Вопрос 1. **Файловые системы. Особенности организации устройств внешней памяти на магнитных дисках. Структуры файлов на дисках. Способы организации архивов файлов. Принципы именования.**

Файл - это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Файловая система - это программная система, управляющая файлами, и одновременно совокупность всех файлов во внешней памяти.

В начале развития вычислительной техники для долговременного хранения данных использовались два вида устройств — магнитные ленты и магнитные барабаны. Магнитные ленты обеспечивали последовательный доступ к данным, имели большую емкость и низкую скорость доступа. Магнитные барабаны обеспечивали произвольный доступ к данным и работали намного быстрее магнитных лент, но имели маленькую емкость.

На смену магнитным лентам и барабанам пришли магнитные диски — устройства внешней памяти с несколькими магнитными поверхностями и подвижными головками чтения/записи. Также магнитные диски на устройстве могли сменяться, что обеспечивало потенциально неограниченный объем архива данных. Магнитные диски обладают большей емкостью, чем барабаны, и обеспечивали произвольный доступ к данным с приемлемой скоростью. При выполнении обмена с диском выполняется три основных действия: подвод головок к нужному цилиндру, поиск на дорожке нужного блока, обмен с этим блоком, причем каждый последующий этап происходит значительно быстрее предыдущих: перемещение головок не может происходить слишком быстро, так как они начинают колебаться при резкой остановке, и надо ждать, пока колебания затухнут, скорость вращения диска тоже ограничена характеристиками материала. То есть нельзя получить выигрыш по времени, считывая только часть блока, так как время на чтение блока сильно меньше, чем время подвода головок к нужной точке. В то же время возможность обмена с диском данными, не кратными размеру блока, ведет к внешней фрагментации памяти.

Исторически существуют два основных подхода к организации структуры файлов на диске. Файл может быть представлен как последовательность записей, каждая из которых является последовательностью байт постоянного или переменного размера. При работе с файлом можно читать или писать записи последовательно либо позиционироваться на запись с указанным номером. В некоторых файловых системах возможна структуризация записей на поля и объявление указываемых полей ключами записи, что обеспечивается за счет невидимых пользователю служебных структур и позволяет получать запись из файла по заданному ключу быстрее, чем полным просмотром файла. Вторым подходом является представление файла как непрерывной последовательности байт. Соответственно, при работе с файлом из него можно читать указанное число байт начиная с начала или выполнив позиционирование на байт с указанным номером. При этом в реальных файловых системах файл представляется как последовательность блоков, но это представление скрыто от пользователя.

В современных файловых системах обеспечивается многоуровневое именование файлов за счет существования во внешней памяти каталогов — файлов со специальной структурой, содержащих (в каком-то виде) имена каталогов и файлов, хранящихся в данном каталоге. То есть полное имя файла состоит из списка имен каталогов и имени файла в каталоге, который его непосредственно содержит. Разница между способами именования файлов в разных файловых системах состоит в том, с чего начинается полное имя файла. Первое имя должно соответствовать корневому каталогу файловой системы. Существуют два разных подхода: с одной стороны, каждый архив файлов (то есть полное дерево каталогов) может целиком располагаться на одном логическом диске (разделе физического дискового пакета), и тогда

полное имя файла будет начинаться с имени соответствующего дискового устройства. Это можно назвать изолированной файловой системой. Другим подходом является представление всех файлов в виде единого дерева, и сама файловая система отвечала за распределение файлов по физическим носителям в зависимости от их использования. Такая система называется полностью централизованной. Она удобнее изолированной, но имеет большие проблемы с переносом поддеревьев файловой системы на другую вычислительную установку, потому что для этого надо сначала собрать файлы на один внешний носитель.

Компромиссным подходом можно назвать файловую систему UNIX, где поддерживаются изолированные архивы файлов, один из которых объявляется корневой файловой системой. К корневой файловой системе могут быть присоединены другие изолированные файловые системы, что образует одну общую файловую систему.

Вопрос 2. **Файловые системы. Способы авторизации доступа к файлам. Организация мультидоступа.**

Файл - это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Файловая система - это программная система, управляющая файлами, и одновременно совокупность всех файлов во внешней памяти.

Если с файлами файловой системы работает больше одного пользователя, нужно обеспечивать авторизацию доступа к файлам. В общем случае используется мандатный способ защиты: для каждого пользователя и для каждого файла указывается, что пользователь может с этим файлом делать. Понятно, что для этого надо для каждого файла хранить список пользователей, которые с ним могут что-то делать, что влечет накладные расходы по памяти и времени (поиска пользователя по всему списку). В современных файловых системах используется дискреционный подход к контролю доступа, которые подходит в большинстве случаев. С каждым пользователем связывается пара целочисленных идентификаторов — идентификатор группы пользователя и собственный идентификатор пользователя, а для каждого файла хранится полный идентификатор пользователя-создателя (собственный идентификатор + идентификатор группы) и то, что может делать с файлом 1) пользователь-создатель 2) член группы пользователя-создателя 3) все остальные. Действия с файлом включают в себя чтение, запись и выполнение, что в итоге дает компактность хранимой информации и быстроту проверки прав доступа.

Если больше одного процесса хочет работать с одним и тем же файлом, им требуется организовать возможность делать это корректно. Если все хотят только читать файл, то проблем не возникает, однако при изменения файла они могут быть. Поэтому обычно при открытии файла указывается, как процесс будет с файлом работать (читать или писать). Если этот файл уже кем-то другим открыт, и режим открытия не совместим с требуемым (а совместимы только режимы чтения), то в зависимости от особенностей системы (или параметров функции открытия файла) опоздавшему процессу или сообщается о невозможности открыть файл, либо процесс блокируется, пока эта возможность не появится.

Вопрос 3. **Области применения файловых систем. Требования к базам данных со стороны информационных систем: согласованность данных, языки запросов, восстановление согласованного состояния после сбоев, реальный режим мультидоступа.**

Чаще всего файлы используются для хранения текстовых данных, создаются и модифицируются с помощью различных текстовых редакторов. С точки зрения файловой системы, у текстового файла простая структура — последовательность записей, содержащих строки текста, или последовательность байт, среди которых встречаются специальные символы. Сложность логической структуры текстового файла определяется текстовым редактором, но файловая система ее не знает. Если взять за пример компиляцию программы, компилятор должен знать об устройстве своих входных файлов (текста программы) и выходных (файлов объектных модулей). Но с точки зрения файловой системы файл объектного модуля — это просто последовательность записей или байт. В целом, задача интерпретации содержимого есть задача прикладных программ. Файловая система обеспечивает хранение слабо структурированной информации.

Информационная система ориентирована на хранение, выбор и модификацию данных соответствующей прикладной области. Структура таких данных может быть сложной, но в разных прикладных областях часто бывает много общего между разными структурами данных. Поэтому средства управления данными почти повторялись от одной информационной системы к другой. Желание избавиться от этой избыточности повлекло за собой создание систем управления базами данных.

Для информационной системы база данных должна обладать согласованностью данных, то есть должны быть выполнены все ограничения целостности данной базы данных (к примеру, содержимое файла должно быть согласованно с содержимым других файлов). Для поддержания согласованности данных (то есть целостного состояния базы данных) необходима система, которая берет на себя эти и другие функции (вместо того, чтобы этим каждый раз заново занималась информационная система). Такую систему называют системой управления базой данных.

При работе с базой данных нужно иметь возможность извлекать из базы данных нужные данные, то есть должен быть определен язык запросов к базе данных, позволяющий пользователю от использования низкоуровневых операций для работы с файлами перейти к составлению запросов, не задумываясь, как именно эти запросы будут выполнятся СУБД.

При возникновении сбоев (к примеру, при аварийном выключении питания компьютера, повреждении внешнего хранилища данных или отключения пользователя до завершения работы) должна быть возможность восстановить корректное (то есть согласованное состояние БД, за что тоже отвечает СУБД, в частности, используя механизм журнализации.

Если с БД должно работать больше одного пользователя одновременно, нужен механизм, позволяющий обеспечить это пользователям так, чтобы они не мешали друг другу, а в идеале — чтобы не замечали, что работают с БД не в одиночку. Это СУБД обеспечивает с помощью механизма транзакций. (Транзакции также помогают обеспечить целостность данных)

Билет 4. **Основные функции СУБД, типовая организация СУБД.**

СУБД поддерживает следующие функции, удовлетворяющие потребности информационных систем: поддержку целостности файлов, восстановление согласованного состояния данных после сбоев, обеспечение параллельной работы нескольких пользователей, поддержку языка манипулирования данными.

СУБД управляет данными, содержащимися в БД, в частности, организует во внешней памяти структуры для хранения данных и метаданных и для служебных целей (для быстрого доступа к данным с помощью индексов, к примеру). Для этого могут как использоваться существующие файловые системы, так и производиться работа напрямую с устройствами внешней памяти, но в любом случае эта работа спрятана от пользователя.

СУБД управляет буферами оперативной памяти, что необходимо для увеличения скорости работы с данными, так как если при обращении к любому элементу данных будет производиться обмен со внешней памятью, то вся система будет работать со скоростью устройства внешней памяти, что сильно ее замедлит. Размер БД обычно больше объема ОП (хотя есть БД, которые полностью помещаются в ОП), поэтому СУБД должна определять, что именно нужно буферизовать, так как даже если операционная система сама по себе производит буферизацию, СУБД лучше знает, как устроена база данных и какую ее часть нужно буферизовать.

СУБД управляет транзакциями, что необходимо для обеспечения многопользовательского доступа и обеспечения целостности данных. Транзакция — это последовательность операций над базой данных, рассматриваемых СУБД как единое целое — или все операции выполняются, или не выполняется ни одна. Транзакции связаны с целостностью данных следующим образом: не всегда условия ограничения целостности позволяют выполнить некоторую операцию за одну, пусть и сложную, команду, а промежуточные результаты нескольких команд могут приводить БД в несогласованное состояние. Транзакции позволяют отложить проверку целостности, позволив пользователю сделать все, что нужно. Транзакции связаны с многопользовательским доступом понятием сериализации транзакций. Сериализация транзакций — такой порядок выполнения транзакций, при котором суммарный эффект параллельных или квазипараллельных транзакций эквивалентен эффекту их некоторого последовательного выполнения. Если удается добиться сериального выполнения транзакций, то для каждого пользователя присутствие других будет практически незаметно.

СУБД обеспечивает журнализацию, необходимую для восстановления БД в случае сбоев. Сбои бывают мягкие (потеря недолговременной памяти) и жесткие (потеря памяти на внешнем носителе). Для восстановления после сбоя необходима дополнительная информация, то есть хранение избыточных данных. Методом поддержания этой избыточной информации является ведение журнала изменения. Принято использовать стратегию write ahead log, то есть сначала писать об операции в журнал, а потом ее проводить.

СУБД поддерживает языки БД, которые позволяют работать с БД. Раньше было деление на два языка — язык определения схемы БД и язык манипулирования данными, теперь язык один и позволяет делать все, что надо пользователю.

Логически СУБД (современная реляционная) включает ядро, компилятор языка БД, подсистему поддержки времени выполнения и набор утилит. Ядро отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, транзакциями и журнализацию. Компилятор компилирует операторы языка БД в некоторую выполняемую

программу в машинном коде или внутреннем машинно-независимом коде. Утилиты отвечают за загрузку и выгрузку БД, сбор статистики и глобальную проверку целостности.

Билет 5. **Файл-серверные, клиент-серверные и встраиваемые СУБД**

В файл-серверных СУБД база данных хранится на специализированном файл-сервере, а СУБД запускается на каждом клиенте. Синхронизация многопользовательского доступа реализуется на файл-сервере и на уровне файловой системы, что приводит к блокировке файлов. Файл-сервер только хранит данные, обрабатывают их машины клиентов, поэтому ответ от файл-сервера содержит блоки данных для каждого запроса, что сильно нагружает сеть.

В клиент-серверных СУБД основные компоненты СУБД выполняются на отдельном сервере, на нем же хранится база данных. У клиента находится интерфейса СУБД и выполняется код приложения. Всю работу с базой данных осуществляет сервер, по сети передается только ответ. Недостатком является, во-первых, зависимость клиентов от сервера — если упал сервер, работа встанет у всех, во-вторых — недостаточная нагрузка пользовательских машин, которые с каждым годом все мощнее, в третьих — необходимость прямого соединения между клиентом и сервером БД. Улучшением последнего пункта является трехзвенная клиент-серверная архитектура, которая добавляет дополнительный сервер приложений — прослойку между клиентами. Сервер приложений выполняет код приложения и работает с базой данных, что повышает безопасность и улучшает гибкость — при обновлении СУБД достаточно обновить сервер приложений, не надо обновлять клиентов.

Встраиваемые СУБД встраиваются в код приложений и полностью выполняются на том же компьютере и даже в том же процессе. Обычно это библиотека, которая позволяет использовать внутри программы функции СУБД. Проблема встраиваемых СУБД состоит в незащищенности базы данных от клиента.

Билет 6 **СУБД, хранящие данные во внешней памяти, и СУБД, сохраняющие данные в основной памяти (in-memory)**

Исторически большинство СУБД хранит базу данных во внешней памяти. Хранилище внешней памяти доступно СУБД через системные вызовы операционной системы и структурировано в виде блоков. Нужный блок из внешней памяти буферизуется в оперативной памяти, где с ним продолжается работа. In-memory СУБД располагает всю базы данных в оперативной памяти и использует внешнюю память для обеспечения долговременности транзакций. За счет полного использования оперативной памяти достигается очень большая скорость работы с базой данных. Существует несколько вариантов in-memory СУБД: вообще не использовать внешнюю память, обеспечивая надежность за счет дублирования базы данных в различных узлах кластерной системы, хранить журнал изменений во внешней памяти, или хранить базу данных и в оперативной памяти, и во внешней, читая из оперативной памяти, а записывая в оба места.

Появление SSD уменьшило разрыв по скорости доступа между внешней памятью и оперативной памятью, если со временем этот разрыв исчезнет (или оперативная память станет энергонезависимой), производительность существенно увеличится, так как СУБД можно будет сделать одноуровневой, не лишившись надежности и долговременности хранения данных.

Билет 7. **Однопроцессорные, параллельные с общей памятью, параллельные с общими дисками и параллельными без использования общих ресурсов СУБД**

Однопроцессорные СУБД не используют аппаратные возможности параллелизма и выполняются на одном процессоре. Параллельные СУБД с общей памятью имеют общую память, что позволяет избежать пересылок данных, но требует сложной синхронизации. для упрощения может использоваться следующий подход: одно ядро назначается управляющим, отвечает за компиляцию запросов и их выполнение на других ядрах, все остальные ядра работают обработчиками запросов, что обеспечивает почти линейный прирост производительности при увеличении числа ядер.

Параллельные СУБД с общими дисками работают на кластерах и имеют общую дисковую подсистему, на которой хранится база данных. То есть каждый узел имеет собственную оперативную память. Общие диски выполняют части запросов.

Параллельные без использования общих ресурсов СУБД работают на кластерах, но каждый узел со своим разделом базы данных. При компиляции запроса он разбивается на части, выполняемые различными кластерами. Это обеспечивает наилучшее горизонтальное масштабирование, но возникает проблема разбиения базы данных по узлам кластера так, чтобы запросы выполнялись наиболее быстро, а еще — как производить такое реразбиение при увеличении числа узлов кластера (при масштабировании).

Билет 8. **Дореляционные модели данных**
Модель данных — набор родовых понятий и признаков, которыми должны обладать СУБД и управляемые ими БД, которые на этой модели основываются. С помощью модели данных представляются структура объектов и отношения между ними. Модель данных состоит из структурной части, содержащей основные логические структуры данных, которые могут применяться на уровне пользователя, манипуляционной части, содержащей спецификацию языков, предназначенных для написания запросов, и целостной части, которая специфицирует механизмы ограничений целостности, которые должны поддерживаться в реализации СУБД данной модели.

К дореляционным моделям данных относятся иерархическая, сетевая и модель данных инвертированных таблиц.

В модели инвертированных таблиц база данных — это набор таблиц и индексов, при этом пользователю видны и индексы. Строки таблиц упорядочиваются в некотором видимом пользователю порядке, а для каждой таблицы можно определить произвольное число ключей поиска, для которых будут построены индексы. Эти индексы автоматически поддерживаются системой. Операции данной модели делятся на операции позиционирования в таблице (прямые или относительно текущего ключа) и операции выборки или изменения записей. Соовтетственно, позиционироваться пользователь может по индексам. Поддержка целостности — обязанность приложения.

В иерархической модели данных база данных состоит из упорядоченного набора деревьев одного типа. Тип дерева — корневой тип записи и упорядоченный набор типов поддеревьев, каждое из которых является деревом. Тип дерева — иерархически организованный набор типа записей. Между типами записи поддерживаются типы связей предок-потомок, при этом не может существовать потомков без предков. Определен полный порядок обхода дерева сверху-вниз, слева-направо. Пользователь может ходить по дереву и модифицировать его (вставлять новые записи). Целостность поддерживается только для связей потомок-предок.

Сетевая модель данных является расширением иерархической: у потомка может быть любое число предков, база данных состоит из набора записей и набора бинарных связей между этими записями. Предок может быть только в одном экземпляре связи, если потомок имеет предка, то он единственный в этом экземпляре связи.

Билет 9. **Реляционная модель данных: общее понятие и составные части.**

В реляционной модели база данных состоит из именованных отношений — неупорядоченных таблиц, строками которых являются записи — кортежи. Отношение характеризуется телом, содержащим кортежи, и заголовком, описывающим столбцы таблицы. Заголовок есть множество пар (имя столбца, тип данных). Схема базы данных в реляционной модели — набор именованных заголовков отношений вида (имя атрибута, домен атрибута). Домен — ограничение множества значений некоторого базового типа.
Отношение есть множество кортежей, поэтому к нему применимы обычные теоретико-множественные операции — объединение, пересечение и так далее. Но результатом такой операции над отношением будет отношение, только если у отношений — операндов совпадают заголовки, поэтому в качестве средства манипулирования реляционными базами данных используется специальный набор операций, называемый реляционной алгеброй Кодда.

В реляционной модели существуют два механизма поддержки целостности — ограничение целостности сущности (ограничение первичного ключа) и ограничение ссылочной целостности (ограничение внешнего ключа). Для заголовка любого отношения БД должен быть определен первичный ключ — такое минимальное подмножество заголовка, что в любом теле этого отношения, которое может появиться в базе данных, значение первичного ключа в любом кортеже этого тела уникально. Если первичный ключ не объявляется явно, то им считается весь заголовок. Ограничение ссылочной целостности может быть сформулировано так: «в любом теле отношения R1, которое может появиться в БД, для не пустого значения внешнего ключа, ссылающегося на отношение R2, для любого кортеже тела отношения R1 должен найтись кортеж в теле отношения R2 с совпадающим значением первичного ключа.»

Билет 10. **Основные черты модели данных SQL**

SQL-ориентированная база данных есть набор таблиц, содержащих мультимножество строк, соответствующих заголовку таблицы. Для таблицы поддерживается порядок столбцов, соответствующий порядку их определения. Существуют две основные разновидности таблиц: традиционная и типизированная. Традиционная таблица есть последовательность столбцов с указанными типами данных. Значимым различием между реляционной моделью и моделью данных SQL является существование в последней типа коллекций, позволяющего вложенность. При определении типизированной таблицы указывается ранее определенный структурный тип, и если у него n атрибутов, то в таблице n+1 столбец. Лишний столбец (первый) называется самоссылающимся и содержит типизированные уникальные идентификаторы строк, которые могут генерироваться системой, указываться пользователем или состоять из комбинации значений других столбцов. Типом самоссылающегося столбца является ссылочный тип.

Язык SQL содержит один оператор выборки SELECT, имеющий следующую структуру:
SELECT [DISTINCT] списокСтолбцов1
FROM списокССылокНаТаблицы
[WHERE условноеВыражение1]
[GROUP BY списокСтолбцов2]
[HAVING условноеВыражение2]
[ORDER BY списокСтолбцов3]

Выборка данных производится из таблиц, указанных в разделе FROM, если их больше одной, то образуется одна общая таблица путем операции расширенного декартова умножения. Таблицы могут быть как реально хранимыми в базе данных, так и порожденными (результатами вложенных операторов SELECT). В разделе WHERE выполняется фильтрация и остаются только строки, для которых выполняется условие WHERE. Если есть раздел GROUP BY, то происходит формирование результирующей сгруппированной таблицы. HAVING работает как WHERE, только применяется к результату группировки (к группам).

В модели SQL отсутствует обязательное предписание об ограничении целостности сущности, так как таблицы могут содержать мультимножества строк. Другими словами, могут присутствовать таблицы, для которых первичный ключ не определен. Ссылочная целостность поддерживается в трех вариантах: FULL, SIMPLE, PARTIAL. FULL требует, чтобы для любого внешнего ключа в соответствующей таблице существовала строка с точно таким же первичным ключом, SIMPLE позволяет внешнему ключу быть NULL (то есть не соответствовать некоторому первичному ключу), PARTIAL добавляет возможность некоторым полям внешнего ключа быть NULL, при условии совпадения остальных с частью некоторого первичного ключа.

Еще в SQL можно явно определять ограничения целостности на уровне домена, таблицы или базы данных.

Билет 11. **Типы данных, наследование типов в SQL**

В SQL поддерживаются несколько категорий типов данных. Точные числовые типы включают целочисленные типы и типы с фиксированной запятой. Приближенные числовые типы – float, double. Типы символьных строк — фиксированного размера, не длиннее фиксированного размера и произвольного размера. Битовые строки — как символьные, только хранят не символы, а произвольные байты. Типы даты и времени — дата — с точностью до дня, время — число долей секунды текущего дня, timestamp — комбинация даты и времени. Временные интервалы.

Булевский тип в SQL содержит три значения — true, false, unknown. Это связано с присутствием неопределенного значения NULL, которое может быть интерпретировано как отсутствие значения. Для любой бинарной операции A op NULL = NULL, для любой операции сравнения A comp NULL = unknown. При этом на самом деле одного unknown не хватает, так как значение может отсутствовать по-разному: просто быть неизвестным, не существовать для данного контекста или иметь некоторые известные характеристики.

Типы коллекций бывают двух видов: тип массива, который на практике не используется, и тип мультимножества. Элементы типа коллекции могут быть любого типа данных, определенного к моменту определения данного типа коллекции. В том числе — анонимным строчным типом, что позволяет существовать вложенным таблицам.

Пользовательские типы тоже бывают двух видов — индивидуальный тип и структурный тип. Индивидуальный тип — именованный тип данных, основанный на единственном предопределенном типе. Он не наследует от опорного типа операции. Структурный тип данных — именованный тип данных, включающий один или более атрибутов любого из допустимых типов данных. Для структурного типа можно определять методы, определяющие его поведение.

Что касается наследования типов, при определении типизированной таблицы указывается ранее определенный структурный тип. Таблица содержит столбцов на один больше, чем атрибутов в типе, при этом первый (дополнительный) столбец называется самоссылающимся и содержит типизированные уникальные идентификаторы строк, которые могут генерироваться системой, указываться пользователем или представлять собой комбинацию остальных столбцов. Типом самоссылающегося столбца является ссылочный тип, ассоциированный со структурным типом, определяющим типизированную таблицу.

Типизированные тпаблицы можно определять с использованием механизма наследования. Можно определить подтаблицу типизированной подтаблицы тогда и только тогда, когда структурный тип подтаблицы является непоследовательным подтипом структурного типа супертаблицы. Подтаблица наследует у супертаблицы способ генерации значений ссылочного типа и все ограничения целостности.

Билет 12. **Основные черты модели данных ODMG.**

Объектно-ориентированная модель данных отличается от модели данных SQL и истинной реляционной модели, прежде всего, в одном принципиальном аспекте: в объектно-ориентированной модели база данных — это набор объектов (контейнеров данных) произвольного типа, а не таблицы или отношения.

В стандарте ODMG в качетсве базового средства манипулирования объектными базами данных предлагается язык OQL (Object Query Language). Этот язык опирается на объектную модель ODMG (поддерживает средства доступа ко всем структурам данных, допускаемых в модели). По синтаксису он близок к SQL/92, его расширения относятся к объектно-ориентированным понятиям (сложные объекты, объектные идентификаторы, путевые выражения, полиморфизм, вызов операций, отложенное связывание). В OQL можно работать как с множествами объектов, так и со структурами, списками и массивами. OQL – функциональный язык, то есть возможна неограниченная композиция операций. OQL – не вычислительно полный язык. В OQL не определяются явные операции обновления (считается, что у объектов должны быть для этого методы), и обеспечивается декларативный доступ к объектам.

Результатом допустимых в OQL выражений запросов могут быть коллекция объектов, индивидуальный объект, коллекция литеральных значений, индивидуальное литеральное значение.

Что касается ограничений целостности, в модели ODMG два объекта считаются совпадающими тогда и только тогда, когда у них один и тот же OID. При этом состояние объектов с разными OID может полностью совпадать, поэтому в этой модели нет обязательного ограничения целостности сущности, как в реляционной модели данных. Но можно объявить для атомарного объекта ключ — набор свойств, однозначно идентифицирующий состояние каждого объекта, входящего в экстент класса. (Экстент — множество ссылок на все объекты класса). Ссылочная целостность поддерживается, если определена связь «один-ко-многим». В таком случае объект на стороне связи «один» рассматривается как предок, и не может быть потомков без предков.

Билет 13. **Типы данных, наследование типов в модели данных ODMG.**

В объектной модели данных есть две разновидности типов: литеральные (с явным определением области значений) и объектные, которые делятся на атомарные и типы коллекций.

Литерал — это внешнее обозначение значения типа. Литеральные типы данных — это обычные типы данных, к примеру, скалярные числовые типы, символьные и булевские типы, конструируемые типы записей и коллекций. Литеральный тип записи — определяемый пользователем структурный тип, отличающийся в объектно-ориентированной модели данных тем, что может содержать объекты (на самом деле неявные ссылки на объекты). У любого объекта есть объектный идентификатор (OID), возникающий при создании объекта и являющийся его именем. С его помощью можно получить доступ к атрибутам и методам объекта. Он же служит и аналогом указательного значения на объект.

Существуют четыре типа коллекций — множество, мультимножество (множество с дубликатами), список (упорядоченный набор с дубликатами), словарь (множество пар (уникальный ключ, значение)). Типом элемента любой коллекции может являться любой тип, кроме того же типа коллекции. У объектных типов коллекций есть набор базовых операций (пересечение, вычитание, объединение и тд) и операция создания объекта.

Следует упомянуть, что такое экстент — экстент — это объект типа множества, элементами которого являются объекты данного атомарного объектного типа (в нем сохранены OID всех объектов данного атомарного типа, существующие на данный момент).

У каждого объектного типа есть операции создания и инициализации нового объекта, возвращающая его OID. При определении атомарного объектного типа указывается его внутренняя структура (атрибут или связь) и набор операций, которые к нему можно применять. Для определения объектного типа можно использовать механизм наследования.

Атрибут — свойство объекта, значение которого можно получить по OID. Значением атрибута может быть и литерал, и другой объект, но без обратной ссылки. Связь — инверсное свойство, значением которого может быть только объект. Связи определяются между атомарными объектными типами и в ODMG бывают только бинарные. Могут быть «один к одному», «один ко многим» или «многие ко многим». Связь явно определяется указанием путей обхода (по одному пути для каждого направления).

Билет 14. **. Основные черты истинно реляционной модели данных**

В истинной реляционной модели база данных — это набор долговременно хранимых именованных переменных отношений, каждая из которых определена на некотором типе отношений. В каждый момент времени каждая переменная отношения базы данных содержит некоторое значение отношения соответствующего типа.

В качестве эталонного средства манипулирования данными в истинной реляционной модели можно использовать реляционную алгебру Кодда, но Дейт и Дарвен предложили новую реляционную алгебру А, основанную на реляционных аналогах булевыских операций конъюнкции, дизюнкции и отрицания. Через ее операции выражаются все операции алгебры Кодда.

В число обязательных требований истинной реляционной модели входит требование определения хотя бы одного возможного ключа (подмножества заголовка переменной отношения, обладающее свойствами первичного ключа) для каждой переменной отношения. Кроме того, любое условное выражение, являющееся или лиогически эквивалентное замкнутой правильно построенной формуле реляционного исчисления должно быть допустимо в качестве спецификации ограничения целостности. Средства поддержки декларативной ссылочной целостности фигурируют только в разделе рекомендуемых возможностей.

Билет 15. **Типы данных, наследование типов в истинно реляционной модели данных**

В истинно реляционной модели данных есть три категории типов даных: скалярные типы, кортежные типы и типы отношений. Скалярный тип данных — инкапсулированный тип, реальная внутренняя структура которого скрыта от пользователя. Можно определять новые скалярные типы и операции над ними. Типом атрибута нового скалярного типа может быть любой тип, определенный к этому моменту. Кортежный тип — безымянный тип данных, типом атрибута которого так же может являться любой тип данных. Значением кортежного типа является кортеж, представляющий собой множество триплетов (имя атрибута, тип атрибута, значение атрибута), соответствующее заголовку кортежа этого кортежного типа. Тип отношения — безымянный тип данных, значением которого является заголовок отношения, совпадающий с заголовком кортежа этого типа отношения, и тело отношения, представляющее собой множество кортежей, соответствующих этому заголовку. Кортежные типы и типы отношений не являются инкапсулированными — есть прямой доступ к атрибутам.

Для всех типов данных есть модель множественного наследования, позволяющая определять новые типы данных на основе уже существующих. Модель наследования по Дейту и Дарвену не является частью истинной реляционной модели данных, но заслуживает быть упомянутой — это наследование по ограничению: при определении подтипа супертипа, множество значений подтипа содержится во множестве значений супертипа.

Одиночное наследование требует, чтобы множества значений подтипов не пересекались и в сумме давали множество значений супертипа. В подтипе при одиночном наследовании нельзя определить новый атрибут, если он не выражается через атрибуты супертипа.

При множественном значении от нескольких подтипов, имеющих общий супертип, можно унаследовать подтип, включающий их пересечение. При множественном наследовании есть проблема совпадающих по названию (но, возможно, не по смыслу) атрибутов подтипов. Для решения этой проблемы можно или отказаться от множественного наследования, или переименовать атрибут в одном из типов, или переименовать атрибут в создаваемом подтипе, но все эти варианты не очень хорошие.

Билет 16. **Реляционная алгебра Кодда**

Основная идея реляционной алгебры следующая: так как отношения — множества, то средства манипулирования отношениями могут базироваться на традиционных теоретико-множественных операциях, дополненных специальными операциями, специфичными для реляционных баз данных.

Реляционная алгебра Кодда — набор операций, которые гарантированно производят отношения. Этот набор включает объединение, пересечение, вычитание, взятие расширенного декартова произведения, переименование атрибутов, ограничение, проекцию, соединение, деление и присваивание.

При выполнении операции объединения двух отношений с одинаковыми заголовками производится отношение, включающее все кортежи, входящие хотя бы в одно из отношений-операндов. При этом два отношения совместимы по объединению только тогда, когда обладают одинаковыми заголовками. Операция пересечения — аналогично, но входить кортежи должны в оба отношения. Вычитание — входить в первый, не входить во второй. Расширенное декартово произведение для отношений без общих атрибутов в заголовках дает отношение, кортежи которого есть объединение кортежей операндов. При этом отношения совместимы по этой операции только если у них нет общих атрибутов. Операция переименования производит отношение с таким же телом, но другим именем атрибута. Операция ограничения дает отношение, включающее только кортежи, удовлетворяющие условию ограничения. Операция проекции дает отношение, кортежи которого есть подмножества кортежей отношения-операнда. Соединение (тета-соединение) двух отношений по некоторому условию дает отношение, кортежи которого получаются из объединения кортежей операндов и удовлетворяют условию. Результирующее отношение операции деления состоит из унарных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) включает множество значений второго операнда. Операция присваивания позволяет сохранить результат вычисления реляционного выражения в существующем отношении базы данных.

Поскольку результатом этих операций (кроме присваивания) является отношения, можно образовывать реляционные выражение, в которых вместо операндов находятся вложенные реляционные выражения.

Билет 17. **Алгебра А.**
Пусть r – отношение, A – имя атрибута отношения r, T – имя соответствующего типа атрибута, v – значение типа T. Заголовком Hr отношения r называется множество атрибутов, упорядоченных пар (A, T). Все имена атрибутов различны. Кортеж tr, соответствующий заголовку Hr – множество упорядоченных триплетов (A, T, v). Тело Br отношения r – множество кортежей tr.
В алгебру А входят следующий операции: реляционного дополнения, удаления атрибута, переименования атрибута, реляционной конъюнкции и реляционной дизъюнкции.

Операция <not> производит дополнение заданного отношения. Заголовок остается неизменным, тело результата включает все кортежи, соответствующие заголовку, но не входящие в тело операнда.

Операция <remove> с операндами отношение r и имя атрибута A производит отношение, из заголовка которого убран атрибут A (который изначально должен там быть), а тело содержит все кортежи тела r, из которых убран триплет, соответствующий атрибуту A.

Операция r <rename> (A, B) требует существования в отношении r атрибута A и отсутствия в нем атрибута B. Заголовок результата вместо имени атрибута A содержит B, тело содержит все кортежи тела операнда, с заменой триплета (A, T, v) на (B, T, v).

Операция r1 <and> r2 требует при наличии совпадающих имен атрибутов в операндах совпадения типов (доменов) соответствующих атрибутов. Заголовок результата есть объединение заголовков операндов, тело результата есть кортежи, являющиеся объединением кортежей операндов.

Операция r1 <or> r2 требует того же, что и дизъюнкция. Загловок результата есть объединение заголовков операндов, тело результата есть кортежи, являющиеся результатом объединения кортежей, хотя бы один из которых принадлежит телу одного из операндов. (При этом при наличии совпадающих имен атрибутов значения этих атрибутов в кортежах должны совпадать).

Билет 18. **Полнота алгебры А**
Покажем, что алгебра А является полной, то есть через ее операции можно выразить все операции алгебры Кодда. В алгебру А входят следующий операции: реляционного дополнения, удаления атрибута, переименования атрибута, реляционной конъюнкции и реляционной дизъюнкции. Алгебра Кодда включает объединение, пересечение, вычитание, взятие расширенного декартова произведения, переименование атрибутов, ограничение, проекцию, соединение, деление и присваивание.

Аналогом операции PROJECT является операция <remove>. UNION есть частный случай <OR>, TIMES, INTERSECT, NATURAL JOIN – частные случаи <AND>.
R1 MINUS r2 = r1 <AND> <NOT> r2
r WHERE comp (операция взятия ограничения) выражается через комбинацию операций <and>, <remove>, <rename> следующим образом: для равенства надо сделать r <and> (X), где X – r, из которого удаляются все лишние атрибуты (остается атрибут, находящийся по правую сторону равенства), и оставшийся атрибут переименовывается в атрибут, находящийся по левую сторону равенства. Для остальных операций сравнения, в силу конечности множества значений типов, операцию сравнения можно заменить на операцию проверки на равенство со множеством подходящих значений, которое может быть явно задано как литеральная константа.

Для получения результата соединения общего вида нужно выполнить над одним из отношений операции <rename> для избавления от общих имен атрибутов, над результатом и вторым операндом провести <and> для получения расширенного декартового соединения, и ограничить результат описанным ранее образом для удовлетворения условию.

Для реляционного деления верно следующее: r1(A, B), r2(B), r1 DIVIDE BY r2 = (r1 PROJECT A) MINUS (((r2 TIMES (r1 PROJECT A)) MINUS r1) PROJECT A), а все операции в правой части могут быть выражены через операции алгебры А.

Действительно, результатом выполнения операции r1 PROJECT A является унарное отношение со схемой {A}, кортежи тела которого содержат все значения атрибута А из тела отношения r1. Результат выражения r2 TIMES (r1 PROJECT A) – это бинарное отношение со схемой {A, B}, в тело которого входят все возможные комбинации значений атрибута B в теле отношения r2 и атрибута А в теле отношения r1. В теле результата вычисления выражения (r2 TIMES (r1 PROJECT A)) MINUS r1 останутся только те кортежи, которые не входят во второй операнд, т. е. кортежи с таким значением атрибута А, что значение атрибута B, принадлежащее телу r2, не является значением атрибута B ни в одном кортеже тела отношения r1. Следовательно, если мы возьмем проекцию результата выражения (r2 TIMES (r1 PROJECT A)) MINUS r1 на атрибут А, то в результирующем унарном отношении останутся только те значения А, которые не должны попасть в результат операции r1 DIVIDE BY r2. После выполнения завершающей операции MINUS мы получим желаемый результат.

Билет 19. **Избыточность алгебры А.**

В формальной математической логике стандартным базисом является набор (not, and, or), но этот набор является избыточным (так как, к примеру, a AND b = not (not a or not b). Аналогичные тождества справедливы для операций из алгебры А. Таким образом, можно убрать одну из операций (конъюнкцию или дизъюнкцию). В то же время существуют операции, через которые выражается весь стандартный базис — штрих Шеффера sh(A, B) = not a or not b и стрелка Пирса — pi(A, B) = not a and not b. (sh(a, a) = not a, sh(not a, not b) = a or b, not sh(a, b) = a and b. То есть алгебру А можно свести к трем операциям: <sh> (или <pi>), <rename> и <remove>.

Более того, операцию <rename> можно выразить через <and> и <remove> (а значит, и через <sh> и <remove>). Делается это так: применив к отношению операцию <and> с отношением-литеральной константой, кортежи которого содержат два одинаковых значения, имена атрибутов которых совпадают с именем, нуждающимся в замене, и новым именем, получим отношение, в котором есть дополнительный атрибут, совпадающий с нуждающимся в переименовании в каждом кортеже всем, кроме имени. Осталось удалить этот атрибут, и требуемый результат получен.

Таким образом алгебра А может быть сокращена до двух операций - <sh> (или <pi>) и <rename>, и не может быть сокращена дальше, так как одна из этих операций уменьшает число атрибутов заголовка, а вторая нет, поэтому эти операции не могут быть выражены друг через друга.

Билет 20. **Реляционное исчисление кортежей**

В исчислении кортежей областями определения переменных являются тела отношений базы данных, то есть допустимым значением каждой переменной является кортеж тела некоторого отношения. Для определения кортежной переменной используется оператор RANGE: RANGE имя_переменной is имя_отношения. При использовании кортежных переменных можно ссылаться на значение атрибута переменной с помощью оператора .: имя_переменной.имя_атрибута.

Правильно построенная формула (WFF) служит для выражения условий, накладываемых на кортежные переменные. Основой WFF являются простые условия — операции сравнения скалярных значений (атрибутов переменных или литеральных констант), вида имя_переменной.имя_атрибута = литеральная_константа, или имя_переменной.имя_атрибута = имя_другой_переменной.имя_другого_атрибута. Более сложные WFF строятся с помощью логических связок NOT, AND, OR, IF .. THEN, IF a THEN b = NOT a OR b. Приоритет операций обычный, скобки разрешены. То есть если form – WFF, comp – простое сравнение, то NOT form, comp AND form, comp OR form, IF comp THEN form – WFF.

При построении WFF допускается использование кванторов существования (EXISTS) и всеобщности (FORALL). Если form – WFF с переменной var, то EXISTS var (form) и FORALL var (form) – WFF. Первая формула принимает значение true, если в области определения переменной var есть значение, для которого form = true, вторая — если form = true для всей области определения переменной var. Переменные в WFF могут быть свободными и связанными. Все переменные, входящие в WFF, в которой не используются кванторы — свободные. Если для набора значений свободных кортежных переменных при вычислении WFF было получено значение true, то этот набор может входить в результат. Если же имя переменной использовано в конструкции EXISTS var (form) или FORALL var (form), то в этой WFF и всех WFF, из нее построенных, var – связанная переменная, то есть она не видна за пределами минимальной WFF, связавшей эту формулу. Заметим, что связанное в какой-то конструкции вхождение переменной var никак не влияет на другие WFF, в которые эта конструкция не входит (то есть можно использовать имя повторно, его область видимости ограничена.

WFF обеспечивает средства формулировки условий выборки из отношений базы данных, еще одним компонентом является целевой список, определяющий набор и имена атрибутов результирующего отношения. Целевой список строится из целевых элементов вида имя_свободной_переменной.имя_атрибута, имя_свободной_переменной или новое_имя = имя_свободной_переменной.имя_атрибута. Второй случай эквивалентен указанию всех атрибутов переменной.

Выражением реляционного исчисления кортежей называется конструкция вида целевой_список WHERE WFF. Значением выражения является отношение, тело которого определяется WFF, а заголовок — целевым списком.

Билет 21. **Реляционное исчисление доменов**

В исчислении доменов областью определения переменных являются домены. Переменные доменов — скалярные переменные, в качестве значений содержащие элементы какого-нибудь домена. Основным формальным отличием исчисления доменов от исчисления кортежей является наличие дополнительного множества предикатов, позволяющих выражать условия членства. Если R – n-арное отношение с атрибутами a(i), то условие членства имеет вид R(ai(1) : vi(1), … ai(m) : vi(m)), m <= n, где vij – литерально задаваемая константа или имя доменной переменной. Условие членства принимает значение true тогда и только тогда, когда в отношении R существует кортеж, содержащий указанные значения указанных атрибутов.

Реляционное исчисление доменов очень похоже на реляционное исчисление кортежей, то есть может содержать кванторы, свободные и связанные доменные переменные, и имеет вид целевой_список WHERE WFF.

Правильно построенная формула (WFF) служит для выражения условий, накладываемых на кортежные переменные. Основой WFF являются простые условия — операции сравнения скалярных значений (атрибутов переменных или литеральных констант), вида имя_переменной.имя_атрибута = литеральная_константа, или имя_переменной.имя_атрибута = имя_другой_переменной.имя_другого_атрибута. Более сложные WFF строятся с помощью логических связок NOT, AND, OR, IF .. THEN, IF a THEN b = NOT a OR b. Приоритет операций обычный, скобки разрешены. То есть если form – WFF, comp – простое сравнение, то NOT form, comp AND form, comp OR form, IF comp THEN form – WFF.

При построении WFF допускается использование кванторов существования (EXISTS) и всеобщности (FORALL). Если form – WFF с переменной var, то EXISTS var (form) и FORALL var (form) – WFF. Первая формула принимает значение true, если в области определения переменной var есть значение, для которого form = true, вторая — если form = true для всей области определения переменной var. Переменные в WFF могут быть свободными и связанными. Все переменные, входящие в WFF, в которой не используются кванторы — свободные. Если для набора значений свободных кортежных переменных при вычислении WFF было получено значение true, то этот набор может входить в результат. Если же имя переменной использовано в конструкции EXISTS var (form) или FORALL var (form), то в этой WFF и всех WFF, из нее построенных, var – связанная переменная, то есть она не видна за пределами минимальной WFF, связавшей эту формулу. Заметим, что связанное в какой-то конструкции вхождение переменной var никак не влияет на другие WFF, в которые эта конструкция не входит (то есть можно использовать имя повторно, его область видимости ограничена.

Целевой список строится из целевых элементов вида имя_свободной_переменной.имя_атрибута, имя_свободной_переменной или новое_имя = имя_свободной_переменной.имя_атрибута. Второй случай эквивалентен указанию всех атрибутов переменной.

Билет 22. **Функциональные зависимости, замыкание множества функциональных зависимостей, аксиомы Армстронга, замыкание множества атрибутов. Минимальное покрытие множества функциональных зависимостей.**

Пусть задана переменная отношения R, X и Y – составные атрибуты (произвольные подножества заголовка отношения). Атрибут Y функционально зависит от атрибута X тогда и только тогда, когда каждому значению X соответствует в точности одно значение Y. В другую сторону, атрибут X функционально определяет атрибут Y (является детерминантом). R.X → R.Y

Функциональные зависимости, которые должны быть верны для любого допустимого значения переменной отношения могут рассматриваться как инварианты, или ограничения целостности этой переменной отношения. Соответственно, за соблюдением таких функциональных зависимостей должна следить СУБД, поэтому важно уметь сократить набор функциональных зависимостей до эквивалентного минимума.

Замыканием множества функциональных зависимостей является множество функциональных зависимостей, включающее все функциональные зависимости, логически выводимые из начального множества. Функциональная зависимость A→ C называется транзитивной, если существуют ФЗ A→B и B→C, но отсутствует прямая функциональная зависимость A→C.

Аксиомы Армстронгла — это набор правил вывода новых функциональных зависимостей из существующих, позволяющих решить проблему поиска замыкания множества функциональных зависимостей. Пусть A, B, C – атрибуты отношения R, AB = A UNION B. Тогда: 1) если B входит в A, то A → B. (рефлексивность) 2) если A → B, то AC → BC. (пополнение) 3) если A → B, B →C, то A → C. (транзитивность)

Истинность первой аксиомы очевидна, так как тривиальная функциональная зависимость всегда выполняется. Пусть вторая аксиома ложна. Тогда в некотором допустимом теле отношения найдутся кортежи t1, t2: t1(AC) = t2(AC), но t1(BC) != t2(BC) (то есть проекции на соответствующие множества атрибутов не совпадают). По аксиоме рефлексивности t1(A) = t2(A), из A → B тогда t1(B) = t2(B), но тогда из t1(BC) != t2(BC) => t1(C) != t2(C) , что противоречит AC → C. Аналогично докажем третью аксиому. Пусть есть t1, t2: t1(A) = t2(A), t1(C) != t2(C). Из A→B t1(B) = t2(B), из B → C t1(C) = t2(C). Противоречие, аксиома доказана.

Можно доказать, что аксиомы Армстронга полны и совершенны — то есть что замыкание множества функциональных зависимостей может быть получено использованием только их, и не могут быть при этом получены лишние зависимости.

Замыканием Z+ множества атрибутов Z над множеством функциональных зависимостей S называется наибольшее множество таких атрибутов Y заданного отношения R, что функциональная зависимость Z→Y входит в замыкание S+ множества S.

Множество функциональных зависимостей S2 называется покрытием множества функциональных зависимостей S1, если любая ФЗ, выводимая из S1, выводится и из S2. Два множества ФЗ называются эквивалентными, если являются покрытиями друг друга. Множество ФЗ называется минимальным, если 1) правая часть любой его ФЗ является простым атрибутом 2) детерминант каждой ФЗ обладает свойством минимальности (из него нельзя удалить никакой атрибут без изменения замыкания множества ФЗ) 3) удаление любой ФЗ приводит к изменению замыкания. Для любого множества ФЗ существует и может быть

построено эквивалентное ему минимальное множество. Минимальным покрытием множества ФЗ называется любое эквивалентное минимальное множество.

Билет 23. **Декомпозиция без потерь и функциональные зависимости, теорема Хита.**

Пусть задана переменная отношения R, X и Y – составные атрибуты (произвольные подножества заголовка отношения). Атрибут Y функционально зависит от атрибута X тогда и только тогда, когда каждому значению X соответствует в точности одно значение Y. В другую сторону, атрибут X функционально определяет атрибут Y (является детерминантом)
$R.X \rightarrow R.Y$

Декомпозиция — это разбиение путем проецирования отношения, находящегося в предыдущей нормальной форме, на два или более отношений, удовлетворяющих требованиям следующей нормальной формы. Декомпозиция без потерь — обратимая декомпозиция, то есть та, для которой возможно собрать исходное отношение из декомпозированных без потери информации.

Теорема Хита: пусть задано отношение r (A, B, C) , для которого $A \rightarrow B$. Тогда r = (r PROJECT (A, B)) NATURAL JOIN (r PROJECT (A, C)). Действительно, пусть результат естественного соединения есть r1, а кортеж (a, b, c) содержится в r. Тогда по определению операции взятия проекции (a, b) принадлежит первой части соединения, (a, c) – второй, откуда (a, b, c) содержится в r1. Значит, тело r входит в тело r1. Пусть в r1 есть кортеж (a, b, c), тогда (a, b) входит в левую часть соединения, (a, c) – в правую. Это возможно только тогда, когда в r есть кортеж (a, b1, c). Но так как $A \rightarrow B$, то b1 = b, откуда r1 = r.

Билет 24. **Проектирование реляционных баз данных с использованием нормализации: первая, вторая и третья нормальные формы**

Процесс проектирования реляционных базы данных осуществляется методом последовательных приближений к удовлетворительному набору схем отношений. Этот процесс называется процессом нормализации схем отношений, причем каждая следующая нормальная форма в некотором смысле лучше предыдущей и содержит свойства предыдущих нормальных форм. В основе метода нормализации лежит декомпозиция отношения, находящегося в предыдущей нормальной форме, на два или более отношения, находящихся в следующей нормальной форме.

Каждой нормальной форме соответствует набор ограничений. Отношение находится в некоторой нормальной форме, если удовлетворяет ее ограничениям.

Первая нормальная форма: значения всех атрибутов отношения атомарны. Атомарность значения значит, что значение типизировано, и с ним можно работать только с помощью операций соответствующего типа данных. Требование первой нормальной формы является базовым требованием классической реляционной модели, поэтому считается, что первая нормальная форма уже достигнута.

Вторая нормальная форма: первая нормальная форма + каждый неключевой атрибут минимально функционально зависит от первичного ключа. Атрибут B минимально зависит от атрибута A, если выполняется минимальная слева ФЗ $A \rightarrow B$ (то есть связь с минимальным детерминантом). Если ФЗ атрибутов от возможного ключа не являются минимальными, это приводит к аномалиям обновления — то есть трудностям при выполнении операций добавления, удаления и модификации кортежей.

Третья нормальная форма: вторая нормальная форма + каждый неключевой атрибут нетранзитивно функционально зависит от первичного ключа. Для транзитивных ФЗ тоже существуют аномалии обновления.

Билет 25. **Проектирование реляционных баз данных с использованием нормализации: теорема Риссонена, нормальная форма Бойса-Кодда.**

Процесс проектирования реляционных базы данных осуществляется методом последовательных приближений к удовлетворительному набору схем отношений. Этот процесс называется процессом нормализации схем отношений, причем каждая следующая нормальная форма в некотором смысле лучше предыдущей и содержит свойства предыдущих нормальных форм. В основе метода нормализации лежит декомпозиция отношения, находящегося в предыдущей нормальной форме, на два или более отношения, находящихся в следующей нормальной форме.

Каждой нормальной форме соответствует набор ограничений. Отношение находится в некоторой нормальной форме, если удовлетворяет ее ограничениям.

Отношения являются независимыми проекциями отношения, если они могут обновляться независимо и при этом результат их естественного соединения всегда будет таким, как если бы обновлялось исходное отношение.

Теорема Риссонена. Проекции r1 и r2 отношения r являются независимыми тогда и только тогда, когда 1) каждая функциональная связь в отношении r выводится на основе аксиом Армстронга из функциональных связей в r1, r2 2) общие атрибуты r1, r2 образуют возможный ключ хотя бы для одного из этих отношений.

Атомарным отношением называется отношение, которое невозможно декомпозировать на независимые проекции.

Переменная отношения находится в нормальной форме Бойса-Кодда тогда и только тогда, когда любая ее нетривиальная и минимальная функциональная связь имеет в качестве детерминанта некоторый возможный ключ отношения.

Билет 26. **Многозначные зависимости, теорема Фейджина, четвертая нормальная форма.**

Процесс проектирования реляционных базы данных осуществляется методом последовательных приближений к удовлетворительному набору схем отношений. Этот процесс называется процессом нормализации схем отношений, причем каждая следующая нормальная форма в некотором смысле лучше предыдущей и содержит свойства предыдущих нормальных форм. В основе метода нормализации лежит декомпозиция отношения, находящегося в предыдущей нормальной форме, на два или более отношения, находящихся в следующей нормальной форме.

Каждой нормальной форме соответствует набор ограничений. Отношение находится в некоторой нормальной форме, если удовлетворяет ее ограничениям.

В переменной отношения R с атрибутами A, B, C имеется многозначная зависимость B от A (A →→B) тогда и только тогда, когда множество значений атрибута B, соответствующее паре значений атрибутов A и C зависит от значения A и не зависит от значения C. Многозначная зависимость является обобщением понятия функциональной зависимости. Поэтому из A → B => A →→ B.

Теорема Фейджина: в отношении R(A, B, C) выполняется A →→ B тогда и только тогда, когда A→→C . Действительно, пусть A →→ B, a – значение атрибута A в некотором кортеже R, {b} – множество значений атрибута B из тех кортежей, где значение атрибута A = a. Пусть для этого значения a не выполняется A →→ C, то есть существует такое допустимое значение c атрибута C и b из {b}, что {a, b, c} не принадлежит телу отношения. Но это противоречит A →→B. Значит, из A →→B => A →→ C. Необходимость доказывается аналогично с заменой B на A.

Таким образом многозначные зависимости A→→B и A→→C всегда идут в паре, что записывается как A→→B|C.

Билет 27. **Зависимости проекции-соединения, пятая нормальная форма.**

Процесс проектирования реляционных базы данных осуществляется методом последовательных приближений к удовлетворительному набору схем отношений. Этот процесс называется процессом нормализации схем отношений, причем каждая следующая нормальная форма в некотором смысле лучше предыдущей и содержит свойства предыдущих нормальных форм. В основе метода нормализации лежит декомпозиция отношения, находящегося в предыдущей нормальной форме, на два или более отношения, находящихся в следующей нормальной форме.

Каждой нормальной форме соответствует набор ограничений. Отношение находится в некоторой нормальной форме, если удовлетворяет ее ограничениям.

Отношение называется n-декомпозированным, если оно может быть декомпозировано без потерь на n проекций.

В переменной отношения R с атрибутами A, B, C имеется многозначная зависимость B от A (A →→B) тогда и только тогда, когда множество значений атрибута B, соответствующее паре значений атрибутов A и C зависит от значения A и не зависит от значения C. Многозначная зависимость является обобщением понятия функциональной зависимости. Поэтому из A → B => A →→ B. Многозначная зависимость называется тривиальной, если A является подмножеством B или в объединении с ним дает заголовок отношения R. Тривиальная многозначная зависимость всегда удовлетворяется, в первом случае она вырождается в тривиальную функциональную зависимость, во втором случае удовлетворяется очевидным образом.

Пусть задана переменная отношения R, A,...Z – произвольные составные атрибуты. В R удовлетворяется зависимость проекции-соединения *(A, … Z) тогда и только тогда, когда любое допустимое значение r переменной отношения R можно получить путем естественного соединения проекций этого значения на атрибуты A...Z. Эта зависимость называется подразумеваемой возможными ключами, если каждый указанный атрибут является суперключом R (включает хотя бы один возможный ключ). Зависимость проекции-соединения называется тривиальной, если хотя бы один из указанных атрибутов совпадает с заголовком отношения.

Переменная отношения R находится в пятой нормальной форме тогда и только тогда, когда каждая нетривиальная зависимость проекции-соединения подразумевается возможными ключами R.

Билет 28. **Семантическая модель Entity-Relationship (Сущность-Связи).**

К появлению семантических моделей данных привела потребность проектировщиков баз в удобных и мощных средствах моделирования предметной области. Основным назначением семантической модели является обеспечения возможности выражения семантики данной. В терминах семантической модели производится концептуальная схема базы данных, которая затем вручную преобразуется к реляционной (или какой-то другой).

Построение наглядной концептуальной схемы базы данных позволяет более полно оценить специфику моделируемой предметной области и избежать ошибок стадии проектирования схемы реализуемой базы данных. Производимая на этом этапе документация полезна на всех последующих этапах проектирования и использования базы данных. Без семантического моделирования можно обойтись только в случаях достаточно малого числа таблиц.

Семантическая модель сущность-связи была предложена Питером Ченом. Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных элементов. Основными понятиями ER-модели являются сущность, связь и атрибут.

Сущность — объект, информация о котором должна сохраняться и быть доступным, представляется как прямоугольник с именем сущности. При определении типа сущности необходимо гарантировать, что каждый экземпляр сущности может быть отличим от любого другого экземпляра той же сущности. Необходимо также сообщить системе автоматизации проектирования базы данных, как будут различаться эти экземпляры, то есть как сконструировать уникальный идентификатор экземпляра типа сущности. Уникальным идентификатором может быть атрибут, комбинация атрибутов, связь, комбинация связей или комбинация связей и атрибутов.

Связь — графически изображаемая ассоциация, устанавливаемая между двумя типами сущностей. В рассматриваемом варианте модели ассоциация всегда бинарная и может быть как между разными типами сущностей, так и между типом и им же (рекурсивная). В связи выделяются два конца, на каждом из которых указывается имя конца связи, степень конца связи, обязательность. Степень конца связи значит сколько экземпляров сущности на конце связи может в ней участвовать — одноточечный вход, если один, трехточечный — если много. Обязательность показывается сплошной линией, необязательность — прерывистой.

Атрибутом сущности является любая деталь, служащая для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Имена атрибутов заносятся в прямоугольник, изображающий сущность.

Почти аналогично схемам реляционных баз данных, для ER-диаграмм существует понятие нормальных форм. В первой нормальной форме устраняются атрибуты, содержащие множественные значения, во второй — атрибуты, зависящие только от части уникального идентификатора, в третьей — атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор. Формы выше третьей обычно не используются.

Еще в элементы модели входят подтипы и супертипы сущностей, то есть возможность наследования, уточняемые степени связи (то есть не один или много, а точное число, интервалы или более сложные конструкции), взаимно исключающие связи, каскадное удаление экземпляров сущностей (то есть сильные связи, означающие существование какой-то сущности только в комплекте с другой, связанной один ко многим, домены (возможность определить допустимое значение множества значений атрибута).

Что касается наследования, сущность может быть расщеплена на два или большее число взаимно исключающих подтипов, каждый из которых включает общие атрибуты и/или связи. В подтипах могут определяться собственный атрибуты и связи. Если у типа сущности A имеются подтипы B1, … Bn, то 1) любой экземпляр сущности Bi является экземпляром типа сущности A 2) любой экземпляр сущности A является экземпляром некоторого подтипа сущности Bi. Ни для каких подтипов Bi и Bj не существует экземпляра, имеющего одновременно два подтипа. То есть подтипы образуют полное множество.

Допускается наличие двух или более разбиений сущности на подтипы.

Билет 29. **Получение реляционной схемы из ER-диаграммы**

Каждый простой тип сущности превращается в таблицу. (Простым типом сущности называется тип сущности, не являющийся подтипом и не имеющий подтипов). Имя сущности становится именем таблицы, экземплярам типа сущности соответствуют строки соответствующей таблицы. Каждый атрибут становится столбцом таблицы с тем же именем. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения, столбцы, соответствующие обязательным атрибутам — не могут. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если их несколько, выбирается один. Если в состав уникального идентификатора входят связи, в число столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи.

Связи многие к одному и один к одному становятся внешними ключами, необязательные связи позволяют наличие неопределенных значений в столбцах внешнего ключа. Для поддержки связи многие ко многим создается дополнительная таблица с двумя столбцами, содержащими уникальные идентификаторы экземпляров связываемых сущностей.

Индексы создаются для первичного ключа (уникальные), внешних ключей, и тех атрибутов, для которые это полезно.

Если в ER-диаграмме присутствуют подтипы, то есть два способа отразить их в реляционную схему: собрать все подтипы в одной таблице или сделать отдельную таблицу для каждого подтипа. Первый случай соответствует логике супертипов и подтипов, обеспечивает простой доступ к экземплярам супертипа и подтипов, и уменьшает общее число таблиц, но требует дополнительной логики работы с разными наборами столбцов и разными ограничениями целостности и может привести к проблемам производительности, став узким местом при многопользовательском доступе. Для первого случая таблица создается для максимального супертипа, а для подтипов могут создаваться представления, и в таблице присутствует столбец, содержащий код типа, которые становится частью первичного ключа. Для второго случая действуют более понятные правила работы с подтипами, нет проблемы многопользовательского доступа, но требуется много таблиц и сложно работать с супертипом. Супертип воссоздается в таком случае с помощью объединения проекций таблиц, соответствующих подтипам, на заголовок таблицы супертипа.

Взаимно исключающие связи также могут быть представлены двумя способами: с общим хранением внешних ключей и с раздельным хранением внешних ключей. Если внешние ключи всех потенциально связанных таблиц имеют общий формат, то можно применять первый способ, который требует два столбца — идентификатор связи и идентификатор сущности. Если результирующие внешние ключи не относятся к одному домену, то нужно использовать второй способ, то есть создавать для каждой связи явные столбцы внешних ключей, которые могут содержать неопределенные значения. При первом подходе в таблице только два дополнительных столбца, но усложняется операция соединения, если нужно использовать одну из связей. Для второго подхода соединение является явным, но нужно иметь столько столбцов, сколько имеется альтернативных связей, и будет храниться много неопределенных значений.

Билет 30. **Диаграммы классов языка UML**
Диаграммой классов в UML называется диаграмма, на которой показан набор классов и некоторых других сущностей, а также связи между этими классами, и, возможно, комментарии и ограничения. Ограничения могут задаваться как на естественном языке, так и на языке объектных ограничений OCL.

Класс — это именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой. Изображается в виде прямоугольника. У класса должно быть уникальное имя, которое рекомендуется сделать коротким, осмысленным и читаемым.

Атрибутом класса называется именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства. Атрибут является абстракцией состояния объекта. Класс может иметь любое число атрибутов. Любой атрибут должен иметь некоторое значение. Имена атрибута записываются под именем класса, и рекомендации для них такие же, как и для имени класса.

Операция класса — это именованная услуга, которую можно запросить у любого объекта этого класса. Класс может содержать любое число операций. Операции записываются под атрибутами и могут содержать сигнатуру и тип значения операции.

В диаграмме классов могут участвовать связи трех разных категорий: зависимости (связи по применению, когда изменение в спецификации одного класса может повлиять на поведение другого, использующего первый), показываемые прерывистой линией со стрелкой, направленной к классу, от которого имеется зависимость.

Обобщения — связь между общей сущностью — суперклассом и специализированной разновидностью сущности, подклассом. Подклассы могут использоваться везде, где могут использоваться суперклассы — это называется полиморфизм по включению. Связь-обобщение изображается сплошной линией с большой незакрашенной стрелкой, направленной к суперклассу. Допускается множественное наследование, в котором есть ряд проблем, в том числе проблема именования атрибутов.

Связь-ассоциация показывает, что объекты одного класса некоторым образом связаны с объектами другого или того же самого класса. Ассоциация может связывать n классов. Изображается связь-ассоциация как линия. Дополнительными параметрами связи-ассоциации является имя, характеризующее природу связи, роль каждого класса, участвующего в ассоциации, кратность, показывающая, сколько объектов класса может участвовать в каждом экземпляре ассоциации. Обычная ассоциация характеризует связь между равноправными классами, если связь имеет вид часть-целове, то такая ассоциация называется агрегатной и изображается дополнительным ромбом на стороне класса-целого, закрашенным, если часть не может существовать без целого.

При наличии простой ассоциации между классами предполагается возможность навигации между объектами, входящими в один экземпляр ассоциации.

Билет 31. **Язык объектных ограничений OCL.**
OCL – язык объектных ограничений, на котором могут быть написаны ограничения классов в диаграммах классов языка UML. Из UML в OCL заимствованы следующие понятия: класс, атрибут, операция, объект (экземпляр класса), ассоция, тип данных, значение (экземпляр типа данных). Для понимания OCL существенны определяемые в UML различия между объектом класса и значением типа: объект обладает уникальным идентификатором и может сравниваться с другим объектом только по идентификатору, поэтому можно определить операции над множеством объектов в терминах их идентификаторов. Объект может быть ассоциирован через бинарную связь с другим объектом, что позволяет определить операцию перехода от одного объекта к другому, с ним связанному. Для значения при сравнении двух значений проверяются сами эти значения, и значения не могут участвовать в связях.

В дополнение к скалярным типам данных UML, в OCL предопределены структурные типы, являющиеся разновидностями типов коллекций: множество, мультимножество, последовательность. Элементами каждого из типов коллекций могут быть либо объекты, либо значения.

Под инвариантом класса в OCL понимается условие, которому должны удовлетворять все объекты данного класса при создании и в течении всего времени своего существования. Синтаксис инварианта следующий:
context <class_name> inv:
<OCL-выражение>
Где <class_name> - имя класса, для которого определяется инвариант, inv и context – ключевые слова, говорящие что определяется 1) инвариант 2) для объектов конкретного класса.

OCL — типизированный язык, поэтому у каждого выражения есть тип, и в инварианте тип выражения должен быть логическим. OCL-выражение инварианта основывается на композиции некоторого множества операций. Операции над значениями скалярных типов данных, над объектами, над множествами, над мультимножествами и над последовательностями. Определены три операции над объектами: получение значения и атрибута, переход по соединению, вызов операции класса, имеющие вид объект.имя_атрибута (или .роль, или .операция). Результатом перехода по соединению является коллекция, содержащая все объекты, ассоциированные с данным по данному соединению. Операции над коллекциями записываются аналогично, но вместо оператора точка используется оператор стрелка.

В OCL определены три одноименные операции select, которые обрабатывают заданное множество, мультимножество или последовательность на основе заданного логического выражения над элементами коллекции. Результатом операции является новое множество, мультимножество или последовательность из тех элементов, для которых условие удовлетворяется. Аналогично определены операции collect, результатом которых является мультимножество для операнда-множества или мультимножества и последовательность для операнда-последовательности. Результат состоит из результатов применения выражения к каждому элементу входной коллекции. Операция exists выдает true, если для какого-то элемента операнда-коллекции значением логического выражения является true, операция forall – если это верно для всех элементов операнда-коллекции.

Билет 32. **Организация внешней памяти в SQL-ориентированных базах данных, хранение таблиц по строкам и столбцам**

В современных SQL-ориентированных базах данных есть несколько особенностей, влияющих на организацию внешней памяти. Во-первых, есть двухуровневая система: на одном уровне производится непосредственное управление данными во внешней памяти, буферами оперативной памяти, транзакциями и журнализацией, на другом — реализация языка SQL Во-вторых, поддерживаются таблицы-каталоги, содержащие метаданные, описывающие все объекты базы данных и при этом ими же являющиеся. В-третьих, основной набор объектов внешней памяти имеет простую регулярную структуру, но при этом во внешней памяти поддерживаются дополнительные управляющие структуры — индексы, позволяющие эффективно выполнять операции SQL языкового уровня. В-четвертых, поддерживается избыточность хранения данных для выполнения требований надежности.

Соответственно, во внешней памяти содержатся: строки таблиц — основная часть БД, большей частью видимая пользователю, управляющие структуры служебная и журнальная информация.

Существуют два принципиальных подхода к физическому хранению таблиц.

Наиболее распространенным является построчное хранение, при котором единицей физического хранения является строка таблицы. Требуется, чтобы строка целиком хранилась в одном блоке внешней памяти, что обеспечивает быстрый к ней доступ, но ведет к дублированию общих значений разных строк и замедлению работы, если строки длинные, а нужна только их часть (частый случай в аналитических базах данных).

Страница данных — это блок данных во внешней памяти, в котором хранятся строки каких-то таблиц. У каждой строки есть уникальный идентификатор tid = <N, i>, где N – номер страницы, I – смещение от начала блока до описателя. В описателе хранится ссылка на начало строки в этой же таблице или (если в процессе работы строка перестала помещаться на странице) ссылка на другую страницу. Число строк в странице не ограничено, для реализации динамического числа строк в странице используют метод двух указателей: область описателей растет сверху вниз, область хранения — снизу вверх.

Для хранения очень больших строк самым простым решением является использование отдельных файлах, другим возможным решением — хранение длинных строк в отдельном наборе страниц внешней памяти, связанном физическими ссылками. Оба этих способа ставят под вопрос как надежность хранения, так и скорость работы с длинными данными. Еще одним методом является хранение длинных данных на основе B-деревьев, а в сегодняшнее время поддерживается внутренняя файловая система внутри SQL-сервера.

Обычно в одной странице хранятся строки одной таблицы, но возможен и обратный случай, повышающий эффективность некоторых операций, но требующий хранения дополнительной служебной информации.

При добавлении нового столбца в таблицу физической реорганизации таблицы не происходит, но в описатель строки добавляется новый столбец. Строки расширяются только при занесении информации в новое поле.

Еще один нетривиальный момент — распределение памяти в страницах данных. К примеру, если в ходе транзакции блок диска освободился, то он не может быть использован другой транзакцией до подтверждения первой из-за возможности отката.

Еще одним способом повышения эффективности СУБД является кластеризация таблицы — указание, что в одном блоке внешней памяти надо хранить те строки таблицы, у которых значения столбца кластеризации близки. Тогда некоторые запросы выполняются быстрее, но со временем кластеризация ухудшается и требует физической реструктуризации таблицы. Также возможна совместная кластеризация. Обратным подходом является декластеризация, используемая с целью использования возможностей распараллеливания обменов с внешней памятью.

Альтернативным подходом является хранение таблицы по столбцам, что экономит память и дает возможность оптимизировать операции соединения и быстрее считать значения агрегатных функций, но, очевидно, требует дополнительных действий для сборки целой строки. Для каждой строки таблицы здесь хранится строка, состоящая из ссылок на места расположения соответствующих значений столбцов.

Билет 33. **Индексы на основе B-деревьев**

Основным назначением индексов является обеспечение эффективного прямого доступа к строке таблицы по ключу. Обычно индекс определяется для одной таблицы, и ключом является значение ее поля. Если ключом индекса является возможный ключ таблицы, то индекс должен обладать свойством уникальности. Полезным свойством индекса является обеспечение последовательного просмотра строк таблицы в заданном диапазоне значений ключа в порядке возрастания или убывания его значения. Общей идеей любой организации такого индекса является хранение упорядоченного списка значений ключа с привязкой к каждому значению списка идентификаторов строк.

Наиболее популярным методом организации индексов в базе данных является использование B-деревьев. B-дерево — это сбалансированное сильно ветвистое дерево во внешней памяти, сбалансированность значит, что длина пути от корня дерева к любому его листу одна и та же. Ветвистость — это свойство каждого узла дерева ссылаться на большое число узлов-потомков. С точки зрения физической организации B-дерево представляется как мультисписочная структура страниц внешенй памяти, то есть каждому узлу дерева соответствует блок внешней памяти. Недостаток B-деревьев состоит в трудности балансировки.

Модификацией B-дерева является B+-дерева, которое достаточно просто балансировать. В B+ - дереве внутренние и листовые страницы имеют разную структуру. Типовая структура внутренней страницы выглядит как упорядоченная последовательность ключей и ссылок на узлы более низкого уровня, причем узел, заключенный между двух ключей, содержит только ключи, заключенные между этих ключей. Листовая страница выглядит как упорядоченная последовательность ключей и списков идентификаторов строк, причем в n-м списке содержатся строки с n-м ключом.

Поиск в B+ дереве есть прохождение от корня к листу в соответствии с заданным значением ключа. Поскольку B+ деревья сильно ветвисты и сбалансированы, для поиска требуется одно и то же обычно небольшое число обменов с внешней памятью. Более точно, если во внутренней странице помещается n ключей, то для хранения m записей требуется дерево глубиной logn(m).

Основным плюсом B+ -деревьев является автоматическое поддержание свойства сбалансировасти. Если при вставке новой записи закончилось место в дереве, выполняется расщепление страницы, куда производилась запись, и результат примерно пополам записывается в две страницы. Если не осталось места для записи ключа (чтобы к новой странице можно было добраться), производится аналогичная операция уровнем выше, и так может дойти до корневой страницы (но на самом деле такое происходит очень редко). С удалением аналогично, но вместо расщепления производится слияние. Стоит заметить, что для повышения эффективности следует производить расщепление и слияние не в тот момент, когда место закончилось совсем, а так, чтобы иметь некоторый запас.

С B+-деревьями из-за автоматической балансировки сложно работать в режиме мультидоступа, так как блокировать все дерево затратно (потому что оно сильно ветвистое и по факту проблемы могут и не возникнуть), а для неполной блокировки надо знать, насколько вверх поднимется расщепление.

Билет 34. **Индексы на основе хэширования: расширяемое и линейное хэширование**

Основным назначением индексов является обеспечение эффективного прямого доступа к строке таблицы по ключу. Обычно индекс определяется для одной таблицы, и ключом является значение ее поля. Если ключом индекса является возможный ключ таблицы, то индекс должен обладать свойством уникальности.

Альтернативным по отношению к использованию B-деревьев подходом организации индексов является использование техники хеширования. Общей идеей является применение к значению ключа некоторой функции свертки, вырабатывающей значение меньшего размера. Значение хэш-функции затем используется для доступа к записи. Основным требованием к хэш-функции является равномерное распространение значений свертки. При возникновении коллизий (одна и та же свертка для разных ключей) образуются цепочки переполнения. Главным ограничением здесь является фиксированный размер таблицы. Если таблица слишком сильно заполнена или переполнена, то время поиска по цепочке переполнения сведет на нет главное преимущество хэширования — доступ к записи почти всегда за одно обращение. Расширение таблицы требует ее полной передлки на основе новой хэш-функции.

Двумя наиболее часто используемыми методами хэширования являются расширяемое и линейное хэширования.

В основе подхода расширяемого хэширования лежит принцип использования деревьев цифрового поиска в оперативной памяти. В ОП поддерживается справочник, организованный на основе бинарного дерева цифрового поиска, ключами которого являются значения хэш-функции, а в листовых вершинах хранятся номера страниц записей во внешней памяти. Здесь под коллизией понимается переполнение страницы внешней памяти, и в этом случае страница расщепляется на две, и дерево цифрового поиска переформируется. При этом может потребоваться расширение справочника.

Метод расширяемого хеширования заключается в том, что хеш-таблица представлена как каталог, а каждая ячейка будет указывать на бакет, который имеет определенную вместимость. Сама хеш-таблица будет иметь глобальную глубину G, а каждая из емкостей имеет локальную глубину li . Глобальная глубина G показывает сколько последних бит будут использоваться для того чтобы определить в какую емкость следует заносить значения. А из разницы локальной глубины и глобальной глубины можно понять сколько ячеек каталога ссылаются на емкость. Алгоритм для ключа k с значением свёртки h(k):

1) По первым G битам свёртки h(k) решаем в какой бакет отправить ключ.
2) Если в бакете есть свободное место, то помещаем туда ключ, если бакет переполнен, смотрим на локальную глубину бакета:
    2.1) Если локальная глубина меньше, чем глобальная глубина, то создаём новый бакет и заново перераспределяем все ключи в текущем бакете с учётом новой длины бит. Не забыть увеличить глубину обоих бакетов на 1. Возвращаемся к шагу 1.
    2.2) Если локальная глубина равна глобальной, то мы увеличиваем глобальную глубину на 1, удваивая при этом количество ячеек, количество указателей на бакеты, а также увеличиваем количество последних бит по которым мы распределяем значения.

Основой метода линейного хэширования является то, что для адресации страницы внешней памяти всегда используются младшие биты значения хэш-функции. Если возникает потребность в расщеплении, то записи перераспределяются по страницам так, чтобы адресация осталась правильной.

Билет 35. **Интерфейс ядра SQL-ориентированных СУБД - RSS**

На уровне RSS отсутствует именование объектов базы данных уровня SQL. Вместо имен объектов используются их уникальные идентификаторы, являющиеся прямыми или косвенными адресами внутренних описателей объектов в памяти. Замена имен на идентификаторы производится компилятором SQL на основе информации из системных таблиц. Можно выделить следующие группы операций: сканирования, создания и уничтожения, модификации содержимого, модификации заголовков, управления транзакцией, явной синхронизации.

Операции группы сканирования позволяют последовательно в некотором порядке прочитать кортежи таблицы или списка, удовлетворяющие требуемым условиям. Возможно прямое сканирование, требующее только идентификатор таблиц, предполагающее последовательный просмотр всех страниц сегмента, с выделением кортежей, входящих в данную таблицу, что дорого. Порядок выборки определяется физическим размещением кортежей. Возможно сканирование через индекс, позволяющее к тому же указать диапазон сканирования. Процесс сканирования состоит в последовательном продвижении по листовым вершинам индекса между границами диапазона с выбором идентификаторов кортежей и чтения соответствующих кортежей. В худшем случае может потребоваться столько чтений из внешней памяти, сколько идентификаторов кортежей было встречено. Порядок сканирования соответствует порядку сортировки ключей иднекса. При сканировании списка сканирование максимально эффективно, так как читаются только страницы, содержащие кортежи из данного списка в порядке занесения кортежей в список. В результате успешного выполнения операции открытия сканирования получается идентификатор сканирования, позволяющий работать дальше — получать следующий кортеж, соответствующий некоторому условию, или закрыть сканирование.

Операции создания и уничтожения требуют в качестве параметра в случае создания спецификатор структуры объекта, и идентификатор объекта в случае уничтожения.

Для модификации объектов нужно иметь идентификатор объекта и параметры модификации. При модификации производится коррекция всех индексов, определенных на таблице. После удаления кортежа из таблицы сканирование указывает на промежуток между двумя кортежами, из которого надо сдвинуться для дальнейшей работы.

Каждая операция RSS выполняется в пределах некоторой транзакции. Интерфейс RSS включает команды начала, окончания транзакции, установки точки сохранения и отката. При выполнении транзакций пользователи изолированы друг от друга за счет наличия в RSS механизма неявной синхронизации. До конца транзакции никакие изменения базы данных, произведенные в пределах этой транзакции не могут быть использованы в других транзакциях (попытка приведет к временной синхронизационной блокировке).

Операция явной синхронизации позволяет установить явную синхронизационную блокировку таблицы, что гарантирует, что никакая другая транзакция до конца данной не сможет изменить эту таблицу или даже прочитать (в зависимости от режима блокировки)

Билет 36. **ACID-транзакции. Средства СУБД для поддержки свойств атомарности, согласованности, изолированности и постоянства хранения.**

Корректное поддержание транзакций является одновременной основой обеспечения целостности баз данных и изолированности пользователей в многопользовательских системах. В соответствии с понятием ACID под транзакцией понимается последовательность операций над базой данных, обладающая следующими свойствами:

Атомарность — результаты всех операций, успешно выполненных в пределах транзакции, или отражены все в состоянии базы данных, или не отражена ни одна. Это позволяет относится к транзакции как к составной операции над базой данных. Для поддержки ограничений целостности допускается их нарушение внутри транзакции с тем условием, что к моменту завершения транзакции условия целостности должны быть соблюдены. Если это не так, вместо фиксации результатов транзакции происходит ее откат. Различают немедленно проверяемые и откладываемые ограничения целостности. При нарушении немедленно проверяемого ограничения отвергается соответствующий оператор (в момент попытки исполнения), при неудовлетворении откладываемому ограничению целостности вместо фиксации транзакции происходит откат.

Согласованность — транзакция может быть успешно завершена с фиксацией результатов своих операций только если действия операций не нарушают целостность базы данных. Расширением этого свойства является разрешение во время выполнения транзакции устанавливать точки согласованности и явным образом проверять ограничения целостности.

Изоляция — две одновременно выполняемые транзакции никоим образом не должны действовать одна на другую. Предельной задачи СУБД является создание иллюзии того, что каждый из пользователей работает с базой данных в одиночку. Существует несколько уровней изолированности, каждый из которых должен гарантировать отсутствие аномалий определенного типа (каждый следующий содержит предыдущие). Первый уровень отвечает за отсутствие потерянных изменений — случай, когда транзакция изменяет базу данных, но в том же месте изменение производит другая транзакция, которая затем откатывается, но откат производится к начальному состоянию, то есть первая транзакция не видит своих изменений объекта. Второй уровень отвечает за отсутствие чтения грязных данных — случая, когда транзакция видит данные, измененные незавершенной другой транзакцией. Третий уровень отвечает за отсутствие неповторяющиеся чтений — транзакция дважды читает одно и то же, но получает разные результаты из-за завершения какой-то другой транзакции. Еще возможная более тонкая проблема кортежей-фантомов, возникающая в процессе выполнения одной транзакцией оператора выборки по условию, выполняемого два раза, в промежутке между которыми возникает новый кортеж, подходящий под условия за счет завершения какой-то другой транзакции.

Долговечность — после успешного завершения транзакции все изменения должны гарантированно сохраняться даже в случае сбоев, точнее, должна быть возможность восстановления согласованного состояния базы данных после любого роба сбоев. Для этой возможности необходимо хранение избыточной информации, что чаще всего сейчас поддерживается с помощью механизма журнализации изменений.

Билет 37. **Сериализация транзакций, виды конфликтов транзакций и порождаемые ими феномены поведения транзакций. Двухфазный протокол синхронизационных блокировок.**

Чтобы добиться изолированности транзакций, в СУБД должны использоваться какие-либо методы регулирования совместного выполнения транзакций. План выполнения набора транзакций называется сериальным, если результат совместного выполнения эквивалентен результату некоторого последовательного выполнения этих транзакций. Сериализация транзакций — механизм их выполнения по некоторому сериальному плану. Обеспечение такого механизма является основной функцией компонента СУБД, ответственного за управление транзакциями. Основная проблема реализации состоит в выборе метода сериализации, которые не слишком сильно ограничивал бы чередование их операций или реальную параллельность.

Между транзакциями T1 и T2 могут существовать следующие виды конфликтов: W/W – T2 пытается изменять объект, измененный не завершившейся T1 (влечет потерянные изменения (транзакция изменяет базу данных, но в том же месте изменение производит другая транзакция, которая затем откатывается, но откат производится к начальному состоянию, то есть первая транзакция не видит своих изменений объекта)). R/W – T2 пытается изменять объект, прочтенный не завершившейся T1 (неповторяющиеся чтения (транзакция дважды читает одно и то же, но получает разные результаты из-за завершения какой-то другой транзакции)). W/R – T2 пытается читать объект, измененный не закончившейся T1 (грязное чтение (транзакция видит данные, измененные незавершенной другой транзакцией)).

Наиболее распространенным в централизированных СУБД является подход, основанный на соблюдении двухфазного протокола синхронизационных захватов объектов баз данных. В общих чертах подход состоит в том, что перед выполнением любой операции от имени транзакции запрашивается синхронизационная блокировка объекта в соответствующем режиме (чтение или запись), при этом совместимы только режимы чтения. Транзакция, затребовавшая несовместимую блокировку, блокируется до снятия блокировки с объекта. Для обеспечения сериализации требуется снимать блокировки только по завершении транзакции-инициатора, поэтому выполнение транзакции разбивается на две фазы — накопление блокировок и снятие блокировок.

Отдельный вопрос — что считать объектом для синхронизационного захвата, так как любая операция над объектом базы данных фактически воздействует и на объемлющие его объекты. В большинстве современных систем используются покортежные синхронизационные блокировки.

Билет 38. **Гранулированные и предикатные блокировки.**

При применении этого подхода синхронизационные блокировки могу запрашиваться по отношению к объектам разного уровня. Требуемый уровень объекта определяется выполняемой операцией. Объект может быть заблокирован в режиме S (совместный) или X (монопольный). Для согласовании блокировок разного уровня вводится специальный протокол гранулированных блокировок и новые типы блокировок. Перед установкой блокировки на некоторый объект соответствующий объект верхнего уровня должен быть заблокирован в режиме IS (intended for share lock, объект нижнего уровня собираются блокировать в режиме S), IX (intended for exclusive lock, объект нижнего уровня собираются блокировать в режиме X), SIX (shared, intended for exclusive lock, объекты нижнего уровня собираются блокировать в режиме X).

Совместимость видов блокировок имеет достаточно большой размер, но выводится очевидным образом :) (no). Для атомарных объектов разумны только блокировки в режимах S и X (из которых совместимы только S/S). Если некоторый составной объект находится в режиме X, с ним ничего больше делать нельзя (его собираются менять целиком). Если составной объект в режиме S, его можно использовать в режиме S или IS. Режим IX совместим с IS и IX. Режим IS сместим со всем, кроме X, а SIX только с IS.

Метод гранулированных синхронизационных захватов не решает проблему фантомов, так как для решения этой проблемы требуется перейти от блокировок индивидуальных объектов базы данных к блокировке условий (предикатов), которым эти объекты удовлетворяют. Проблема фантомов не возникает при использовании для блокировок уровня таблиц именно потому, что таблица есть неявное условие для входящих в нее кортежей.

Одним из вариантов является подход, основанный на том, что при открытии сканирования таблицы по индексу в RSS передается дополнительная информация (диапазон сканирования), которая ограничивает множество кортежей, среди которых не должны возникать фантомы. То есть можно допускать совместную работу над таблицей транзакций, диапазоны сканирований которых не пересекаются. Еще одним полезным элементом здесь является преобразование оператора SQL со сложным условием в последовательность обращений к подсистеме управления памятью с простыми условиями (вида имя_поля op Значение, где op =, <, >. Более того, простое условие явно указывается в операции открытия сканирования таблицы, при открытии сканирования можно указать, для какой цели оно будет использоваться, и неявные условия задаются операциями вставки и удаления кортежей, а также операциями обновления кортежей. Поэтому простые условия можно использовать как основу предикатных захватов.

Билет 39. **Синхронизационные тупики, способы их обнаружения и разрушения.**

Одним из самых больших недостатков метода сериализации транзакции на основе синхронизационных блокировок является возможность возникновения тупиков между транзакциями. Тупик — это ситуация, когда (в самом простом случае) есть две транзакции, желающие получить доступ в несовместимых режимах к двум объектам, и каждая получила доступ к одному. При попытке получить доступ к второму объекту они вынуждены ждать друг друга, то есть никогда не прийти к успеху.

Основой обнаружения тупиков является построения графа ожидания транзакций — ориентированного двудольного графа, в котором одним типом вершин являются транзакции, другим — объекты блокировки. Дуги соединяют только разнотипные вершины. Дуга из транзакции к объекту значит, что транзакция заблокировала объект. Дуга из объекта в транзакцию значит, что транзакция хочет заблокировать объект. Очевидно, что если есть цикл, то есть тупик. Для распознавания тупиков периодически строится граф и производится поиск тупиков. Традиционной техников поиска циклов в ориентированном графе является редукция графа. Удаляются все дуги, исходящие из транзакций, в которые не входят дуги — эти транзакции уже получили все блокировки и не могут оказаться в тупике. Удаляются дуги, входящие в транзакции, из которых не исходит дуг — эти транзакции еще ничего не блокируют, поэтому не могут стать частью тупика. Для тех объектов, для которых не осталось входящих дуг, но существуют исходящие, ориентация одной из них изменяется на противоположную (это моделирует удовлетворение запроса блокировки), и действия повторяются, пока граф не перестанет меняться. Если в графе останутся дуги, то они образуют цикл.

Разрушение тупика начинается с выбора в цикле транзакций транзакции-жертвы — той транзакции, которой решено пожертвовать, чтобы обеспечить возможность продолжения работы других транзакций. Выбрать жертву сложно, так как существует много противоречивых критериев. С одной стороны, убив транзакцию, удерживающую наибольшее число блокировок объектов, мы освободим много объектов, что с большой вероятностью приведет к решению тупика, с другой стороны на убитую транзакцию было потрачено много системных ресурсов, завершать ее неразумно с системной точки зрения. Можно убивать самую молодую транзакцию, которая израсходовала мало ресурсов, но одновременно у нее и мало блокировок, так что она с большой вероятностью не решит тупик. Можно выбирать транзакцию случайным образом, что в среднем даст хороший результат, но здесь не учитывается возможная приоритетность транзакций.

После выбора жертвы ее откатывают (полностью или частично). Это нарушение принципа изолированности пользователей, но избежать этого нельзя.

Билет 40. **Сериализация транзакций на основе временных меток**

Основной идеей метода временных меток является следующее: если транзакция T1 началась раньше T2, то система обеспечивает такой сериальный план, как если бы T1 была полностью выполнена до начала T2. Для этого каждой транзакции T предписывается временная метка t(T), соответствующая времени начала выполнения транзакции T. При выполнении операции над объектом транзакция помечает его своим идентификатором, временной меткой и типом операции (чтение или изменение). Перед выполнением операции над объектом транзакция T2 выполняет следующие действия: проверяет, помечен ли объект. Если нет, метит и продолжает работать. Иначе проверяет, завершилась ли пометившая транзакция. Если завершилась, помечает и продолжает работать. Иначе проверяет конфликтность операций — если операции совместимы, то запоминается идентификатор и время T2 и транзакция продолжает работать. Если операции конфликтуют, то если пометившая объект транзакция моложе, то производится откат ее и всех других транзакций, идентификаторы которых сохранены при объекте, и T2 продолжает работать. Если пометившая объект транзакция старше, то производится откат транзакции T2, T2 получает новую временную метку и начинает заново.

К недостаткам этого метода относят потенциально более частые откаты транзакций, чем при использовании синхронизационных захватов, что связано с более грубым определением конфликтности транзакций. Кроме того, в распределенных системах сложно работать с временными метками, чтобы они были корректны. Но зато не нужно распознавать тупики, а построение графа ожиданий в распределенных системах очень дорогое.

Билет 41. **Версионный вариант алгоритма временных меток**

Основной идеей версионных алгоритмов является допущение существования в базе данных нескольких версий одного и того же объекта, что позволяет, главным образом, выполнять операции чтения над некоторой предыдущей версией объекта базы данных. В результате чтение выполняется без задержек и тупиков, а также без некоторых откатов.

В алгоритме MVTO (Multiversion Timestamp Ordering) порядок выполнения операций одновременно выполняемых транзакций задается порядком временных меток, которые транзакции получают в момент старта. Временные метки также используются для идентификации версий данных при чтении и модификации — каждая версия получает временную метку той транзакции, которая ее записала. Алгоритм не только следит за порядком выполнения операций транзакций, но и отвечает за трансформацию операций над объектами в операции над версиями объектов.

Временная метка, полученная транзакцией T в начале работы, будет обозначаться t(T), операция чтения объекта в транзакции Ti – Ri(O), для обозначения чтения i-й транзакции версии, созданной k-й транзакции будем использовать Ri(Ok), запись транзакцией Ti версии объекта — Wi(Oi).

Алгоритм работает следующим образом:
Ri(O) → Ri(Ok), где Ok – версия объекта, помеченная наибольшей временной меткой t(Tk) такой, что t(Tk) <= t(Ti), то есть версия, созданная транзакцией, которая не моложе Ti, но самая молодая из создававших версии объекта.
Wi(O): если есть незафиксированная Tn, выполнившая Rn(Ok): t(Tk) <= t(Ti) < t(Tn), то запись не выполняется, Ti откатывается. Иначе Wi(O) → Wi(Oi), то есть образуется еще одна версия объекта.

При откате любой транзакции уничтожаются все созданные ею версии объектов базы данных и откатываются все транзакции, прочитавшие хотя бы одну из этих версий. То есть откаты транзакций могут быть каскадными.

Выполнение операции фиксации откладывается, пока не завершатся все транзакции, записавшие версию данных, прочитанную Ti. Без этого условия не соблюдалось бы свойство долговечности, так как пришлось бы откатывать успешно завершившиеся транзакции.

Основным преимуществом данного метода является отсутствие задержек и откатов при выполнении чтения, а недостатков — возможность возникновения каскадных откатов при записи и накопление версий одного и того же объекта, для которых сложно определить, какие из них больше не нужны.

Билет 42. **Версионный вариант двухфазного протокола синхронизационных блокировок**

Основной идеей версионных алгоритмов является допущение существования в базе данных нескольких версий одного и того же объекта, что позволяет, главным образом, выполнять операции чтения над некоторой предыдущей версией объекта базы данных. В результате чтение выполняется без задержек и тупиков, а также без некоторых откатов.

В версионном варианте двухфазного протокола синхронизационных блокировок поддерживается до двух версий объектов базы данных. Текущей версией объекта называется версия, созданная зафиксированной транзакцией с наиболее поздним временем фиксации. Незафиксированной версией — версию, созданную еще не завершившейся транзакцией. В каждый момент времени может существовать не более одной незафиксированной версии каждого объекта.

Операции выполняются следующим образом: чтение ведется из текущей версии объекта. Запись, которая приводит к созданию новой версии объекта, выполняется только после завершения (фиксации или отката) транзакции, создавшей незафиксированную версию объекта. Выполнение фиксации транзакции откладывается до тех пор, пока не завершатся все транзакции, читавшие текущие версии объектов, которые завершающаяся транзакция должна заменить незафиксированными версиями.

Для реализации такого поведения используются специальные виды синхронизационных блокировок: ReadLock – блокировка объекта перед чтением его текущей версии, гарантирует, что при повторном чтении будет прочитано то же самое. WriteLock – любой объект блокируется в этом режиме перед операцией, приводящей к созданию новой версии объекта, гарантирует, что в любой момент времени будет существовать не более одной незафиксированной версии объекта. CommitLock – блокировка на время фиксации транзакции, затрагивает все объекты, чью новую версию эта транзакция создавала. Удовлетворение этой блокировки гарантирует, что все транзакции, которые читали объекты, которые были изменены текущей транзакцией, завершились.

Совместимы RL/RL (очевидным образом), RL/WL и WL/RL (за счет существования двух версий, то есть сохранения версии для чтения). Тупики возможны. (Например, если две транзакции хотят изменить два объекта, и каждая из них успела захватить по одному объекту, они обе должны ждать, пока другая не завершиться).

Билет 43. **Версионно-блокировочный протокол сериализации транзакций для поддержки только читающих транзакций.**

Основной идеей версионных алгоритмов является допущение существования в базе данных нескольких версий одного и того же объекта, что позволяет, главным образом, выполнять операции чтения над некоторой предыдущей версией объекта базы данных. В результате чтение выполняется без задержек и тупиков, а также без некоторых откатов.

При применении указанного протокола при создании транзакции явно указывается ее тип — read-only или update, читающая или изменяющая.

Изменяющиеся транзакции выполняются в соответствии с обычным двухфазным протоколом синхронизационных блокировок, то есть перед выполнением операции над объектом он блокируется в соответствующем режиме (чтение или запись), и блокировка удерживается до конца транзакции Каждая запись объекта создает его новую версию, которая при завершении транзакции-создателя помечается временной меткой, соответствующей моменту фиксации транзакции.

Только читающая транзакция при создании получает временную метку. При чтении объекта транзакция получает доступ к его версии, которая была образована транзакцией, хронологически последней зафиксировавшейся до начала текущей транзакции.

Плюсом данного протокола по отношению с версионным вариантом двухфазного протокола является полное отсутствие синхронизационных задержек при работе только читающих транзакций. К тому же эти транзакции никогда не откатываются. Минусом является появление в базе данных произвольного числа версий объектов (вместо двух у двухфазного протокола). Поэтому нужен сборщик мусора, который будет удалять ненужные версии данных. Простейший сборщик удаляет все неиспользуемые версии, временные метки которых меньше временной метки самой старой из живых читающих транзакций.

Билет 44. **Ситуации, требующие восстановления базы данных. Индивидуальные откаты транзакций. Понятие журнала.**

Одним из основных требований к СУБД является надежность хранения баз данных, в том числе возможность восстановления целостного состояния базы данных после сбоев. Для возможности восстановления необходимо хранить некоторую избыточную информацию, которая в большинстве случаев поддерживается в виде журнала изменений.

Основой поддержания целостности базы данных является механизм транзакций, поэтому журнализация и восстановление тесно связаны с понятием транзакции. Во-первых, результаты зафиксированных транзакций должны быть в восстановленной базе данных (то есть транзакция должна быть долговечной). Во-вторых, в восстановленной базе данных не должно быть следов незафиксированных транзакций (потому что иначе база не будет согласованной). То есть восстанавливается хронологически последнее согласованное состояние базы данных.

Восстановление базы данных может быть необходимо в нескольких ситуациях.

Во-первых, при индивидуальном откате транзакции, то есть явном ее завершении соответствующим оператором или инициации этого завершения системой при ошибке со стороны пользователя (к примеру, некорректной операции) или выборе транзакции жертвой при решении синхронизационного тупика. Для восстановления целостности базы данных при индивидуальном откате транзакции нужно устранить последствия операторов модификации базы данных, которые выполнялись в этой транзакции.

Во-вторых, при мягких сбоях — при потере содержимого оперативного памяти в случае аварийного выключения питания или сбоя процессора. При мягком сбое теряется та часть базы данных, которая находилась в буферах оперативной памяти СУБД.

В-третьих, при жестких сбоях — поломке основного внешнего носителя, то есть потере или повреждении его содержимого. Сейчас внешние носители достаточно надежны, так что такая ситуация не является частой, но никто от нее не застрахован. Основой для восстановления в этом случае является архивная копия (которая, очевидно, должна храниться на другом надежном носителе) и журнал изменений.

Журнал изменений — это специальный набор данных, хранящий последовательность записей обо всех изменениях базы данных. Возможны два основных подхода к ведению журнала. Можно поддерживать отдельный локальный журнал для каждой транзакции, и тогда эти журналы будут использоваться для индивидуальных откатов и могут поддерживаться в основной памяти СУБД, и вдобавок поддерживать общий журнал изменений для восстановления после сбоев. Тогда индивидуальные откаты бкдкт выполняться быстро, но информация в локальных и общем журналах будет дублироваться. А можно поддерживать только общий журнал, используя его для всех вариантов восстановления.

Для обеспечения возможности индивидуального отката транзакции по общему журналу все записи журнала о данной транзакции связываются в обратный список. Для незавершенных транзакций в начале списка находится запись о последнем изменении, транзакцией произведенном, для закончившихся — запись о конце транзакции. При этом если транзакция закончилась, то записи журнала о ней обязательно вытолкнуты во внешнюю память, то есть сохранены надежно, что требуется для обеспечения долговременности транзакций. Важно отметить, что индивидуальный откат может быть выполнен только для незавершенных

транзакций (потому что результат завершенных уже может быть использован другими транзакциями, и попытка откатить завершенную транзакцию повлечет каскадный откат транзакций, что противоречит их изолированности и надежности). Для выполнения индивидуального отката последовательно в обратном хронологическом порядке выполняются операции, противоположные операциям, взятым из журнала для данной транзакции (удаление вместо вставки, вставка вместо удаления и так далее). Эти обратные операции тоже журнализируются на случай мягкого сбоя во время индивидуального отката. При успешном завершении индивидуального отката в журнал заносится запись о конце транзакции, то есть с точки зрения журнала транзакция становится зафиксированной.

Билет 45. **Управление буферами основной памяти.**

Одним из основных требований к СУБД является надежность хранения баз данных, в том числе возможность восстановления целостного состояния базы данных после сбоев. Для возможности восстановления необходимо хранить некоторую избыточную информацию, которая в большинстве случаев поддерживается в виде журнала изменений.

Журнализация операций изменения базы данных связана не только с управлением транзакциями, но и с буферизацией блоков базы данных в основной памяти. Процессоры и основная память работают значительно быстрее устройств внешней памяти, поэтому для достижения приемлемой эффективности СУБД нужно буферизировать блоки базы данных в основной памяти.

Если бы каждое изменение журнала выталкивалось во внешнюю память, тогда СУБД фактически работала бы со скоростью внешнего устройства всегда при выполнении операций обновления. Это не очень эффективно (хотя если в силу особенностей использования большинство взаимодействий с базой данных требуют только чтения, то есть записи редки, но требуется большая надежность — почему бы и нет?), поэтому журнал тоже буферизируется — выталкивается во внешнюю память только при полном заполнении. Более того, обычно используются несколько буферов, потому что в случае одного буфера на время выталкивания во внешнюю память придется сделать паузу для всех операций журнализации. Поэтому подбирается такое число буферов, чтобы журнал мог работать непрерывно.

Стоит заметить, что речь идет именно о буферах физической основной памяти, управляемой непосредственно СУБД, а не виртуальной памяти СУБД, управляемой системой, поскольку логика работы с памятью СУБД и ОС различается. В правильно организованных СУБД поддерживается собственная стратегия замещения страниц буферного пула. В случае операционной системы, если процессу требуется страница виртуальной памяти, но свободных страниц нет, по некоторому критерию выбирается страница, которая будет освобождена. В случае СУБД, если при выполнении некоторой операции в некотоой транзакции нужен доступ к блоку базы данных, копии которого в буферном пуле нет, то СУБД должна выделить страницу буферного пула, считать в нее этот блок и предоставить его запросившей операции. Если буферный пул заполнен, то какая-то его страница будет освобождена.

Основная проблема состоит в критерии выбора страницы-жертвы. ОС зачастую выбирает жертвой страницу, к которой предположительно дольше всего не будет обращений, то есть, к примеру, страницу, к которой давно никто не обращался. Для СУБД тоже часто используется какая-то разновидность такого выбора, но СУБД имеет больше информации о своих страницах, чем ОС.

Например, если выполняется сканирование таблицы без использования индекса и был затребован некоторый блок базы данных, то подсистема управления буферным пулом догадывается, что эта страница будет нужна, пока не будет прочитан последний кортеж сканируемой таблицы, на ней располагаемый. Более того, известно, какой блок базы данных потребуется после завершения просмотра данного блока, то есть этот блок может быть заранее запрошен.

Кроме того, некоторые блоки заведомо требуются чаще других — в частности корневые блоки структур индексов. Поэтому в стратегии замещения страниц буферного пула обычно используется механизм Last Recently Used с приоритетами страниц (то есть с разной скоростью их старения).

Билет 46. **Физическая синхронизация.**

Одним из основных требований к СУБД является надежность хранения баз данных, в том числе возможность восстановления целостного состояния базы данных после сбоев. Для возможности восстановления необходимо хранить некоторую избыточную информацию, которая в большинстве случаев поддерживается в виде журнала изменений.

Журнализация операций изменения базы данных связана не только с управлением транзакциями, но и с буферизацией блоков базы данных в основной памяти. Процессоры и основная память работают значительно быстрее устройств внешней памяти, поэтому для достижения приемлемой эффективности СУБД нужно буферизировать блоки базы данных в основной памяти.

В СУБД могут одновременно выполняться несколько транзакций, что может повлечь необходимость одновременного доступа к одному блоку от двух операций (то есть к одной и той же буферной странице). Вообще говоря, логическая синхронизация, используемая для сериализации транзакций, не обеспечивает координацию параллельного доступа к страницам буферного пула. К примеру, при выполнении двух операций изменений кортежей, находящихся на одной странице, если блокировка происходит на уровне кортежей, то параллельное выполнение этих операций системой будет допущено. Но изменение кортежа может потребовать его перемещения внутри страницы, что не приведет ни к чему хорошему, если таких перемещений будет выполняться несколько одновременно. То же самое и с операциями, обновляющими индексы — некоординированное параллельное обновление B-дерева с большой вероятностью его сломает.

Поэтому необходимо поддерживать дополнительную «физическую» синхронизацию, в которой единицами блокировки служат страницы буферного пула базы данных. Но такие блокировки нужны не только для координации параллельного доступа к страницам при параллельном выполнении транзакций. Могут возникать ошибки, которые обнаружатся только в середине операции уровня RSS, когда было изменено уже несколько страниц буферного пула. (Например, попытка вставки кортежа в таблицу с несколькими индексами, из которых один не позволит эту вставку из-за нарушения свойства уникальности. При обнаружении такой ошибки нужно убрать все ее следы. Проще всего провести обратные изменения всех страниц, но для этого требуется сохранять блокировку страниц до конца операции.

То есть для подсистемы управления буферным пулом операции уровня RSS являются почти тем же, что транзакции для подсистемы управления транзакциями. Достаточным условием корректного выполнения операций является соблюдение двухфазного протокола синхронизационных блокировок над станицами буферного пула в пределах операций. Можно заметить, что необходимым это условие не является — каждую операцию уровня RSS можно разбить на последовательность микроопераций, и производить блокировку уже в их пределах. Общий принцип следующий: в пределах одной микрооперации блокируются все блоки базы данных, которые обязаны быть изменены согласовнным образом.

Билет 47. **Протокол Write Ahead Log.**

Одним из основных требований к СУБД является надежность хранения баз данных, в том числе возможность восстановления целостного состояния базы данных после сбоев. Для возможности восстановления необходимо хранить некоторую избыточную информацию, которая в большинстве случаев поддерживается в виде журнала изменений.

Журнализация операций изменения базы данных связана не только с управлением транзакциями, но и с буферизацией блоков базы данных в основной памяти. Процессоры и основная память работают значительно быстрее устройств внешней памяти, поэтому для достижения приемлемой эффективности СУБД нужно буферизировать блоки базы данных в основной памяти.

Имеются два вида буферов — буфера журнала и буферный пул страниц основной памяти. Они содержат связанную информацию. Оба буфера могут выталкиваться во внешнюю память. Журнальным буфер выталкивается при его полном заполнении, страницы буферного пула основной памяти — при необходимости переместить в буфер новые страницы, места для которых нет. Проблема заключается в выработке общей политики выталкивания, которая обеспечила бы возможность восстановления состояния базы данных после сбоев.

Для индивидуальных откатов транзакций этой проблемы нет, потому что при индивидуальном откате содержимое ни оперативной, ни основной памяти не утрачено, и при восстановлении можно пользоваться любым из буферов. Но если произошел мягкий сбой и буфера потерялись, то для восстановления целостности базы данных состояния журнала и базы данных во внешней памяти должны быть согласованны.

Основным принципом является то, что запись об изменении объекта базы данных должна оказаться во внешней памяти журнала раньше, чем измененный объект окажется во внешней памяти базы данных. Этот протокол называется Write Ahead Log: если нужно вытолкнуть во внешнюю память буферную страницу, содержащую измененный объект базы данных, то перед этим надо гарантировать выталкивание во внешнюю память журнала буферной страницы журнала, содержащей запись об изменении этого объекта.

При следовании протоколу WriteAheadLog если во внешней памяти базы данных есть некоторый объект, который был модифицирован, то во внешней памяти журнала есть запись, описывающая эту операцию. Обратно неверно.

Дополнительным условием является требование, что каждая успешно завершенная транзакция должна быть реально зафиксирована во внешней памяти. При любом сбое должно быть возможно восстановление до состояния базы данных, содержащего результаты всех транзакций, зафиксированных до сбоя.

Минимальным требованием, гарантирующий возможность такого восстановления, является выталкивание при фиксации транзакции во внешнюю память журнала всех записей об изменении базы этой транзакции. При этом последней записью является специальная запись о конце транзакции.

Билет 48.**Физически согласованное состояние базы данных. Восстановление базы данных после мягкого сбоя.**

Одной из основных проблем при восстановлении после мягкого сбоя является то, что одна логическая операция изменения базы данных может изменять несколько физических блоков базы данных, например, блок данных и несколько блоков индексов. Разные блоки буферизуются и выталкиваются независимо, что может привести к несогласованности внешней памяти после мягкого сбоя, когда часть блоков соответствует объекту до изменения, а часть — после.

Состояние внешней памяти базы данных называется физически согласованным, если наборы страниц всех объектов согласованны, то есть соответствуют состоянию любого объекта или до его изменения, или после. Будем считать, что точки физической согласованности базы данных отмечаются в журнале. Тогда восстановление после мягкого сбоя имеет следующий вид, в зависимости от отношения времени транзакции и точки согласованности.

Если транзакция завершилась до точки согласованности, то ничего делать не надо, так как все ее результаты гарантированно находятся во внешней памяти. Если транзакция началась после точки согласованности, и не успела завершиться то тоже ничего делать не надо, так как ее результатов во внешней памяти нет никаких. Если такая транзакция завершиться успела, то нужно повторить для нее все операции в прямом направлении (раз транзакция завершилась, то в журнале есть все нужные записи). Если транзакция началась до точки согласованности и завершилась до сбоя, то нужно повторно выполнить операции, произведенные после точки согласованности, потому что во внешней памяти есть только среды операций, произведенных до точки согласованности (а так как транзакция завершилась, то у нас есть журнал со всеми нужными данными). Если транзакция началась до точки согласованности, но не успела завершиться до сбоя, то нужно отменить (выполнить в обратном порядке) операции, произведенные до сбоя (они у нас есть из-за точки согласованности, и транзакция должна быть отменена, так как не успела завершиться).

Билет 49. **Способы восстановления физически согласованного состояния.**

Одной из основных проблем при восстановлении после мягкого сбоя является то, что одна логическая операция изменения базы данных может изменять несколько физических блоков базы данных, например, блок данных и несколько блоков индексов. Разные блоки буферизуются и выталкиваются независимо, что может привести к несогласованности внешней памяти после мягкого сбоя, когда часть блоков соответствует объекту до изменения, а часть — после.

Состояние внешней памяти базы данных называется физически согласованным, если наборы страниц всех объектов согласованны, то есть соответствуют состоянию любого объекта или до его изменения, или после.

Для обеспечения наличия точек физической согласованности используются два основных подхода: использование теневого механизма и журнализация постраничных изменений базы данных.

Теневой механизм был изначально предложен для поддержания целостности файлов при мягких сбоях. Для файла, представимого как набор блоков, при модификации любого блока во внешней памяти выделяется новый блок, где эта модификация происходит. Текущая таблица отображений логических блоков в физические изменяется, а теневая остается прежней. Если происходит сбой, во внешней памяти остается теневая копия, содержащая состояние файла до его открытия. Для восстановления файла достаточно прочитать заново его теневую таблицу отображения.

В контексте базы данных теневой механизм используется так: периодически выполняются операции установки точки физической согласованности, при которых завершаются все логические операции и выталкиваются все страницы буферов, содержимое которых было изменено. Теневая таблица отображения заменяется текущей (текущая становится теневой). Здесь есть проблема, связанная с необходимостью атомарности операции этой замены, чтобы в любой момент времени теневая таблица отображения была корректной. Если в процессе операции возникнет мягкий сбой, то и текущая таблица будет утрачена, и теневая повреждена. Чтобы этого не произошло, во внешней памяти поддерживаются две области хранения таблицы отображений, и область начинает использоваться только после полного изменения. Флаг, показывающий, какую область надо хранить, хранится в одном блоке, а запись на внешний диск одного блока считается атомарной, так что здесь проблем нет. Недостаток теневого метода в том, что производится очень большой перерасход внешней памяти.

Журнализация постраничных изменений производится вместе с логической журнализацией операций изменения базы данных. После мягкого сбоя первый этап состоит в постраничном откате недовыполненных логических операций.

Чтобы распознать, нуждается ли страница внешней памяти базы данных в восстановлении, при выталкивании страницы из буферного пула в нее помещается номер последней записи о постраничном изменении этой страницы, он же запоминается в самой записи. Тогда чтобы понять, надо ли применять данную запись о постраничном изменении, достаточно сравнить номер, содержащийся в блоке, с номером, содержащимся в журнальной записи. Если в блоке номер меньше, то это значит, что страница вытолкнута во внешнюю память не была, и применять запись не требуется.

В подходе с журнализацией постраничных изменений имеются два поднаправления. В первом поддерживается общий журнал логических и страничный операций, имеющий более сложную структуру и больший размер. Во втором подходе поддерживается отдельный жрунал постраничных изменений, называемый физическим. Логический и физический журнал имеют разную природу, во-первых, логический должен поддерживать выполнение операций как в прямом, так и в обратном порядке, а физический только в обратном. Во-вторых, логический журнал начинает заполняться заново только после резервного копирования базы данных или архивирования журнала, до этого времени он растет. Предельный размер журнала определяется администратором базы данных и должен согласовываться с интервалом времени, через которое производится резервное копирование. Физический журнал существует сравнительно недолго и занимает меньше места. При установлении точки физической согласованности прекращают инициироваться новые логические операции, после завершения всех выполняемых происходит выталкивание во внешнюю память модифицированных страниц буферного пула, затем во внешнюю память логического журнала выталкивается запись о точке физической согласованности, и если все прошло успешно, то физический журнал пишется заново и можно работать дальше. Если выталкивание записи о точке физической согласованности прошло неуспешно, это воспринимается как мягкий сбой.

Билет 50. **Архивация базы данных и журнала. Восстановление базы данных после жесткого сбоя.**

Журнал изменений — это специальный набор данных, хранящий последовательность записей обо всех изменениях базы данных. Потенциальное переполнение логического журнала регулируется так: устанавливаются желтая и красная зоны. При достижении желтой зоны запрещается образование новых транзакций. Если работавшие транзакции не успеют завершиться до достижения красной зоны, то они будут откачены. После завершения транзакций (или отката), производится архивация базы данных или журнала.

Самым простым способом архивирования является создание копии базы данных по указанию администратора или при переполнении журнала. Но можно выполнять архивацию базы данных реже, чем переполняется журнал, архивируя журнал вместо базы данных. Тогда при восстановлении достаточно иметь исходную архивную копию, последовательность архивных журналов и последний логический журнал. В лоб выполнять все имеющиеся в журналах операции долго, но их можно сильно оптимизировать, выделив для каждого объекта последовательности журнальных записей в хронологическом порядке и заменив их одной записью-результатом. Также можно совместить несколько последовательных журналов, полных или сжатых. То есть остается исходная архивная копия, один сжатый архивный журнал и последний логический журнал. При этом работа по сжатию журналов может выполняться по ходу их создания, а не в сам момент восстановления.

При жестком сбое теряются данные из внешней памяти. Очевидно, в таком случае журнала изменений недостаточно, нужна архивная копия базы данных. При восстановлении сначала копируется база данных из архивной копии, затем для всех завершившихся транзакций их операции выполняются (более точно — выполняются все операции из журнала и откат для тех транзакций, которые не успели завершиться). После этого получается хронологически последнее до момента жесткого сбоя логически согласованное состояние базы данных.

При некоторой дисциплине выполнения логических операций базы данных при восстановлении можно выполнять операции из журнала последовательно, не обращая внимания на принадлежность их разным транзакциям. В частности, если сериализация транзакций основана на блокировках объектов, то запись в буфере логического журнала об изменении объекта появляется только после удовлетворения блокировки, и только после записи выполняется сама операция изменения.

Вообще говоря, можно восстановиться после жесткого сбоя не только до последнего логически согласованного состояния, но и до момента сбоя, то есть получить возможность продолжать незаконченные транзакции. Но так обычно не делают, потому что восстановление после жесткого сбоя — долгая операция.

Если журнал операций утрачен, то все, что можно сделать — вернуться к архивной копии.