

Программирование и безопасность баз данных web-приложений

1. Назовите уровни проектирования базы данных и поясните, какие задачи решаются на каждом уровне.

Три уровня архитектуры:

1. **Логический уровень** – Цель – преобразование концептуальной модели на основе выбранной модели данных в логическую модель, не зависящую от особенностей используемой в дальнейшем СУБД для физической реализации базы данных

2. **Концептуальный уровень (Conceptual Level)** – Цель – создание концептуальной модели данных исходя из представлений пользователей о предметной области

3. **Физический уровень** – уровень, описание конкретной реализации базы данных

2. Перечислите модели данных и кратко охарактеризуйте каждую модель.

Модель данных – это совокупность структур данных и операций их обработки. Рассмотрим три основных типа моделей данных: иерархическую, сетевую и реляционную.

Иерархическая модель представляет собой совокупность элементов, расположенных в порядке их подчинения от общего к частному и образующих перевернутое по структуре дерево (граф). К основным понятиям иерархической структуры относятся уровень, узел и связь. Узел – это совокупность атрибутов данных, описывающих некоторый объект.

Реляционная модель данных объекты и связи между ними представляет в виде таблиц, при этом связи тоже рассматриваются как объекты. Все строки, составляющие таблицу в реляционной базе данных, должны иметь первичный ключ. Каждая реляционная таблица представляет собой двумерный массив и обладает следующими свойствами:

1. Каждый элемент таблицы соответствует одному элементу данных.

2. Все столбцы в таблице однородные, т.е. все элементы в столбце имеют одинаковый тип и длину.

3. Каждый столбец имеет уникальное имя.

4. Одинаковые строки в таблице отсутствуют;

Сетевая модель данных — логическая модель данных, являющаяся

расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, у которой любой элемент мб связан с любым другим элементом. Разница между иерархической моделью данных и сетевой состоит в том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься любое число предков.

Сетевая БД состоит из набора экземпляров определенного типа записи и набора экземпляров определенного типа связей между этими записями.

3. Поясните процесс нормализации данных.

Нормализацией называется *обратимый* пошаговый процесс замены данной совокупности отношений другой схемой с устранением избыточных функциональных зависимостей, причем:

- ◆ не должны появляться ранее отсутствовавшие кортежи
- ◆ на отношениях новой схемы должно выполняться исходное множество функциональных зависимостей

Простой атрибут - атрибут, значения которого неделимы

Сложный атрибут получается соединением нескольких атомарных атрибутов, которые могут быть определены на одном или разных доменах
6 нормальных форм (+2) – приложение
1NF, 2NF, 3NF

4. Что такое правила Кодда?

Д-р Эдгар Ф. Кодд, после своего обширного исследования реляционной модели систем баз данных, разработал двенадцать собственных правил, которые, по его мнению, должны подчиняться базе данных, чтобы считаться истинной реляционной базой данных.

Эти правила могут применяться в любой системе баз данных, которая управляет сохраненными данными, используя только свои реляционные возможности. Это базовое правило, которое служит основой для всех остальных правил.

1. Правило информации.

Информация в реляционной базе данных должна быть представлена исключительно на логическом уровне и только в таблицах, все значения в таблице должны быть атомарными.

2. Правило гарантированного доступа.

Доступ к таблице должен осуществляться только с использованием имени таблицы, именем атрибута, ключа.

3. Поддержка недействительных значений.

Настоящая реляционная база данных должна позволять вводить неопределенное значение (null) и должны быть значения которые есть, но

пока нам не известны.

4. Правило динамического каталога, основанного на реляционной модели.

Доступ к описанию базы данных осуществляется по тем же правилам, что и к реально хранимым данным.

5. Правило исчерпывающего подъязыка данных.

Доступ к базе данных возможен только с использованием языка, имеющего линейный синтаксис, который поддерживает определение данных, манипулирование данными и операции управления транзакциями. Этот язык можно использовать напрямую или с помощью какого-либо приложения. Если база данных разрешает доступ к данным без помощи этого языка, то это считается нарушением.

6. Правило обновления представлений.

Все представления базы данных, которые теоретически могут быть обновлены, также должны обновляться системой.

7. Правило добавления, обновления и удаления.

Таблицы должны позволять добавлять, удалять и корректировать записи.

8. Правило физической независимости данных.

Данные, хранящиеся в базе данных, должны быть независимыми от приложений, которые обращаются к базе данных. Любые изменения в физической структуре базы данных не должны влиять на доступ к данным со стороны внешних приложений.

9. Правило логической независимости данных.

Логические данные в базе данных должны быть независимы от представления пользователя (приложения). Любые изменения в логических данных не должны влиять на приложения, использующие их. Например, если две таблицы объединены или одна разбита на две разные таблицы, это не должно повлиять или изменить пользовательское приложение. Это одно из самых сложных правил для применения.

10. Правило независимости условий целостности.

База данных должна быть независимой от приложения, которое ее использует. Все ограничения целостности могут быть независимо изменены без каких-либо изменений в приложении.

11. Правило независимости условий распространения.

Реляционная база данных не должна быть привязана к конкретным потребностям конкретного пользователя.

12. Правило единственности.

Работа с базой данных должна осуществляться на едином языке. И никакой язык низкого уровня не должен позволять обходить ограничения.

5. Поясните, что такое денормализация данных. В каких случаях следует использовать денормализацию?

Денормализация – намеренное приведение структуры базы данных в состояние, не соответствующее критериям нормализации, проводимое с

целью ускорения операций чтения из базы за счет добавления избыточных данных

Это не недоделанная нормализация, это намеренное нарушение нормальных форм, для увеличения производительности.

- ◆ Большое количество соединений таблиц.
- ◆ Расчетные значения.
- ◆ Длинные поля.

6. Перечислите основные модели приложений.

Приложение – база данных

Приложение – несколько баз данных

SOA

7. Что такое SOA?

SOA – Service-Oriented Architecture – Сервис-ориентированная архитектура

SOA — модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных заменяемых компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам

8. В чем состоят особенности баз данных веб-приложений?

Пользователи

Передача данных

Клиентские устройства

Безопасность

9. Какие стандартные API используются для доступа к данным?

Доступ к данным – прикладной программный интерфейс для СУБД

Набор функций:

- ◆ установление и закрытие соединения
- ◆ обновление данных
- ◆ передача запросов серверу
- ◆ получение результатов выполнения запросов
- ◆ получение кодов ошибок
- ◆ характеристики структуры набора результата

10. Что такое ADO.Net?

ADO.NET предоставляет собой технологию работы с данными, которая основана на платформе .NET Framework. Эта технология представляет нам набор классов, через которые мы можем отправлять запросы к базам данных, устанавливать подключения, получать ответ от базы данных и производить ряд других операций.

11. Опишите основные классы в ADO.Net

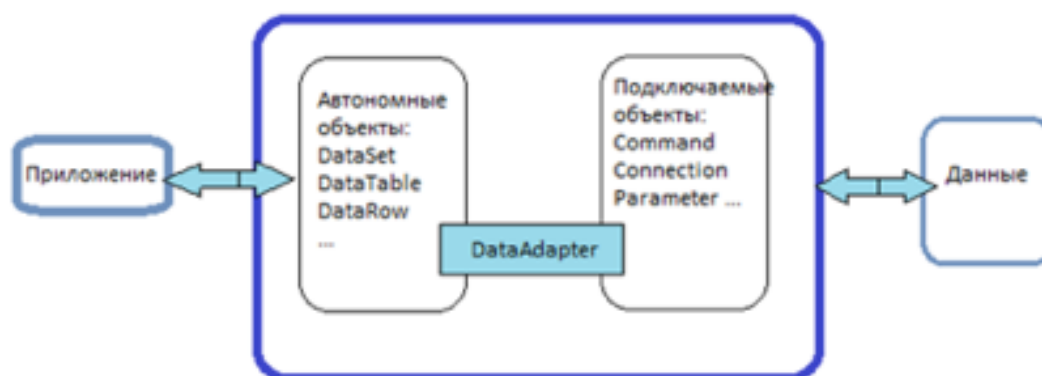
Тип объекта	Базовый класс	Соответствующие интерфейсы	Назначение
Connection	DbConnection	IDbConnection	Позволяет подключаться к хранилищу данных и отключаться от него. Кроме того, объекты подключения обеспечивают доступ к соответствующим объектам транзакций
Command	DbCommand	IDbCommand	Представляет SQL-запрос или хранимую процедуру. Кроме того, объекты команд предоставляют доступ к объекту чтения данных конкретного поставщика данных
DataReader	DbDataReader	IDataReader, IDataRecord	Предоставляет доступ к данным только для чтения в прямом направлении с помощью курсора на стороне сервера
DataAdapter	DbDataAdapter	IDataAdapter, IDbDataAdapter	Пересылает наборы данных из хранилища данных к вызывающему процессу и обратно. Адаптеры данных содержат подключение и набор из четырех внутренних объектов команд для выборки, вставки, изменения и удаления информации в хранилище данных
Parameter	DbParameter	IDataParameter, IDbDataParameter	Представляет именованный параметр в параметризованном запросе
Transaction	DbTransaction	IDbTransaction	Инкапсулирует транзакцию в базе данных

12. Какие в ADO.Net есть режимы работы с данными? Поясните каждый режим.

Режим работы с базами данных при отсутствии постоянного соединения с базой данных называют отсоединенным (disconnected). Соединенный режим представляет собой сильносвязанную среду, которая может содержать состояния и соединения.

При работе в отсоединенном режиме соединение осуществляется таким же образом, как и в соединенном режиме. Данные получают с помощью классов преобразования данных источников данных. В отличие от устройства считывания данных, которое связано соединением с определенной базой данных, набор данных не имеет связей ни с какой базой данных, даже с той, из которой были получены хранящиеся в нем данные.

- Постоянное подключение
- Отсоединенные данные



13. Что такое поставщик данных?

Поставщик данных (data provider) — это набор классов ADO.NET, которые позволяют получать доступ к определенной базе данных, выполнять команды SQL и извлекать данные. По сути, поставщик данных — это мост между вашим приложением и источником данных.

14. Перечислите известных вам поставщиков данных.

Тип объекта	Базовый класс	Соответствующие интерфейсы	Назначение
Connection	DbConnection	IDbConnection	Позволяет подключаться к хранилищу данных и отключаться от него. Кроме того, объекты подключения обеспечивают доступ к соответствующим объектам транзакций
Command	DbCommand	IDbCommand	Представляет SQL-запрос или хранимую процедуру. Кроме того, объекты команд предоставляют доступ к объекту чтения данных конкретного поставщика данных
DataReader	DbDataReader	IDataReader, IDataRecord	Предоставляет доступ к данным только для чтения в прямом направлении с помощью курсора на стороне сервера
DataAdapter	DbDataAdapter	IDataAdapter, IDbDataAdapter	Пересылает наборы данных из хранилища данных к вызывающему процессу и обратно. Адаптеры данных содержат подключение и набор из четырех внутренних объектов команд для выборки, вставки, изменения и удаления информации в хранилище данных
Parameter	DbParameter	IDataParameter, IDbDataParameter	Представляет именованный параметр в параметризованном запросе
Transaction	DbTransaction	IDbTransaction	Инкапсулирует транзакцию в базе данных

15. Что такое CLR?

CLR – Common Language Runtime – общезыковая исполняющая среда

- ◆ Поддерживаемые языки C# и VB
- ◆ Управляемая среда
- ◆ Домены приложений
- ◆ CLR развернута в SQL Server
- ◆ Поточковая обработка результирующих наборов – по одной строке
- ◆ Сложная логика
- ◆ Решение математических задач
- ◆ Подключение к удаленным службам
- ◆ Работа с удаленными хранилищами данных
- ◆ Управление внешними файлами

16. Как используется CLR в SQL Server?

Используя SQL CLR можно создавать написанные на высокопроизводительных языках свои хранимые процедуры, триггеры, пользовательские типы и функции, а также агрегаты.

- ◆ Процедуры
- ◆ Функции – скалярные и табличные
- ◆ Типы
- ◆ Триггеры

17. Как используется .Net-сборка в SQL Server?

Эта технология позволяет расширять функциональность SQL сервера с помощью .NET языков, например C# или VB.NET.

Сборки — это файлы DLL, используемые в экземпляре SQL Server для развертывания функции, хранимые процедуры, триггеры, определяемые пользователем статистические функции и определяемые пользователем типы, написанные на одном из языков управляемого кода, проводимых Microsoft .NET Framework

18. Что сборка может содержать?

Сборки имеют следующие составляющие:

- ◆ Манифест, который содержит метаданные сборки
- ◆ Метаданные типов. Используя эти метаданные, сборка определяет местоположение типов в файле приложения, а также места размещения их в памяти
- ◆ Собственно код приложения на языке MSIL, в который компилируется код C#
- ◆ Ресурсы

19. Опишите процедуру регистрации сборок в SQL Server.

```

--создание сборки
--drop assembly Student_Assembly;
create assembly Student_Assembly from 'c:\SQL_pack.dll';
go
-- создание процедуры сборки - название сборки - подраздел - название процедуры в проекте
create procedure AllStudents
as
external name Student_Assembly.StoredProcedures.AllStudents;
go
create procedure StudentsGrades
as
external name Student_Assembly.StoredProcedures.StudentsGrades;
go
create procedure StudentsPartTimeGrades @StartDate datetime, @EndDate datetime
as
external name Student_Assembly.StoredProcedures.StudentsPartTimeGrades;
go
-- вызов зарегистрированных процедур
EXEC AllStudents;
EXEC StudentsPartTimeGrades @StartDate = '2013-01-01', @EndDate = '2013-02-01';
EXEC StudentsGrades;

```

20. Какие методы обязательно должны быть реализованы для пользовательского типа?

21. Какие представления используются для просмотра сведений о сборках?

```

select * from sys.assemblies;
select * from sys.assembly_files;
select * from sys.assembly_modules;
select * from sys.module_assembly_usages;

```

Results							
	name	principal_id	assembly_id	cl_name	permission_set	permission_set_desc	is_visible
1	Microsoft.SqlServer.Types	4	1	microsoft.sqlserver.types, version=11.0.0.0, cult...	3	UNSAFE_ACCESS	1
2	Student_Assembly	1	65538	sql_pack, version=0.0.0.0, culture=neutral, publ...	1	SAFE_ACCESS	1

22. Что такое GIS? Перечислите известные вам GIS.

Геоинформационные системы (ГИС — географическая информационная система) — системы, предназначенные для сбора, хранения, анализа и графической визуализации пространственных данных и связанной с ними информации о представленных в ГИС объектах

- ◆ ESRI – ArcGIS
- ◆ MapInfo – MapInfo Pro
- ◆ Autodesk – AutoCAD Map 3D
- ◆ LizardTech – GeoExpress
- ◆ Кредо-Диалог – CREDO
- ◆ QGIS

23. Как обрабатываются пространственные данные в СУБД?

24. Что такое атрибутивная информация?

Атрибутивная информация - это информация, описывающая различные характеристики и параметры географической составляющей

25. Что такое топология пространственных данных?

Топология регулирует пространственные отношения связности и соседства векторных объектов (точек, линий и полигонов) в ГИС. Топологические данные полезны для обнаружения и исправления ошибок оцифровки (например, две линии дорог не сходятся на месте перекрестка). Корректная топология необходима для проведения некоторых типов пространственного анализа, таких как сетевой анализ.

26. Что такое SRID?

- SRID – Spatial Reference ID –
пространственная система привязки
координат
- Могут сравниваться только объекты в
одной SRID

```
-- Определение SRID
select geom,
       geom.ToString() as WKT,
       geom.STSrid as SRID
from Map.Belarus;

-- замена SRID
update Map.Belarus set geom.STSrid=4141;
```



	geom	WKT	SRID
1	0x661000000104348300000022EC896E583374034F5BA456DF...	POLYGON ((23.515222 53.956063, 23.515178 53.9558...	4326

27. Какое системное представление позволяет получить сведения о SRID?

Представление каталога - [sys.spatial_reference_systems](#)

28. Перечислите типы пространственных данных в MS SQL Server.

SQL Server поддерживает два пространственных типа данных: geometry и geography .

- ◆ Пространственный тип данных geometry представляет данные в евклидовой (плоской) системе координат.
- ◆ Пространственный тип данных geography представляет данные в системе координат для сферической Земли.

29. Приведите классификацию методов для работы с пространственными данными в MS SQL Server. Укажите несколько методов каждого типа.

Методы по стандарту OGC и дополнительные
Статические методы GEOMETRY::STGeomFromText()
Методы экземпляров класса @g.STContains()

30. Откуда и каким образом могут быть импортированы пространственные данные в SQL Server?

SqlSpatial
Shape2Sql
Open-source - Catfood.Shapefile

31. Можно ли построить индекс по столбцу пространственных данных в MS SQL Server? Если да, то каким образом?

Да

- ◆ CREATE SPATIAL INDEX
- ◆ GEOMETRY_GRID / GEOGRAPHY_GRID
- ◆ BOUNDING_BOX / GRIDS

32. Что такое тесселяция?

После декомпозиции индексированного пространства в сеточную иерархию пространственный индекс построено считывает данные из пространственного столбца. После считывания данных для пространственного объекта (или экземпляра) пространственный индекс выполняет *процесс тесселяции* для этого объекта. Процесс тесселяции помещает объект в сеточную иерархию, устанавливая связь между объектом и набором сеточных ячеек, с которыми он соприкасается (*контактные ячейки*). Начиная с уровня 1 сеточной иерархии, процесс тесселяции сначала обрабатывает уровень *в ширину* . В принципе процесс может продолжиться для всех четырех уровней, по одному уровню за раз.

Результатом процесса тесселяции является набор контактных ячеек,

которые записываются в пространственный индекс объекта. Ссылаясь на эти записанные ячейки, пространственный индекс может найти объект в пространстве по его расположению относительно других объектов в пространственном столбце, которые также хранятся в индексе.

33. Как можно хранить иерархические данные в СУБД?

В виде дерева(hierarchyid)

34. Поясните использование иерархического типа данных в MS SQL Server.

Иерархический тип данных

- hierarchyid
- системный тип данных переменной длины
- используется для представления положения в иерархии
- путь логически представлен в виде последовательности меток всех посещенных дочерних узлов, начиная с корня

Сначала создаётся корень иерархии(hierarchyid::GetRoot()), далее к корневой записи добавляются потомки(hierarchyid::GetRoot().GetDescendant())

35. Какие методы можно использовать для данных столбца иерархического типа?

Иерархический тип данных

- GetRoot
- GetLevel
- GetDescendant (child1, child2)
- GetAncestor (n)
- IsDescendantOf
- GetReparentedValue (oldRoot, newRoot)
- Parse
- ToString

GetDescendant (child1, child2)

- Если родитель — NULL, возвращает NULL
- Если родитель — не NULL, а потомки child1 и child2 — NULL, возвращает одного потомка
- Если родитель и потомок child1 — не NULL, а child2 NULL, возвращает потомка, который больше чем child1, и наоборот
- Если родитель, child1 и child2 не NULL, возвращает дочерний узел, больше child1 и меньше child2
- Если child1 (child2) не NULL и не является дочерним узлом, то возникает исключение
- Если child1 >= child2, то возникает исключение

36. Можно ли построить индекс по столбцу иерархического типа данных? Если да, то каким образом?

Есть два подхода к индексированию иерархических данных:

В глубину

В индексе преимущественно в глубину строки поддерева хранятся рядом друг с другом. В индексе преимущественно в глубину все узлы поддерева узла хранятся вместе. Поэтому индекс преимущественно в глубину эффективен для обработки запросов по поддеревьям.

В ширину

Если используется индексирование в ширину, строки одного

уровня иерархии хранятся вместе. В индексе преимущественно в ширину все прямые потомки узла хранятся в одном месте. Поэтому индекс преимущественно в ширину эффективен для запросов по прямым потомкам.

Для организации данных в ширину можно использовать метод GetLevel().

```
CREATE INDEX Breadth ON Employee (node, level);
```

37. Как обрабатываются иерархические запросы в СУБД Oracle?



- Start with – начальный узел иерархии
- Connect by – связь текущего с родительским
- Порядок прохода записей – от корня обход в глубину

```
SELECT EMPNO,  
       ENAME,  
       MGR  
FROM EMP  
START WITH MGR IS NULL  
CONNECT BY prior EMPNO = MGR
```

38. Как проходит обход иерархии в иерархических запросах в СУБД Oracle?

Порядок прохода записей – от корня обход в глубину

синтаксис

Select Column1, Column2, ... From <Table1>, <Table2>, ... Where <Condition3>

Connect By <Condition2> Start With <Condition1>

1. Сначала, команда получит все строки в таблице, соответствующие условиям в пункте start with (condition1), как корень дерева(корень или уровень 1).
2. После, для каждого корня требуется скан всей таблицы, чтобы получить следующие записи соответствующие условиям в пункте connect by (условие2) (node уровня 2), для каждого node уровня 2 сканируется все таблица, чтобы получить следующие записи соответствующие условиям в пункте connect by (node уровня 3), и

так продолжается до тех пор, пока не останется больше записей соответствующих условиям connect by то предыдущий node будет уровнем листа в дереве.

3. Под конец, проверяем условия пункта where (уровень3), чтобы получить записи пункта "select tree".

39. Что обозначают конструкции иерархического запроса PRIOR и CONNECT BY?

```
----- 8 -- CONNECT_BY_ROOT PRIOR
SELECT EMPNO,
       ENAME,
       LPAD(' ', 3*LEVEL)||ENAME AS Employee_Name,
       PRIOR ENAME PARENT_NODE,
       CONNECT_BY_ISLEAF IS_LEAF,
       CONNECT_BY_ROOT ENAME ROOT_NODE
FROM EMP
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO = MGR;
```

- PRIOR – родительский узел
- CONNECT_BY_ROOT – корневой узел иерархии

EMPNO	ENAME	EMPLOYEE_NAME	PARENT_NODE	IS_LEAF	ROOT_N
1	7839 KING	KING	(null)	0	KING
2	7566 JONES	JONES	KING	0	KING
3	7788 SCOTT	SCOTT	JONES	0	KING
4	7876 ADAMS	ADAMS SCOTT		1	KING
5	7902 FORD	FORD JONES		0	KING
6	7369 SMITH	SMITH FORD		1	KING

40. Что обозначают конструкции иерархического запроса ORDER SIBLINGS, SYS_CONNECT_BY_PATH?

```
SELECT SYS_CONNECT_BY_PATH(ENAME, '/') as Path
FROM EMP
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO = MGR;
```

- SYS_CONNECT_BY_PATH – материализованный путь в иерархии

```
----- 5 -- ORDER SIBLINGS BY
SELECT LPAD(' ', 3*LEVEL)||ENAME AS Employee_Name
FROM EMP
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO = MGR
ORDER SIBLINGS BY ENAME;
```

- ORDER SIBLINGS – упорядочение в рамках уровня

EMPLOYEE_NAME
KING
BLAKE
ALLEN
JAMES
MARTIN
TURNER
WARD
CLARK
MILLER

41. Что обозначают конструкции иерархического запроса CONNECT_BY_ISLEAF, CONNECT_BY_ROOT?

```

SELECT EMPNO,
       ENAME,
       LPAD(' ', 3*LEVEL) || ENAME AS Employee_Name,
       CONNECT_BY_ISLEAF IS_LEAF
FROM EMP
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO = MGR;

```

- CONNECT_BY_ISLEAF – является ли узел листовым
- CONNECT_BY_ROOT – корневой узел иерархии

42. Что обозначают конструкции иерархического запроса CONNECT_BY_LOOP, CONNECT_BY_NOCYCLE?

```

-- 1
SELECT lang_code,
       next_lang
FROM lngcode
START WITH lang_code = 'RUS'
CONNECT BY PRIOR lang_code = next_lang;

```

- CONNECT_BY_LOOP – зацикливание

```

SELECT lang_code,
       next_lang
FROM lngcode
START WITH lang_code = 'RUS'
CONNECT BY NOCYCLE
PRIOR lang_code = next_lang;

```

- CONNECT BY NOCYCLE – соединение до зацикливания

43. Что обозначают конструкции иерархического запроса LEVEL, CONNECT_BY_ISCYCLE?

```

SELECT  LEVEL,
        EMPNO,
        ENAME,
        MGR
FROM EMP
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO = MGR;

```

- LEVEL – уровень иерархии

```

SELECT  lang_code,
        next_lang,
        CONNECT_BY_ISCYCLE ISCYCLE
FROM lngcode
START WITH lang_code = 'RUS'
CONNECT BY NOCYCLE
PRIOR lang_code = next_lang;

```

- CONNECT_BY_ISCYCLE – 1, если вызывает заикливание, 0 – в остальных случаях

44. Что такое CTE? Для чего используется?

Common Table Expressions– это технология, которая представляет собой одну из форм повторного использования результатов одного SQL запроса в другом.

CTE играет роль представления, которое создается в рамках одного запроса и, не сохраняется как объект схемы.

- ◆ Создания рекурсивных запросов
- ◆ Многократных ссылок на результирующую таблицу из одной и той же инструкции
- ◆ Замены представлений
- ◆ Группирования по столбцу, производного от скалярного подзапроса выборки или функции, которая недетерминирована

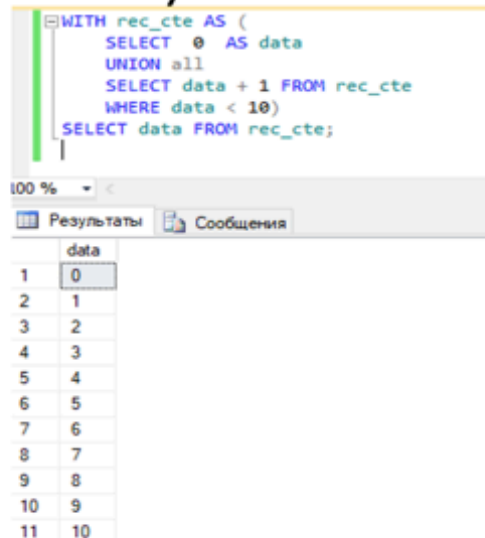
+плюсы:

- ◆ Позволяет значительно повысить читаемость и упростить работу со сложными запросами
- ◆ Можно составлять промежуточные CTE
- ◆ Могут быть определены в пользовательских подпрограммах

45. Что такое рекурсивные CTE?

- Рекурсивность – обращение через CTE к самому себе

```
WITH rec_cte AS (
    SELECT 0 AS data
    UNION all
    SELECT data + 1 FROM rec_cte
    WHERE data < 10)
SELECT data FROM rec_cte;
```



	data
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10

От обычного CTE-запроса рекурсивный отличается только рекурсивной частью, которая вводится предложением UNION ALL. Обратите внимание, что в рекурсивной части присутствует ссылка на имя CTE, т.е. внутри CTE ссылается само на себя. Это, собственно, и есть рекурсия. Естественно, анкорная и рекурсивная части должны иметь одинаковый набор столбцов.

46. Повышают ли CTE производительность запроса?

НЕТ.

CTE позволяет увеличить эффективность извлечения данных. Суть в том, что обычно для определения какие строки требуются для отображения определенной страницы, нам нужны ключевое поле и поле, по которому выполняется сортировка, которое даже не всегда нужно извлекать. А для генерации страницы обычно нужно большее количество столбцов, но небольшое количество строк. Выигрыш происходит за счет того, что для определения строк определенной страницы мы используем маленький и быстрый некластерный индекс, а для извлечения строк одной страницы – кластерный индекс но с небольшим количеством строк.

47. Как обрабатываются графы в MS SQL Server?

С SQL Server 2017

Можно создать один граф на базу данных

Граф состоит из таблиц узлов и ребер

База данных графа представляет собой коллекцию узлов (или вершин) и ребер (или связей). Узел представляет сущность (например, пользователя или организацию), а ребро — связь между двумя узлами, которые оно соединяет (например, отметки "Нравится" или друзья). С этими узлами и краями могут быть связаны свойства. Ниже приведены некоторые функции, благодаря которым графовая база данных является уникальной.

- ◆ Ребра или связи в графовой базе данных являются сущностями первого класса, с которым могут быть связаны атрибуты или свойства.
- ◆ Одно ребро может гибко соединить несколько узлов в графовой базе данных.
- ◆ Вы можете легко выразить запросы на сопоставление шаблонов и навигацию со множеством переходов.
- ◆ Вы можно легко выразить транзитивное замыкание и полиморфные запросы.

```

--- table creation
CREATE TABLE dbo.Club(
    ID int IDENTITY(1,1) NOT NULL,
    name varchar(100) NULL)
AS NODE;

CREATE TABLE dbo.playedFor
AS EDGE;

CREATE TABLE dbo.Player(
    ID int IDENTITY(1,1) NOT NULL,
    name varchar(100) NULL)
AS NODE;

--- insert into nodes

INSERT INTO Club(name)
VALUES('Tottenham Hotspur'), ('Chelsea'), ('Manchester City'), ('Arsenal'), ('West Ham United');
GO
INSERT INTO Player(name)
VALUES('Frank Lampard'), ('Petr Cech'), ('Cesc Fabregas'), ('Gael Clichy'), ('William Gallas');
GO

--- insert into edges

-----Frank Lampard
INSERT playedFor ($to_id,$from_id)
VALUES
    ((SELECT $node_id FROM dbo.Club WHERE ID = 1), (SELECT $node_id FROM dbo.Player WHERE ID = 1)),
    ((SELECT $node_id FROM dbo.Club WHERE ID = 2), (SELECT $node_id FROM dbo.Player WHERE ID = 1)),
    ((SELECT $node_id FROM dbo.Club WHERE ID = 5), (SELECT $node_id FROM dbo.Player WHERE ID = 1))

-----William Gallas
INSERT playedFor ($to_id,$from_id)
VALUES
    ((SELECT $node_id FROM dbo.Club WHERE ID = 1), (SELECT $node_id FROM dbo.Player WHERE ID = 5)),
    ((SELECT $node_id FROM dbo.Club WHERE ID = 3), (SELECT $node_id FROM dbo.Player WHERE ID = 5)),
    ((SELECT $node_id FROM dbo.Club WHERE ID = 2), (SELECT $node_id FROM dbo.Player WHERE ID = 5))

```

48. Укажите виды и особенности создания таблиц в графовых базах данных в MS SQL Server.

ТАБЛИЦЫ УЗЛА

таблицы создаются аналогично обычным реляционным за исключением указания типа – **AS NODE** (т.е. узел).

При создании таблицы узлов, помимо пользовательских, автоматически создаются еще два псевдостолбца – `graph_id` и `$node_id`.

Столбец `$node_id` является уникальным идентификатором узла, представленным в формате **JSON**.

```
CREATE TABLE utqG (  
    q_id INT PRIMARY KEY,  
    q_name VARCHAR(35) NOT NULL,  
) AS NODE;
```

ТАБЛИЦЫ РЕБРА

Она отличается от таблиц узлов типом – теперь это **EDGE**, а не **NODE**

к и для узлов, автоматически были созданы несколько псевдостолбцов, среди которых доступными пользователю являются:

- `$edge_id` – идентификатор ребра, формируется автоматически;
- `$from_id` – идентификатор узла, откуда исходит ребро;
- `$to_id` – идентификатор узла, куда входит ребро.

```
CREATE TABLE utbG (  
    b_datetime datetime NOT NULL,  
    b_vol tinyint NOT NULL,  
) AS EDGE;
```

49. Укажите особенности добавления ребер и узлов в графовых базах данных в MS SQL Server

В узлы добавляются данные как обычно, например:.

```
insert into Person(id, name) values (1, 'John');           // 1 - id, John -  
имя
```

В рёбра записываются данные следующим образом:

```
insert название_таблицы_рёбер ($to_id, $from_id)  
values  
((select $node_id FROM Person where ID = 1), select select $node_id  
FROM Person where ID = 2)); , где  
$to_id указывает направление ребра  
$from_id указывает, от какого узла идёт ребро  
$node_id - id узла
```

50. Перечислите известные вам методы выборки в графовых базах данных в MS SQL Server.

1. обычный select из таблицы узлов или рёбер (select * from Person)
2. select с условием MATCH

-- Найти людей, которым нравятся рестораны, расположенные в том же городе, где они живут

```
SELECT Person.name
FROM Person, likes, Restaurant, livesIn, City, locatedIn
WHERE MATCH (Person-(likes)->Restaurant-(locatedIn)->City
AND Person-(livesIn)->City);
```

Здесь Person, Restaurant, Citi - узлы,
likes, locatedIn, livesIn - рёбра с визуализацией

51. Что такое валидные и правильно построенные XML документы?

Корректные XML-документы (well-formed) — документы, полностью соответствующие правилам оформления XML. Корректность проверяется XML-парсером.

Валидные XML-документы (valid) — корректные XML-документы, которые соответствуют заранее определенному набору правил. Валидность проверяется валидатором. Т.е. проверка идет на основе какого-то шаблона, образца.

Правильно построенный (Well-formed). Правильно построенный документ соответствует всем общим правилам синтаксиса XML, применимым к любому XML-документу. И если, например, начальный тег не имеет соответствующего ему конечного тега, то это неправильно построенный документ XML. Документ, который неправильно построен, не может считаться документом XML; XML-процессор (парсер) не должен обрабатывать его обычным образом и обязан классифицировать ситуацию как фатальная ошибка.

Действительный (Valid). Действительный документ дополнительно соответствует некоторым семантическим правилам. Это более строгая дополнительная проверка корректности документа на соответствие заранее определённым, но уже внешним правилам, в целях минимизации количества ошибок, например, структуры и состава данного, конкретного документа или семейства документов. Эти правила могут быть разработаны как самим пользователем, так и сторонними разработчиками, например, разработчиками словарей или стандартов обмена данными. Обычно такие правила хранятся в специальных файлах — схемах, где самым подробным образом описана структура документа, все допустимые названия элементов, атрибутов и многое другое. И если документ, например, содержит не определённое заранее в схемах название элемента, то XML-документ считается недействительным; проверяющий XML-процессор (валидатор) при проверке на соответствие правилам и схемам обязан (по выбору пользователя) сообщить об ошибке.

52. Что такое DTD?

Определение типов документа (DTD) декларирует допустимые строительные блоки XML документа. Оно задает структуру документа со списком допустимых элементов и атрибутов.

DTD может декларироваться как в коде самого XML документа, так и во внешнем файле с подключением его к XML документу.

Цель DTD состоит в том, чтобы определить структуру XML документа. Это делается путем определения списка допустимых элементов

Зачем нужно использовать DTD?

С DTD ваш XML файл может нести собственный формат.

С DTD различные, не связанные друг с другом группы людей могут приходить к соглашению о стандартах пересекающихся данных.

С DTD вы можете быть уверены, что получаемые из внешних источников данные будут корректными.

53. Что такое XML-схема?

XML схема — это основанная на XML альтернатива DTD.

XML схема описывает структуру XML документа.

- ◆ Самодостаточен – содержит данные и метаданные
- ◆ Может содержать данные сложной структуры
 - ◆ Деревья, рекурсия, графы
 - ◆ Структурированные данные: повторяющиеся однородные структуры
 - ◆ Полуструктурированные данные: разнородные структуры, нерегулярные и редкие данные
 - ◆ Данные с разметкой тэгами
 - ◆ Гарантирует порядок данных
 - ◆ Отношения между данными выражаются
 - ◆ Как "содержится в" / "включено в"
 - ◆ Атрибутами-ссылками на элементы – при наличии схемы

54. Какой тип данных используется для хранения и обработки XML данных в СУБД MS SQL Server? В каких случаях может быть использован?

Может содержать:

- ◆ Документы XML
- ◆ Фрагменты XML
- ◆ Может связываться с XML Schema collection
- ◆ Запросы через XQuery
- ◆ Модификация через XML-DML
- ◆ Возможно индексирование XML
- ◆ Проверка на корректность и соответствие XML Schema

Сценарии использования XML

- Транспорт данных/обмен данными
 - Business-to-business (B2B)
 - business-to-consumer (B2C)
 - application-to-application (A2A)
- Управление хранением документов
- Разделение хранения и презентации данных
- Обмен сообщениями - Simple Object Access Protocol (SOAP)

Варианты хранения XML-данных

SQL Server поддерживает несколько вариантов хранения XML-данных.

- ◆ Естественное хранение в виде типа **xml**

Данные при этом хранятся во внутреннем представлении, которое обеспечивает неизменность XML-содержимого данных. Это внутреннее представление включает в себя сведения об иерархии контейнеров, порядке документов и значений элементов и атрибутов. Точнее говоря, при этом обеспечивается неизменность InfoSet-содержимого XML-данных. InfoSet-содержимое не всегда идентично текстовым XML-данным, потому что следующая информация при этом не сохраняется: несущественные пробелы, порядок атрибутов, префиксы пространств имен и XML-декларация.

Для типизированного (то есть связанного с **XML** -схемой) типа данных **xml** модуль проверки после обработки схемы (PSVI) добавляет в информационный набор данные о типах и кодирует их во внутреннее представление. Это значительно ускоряет синтаксический анализ.

- ◆ Сопоставление XML-данных и данных, хранящихся в реляционном формате

Используя аннотированную схему (AXSD), можно разбить XML на столбцы одной или нескольких таблиц. Это обеспечивает правильность данных на реляционном уровне. В результате гарантируется сохранность иерархической структуры данных, хотя порядок элементов не учитывается. Схема не может быть рекурсивной.

- ◆ Хранение больших объектов, **[n]varchar(max)** и **varbinary(max)**

При этом хранится идентичная копия данных. Это полезно в приложениях специального назначения, например в приложениях, обрабатывающих юридическую документацию. Большинству приложений точная копия данных не нужна — им хватает XML-содержимого (правильности элементов InfoSet).

Обычно используется сочетание этих подходов. Например, XML-данные можно сохранить в столбце типа **xml**, производя продвижение его свойств до уровня реляционных столбцов. Или же можно использовать

технологии сопоставления для хранения нерекурсивных фрагментов в столбцах, отличных от XML, а в столбцах типа **xml** хранить

55. Как происходит преобразование XML данных в реляционные данные в MS SQL Server?

Все узлы на 1 уровне становятся строками таблицы, все свойства внутри – значениями столбцов. Если свойство многоуровневое, то это считается внешним ключом на таблицу, которая тоже создаётся (это не точно).

56. Как происходит преобразование реляционных данных в XML данные в MS SQL Server? Какие есть виды такого преобразования? Какие существуют директивы преобразования?

Содержимое типа данных **xml** сериализуется обычным образом. То есть узлы элементов сопоставляются разметке элементов, а текстовые узлы сопоставляются текстовому содержимому.

Есть неявное преобразование (из **char**, **binary** и тд), можно использовать **CAST** и **CONVERT** (явное преобразование).

Директивы: **RAW**, **AUTO**, **EXPLICIT**, **PATH**.

Режим **RAW** генерирует один элемент **<row>** для каждой строки в наборе строк, который возвращается оператором **SELECT**. Вы можете создать иерархию XML, написав вложенные запросы **FOR XML**.

Режим **AUTO** создает вложение в результирующем XML с помощью эвристики на основе способа, указанного в инструкции **SELECT**. Вы имеете минимальный контроль над формой сгенерированного XML.

Режим **EXPLICIT** позволяет лучше контролировать форму XML. Вы можете по желанию смешивать атрибуты и элементы при определении формы XML.

Режим **PATH** вместе с возможностью вложенного запроса **FOR XML** упрощает режим **EXPLICIT**. В режиме **PATH** имена столбцов или псевдонимы столбцов обрабатываются как выражения **XPath**. Эти выражения показывают, как значения отображаются в XML.

57. Что такое XPath? Какие методы реализуют запросы XPath в MS SQL Server?

язык запросов Xpath

- ◆ XML Path Language — язык запросов к элементам XML-документа
- ◆ **query()** – выполнение запроса
- ◆ **exist()** – проверка существования

- ◆ value() – возвращает значение атрибута
- ◆ nodes() – возвращает набор узлов
- ◆ modify() – изменение документа
 - ◆ insert
 - ◆ delete
 - ◆ replace value of

58. Что такое XQuery? Какие методы реализуют запросы XQuery в MS SQL Server?

XQuery — язык запросов, разработанный для обработки данных в формате XML. XQuery предназначен для запроса XML-данных - не только XML файлы, но все, что может выглядеть как XML, включая базы данных.

query принимает оператор XQuery в качестве ввода и возвращает экземпляр типа данных xml;

exist принимает оператор XQuery в качестве ввода и возвращает 0, 1 или null в зависимости от результата запроса;

value принимает оператор XQuery в качестве ввода и возвращает единственное скалярное значение;

nodes полезен, когда вы хотите разделить тип данных xml на реляционные данные. Это позволяет идентифицировать узлы, которые будут размещены в новых строках;

modify. В отличие от стандартизированной версии XQuery, которая в своей версии 1.0 не имеет спецификаций для операций изменения, SQL Server поддерживает добавление, модификацию и удаление документов XML при использовании метода modify o.

59. Какие виды индексов можно строить по XML столбцу в MS SQL Server?

XML-индексы разделяются на следующие категории.

- ◆ Первичный XML-индекс
- ◆ Вторичные XML-индексы.

Первым индексом, создаваемым для столбца типа данных **xml**, должен быть первичный XML-индекс. При наличии первичного XML-индекса поддерживаются вторичные индексы трех типов: PATH, VALUE и PROPERTY. Эти вторичные индексы могут способствовать повышению производительности выполнения разных типов запросов.

Полнотекстовый индекс

- XML-разметка воспринимается как граница слов
- Содержание тегов и атрибутов индексируется
- Синтаксис – как и для всех остальных типов

CREATE FULLTEXT INDEX ON t (xDoc)

- Можно комбинировать с XQuery:
 - ◆ Сначала full-text index для предварительного отбора
 - ◆ Потом XML-индекс

60. Как тип данных XML применяется в триггерах базы данных в MS SQL Server?

```

CREATE TRIGGER TR_DATABASE_DDL_TRACKING
ON DATABASE FOR DDL_DATABASE_LEVEL_EVENTS
AS
DECLARE @PostTime datetime2 = EVENTDATA().value('(/EVENT_INSTANCE/PostTime)[1]', 'datetime2');
DECLARE @LoginName sysname = EVENTDATA().value('(/EVENT_INSTANCE/LoginName)[1]', 'sysname');
DECLARE @TSQLCommand nvarchar(max) = EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]',
PRINT 'DDL Event Occurred';
PRINT @LoginName;
PRINT 'executed';
PRINT @TSQLCommand;
PRINT 'at';
PRINT @PostTime;
GO

CREATE TABLE TestTable (TestTableID int);
GO
DROP TABLE TestTable;
GO

```

Messages

```

DDL Event Occurred
ADVENTUREWORKS\Student
executed
CREATE TABLE TestTable (TestTableID int);
at
2017-04-01 05:47:07.8400000
DDL Event Occurred
ADVENTUREWORKS\Student
executed
DROP TABLE TestTable;
at
2017-04-01 05:47:07.8800000

```

```

CREATE TRIGGER TR_DDL_ProcNamingConvention
ON DATABASE
FOR CREATE_PROCEDURE
AS
BEGIN
SET NOCOUNT ON;

DECLARE @EventData xml;
DECLARE @ObjectName sysname;

SET @EventData = EVENTDATA();
SET @ObjectName = @EventData.value('(/EVENT_INSTANCE/ObjectName)[1]', 'sysname');

IF @ObjectName LIKE 'sp%'
BEGIN
PRINT 'Stored Procedure names are not permitted to start with sp';
ROLLBACK TRAN;
END;
END;
GO

```

Messages

```

Stored Procedure names are not permitted to start with sp
Msg 3609, Level 16, State 2, Procedure sp_GetVersion, Line 68
The transaction ended in the trigger. The batch has been aborted.

```

```

CREATE PROC GetVersion
AS
SELECT @@VERSION;
GO

CREATE PROC sp_GetVersion
AS
SELECT @@VERSION;
GO

DROP PROC GetVersion;
GO

```

```

CREATE TRIGGER TR_DDL_CREATE_TABLE_PK
ON DATABASE
FOR CREATE_TABLE,ALTER_TABLE
AS BEGIN
SET NOCOUNT ON;

DECLARE @EventData xml;
DECLARE @SchemaName sysname;
DECLARE @ObjectName sysname;
DECLARE @FullName nvarchar(max);

SET @EventData = EVENTDATA();
SET @SchemaName = @EventData.value('(/EVENT_INSTANCE/SchemaName)[1]', 'sysname');
SET @ObjectName = @EventData.value('(/EVENT_INSTANCE/ObjectName)[1]', 'sysname');
SET @FullName = QUOTENAME(@SchemaName)+'.'+QUOTENAME(@ObjectName);

IF OBJECTPROPERTY(OBJECT_ID(@FullName), 'TableHasPrimaryKey') <> 1
BEGIN
PRINT 'Table needs to have a Primary Key';
ROLLBACK TRAN;
END;
END;

```

Messages

```

Table needs to have a Primary Key
Msg 3609, Level 16, State 2, Line 104
The transaction ended in the trigger. The batch has been aborted.

```

```

CREATE TABLE dbo.ValueList
( ValueListID int IDENTITY(1,1),
Value decimal(18,2)
);
GO

```

61. Какой тип данных используется для хранения и

обработки XML данных в СУБД ORACLE?

Oracle XMLTYPE:

Тип данных, облегчающий обработку данных XML

может использоваться в хранимых процедурах

может представлять XML в SQL

имеет встроенные функции, которые работают с содержимым

XML

функция EXTRACTVALUE извлекает значения элемента из документа XML

Функция EXISTSNode дает возможность использовать в SQL условие отбора XPath (язык отбора, принятый в технологиях XML)

62. Как извлечь элемент из XML переменной в СУБД ORACLE?

Для выборки можно использовать специально придуманные для XMLTYPE функции. Так, функция EXTRACTVALUE извлекает значения элемента из документа XML:

```
SELECT id, EXTRACTVALUE(description, '/cover/title')
FROM books;
```

63. Как извлечь атрибут из XML переменной в СУБД ORACLE?

```
create or replace procedure findProcedure(word in VARCHAR2, aa out
VARCHAR2 )
is
begin
  select r.xml.GETSTRINGVAL() xml
  into aa from report r
  where r.xml.EXISTSNode('/XML/SUBJECT/ROWSET/ROW[FACULTY=""||
word||""]')=1;
end findProcedure;
set SERVEROUTPUT ON
```

Функция EXISTSNode дает возможность использовать в SQL условие отбора XPath (язык отбора, принятый в технологиях XML):

```
SELECT id, b.description.XMLDATA
FROM books b
WHERE b.description.EXISTSNode('/cover[author="Sanjay
Mishra"]')=1;
```

64. Как извлечь набор вложенных узлов из XML переменной в СУБД ORACLE?

65. Что такое XMLTABLE в СУБД ORACLE?

XMLTABLE разворачивает элементы документа XML в столбцы таблицы.

66. Как превратить таблицу в XML фрагмент в СУБД ORACLE?

Для сохранения и управления XML-данными в реляционной таблице применяется специальный тип данных XMLType. Использовать тип XMLType можно точно так же, как и обычные типы данных в базе данных Oracle. Благодаря ему, правильно оформленный XML-документ теперь можно сохранять в базе данных в виде XML-теста с использованием базового типа данных CLOB.

67. Как добавить атрибуты из реляционных данных в XML фрагмент в СУБД ORACLE?

```
SELECT XMLElement("Emp", XMLAttributes(
                                e.employee_id as "ID",
                                e.first_name || ' ' || e.last_name AS "name"))
AS "RESULT"
FROM hr.employees e
WHERE employee_id > 200;
```

This query produces the following typical XML result fragment:

```
RESULT
-----
<Emp ID="201" name="Michael Hartstein"></Emp>
<Emp ID="202" name="Pat Fay"></Emp>
<Emp ID="203" name="Susan Mavris"></Emp>
<Emp ID="204" name="Hermann Baer"></Emp>
<Emp ID="205" name="Shelley Higgins"></Emp>
<Emp ID="206" name="William Gietz"></Emp>

6 rows selected.
```

68. Как превратить две, связанные по внешнему ключу таблицы, в XML фрагмент с подчиненными узлами в СУБД ORACLE?

Example 18-9 XMLFOREST: Generating an Element from a User-Defined Data-Type Instance

```
SELECT XMLForest(  
  dept_t(department_id,  
    department_name,  
    cast(MULTISET  
      (SELECT employee_id, last_name  
        FROM hr.employees e WHERE e.department_id = d.department_id)  
      AS emplist_t))  
  AS "Department")  
AS deptxml  
FROM hr.departments d  
WHERE department_id=10;
```

This produces an XML document with element **Department** containing attribute **DEPTNO** and child element **DNAME**.

```
DEPTXML  
-----  
<Department DEPTNO="10">  
  <DNAME>Administration</DNAME>  
  <EMP_LIST>  
    <EMP_T EMPNO="200">  
      <ENAME>Whalen</ENAME>  
    </EMP_T>  
  </EMP_LIST>  
</Department>  
  
1 row selected.
```

69. Что такое окно в Transact SQL?

Окно - стандартный термин SQL, служащий для описания контекста в котором работает функция. Для указания окна в SQL используется предложение OVER.

Напомню, окно – это набор строк, по которым производится вычисление функции. Инструкция OVER разбивает весь набор строк на отдельные группы – окна согласно заданному условию.

70. Что такое оконная функция?

Оконными функциями (window functions) называются функции, которые применяются к наборам строк и определяются посредством предложения OVER. Эти функции основаны на глубоком принципе языка SQL (обоих стандартов - ISO и ANSI) - принципе работы с окнами (windowing). Основа этого принципа - возможность выполнять различные вычисления с набором, или окном, строк и возвращать одно значение.

71. Поясните значение конструкции OVER в параметрах окна.

Если быть более точным, то окно представляет собой набор строк, или отношение, предоставляемые как входные данные этапа обработки логического запроса, в котором определено это окно. Но самый важный нюанс заключается в том, что предложение OVER определяет окно функции по отношению к текущей строке. Иначе говоря, в каждой строке предложение OVER определяет окно независимо от остальных строк.

72. Какие типы оконных функций в MS SQL Server вы знаете?

В стандарте SQL предусмотрена поддержка нескольких типов оконных функций: агрегатные, ранжирующие, аналитические (или распределения) и сдвига.

73. Кратко охарактеризуйте каждый тип оконных функций.

Оконные функции агрегирования представляют собой то же, что и агрегатные функции группировки, но вместо применения к группам в групповых запросах они применяются к окнам, определяемых в предложении OVER.

Ранжирующие функции возвращают ранжирующее значение для каждой строки в секции. В зависимости от используемой функции значения некоторых строк могут совпадать.

Аналитические функции вычисляют статистическое значение на основе группы строк. В отличие от агрегатных функций, аналитические функции могут возвращать несколько строк для каждой группы. Аналитические функции можно использовать для вычисления скользящих средних, промежуточных итогов, процентных долей или первых N результатов в группе.

74. Для чего применяется секция PARTITION BY?

Предложение PARTITION BY определяет столбец, по которому будет производиться группировка, и он является ключевым в разбиении набора строк на окна.

75. Для чего применяется секция ORDER BY?

Оператор SQL ORDER BY выполняет сортировку выходных значений. Оператор SQL ORDER BY можно применять как к числовым столбцам, так и к строковым. В последнем случае, сортировка будет происходить по алфавиту.

76. Поясните особенности применения агрегатных

функций в сочетании с конструкцией ORDER BY в параметрах окна.

77. Для чего применяется выражения ROWS и RANGE?

Предложение ROWS ограничивает строки в окне, указывая фиксированное количество строк, предшествующих или следующих за текущей. Оба предложения ROWS и RANGE используются вместе с ORDER BY.

78. Какие методы используются для обработки строк и диапазонов?

LEN: возвращает количество символов в строке. В качестве параметра в функцию передается строка, для которой надо найти длину:

LTRIM: удаляет начальные пробелы из строки. В качестве параметра принимает строку:

RTRIM: удаляет конечные пробелы из строки. В качестве параметра принимает строку:

CHARINDEX: возвращает индекс, по которому находится первое вхождение подстроки в строке. В качестве первого параметра передается подстрока, а в качестве второго - строка, в которой надо вести поиск:

PATINDEX: возвращает индекс, по которому находится первое вхождение определенного шаблона в строке:

LEFT: вырезает с начала строки определенное количество символов. Первый параметр функции - строка, а второй - количество символов, которые надо вырезать сначала строки:

RIGHT: вырезает с конца строки определенное количество символов. Первый параметр функции - строка, а второй - количество символов, которые надо вырезать сначала строки:

SUBSTRING: вырезает из строки подстроку определенной длиной, начиная с определенного индекса. Первый параметр функции - строка, второй - начальный индекс для вырезки, и третий параметр - количество вырезаемых символов:

REPLACE: заменяет одну подстроку другой в рамках строки. Первый параметр функции - строка, второй - подстрока, которую надо заменить, а третий - подстрока, на которую надо заменить:

REVERSE: переворачивает строку наоборот:

CONCAT: объединяет две строки в одну. В качестве параметра принимает от 2-х и более строк, которые надо соединить

LOWER: переводит строку в нижний регистр:

РАБОТА С ДИАПАЗОНАМИ

Оператор **TOP** позволяет выбрать определенное количество строк из

таблицы

PERCENT позволяет выбрать процентное количество строк из таблицы. Например, выберем 75% строк

Для извлечения набора строк из любого места, применяются операторы **OFFSET** и **FETCH**. Важно, что эти операторы применяются только в отсортированном наборе данных после выражения **ORDER BY**.

79. Для чего предназначены функции **LAG ()** и **LEAD ()**?

Эти функции возвращают значение выражения, вычисленного для предыдущей строки (**LAG**) или следующей строки (**LEAD**) результирующего набора соответственно.

80. Что такое функции ранжирования в MS SQL Server? Перечислите и кратко охарактеризуйте функции ранжирования в MS SQL Server.

Ранжирующие функции возвращают ранжирующее значение для каждой строки в секции. В зависимости от используемой функции значения некоторых строк могут совпадать.

RANK, DENSE_RANK, ROW_NUMBER и **NTILE**

Эти функции, как и функция **ROW_NUMBER()**, тоже нумеруют строки, но делают это несколько отличным способом. Это отличие проявляется в том, что строки, которые имеют одинаковые значения в столбцах, по которым выполняется упорядочивание, получают одинаковые номера (ранги). Например, значения (отсортированные по возрастанию) Возникает вопрос, с какого номера продолжится нумерация, если, скажем, в последовательности чисел появится 7 и т.д.? Здесь есть два варианта:

- 1) с номера 4, т.к. это следующий номер по порядку;
- 2) с номера 6, т.к. следующая строка будет шестая по счету.

Такая "неоднозначность" и привела к появлению двух функций вместо одной - **RANK** и **DENSE_RANK**, первая из которых продолжит нумерацию с 6, а вторая (плотная)

DENSE_RANK = Эта функция возвращает ранг каждой строки в разделе набора результатов без пробелов в значениях ранжирования. Ранг конкретной строки равен единице плюс количество различных значений ранга, которые предшествуют этой конкретной строке.\

ROW_NUMBER Нумерует вывод набора результатов. Более конкретно, возвращает порядковый номер строки в разделе набора результатов, начиная с 1 для первой строки в каждом разделе.

ROW_NUMBER и RANK похожи. ROW_NUMBER нумерует все строки последовательно (например, 1, 2, 3, 4, 5). RANK предоставляет одинаковые числовые значения для связей (например, 1, 2, 2, 4, 5)

nTILE Распределяет строки в упорядоченном разделе по указанному количеству групп. Группы нумеруются, начиная с одной. Для каждой строки NTILE возвращает номер группы, к которой принадлежит строка

81. Что такое аналитические функции в MS SQL Server? Перечислите и кратко охарактеризуйте аналитические функции в MS SQL Server.

Аналитические функции вычисляют статистическое значение на основе группы строк. В отличие от агрегатных функций, аналитические функции могут возвращать несколько строк для каждой группы. Аналитические функции можно использовать для вычисления скользящих средних, промежуточных итогов, процентных долей или первых N результатов в группе.

PERCENT_RANK, CUME_DIST, PERCENTILE_CONT и PERCENTILE_DISC

PERCENT_RANK Вычисляет относительный ранг строки в группе строк в SQL Server 2019 (15.x). Используйте PERCENT_RANK, чтобы оценить относительное положение значения в наборе результатов запроса или в разделе. PERCENT_RANK похож на функцию CUME_DIST.

CUME_DIST Для SQL Server эта функция вычисляет совокупное распределение значения в группе значений. Другими словами, CUME_DIST вычисляет относительную позицию указанного значения в группе значений. Предполагая возрастающий порядок, CUME_DIST значения в строке *r* определяется как число строк со значениями, меньшими или равными этому значению в строке *r*, деленное на количество строк, оцененных в наборе результатов раздела или запроса. CUME_DIST похож на функцию PERCENT_RANK.

PERCENTILE_CONT Вычисляет процентиль на основе непрерывного распределения значения столбца в SQL Server. Результат интерполируется и может не совпадать ни с одним из определенных значений в столбце.

PERCENTILE_DISC Вычисляет конкретный процентиль для отсортированных значений во всем наборе строк или в отдельных разделах набора строк в SQL Server. Для заданного значения percentila P PERCENTILE_DISC сортирует значения выражений в предложении ORDER BY. Затем он возвращает значение с наименьшим заданным значением CUME_DIST (по отношению к той же спецификации сортировки), которое

больше или равно P. Например, PERCENTILE_DISC (0.5) будет вычислять 50-й процентиль (то есть медиану) выражение. PERCENTILE_DISC вычисляет процентиль на основе дискретного распределения значений столбца. Результат равен определенному значению столбца.

82. Что такое расширенные группировки в MS SQL Server? Перечислите и кратко охарактеризуйте расширенные группировки в MS SQL Server.

Дополнительно к стандартным операторам GROUP BY и HAVING SQL Server поддерживает еще четыре специальных расширения для группировки данных: **ROLLUP**, **CUBE**, **GROUPING SETS** и **OVER**.

Оператор **ROLLUP** добавляет суммирующую строку в результирующий набор, в конце таблицы была добавлена дополнительная строка, которая суммирует значение столбцов. При сортировке с помощью ORDER BY следует учитывать, что она применяется уже после добавления суммирующей строки.

CUBE похож на ROLLUP за тем исключением, что CUBE добавляет суммирующие строки для каждой комбинации групп.

Оператор **GROUPING SETS** аналогично ROLLUP и CUBE добавляет суммирующую строку для групп. Но при этом он не включает сами группам. При этом его можно комбинировать с ROLLUP или CUBE. Например, кроме суммирующих строк по каждой из групп добавим суммирующую строку для всех групп

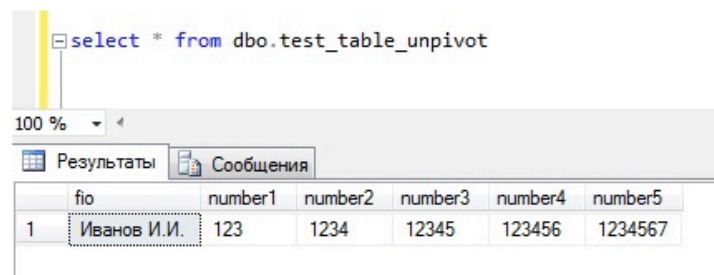
Выражение **OVER** позволяет суммировать данные, при этом возвращая те строки, которые использовались для получения суммированных данных. Выражение **OVER** ставится после агрегатной функции, затем в скобках идет выражение **PARTITION BY** и столбец, по которому выполняется группировка.

83. Для чего предназначена конструкция UNPIVOT в MS SQL Server? Опишите синтаксис UNPIVOT.

UNPIVOT – это оператор Transact-SQL, который выполняет действия, обратные PIVOT. Сразу скажу, что да он разворачивает таблицу в обратную сторону, но в отличие от оператора PIVOT он ничего не агрегирует и уж тем более не раз агрегирует.

```
CREATE TABLE [dbo].[test_table_unpivot](
    [fio] [varchar](50) NULL,
    [number1] [int] NULL,
    [number2] [int] NULL,
    [number3] [int] NULL,
    [number4] [int] NULL,
    [number5] [int] NULL,
) ON [PRIMARY]
GO
```

Данные будут, например, такие:



The screenshot shows a SQL query window with the text: `select * from dbo.test_table_unpivot`. Below the query window, the 'Results' tab is active, displaying a table with 6 columns: fio, number1, number2, number3, number4, and number5. The first row contains the values: 1, Иванов И.И., 123, 1234, 12345, 123456, 1234567.

	fio	number1	number2	number3	number4	number5
1	Иванов И.И.	123	1234	12345	123456	1234567

И допустим, нам необходимо развернуть эту таблицу, для этого мы будем использовать оператор UNPIVOT, а запрос будет выглядеть следующим образом:

```
SELECT fio, column_name, number
FROM dbo.test_table_unpivot
UNPIVOT(
    number for column_name in (
        [number1], [number2], [number3], [number4], [number5]
    )
) AS test_unpivot
```

	fio	column_name	number
1	Иванов И.И.	number1	123
2	Иванов И.И.	number2	1234
3	Иванов И.И.	number3	12345
4	Иванов И.И.	number4	123456
5	Иванов И.И.	number5	1234567

Где,

- fio – столбец с ФИО, он в принципе не изменился;
- column_name – псевдоним столбца, который будет содержать названия наших колонок;
- number – псевдоним для значений из столбцов number1, number2...

84. Для чего предназначена конструкция PIVOT в MS SQL Server? Опишите синтаксис PIVOT.

PIVOT – это оператор Transact-SQL, который поворачивает результирующий набор данных, т.е. происходит транспонирование таблицы, при этом используются агрегатные функции, и данные соответственно группируются. Другими словами, значения, которые расположены по вертикали, мы выстраиваем по горизонтали.

Данный оператор может потребоваться тогда, когда необходимо, например, предоставить какой либо отчет в наглядной форме по годам, допустим для бухгалтеров и экономистов, так как именно они любят представления данных в таком виде. Также он может пригодиться и просто для представления какой-либо статистики

```
CREATE TABLE [dbo].[test_table_pivot](
    [fio] [varchar](50) NULL,
    [god] [int] NULL,
    [summa] [float] NULL
) ON [PRIMARY]
GO
```

```
SELECT fio, [2011], [2012], [2013], [2014], [2015]
FROM dbo.test_table_pivot
PIVOT (SUM(summa)for god in ([2011],[2012],[2013],[2014],[2015])
      ) AS test_pivot
```

Здесь у нас:

- ♦ **fio** — столбец, по которому мы будем осуществлять группировку;
- ♦ **[2011],[2012],[2013],[2014],[2015]** — названия наших столбцов по горизонтали, ими выступают значения из колонки god;
- ♦ **sum(summa)** — агрегатная функция по столбцу summa;
- ♦ **for god in ([2011],[2012],[2013],[2014],[2015])** — тут мы указываем колонку, в которой содержатся значения, которые будут выступать в качестве названия наших результирующих столбцов, по факту в скобках мы указываем то же самое, что и чуть выше в select;
- ♦ **as test_pivot** — это обязательный псевдоним, не забывайте его указывать, иначе будет ошибка.

85. Укажите порядок выполнения SELECT оператора в SQLITE.

SELECT DISTINCT -> FROM -> WHERE -> GROUP BY -> HAVING ->
ORDER BY -> LIMIT OFFSET

86. Что такое класс хранения? Перечислите классы хранения, применяемые в SQLITE.

SQLite вместо типов данных использует классы данных. **Классы данных в SQLite3** – это более широкое понятие, нежели тип. Любое значение в SQLite3 может иметь один из пяти классов:

- **NULL** – пустое значение или его отсутствие. Значение NULL в SQLite3, как и во многих других СУБД и языках программирования уникально, то есть оно не равно ни другому NULL ни еще какому-либо значению. На NULL можно только проверить;
- **INTEGER** – класс данных INTEGER в SQLite3 используется для хранения целочисленных значений, которые хранятся в 1, 2, 3, 4, 6 или 8 байтах, в зависимости от самого значения;
- **REAL** – класс данных REAL в SQLite3 предназначен для хранения вещественных чисел, данные в классе REAL хранятся в формате восьмибайтного числа IEEE с плавающей точкой (вспомните школьную математику, где вам рассказывали про

мантиссу и порядок);

- **TEXT** – класс данных TEXT в SQLite3 используется для хранения строковых значений, в базе данных данные с классом TEXT хранятся с использованием кодировки UTF-8 или UTF-16 (это можно настроить, рекомендуем использовать UTF-8);
- **BLOB** – класс данных BLOB в SQLite3 используется для хранения бинарных данных, данные с классом BLOB хранятся в том виде, в котором они были введены.

В SQLite3 довольно интересно реализован процесс хранения и обработки данных. Например, числа в базе данных с классом INTEGER хранятся на диске по-разному и могут занимать разное количество байт, но, когда мы делаем запрос, числа поступают в оперативную память и приводятся к восьмибайтному формату.

87. В чем состоит понятие аффинированных типов?

Аффинированный тип данных – это рекомендуемый тип данных для столбца (можно сказать, что это приоритетный тип данных), конечно, это не отменяет того, что в любой столбец можно записать любое значение, но в некоторых ситуациях СУБД будет отдавать приоритет **аффинированному типу данных** и стараться преобразовать значения к рекомендуемому типу.

BLOB; TEXT; REAL; INTEGER; NUMERIC.

88. Как применяется аффинированность для классов хранения?

Дам пояснения: первое, что нужно запомнить – в **SQLite3 нет типов данных**, а есть всего лишь пять классов данных.

В SQLite3 действует пять правил определения аффинированности столбца:

1. Если объявление типа содержит строку «INT», то столбец ассоциируется с аффинированным INTEGER
2. Если объявление типа столбца содержит любую из строк «CHAR», «CLOB», или «TEXT», то аффинированность определяется как TEXT. Обратите внимание, что тип VARCHAR содержит подстроку «CHAR», и поэтому ему тоже сопоставляется аффинированный TEXT
3. Если объявление типа столбца содержит строку «BLOB» или если тип не указан, то столбец аффинируется с NONE
4. Если объявление типа столбца содержит любую из строк «REAL», «FLOA» или «DOUB», аффинированность определяется как REAL
5. В остальных случаях столбцу сопоставляется аффинированный NUMERIC.

89. Что такое COLLATE? Укажите типы COLLATE.

Каждый столбец каждой таблицы имеет связанную с ним функцию сортировки. Если функция не определена явно, то по умолчанию используется BINARY. В случае задействования конструкции COLLATE в определении столбца назначается альтернативная сортирующая функция.

В SQLite (да и в любой базе) вводится понятие **collation**: способ сравнения двух строк между собой. Итак, **когда SQLite необходимо сравнить две строки, то это сравнение всегда основано на каком-то collation.**

Существует три встроенных collation:

BINARY: обычное побайтовое сравнение двух блоков памяти: старый, добрый *memcmp()* (используется по умолчанию, если не указан другой collation);

RTRIM: тоже, что и BINARY, но игнорирует концевые пробелы ('abc ' = 'abc');

NOCASE: тоже, что и BINARY, но игнорирует регистр для 26 букв латинского алфавита (и только для них).

90. В чем состоит назначение столбца ROWID?

У каждой таблицы базы данных SQLite по умолчанию есть внутренний индекс, который называется ROWID. Иногда этот индекс совпадает с первичным ключом таблицы PRIMARY KEY, иногда не совпадает, но ROWID есть у каждой таблицы. Если говорить просто и понятно, то **ROWID – это дополнительный столбец, который есть у любой таблицы SQLite и является индексом для данной таблицы.**

Помимо того, что ROWID – это индекс, это еще один способ поддержания целостности данных. Давайте разберемся с индексом **ROWID в базах данных SQLite.** ROWID – это 64-разрядное число, которое однозначно идентифицирует любую строку в базе данных SQLite.

91. В чем состоит назначение опции WITHOUT ROWID?

В SQLite можно создавать таблицы без столбца ROWID, то есть без внутреннего индекса. Такой подход несет незначительные плюсы: иногда это ускоряет выборку данных из базы данных и немного уменьшает объем базы данных. Если вы решили создать таблицу без ROWID в SQLite, то вам необходимо сделать две вещи: во-первых, у таблицы должен быть обязательно первичный ключ, во-вторых, вам необходимо использовать **ключевую фразу WITHOUT ROWID.** Еще одной особенностью таблиц WITHOUT ROWID является то, что ограничение уровня столбца AUTOINCREMENT работать не будет.

Если вам нужно повысить скорость работы таблицы с первичным ключом INTEGER, создавайте таблицу без внутреннего индекса, этим вы

ускорите выборку и уменьшите объем базы данных. Если таблицы имеет индекс ROWID и индекс в виде INTEGER PRIMARY KEY, то SQLite делает перебор двумя циклами: первый цикл идет по столбцу ROWID, второй цикл идет по столбцу INTEGER PRIMARY KEY, хотя значения этих столбцов совпадают.

Если в таблицах хранятся строки с малыми значениями, то таблицы без индекса (WITHOUT ROWID) будут работать несколько быстрее, чем со столбцом ROWID.

92. В чем заключается механизм разрешения конфликта при вставке/обновлении данных?

93. Что такое виртуальная таблица?

Представление (VIEW) — объект базы данных, являющийся результатом выполнения запроса к базе данных, определенного с помощью оператора SELECT, в момент обращения к представлению.

Представления иногда называют «**виртуальными таблицами**». Такое название связано с тем, что представление доступно для пользователя как таблица, но само оно не содержит данных, а извлекает их из таблиц в момент обращения к нему. Если данные изменены в базовой таблице, то пользователь получит актуальные данные при обращении к представлению, использующему данную таблицу; кэширования результатов выборки из таблицы при работе представлений не производится. При этом, механизм кэширования запросов (query cache) работает на уровне запросов пользователя безотносительно к тому, обращается ли пользователь к таблицам или представлениям.

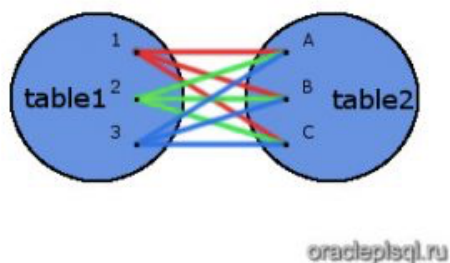
94. Укажите типы соединения и объединения таблиц, используемые в SQLITE.

Joins используется для объединения записей из двух или более таблиц в базе данных. SQL определяет три основных типа объединений:

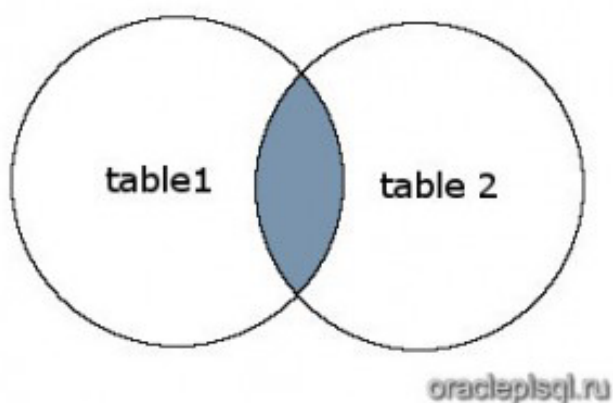
- **CROSS JOIN** - сопоставляет каждую строку первой таблицы с каждой строкой второй таблицы. В отличие от соединения INNER или OUTER, CROSS JOIN не имеет условий для объединения двух таблиц.
- **INNER JOIN** - создает новую таблицу результатов, комбинируя значения столбцов двух таблиц (table1 и table2) на основе условия соединения
- **OUTER JOIN**
 - **LEFT OUTER JOIN** - возвращает все строки из таблиц с левосторонним соединением, указанным в условии ON, и

только те строки из другой таблицы, где объединяемые поля равны (выполняется условие объединения)

На этом рисунке SQLite CROSS JOIN возвращает каждую строку из table1, соответствующую каждой строке из table2.



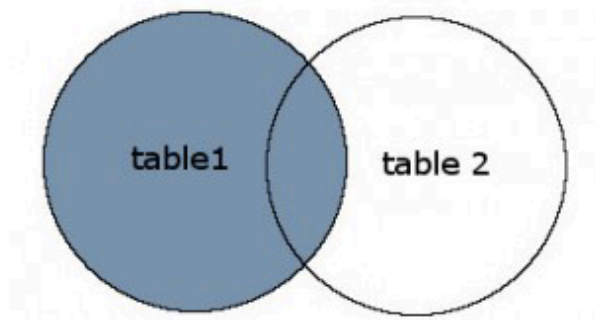
На этом рисунке SQLite INNER JOIN возвращает затененную область:



```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Рисунок

На этом рисунке SQLite LEFT OUTER JOIN возвращает затененную область:



95. Перечислите все ограничения целостности для таблицы или столбца, поддерживаемые SQLITE.

На уровне столбца

1. Значение NOT NULL. Столбец, которому мы сказали NOT NULL не может хранить в себе значение NULL, тогда при выполнении команды INSERT, если вы забудете добавить значение в столбец, SQLite вернет вам ошибку.
2. Значение UNIQUE. Если вы зададите столбцу правило UNIQUE, то SQLite позаботится о том, чтобы в столбце хранились только уникальные значения.
3. Значение CHECK в базах данных SQLite задает диапазон значений, например, у вас есть таблица с сотрудниками и зарплатой, сотрудник не может получать зарплату ниже MPOT, так обеспечивает целостность данных CHECK.
4. Значение DEFAULT позволяет задать значение для столбца по умолчанию: SQLite будет подставлять значение по умолчанию в том случае, когда вы забудете добавить значение для столбца при добавлении новой строки в таблицу.
5. Тип данных столбца, вернее, класс данных столбца, но если уж быть совсем точным в контексте SQLite3, то аффинированный тип данных столбца является ограничением и нужен для поддержания и обеспечения целостности данных.
6. Говорили про конструкцию COLLATE, которая позволяет задать правила сравнения. На первый взгляд эта конструкция **не относится к механизмам обеспечения целостности данных**, но если взглянуть на нее так: мы можем использовать

фильтрацию WHERE при удалении данных из таблицы командой DELETE, когда мы используем WHERE, SQLite начинает сравнивать значения, чтобы решить: нужно его удалять или нет.

На уровне таблицы

1. Первичный ключ. Первичный ключ в базах данных SQLite может быть составным или обычным. Первичный ключ в базах данных SQLite3 является ограничением уровня таблицы и для любой системы управления базами данных первичный ключ является правилом, которое нельзя нарушать, **тем самым обеспечивается целостность данных**. Если вы помните, то ключевые атрибуты обязательны для второй нормальной формы.
2. Ограничение CHECK. Если честно, то я не определился к какому уровню относится ограничение CHECK в SQLite. С одной стороны: данное ограничение обеспечивает целостность данных только одного столбца. С другой стороны, этому ограничению в базах данных SQLite можно дать имя. CHECK ограничивает диапазон значений, который может храниться в столбце.
3. Внешний ключ является ограничением уровня таблицы и позволяет **обеспечить целостность данных в базе данных**, вернее, в связанных таблицах. При помощи внешнего ключа реализуются всевозможные связи между таблицами базы данных. Используя внешние ключи, мы проводим нормализацию отношений.

96. Укажите и кратко поясните все варианты применения ограничения FOREIGN KEY при обновлении/удалении соответствующего первичного ключа.

Ограничение ссылочной целостности задает требование, согласно которому для каждой записи в дочерней таблице должна иметься запись в родительской таблице. При этом изменение значения столбца связи в записи родительской таблицы при наличии дочерней записи блокируется, равно как и удаление родительской записи (запрет каскадного изменения и удаления)

Сделать у нас это не получится, потому что сработает ограничение внешнего ключа, мы не можем удалить данные из таблицы справочника авторов, пока на них ссылаются значения из таблицы книг. Ошибка будет следующей: Error: FOREIGN KEY constraint failed.

Теперь мы удалили упоминание об авторе в таблице книг, а затем смогли удалить данные об авторе из таблицы auth, по-другому удалить данные из справочника мы не сможем: сперва удаляем данные из ссылающейся таблицы, а потом удаляем данные из таблицы, на которую

ссылаемся – **это правило внешнего ключа**, которое обеспечивает целостность данных и защищает нас от аномалии удаления данных.

Изменение существующего значения внешнего ключа на значение, которого нет в соответствующем столбце главной таблицы. В нашем примере ограничение не позволит выполнить такой оператор **UPDATE**

Удаление строки из главной таблицы, для которой есть связанные строки в подчиненной таблице. Согласованность данных здесь может поддерживаться разными способами, в соответствии со значением опции в необязательном предложении

97. Поясните понятие «каскадные операции» для ограничения FOREIGN KEY.

Соответствующие строки обновляются или удаляются из ссылающейся таблицы, если данная строка обновляется или удаляется из родительской таблицы.

каскадное удаление, т.е. при удалении строки из главной таблицы будут удалены также связанные строки из подчиненной таблицы.

98. Что такое отложенная проверка ограничения FOREIGN KEY? Как ее установить?

Все внешние ключи в SQLite подразделяются на немедленные и отложенные. По умолчанию внешние ключи создаются как немедленные.

Если запрос модифицирует содержимое базы данных таким образом, что нарушается ограничение немедленного внешнего ключа, то по завершению запроса будет брошено исключение (exception) и результат запроса будет отменен.

В противоположность этому, если запрос модифицирует содержание базы таким образом, что нарушается ограничение отложенного внешнего ключа, то сообщение об этой ошибке не будет выдано немедленно. Ограничения отложенных внешних ключей не проверяется до попытки завершить транзакцию с помощью COMMIT.

Чтобы пометить внешний ключ как отложенный, нужно включить в его определение следующее выражение:

DEFERRABLE INITIALLY DEFERRED -- Отложенное ограничение внешнего ключа

```
trackartist INTEGER REFERENCES artist(artistid)
DEFERRABLE INITIALLY DEFERRED
```

99. Что такое PRAGMA? Для чего используется и каким образом задается?

SQL команда **PRAGMA** служит для задания всевозможных настроек у

соединения или у самой БД

```
PRAGMA name; // запросить текущее значение
параметра name
PRAGMA name = value; // задать параметр name значением
value
PRAGMA encoding = "UTF-8"; // тип данных БД, всегда используйте
UTF-8
PRAGMA foreign_keys = 1; // включить поддержку foreign keys,
по умолчанию - ОТКЛЮЧЕНА
PRAGMA cache_size = 3000; // Задает размер кэша. Но только на
одну сессию (т.е. до закрытия подключения).
```

Настройку соединения (очевидно) следует проводить сразу после открытия и до его использования. Их можно задавать в строке подключения или выполнять в виде запросов. **Запомните важную вещь, что если вы меняете что-то через запрос, то это действует только на время одной сессии, т.е. подключения.**

100. Что происходит при неизвестной PRAGMA? Перечислите известные вам утверждения PRAGMA.

Программа, которая работает с файлом базы, сама настраивает свойства подключения и может их изменять при открытом соединении. Это значит, что необязательно задавать все нужные опции при подключении.

```
PRAGMA name; // запросить текущее значение
параметра name
PRAGMA name = value; // задать параметр name значением
value
PRAGMA encoding = "UTF-8"; // тип данных БД, всегда используйте
UTF-8
PRAGMA foreign_keys = 1; // включить поддержку foreign keys,
по умолчанию - ОТКЛЮЧЕНА
PRAGMA cache_size = 3000; // Задает размер кэша. Но только на
одну сессию (т.е. до закрытия подключения).
```

101 - 102. Есть ли класс хранения для даты в SQLITE? Как осуществляется работа с датами в SQLITE? Перечислите распознаваемые форматы дат.

Нет, но вместо этого у SQLite3 есть функции, способные работать с датой и временем в том случае, если будет соблюден **формат записи даты и времени**. Обратим ваше внимание на то, что дата и время в SQLite3 может храниться с классом данных TEXT, REAL и INTEGER, для

каждого из этих классов есть определенный формат записи даты и времени:

Класс данных	Формат времени
TEXT	Дата в формате "YYYY-MM-DD HH:MM:SS.SSS"
REAL	В вещественном формате дата и время должны быть записаны, как число дней с момента отсчета юлианского календаря, то есть число дней с полудня 24 ноября, 4714 г. до нашей эры. Считайте дни и записывайте число.
INTEGER	Количество секунд с 1970-01-01 00:00:00 UTC

Вы спокойно можете выбрать любой из форматов хранения даты и времени в базе данных SQLite3, а при помощи встроенных функций отдавать клиенту дату и время в том формате, в котором ему будет удобно.

Для работы с датой и временем SQLite предлагает 5 встроенных функций:

1. **date**(*timestring, modifier, modifier, ...*) - возвращает дату в формате: YYYY-MM-DD
2. **time**(*timestring, modifier, modifier, ...*) - возвращает время в формате HH:MM:SS
3. **datetime**(*timestring, modifier, modifier, ...*) - возвращает "YYYY-MM-DD HH:MM:SS"
4. **julianday**(*timestring, modifier, modifier, ...*) - возвращает число дней с полудня 24 ноября, 4714 г. до нашей эры
5. **strftime**(*format, timestring, modifier, modifier, ...*) - возвращает дату, отформатированную в соответствии со строкой формата, указанной в качестве первого аргумента

103. Какие операторы используются для сравнения строк? Поясните каждый из них.

В SQLite3 есть три встроенных функции, которые выполняют сравнение строк: BINARY, NOCASE и RTRIM:

1. Функция **BINARY** сравнивает строки при помощи функции memcmp(), ее результат не зависит от кодировки, поскольку происходит побайтное сравнение.
2. Принцип работы **NOCASE** такой же, как и у BINARY, за исключением первых 26-ти прописных букв ASCII, которые перед сравнением преобразуются в свои эквиваленты в нижнем регистре.

3. **RTRIM** работает, как и **BINARY**, но откидывает пробелы в конце строки.

104. Укажите функции для работы со строками в SQLITE.

INSTR

SQLite функция instr возвращает позицию подстроки в строке.

LENGTH

SQLite функция length возвращает длину указанной строки (измеряется в символах).

LOWER

SQLite функция lower преобразует все символы в указанной строке в нижний регистр. Если в строке есть символы, которые не являются буквами, эта функция их не затрагивает.

LTRIM

SQLite функция ltrim удаляет все указанные символы из левой части строки.

REPLACE

SQLite функция replace заменяет все вхождения указанной строки.

RTRIM

SQLite функция rtrim удаляет все указанные символы из правой части строки.

SUBSTR

SQLite функция substite позволяет извлечь подстроку из строки.

TRIM

SQLite функция trim удаляет все указанные символы с обеих сторон строки.

UPPER

SQLite функция upper преобразует все символы в указанной строке в верхний регистр. Если в строке есть символы, которые не являются буквами, эта функция их не затрагивает.

105. Какие виды индексов используются в SQLITE? Укажите правила создания индексов.

Индексы в базах данных SQLite являются объектами базы данных, это означает, что имя индекса должно быть уникальным во всей базе данных.

Композитные - несколько индексов для одной таблицы

Частичные - индексы не для всех значений столбца (не для всех строк

в таблице)

106. Каким образом принимается решение о использовании индекса при запросе к таблице БД?

Сделаем выводы: индексы в базах данных ускоряют запросы на выборку данных за счет уменьшения операций по сравнению значений, но замедляют другие операции манипуляции с данными.

Следует избегать индексов, если

- ◆ Таблицы небольшие
- ◆ Таблицы часто меняются
- ◆ Столбцы, которые часто используются или имеют большое количество значений NULL

107. Что такое представление в SQLITE?

VIEW – это именованный запрос **SELECT**, который хранится в базе данных. Каждый раз, когда мы обращаемся к VIEW, СУБД выполняет этот запрос **SELECT**, а следом за ним, она выполняет наш запрос.

108. Допускается ли выполнение операторов INSERT, DELETE, UPDATE для представлений в SQLITE?

представления в SQLite нельзя редактировать, их можно только создавать, удалять и делать выборку из VIEW

Заметим, что для работы с представлениями в SQLite реализован специальный **INSTEAD OF** триггер, который можно реализовать для одной из трех команды манипуляции данными: **UPDATE**, **INSERT** и **DELETE**.

109. Поясните назначение ключевого слова TEMP при создании объекта БД. Для каких объектов допустимо применение ключевого слова TEMP?

Если мы хотим создать временную таблицу в базе данных SQLite, то необходимо использовать ключевое слово **TEMP** или **TEMPORARY**.

Таблицы, представления

110. Что такое триггер? Перечислите все известные вам виды триггеров в SQLITE.

Триггер – это особая разновидность хранимых процедур в базе данных. Особенность триггеров заключается в том, что **SQL код**, написанные в теле триггера, будет исполнен после того, **как в базе данных произойдет какое-либо событие**.

Итак, триггеры можно разделить на три вида по их применению:

- ♦ **триггер BEFORE**, который срабатывает до выполнения какого-либо события в базе данных;
- ♦ **триггер AFTER**, который срабатывает после выполнения события в базе данных;
- ♦ **INSTEAD OF триггер**, который используется для манипуляции данными представлений.

Так же мы можем разделить триггеры по типам SQL команд:

- ♦ **DELETE триггер**. Триггер DELETE запускается при попытке **удаления данных/строк из таблицы базы данных**;
- ♦ **UPDATE триггер**. Триггер UPDATE будет запущен при попытке **обновления/модификации данных в таблице базы данных**;
- ♦ **INSERT триггер**. Триггер INSERT будет запущен в том случае, если вы попытаетесь **вставить/добавить строку в таблицу базы данных**.

111. Допускается ли применение триггеров для представлений в SQLITE? Если да, то каких?

Триггеры могут быть назначены для представлений с целью расширить набор операций манипуляции данными того или иного представления. Такой вид триггеров получил название **INSTEAD OF триггер**.

Общий синтаксис триггера INSTEAD OF UPDATE для обновления данных представления можно записать как:

```
1 CREATE TRIGGER trigg_name
2
3 INSTEAD OF UPDATE OF column_name ON view_name
4
5 BEGIN
6
7 -- делаем команду UPDATE для таблицы, на основе которой создана VIEW
8
9 END;
```

Синтаксис триггера INSTEAD OF INSERT для добавления строк в представление выглядит так:

```
1 CREATE TRIGGER trigg_name
2
3 INSTEAD OF INSERT ON view_name
4
5 BEGIN
6
7 -- делаем команду INSERT для таблицы, на основе которой создана VIEW
8
9 END;
```

Синтаксис триггера INSTEAD OF DELETE для удаления строк из представления выглядит так:

```
1 CREATE TRIGGER trigg_name
2
3 INSTEAD OF DELETE ON view_name
4
5 BEGIN
6
7 -- делаем команду DELETE для таблицы, на основе которой создана VIEW
8
9 END;
```

112. Удаляется ли триггер, если удалена таблица, с которой он связан?

Триггеры автоматически удаляются при удалении таблицы, с которой они связаны.

113. Удаляется ли триггер, если удалена таблица, на которую есть ссылка в теле триггера?

нет, потому что триггер такие ситуации не отслеживает. При выполнении этого триггера будет ошибка

114. Как обрабатываются ошибки в триггере?

Специальная функция RAISE () может использоваться в программе-триггере для вызова исключения.

Функция RAISE может принимать два аргумента. Первый аргумент описывает действие при возникновении конфликта:

1. Значение IGNORE. Говорит SQLite3 о том, чтобы она игнорировала строку породившую конфликт и продолжала выполнение последующих операций.

2. Значение ROLLBACK. ROLLBACK говорит о том, что SQLite должна откатить все операции к исходному состоянию при возникновении конфликта. При этом пользователь может изменить и повторить запрос.
3. Значение ABORT. Данное значение похоже на ROLLBACK, но разница в том, что оно отменяет не все выполненные ранее SQL запросы, а только тот запрос, при котором возникла конфликтная ситуация.
4. Значение FAIL. Данное значение говорит СУБД о том, что нужно прервать выполнение текущей операции и сохранить результаты успешных операций, при этом операции, следующие за конфликтной, выполнены не будут.

Второй аргумент, который принимает функция триггера RAISE – это пояснение к ошибке. Пояснение – это обычная строка, которую вы вводите с клавиатуры. По сути данное пояснение является сообщением об ошибке, которая произошла в результате операции манипуляции данными.

```
CREATE TRIGGER books_result BEFORE INSERT
ON auth_book
BEGIN
SELECT RAISE(FAIL, 'Произошла ошибка, вы неправильно связали автора и книгу') FROM auth_book
WHERE (NEW.auth_id = 1 AND books_id = 2) OR (NEW.auth_id = 1 AND NEW.books_id = 3) OR
(NEW.auth_id = 2 AND NEW.books_id = 3) OR (NEW.auth_id = 2 AND NEW.books_id = 3) OR
(NEW.auth_id = 3 AND NEW.books_id = 2) OR (NEW.auth_id = 3 AND NEW.books_id = 1) OR
(NEW.auth_id = 4 AND NEW.books_id = 2) OR (NEW.auth_id = 4 AND NEW.books_id = 1);
END;
```

115. Можно ли создать несколько разных триггеров одного типа?

Для каждой операции INSERT, UPDATE, DELETE можно определить только один INSTEAD OF-триггер.

Можно определить несколько AFTER-триггеров для каждой операции (INSERT, UPDATE, DELETE).

116. В каком порядке выполняются триггеры?

1. BEFORE statement
2. BEFORE statement for each row (insert, update, delete)
3. AFTER statement for each row (insert, update, delete)
4. AFTER statement

117. Что такое рекурсивные и вложенные триггеры?

Термин **вложение** триггеров указывает на то, могут ли триггеры, вызываемые инструкциями DML, вызывать другие триггеры. Например, если параметр сервера Nested Triggers включен и некоторый триггер обновляет таблицу A, а эта таблица сама имеет триггер, то и он тоже будет вызван.

Рекурсивный триггер является уникальным типом вложенного триггера AFTER. Если триггер выполняет инструкцию DML, которая снова вызывает его самого, он называется **рекурсивным**. Триггер считается рекурсивным только в том случае, когда он напрямую инициирует самого себя.

118. Как определить, можно ли использовать рекурсивные триггеры?

```
PRAGMA recursive_triggers = ON;
```

119. Что такое внутренние таблицы SQLite?

Внутренняя таблица, создается с помощью команды ANALYZE. Используется для хранения справочной информации о таблицах и индексах, которой может воспользоваться планировщик для поиска эффективного способа выполнения запросов.

120. Перечислите известные вам внутренние таблицы SQLite и опишите, какие данные из них можно извлечь?

SQLITE_MASTER

В данной таблице одна строка – это один объект базы данных. В дополнение к пользовательским объектам в «sqlite_master» хранятся и внутренние объекты базы, за исключением самой таблицы «sqlite_master».

SQLITE_SEQUENCE

Внутренняя таблица, необходимая для реализации **AUTOINCREMENT**. Для каждой пользовательской таблицы использующей инкремент, соответствует строка таблицы «sqlite_sequence».

SQLITE_STAT1

Используется для хранения справочной информации о таблицах и индексах, которой может воспользоваться планировщик для поиска эффективного способа выполнения запросов.

SQLITE_STAT2

Устаревшая внутренняя таблица для версий SQLite 3.6.18 – 3.7.8. Содержала дополнительную информацию о распределении ключей.

SQLITE_STAT3

Внутренняя таблица для версий SQLite 3.7.9 и выше. Используется, если база собрана с параметром SQLITE_ENABLE_STAT3 или SQLITE_ENABLE_STAT4. Содержит дополнительную информацию о распределении ключей в пределах индекса.

SQLITE_STAT4

Внутренняя таблица для версий SQLite 3.8.1 и выше. Используется, если база собрана с параметром SQLITE_ENABLE_STAT4. Содержит дополнительную информацию о распределении ключей в пределах индекса.

SQLITE_STAT3 vs SQLITE_STAT4

sqlite_stat4 является обобщением таблицы sqlite_stat3. В таблице sqlite_stat3 предоставляется информация о крайнем левом столбце индекса, тогда как таблица sqlite_stat4 предоставляет информацию обо всех столбцах индекса.