**1. Понятие базы данных, СУБД, виды БД.**

**База данных** (БД) - это совокупность специальным образом организованных данных хранимых в памяти вычислительной системы, и отражающих состояние и взаимодействие объектов в предметной области

**Средство управления базой данных** (СУБД) – специальное ПО с помощью которого пользователи могут определять, создавать и поддерживать БД и осуществлять к ней контролируемый доступ.

Виды БД:

1. **Иерархическая** – в таких БД вся информация представлена в виде деревьев, узлы которых называются *записями*. Например, “Подразделение -> Дочернее предприятие -> Сотрудник №Х”. Корневая запись каждого такого дерева обязательно содержит уникальный ключ. Ключи некорневых записей должны быть уникальны только в пределах своего дерева. Каждая запись идентифицируется полным сцепленным ключом (совокупностью ключей всех записей от корня дерева к ней). Каждый экземпляр дерева называется *групповым отношением*, его корень - *владельцем* группового отношения, а остальные записи - его *членами*.

Минусы: нельзя реализовать отношение “многие-ко-многим”, не внося избыточность данных; запросы на поиск информации неоднозначны; медленный поиск. Пример иерархической БД -- реестр Windows

1. **Сетевая** - такие БД тоже состоят из групповых отношений, владельца и членов, однако запись может быть членом более одного группового отношения. Главное достоинство: исключение дублирования данных. Главный недостаток: сложность администрирования; медленное выполнение запросов.
2. **Реляционная** - для хранения информации используются таблицы. Подробности см. в вопросах 8 и 9
3. **Постреляционная** - развивает реляционные БД и допускает вложенные таблицы. На практике используется редко
4. **Объектно-ориентированная** - лучший вид БД, хранящий сущности как объекты в стиле ООП. На данный момент распространение этого вида сдерживает отсутствие строгой модели и наличие огромного кол-ва реляционных БД

**2. Определение/Функции СУБД**

**Назначение СУБД**

**1. Управление данными во внешней памяти**

**2. Управление буферами оперативной памяти**

**3. Управление транзакциями**

**4. Журналирование**

**Журнал - особая часть БД, недоступная пользователям СУБД в которую поступают записи о всех изменениях**

**5. Поддержка языков БД**

**Бывают 2 типов.**

1. **Язык управления БД (определяет схемы БД)**
2. **Язык манипулирования данными БД**

**6. Поддержка словаря данных**

**Содержит описание, методологию логических элементов БД (таблиц, полей)**

**3. Архитектура БД. Файловые/клиент-серверные БД**

В настоящее время используется 3-уровневая архитектура БД. Ее цель: отделить пользовательское представление БД от физического. Для обеспечения этого вводят концептуальную модель данных, отражающую для пользователя информационное содержание БД и ее логическую структуру, но скрывающую физическую реализацию.

3-уровневая архитектура состоит из:

1. Внутренней модели (физического уровня) - на этом уровне БД представлена в виде совокупности файлов, хранимых с известной структурой записей, с определенными связями между ними и извлекаемых методами доступа СУБД к этим записям.
2. Концептуальной модели (промежуточного уровня), описывающую:  
   а) Все сущности, атрибуты и их связи между собой  
   б) Ограничения, накладываемые на данные  
   в) Семантическую информацию о данных
3. Внешней модели (пользовательского уровня) - этот уровень разделяет логическое представление БД на несколько, уникальных для каждого пользователя. Он предоставляет конкретному пользователю только те свойства и связи БД, которые необходимы ему для работы.

В файл-серверных БД клиенту передается файл, который он изменяет у себя на устройстве и возвращает обратно БД в измененном виде. Это создает накладные расходы, связанные с передачей данных, однако не требует большой мощности сервера.

В клиент-серверных БД запросы клиента обрабатываются непосредственно на сервере, что позволяет обрабатывать одновременно запросы нескольких пользователей, снижает расходы на передачу данных, но при этом требует большой мощности сервера.

**4. Инфологическое проектирование БД. Диаграмма «Сущность-Связь». Сущность.**

**Инфологическое (концептуальное) проектирование** – процесс создания концептуальной (инфологической) модели данных, связанной с их смысловым содержанием и независящей от способа их представления.

Инфологическое проектирование решает 3 задачи:

1. Уточнение объектов предметной области
2. Отношений между ними
3. Их характеристик

**Сущность** — это любой конкретный или абстрактный объект в рассматриваемой предметной области, информацию о котором необходимо хранить в базе данных

Сущности бывают **слабые** (зависимые от других сущностей, обозначаются двойным прямоугольником на ER-диаграмме) и **сильные** (независимые, обозначаются обычным прямоугольником)

**5. Инфологическое проектирование БД. Атрибут. Виды атрибутов. Ключевые атрибуты.**

**Атрибут —** это именованная характеристика сущности, принимающая некоторый набор значений. Набор всех возможных значений атрибута называется **доменом**

Атрибуты бывают **простыми** (атомарными) и **составными** (состоят из других атрибутов)

Атрибуты бывают **однозначными** (с одним значением одновременно), **многозначные** (с несколькими значениями одновременно) и **производными** (зависящими от значений других атрибутов)

Наименование атрибута должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей. Атрибуты используются для определения того, какая информация должна быть собрана о сущности.

**Ключ** или ключевой атрибут — атрибут (столбец) или набор атрибутов, который однозначно идентифицирует сущность/объект/таблицу в базе данных. Ключи бывают:

1. *Первичный* (PK) — ключ, который используется для идентификации объекта.
2. *Потенциальный* (или альтернативный, или ключ-кандидат) (AK) — ключ, по каким-либо причинам неиспользуемый как первичный.
3. *Составной* — ключ, который использует несколько атрибутов.
4. *Суррогатный* — ключ, значение которого генерируется СУБД.

**6. Инфологическое проектирование БД. Связь. Показатель кардинальности.**

**Связь** — осмысленная ассоциация между сущностями. На ER-диаграмме отображается как ромбик. Связи необходимы для поддержания целостности данных.

Связи бывают **слабые** (могут отсутствовать) и **сильные** (есть всегда)

**Кардинальность** связи описывает кол-во возможных связей для каждой сущности-участника связи. Наиболее распространены следующие виды кардинальностей:

1. «Один-к-одному» (1:1) — одному экземпляру сущности А соответствует 1 или 0 экземпляров сущности B
2. «Один-ко-многим» (1:N) — одному экземпляру сущности А соответствует 0, 1 или несколько экземпляров сущности B
3. «Многие-ко-многим» (N:M) — является комбинацией связей «один-ко-многим» и «многие-к-одному»:

**7. Моделирование локальных представлений. Способы объединения моделей локальных представлений.**

**8. Реляционная модель данных. Атрибуты, домены, кортежи, отношения.**

В реляционной БД все объекты представлены отношениями.

**Отношение** - множество кортежей, соответствующих одной схеме отношений. Отношения обладают следующими свойствами:

1. Не содержат 2-ух одинаковых кортежей (по определению множества)
2. Кортежи и атрибуты не упорядочены (по определению множества)

**Схема отношений** - это именованное множество упорядоченных пар (имя атрибута, имя домена). Мощность этого множества - степень отношения

**Кортеж** - множество пар (имя атрибута, значение атрибута), содержащее по одному вхождению каждого имени атрибута из схемы отношений.

**Атрибут** — именованный неделимый элемент данных. Множество его допустимых значений - домен, на котором определен атрибут. При этом разные атрибуты могут иметь одинаковые домены

**Схема БД** - набор именованных схем отношений

**Реляционная БД** - набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

**9. Реляционная модель данных. Потенциальные ключи. Первичный ключ. Внешний ключ.**

**Потенциальный ключ** - это подмножество атрибутов отношения, обладающее следующими свойствами:

1. *Уникальность* - не существует одинаковых значений потенциального ключа
2. *Безызбыточность* - потенциальный ключ не содержит потенциальных ключей

Потенциальный ключ бывает **простым** и **составным**.

Один из потенциальных ключей выбирается как **первичный** (PK) и служит для идентификации отношения в системе отношений. Первичный ключ используется в качестве общих столбцов для таблиц. Если такой атрибут выделить нельзя, то вводят *суррогатный* ключ, автоматически заполняющийся уникальными значениями и никогда не изменяющийся.

Ни в одном из потенциальных ключей NULL-значения недопустимы

Внешний ключ (FK) - такое подмножество атрибутов дочернего отношения, что для любого его непустого значения обязательно найдется равное значение первичного ключа главного родительского отношения. Внешний ключ используется в дочернем отношении. Значения внешних ключей не обязаны быть уникальными в своем отношении. При этом NULL-значения допускаются

**10. Нормальные формы. (1-5NF + BCNF)**

**Отношение находится в 1НФ**, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.

**Отношение находится во 2НФ**, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа(ПК).

Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.

**Отношение находится в 3НФ**, когда находится во 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.

**Требования нормальной формы Бойса-Кодда**

Требования нормальной формы Бойса-Кодда следующие:

* Таблица должна находиться в третьей нормальной форме. Здесь все как обычно, т.е. как и у всех остальных нормальных форм, первое требование заключается в том, чтобы таблица находилась в предыдущей нормальной форме, в данном случае в третьей нормальной форме;
* Ключевые атрибуты составного ключа не должны зависеть от неключевых атрибутов.

**Отношение находится в 4НФ**, если оно находится в НФБК и все нетривиальные многозначные зависимости фактически являются функциональными зависимостями от ее потенциальных ключей.

В отношении R (A, B, C) существует многозначная зависимость R.A -> -> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от С.

**Отношения находятся в 5НФ**, если оно находится в 4НФ и отсутствуют сложные зависимые соединения между атрибутами.Если «Атрибут\_1» зависит от «Атрибута\_2», а «Атрибут\_2» в свою очередь зависит от «Атрибута\_3», а «Атрибут\_3» зависит от «Атрибута\_1», то все три атрибута обязательно входят в один кортеж.

**11. Реляционная алгебра и ее операции.**

Реляционная алгебра базируется на теории множеств и является основой логики работы баз данных.

Операции:

1. Объединение - операция над двумя отношениями, в результате которой получается новое отношение, состоящее из всех кортежей исходных отношений. Общие для исходных отношений кортежи в новом отношении встречаются только по одному разу.
2. Пересечение - операция над двумя отношениями, в результате которой получается новое отношение, состоящее из кортежей, принадлежащих обоим исходным отношениям.
3. Разность - операция над двумя отношениями, в результате которой получается новое отношение, состоящее из кортежей, принадлежащих первому отношению и не принадлежащих второму.
4. Декартово произведение - операция над двумя отношениями, в результате которой получается новое отношение, состоящее из всех возможных кортежей, являющихся попарными сочетаниями кортежей исходных отношений
5. Проекция - также производится над кортежами одного отношения. Результат проекции - новое отношение содержащее только заданные атрибуты исходного отношения.
6. Операция выборки - производится над кортежами одного отношения. Результат выборки - новое отношение, состоящее из котежей исходного отношения, удовлетворяющих заданному условию.
7. Операция соединения - операция над двумя отношениями, имеющими общие атрибуты, в результате которой получается новое отношение, состоящее из всех атрибутов исходных отношений и объединяющее только те кортежи исходных отношений в которых значения общих атрибутов совпадают.

**12. Функциональные зависимости. Математические свойства ФЗ, теоремы.**

**Функциональная зависимость** - атрибут Y зависит от атрибута Х, если в любой момент времени каждому значению Х соответствует одно значение Y. При этом Х и Y могут представлять не только единичные атрибуты, но и группу или подмножество, состоящее из нескольких атрибутов одного отношения.

Подмножество атрибутов Х называется детерминантом функции зависимости.

Математические свойства ФЗ, теоремы:

1. рефлективность

Если Y является подмножеством Х, то Х определяет Y

1. дополнение

А->В => AC -> BC, где С - любое подмножество атрибутов отношения

1. транзитивность

А->В

1. самоопределение

Х->Х

1. декомпозиция

А->ВС => А->В, А->С

1. композиция

А->В и С->D => АС->ВD

1. Теорема о всеобщей зависимости или теорема всеобщего объединения:

Если А->В и С->D => А + (С-В) -> BD

Правила Армстронга позволяют выводить новые ФЗ на основе других ФЗ. Применяя их, можно вывести множество всех ФЗ данного отношения. Такой множество называется **замыканием.**

Для каждого отношения можно найти такое множество всех ФЗ, в котором ни одна зависимость не может быть выведена из другой. Такое множество ФЗ заданных отношений называется **неприводимым**.

Доказано, что неприводимое множество ФЗ должно обладать следующими свойствами:

1. в правой части ФЗ должен быть только один атрибут
2. из левой части каждой ФЗ нельзя удалить ни одного атрибута без потери этой ФЗ.

Такое множество для каждого отношения может быть только одно.

**Нежелательные ФЗ:**

Некоторые ФЗ, входящие в неприводимое множество ФЗ отношения, являются нежелательными, т.к. приводят к дублированию информации, что создает проблемы обновления данных и представляет угрозу целостности БД.

**13. Декомпозиция отношений. Декомпозиция без потерь.**

**Декомпозиция -** разбиение имеющегося отношения, в результате которого количество отношений в БД увеличивается, но общее количество хранимых данных, как правило, сохраняется за счет устранения дублирования данных.

**Декомпозиция без потерь:**

Декомпозицией отношения также можно считать взятие одной или нескольких проекций исходного отношения так, чтобы эти проекции в совокупности содержали все атрибуты исходного отношения.

**Теорема Хеза**: Пусть R(A,B,C) - отношение, А, В, С - атрибуты или множество атрибутов этого отношения. Если имеется ФЗ А->В, то проекции R1(A,B) и R2(A,C) образуют декомпозицию без потерь. Другими словами, любую ФЗ отношения можно вынуть(?) из отношения, оставив ее детерминант в исходном отношении. При этом никакая информация не будет утеряна.

**14. Нормализация БД. Преимущества/недостатки.**

*Нормализация* представляет собой процесс, направленный на уменьшение избыточности информации в базе данных. Кроме самих данных, в базе данных также могут быть нормализованы различные наименования, имена объектов и выражения.

*Нормализация* — это процесс, направленный на уменьшение избыточности информации в реляционной базе данных.

Основные преимущества нормализации:

* Лучшая общая организация базы данных
* Сокращение избыточности информации
* Непротиворечивость информации внутри базы данных
* Более гибкий проект базы данных
* Большая безопасность данных

Недостатком нормализованной БД является необходимость считывать связанные данные из нескольких таблиц при выполнении одного запроса, что негативно сказывается на производительности, т.к. замечено, что ненормализованные или не вполне нормализованные данные отыскиваются быстрее, если они хранятся в одной таблице, по сравнению со случаем поиска данных в связанных таблицах. Подобное ускорение тем заметнее, чем больше число записей в связанных таблицах.

**15. Транзакции. Свойства транзакций. Журнал транзакций.**

**Транзакция** - единица работы СУБД, которая может быть выполнена либо целиком, либо вообще не выполнена.

Объем транзакций может варьироваться от одного оператора SQL до всех действий, выполненных с БД.

Свойства:

1. атомарность - результаты всех операций, успешно выполненных в пределах транзакции, должны быть отражены в состоянии БД либо не должно быть отражено действие ни одной операции
2. согласованность - транзакции могут быть успешно завершены с фиксацией результатов своих операций только в том случае, когда действия операций не нарушают целостности БД, т.е. удовлетворять набору ограничений целостности, определенных для этой БД
3. изоляция - требует, чтобы две одновременно выполняемые транзакции никаким образом не действовали одна на другую. Т.е. результаты выполнения операции транзакция1 не должны быть видны никакой транзакции2 до тех пор, пока транзакция1 не завершится успешным образом
4. долговечность - после успешного завершения транзакции все изменения, которые были внесены в состояние БД операциями транзакции, должны гарантированно сохраниться даже в случае сбоев аппаратуры или ПО

Журнализация транзакций:

Ведение журналов транзакций преследует две цели:

1. возможность отката транзакции
2. восстановление БД в случае аварийной ситуации или сбоев

Журналы делятся на Undo и Redo.

Undo-журналы: используются для отката и ведутся для каждой транзакции отдельно. Как только очередная транзакция зафиксирована или откачена, то информация из соответствующего Undo-журнала удаляется. Рекомендуемой количество Undo-журналов в БД = n/2, где n-количество одновременно работающих пользователей

Redo-журналы: необходимы для повторного выполнения транзакций при восстановлении данных. Это единый системный журнал, в котором записывается результат всех зафиксированных транзакций. В аварийной ситуации, которая привела к порче данных, они восстанавливаются из резервной копии. Все транзакции, которые прошли в системе после снятия резервной копии, восстанавливаются по Redo-журналу. Такой механизм полностью гарантирует выполнение свойства долговечности. Для того, чтобы гарантировать возможность безусловного восстановления всех зафиксированных транзакций, принято правило **упреждающей записи** в журнал транзакций. По команде COMMIT результат транзакции сначала заносится в Redо-журнал, а только затем записывается в табличное пространство.

**16. Разграничение доступа. Привилегии и роли.**

**Разграничение доступа:**

Создание учетных записей пользователей:

* CREATE USER имя\_пользователя
* IDENTIFIED BY пароль
* ALTER USER имя
* DROP USER, можно указать опцию CASCADE
* CREATE SCHEMA имя\_пользователя

**Привилегии и роли:**

Привилегии позволяют использовать определенные команды языка SQL по отношению к некоторым объектам.

* GRANT

Пример: GRANT SELECT ON students TO PUBLIC - задать управление командой SELECT всем пользователям

Роль - именование группы привилегий, которые можно предоставить пользователям с помощью команды GRANT.

CREATE ROLE имя\_роли

**17. Понятие целостности в реляционной модели.**

Понятия реляционной целостности:

* определитель NULL;
* целостность сущностей;
* ссылочная целостность;
* корпоративные ограничения целостности.

Определитель NULL. Значение Null обозначает тот факт, что значение не определено. Null не принадлежит никакому типу данных и может присутствовать среди значений любого атрибута, определенного на любом типе данных. Двуместная «арифметическая» операция с Null даёт Null. Операция сравнения с Null даёт UNKNOWN.

Целостность сущностей. Требование целостности сущности означает, что первичный ключ должен полностью идентифицировать каждую сущность, а поэтому в составе любого значения первичного ключа не допускается наличие неопределенных значений. Значение атрибута должно быть атомарным.

Ссылочная целостность. Требование целостности по ссылкам состоит в том, что для каждого значения внешнего ключа, появляющегося в кортеже значения-отношения ссылающейся переменной отношения, либо в значении-отношении переменной отношения, на которую указывает ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть полностью неопределенным. Существуют правила удаления кортежа из отношения, на которое ведет ссылка.

Ограничение удаления. Запрещается производить удаление кортежа, для которого существуют ссылки. Сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа

**8 (вар. №2). Реляционная модель данных. Атрибуты, домены, кортежи, отношения.**

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы)

Реляционная модель данных предусматривает структуру данных, обязательными объектами которой являются:

отношение;

атрибут;

домен;

кортеж;

степень;

кардинальность;

первичный ключ.

Отношение - это плоская (двумерная) таблица, состоящая из столбцов и строк

Атрибут - это поименованный столбец отношения.

Домен - это набор допустимых значений для одного или нескольких атрибутов.

Кортеж - это строка отношения.

Степень определяется количеством атрибутов, которое оно содержит

Кардинальность - это количество кортежей, которое содержит отношение.

Первичный ключ - это уникальный идентификатор для таблицы.

Соответствие между формальными терминами реляционной модели данных и неформальными:

* отношение (формальный термин) - таблица (неформальный термин);
* атрибут - столбец;
* кортеж - строка или запись;
* степень - количество столбцов;
* кардинальное число - количество строк;
* первичный ключ - уникальный идентификатор;
* домен - общая совокупность допустимых значений.

**Свойства отношений:**

* уникальное имя отношения;
* уникальное имя атрибута;
* нет одинаковых кортежей;
* кортежи не упорядочены сверху вниз;
* атрибуты не упорядочены слева направо;
* все значения атрибутов атомарные (нормализованное отношение).

**Виды отношений:**

* именованное отношение;
* базовое отношение;
* производное отношение;
* выражаемое отношение;
* представление (view);
* снимки (snapshot);
* результат запроса;
* промежуточный результат.

Именованное отношение - это переменная отношения, определённая в СУБД (системе управления базами данных) посредством оператора **CREATE** (CREATE TABLE, CREATE BASE RELATION, CREATE VIEW, CREATE SNAPSHOT).

Базовое отношение - это именованное отношение, которое не является производным. Существование базового отношения не зависит от существования других отношений.

Производное отношение - это отношение, которое определено через другие именованные отношения. Производное отношение зависит от существования других - базовых - отношений.

Выражаемое отношение - это отношение, которое можно получить из набора именованных отношений посредством некоторого реляционного выражения. Каждое именованное отношение является выражаемым отношений, но не наоборот. Примеры выражаемых отношений - базовые отношения, представления, снимки, промежуточные и окончательные результаты. Множество всех выражаемых отношений - это множество всех базовых и всех производных отношений.

Представление - это именованное производное отношение. Представлены в базе данных в виде определения. Представление не хранится в физической памяти системы управления базой данных (СУБД), а формируется с использованием других именованных отношений.

Снимки (snapshot) - это то же, что и представление, но с физическим сохранением и с периодическим обновлением.

Результат запроса - это неименованное производное отношение. СУБД не обеспечивает постоянного существования результатов запросов. Для сохранения результата запроса его можно присвоить какому-либо именованному отношению.

Промежуточный результат - это неименованное производное отношение, являющееся результатом подзапроса, вложенного в бОльшее выражение.

**9 (вар. №2). Реляционная модель данных. Потенциальные ключи. Первичный ключ. Внешний ключ.**

Реляционная модель данных - это способ рассмотрения данных, то есть предписание для способа представления данных (посредством таблиц) и для способа работы с таким представлением (посредством операторов). Она связана с тремя аспектами данных: структурой (объекты), целостностью и обработкой данных (операторы)

Ключи отношения могут быть следующми:

* суперключ;
* потенциальный ключ;
* первичный ключ;
* внешний ключ;
* суррогатный ключ.

Ключ отношения - это подсхема исходной схемы отношения, состоящая из одного или нескольких атрибутов, для которых декларируется условие уникальности значений в кортежах отношений. При объявлении схемы базового отношения могут быть заданы объявления нескольких ключей.

Ключ отношения может быть простым или составным. Простой ключ – это ключ, состоящий из одного и не более атрибута. Составной ключ -ключ, состоящий из двух и более атрибутов.

Суперключ - это атрибут или множество атрибутов, которое единственным образом идентифицирует кортеж данного отношения. Он может включать дополнительные атрибуты. Суперключ не обладает свойством неизбыточности.

Потенциальный ключ - это подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и неизбыточности. Он обладает следующими свойствами. Уникальность: в таблице нет двух разных строк с одинаковыми значениями в нашем потенциальном ключе. Неизбыточность: нельзя убрать один из столбцов из ключа, так, чтобы он не потерял уникальности. В отношении может быть больше одного потенциального ключа.

Первичный ключ (primary key, PK) - это один из потенциальных ключей отношения, выбранный в качестве основного ключа. Допустимо объявление одного и только одного первичного ключа. Атрибуты первичного ключа не могут принимать значения Null.

Внешний ключ (foreign key, FK) - это ключ, объявленный в базовом отношении, который при этом ссылается на первичный того же самого или какого-то другого базового отношения.

Суррогатный ключ - это служебный атрибут, добавленный к уже имеющимся информационным атрибутам отношения. Предназначение суррогатного ключа - служить первичным ключом отношения. Значение этого атрибута генерируется искусственно.

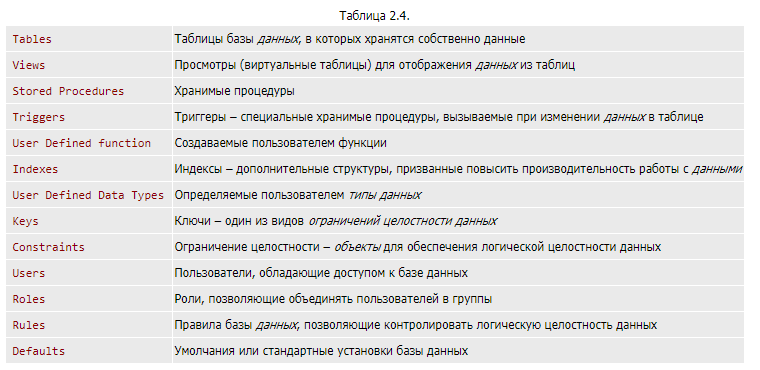
**18. Триггеры. Пример.**

**19. Представления (View). Создание. Удаление. Обновление.**

**20. Вложенные запросы. Кореллированные и некореллированные запросы.**

**21. Основные объекты БД. Определение.**

К основным объектам базы данных SQL Server относятся объекты, представленные в [таблице 2.4](https://intuit.ru/studies/courses/5/5/lecture/124?page=4#table.2.4).

****

Приведем краткий обзор основных объектов баз данных.

#### Таблицы

Все данные в SQL содержатся в объектах, называемых таблицами. Таблицы представляют собой совокупность каких-либо сведений об объектах, явлениях, процессах реального мира. Никакие другие объекты не хранят данные, но они могут обращаться к данным в таблице. Таблицы в SQL имеют такую же структуру, что и таблицы всех других СУБД и содержат:

* cтроки; каждая строка (или запись) представляет собой совокупность атрибутов (свойств) конкретного экземпляра объекта ;
* cтолбцы; каждый столбец (поле) представляет собой атрибут или совокупность атрибутов. Поле строки является минимальным элементом таблицы. Каждый столбец в таблице имеет определенное имя, тип данных и размер.

#### Представления

Представлениями (просмотрами) называют виртуальные таблицы, содержимое которых определяется запросом. Подобно реальным таблицам, представления содержат именованные столбцы и строки с данными. Для конечных пользователей представление выглядит как таблица, но в действительности оно не содержит данных, а лишь представляет данные, расположенные в одной или нескольких таблицах. Информация, которую видит пользователь через представление, не сохраняется в базе данных как самостоятельный объект.

#### Хранимые процедуры

Хранимые процедуры представляют собой группу команд SQL, объединенных в один модуль. Такая группа команд компилируется и выполняется как единое целое.

#### Триггеры

Триггерами называется специальный класс хранимых процедур, автоматически запускаемых при добавлении, изменении или удалении данных из таблицы.

#### Функции

Функции в языках программирования – это конструкции, содержащие часто исполняемый код. Функция выполняет какие-либо действия над данными и возвращает некоторое значение.

#### Индексы

Индекс – структура, связанная с таблицей или представлением и предназначенная для ускорения поиска информации в них. Индекс определяется для одного или нескольких столбцов, называемых индексированными столбцами. Он содержит отсортированные значения индексированного столбца или столбцов со ссылками на соответствующую строку исходной таблицы или представления. Повышение производительности достигается за счет сортировки данных. Использование индексов может существенно повысить производительность поиска, однако для хранения индексов необходимо дополнительное пространство в базе данных.

#### Пользовательские типы данных

Пользовательские типы данных – это типы данных, которые создает пользователь на основе системных типов данных, когда в нескольких таблицах необходимо хранить однотипные значения; причем нужно гарантировать, что столбцы в таблице будут иметь одинаковый размер, тип данных и чувствительность к значениям NULL .

#### Ограничения целостности

Ограничения целостности – механизм, обеспечивающий автоматический контроль соответствия данных установленным условиям (или ограничениям). Ограничения целостности имеют приоритет над триггерами, правилами и значениями по умолчанию. К ограничениям целостности относятся: ограничение на значение NULL, проверочные ограничения, ограничение уникальности (уникальный ключ), ограничение первичного ключа и ограничение внешнего ключа. Последние три ограничения тесно связаны с понятием ключей.

#### Правила

Правила используются для ограничения значений, хранимых в столбце таблицы или в пользовательском типе данных. Они существуют как самостоятельные объекты базы данных, которые связываются со столбцами таблиц и пользовательскими типами данных. Контроль значений данных может быть реализован и с помощью ограничений целостности.

#### Умолчания

Умолчания – самостоятельный объект базы данных, представляющий значение, которое будет присвоено элементу таблицы при вставке строки, если в команде вставки явно не указано значение для этого столбца.

**22. Команды манипулирования данными. SELECT.**

Язык SQL предназначен для манипулирования данными в реляционных базах данных, определения структуры баз данных и для управления правами доступа к данным в многопользовательской среде.

Поэтому, в язык SQL в качестве составных частей входят:

* язык манипулирования данными (Data Manipulation Language, DML)
* язык определения данных (Data Definition Language, DDL)
* язык управления данными (Data Control Language, DCL).

Подчеркнем, что это не отдельные языки, а различные команды одного языка. Такое деление проведено только лишь с точки зрения различного функционального назначения этих команд.

Язык манипулирования данными используется, как это следует из его названия, для манипулирования данными в таблицах баз данных. Он состоит из 4 основных команд:

Select (выбрать)

Для извлечения данных из БД применяется команда **SELECT**. В упрощенном виде она имеет следующий синтаксис: SELECT список\_столбцов FROM имя\_таблицы

Получим все объекты из этой таблицы:

SELECT \* FROM Products;

Однако использование символа звездочки \* считается не очень хорошей практикой, так как, как правило, не все столбцы бывают нужны. И более оптимальный подход заключается в указании всех необходимых столбцов после слова SELECT. Исключение составляет тот случай, когда надо получить данные по абсолютно всем столбцам таблицы. Также использование символа \* может быть предпочтительно в таких ситуациях, когда в точности не известны названия столбцов.

Если нам надо получить данные не по всем, а по каким-то конкретным столбцам, то тогда все эти спецификации столбцов перечисляются через запятую после SELECT:

Спецификация столбца необязательно должна представлять его название. Это может быть любое выражение, например, результат арифметической операции. Так, выполним следующий запрос:

SELECT ProductCount, Manufacturer, Price \* ProductCount

FROM Products;

Здесь при выборке будут создаваться три столбца. Причем третий столбец представляет значение столбца Price, умноженное на значение столбца ProductCount, то есть совокупную стоимость товара.

С помощью оператора **AS** можно изменить название выходного столбца или определить его псевдоним:

SELECT ProductCount AS Title,

Manufacturer,

Price \* ProductCount AS TotalSum

FROM Products;

**23. Команды манипулирования данными. INSERT/UPDATE/DELETE.**

Язык SQL предназначен для манипулирования данными в реляционных базах данных, определения структуры баз данных и для управления правами доступа к данным в многопользовательской среде.

Поэтому, в язык SQL в качестве составных частей входят:

* язык манипулирования данными (Data Manipulation Language, DML)
* язык определения данных (Data Definition Language, DDL)
* язык управления данными (Data Control Language, DCL).

Подчеркнем, что это не отдельные языки, а различные команды одного языка. Такое деление проведено только лишь с точки зрения различного функционального назначения этих команд.

Язык манипулирования данными используется, как это следует из его названия, для манипулирования данными в таблицах баз данных. Он состоит из 4 основных команд:

Для добавления данных применяется команда **INSERT**, которая имеет следующий формальный синтаксис:

INSERT INTO имя\_таблицы (столбец1, столбец2, ... столбецN)

VALUES (значение1, значение2, ... значениеN)

После **INSERT INTO** идет имя таблицы, затем в скобках указываются все столбцы через запятую, в которые надо добавлять данные. И в конце после слова **VALUES** в скобках перечисляются добавляемые значения.

Стоит учитывать, что значения для столбцов в скобках после ключевого слова VALUES передаются по порядку их объявления.

Если мы добавляем значения только для части столбцов, то мы можем не знать, какие значения будут у других столбцов. Например, какое значени получит столбец Id у товара. С помощью оператора **RETURNING** мы можем получить это значение:

INSERT INTO Products

(ProductName, Manufacturer, Price, ProductCount)

VALUES('Desire 12', 'HTC', 8, 21000) RETURNING id;

**24. Команды DDL для работы с таблицами.**

Язык определения данных используется для создания и изменения структуры базы данных и ее составных частей - таблиц, индексов, представлений (виртуальных таблиц), а также триггеров и сохраненных процедур. Основными его командами являются:

|  |  |
| --- | --- |
| CREATE DATABASE | (создать базу данных) |
| CREATE TABLE | (создать таблицу) |
| CREATE VIEW | (создать виртуальную таблицу) |
| CREATE INDEX | (создать индекс) |
| CREATE TRIGGER | (создать триггер) |
| CREATE PROCEDURE | (создать сохраненную процедуру) |
| ALTER DATABASE | (модифицировать базу данных) |
| ALTER TABLE | (модифицировать таблицу) |
| ALTER VIEW | (модифицировать виртуальную таблицу) |
| ALTER INDEX | (модифицировать индекс) |
| ALTER TRIGGER | (модифицировать триггер) |
| ALTER PROCEDURE | (модифицировать сохраненную процедуру) |
| DROP DATABASE | (удалить базу данных) |
| DROP TABLE | (удалить таблицу) |
| DROP VIEW | (удалить виртуальную таблицу) |
| DROP INDEX | (удалить индекс) |
| DROP TRIGGER | (удалить триггер) |
| DROP PROCEDURE | (удалить сохраненную процедуру) |

**25. Хранимые процедуры. Пример.**

**Хранимые процедуры –** это объекты базы данных, в которых заложеналгоритм в виде набора SQL инструкций. Иными словами, можно сказать, что хранимые процедуры – это программы внутри базы данных. Хранимые процедуры используются для сохранения на сервере повторно используемого кода например, Вы написали некий алгоритм, последовательный расчет или многошаговую SQL инструкцию, и чтобы каждый раз не выполнять все инструкции, входящие в данный алгоритм, Вы можете оформить его в виде хранимой процедуры. При этом, когда Вы создаете процедуру SQL, сервер компилирует код, а потом, при каждом запуске этой процедуры SQL сервер уже не будет повторно его компилировать.

Хранимые процедуры создаются с помощью инструкции CREATE PROCEDURE, после данной инструкции Вы должны написать название Вашей процедуры, затем в случае необходимости в скобочках определить входные и выходные параметры. После этого Вы пишите ключевое слово AS и открываете блок инструкций ключевым словом BEGIN, закрываете данный блок словом END. Внутри данного блока Вы пишите все инструкции, которые реализуют Ваш алгоритм или какой-то последовательный расчет, иными словами, программируете на T-SQL.

Для примера давайте напишем хранимую процедуру, которая будет добавлять новую запись, т.е. новый товар в нашу тестовую таблицу. Для этого мы определим три входящих параметра: @CategoryId – идентификатор категории товара, @ProductName — наименование товара и @Price – цена товара, данный параметр будет у нас необязательный, т.е. его можно будет не передавать в процедуру (*например, мы не знаем еще цену*), для этого в его определении мы зададим значение по умолчанию. Эти параметры в теле процедуры, т.е. в блоке BEGIN…END можно использовать, так же как и обычные переменные (*как Вы знаете, переменные обозначаются знаком @*). В случае если Вам нужно указать выходные параметры, то после названия параметра указывайте ключевое слово OUTPUT (*или сокращённо OUT*).

--Создаем процедуру

CREATE PROCEDURE TestProcedure

(

--Входящие параметры

@CategoryId INT,

@ProductName VARCHAR(100),

@Price MONEY = 0

)

AS

BEGIN

--Инструкции, реализующие Ваш алгоритм

--Обработка входящих параметров

--Удаление лишних пробелов в начале и в конце текстовой строки

SET @ProductName = LTRIM(RTRIM(@ProductName));

--Добавляем новую запись

INSERT INTO TestTable(CategoryId, ProductName, Price)

VALUES (@CategoryId, @ProductName, @Price)

--Возвращаем данные

SELECT \* FROM TestTable

WHERE CategoryId = @CategoryId

END

GO

**26. Индексы. Использование B-деревьев при хранении индексов. Операции добавления/изменения/удаления.**

Индексы бывают двух видов: простые и составные. Составной индекс включает более чем один столбец. Таким образом, можно совместно проиндексировать 2 или более столбца, каждый из которых обладает низкой селективностью, а пара их значений – высокой.

Если все столбцы, используемые запросом, входят в составной индекс, то обращение к таблицам можно вовсе избежать, все данные будут считаны только из индекса. Для эффективного использования составного индекса, необходимо, чтобы логические условия были наложены на ведущие столбцы индекса, то есть на те столбцы, которые были указаны первыми при создании индекса.

Как правило, в индексах хранятся значения индексируемых столбцов таблицы и физические адреса строк для каждого из хранимых значений столбцов

B-Дерево содержит по одному индексному элементу для каждой сроки таблицы, в которой имеется не пустое индексное значение.

B-Дерево – сбалансированное сильноветвящееся дерево во внешней памяти. С точки зрения физической организации B-Дерево представляется как мультисписочная структура страниц во внешней памяти, то есть каждому узлу дерева соответствует блок внешней памяти или страница.

Структура внутренней страницы:

N1 ключ(1) N2 ключ(2) N3 ключ(3) … Nn ключ(n) N(n+1) ключ(n+1)

Листовая страница:

ключ(1) сп(1) ключ(2) сп(2) … ключ(t) сп(t), где сп – упорядоченный список идентификаторов кортежей, включающих значения ключа.

Листовые страницы связаны одно- или двунаправленным списком.

Поиск в B-Дереве – это прохождение от корня к листу в соответствии с заданным значением ключа. Поскольку деревья сильноветвящиеся и сбалансированные, то для выполнения поиска по любому значения ключа потребуется одно и тоже, и, как правило, небольшое число обменов с внешней памятью.

В сбалансированном дереве, где длины путей от корня к листу одни и те же, если во внутренней странице помещается n ключей, то при хранении m записей, требуется дерево глубиной log𝑛𝑚. Если n – достаточно велико, то глубина дерева при этом небольшая, и производится обычный поиск.

Наиболее полезным свойством B-Деревьев является автоматическое поддержание свойства сбалансированности.

При занесении новой записи выполняется следующая последовательность действий:

1. Поиск листовой страницы – поиск по ключу. Если в B-Дереве не содержится ключ с заданным значением, то будет получен номер страницы, в которой ему надлежит содержаться, и соответствующие координаты внутри страницы;

2. Помещение записи на место. Листовая страница, в которую требуется занести запись, считывается в буфер и в нем выполняется операция вставки. Размер буфера должен превышать размер страницы внешней памяти;

3. Если после выполнения вставки новой записи размер используемой части буфера не превосходит размера страницы, то на этом выполнение операции занесения заканчивается. Буфер может быть немедленно вытолкнут во внешнюю память или временно сохранен в оперативной в зависимости от политики управления буферами.

4. Если возникло переполнение буфера, то выполняется расщепление страницы. Для этого запрашивается новая страница внешней памяти и используемая часть буфера разбивается так, чтобы другая половина также начиналась с ключа, и вторая половина записывается во вновь выделенную страницу, а в старой странице модифицируется значение размера свободной памяти. Также модифицируются ссылки по списку листовых страниц. Чтобы обеспечить доступ от корня дерева к заново заведенной странице необходимо соответствующим образом модифицировать внутреннюю страницу, являющуюся предком ранее существовавшей листовой страницы, то есть вставить в нее соответствующее значение ключа и ссылку на новую страницу. При выполнении этого действия может снова произойти переполнение – теперь уже внутренней страницы, и она будет расщеплена на две. В результате потребуется вставить значение ключа и ссылку на новую страницу во внутреннюю страницу предка выше по иерархии и так далее;

5. Предельным случаем является переполнение корневой страницы B-Дерева. В этом случае она также расщепляется на две и заводится новая корневая страница, то есть его глубина увеличивается на единицу.

При удалении записи выполняются следующие действия:

1. Поиск записи по ключу. Если запись не найдена, то удалять ничего не нужно;

2. Реальное удаление записи в буфере, в который прочитана соответствующая листовая страница. Если после выполнения этой операции размер занятой в буфере области оказывается таковым, что его сумма с размером занятой области в листовых страницах, являющихся левым или правым братом данной страницы больше чем размер страницы, операция завершается. Иначе производится слияние с левым или правым братом, то есть в буфере производится новый образ страницы, содержащий общую информацию из данной страницы и его левого или правого брата. Ставшая ненужной листовая страница заносится в список свободных страниц. Соответствующим образом корректируется список листовых страниц. Чтобы устранить возможность доступа от корня к освобожденной странице, нужно удалить соответствующее значение ключа и ссылку на освобожденную страницу из внутренней страницы, то есть ее предка. При этом может возникнуть потребность в слиянии этой страницы с ее левым или правым братом;

3. Предельным случаем является полное опустошение корневой страницы дерева, что возможно после слияния последних двух потомков корня. В этом случае корневая страница освобождается, а глубина дерева уменьшается на единицу.

**27. Индексы. Использование хэш-таблиц, битовых карт и индексных таблиц при работе с индексами.**

При этом значения ключей не модифицируются поэтому заранее можно точно определить число значений хешированных ключей и размер кластера, также ключевой столбец должен быть высокоселективными производительность выборки, вставки, обновления и удаления записей, если используются условия равенства для всех ключевых столбцов кластера. Главным преимуществом по сравнению с В-деревом является лучшая

**Битовые индексы**

Битовые индексы обеспечивают быстрое обращения к данным больших таблиц когда доступ организуется по столбцам с низкой или средней селективностью, с использованием различных сочетаний условий равенства.

Битовый индекс построен в виде двоичной карты по значениям ключа. Это означает что для каждой строки таблицы в двоичной карте, т.е. в определённом бите некоторой последовательности байтов поставлено либо 0 либо 1 в соответствии значения ключа конкретной строки.

Битовые карты нецелесообразно применять для часто обновляемых столбцов, т.к. даже простое изменение одной строки обычно приводит к копированию всей соответствующей секции битовой карты

В приведённом пример столбец пол не обновляется, так что если данные из таблицы удаляются редко то можно использовать данный индекс без снижения производительности. Таким образом можно применять индекс на основе битовых карт по отношению к любому низкоселективному столбцу, часто используемому в конструкции WHERE, при условии что набор значений этого столбца ограничен.

Битовые индексы можно использовать для поиска только по условиям равенства (равенства и операция in). Если необходим доступ по интервалу индексируемых значений то предпочтительнее использовать В-деревья

**Индекс таблицы**

Индекс таблица – это таблица которая физически построена в виде двоичного дерева относительно своего первичного ключа. Как правило это таблицы с короткими строками обращения к которым всегда происходит либо по ПК либо полным сканированием. Данные в таких таблицах отсортированы по значениям столбца ПК и сохраняются так, как если бы вся таблица целиком содержалась в одном индексе

Чтобы минимизировать объём работы необходимы для активного управления индексами, следует использовать индекс таблицу лишь в тех случаях, когда данные очень статичны. Не рекомендуется использовать индекс таблицы если строки имеют относительно большую длину, в этом случае будет лучшем выбором использование обычных таблиц и индексов

Ограничением при работе с индекс таблицами является то что их строки не имеют идентификаторов, поэтому не могут быть созданы никакие доп. индексы за исключением обязательного ПК

**28-30. План SQL-запроса. Общие сведения и практика**

План SQL-запроса можно получить с помощью команды **EXPLAIN**, предшествующей непосредственно запросу (Select). Стоит помнить, что команда EXPLAIN может быть применена к любому оператору, который имеет план выполнения (Select, Insert, Delete, Update, Values, Execute, Declare, Create Table As)

Структура плана SQL-запроса представляет собой дерево, состоящее из узлов плана. Самые нижние узлы отвечают за просмотр и выдачу строк таблицы, осуществляемые с помощью определенного метода доступа (Seq, Index, Index Only, Bitmap Heap. После обязательно идет слово ключевое имя Scan и имя таблицы, к которой выполняется запрос)

Агрегирование (GroupAggregate, HashAggregate), соединение таблиц определенным методом (Nested Loop, Hash Join, Merge Join), сортировка (Sort) и оконные функции (WindowAgg) создают в дереве плана отдельные узлы, которые стоят выше и выполняются после выполнения нижележащего.

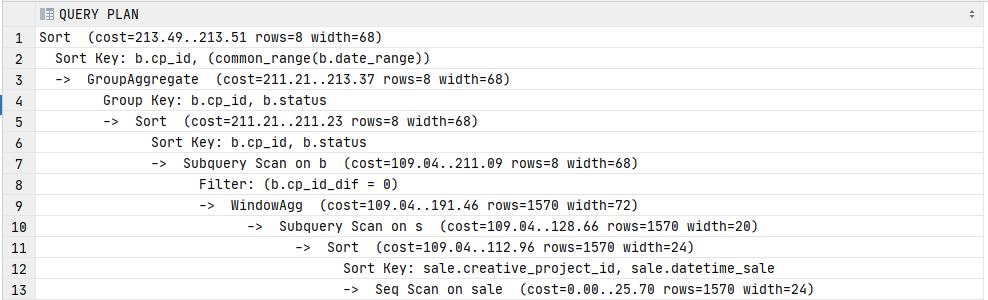
Если в плане запроса есть подзапрос, то он обозначается ключевым словом Subquery.

Если в исходном запросе есть секция Where, то в плане запроса будет присутствовать ключевое слово Filter, которое будет следовать после того запроса, к которому оно применяется.

Каждый узел обязательно содержит в круглых скобках значения трех параметров, если не применен параметр команды EXPLAIN COSTS OFF:

1. cost=x..y - здесь x - это оценка ресурсов, требуемых для начала вывода данных, y - оценка общей стоимости выполнения запроса (включая время обработки нижележащих узлов и вывод). Оценки приведены в условных единицах. Значение имеет только их соотношение
2. rows - это примерное количество строк, используемых на данном узле
3. width - это средний размер строки в байтах

Рассмотрим следующий план запроса:

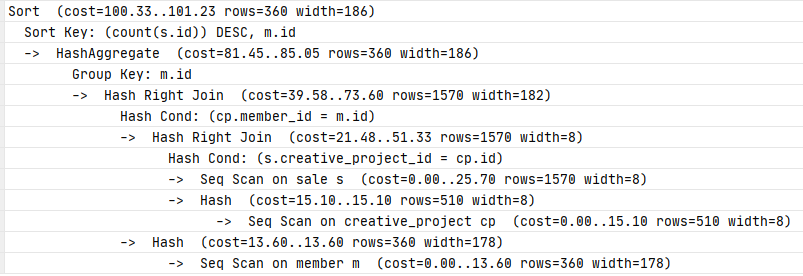


Стрелка слева указывает на то, что вышеописанный узел выполняется над результатами того, на который указывает стрелка.

Читать запрос необходимо снизу вверх:

1. Seq Scan on sale (cost=0.00..25.70 rows=1570 width=24) - Scan означает, что происходит сканирование таблицы и выборка конкретных строк, Seq - что используется метод доступа к таблице “Последовательный просмотр”, sale - название таблицы. cost=0.00..25.70 указывает на то, что для начала вывода данных требуется затратить 0.00 условных единиц, а вывести их - 25.70. rows=1570 - что примерное количество просматриваемых строк равно 1570, а width=24 - что их средний размер равен 24 байта.
2. После отбора строк выполняется вышележащий узел Sort, на котором производится сортировка отобранных ранее строк. Сортировка производится по столбцам creative\_project\_id и datetime\_sale таблицы sale.
3. Subquery Scan on s … - означает, что выполненное ранее является подзапросом с именем s
4. WindowAgg … - означает, что к результатам подзапроса была применена оконная функция
5. Subquery Scan on b … - означает, что это был еще один подзапрос с именем b. Слово Filter указывает, что из него были отобраны только строки, удовлетворяющие условию cp\_id\_dif = 0, где cp\_id\_dif - столбец подзапроса b
6. Выше располагаются два узла: Sort и GroupAggregate. Это означает, что к внешнему запросу была применена группировка по столбцам cp\_id и status подзапроса b, которая вызвала сортировку строк внешнего запроса по этим же столбцам
7. Самым верхним узлом является Sort, что говорит о том, что результаты внешнего запроса были отсортированы по столбцам cp\_id и common\_range(date\_range), где date\_range и cp\_id - столбцы подзапроса b, взятые во внешнем запросе, а common\_range - агрегатная функция (соответственно, common\_range(date\_range) -- это столбец, полученные после применения агрегатной функции common\_range к столбцу date\_range).

Рассмотрим другой запрос, в котором дерево ветвится (есть стрелочки, находящиеся на одном уровне отступа справа. Обычно такое бывает в случае наличия секции Join):



*Эмпирическое правило*: для того, чтобы найти, к какой таблице применяется запрос (эта таблица будет в секции FROM), нужно спускаться по дереву плана запроса до тех пор, пока не будет встречено ключевое слово Scan. Первое из них будет указывать на таблицу, используемую в секции FROM самого первого и глубокого запроса

1. Согласно эмпирическому правилу, запрос применяется к таблице sale. Узел Hash после указывает на то, что параллельно выполняется выборка строк из таблицы creative\_project (из следующего узла).
2. Узел Hash Right Join над узлом Seq Scan on sale s… указывает на то, что строки двух ранее выбранных таблиц sale и creative\_project соединяются методом хеширования. При этом ключевое слово Right указывает, что выполнятся Right Join. Соединение происходит по условию s.creative\_project\_id = cp.id, на которое указывают ключевые слова Hash Cond.
3. Строки полученной на этом узле таблицы соединяются со строками таблицы member, получаемой из следующего узла на том же уровне вложенности, по условию cp.member\_id = m.id.
4. Следующий узел выше указывает на то, что по полученной таблице производится группировка строк хешированием (HashAggregate) по столбцу m.id
5. И самый верхний узел указывает на то, что полученные результаты сортируются по убыванию (есть ключевое слово DESC) по столбцу count(s.id) (агрегатная функция count к столбцу s.id) и по возрастанию по столбцу m.id

**28. План SQL-запроса. Методы просмотра таблиц**

Селективность столбца - это процент строк, имеющих важность для запроса выборки Select. Селективность высокая, если в столбце мало одинаковых значений, и низкая, если много.

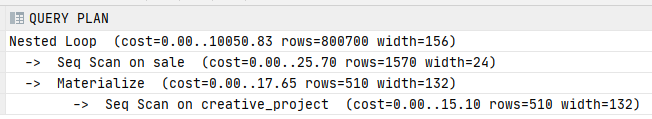
В плане SQL-запроса методы просмотра таблиц обозначаются следующим образом:

1. Seq Scan - последовательный просмотр. Применяется при небольшом размере таблицы, низкой селективности столбца или при невозможности использования других методов. Осуществляется полностью без использования индексов
2. Index Scan - просмотр по индексу. Использует обращение и к индексам, и к непосредственно к таблице. Используется при высокой селективности столбца
3. Index Only Scan - просмотр исключительно по индексу. Использует только обращение к индексам. Возможен при наличии карты видимости таблицы, в которой отмечены те страницы таблицы, которые содержат строки, видные всем транзакциям. Обращение к самой таблице не нужно, если строка видна из индекса. Метод эффективен, если выбираемые данные редко изменяются, и возможен только если все столбцы таблицы индексируемые
4. Bitmap Heap Scan - просмотр на основе битовой карты. Осуществляет поиск по индексу как во 2-ом методе, но после формирует битовую карту, где указано, в каких страницах таблицы содержатся выбираемые строки. При таком подходе из каждой строки за один раз выбирают все строки, которые нужно из нее выбрать.

**29. План SQL-запроса. Методы соединения наборов строк**

В плане SQL-запроса методы соединения наборов строк обозначаются следующим образом:

1. Nested Loop - вложенный цикл. Обычно используется при декартовом произведении, когда в секции FROM перечисляется несколько таблиц (например, FROM a, b). При этом вторая соединяемая таблица может быть помечена дополнительным узлом с ключевым словом Materialize. Метод эффективен для небольших выборок, допускает все виды условий, не требует подготовки. Например:



1. Hash <X> Join - хеширование. Часто используется при соединении Join на условиях равенства. При этом вместо <X> указывается вид Join-соединения (Right, Left, Full). При этом вторая соединяемая таблица может быть помечена дополнительным узлом с ключевым словом Hash (пример смотри в вопросе 41). В данном методе первая таблица становится хеш-таблицей, в которой ключом является столбец, по которому выполняется соединение, а строк в 1-ой таблице обычно меньше, чем во 2-ой. Метод эффективен для больших выборок, но допускает только условия равенства
2. Merge <X> Join - соединение слиянием. Часто используется при большом размере таблиц, соединяемых через Join. Вместо <X> указывается вид Join-соединения. Наборы строк предварительно сортируются по соединяемым столбцам, а затем их строки параллельно читаются и сравниваются. Метод является самым эффективным при больших выборках.

**30. План SQL-запроса. Управление планировщиком.**

При использовании команды EXPLAIN можно использовать ряд параметров для

Все константы по управлению планировщиком

<https://postgrespro.ru/docs/postgresql/9.6/runtime-config-query>

Ебала с джойнами так и не понял для чего оно

https://postgrespro.ru/docs/postgresql/9.6/explicit-joins

**31. Способы соединения таблиц в sql-запросах**

Написать про Join