, EXISTS, ALL, ANY и SOME:

1. Конструкция IN: Оператор IN используется для проверки, принадлежит ли значение столбца множеству значений, возвращаемому подзапросом. Пример использования:

SELECT column1 FROM table1 WHERE column2 IN (SELECT column3 FROM table2);

2. Конструкция EXISTS: Оператор EXISTS используется для проверки наличия результатов в подзапросе. Он возвращает TRUE, если подзапрос возвращает хотя бы одну строку, и FALSE в противном случае. Пример использования:

SELECT column1 FROM table1 WHERE EXISTS (SELECT column2 FROM table2 WHERE condition);

3. Конструкция ALL: Оператор ALL используется для сравнения значения столбца со всеми значениями, возвращаемыми подзапросом. Пример использования:

SELECT column1 FROM table1 WHERE column2 > ALL (SELECT column3 FROM table2);

4. Конструкция ANY/SOME: Операторы ANY и SOME используются для сравнения значения столбца с любым из значений, возвращаемых подзапросом. Они эквивалентны и могут использоваться взаимозаменяемо. Пример использования:

SELECT column1 FROM table1 WHERE column2 > ANY (SELECT column3 FROM table2);

В этих примерах `table1` и `table2` представляют таблицы, а `column1`, `column2` и `column3` представляют столбцы в этих таблицах. `condition` представляет условие, которое должно выполняться в подзапросе.

Конструкции IN, EXISTS, ALL, ANY и SOME позволяют более гибко работать с подзапросами в SQL Server и облегчают выполнение сложных запросов, включающих связанные данные из разных таблиц.

***Группировка данных с использованием cube, rollup***

Группировка данных с использованием CUBE и ROLLUP - это расширения языка SQL, которые позволяют создавать расширенные отчеты с иерархическими итогами и сводными данными. Они предоставляют возможность выполнять множественные уровни группировки данных по различным атрибутам.

1. CUBE: Конструкция CUBE позволяет создавать все возможные комбинации итогов по заданным атрибутам. В результате получается кросс-таблица с полным набором итогов. Это полезно для анализа данных во всех возможных измерениях. Пример использования:

SELECT column1, column2, SUM(column3)

FROM table1

GROUP BY CUBE (column1, column2);

В этом примере будут созданы все возможные комбинации итогов по `column1` и `column2`.

2. ROLLUP: Конструкция ROLLUP позволяет создавать иерархические итоги по заданным атрибутам. Она создает итоги для каждого уровня группировки данных, начиная с самого детального уровня и постепенно добавляя общие итоги. Пример использования:

SELECT column1, column2, SUM(column3)

FROM table1

GROUP BY ROLLUP (column1, column2);

В этом примере будут созданы итоги для каждого уровня группировки данных по `column1` и `column2`.

Использование CUBE и ROLLUP позволяет создавать более детальные итоговые отчеты и проводить анализ данных в различных измерениях. Эти конструкции особенно полезны при работе с большими объемами данных и сложными иерархическими структурами.

***Операторы union (all), intersect, except.***

Операторы UNION, UNION ALL, INTERSECT и EXCEPT - это операторы множественного объединения, которые позволяют комбинировать результаты нескольких запросов в один набор данных.

1. UNION: Оператор UNION используется для объединения результатов двух или более запросов в один набор данных. Дубликаты строк удаляются, и только уникальные строки возвращаются в результате. Пример использования:

SELECT column1, column2 FROM table1

UNION

SELECT column1, column2 FROM table2;

2. UNION ALL: Оператор UNION ALL также объединяет результаты нескольких запросов в один набор данных, но в отличие от оператора UNION, он не удаляет дубликаты строк. В результате могут присутствовать повторяющиеся строки. Пример использования:

SELECT column1, column2 FROM table1

UNION ALL

SELECT column1, column2 FROM table2;

3. INTERSECT: Оператор INTERSECT используется для получения пересечения результатов двух или более запросов. Он возвращает только те строки, которые присутствуют во всех запросах. Пример использования:

SELECT column1, column2 FROM table1

INTERSECT

SELECT column1, column2 FROM table2;

4. EXCEPT: Оператор EXCEPT используется для получения разности результатов двух запросов. Он возвращает только те строки, которые присутствуют в первом запросе и отсутствуют во втором запросе. Пример использования:

SELECT column1, column2 FROM table1

EXCEPT

SELECT column1, column2 FROM table2;

Во всех этих операторах структура запросов должна быть одинаковой, и количество и типы столбцов должны соответствовать друг другу.

Операторы UNION, INTERSECT и EXCEPT могут использоваться только для комбинирования запросов, возвращающих одинаковое количество столбцов. Оператор UNION ALL может объединять запросы с различными структурами столбцов.

***Операторы соединения таблиц.***

Операторы соединения таблиц используются в SQL для объединения данных из двух или более таблиц на основе общих столбцов. Это позволяет комбинировать данные из разных таблиц и получать более полные и связанные результаты запросов. Вот некоторые наиболее распространенные операторы соединения таблиц:

1. INNER JOIN: Оператор INNER JOIN используется для получения только совпадающих строк из двух таблиц. Он соединяет строки, у которых значения в указанных столбцах совпадают. Пример использования:

SELECT \*

FROM table1

INNER JOIN table2 ON table1.column = table2.column;

2. LEFT JOIN (или LEFT OUTER JOIN): Оператор LEFT JOIN возвращает все строки из левой таблицы и совпадающие строки из правой таблицы. Если строки в правой таблице не совпадают, то в результате будут NULL значения для столбцов правой таблицы. Пример использования:

SELECT \*

FROM table1

LEFT JOIN table2 ON table1.column = table2.column;

3. RIGHT JOIN (или RIGHT OUTER JOIN): Оператор RIGHT JOIN возвращает все строки из правой таблицы и совпадающие строки из левой таблицы. Если строки в левой таблице не совпадают, то в результате будут NULL значения для столбцов левой таблицы. Пример использования:

SELECT \*

FROM table1

RIGHT JOIN table2 ON table1.column = table2.column;

4. FULL JOIN (или FULL OUTER JOIN): Оператор FULL JOIN возвращает все строки из обеих таблиц. Если строки в одной таблице не совпадают с другой таблицей, то в результате будут NULL значения для соответствующих столбцов. Пример использования:

SELECT \*

FROM table1

FULL JOIN table2 ON table1.column = table2.column;

Это лишь некоторые из операторов соединения таблиц, которые можно использовать в SQL. Каждый из них предоставляет различные способы объединения данных из разных таблиц в соответствии с определенными правилами. Выбор конкретного оператора зависит от требуемого результата и структуры данных в таблицах.

***Язык T-SQL. Пакеты. Объявление переменных. Операторы присвоения Оператор цикла while.***

T-SQL (Transact-SQL) - это расширение языка SQL, используемое в Microsoft SQL Server для работы с базами данных. В T-SQL есть поддержка пакетов, объявления переменных, операторов присвоения и оператора цикла WHILE. Рассмотрим их подробнее:

1. Пакеты: Пакеты в T-SQL представляют собой набор процедур, функций и других объектов базы данных, которые связаны между собой и могут быть использованы для выполнения определенных задач. Пакеты облегчают организацию и повторное использование кода. В SQL Server пакеты реализуются в виде хранимых процедур.

2. Объявление переменных: В T-SQL можно объявлять переменные, которые используются для хранения временных значений в рамках выполнения запроса или процедуры. Объявление переменной начинается с ключевого слова DECLARE, за которым следует имя переменной и ее тип данных. Пример объявления переменной:

DECLARE @variable\_name data\_type;

3. Операторы присвоения: Операторы присвоения используются для присвоения значения переменной. В T-SQL основным оператором присвоения является оператор "=" (равно). Пример использования:

SET @variable\_name = value;

4. Оператор цикла WHILE: Оператор WHILE используется для выполнения блока кода или запроса до тех пор, пока указанное условие истинно. Он позволяет создавать циклические операции. Пример использования:

DECLARE @counter INT = 1;

WHILE @counter <= 10

BEGIN

-- Код, который будет выполняться в цикле

PRINT 'Counter: ' + CAST(@counter AS VARCHAR);

SET @counter = @counter + 1;

END;

В этом примере блок кода будет выполняться, пока значение переменной `@counter` не превысит 10. В каждой итерации цикла значение переменной `@counter` увеличивается на 1.

Операторы присвоения и цикл WHILE позволяют управлять потоком выполнения кода и манипулировать значениями переменных в T-SQL. Они предоставляют гибкие инструменты для реализации более сложной логики обработки данных в SQL Server.

***Язык T-SQL. Операторы print, if-else, case. Операторы begin-end, waitfor и return.***

В T-SQL, языке запросов и программирования для Microsoft SQL Server, доступны различные операторы, включая операторы вывода (`PRINT`), условные операторы (`IF-ELSE`, `CASE`), операторы блока кода (`BEGIN-END`), оператор `WAITFOR` и оператор `RETURN`. Давайте рассмотрим каждый из них подробнее:

1. Оператор `PRINT`: Оператор `PRINT` используется для вывода сообщений или значений на консоль или в окно результатов запроса. Пример использования:

PRINT 'Hello, world!';

2. Условные операторы:

- `IF-ELSE`: Условный оператор `IF-ELSE` позволяет выполнить блок кода в зависимости от условия. Если условие истинно, выполняется блок кода после `IF`. В противном случае, если есть блок кода после `ELSE`, он будет выполнен. Пример использования:

IF condition

BEGIN

-- Код, выполняемый при истинном условии

END

ELSE

BEGIN

-- Код, выполняемый при ложном условии

END

- `CASE`: Условный оператор `CASE` позволяет выполнить различные действия в зависимости от значения выражения или столбца. Он может использоваться как в инструкции `SELECT`, так и в блоке кода. Пример использования в инструкции `SELECT`:

SELECT column1,

CASE

WHEN condition1 THEN value1

WHEN condition2 THEN value2

ELSE value3

END AS column\_alias

FROM table;

Пример использования в блоке кода:

DECLARE @variable INT;

SET @variable = 1;

CASE @variable

WHEN 1 THEN

BEGIN

-- Код, выполняемый при условии равенства 1

END

WHEN 2 THEN

BEGIN

-- Код, выполняемый при условии равенства 2

END

ELSE

BEGIN

-- Код, выполняемый при других значениях

END

END;

3. Операторы блока кода:

- `BEGIN-END`: Операторы `BEGIN` и `END` используются для определения блока кода. Они обозначают начало и конец блока кода, который может содержать одну или несколько инструкций. Блок кода часто используется совместно с условными операторами, циклами и хранимыми процедурами.

- `WAITFOR`: Оператор `WAITFOR` используется для задержки выполнения кода на определенное время. Он может быть полезен, например, для создания задержки между выполнением двух операций или для временной остановки выполнения

кода. Пример использования:

WAITFOR DELAY '00:00:05'; -- Приостановить выполнение на 5 секунд

4. Оператор `RETURN`: Оператор `RETURN` используется для возврата значения из хранимой процедуры или функции. Он указывает, что выполнение кода должно быть прекращено и возвращено указанное значение. Пример использования:

CREATE FUNCTION dbo.MyFunction()

RETURNS INT

AS

BEGIN

-- Код функции

RETURN 42; -- Вернуть значение 42

END;

Операторы `PRINT`, `IF-ELSE`, `CASE`, `BEGIN-END`, `WAITFOR` и `RETURN` предоставляют различные возможности для организации и управления выполнением кода в T-SQL, в зависимости от требуемой логики и условий.

***Язык T-SQL. Обработка ошибок в конструкциях try-catch. Функция RAISERROR.***

В T-SQL для обработки ошибок используется конструкция `TRY-CATCH`, которая позволяет перехватывать и обрабатывать исключения (ошибки), возникающие в ходе выполнения кода. При возникновении ошибки выполнение кода передается в блок `CATCH`, где можно выполнить определенные действия в ответ на ошибку. Кроме того, для генерации исключений можно использовать функцию `RAISERROR`. Рассмотрим эти возможности подробнее:

1. Конструкция `TRY-CATCH`:

BEGIN TRY

-- Блок кода, в котором возможны ошибки

END TRY

BEGIN CATCH

-- Блок кода, выполняемый при возникновении ошибки

END CATCH

При выполнении кода в блоке `TRY` если происходит ошибка, то выполнение кода переходит в блок `CATCH`, где можно обработать ошибку. В блоке `CATCH` можно выполнять различные действия, например, выводить сообщение об ошибке, записывать информацию об ошибке в журнал или выполнять дополнительную обработку.

2. Функция `RAISERROR`:

Функция `RAISERROR` используется для генерации пользовательских сообщений об ошибках или исключениях. Она позволяет указать пользовательский текст сообщения, уровень ошибки и необходимость прерывания выполнения кода. Пример использования:

RAISERROR ('Ошибка: делим на ноль.', 16, 1);

В данном примере вызывается ошибка с пользовательским текстом "Ошибка: делим на ноль." и уровнем ошибки 16. Уровень ошибки может быть от 1 до 25. При указании уровня ошибки выше 18 выполнение текущего блока кода прерывается и код передается в блок `CATCH`.

Функция `RAISERROR` также позволяет передавать дополнительные параметры для форматирования текста сообщения и включения дополнительной информации об ошибке.

Конструкция `TRY-CATCH` и функция `RAISERROR` являются мощными инструментами для обработки ошибок в T-SQL. Они позволяют контролировать поток выполнения кода и предоставляют возможность более гибкой обработки исключительных ситуаций.

***Локальные и глобальные временные таблицы в SQL Server.***

В SQL Server существуют два типа временных таблиц: локальные и глобальные временные таблицы. Они оба предназначены для временного хранения данных внутри сеанса или сеансов пользователей, но имеют некоторые различия в своем поведении и области видимости.

1. Локальные временные таблицы:

- Локальные временные таблицы создаются с префиксом `#` перед их именем. Например, `#TempTable`.

- Они доступны только в рамках текущего сеанса пользователя, который их создал, и существуют до завершения сеанса или до явного удаления таблицы.

- Локальные временные таблицы могут использоваться во всех блоках кода, включая хранимые процедуры, функции и триггеры.

- Они полезны, когда требуется временное хранение данных, специфичных для текущего сеанса.

Пример создания и использования локальной временной таблицы:

CREATE TABLE #TempTable (

ID INT,

Name VARCHAR(50)

);

INSERT INTO #TempTable (ID, Name)

VALUES (1, 'John'), (2, 'Jane');

SELECT \* FROM #TempTable;

DROP TABLE #TempTable;

2. Глобальные временные таблицы:

- Глобальные временные таблицы создаются с префиксом `##` перед их именем. Например, `##GlobalTempTable`.

- Они видны и доступны для всех пользователей и сеансов в пределах текущей базы данных.

- Глобальные временные таблицы существуют до тех пор, пока все сеансы, использующие их, завершатся, или пока они не будут удалены явно.

- Они могут использоваться для обмена данными между различными сеансами или для временного хранения данных, которые должны быть доступны всем пользователям.

Пример создания и использования глобальной временной таблицы:

CREATE TABLE ##GlobalTempTable (

ID INT,

Name VARCHAR(50)

);

INSERT INTO ##GlobalTempTable (ID, Name)

VALUES (1, 'John'), (2, 'Jane');

SELECT \* FROM ##GlobalTempTable;

DROP TABLE ##GlobalTempTable;

Оба типа временных таблиц в SQL Server могут быть полезными инструментами для временного хранения данных в рамках конкретного сеанса или для обмена данными между различными сеансами. Выбор между локальными и глобальными временными таблицами зависит от конкретных требований вашего приложения и области видимости, которую вы хотите иметь для таблицы.

***Курсоры в SQL Server. Объявление курсора. Общая схема работы с курсором: declare, open, fetch, close, deallocate. Типы курсоров.***

Курсоры в SQL Server используются для итерации по результирующему набору данных и последовательного доступа к его записям. Курсоры позволяют выполнять операции с каждой записью, такие как чтение, обновление или удаление.

Общая схема работы с курсором в SQL Server включает следующие шаги:

1. Объявление курсора (`DECLARE`): Сначала необходимо объявить курсор, указав его имя и определение запроса, который будет использоваться для получения результирующего набора данных. Пример:

DECLARE cursor\_name CURSOR FOR

SELECT column1, column2

FROM table;

2. Открытие курсора (`OPEN`): После объявления курсора его необходимо открыть, чтобы выполнить запрос и получить результирующий набор данных. Пример:

OPEN cursor\_name;

3. Извлечение данных (`FETCH`): Далее можно извлекать данные из курсора одну запись за другой с помощью оператора `FETCH`. Запись считывается и перемещается курсором на следующую позицию в результирующем наборе. Пример:

FETCH NEXT FROM cursor\_name INTO @variable1, @variable2;

В данном примере значения полей `column1` и `column2` из текущей записи результирующего набора копируются в переменные `@variable1` и `@variable2`.

4. Закрытие курсора (`CLOSE`): По завершении работы с курсором его следует закрыть, чтобы освободить ресурсы. Пример:

CLOSE cursor\_name;

5. Освобождение курсора (`DEALLOCATE`): После закрытия курсора его можно освободить с помощью оператора `DEALLOCATE`. Пример:

DEALLOCATE cursor\_name;

Типы курсоров в SQL Server:

- `STATIC` (статический курсор): Результирующий набор фиксирован и доступен только для чтения.

- `DYNAMIC` (динамический курсор): Результирующий набор динамически изменяется, поэтому изменения в наборе могут быть видны при итерации по курсору.

- `FORWARD\_ONLY` (курсор только вперед): Курсор можно передвигать только вперед по результирующему набору. Нельзя перемещаться назад или изменять данные.

- `KEYSET` (курсор по ключевому набору): Курсор содержит только уникальные ключи из результирующего

набора, что обеспечивает более быстрый доступ к данным.

- `FAST\_FORWARD` (курсор прямого доступа): Курсор позволяет только последовательное чтение данных вперед, без возможности перемещения назад или изменения данных.

Выбор типа курсора зависит от конкретных требований приложения и характеристик результирующего набора данных.

***Хранимые процедуры и функции T-SQL. Создание хранимых процедур. Передача параметров. Входные и выходные параметры.***

Хранимые процедуры и функции в T-SQL позволяют создавать предварительно скомпилированные и сохраненные в базе данных блоки кода, которые можно вызывать для выполнения определенных операций. Они предоставляют удобный способ организации и повторного использования логики при работе с базами данных. Рассмотрим создание хранимых процедур и функций, а также передачу параметров в них.

1. Создание хранимых процедур:

Хранимая процедура в T-SQL может принимать ноль или более параметров и выполнять одну или несколько операций. Пример создания хранимой процедуры:

CREATE PROCEDURE ProcedureName

@param1 datatype1,

@param2 datatype2

AS

BEGIN

-- Тело процедуры

-- Код операций

END;

В приведенном примере `ProcedureName` - это имя хранимой процедуры, `@param1` и `@param2` - входные параметры с указанием их типов данных.

2. Передача параметров в хранимую процедуру:

При вызове хранимой процедуры значения переданных параметров должны быть указаны. Пример вызова хранимой процедуры:

EXEC ProcedureName @param1 = value1, @param2 = value2;

В данном примере `value1` и `value2` - это значения, которые передаются в соответствующие параметры при вызове процедуры.

3. Входные и выходные параметры:

Хранимая процедура может иметь входные параметры, которые используются для передачи значений внутрь процедуры, и выходные параметры, которые возвращают значения из процедуры. Пример объявления и использования входных и выходных параметров:

CREATE PROCEDURE ProcedureName

@inputParam datatype,

@outputParam datatype OUTPUT

AS

BEGIN

-- Код операций, использующий @inputParam

SET @outputParam = value; -- Присвоение значения выходному параметру

END;

При вызове процедуры значения входных параметров указываются, а значения выходных параметров присваиваются переменным в вызывающем коде. Пример вызова процедуры с использованием выходного параметра:

DECLARE @outputVariable datatype;

EXEC ProcedureName @inputParam = value, @outputParam = @outputVariable OUTPUT;

PRINT @outputVariable; -- Вывод значения выходного параметра

В данном примере значение `value` передается во входной параметр

`@inputParam`, а значение выходного параметра присваивается переменной `@outputVariable`.

Хранимые функции имеют похожий синтаксис, но возвращают результаты, которые могут быть использованы внутри запросов. Создание хранимых функций и передача параметров в них осуществляется аналогично хранимым процедурам.

***Хранимые процедуры и функции T-SQL. Виды функций. Создание функций. Передача параметров.***

В T-SQL существуют различные типы функций, которые могут быть созданы и использованы в запросах. Рассмотрим основные виды функций, их создание и передачу параметров.

1. Виды функций в T-SQL:

- Скалярные функции (Scalar Functions): Возвращают одно значение и могут быть использованы в выражениях. Например, функция для вычисления суммы или конкатенации строк.

- Табличные функции (Table Functions): Возвращают таблицу как результат и могут быть использованы вместо обычной таблицы в запросах. Табличные функции могут возвращать одну или несколько строк.

- Функции с возвращаемым таблицей (Table-Valued Functions): Представляют собой специальный тип табличных функций, которые возвращают таблицу с определенной структурой. Они могут быть параметризованы, принимать входные параметры и использоваться вместе с оператором `FROM`.

- Агрегатные функции (Aggregate Functions): Выполняют агрегатные операции над группой значений и возвращают одно значение. Примеры агрегатных функций: `SUM`, `AVG`, `COUNT`, `MIN`, `MAX`.

2. Создание функций:

Синтаксис создания функций может отличаться в зависимости от типа функции. Однако общая структура включает определение имени функции, списка параметров и тела функции, содержащего операторы и выражения.

Пример создания скалярной функции:

CREATE FUNCTION FunctionName

(@param1 datatype, @param2 datatype)

RETURNS returnType

AS

BEGIN

-- Тело функции

RETURN result;

END;

Пример создания табличной функции:

CREATE FUNCTION FunctionName

(@param1 datatype, @param2 datatype)

RETURNS TABLE

AS

RETURN

(

SELECT columns

FROM table

WHERE condition

);

3. Передача параметров в функции:

Параметры функций объявляются в списке параметров и могут быть использованы внутри тела функции. При вызове функции значения параметров должны быть переданы.

Пример вызова скалярной функции:

DECLARE @result datatype;

SET @result = dbo.FunctionName(@param1, @param2);

Пример использования табличной функции:

SELECT columns

FROM dbo.FunctionName(@param1, @param2);

В обоих примерах `@param1` и `@param2` - это значения, которые передаются в параметры функции `FunctionName`.

Обратите внимание, что для вызова функции в T-SQL используется

префикс `dbo.` перед именем функции, если функция находится в схеме "dbo".

***Индексы SQL Server. Назначение и применение индексов. Виды индексов. Применение различных видов индексов. Оптимизация запросов.***

Индексы в SQL Server предназначены для ускорения выполнения запросов и повышения производительности базы данных. Индексы представляют структуры данных, создаваемые на основе одного или нескольких столбцов таблицы, которые позволяют быстро находить и извлекать нужные данные. Рассмотрим основные виды индексов и их применение.

1. Кластерные индексы (Clustered Indexes):

- Каждая таблица может иметь только один кластерный индекс.

- Определяют физический порядок данных в таблице.

- Позволяют быстро выполнять операции поиска и сортировки на основе ключа.

- Используются для группировки данных и оптимизации запросов на основе ключевого столбца.

2. Некластерные индексы (Non-Clustered Indexes):

- Каждая таблица может иметь несколько некластерных индексов.

- Создают отдельную структуру данных, содержащую ключевые столбцы и ссылки на физическое расположение данных в таблице.

- Используются для ускорения операций поиска, фильтрации и сортировки.

- Рекомендуется создавать некластерные индексы для столбцов, которые часто используются в запросах для фильтрации и сортировки данных.

3. Покрывающие индексы (Covering Indexes):

- Включают в себя все столбцы, необходимые для выполнения запроса, без обращения к самой таблице.

- Позволяют выполнить запрос полностью через индекс, без обращения к основной таблице.

- Увеличивают производительность запросов за счет уменьшения операций ввода-вывода и обращений к диску.

4. Уникальные индексы (Unique Indexes):

- Гарантируют уникальность значений в индексированном столбце или наборе столбцов.

- Используются для предотвращения вставки дублирующихся значений в индексированные столбцы.

- Могут быть кластерными или некластерными.

5. Полнотекстовые индексы (Full-Text Indexes):

- Предназначены для выполнения полнотекстового поиска, который позволяет искать текстовые данные по ключевым словам и фразам.

- Используются для оптимизации поиска в больших объемах текстовой информации.

Применение различных видов индексов зависит от конкретных требований и запросов к базе данных. Оптимизация запросов с использованием индексов включает правильное создание и использование индексов для улучшения производительности запросов. Необходимо анализировать структуру таблиц и запросов, а также обращаться к инструментам анализа выполнения запросов (например, плану выполнения запроса) для определения наиболее эффективных индексов. При этом следует учитывать, что индексы требуют дополнительного пространства на диске и могут влиять на производительность при выполнении операций изменения данных (вставка, обновление, удаление).

***План запроса. Этапы обработки select запроса. Понятие стоимости запроса. Понятия селективности и плотности. Индексы. Реорганизация, перестроение, включение и отключение индексов.***

План запроса (Query Plan) представляет собой план выполнения запроса, который определяет последовательность операций, выполняемых СУБД для получения результата запроса. План запроса состоит из нескольких этапов обработки SELECT запроса. Рассмотрим основные этапы обработки SELECT запроса:

1. Разбор запроса (Query Parsing): СУБД анализирует синтаксис запроса и проверяет его корректность. В этом этапе СУБД также определяет объекты, с которыми будет работать запрос (таблицы, столбцы и т.д.).

2. Оптимизация запроса (Query Optimization): СУБД выбирает оптимальный план выполнения запроса. В этом этапе СУБД анализирует статистику таблиц, доступные индексы, и другую информацию для принятия решений о порядке выполнения операций и выборе наилучших индексов и методов доступа к данным.

3. Выполнение плана (Plan Execution): СУБД выполняет план запроса, последовательно выполняя операции, необходимые для получения результата запроса. Это может включать операции сканирования таблиц, слияния данных, сортировки, объединения и другие операции, определенные в плане.

Стоимость запроса (Query Cost) представляет собой оценку затрат СУБД на выполнение запроса. Стоимость запроса может быть выражена в виде числа, которое указывает на ожидаемую нагрузку на систему при выполнении запроса. Сравнение стоимости запросов позволяет выбрать наиболее оптимальный план выполнения запроса.

Селективность (Selectivity) относится к количеству выбранных строк относительно общего количества строк в таблице или результате фильтрации. Высокая селективность означает, что малая доля строк будет выбрана из таблицы или результата фильтрации.

Плотность (Density) относится к распределению значений в столбце таблицы. Низкая плотность означает, что в столбце присутствуют многие уникальные значения, а высокая плотность означает, что в столбце много повторяющихся значений.

Индексы являются структурами данных, создаваемыми на основе одного или нескольких столбцов таблицы, которые позволяют ускорить выполнение запросов. Они обеспечивают быстрый доступ к данным и уменьшают необходимость в сканировании

всей таблицы при выполнении запросов. Индексы могут быть созданы на основе одного или нескольких столбцов и могут быть уникальными или неуникальными.

Реорганизация индексов (Index Reorganization) является процессом физического перестроения индексов для устранения фрагментации и повышения производительности. При реорганизации индексы остаются доступными для запросов.

Перестроение индексов (Index Rebuild) представляет собой полное перестроение индекса с нуля. В процессе перестроения индекса таблица блокируется для записи.

Включение и отключение индексов (Index Enable/Disable) позволяет временно отключать или включать индексы для определенных таблиц. Это может быть полезно в случаях, когда необходимо провести операции загрузки данных или произвести массовые изменения, и индексы могут замедлить выполнение этих операций.

Оптимизация индексов и их использование играют важную роль в повышении производительности запросов и общей эффективности базы данных. При проектировании базы данных необходимо учитывать типы запросов и выбирать соответствующие индексы, а также регулярно поддерживать индексы, выполняя реорганизацию или перестроение, чтобы сохранить их эффективность.

***Триггеры SQL Server. Типы триггеров. Создание after-триггера.***

Триггеры в SQL Server представляют собой специальные хранимые процедуры, которые автоматически выполняются при возникновении определенных событий (например, вставка, обновление или удаление данных) в таблице. Рассмотрим типы триггеров и процесс создания after-триггера.

Типы триггеров в SQL Server:

1. After-триггеры (After Triggers): Выполняются после выполнения операции вставки, обновления или удаления данных в таблице. Используются для выполнения дополнительных действий, связанных с изменениями данных.

2. Instead Of-триггеры (Instead Of Triggers): Заменяют стандартное выполнение операций вставки, обновления или удаления данных. Используются для изменения поведения операций на основе определенных условий.

3. DDL-триггеры (DDL Triggers): Срабатывают при выполнении операций на уровне структуры базы данных, таких как создание, изменение или удаление таблиц, представлений и процедур. Используются для отслеживания и контроля изменений структуры базы данных.

Создание after-триггера в SQL Server:

1. Определите хранимую процедуру, которая будет являться триггером.

CREATE PROCEDURE AfterTriggerProcedure

AS

BEGIN

-- Код триггера

END

2. Создайте триггер, связанный с определенной таблицей, который будет выполняться после операции вставки, обновления или удаления.

CREATE TRIGGER AfterTrigger

ON TableName

AFTER INSERT, UPDATE, DELETE

AS

BEGIN

EXEC AfterTriggerProcedure

END

В приведенном примере создается after-триггер с именем "AfterTrigger", связанный с таблицей "TableName". Триггер будет выполняться после операций вставки, обновления или удаления данных и вызывать хранимую процедуру "AfterTriggerProcedure".

При создании триггеров необходимо учитывать их влияние на производительность и обеспечить правильное управление транзакциями и блокировками, чтобы избежать проблем с целостностью данных.

***Триггеры SQL Server. Создание и назначение instead of-триггеров. Таблицы inserted, deleted.***

Instead Of-триггеры (Instead Of Triggers) в SQL Server используются для замены стандартного выполнения операций вставки, обновления или удаления данных. В отличие от after-триггеров, которые выполняются после операций, Instead Of-триггеры выполняются вместо них и позволяют пользователю полностью контролировать и изменять результаты операций.

Instead Of-триггеры обычно используются в следующих случаях:

1. Представления с включением операций DML (Data Manipulation Language): Instead Of-триггеры позволяют определить пользовательское поведение для операций вставки, обновления или удаления данных в представлениях.

2. Контроль целостности данных: Instead Of-триггеры позволяют проверять и изменять данные перед их вставкой, обновлением или удалением. Это может быть полезно для применения бизнес-правил и ограничений целостности данных.

3. Аудит изменений: Instead Of-триггеры могут использоваться для отслеживания и записи изменений данных, выполняемых пользователем.

В контексте Instead Of-триггеров в SQL Server, таблицы "inserted" и "deleted" представляют временные таблицы, которые используются в триггерах для доступа к вставленным, обновленным и удаленным данным.

- Таблица "inserted": Содержит новые или измененные строки, которые вставляются или обновляются в таблице.

- Таблица "deleted": Содержит удаленные или измененные строки, которые удаляются или обновляются в таблице.

Обе таблицы имеют ту же самую структуру, что и таблица, к которой привязан Instead Of-триггер, и могут быть использованы для выполнения операций с данными в триггере.

Пример создания Instead Of-триггера:

CREATE TRIGGER InsteadOfTrigger

ON TableName

INSTEAD OF INSERT, UPDATE, DELETE

AS

BEGIN

-- Код триггера

END

В приведенном примере создается Instead Of-триггер с именем "InsteadOfTrigger", связанный с таблицей "TableName". Триггер будет выполняться вместо операций вставки, обновления или удаления данных и будет содержать пользовательский код для обработки данных.

***Транзакции. Свойства ACID. Транзакции. Уровни изолированности транзакций. Функция trancount.***

Транзакции в базах данных являются механизмом для группировки одного или нескольких операторов базы данных в одно логическое действие. Они обеспечивают целостность данных и позволяют контролировать успешное выполнение или отмену серии операций.

Свойства ACID являются основными характеристиками транзакций в базах данных:

1. Атомарность (Atomicity): Транзакция является атомарной, что означает, что она выполняется полностью или не выполняется вообще. Все операции внутри транзакции либо выполняются успешно и фиксируются (commit), либо отменяются (rollback).

2. Согласованность (Consistency): Транзакция должна поддерживать целостность данных и переводить базу данных из одного согласованного состояния в другое. Если транзакция нарушает целостность данных, она должна быть отменена.

3. Изолированность (Isolation): Транзакции выполняются изолированно друг от друга, что означает, что операции в одной транзакции не видны другим транзакциям до их фиксации (commit). Это предотвращает нежелательные взаимодействия между транзакциями, такие как чтение нефиксированных данных (dirty read) или неповторяющееся чтение (non-repeatable read).

4. Долговечность (Durability): Когда транзакция фиксируется (commit), ее результаты должны быть постоянными и долговечными. Даже в случае отказа системы или сбоя, результаты фиксированных транзакций должны быть сохранены.

Уровни изолированности транзакций определяют степень изоляции между транзакциями и контролируют видимость изменений, внесенных одной транзакцией, для других транзакций. В SQL Server доступны следующие уровни изолированности:

1. READ UNCOMMITTED: Допускает чтение неподтвержденных данных из других транзакций (dirty read). Этот уровень обеспечивает самую низкую степень изоляции.

2. READ COMMITTED: Разрешает чтение только подтвержденных данных. Однако, другие транзакции могут изменять данные, что может привести к неповторяющемуся чтению (non-repeatable read) или фантомному чтению (phantom read).

3. REPEATABLE READ: Гарантирует, что данные, прочитанные внутри транзакции, не изменятся до ее завершения. Это предотвращает неповторяющееся чтение, но не предотвращает фантомное чтение.

4. SERIALIZABLE: Обеспечивает максимальный уровень изоляции. Гарантирует, что транзакции выполняются последовательно, избегая неповторяющегося чтения и фантомного чтения.

Функция @@TRANCOUNT в SQL Server возвращает количество активных транзакций в текущем контексте выполнения. Значение 0 указывает на отсутствие активных транзакций, значение 1 указывает на наличие одной активной транзакции, и так далее. Это может быть использовано для проверки наличия активных транзакций и принятия соответствующих действий в коде T-SQL.

***Операторы TCL. Привилегии. Роли. Назначение привилегий.***

Операторы TCL (Transactional Control Language) в SQL Server предоставляют возможность управления транзакциями и назначением привилегий пользователей. Основные операторы TCL включают COMMIT, ROLLBACK и GRANT/REVOKE.

1. COMMIT: Оператор COMMIT используется для фиксации (завершения) текущей транзакции. Когда COMMIT выполняется, все изменения, внесенные в базу данных в рамках текущей транзакции, становятся постоянными.

2. ROLLBACK: Оператор ROLLBACK используется для отмены текущей транзакции и возврата базы данных к ее предыдущему состоянию. Все изменения, внесенные в рамках текущей транзакции, отменяются.

3. GRANT/REVOKE: Операторы GRANT и REVOKE используются для назначения и отзыва привилегий пользователей на объекты базы данных. GRANT используется для предоставления привилегий, таких как SELECT, INSERT, UPDATE, DELETE и других, пользователю или роли. REVOKE используется для отзыва ранее предоставленных привилегий.

Привилегии в SQL Server позволяют контролировать доступ пользователей к объектам базы данных. Некоторые привилегии включают:

- SELECT: Позволяет пользователю выполнять операции чтения данных из таблиц.

- INSERT: Позволяет пользователю вставлять новые строки данных в таблицы.

- UPDATE: Позволяет пользователю изменять существующие строки данных в таблицах.

- DELETE: Позволяет пользователю удалять строки данных из таблиц.

- EXECUTE: Позволяет пользователю выполнять хранимые процедуры и функции.

Роли в SQL Server используются для группировки пользователей и назначения привилегий на уровне роли, вместо назначения привилегий для каждого отдельного пользователя. Это позволяет более гибко управлять доступом пользователей к объектам базы данных и упрощает процесс назначения и отзыва привилегий.

Назначение привилегий в SQL Server позволяет определить, какие пользователи или роли имеют доступ к объектам базы данных и какие операции они могут выполнять с этими объектами. Через назначение привилегий можно обеспечить безопасность данных и контролировать доступ пользователей к информации в базе данных.

***Дистрибутивы СУБД Oracle. Установка СУБД Oracle 12с на Windows. Global Database Name и SID.***

СУБД Oracle имеет несколько дистрибутивов, которые могут различаться по функциональности и доступности функций. Некоторые из наиболее распространенных дистрибутивов Oracle включают:

1. Oracle Database Standard Edition (SE): Основной дистрибутив Oracle Database, который предлагает базовые функции управления базами данных.

2. Oracle Database Enterprise Edition (EE): Расширенный дистрибутив Oracle Database, который предлагает более широкий набор возможностей, включая дополнительные опции и инструменты для разработки и управления базами данных.

3. Oracle Database Express Edition (XE): Бесплатный дистрибутив Oracle Database, ограниченный в функциональности и размере базы данных, но обеспечивающий базовые возможности работы с Oracle.

Установка СУБД Oracle 12c на Windows включает следующие шаги:

1. Загрузите установочный файл Oracle Database 12c с официального сайта Oracle.

2. Запустите установку, следуя инструкциям мастера установки. Выберите тип установки (например, Enterprise Edition) и указывайте необходимые параметры, такие как расположение файлов базы данных, пароль для системного аккаунта и т. д.

3. Настройте необходимые компоненты, такие как список служб Oracle, параметры сети и другие параметры конфигурации.

4. После завершения установки будет создана новая база данных Oracle. Запустите Oracle Enterprise Manager или SQL\*Plus, чтобы подключиться к базе данных и начать работу.

Global Database Name (глобальное имя базы данных) и SID (System Identifier) - это два идентификатора, используемых для идентификации базы данных Oracle:

- Global Database Name представляет собой уникальное имя базы данных Oracle, которое состоит из имени базы данных (Database Name) и имени домена (Domain Name). Например, "mydb.example.com". Глобальное имя базы данных обычно используется для подключения к базе данных и идентификации в распределенной среде.

- SID (System Identifier) представляет собой уникальный идентификатор экземпляра базы данных Oracle. Он представляет собой короткое имя базы данных и используется для внутренней идентификации экземпляра базы данных. Например, "mydb".

Глобальное имя базы данных и SID можно узнать или настроить в файле конфигурации базы данных Oracle (обычно файле init.ora или spfile.ora), который находится в директории установки Oracle.

Важно отметить, что в более новых версиях Oracle, начиная с Oracle Database 12c, рекомендуется использовать альтернативный подход к настройке базы данных с использованием файла параметров конфигурации (pfile) или файла параметров сервера (spfile), вместо изменения напрямую файлов init.ora.

***Основные системные пользователи. Основные специальные привилегии. Роль DBA.***

В Oracle базах данных существуют несколько основных системных пользователей и специальных привилегий. Вот некоторые из них:

1. SYS: Системный пользователь SYS является основным административным пользователем базы данных. Он имеет полный доступ ко всем объектам и системным таблицам базы данных.

2. SYSTEM: Системный пользователь SYSTEM также является административным пользователем, но с более ограниченными привилегиями, чем у пользователя SYS. Он используется для управления и администрирования базы данных.

3. SYSDBA и SYSOPER: Это роли в базе данных, предоставляющие особые привилегии. Роль SYSDBA предоставляет полный административный доступ к базе данных, включая возможность изменения структуры самой базы данных. Роль SYSOPER предоставляет некоторые основные привилегии администрирования, но с более ограниченными возможностями, чем у роли SYSDBA.

4. ROLE DBA: Роль DBA (Database Administrator) представляет собой особую роль в базе данных, которая обычно назначается для пользователей, ответственных за администрирование базы данных. Роль DBA включает различные привилегии и права доступа, позволяющие выполнять широкий спектр задач администрирования базы данных.

Основные специальные привилегии включают:

- CREATE DATABASE: Позволяет пользователю создавать новые базы данных.

- ALTER DATABASE: Позволяет пользователю изменять настройки и параметры базы данных.

- BACKUP DATABASE: Позволяет пользователю выполнять резервное копирование базы данных.

- RESTORE DATABASE: Позволяет пользователю восстанавливать базу данных из резервной копии.

- GRANT ANY PRIVILEGE: Позволяет пользователю предоставлять любые привилегии другим пользователям.

- AUDIT ANY: Позволяет пользователю выполнять аудит операций в базе данных.

Это лишь некоторые из основных системных пользователей и специальных привилегий в Oracle. В зависимости от конкретной версии Oracle и конфигурации базы данных могут быть и другие системные пользователи и привилегии.

***Понятия базы данных и экземпляра базы данных.***

В терминологии баз данных, понятия базы данных (Database) и экземпляра базы данных (Database Instance) имеют различные значения.

1. База данных (Database):

База данных (Database) - это организованная и структурированная коллекция данных, которая хранится и управляется с использованием системы управления базами данных (СУБД). База данных представляет собой совокупность связанных данных, которые организованы в определенной структуре и хранятся на постоянных носителях (например, на жестком диске). Базы данных обеспечивают постоянное хранение, доступ и управление данными, а также поддержку различных операций, таких как запросы, вставка, обновление и удаление данных.

2. Экземпляр базы данных (Database Instance):

Экземпляр базы данных (Database Instance) - это запущенный процесс или совокупность процессов, которые управляют доступом к данным и выполнением операций над базой данных. Экземпляр базы данных представляет собой рабочую среду, в которой выполняются операции с базой данных. Он включает в себя память, структуры данных в памяти, процессы и службы, необходимые для работы с базой данных.

Когда база данных запускается, создается соответствующий экземпляр базы данных. Экземпляр базы данных управляет кэшированием данных, выполнением запросов, контролем транзакций, управлением памятью и другими аспектами функционирования базы данных. Один экземпляр базы данных может управлять одной или несколькими базами данных, в зависимости от конфигурации.

В общем случае, база данных (Database) представляет собой физическую структуру, содержащую данные, а экземпляр базы данных (Database Instance) - это программное обеспечение и процессы, работающие с этими данными.

***Запуск и останов экземпляра базы данных Oracle.***

Для запуска и остановки экземпляра базы данных Oracle можно использовать следующие команды и инструменты:

1. Запуск экземпляра базы данных Oracle:

- В командной строке операционной системы выполните команду `sqlplus / as sysdba` для подключения к экземпляру базы данных в режиме SYSDBA.

- После успешного подключения выполните команду `startup`, чтобы запустить экземпляр базы данных. Это включит память и процессы, необходимые для работы с базой данных.

2. Остановка экземпляра базы данных Oracle:

- В командной строке операционной системы выполните команду `sqlplus / as sysdba` для подключения к экземпляру базы данных в режиме SYSDBA.

- После успешного подключения выполните команду `shutdown`, чтобы остановить экземпляр базы данных. Будет выполнена контролируемая остановка базы данных, при которой процессы будут закрыты, а память освобождена.

Кроме того, для управления запуском и остановкой экземпляра базы данных Oracle можно использовать следующие инструменты:

- Oracle Enterprise Manager (EM): Графический инструмент для управления базами данных Oracle. Он предоставляет удобный интерфейс для запуска и остановки экземпляров базы данных, а также для выполнения других административных задач.

- Oracle Database Configuration Assistant (DBCA): Утилита командной строки для создания и управления базами данных Oracle. Она позволяет создать новый экземпляр базы данных или изменить конфигурацию существующего экземпляра, включая его запуск и остановку.

При запуске и остановке экземпляра базы данных Oracle обязательно учитывайте права доступа и административные привилегии, необходимые для выполнения этих операций.

***Словарь базы данных: назначение, применение, основные представления.***

***Мультиарендная архитектура Oracle Multitenant.***

Словарь базы данных (Database Dictionary), также известный как словарь данных (Data Dictionary) или метаданные (Metadata), представляет собой набор таблиц и представлений в базе данных Oracle, которые содержат информацию о самой базе данных, ее объектах и структуре.

Назначение словаря базы данных состоит в следующем:

1. Хранение метаданных: Словарь базы данных содержит информацию о всех объектах базы данных, таких как таблицы, представления, индексы, ограничения, схемы и другие. Он сохраняет метаданные, определяющие структуру и свойства объектов.

2. Поддержка целостности данных: Словарь базы данных обеспечивает проверку и поддержку целостности данных. Он хранит информацию о правилах целостности, ограничениях, ссылочной целостности и других атрибутах данных, чтобы гарантировать, что данные остаются согласованными и не нарушают целостность.

3. Управление доступом и безопасностью: Словарь базы данных используется для хранения информации о пользователях, ролях, привилегиях и других аспектах безопасности базы данных. Он определяет, какие пользователи имеют доступ к каким объектам и какие операции им разрешены.

Основные представления словаря базы данных в Oracle включают:

1. DBA\_\*: Эти представления содержат информацию о различных аспектах базы данных, включая объекты, сегменты, пользователей, роли и привилегии. Примеры: DBA\_TABLES, DBA\_VIEWS, DBA\_USERS.

2. ALL\_\*: Эти представления предоставляют информацию, видимую текущему пользователю, включая объекты, на которые у пользователя есть доступ. Примеры: ALL\_TABLES, ALL\_VIEWS, ALL\_USERS.

3. USER\_\*: Эти представления отображают информацию только для текущего пользователя. Примеры: USER\_TABLES, USER\_VIEWS, USER\_OBJECTS.

Теперь о мультиарендной архитектуре Oracle Multitenant (Multitenant Architecture). Это особенность введенная в Oracle Database 12c, которая позволяет создавать и управлять несколькими базами данных, называемыми "контейнерами" (containers), с использованием общего набора ресурсов, называемого "подставками" (pluggable databases, PDBs).

Мультиарендная архитектура Oracle Multitenant имеет следующие особенности:

1. Контейнерная база данных (CDB): Это главная база данных, котор

ая содержит системные объекты и общие ресурсы, а также управляет подставками. Контейнерная база данных представляет собой контейнер для подставок.

2. Подставки (PDB): Это изолированные базы данных, которые подключаются к контейнерной базе данных. Каждая подставка имеет свою собственную системную и пользовательскую схему и может содержать свои объекты данных.

3. Управление множеством баз данных: Мультиарендная архитектура упрощает управление несколькими базами данных, поскольку общие ресурсы и системные объекты находятся в контейнерной базе данных, а подставки могут быть добавлены, удалены или перемещены сравнительно легко.

Мультиарендная архитектура Oracle Multitenant обеспечивает более эффективное использование ресурсов, упрощает управление базами данных и позволяет создавать множество изолированных баз данных в рамках одной физической базы данных.

***Файлы экземпляра Oracle. Файл параметров, управляющие файлы, файлы паролей, файлы трассировки.***

В Oracle база данных, файлы экземпляра (instance files) представляют собой различные файлы, которые используются для работы и управления экземпляром базы данных. Вот некоторые основные файлы экземпляра Oracle:

1. Файл параметров (Parameter File):

- Файл параметров, также известный как файл инициализации (Initialization File) или файл SPFILE (Server Parameter File), содержит настройки и параметры конфигурации экземпляра базы данных Oracle. Он определяет такие вещи, как размеры памяти, пути к файлам данных, настройки сети и другие параметры.

- Файл параметров имеет расширение ".ora" или ".spfile" и может быть расположен на сервере базы данных или в специальном хранилище (например, ASM - Automatic Storage Management).

2. Управляющие файлы (Control Files):

- Управляющие файлы являются важной частью экземпляра базы данных Oracle. Они содержат метаданные и информацию о структуре базы данных, а также управляют операциями восстановления и открытия базы данных.

- В обычной конфигурации используется несколько управляющих файлов, обычно три. Эти файлы должны быть расположены на отдельных физических устройствах для обеспечения отказоустойчивости.

3. Файлы паролей (Password Files):

- Файлы паролей используются для аутентификации пользователей, имеющих привилегии SYSDBA или SYSOPER, для доступа к базе данных Oracle.

- Файлы паролей содержат зашифрованные пароли имен пользователей, а также определяют, какие пользователи имеют доступ к привилегиям SYSDBA и SYSOPER.

4. Файлы трассировки (Trace Files):

- Файлы трассировки содержат информацию о выполнении операций, ошибках и событиях, которые происходят в базе данных Oracle. Они полезны для анализа и решения проблем, включая ошибки и неожиданное поведение.

- Файлы трассировки создаются автоматически системой базы данных Oracle и могут содержать записи об анализе производительности, отладке запросов, ошибках SQL и других событиях.

Каждый из этих файлов играет важную роль в работе и управлении экземпляром базы данных Oracle, обеспечивая настройки, метаданные, аутентификацию и отладочную информацию для успешной работы базы данных.

***Файлы базы данных Oracle. Файлы данных, журналы, архивы.***

В Oracle база данных, файлы базы данных играют важную роль в хранении данных, журналов транзакций и архивирования. Вот основные типы файлов базы данных Oracle:

1. Файлы данных (Data Files):

- Файлы данных содержат фактические данные таблиц и индексов, хранящихся в базе данных Oracle. Это основные файлы, в которых хранятся данные пользователей.

- Файлы данных имеют расширение ".dbf" и обычно представлены в виде отдельных файлов на файловой системе или устройствах ASM (Automatic Storage Management). Они разделены на блоки, которые являются минимальными единицами хранения данных.

2. Файлы журналов (Redo Log Files):

- Файлы журналов, также известные как redo log files, используются для записи изменений, сделанных в базе данных Oracle в виде журнала транзакций (transaction log).

- Файлы журналов играют важную роль в обеспечении целостности данных и восстановлении базы данных в случае сбоев или сбоев системы. Они записывают изменения до момента фиксации (commit) транзакций, чтобы обеспечить возможность отката (rollback) и восстановления базы данных.

3. Файлы архивов (Archive Log Files):

- Файлы архивов содержат архивные копии файлов журналов (redo log files), которые были сохранены после их полной записи.

- Файлы архивов могут использоваться для восстановления базы данных в определенной точке времени или для восстановления после сбоев. Они играют важную роль в обеспечении доступности и надежности данных.

Файлы данных, журналов и архивов являются основными компонентами базы данных Oracle и важны для обеспечения сохранности данных, целостности и возможности восстановления. Они должны быть правильно управляемыми и резервными копиями для обеспечения надежности и безопасности базы данных.

***Абстрактная модель Oracle. Логическая структура внешней памяти.***

Абстрактная модель Oracle представляет собой концептуальное представление о том, как данные хранятся и организованы в базе данных Oracle. Эта модель предоставляет абстракцию над физической реализацией базы данных и определяет логическую структуру данных.

Основные компоненты абстрактной модели Oracle включают:

1. Таблицы (Tables):

- Таблицы являются основными объектами хранения данных в базе данных Oracle. Они представляют собой логические структуры, в которых данные организованы в виде строк и столбцов.

- Таблицы содержат наборы столбцов, которые определяют типы данных и ограничения для каждого поля. Каждая строка в таблице представляет отдельную запись с данными.

2. Индексы (Indexes):

- Индексы используются для ускорения выполнения запросов и обеспечения более эффективного доступа к данным. Они представляют собой структуры данных, которые содержат отображение значений столбцов таблицы на физические адреса блоков данных.

- Индексы позволяют быстро находить и извлекать данные на основе определенных критериев поиска, таких как значения столбцов или условия.

3. Кластеры (Clusters):

- Кластеры используются для группировки связанных данных из разных таблиц и хранения их физически близко друг к другу на диске. Они позволяют улучшить производительность запросов, которые часто объединяют данные из разных таблиц.

- Кластеры используют общую ключевую структуру для связывания данных и оптимизации выполнения запросов.

4. Представления (Views):

- Представления представляют собой виртуальные таблицы, которые основаны на данных из одной или нескольких таблиц. Они предоставляют удобный способ представления данных в определенном формате или с определенными условиями.

- Представления могут использоваться для упрощения выполнения запросов, обеспечения безопасности данных и улучшения производительности.

Логическая структура внешней памяти базы данных Oracle включает файлы данных, файлы журналов и другие файлы, которые хранятся на диске. Эти файлы организованы в специальные структуры, такие как управляющие файлы, файлы данных и файлы журналов, чтобы обеспечить эффективное управление данными и журналирование транзакций. Oracle также использует алгоритмы кэширования и буферизации данных, чтобы минимизировать обращение к диску и улучшить производительность системы.

***Абстрактная модель Oracle. Физическая структура внешней памяти.***

Физическая структура внешней памяти в базе данных Oracle определяет способ, в котором данные физически организованы и хранятся на диске. Она включает в себя следующие компоненты:

1. Управляющие файлы (Control Files):

- Управляющие файлы являются первичными файлами базы данных Oracle. Они содержат метаданные базы данных, такие как имена файлов данных и журналов, информацию о структуре базы данных, параметры конфигурации и другую важную информацию.

- Управляющие файлы несут ответственность за управление базой данных и обеспечивают целостность и надежность системы.

2. Файлы данных (Data Files):

- Файлы данных представляют собой физические файлы, в которых хранятся фактические данные таблиц и индексов базы данных Oracle.

- Файлы данных содержат блоки данных, которые являются минимальными единицами хранения информации. Они организованы в структуры, называемые сегментами, которые, в свою очередь, состоят из экстентов (групп блоков).

3. Файлы журналов (Redo Log Files):

- Файлы журналов (redo log files) используются для записи изменений, сделанных в базе данных Oracle, в виде журнала транзакций.

- Файлы журналов играют важную роль в обеспечении надежности и возможности восстановления базы данных. Они записывают изменения до фиксации транзакций, чтобы обеспечить откат и восстановление данных в случае сбоев.

4. Файлы контроля целостности (Control File Backup and Multiplexed Copies):

- Файлы контроля целостности содержат резервные копии управляющих файлов и могут использоваться для восстановления базы данных в случае повреждения или потери основных управляющих файлов.

5. Файлы параметров (Parameter Files):

- Файлы параметров содержат настройки и параметры конфигурации базы данных Oracle, такие как размеры кэшей, режимы работы, параметры сети и другие настройки.

Физическая структура внешней памяти базы данных Oracle определяет расположение и организацию файлов на диске. Правильное управление и настройка физической структуры позволяют достичь оптимальной производительности и надежности базы данных.

***Абстрактная модель Oracle. Структура SGA.***

Абстрактная модель Oracle включает в себя структуру System Global Area (SGA), которая является общей памятью, используемой базой данных Oracle. SGA содержит различные компоненты, которые обеспечивают управление и доступ к данным в базе данных. Основные компоненты SGA включают:

1. Буферный кэш (Buffer Cache):

- Буферный кэш представляет собой область памяти, в которой хранятся копии блоков данных из файлов данных. Когда Oracle читает данные из диска или записывает изменения, они сначала помещаются в буферный кэш, чтобы обеспечить более быстрый доступ к данным в памяти.

2. Shared Pool:

- Shared Pool представляет собой область памяти, которая содержит различные компоненты, используемые для обработки SQL-запросов и выполнения операций. В Shared Pool хранятся предварительно откомпилированные SQL-запросы, выполненные планы запросов, кэш словаря и другие структуры данных.

3. Large Pool:

- Large Pool представляет собой дополнительную область памяти, которая используется для обработки операций, таких как сортировка, параллельное выполнение и управление памятью для серверных процессов.

4. Java Pool:

- Java Pool предназначен для хранения Java-объектов и других ресурсов, связанных с выполнением Java-кода в базе данных Oracle.

5. Регион Shared Server (Shared Server Region):

- Регион Shared Server используется в конфигурациях с общим доступом к серверу (shared server), где несколько клиентов могут использовать один процесс сервера для выполнения своих запросов.

6. Регион Large Pool для загрузчика (Loader Large Pool Region):

- Регион Large Pool для загрузчика используется во время загрузки данных для управления памятью и выполнения операций загрузки данных.

Структура SGA является общей для всех пользователей и сеансов, работающих с базой данных Oracle. Она играет важную роль в обеспечении производительности и эффективности базы данных, управлении памятью и кэшировании данных, а также обработке SQL-запросов и выполнении операций. Корректная настройка и размер SGA влияют на производительность и производительность базы данных Oracle.

***Абстрактная модель Oracle. Серверные процессы Oracle.***

Абстрактная модель Oracle включает в себя различные серверные процессы, которые выполняют различные задачи и обеспечивают функционирование базы данных Oracle. Каждый серверный процесс выполняет свою роль и выполняет определенные функции. Вот некоторые из основных серверных процессов Oracle:

1. Процесс базы данных (Database Process):

- Процесс базы данных (DBWn) отвечает за запись изменений из буферного кэша на диск и управление файлами данных.

- Процесс базы данных (LGWR) отвечает за запись журналов транзакций на диск и обеспечивает надежность и восстановление базы данных.

- Процесс базы данных (CKPT) контролирует цикл контрольной точки, который определяет, когда данные должны быть записаны на диск.

2. Серверный процесс (Server Process):

- Серверные процессы (Pnnn) обслуживают клиентские запросы и выполняют операции базы данных в ответ на эти запросы.

- Каждый серверный процесс обслуживает одного или нескольких клиентов и выполняет различные задачи, такие как выполнение SQL-запросов, обработка транзакций и управление памятью.

3. Процесс архивации журнала (Archiver Process):

- Процесс архивации журнала (ARCH) отвечает за архивирование журналов транзакций и создание резервных копий, которые могут использоваться для восстановления базы данных в случае сбоя.

4. Процесс проверки целостности блоков (Block Integrity Process):

- Процесс проверки целостности блоков (DBWR) периодически сканирует блоки данных на наличие повреждений и выполняет ремонтные операции, если это необходимо.

5. Процесс анализа SQL (SQL Analyze Process):

- Процесс анализа SQL (CBO) выполняет анализ и оптимизацию SQL-запросов, чтобы определить оптимальные пути выполнения и планы запросов.

Это только некоторые из серверных процессов Oracle. Каждый процесс имеет свою роль и выполняет определенные функции, необходимые для эффективной и надежной работы базы данных Oracle. Эти процессы работают вместе, чтобы обеспечить функционирование и управление базой данных.

***Абстрактная модель Oracle. Фоновые процессы Oracle.***

Абстрактная модель Oracle включает в себя также различные фоновые процессы, которые выполняются в фоновом режиме и поддерживают нормальное функционирование базы данных Oracle. Эти процессы выполняют различные задачи и обеспечивают поддержку базы данных. Вот некоторые из основных фоновых процессов Oracle:

1. Процесс контроля и восстановления (System Monitor Process, SMON):

- Процесс SMON отвечает за контроль и восстановление базы данных Oracle. Он выполняет различные задачи, такие как восстановление после сбоев, очистка неиспользуемых данных, освобождение ресурсов и управление пространством таблиц.

2. Процесс сигнализации (Process Monitor Process, PMON):

- Процесс PMON отслеживает активность клиентских сессий и процессов. Он следит за отключенными или зависшими сессиями, освобождает заблокированные ресурсы и восстанавливает сессии после сбоев.

3. Процесс записи журналов в фоновом режиме (Log Writer Process, LGWR):

- Процесс LGWR записывает журналы транзакций на диск. Он обеспечивает надежность базы данных, сохраняя журналы транзакций для восстановления и обеспечивая целостность данных.

4. Процесс проверки контрольной точки (Checkpoint Process, CKPT):

- Процесс CKPT отслеживает цикл контрольной точки и выполняет запись изменений данных на диск. Он обеспечивает согласованность базы данных, синхронизируя данные в буферном кэше и на диске.

5. Процесс фоновой записи данных (Database Writer Process, DBWn):

- Процесс DBWn отвечает за запись данных из буферного кэша на диск. Он выполняет фоновую запись изменений данных на диск, чтобы обеспечить сохранность данных в случае сбоев.

6. Процесс архивации журнала (Archiver Process, ARCH):

- Процесс ARCH отвечает за архивирование журналов транзакций, записанных в журнальных файлах. Он создает резервные копии журналов для возможности восстановления базы данных.

Эти фоновые процессы работают незаметно для пользователей и выполняют задачи, которые обеспечивают надежность, целостность и эффективность работы базы данных Oracle.

***Процесс-слушатель Oracle и его основные параметры.***

Процесс-слушатель (Listener) Oracle является службой, которая прослушивает входящие сетевые соединения к базе данных Oracle и устанавливает связь между клиентскими приложениями и экземпляром базы данных. Он работает на сервере базы данных и обрабатывает запросы на подключение от клиентских приложений.

Основные параметры, связанные с процессом-слушателем Oracle, включают:

1. Имя слушателя (Listener Name): Это уникальное имя, присвоенное слушателю Oracle. Оно определяется во время установки базы данных и может быть изменено в дальнейшем.

2. Протокол (Protocol): Слушатель поддерживает различные протоколы для связи с клиентскими приложениями. Некоторые из распространенных протоколов включают TCP/IP, IPC (Interprocess Communication), и TCP/IP с использованием SSL (Secure Sockets Layer).

3. Порт (Port): Это номер порта, на котором слушатель прослушивает входящие сетевые соединения. Клиентские приложения должны использовать соответствующий порт для установки соединения с базой данных.

4. Программа-диспетчер (Dispatcher): Программа-диспетчер обрабатывает входящие сетевые запросы и распределяет их между доступными серверными процессами, которые выполняют операции базы данных. Можно настроить несколько программ-диспетчеров для балансировки нагрузки.

5. Сервис (Service): Сервис Oracle представляет собой идентификатор, который позволяет клиентским приложениям обращаться к конкретной базе данных или экземпляру. Каждый сервис имеет уникальное имя и связывается с определенным экземпляром базы данных.

6. Файл параметров слушателя (Listener Parameter File): Файл параметров слушателя содержит настройки и параметры, определяющие поведение слушателя. В этом файле можно указать имя слушателя, протоколы, порты и другие параметры конфигурации.

Это лишь некоторые из основных параметров, связанных с процессом-слушателем Oracle. Конфигурация слушателя определяется в файле параметров, который может быть изменен в соответствии с требованиями и настройками конкретной системы.

***Сетевые настройки Oracle. Установление соединения по сети.***

Для установления соединения с базой данных Oracle по сети необходимо правильно настроить сетевые параметры Oracle на клиентской и серверной сторонах. Вот основные шаги, которые нужно выполнить для установления соединения по сети:

1. Установите Oracle клиент: На клиентской машине, где расположено приложение, требуется установить Oracle клиент. Клиентская установка содержит необходимые компоненты и драйверы для подключения к базе данных Oracle.

2. Настройте файл tnsnames.ora: Файл tnsnames.ora содержит информацию о базах данных Oracle, к которым клиентские приложения могут подключаться. В этом файле указываются имя хоста, порт, имя сервиса и другие сетевые параметры для каждой базы данных. Убедитесь, что файл tnsnames.ora на клиентской машине содержит запись для нужной базы данных.

3. Проверьте доступность сервера: Убедитесь, что сервер базы данных Oracle доступен по сети. Убедитесь, что сервер запущен и слушатель Oracle работает на правильном порту.

4. Установите соединение с использованием имени сервиса: В клиентском приложении укажите имя сервиса из файла tnsnames.ora для установки соединения с базой данных Oracle. Клиентское приложение будет использовать информацию из файла tnsnames.ora для определения адреса сервера и других сетевых параметров.

5. Укажите учетные данные: При установке соединения по сети, клиентское приложение должно предоставить правильные учетные данные (имя пользователя и пароль) для аутентификации на сервере базы данных.

6. Проверьте соединение: После указания всех необходимых параметров и учетных данных, клиентское приложение должно попытаться установить соединение с сервером базы данных Oracle. Если соединение установлено успешно, приложение может начать взаимодействие с базой данных.

Установление соединения по сети в Oracle требует правильной настройки сетевых параметров на клиентской и серверной сторонах, а также правильных учетных данных для аутентификации. Важно убедиться, что все необходимые сетевые компоненты и драйверы установлены на клиентской машине и что сервер базы данных доступен по сети.

***Табличные пространства СУБД Oracle и их основные параметры.***

В базе данных Oracle табличные пространства (tablespaces) представляют собой логические контейнеры, используемые для хранения данных, индексов и других объектов базы данных. Каждая таблица и индекс в Oracle должны быть размещены в определенном табличном пространстве. Вот некоторые основные параметры, связанные с табличными пространствами в Oracle:

1. Имя табличного пространства (Tablespace Name): Каждое табличное пространство имеет уникальное имя, которое идентифицирует его в базе данных Oracle. Имя табличного пространства должно быть уникальным в пределах базы данных.

2. Тип табличного пространства (Tablespace Type): В Oracle существует несколько типов табличных пространств, таких как DATA, TEMP, UNDO и SYSTEM. Табличное пространство типа DATA используется для хранения таблиц и индексов, TEMP - для временных данных, UNDO - для хранения информации о транзакциях и откате, а SYSTEM - для системных объектов и метаданных.

3. Расположение табличного пространства (Tablespace Location): Табличное пространство может быть расположено в определенном каталоге или файловой системе на сервере базы данных. Расположение указывается во время создания табличного пространства.

4. Размер табличного пространства (Tablespace Size): Размер табличного пространства определяет доступное пространство для хранения данных в нем. Размер может быть указан в байтах, килобайтах, мегабайтах, гигабайтах или терабайтах.

5. Автоматическое увеличение размера (Autoextend): В Oracle можно настроить автоматическое увеличение размера табличного пространства при достижении его предельного значения. Это позволяет пространству автоматически расширяться при необходимости.

6. Опции хранения (Storage Options): Табличные пространства могут иметь различные опции хранения, такие как INITIAL, NEXT и MAXEXTENTS, которые определяют начальный размер, размер автоматического увеличения и максимальное количество расширений для объектов в табличном пространстве.

7. Ограничения (Constraints): В табличных пространствах можно установить ограничения на использование ресурсов, такие как максимальный размер, максимальное количество файлов данных и другие.

Табличные пространства в Oracle предоставляют гибкость и управляемость в хранении данных. Они позволяют разделить данные по логическим категориям и управлять доступом и хранением объектов базы данных.

***Роли и привилегии СУБД Oracle и их основные параметры.***

В базе данных Oracle роли и привилегии используются для управления доступом пользователей к различным объектам и операциям в базе данных. Роли представляют собой набор привилегий, которые могут быть назначены пользователям или другим ролям. Вот некоторые основные параметры, связанные с ролями и привилегиями в Oracle:

1. Роли (Roles): Роли в Oracle представляют собой наборы привилегий, которые могут быть назначены пользователям или другим ролям. Роли облегчают управление привилегиями, так как можно назначить роль одному или нескольким пользователям вместо назначения отдельных привилегий каждому пользователю. Примеры стандартных ролей в Oracle включают роль DBA (Database Administrator), роль CONNECT, роль RESOURCE и другие.

2. Привилегии (Privileges): Привилегии представляют собой права доступа к определенным объектам и операциям в базе данных Oracle. Привилегии могут быть назначены пользователям или ролям. Некоторые примеры привилегий включают SELECT, INSERT, UPDATE, DELETE для таблиц, CREATE, DROP, ALTER для объектов и другие. Привилегии могут быть назначены на уровне объектов или на уровне системы.

3. Системные привилегии (System Privileges): Системные привилегии предоставляют права доступа к системным операциям и функциональности базы данных Oracle. Некоторые примеры системных привилегий включают CREATE SESSION (разрешает пользователю подключаться к базе данных), CREATE TABLE (разрешает пользователю создавать таблицы) и другие. Системные привилегии могут быть назначены пользователям или ролям.

4. Объектные привилегии (Object Privileges): Объектные привилегии предоставляют права доступа к определенным объектам базы данных Oracle, таким как таблицы, представления, процедуры и другие. Некоторые примеры объектных привилегий включают SELECT, INSERT, UPDATE, DELETE для конкретной таблицы или представления. Объектные привилегии могут быть назначены пользователям или ролям на уровне конкретных объектов.

5. Наследование привилегий (Privilege Inheritance): В Oracle привилегии могут наследоваться от ролей. Это означает, что если роль имеет определенные привилегии, то пользователи, назначенные этой роли, также наследуют эти привилегии. Это упрощает управление привилегиями и обеспечивает единообразный доступ к данным и операциям.

6. Грант привилегий (Grant Privileges): Администратор базы данных Oracle может предоставлять или отзывать привилегии учетным записям пользователей или ролям с помощью оператора GRANT и REVOKE. Грантирование привилегий позволяет управлять доступом и контролировать операции, которые могут выполнять пользователи.

Роли и привилегии в Oracle играют важную роль в обеспечении безопасности и управлении доступом к данным и операциям в базе данных. Они позволяют администраторам управлять правами доступа пользователей и обеспечивать соответствие политикам безопасности базы данных.

***Пользователь СУБД Oracle и его основные параметры.***

Пользователь в СУБД Oracle представляет собой учетную запись, которая используется для подключения к базе данных и выполнения операций внутри нее. Вот некоторые основные параметры, связанные с пользователем в Oracle:

1. Имя пользователя (Username): Это уникальное имя, которое идентифицирует пользователя в базе данных Oracle. Имя пользователя может содержать буквы, цифры и специальные символы, но должно соответствовать определенным правилам, таким как длина и использование допустимых символов.

2. Пароль (Password): Пароль используется для аутентификации пользователя при подключении к базе данных. Пароль должен быть достаточно сложным и безопасным, чтобы предотвратить несанкционированный доступ к учетной записи пользователя.

3. Роли (Roles): Роли в Oracle могут быть назначены пользователю и определяют его набор привилегий. Роли позволяют управлять доступом пользователя к объектам и операциям в базе данных. Некоторые стандартные роли в Oracle включают роль DBA (Database Administrator), роль CONNECT и роль RESOURCE.

4. Профиль (Profile): Профиль пользователя определяет параметры ограничений и настроек для этого пользователя. Профиль может включать ограничения на максимальную продолжительность сеанса, максимальное количество попыток аутентификации, доступные ресурсы и другие параметры безопасности и ограничения.

5. Привилегии (Privileges): Привилегии определяют, какие операции и объекты пользователь может использовать в базе данных. Привилегии могут быть предоставлены пользователю непосредственно или через роли. Некоторые примеры привилегий включают SELECT, INSERT, UPDATE, DELETE для таблиц и другие операции над объектами базы данных.

6. Схема (Schema): Схема в Oracle представляет собой логическую структуру, в которой пользователь может создавать объекты, такие как таблицы, представления, процедуры и другие. Каждый пользователь имеет свою собственную схему, в которой он может управлять своими объектами.

7. Ограничения (Constraints): Ограничения определяют правила целостности данных для объектов, созданных пользователем. Они могут быть определены для таблицы, чтобы гарантировать, что данные в ней соответствуют определенным правилам или ограничениям.

Пользователь в Oracle имеет различные параметры и настройки, которые определяют его возможности и

ограничения при работе с базой данных. Администратор базы данных Oracle может управлять этими параметрами и назначать различные привилегии пользователям для обеспечения безопасности и эффективности работы с базой данных.

***Профиль безопасности СУБД Oracle и его основные параметры.***

В СУБД Oracle профиль безопасности (Security Profile) представляет собой набор параметров и ограничений, которые могут быть назначены пользователям или ролям. Профиль безопасности определяет правила и политики для управления доступом пользователей к базе данных. Вот некоторые основные параметры, связанные с профилем безопасности в Oracle:

1. Длительность сеанса (Session Duration): Определяет максимальную продолжительность сеанса пользователя в базе данных. Этот параметр контролирует, сколько времени пользователь может оставаться подключенным к базе данных, прежде чем его сеанс будет автоматически прерван.

2. Попытки аутентификации (Failed Login Attempts): Определяет максимальное количество неудачных попыток аутентификации пользователя. Если пользователь превышает это количество, его учетная запись может быть заблокирована для предотвращения возможности подбора пароля.

3. Срок действия пароля (Password Expiration): Определяет период, после которого пароль пользователя должен быть изменен. После истечения срока действия пароля пользователь будет вынужден изменить пароль при следующем входе в систему.

4. Параметры сложности пароля (Password Complexity): Устанавливает требования к сложности пароля пользователя. Это может включать минимальную длину пароля, использование различных типов символов (буквы верхнего и нижнего регистра, цифры, специальные символы) и другие политики, чтобы обеспечить безопасность паролей.

5. Ресурсы (Resources): Ограничивает доступ пользователя к определенным ресурсам базы данных, таким как максимальное количество сеансов, максимальное количество процессорного времени, доступный объем памяти и другие системные ресурсы. Это позволяет контролировать использование ресурсов пользователем и предотвращать возможные перегрузки.

6. Ограничения профиля (Profile Limits): Дополнительные ограничения могут быть назначены в профиле безопасности для определенных параметров, таких как максимальный размер временного пространства, максимальное количество параллельных запросов, максимальное количество операций ввода-вывода и т. д.

Профиль безопасности в Oracle предоставляет средства для управления безопасностью и контроля доступа пользователей к базе данных. Администратор базы данных может назначать различные профили безопасности пользователям и ролям

в зависимости от их требований и роли в системе.

***Таблица в СУБД Oracle и ее основные параметры. Типы данных базы данных. Ограничения целостности в таблицах.***

В СУБД Oracle таблица представляет собой основную структуру для хранения данных. Вот некоторые основные параметры таблицы в Oracle:

1. Имя таблицы (Table Name): Каждая таблица в Oracle имеет уникальное имя, которое идентифицирует ее в базе данных.

2. Столбцы таблицы (Table Columns): Таблица состоит из столбцов, которые определяют типы данных и хранят фактическую информацию. Каждый столбец имеет имя и определенный тип данных.

3. Типы данных (Data Types): Oracle предоставляет различные типы данных, которые могут быть использованы в столбцах таблицы, такие как NUMBER (число), VARCHAR2 (строка переменной длины), DATE (дата и время), и другие. Тип данных определяет характеристики значения, которое может быть сохранено в столбце.

4. Ограничения целостности (Integrity Constraints): Ограничения целостности определяют правила, которые данные должны соблюдать, чтобы обеспечить целостность базы данных. В Oracle можно определить различные типы ограничений, такие как PRIMARY KEY (первичный ключ), FOREIGN KEY (внешний ключ), UNIQUE (уникальное значение) и CHECK (проверка условия).

5. Индексы (Indexes): Индексы позволяют ускорить поиск и сортировку данных в таблице. Они создаются на одном или нескольких столбцах таблицы и помогают оптимизировать выполнение запросов, ускоряя доступ к данным.

6. Размер таблицы (Table Size): Размер таблицы отражает количество записей (строк) и объем занимаемого места на диске. Размер таблицы может быть определен как общий размер всех столбцов в таблице, включая индексы и другие объекты, связанные с таблицей.

Таблицы в Oracle играют ключевую роль в организации и хранении данных. Они предоставляют структуру для хранения информации, а также определяют ограничения целостности, типы данных и другие параметры для эффективной работы с данными в базе данных.

***Иерархические запросы в СУБД Oracle.***

В СУБД Oracle для работы с иерархическими данными можно использовать иерархические запросы с использованием специального оператора CONNECT BY. Иерархические запросы позволяют выполнять операции, такие как построение дерева, поиск родительских или дочерних элементов, агрегацию данных на разных уровнях и другие операции, связанные с иерархической структурой данных.

Основной оператор для работы с иерархическими запросами в Oracle - это оператор CONNECT BY. Он используется для определения связей между родительскими и дочерними элементами в иерархии. Оператор CONNECT BY обычно используется в комбинации с оператором START WITH, который указывает начальный узел или условие начала построения иерархии.

Вот пример использования оператора CONNECT BY для построения иерархии:

SELECT employee\_id, employee\_name, manager\_id

FROM employees

START WITH employee\_id = 1

CONNECT BY PRIOR employee\_id = manager\_id;

В этом примере мы выбираем информацию о сотрудниках из таблицы "employees". Оператор START WITH указывает, что мы начинаем сотрудника с employee\_id = 1. Оператор CONNECT BY указывает связь между родительским и дочерним элементом, где PRIOR employee\_id представляет родительский элемент, а manager\_id - дочерний элемент. Результатом будет иерархический список сотрудников, начиная с заданного сотрудника.

Иерархические запросы в Oracle позволяют эффективно работать с иерархическими данными, проводить анализ и построение деревьев, а также выполнять различные операции на разных уровнях иерархии.

***Временные таблицы СУБД Oracle.***

В СУБД Oracle временные таблицы предоставляют возможность создания таблиц, которые используются для временного хранения данных в рамках текущей сессии или транзакции. Временные таблицы обладают следующими особенностями:

1. Временный характер данных: Данные, сохраненные во временных таблицах, существуют только в течение времени выполнения сессии или транзакции. По завершении сессии или транзакции данные из временных таблиц автоматически удаляются.

2. Локальные и глобальные временные таблицы: В Oracle существуют два типа временных таблиц - локальные и глобальные. Локальные временные таблицы видны только в пределах текущей сессии, тогда как глобальные временные таблицы могут быть видны и использованы другими сессиями.

3. Уникальные идентификаторы имен: Имена временных таблиц автоматически генерируются и уникальны в рамках сессии или базы данных, чтобы избежать конфликтов имен.

4. Типы временных таблиц: В Oracle временные таблицы могут быть созданы как обычные таблицы с определенными столбцами и ограничениями, а также с использованием дополнительных опций, таких как ON COMMIT DELETE ROWS (удаление строк при завершении транзакции) или ON COMMIT PRESERVE ROWS (сохранение строк при завершении транзакции).

Временные таблицы в Oracle широко используются для временного хранения промежуточных результатов, временных данных или для выполнения сложных операций, таких как сортировка, объединение или агрегация данных. Они помогают повысить эффективность запросов и уменьшить использование ресурсов, так как данные в них существуют только во время выполнения определенных операций.

***Индексы базы данных СУБД Oracle. Виды и особенности применения индексов.***

В СУБД Oracle индексы представляют собой структуры данных, которые позволяют ускорить выполнение запросов, обеспечивая быстрый доступ к данным в таблицах. Индексы создаются на одном или нескольких столбцах таблицы и используются для ускорения операций поиска, сортировки и объединения данных. Они помогают уменьшить количество операций чтения данных с диска, улучшая производительность запросов.

В СУБД Oracle существуют различные типы индексов, включая:

1. Одностолбцовый индекс (Single-Column Index): Создается на одном столбце таблицы. Обеспечивает быстрый доступ к данным на основе значения этого столбца.

2. Многостолбцовый индекс (Composite Index): Создается на нескольких столбцах таблицы. Обеспечивает быстрый доступ к данным на основе комбинации значений этих столбцов.

3. Уникальный индекс (Unique Index): Гарантирует уникальность значений в индексированных столбцах. Используется для обеспечения целостности данных и ускорения операций поиска.

4. Полнотекстовый индекс (Full-Text Index): Предназначен для выполнения поиска по текстовым данным. Используется для быстрого выполнения полнотекстовых запросов.

5. Битмап-индекс (Bitmap Index): Используется для индексации столбцов с ограниченным числом уникальных значений. Позволяет эффективно выполнять операции объединения и пересечения.

6. Функциональный индекс (Function-Based Index): Создается на основе выражений или функций, применяемых к столбцам таблицы. Позволяет индексировать вычисляемые значения и ускоряет операции поиска.

Применение индексов в СУБД Oracle зависит от конкретных требований и характера данных. Индексы следует создавать на столбцах, которые часто используются в условиях поиска или сортировки данных. Однако, индексы также занимают дополнительное пространство на диске и могут замедлять операции модификации данных, такие как вставка, обновление и удаление. Поэтому необходимо балансировать использование индексов, учитывая потребности в производительности и объеме данных.

***Последовательность СУБД Oracle и ее параметры.***

В СУБД Oracle последовательность (sequence) представляет собой объект базы данных, который генерирует уникальные числовые значения в возрастающем или убывающем порядке. Последовательности широко используются для генерации первичных ключей или других уникальных идентификаторов в таблицах.

При создании последовательности в Oracle можно задать следующие параметры:

1. Название последовательности (Sequence name): Уникальное имя, по которому последовательность идентифицируется в базе данных.

2. Начальное значение (Start value): Начальное значение, с которого должна начинаться генерация последовательности. По умолчанию равно 1.

3. Приращение (Increment): Шаг, на который увеличивается (или уменьшается, если задано отрицательное значение) значение последовательности при каждой генерации. По умолчанию равно 1.

4. Минимальное и максимальное значение (Minvalue, Maxvalue): Определяют минимальное и максимальное значения, в пределах которых может генерироваться последовательность.

5. Цикличность (Cycle): Определяет, должна ли последовательность циклически повторяться после достижения максимального или минимального значения. Если цикличность включена, то после достижения максимального значения последовательность начинает снова с минимального значения.

6. Кэширование (Cache): Определяет количество предварительно вычисленных значений последовательности, которые будут кэшироваться в памяти для улучшения производительности.

7. Отключение автоматической генерации (NOCYCLE): Определяет, что последовательность не должна автоматически генерировать новые значения при вызове функции NEXTVAL, если было достигнуто максимальное значение и цикличность не включена.

Параметры последовательности могут быть определены при ее создании или изменены позже с помощью команды ALTER SEQUENCE.

Пример создания последовательности в Oracle:

CREATE SEQUENCE my\_sequence

START WITH 1

INCREMENT BY 1

MINVALUE 1

MAXVALUE 100

CYCLE

CACHE 10;

В этом примере создается последовательность с именем "my\_sequence", начинающаяся с 1 и увеличивающаяся на 1 при каждой генерации. Минимальное значение равно 1, максимальное значение равно 100. Последовательность циклическая, и кэшируется 10 значений.

***Кластер и его параметры.***

В контексте СУБД Oracle кластер (cluster) представляет собой объект базы данных, который позволяет объединять логически связанные строки из одной или нескольких таблиц и хранить их физически близко друг к другу на диске. Кластеры используются для улучшения производительности и уменьшения фрагментации данных.

При создании кластера в Oracle можно задать следующие параметры:

1. Название кластера (Cluster name): Уникальное имя, по которому кластер идентифицируется в базе данных.

2. Таблицы-члены (Cluster members): Список таблиц, которые являются членами кластера. Эти таблицы должны иметь один или несколько общих столбцов, по которым строки будут объединяться в кластер.

3. Метод кластеризации (Cluster key): Столбец или столбцы, по которым происходит кластеризация строк в кластере. Значения этих столбцов используются для определения физического расположения строк на диске.

4. Атрибуты кластера (Cluster attributes): Дополнительные параметры, определяющие поведение кластера, такие как физическое расположение данных на диске, порядок сортировки и другие.

5. Физическая структура кластера (Cluster storage structure): Определяет способ хранения данных кластера на диске, такой как использование сегментов, экстентов и блоков.

Параметры кластера могут быть определены при его создании или изменены позже с помощью команды ALTER CLUSTER.

Пример создания кластера в Oracle:

CREATE CLUSTER my\_cluster

(cluster\_key\_column NUMBER)

SIZE 100

HASHKEYS 10

HASH IS my\_hash\_function;

В этом примере создается кластер с именем "my\_cluster". Кластеризация строк происходит по столбцу "cluster\_key\_column", который имеет тип данных NUMBER. Размер кластера равен 100 блокам. Используется хеширование с 10 хеш-ключами, и указана пользовательская хеш-функция "my\_hash\_function".

Кластеры в Oracle позволяют улучшить производительность операций чтения и записи данных, особенно при обращении к связанным строкам из разных таблиц. Однако их использование требует тщательного планирования и анализа требований при проектировании базы данных.

***Представление в СУБД Oracle и его параметры.***

Представление (view) в СУБД Oracle представляет собой виртуальную таблицу, которая основана на результатах выполнения запроса к одной или нескольким таблицам. Представление не содержит фактических данных, а является логической структурой, которая предоставляет доступ к данным, сохраненным в базе данных.

При создании представления в Oracle можно задать следующие параметры:

1. Название представления (View name): Уникальное имя, по которому представление идентифицируется в базе данных.

2. Запрос (Query): SQL-запрос, который определяет данные, отображаемые представлением. Запрос может включать операторы SELECT, JOIN, WHERE и другие для фильтрации, объединения и преобразования данных.

3. Колонки представления (View columns): Список столбцов, которые будут отображаться в представлении. Это может быть полный набор столбцов из исходных таблиц или только выбранные столбцы.

4. Ограничения представления (View constraints): Ограничения, которые могут быть определены для представления, такие как ограничения уникальности или ограничения целостности.

5. Права доступа (Access privileges): Разрешения на чтение и запись данных в представление. Можно указать, кто имеет доступ к представлению и какие операции разрешены.

6. Зависимости (Dependencies): Представление может зависеть от других таблиц, представлений или объектов базы данных. Зависимости определяют, какие объекты должны быть доступны, чтобы представление могло быть успешно выполнено.

Пример создания представления в Oracle:

CREATE VIEW my\_view AS

SELECT column1, column2

FROM my\_table

WHERE condition;

***Материализованное представление и его параметры.***

Материализованное представление (materialized view) в СУБД Oracle представляет собой физически сохраненный результат выполнения запроса к одной или нескольким таблицам. В отличие от обычного представления, которое является логической структурой и выполняет запрос каждый раз при обращении к нему, материализованное представление хранит фактические данные и обновляется автоматически или по требованию.

При создании материализованного представления в Oracle можно задать следующие параметры:

1. Название представления (Materialized view name): Уникальное имя, по которому материализованное представление идентифицируется в базе данных.

2. Запрос (Query): SQL-запрос, который определяет данные, сохраняемые в материализованном представлении. Запрос может включать операторы SELECT, JOIN, WHERE и другие для фильтрации, объединения и преобразования данных.

3. Тип обновления (Refresh type): Определяет, как и когда материализованное представление будет обновляться. Варианты включают полное обновление, инкрементное обновление, обновление по расписанию или в реальном времени.

4. Расписание обновления (Refresh schedule): Если обновление материализованного представления выполняется по расписанию, можно указать расписание, включая время и частоту обновления.

5. Фильтры и условия (Filters and conditions): Можно определить фильтры и условия, которые ограничивают данные, сохраняемые в материализованном представлении. Это позволяет создавать материализованные представления только для определенных подмножеств данных.

6. Сегментация и хранение (Segmentation and storage): Можно указать, как данные будут сегментироваться и храниться для материализованного представления, включая параметры хранения, такие как размер блока, алгоритм сегментации и другие.

Пример создания материализованного представления в Oracle:

CREATE MATERIALIZED VIEW my\_materialized\_view

REFRESH FAST ON DEMAND

AS

SELECT column1, column2

FROM my\_table

WHERE condition;

В этом примере создается материализованное представление с именем "my\_materialized\_view". Оно будет обновляться быстрым обновлением при запросе, и его данные будут получены из таблицы "my\_table" с применением определенного условия.

Материализованные представления в Oracle предоставляют возможность сохранять и предварительно вычислять результаты сложных запросов, что может значительно улучшить производительность запросов к базе данных.

***Частные и публичные синонимы СУБД Oracle.***

В СУБД Oracle синонимы представляют собой альтернативные имена для объектов базы данных, таких как таблицы, представления, процедуры и другие. Они позволяют использовать более удобные или осмысленные имена для объектов, а также обеспечивают уровень абстракции и безопасности при доступе к объектам.

Существуют два типа синонимов в Oracle: частные (private) и публичные (public).

1. Частные синонимы (private synonyms): Частные синонимы принадлежат определенному пользователю и видны только этому пользователю. Они могут быть созданы и использованы только в контексте конкретного пользователя. Частные синонимы обычно используются для создания альтернативных имен внутри схемы пользователя.

Пример создания частного синонима:

CREATE SYNONYM my\_private\_synonym FOR my\_table;

В этом примере создается частный синоним с именем "my\_private\_synonym" для таблицы "my\_table".

2. Публичные синонимы (public synonyms): Публичные синонимы доступны для всех пользователей в базе данных. Они создаются и хранятся в общей области и могут быть использованы любым пользователем без необходимости явного указания имени схемы или владельца объекта. Публичные синонимы широко используются для предоставления удобных имен для общедоступных объектов базы данных.

Пример создания публичного синонима:

CREATE PUBLIC SYNONYM my\_public\_synonym FOR my\_table;

В этом примере создается публичный синоним с именем "my\_public\_synonym" для таблицы "my\_table".

Чтобы использовать синонимы, пользователи могут обращаться к объектам базы данных по их именам с использованием синонимов, вместо явного указания полных имен объектов. Это делает код более читабельным и удобным для использования.

Пример использования синонима в запросе:

SELECT \* FROM my\_private\_synonym;

В этом примере выполняется выборка данных из таблицы, на которую указывает частный синоним "my\_private\_synonym".

***Секционирование таблиц. Виды секционирования.***

Секционирование таблиц - это процесс разделения больших таблиц на более мелкие физические структуры, называемые разделами или разделами, с целью улучшения производительности запросов и обеспечения более эффективного управления данными. Каждый раздел содержит отдельное подмножество данных, которые логически связаны.

В СУБД Oracle существуют различные виды секционирования таблиц:

1. Диапазонное секционирование (Range Partitioning): Таблица разбивается на разделы на основе диапазона значений в определенном столбце. Каждый раздел содержит данные, удовлетворяющие определенному диапазону значений. Например, таблица с диапазонным секционированием по столбцу "дата" может иметь разделы для каждого месяца или года.

2. Списочное секционирование (List Partitioning): Таблица разбивается на разделы на основе конкретных значений в определенном столбце. Каждый раздел содержит данные, соответствующие определенному списку значений. Например, таблица с списочным секционированием по столбцу "регион" может иметь разделы для каждого региона (например, "Северный", "Южный", "Западный" и т. д.).

3. Хэш-секционирование (Hash Partitioning): Таблица разбивается на разделы на основе хэш-функции, применяемой к определенному столбцу. Хэш-функция распределяет строки данных по разделам, обеспечивая равномерное распределение данных. Хэш-секционирование обычно используется, когда требуется равномерное распределение данных по разделам для более равномерного распределения нагрузки.

4. Секционирование по списку диапазонов (Range-List Partitioning): Это комбинация диапазонного и списочного секционирования. Таблица разбивается на разделы на основе диапазона значений в одном столбце и затем каждый раздел дополнительно разбивается на подразделы на основе списка значений в другом столбце.

5. Секционирование по списку хэшей (Hash-List Partitioning): Это комбинация хэш-секционирования и списочного секционирования. Таблица разбивается на разделы на основе хэш-функции, применяемой

к определенному столбцу, а затем каждый раздел дополнительно разбивается на подразделы на основе списка значений в другом столбце.

Секционирование таблиц позволяет эффективно управлять большими объемами данных, улучшает производительность запросов и обеспечивает более эффективное использование ресурсов хранения. Кроме того, секционирование позволяет выполнять операции обслуживания данных, такие как удаление или архивирование, над отдельными разделами без необходимости обработки всей таблицы.

***Транзакции в СУБД Oracle. Виды транзакций. Понятие автономной транзакции.***

В СУБД Oracle транзакция - это логическая единица работы, состоящая из одной или нескольких операций базы данных. Транзакция представляет собой последовательность операций, которые должны быть выполнены либо полностью (коммит), либо отменены (откат), чтобы сохранить целостность данных.

Виды транзакций в СУБД Oracle:

1. Однотабличная транзакция (Single-table transaction): Это транзакция, которая взаимодействует только с одной таблицей. Все операции внутри транзакции выполняются на одной таблице.

2. Многотабличная транзакция (Multi-table transaction): Это транзакция, которая включает операции на нескольких таблицах. Все операции внутри транзакции выполняются на нескольких таблицах.

3. Распределенная транзакция (Distributed transaction): Это транзакция, которая взаимодействует с данными, находящимися в нескольких базах данных. Распределенные транзакции используются в средах с несколькими узлами баз данных или при работе с базами данных, находящимися на разных серверах.

Понятие автономной транзакции (Autonomous Transaction) относится к особому типу транзакций в СУБД Oracle. Автономная транзакция - это независимая транзакция, которая выполняется параллельно с основной транзакцией, но не зависит от ее коммита или отката. Автономные транзакции могут быть полезными в ситуациях, когда требуется выполнить операции базы данных, которые не должны быть отменены основной транзакцией.

Автономные транзакции вводятся с помощью использования принципа "SAVEPOINT" и специальных процедур или функций, таких как "AUTONOMOUS\_TRANSACTION\_PRAGMA" и "AUTONOMOUS\_TRANSACTION". Они позволяют выполнять операции базы данных, такие как запись в журнал, отправку уведомлений или выполнение служебных операций, независимо от основной транзакции. По завершении автономной транзакции изменения фиксируются автоматически без влияния на основную транзакцию.

***SQLite. Общая характеристика СУБД. Архитектура. Внутренние таблицы.***

SQLite - это компактная, встраиваемая СУБД, которая предоставляет реляционную модель данных. Она известна своей простотой в использовании, надежностью и хорошей производительностью. Особенностью SQLite является то, что она работает непосредственно с файлами на диске и не требует отдельного сервера или настроек. SQLite поддерживает большинство стандартных операторов SQL и предоставляет богатый набор функций для работы с данными.

Архитектура SQLite включает в себя следующие основные компоненты:

1. Парсер (Parser): Этот компонент анализирует SQL-запросы и преобразует их во внутреннюю структуру данных SQLite.

2. Оптимизатор (Optimizer): Оптимизатор выполняет анализ запроса и выбирает оптимальный план выполнения, чтобы достичь наилучшей производительности.

3. Компилятор (Compiler): Компилятор преобразует оптимизированный запрос в машинный код для выполнения операций базы данных.

4. Хранилище данных (Data Storage): Данные в SQLite хранятся в виде файлов на диске. Они организованы в виде таблиц, индексов и других структур данных.

Внутренние таблицы в SQLite - это системные таблицы, которые используются для управления метаданными и внутренними операциями SQLite. Некоторые из внутренних таблиц включают:

- sqlite\_master: Эта таблица содержит информацию о других таблицах в базе данных, таких как их имена, типы и SQL-определения.

- sqlite\_sequence: Эта таблица используется для автоматической генерации значений первичных ключей для таблиц, имеющих столбцы с автоинкрементом.

- sqlite\_stat1, sqlite\_stat3: Эти таблицы содержат статистическую информацию о таблицах и индексах, которая используется оптимизатором для выбора наилучшего плана выполнения запросов.

- sqlite\_temp\_master: Эта таблица содержит информацию о временных таблицах, созданных в рамках текущего сеанса.

Внутренние таблицы SQLite являются важной частью работы СУБД и используются для управления и оптимизации запросов, а также для хранения метаданных базы данных.

***SQLite. Таблицы. Классы хранения и аффинированность типов.***

В SQLite таблицы используются для хранения данных в реляционной структуре. Каждая таблица состоит из столбцов (полей) и строк (записей), где каждый столбец имеет определенный тип данных.

SQLite поддерживает следующие типы данных для столбцов таблиц:

1. INTEGER: Целочисленный тип данных, который может хранить целые числа.

2. REAL: Тип данных с плавающей запятой, который может хранить числа с плавающей точкой.

3. TEXT: Тип данных для хранения текстовых строк переменной длины.

4. BLOB: Тип данных для хранения двоичных данных, таких как изображения или файлы.

5. NULL: Тип данных для хранения пустых значений.

Кроме того, SQLite имеет концепцию классов хранения (storage classes), которые определяются не только типом данных столбца, но и его значением:

1. NULL: Значение NULL, которое указывает на отсутствие значения.

2. INTEGER: Целочисленное значение, которое может быть хранено в виде 1, 2, 3, и т.д.

3. REAL: Значение с плавающей точкой, которое может быть представлено в виде числа с десятичной точкой, например, 3.14.

4. TEXT: Строковое значение, представленное в виде текста в кавычках, например, "Hello, World!".

5. BLOB: Двоичное значение, представленное в виде последовательности байтов.

Важно отметить, что в SQLite нет строгой аффинированности типов данных. Это означает, что столбец одного типа данных может хранить значения другого типа без явного преобразования. SQLite будет автоматически выполнять неявные преобразования типов при необходимости. Например, строковое значение может быть сохранено в столбце с типом INTEGER, если это значение может быть корректно преобразовано в целое число.

Однако, при работе с SQLite рекомендуется следовать соответствующим типам данных для обеспечения целостности данных и корректного выполнения запросов.

***SQLite. Особенности сравнения, сортировки и группировки дат, строк и числовых данных.***

В SQLite сравнение, сортировка и группировка данных осуществляются с учетом некоторых особенностей для различных типов данных:

1. Даты и времена: SQLite поддерживает хранение и обработку дат и временных значений. При сравнении дат и временных значений SQLite использует стандартные сравнительные операторы (<, >, <=, >=). При сортировке дат и временных значений SQLite учитывает их хронологический порядок. Группировка дат и временных значений также выполняется с учетом их значения.

2. Строки: При сравнении строк SQLite использует лексикографическое сравнение, основанное на порядке символов в Unicode. Строки сравниваются посимвольно, начиная с первого символа. Если первые символы строк совпадают, сравниваются следующие символы, и так далее. При сортировке строк SQLite также использует лексикографическое сравнение. Группировка строк выполняется на основе их значений.

3. Числа: При сравнении чисел SQLite использует стандартные операторы сравнения (<, >, <=, >=). При сортировке чисел SQLite учитывает их числовой порядок. Группировка чисел выполняется на основе их значений.

Важно отметить, что SQLite не учитывает локализацию и региональные настройки при сравнении, сортировке и группировке данных. Это означает, что порядок сортировки может отличаться от ожидаемого, особенно при работе с мультиязычными данными. В таких случаях рекомендуется использовать специальные функции SQLite, такие как `COLLATE`, для настройки порядка сравнения строк в соответствии с требуемыми правилами и локализацией.

Также следует отметить, что SQLite поддерживает множество встроенных функций для обработки и преобразования данных, которые могут быть использованы для более сложных операций сравнения, сортировки и группировки.

***SQLite. Операторы объединения и соединения.***

В SQLite доступны операторы объединения (`UNION`, `UNION ALL`) и операторы соединения (`JOIN`, `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL JOIN`) для выполнения операций объединения и соединения данных из разных таблиц. Вот их общее описание:

Операторы объединения:

1. `UNION`: Объединяет результаты двух или более SELECT-запросов и удаляет дублирующиеся строки. Количества и типы столбцов во всех SELECT-запросах должны быть совместимыми.

2. `UNION ALL`: Объединяет результаты двух или более SELECT-запросов, включая все строки (в том числе дублирующиеся). Количества и типы столбцов во всех SELECT-запросах должны быть совместимыми.

Операторы соединения:

1. `JOIN` или `INNER JOIN`: Выполняет внутреннее соединение между двумя таблицами на основе условия соединения. Возвращаются только строки, для которых выполняется условие соединения.

2. `LEFT JOIN` или `LEFT OUTER JOIN`: Выполняет левое внешнее соединение между двумя таблицами. Возвращаются все строки из левой таблицы и соответствующие строки из правой таблицы на основе условия соединения.

3. `RIGHT JOIN` или `RIGHT OUTER JOIN`: Выполняет правое внешнее соединение между двумя таблицами. Возвращаются все строки из правой таблицы и соответствующие строки из левой таблицы на основе условия соединения.

4. `FULL JOIN` или `FULL OUTER JOIN`: Выполняет полное внешнее соединение между двумя таблицами. Возвращаются все строки из обеих таблиц, дополняя их недостающими значениями NULL, если нет соответствующих строк в другой таблице.

При использовании операторов соединения также указывается условие соединения, которое определяет, какие строки должны быть соединены. Условие соединения обычно указывается с помощью оператора сравнения (=, <>, <, >, <=, >=) или других логических операторов.

Примеры использования этих операторов можно найти в документации SQLite или во множестве онлайн-ресурсов, посвященных SQL и SQLite.

***SQLite. Использование обобщённых табличных выражений.***

Обобщенные табличные выражения (CTE, Common Table Expressions) в SQLite позволяют создавать временные результаты запросов, которые могут быть использованы внутри других запросов. Они представляют собой именованные временные таблицы, которые могут быть объявлены внутри запроса и использованы для упрощения и повторного использования кода.

Обобщенные табличные выражения могут быть особенно полезны в следующих случаях:

1. Рекурсивные запросы: SQLite поддерживает рекурсивные CTE, что позволяет выполнить итеративный запрос, который ссылается на самого себя. Это может быть полезно для работы с иерархическими данными или выполнения других сложных рекурсивных операций.

2. Повторное использование кода: CTE позволяют объявить именованные временные таблицы, которые можно использовать в разных частях запроса. Это помогает упростить и структурировать запросы, особенно в случае сложных операций или запросов с несколькими уровнями вложенности.

3. Улучшение читаемости запросов: Использование CTE может сделать запросы более понятными и легкими для чтения, особенно если в запросе присутствуют множественные операции объединения или сложные логические выражения.

Пример использования обобщенных табличных выражений в SQLite:

WITH

cte1 AS (

SELECT \* FROM table1 WHERE condition1

),

cte2 AS (

SELECT \* FROM table2 WHERE condition2

)

SELECT \* FROM cte1 JOIN cte2 ON cte1.id = cte2.id;

В этом примере мы создаем два обобщенных табличных выражения `cte1` и `cte2`, которые представляют результаты двух отдельных SELECT-запросов. Затем мы объединяем эти результаты в основном запросе.

Обобщенные табличные выражения могут быть мощным инструментом для работы с данными в SQLite, и они широко применяются в сложных запросах и аналитических операциях.

***SQLite. Ограничения целостности.***

SQLite поддерживает различные ограничения целостности для обеспечения правильности и согласованности данных. Вот некоторые из них:

1. Ограничение уникальности (UNIQUE): Это ограничение гарантирует, что значения в определенных столбцах таблицы являются уникальными. При попытке вставки или изменении данных, нарушающих уникальность, SQLite генерирует ошибку.

2. Ограничение первичного ключа (PRIMARY KEY): Это ограничение определяет один или несколько столбцов таблицы в качестве первичного ключа, который должен быть уникальным для каждой строки и не может быть пустым (NULL). SQLite автоматически создает индекс для первичного ключа, чтобы обеспечить эффективность поиска.

3. Ограничение внешнего ключа (FOREIGN KEY): Это ограничение определяет связь между двумя таблицами на основе значения в столбце. Оно гарантирует, что значения внешнего ключа соответствуют значениям первичного ключа в связанной таблице. SQLite поддерживает ограничение внешнего ключа, но его неактивно по умолчанию. Чтобы включить его, необходимо использовать опцию `FOREIGN\_KEYS` при открытии базы данных.

4. Ограничение NOT NULL: Это ограничение требует, чтобы значения в указанных столбцах не были пустыми (NULL).

5. Ограничение CHECK: Это ограничение определяет пользовательское условие, которое должно быть выполнено для каждой строки в таблице. Например, ограничение CHECK может проверять, чтобы числовое значение находилось в определенном диапазоне или чтобы строковое значение соответствовало определенному шаблону.

Ограничения целостности могут быть объявлены при создании таблицы с помощью оператора `CREATE TABLE` или изменены позднее с использованием оператора `ALTER TABLE`. SQLite автоматически проверяет эти ограничения при вставке, изменении или удалении данных в таблице и генерирует ошибку, если они нарушаются.

Важно отметить, что SQLite не поддерживает некоторые расширенные ограничения целостности, такие как отложенные ограничения или каскадное удаление/обновление. Однако, основные ограничения целостности могут быть эффективно использованы для поддержки целостности данных в SQLite.

***SQLite. Указания PRAGMA.***

В SQLite `PRAGMA` - это специальная команда, которая позволяет получить или установить различные параметры и настройки базы данных SQLite. Она используется для управления различными аспектами работы SQLite и предоставляет доступ к метаданным и настройкам базы данных. Вот некоторые распространенные указания `PRAGMA`:

1. PRAGMA table\_info(table\_name): Возвращает информацию о столбцах таблицы, такую как имя столбца, тип данных, ограничения целостности и т.д.

2. PRAGMA database\_list: Возвращает список баз данных, доступных в текущей сессии.

3. PRAGMA foreign\_keys: Возвращает текущее состояние ограничения внешнего ключа. Может быть использовано для проверки, включено ли ограничение внешнего ключа.

4. PRAGMA journal\_mode: Возвращает или изменяет режим журнала базы данных. Режим журнала определяет, как SQLite ведет журнал изменений данных и обеспечивает целостность базы данных.

5. PRAGMA cache\_size: Возвращает или изменяет размер кэша страниц базы данных. Это позволяет управлять количеством страниц, которые будут храниться в кэше для улучшения производительности.

6. PRAGMA page\_size: Возвращает или изменяет размер страницы базы данных. Размер страницы определяет минимальную единицу чтения/записи данных в базе данных.

7. PRAGMA integrity\_check: Проверяет целостность базы данных и возвращает сообщение о любых обнаруженных проблемах.

8. PRAGMA foreign\_key\_list(table\_name): Возвращает список внешних ключей, связанных с указанной таблицей.

9. PRAGMA user\_version: Возвращает или изменяет версию базы данных, которая может использоваться для отслеживания и управления версиями схемы базы данных.

10. PRAGMA synchronous: Возвращает или изменяет уровень синхронизации базы данных. Уровень синхронизации определяет, насколько SQLite ожидает подтверждения записи на диск после каждой операции записи.

Указания `PRAGMA` могут быть использованы с помощью SQL-запросов в SQLite для получения и установки различных настроек и параметров базы данных.

***SQLite. Виртуальные таблицы.***

В SQLite виртуальные таблицы (Virtual Tables) - это особый тип таблиц, который предоставляет доступ к данным, хранящимся во внешних источниках или созданным с использованием расширений SQLite. Виртуальные таблицы предоставляют интерфейс, позволяющий SQLite выполнять запросы, фильтровать данные и объединять их с другими таблицами, используя обычные SQL-запросы.

Виртуальные таблицы полезны в следующих сценариях:

1. Подключение к внешним источникам данных: SQLite может использовать виртуальные таблицы для подключения к различным внешним источникам данных, таким как файлы CSV, XML, JSON, а также к другим базам данных через расширения.

2. Полнотекстовый поиск: SQLite предоставляет виртуальные таблицы для выполнения полнотекстового поиска с использованием расширения FTS (Full-Text Search). Это позволяет эффективно выполнять поиск и анализ текстовых данных в SQLite.

3. Хранение временных данных: Виртуальные таблицы могут использоваться для хранения временных данных в памяти, таких как временные результаты запросов или промежуточные данные.

4. Расширения SQLite: Разработчики могут создавать свои собственные виртуальные таблицы, расширяя функциональность SQLite. Это позволяет добавлять пользовательские типы данных, операции и функции, которые можно использовать в SQL-запросах.

Для создания виртуальной таблицы в SQLite необходимо определить ее схему и поведение с использованием подходящего расширения или API. Например, для виртуальных таблиц, основанных на внешних источниках данных, можно использовать расширение SQLite Virtual Table (сокращенно FTS5), а для полнотекстового поиска - расширение FTS5.

После создания виртуальной таблицы она может быть использована в SQL-запросах так же, как и обычная таблица, позволяя выполнять операции выборки, вставки, обновления и удаления данных.

***SQLite. Индексы.***

В SQLite индексы используются для ускорения выполнения запросов, позволяя быстро находить и выбирать данные из таблиц. Индексы создаются на одном или нескольких столбцах таблицы и содержат отсортированные значения этих столбцов вместе с ссылками на соответствующие строки данных.

Основные особенности индексов в SQLite:

1. Уникальность: Индексы могут быть уникальными, что означает, что значения в индексе должны быть уникальными для каждой строки таблицы. Уникальные индексы могут использоваться для обеспечения уникальности значений в столбцах таблицы, таких как первичные ключи.

2. Множественные столбцы: В SQLite можно создавать индексы на нескольких столбцах, что позволяет эффективно искать данные по комбинации значений этих столбцов.

3. Типы индексов: SQLite поддерживает различные типы индексов, включая B-дерево индексы (B-Tree indexes) и хеш-индексы (Hash indexes). B-дерево индексы являются наиболее распространенным типом индексов в SQLite и обеспечивают эффективный поиск и сортировку данных.

4. Автоматическое использование индексов: Оптимизатор SQLite автоматически выбирает и использует индексы для оптимизации выполнения запросов. При наличии подходящего индекса SQLite может использовать его для быстрого поиска, сортировки или объединения данных.

5. Обновление индексов: При выполнении операций вставки, обновления или удаления данных SQLite автоматически обновляет соответствующие индексы для поддержания их актуальности. Это гарантирует, что индексы всегда отражают актуальное состояние данных в таблице.

6. Создание и удаление индексов: Индексы могут быть созданы или удалены с помощью операторов CREATE INDEX и DROP INDEX. Также можно указать, что столбец или столбцы являются индексными при создании таблицы с использованием оператора CREATE TABLE.

Использование индексов в SQLite может значительно повысить производительность запросов, особенно для таблиц с большим объемом данных или при выполнении операций поиска, сортировки и объединения. Однако следует помнить, что индексы занимают дополнительное место на диске и требуют некоторых ресурсов при обновлении данных, поэтому их следует создавать с учетом конкретных потребностей приложения.

***SQLite. Представления.***

В SQLite представления (Views) представляют собой виртуальные таблицы, которые основаны на результатах выполнения запроса. Представления позволяют абстрагироваться от сложных запросов и предоставляют удобный интерфейс для доступа к данным.

Основные особенности представлений в SQLite:

1. Виртуальные таблицы: Представления в SQLite являются виртуальными таблицами, что означает, что они не хранят данные физически, а предоставляют доступ к данным, хранящимся в других таблицах. При выполнении запроса к представлению SQLite выполняет соответствующий запрос, объединяя данные из различных таблиц.

2. Абстрагирование данных: Представления позволяют абстрагироваться от сложных запросов, объединений таблиц и фильтрации данных. Они могут содержать вычисляемые столбцы, агрегатные функции и другие операции, делая доступ к данным более удобным и понятным для пользователей.

3. Обновление данных: В SQLite представления могут быть обновляемыми или только для чтения. Обновляемые представления позволяют выполнить операции вставки, обновления и удаления данных через представление, причем изменения будут отражены в соответствующих базовых таблицах. Только для чтения представления предоставляют только операции чтения данных.

4. Создание и удаление представлений: Представления могут быть созданы с помощью оператора CREATE VIEW, где указывается имя представления и соответствующий запрос. Представление может быть удалено с помощью оператора DROP VIEW.

5. Использование представлений: После создания представления оно может быть использовано в запросах так же, как и обычная таблица. Представления могут быть включены в другие запросы, объединены с другими таблицами и использованы в операциях фильтрации и сортировки данных.

Представления в SQLite предоставляют гибкость и удобство в работе с данными. Они позволяют сделать запросы более понятными и упрощают доступ к сложным данным.

***SQLite. Триггеры.***

В SQLite триггеры (triggers) представляют собой специальные объекты, которые автоматически выполняются при определенных событиях, происходящих в базе данных, таких как вставка, обновление или удаление данных из таблицы. Триггеры позволяют автоматизировать выполнение дополнительных действий или проверок при изменении данных.

Основные особенности триггеров в SQLite:

1. Триггерные события: Триггеры в SQLite могут быть назначены для событий INSERT, UPDATE и DELETE. То есть, они могут быть запущены при вставке, обновлении или удалении данных из таблицы.

2. Предусловия: Триггеры могут иметь предусловия (trigger condition), которые определяют условия, при выполнении которых триггер будет запущен. Например, триггер может быть настроен на выполнение только при определенном значении в столбце или при выполнении определенного условия.

3. Типы триггеров: В SQLite поддерживаются два типа триггеров: BEFORE и AFTER. BEFORE-триггеры выполняются перед событием, а AFTER-триггеры выполняются после события. BEFORE-триггеры могут использоваться для проверок или изменений данных перед их фактическим изменением, а AFTER-триггеры могут использоваться для выполнения дополнительных действий после изменения данных.

4. Действия триггера: Внутри триггера можно выполнять различные действия, такие как вставка, обновление или удаление данных из других таблиц, изменение значений столбцов, выполнение проверок и дополнительных операций.

5. Создание и удаление триггеров: Триггеры могут быть созданы с помощью оператора CREATE TRIGGER, где указывается имя триггера, таблица, событие и код триггера. Триггеры могут быть удалены с помощью оператора DROP TRIGGER.

Триггеры в SQLite предоставляют механизм для автоматического выполнения дополнительных действий при изменении данных в таблице. Они могут быть использованы для проверок целостности данных, выполнения сложных операций или обновления связанных данных.

***SQLite. Транзакции и блокировки.***

В SQLite транзакции и блокировки играют важную роль при обработке и управлении изменениями данных в базе данных. Вот некоторая информация о транзакциях и блокировках в SQLite:

1. Транзакции: Транзакция в SQLite представляет собой последовательность операций, которые выполняются как единое целое. Транзакции обеспечивают целостность данных и позволяют группировать операции в логические единицы работы. В SQLite транзакции могут быть явно начаты с помощью команды BEGIN TRANSACTION и завершены командой COMMIT или отменены с помощью команды ROLLBACK.

2. ACID-свойства: SQLite обеспечивает ACID-свойства для транзакций. ACID означает атомарность (Atomicity), согласованность (Consistency), изолированность (Isolation) и устойчивость (Durability). Эти свойства гарантируют, что транзакции выполняются надежно и безопасно, обеспечивая целостность данных.

3. Уровни изоляции: SQLite поддерживает несколько уровней изоляции транзакций, таких как SERIALIZABLE, REPEATABLE READ, READ COMMITTED и READ UNCOMMITTED. Уровень изоляции определяет, какие изменения видны другим транзакциям и какие блокировки применяются к данным. По умолчанию используется уровень изоляции READ COMMITTED.

4. Блокировки: SQLite использует механизм блокировок для обеспечения параллельного доступа к данным и предотвращения конфликтов при одновременных изменениях. SQLite поддерживает различные типы блокировок, такие как шаровая блокировка (shared lock) и эксклюзивная блокировка (exclusive lock). Блокировки позволяют транзакциям получать доступ к данным в соответствии с уровнем изоляции и предотвращать конфликты записи/чтения.

5. Оптимизация блокировок: SQLite использует различные оптимизации блокировок для улучшения производительности. Например, SQLite использует блокировки на уровне страниц, а не на уровне таблиц, чтобы минимизировать блокировки и обеспечить более эффективную работу с данными.

6. Отложенная запись: SQLite использует отложенную запись (deferred writing), что означает, что изменения данных могут быть буферизованы и записаны на диск только при необходимости, что повышает производительность при выполнении транзакций.

В целом, транзакции и блокировки в SQLite играют важную роль при обработке изменений данных и обеспечении целостности. Они обеспечивают атомарность, согласованность, изолированность и устойчивость данных, а также предотвращают конфликты при параллельном доступе к данным.