1Концепция баз данных

Под термином **концепция** в этом разделе понимается определённый способ понимания, трактовки каких-либо явлений; основная точка зрения, руководящая идея. Итак, основные концепции БД:

– **Отчуждение данных от программ.**

Данные должны быть независимы от программ. Структура данных не влияет на программы и наоборот.

– **Хранение описания данных вместе с самими данными**

Необходимо, чтобы любой пользователь (человек или программа) мог узнать какая информация храниться в БД, какова структура этих данных, какие типы имеют конкретные данные. Такая служебная информация (метаданные) должны храниться в самой БД - это сильно упрощает реализацию концепции отчуждения данных от программ.

– **Отчуждение данных от носителей.**

Пользователь не знает, где хранятся данные (диски, серверы). Доступ к информации происходит без знания физического расположения.

– **Поддержание БД в согласованном (целостностном) состоянии.**

Согласованность БД (database integrity) — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам. В большинстве случаев ограничения согласованности определяются особенностями предметной области. Одна из важнейших задач БД - поддержание в любой момент времени взаимной непротиворечивости, правильности и точности хранящихся данных

– **Защита информации**.

Защита информации — это очень широкое понятие, охватывающее такие процессы, как защита от сбоев аппаратной или программной природы, защита от злонамеренного или незлонамеренного искажения и порчи информации, защита от ошибок программ или пользователей БД.

– **Поддержка многозадачной (многопользовательской) работы.**

БД должна поддерживать одновременную работу множества пользователей. Обеспечивает эффективный доступ к данным без конфликтов.

2Не реляционные модели данных. Примеры реализации.

Нереляционная база данных — термин, обозначающий ряд подходов, направленных на реализацию хранилищ данных, в которых делается попытка решить проблемы быстродействия, масштабируемости и доступности данных за счет их атомарности и согласованности. Крупные корпорации, оперирующие гигантскими

объемами данных, разработали несколько нереляционных СУБД и это мода была активно поддержана IT-сообществом. Вскоре, однако, выяснилось, что нереляционные СУБД поддерживают весьма узкий круг задач, связанных с накоплением, очисткой и выборкой малосвязанных данных

3Реляционная модель данных. Таблицы, записи, поля, связи.

Реляционной является БД, в которой все данные доступные пользователю, организованы в виде набора связанных двумерных таблиц, а все операции над данными сводятся к операциям реляционной алгебры.

Таблица в реляционной модели данных соответствует (содержит данные) одной сущности предметной области и состоит из фиксированного числа **полей**, собранных в **записи**, каждая запись соответствует экземпляру сущности

Студенты

Номер студенческого билета	*ФИО*	*Дата рождения*	*Пол*	*Номер группы*
135236	Иванов И.И.	31.12.2000	М	ИУ6-21
772899	Петрова П.П.	12.12.2000	Ж	ИУ6-21
566334	Кузнецов К.К.	11.11.2001	М	ИУ6-22
122344	Иванов И.И.	12.12.2000	М	ИУ6-23

Запись: Представляет собой конкретный экземпляр данных в таблице. Каждая запись содержит информацию о сущности и хранится в виде строки в таблице. Например, если у нас есть таблица "Студенты", каждая запись будет представлять отдельного студента, и каждая строка в таблице будет содержать данные об этом студенте.

Поля: Представляют собой конкретные характеристики или атрибуты сущности. Каждый столбец в таблице соответствует определенному атрибуту, и каждая ячейка в столбце содержит значение атрибута для конкретной записи. Продолжая пример с таблицей "Студенты", атрибутами могут быть "ФИО", "Пол", "Номер группы", и так далее.

Можно различать **связи** между двумя сущностями по количеству экземпляров присутствующих в связи с разных сторон:

1.Связь много к одному - много студентов учатся в одной группе;

2.Связь один ко многим – один студент изучает много предметов;

3.Связь один к одному - врач является пациентом;

4.Связь много ко многим - пациенты ходят на прием к врачам,

4Базы данных и СУБД. Определение реляционной СУБД.

База данных (**БД**) представляет собой структурированное хранилище данных, предназначенное для эффективного сохранения, организации и извлечения информации. **Система управления базами данных** (**СУБД**) является программным обеспечением, предназначенным для управления созданием, обновлением и обращением к данным в базе данных.

Реляционная модель баз данных и **РСУБД** гарантируют, что все операции манипулирования данными обладают ACID свойствами, с одной стороны это обеспечивает постоянную согласованность БД, с другой стороны мешает обеспечить высокую доступность и быстродействие особенно в случае, когда данные распределены по нескольким серверам.

Атомарность гарантирует, что транзакция считается выполненной только в том случае, если все ее операции выполнились успешно. Если хотя бы одна операция транзакции завершится неудачно, то все изменения, внесенные предыдущими операциями, откатываются.

Согласованность гарантирует, что транзакция переводит базу данных из одного согласованного состояния в другое. То есть, если база данных была в согласованном состоянии до начала транзакции, она должна оставаться согласованной после ее завершения.

Insulativity - изолированность. Каждая транзакция, которая выполняется, не зависит от остальных. Изоляция означает, что несмотря на параллельный доступ к базе данных множества транзакций, каждая из них должна проходить изолированно от остальных. Свойство изоляции транзакций гарантирует, что транзакция не увидит изменений, внесенных другими транзакциями в БД, пока они выполняются

Duration - надежность. Все результаты, которые были достигнуты в ходе успешной транзакции, наверняка сохраняются в базе данных. Как только транзакция завершена, база данных должна гарантировать, что ее результаты не будут потеряны. Это свойство обеспечивается механизмами восстановления базы данных, гарантируя сохранность всех зафиксированных транзакций.

5Основные функции СУБД

Концепции БД	Функции СУБД
Отчуждение данных от носителей.	Управление данными во внешней памяти
Отчуждение данных от программ.	Поддержка языков БД
Хранение описания данных вместе с самими данными	Ведение словаря БД
Поддержание БД в согласованном (целостном) состоянии.	Обеспечение согласованности БД Управление транзакциями Управление блокировками и клинчами
Защита информации	Управление журналами изменений в БД Управление транзакциями Обеспечение безопасности БД
Поддержка многопользовательской (многозадачной) работы	Управление транзакциями Управление блокировками и клинчами

Управление данными во внешней памяти:

Обеспечивает эффективное хранение и доступ к данным, которые могут не помещаться целиком в оперативной памяти, используя механизмы работы с внешней памятью.

Поддержка языков БД:

Предоставляет средства для работы с языками запросов и манипуляции данными, обеспечивая пользовательский интерфейс для взаимодействия с базой данных.

Ведение словаря БД:

Управляет метаданными и словарем базы данных, который содержит информацию о структуре, типах данных, связях и других характеристиках данных.

Обеспечение согласованности БД:

Гарантирует, что данные в базе находятся в согласованном состоянии, соблюдая правила целостности и предотвращая нарушения структуры данных.

Управление транзакциями:

Обеспечивает выполнение транзакций как неделимых операций, гарантируя их успешное выполнение или откат в случае сбоя, с использованием свойств ACID.

Управление блокировками и клинчами:

Контролирует доступ к данным в параллельных транзакциях, используя механизмы блокировок и клинчей для предотвращения конфликтов.

Управление журналами изменений в БД:

Регистрирует изменения, внесенные в базу данных, в виде журналов, обеспечивая восстановление данных в случае сбоев и поддерживая надежность транзакций.

Обеспечение безопасности БД:

Контролирует доступ к данным, обеспечивая аутентификацию и авторизацию пользователей, а также предотвращая несанкционированный доступ и изменение данных.

6Транзакции. Свойства транзакции.

Транзакция — логическая единица работы, состоящая из одного или нескольких операторов SQL, которую СУБД рассматривает и обрабатывает как неделимое действие, переводящее БД из одного согласованного состояния в другое согласованное состояние. Допускается, что в процессе транзакции согласованность может нарушаться, но извне транзакции этого не видно.

Свойство транзакции это ACID 4 вопрос

7Журналирование. Зачем нужны журналы транзакций?

Обеспечение надежности и безопасности хранения данных всегда требует дополнительных затрат как памяти, так и быстродействия СУБД, а любое восстановление после сбоя требует дополнительной информации. Такая информация хранится в журналах изменений, т.е. при любых операциях, совершаемых СУБД, сведения об изменениях данных заносятся в специальные журналы изменений. Все изменения в реляционной БД происходят в рамках транзакций, поэтому **журнал изменений** часто называют **журналом транзакций.**

Журнализация транзакций - функция СУБД, суть которой в сохранении информации, необходимой для восстановления базы данных в согласованное состояние в случае логических или физических сбоев

В простейшем случае журнализация транзакций заключается в последовательной записи во внешнюю память всех изменений, выполняемых в базе данных. Записывается следующая информация:

- индикаторы начала транзакции (порядковый номер, тип и время изменения);
- пользователь от имени которого проводились изменения;
- код транзакции на языке SQL;
- имя объекта, подвергшегося изменению (например, прикладной таблицы);
- предыдущее состояние объекта и новое состояние объекта;
- индикаторы фиксации, показывающие, была ли завершена транзакция, и, если да, то когда.

8Восстановление СУБД после сбоев.

Сбой — утрата системой работоспособности.

Сбои в работе СУБД, вызывает неправильно написанное программное обеспечение (программные ошибки), неверные инструкции, переданные оператором (ошибки пользователя), отказ аппаратуры компьютера (аппаратные ошибки), внешние воздействия на аппаратный комплекс (катастрофы).

Мягкий сбой системы обычно возникает при аварийном выключении электрического питания, при возникновении сбоя аппаратной части сервера, не затрагивающего устройства долговременного хранения. При таком сбое данные, хранящиеся на дисках, остаются неповрежденными, но утрачивается содержимое буферов базы данных в оперативной памяти.

Для восстановления состояния оперативной памяти используются данные **«точки согласованности»**. В журнале транзакций отмечаются точки физической согласованности базы данных, так называемые точки согласованности – моменты времени, когда в долговременной памяти содержатся данные точно соответствующие состоянию оперативной памяти, и при этом в долговременную память полностью вытолкнут буфер журнала транзакций. По сути «точка согласованности» — это дамп памяти сервера вместе с некоторой служебной информацией. В момент точки согласованности содержимое ОЗУ согласованно с содержимым долговременной памяти (не путайте с согласованностью БД). Сервер ORACLE обычно делает точки согласованности не реже, чем раз в две секунды.

На рис. 2.22 представлена ситуация мягкого сбоя: на оси времени отмечены собой, в результате которого утеряно содержимое оперативной памяти и момент создания ближайшей к сбою точки согласованности. Возврат состояния ОЗУ к точке согласованности – первое действие, которое осуществляет СУБД после мягкого сбоя, т.е. в очищенную память «заливается» содержимое точки согласованности, сохраненное в журнале изменений СУБД.

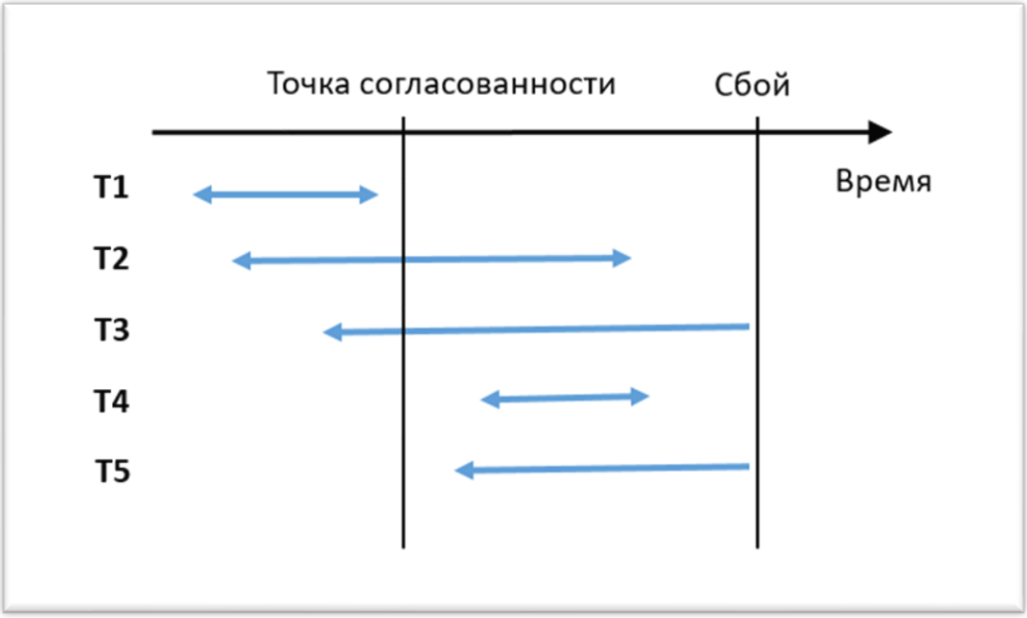


Рис. 2.22. Восстановление СУБД после мягкого сбоя

Отметим, что после восстановления ОЗУ в памяти находится информация о промежуточных результатах незавершенных транзакций. После восстановления ОЗУ СУБД анализирует журнал транзакций и проделывает следующие операции:

1.Откатываются все транзакции, которые начались до точки согласованности и окончились после нее – как T2 или вовсе не закончились, как T3. Это действие выполняется по журналу undo log и очищает ОЗУ от промежуточных результатов незавершенных транзакций.

2.Накатываются все транзакции, которые завершились после точки согласованности – такие как T2 и T4. Это действие выполняется по журналу redo log.

3.Запускаются на выполнение транзакции, которые выполнялись в момент сбоя, такие как T3 и T5, т.е. не имеющих отметок о завершении.

4.Для транзакции типа T1, которые закончилась до момента «точки согласованности» никаких действий производить не требуется, их результаты уже надежно сохранены в базе данных.

После выполнения этих действий СУБД полностью восстанавливается и может продолжать работу.

Жесткий сбой системы возникает из-за разрушения данных записанных на устройства долговременного хранения. Отказ систем хранения данных в настоящее время это очень редкое событие, гораздо чаще возникает ситуация, когда некомпетентные или неосторожные действия пользователя приводят к утрате данных. Процесс восстановления БД после жесткого сбоя показан на рис. 2.23.

Предположим, что жесткий сбой СУБД произошёл в момент времени t1 и обнаружен был не сразу, а в момент t2, когда пользователи сообщили администратору СУБД о невозможности работы. После анализа ситуации администратор останавливает СУБД в момент t3 и приступает к устранению результатов сбоя.

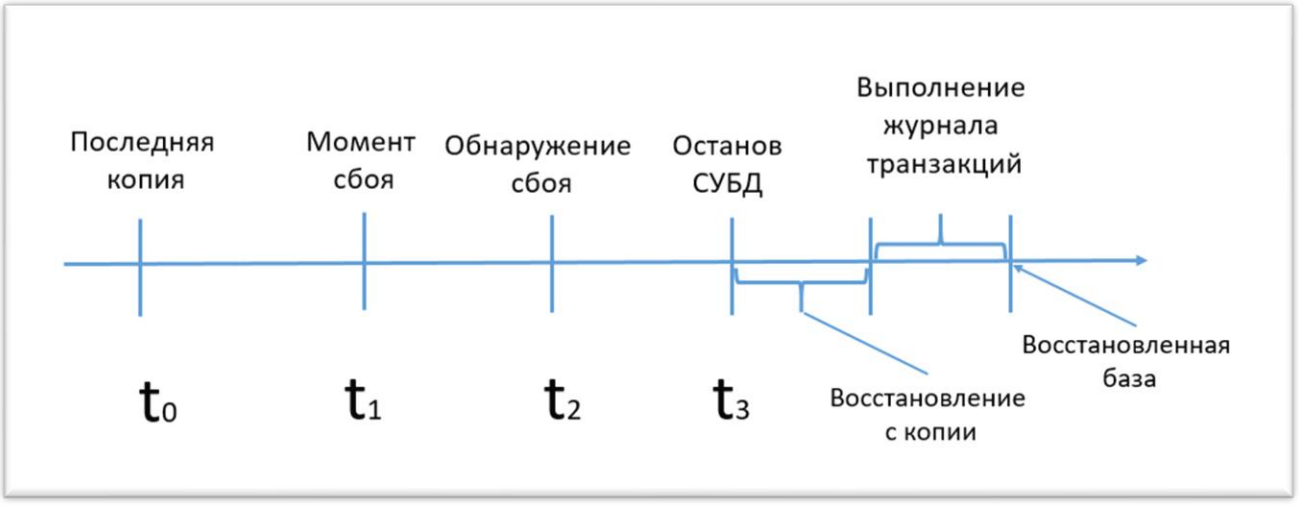


Рис. 2.23. Процесс восстановления БД после жесткого сбоя

Теперь надо восстановить состоянии БД с копии сделанной в момент t0 (о копировании БД см. раздел 2.2.4). Восстановление с резервной копии может быть проведено средствами операционной системы или средствами СУБД, запущенной уже в однопользовательском режиме. Остается только выполнить все транзакции, записанные в журнал archived log с момента t0 до момента t3. Если к сбою привела ошибка пользователя, то перед выполнением журнала необходимо убрать оттуда транзакцию, которая привела к уничтожению или искажению данных. Выполнение (или как говорят «накат») журнала может занять очень большое время, если последняя резервная копия создана давно, поэтому политика резервного копирования должна учитывать сроки возможного восстановления БД после жесткого сбоя.

Под **катастрофой** мы будем здесь понимать крупное неблагоприятное событие (авария, стихийное бедствие, пожар и др.), влекущее за собой полный отказ узла сети, без возможности его быстрого восстановления. Надо понимать, что обычно крупные узлы больших систем размещаются на специальных объектах, называемых ЦОД (Центр Обработки Данных). ЦОДы (как и сети ЦОДов) создаются таким образом, чтобы минимизировать риски от катастроф. Единственным выходом при потере ЦОДа в результате катастрофы, являет переход на резервный ЦОД, который должен содержать реплики серверов вышедшего из строя ЦОДа (про репликацию см. раздел 2.2.11). Создание сетей ЦОДов это прежде всего инженерно-строительная задача и ее рассмотрение выходит за границы данной книги. Более подробно см. [23].

9Архитектура СУБД. Взаимодействие СУБД с клиентом.

Под **архитектурой СУБД** понимается совокупность ее функциональных компонентов, а также средств обеспечения их взаимодействия друг с другом, с пользователями и с системным персоналом

Трехуровневая архитектура систем баз данных ANSI/SPARC:

1. Внешний уровень (Пользовательский):

Описание: Этот уровень представляет данные с точки зрения пользователей. Каждый пользователь работает с определенным представлением данных, которое содержит только необходимую ему информацию.

Пример: Бухгалтер видит данные о продажах в стоимостном выражении, а начальник склада видит объемы продаж в штуках.

2. Промежуточный уровень (Концептуальный):

Описание: Концептуальный уровень обобщает представление данных с общей точки зрения. Он описывает структуру базы данных, сущности, их атрибуты, и связи между ними. Этот уровень поддерживает все внешние представления. Логическая схема бд

Пример: Описание всех сущностей, их атрибутов и связей, ограничения на данные, безопасность и согласованность данных.

3. Внутренний уровень (Физический):

Описание: Этот уровень представляет физическое хранение данных в памяти компьютера. Он описывает, как данные организованы на конкретных устройствах хранения.

Пример: Распределение дискового пространства, сведения о размещении записей, методы сжатия данных.

Важные принципы:

Логическая независимость: Возможность изменять одно приложение без влияния на другие приложения, работающие с той же базой данных.

Физическая независимость: Возможность переноса хранимой информации с одних носителей на другие при сохранении работоспособности всех приложений.

Файл-серверная архитектура представляет собой старую и несовершенную форму физической организации информационных систем с использованием систем управления базами данных (СУБД). В такой архитектуре файлы данных размещаются централизованно на файл-сервере, а каждый клиентский компьютер (рабочая станция) содержит свою собственную СУБД. Доступ к данным осуществляется через локальную сеть, где файлы с информацией передаются между сервером и клиентами.

Преимущества:

- Низкая нагрузка на процессор файлового сервера: Так как обработка данных происходит на клиентских компьютерах, нагрузка на сервер остается относительно низкой.

Недостатки:

- Высокая загрузка локальной сети: Передача файлов по сети может привести к высокой загрузке, особенно при интенсивном обмене данными.
- Затрудненное централизованное управление: Управление и обслуживание системы затруднены из-за децентрализованной структуры.
- Сложности с надежностью, доступностью и безопасностью: Обеспечение высокой надежности, доступности и безопасности данных затруднительно.

Применялась чаще всего в локальных приложениях, использующих функции управления БД; в системах с низкой интенсивностью обработки данных и низкими пиковыми нагрузками на БД.

Клиент-серверная архитектура — это концепция построения информационных систем, при которой функциональность системы разделена между клиентской и серверной частями. Пользователь взаимодействует с клиентской частью, которая обрабатывает запросы пользователя и преобразует их в язык SQL. Серверная часть выполняет запросы и возвращает результаты клиенту, который отображает данные пользователю.

Архитектура клиент-сервер обладает несколькими вариантами размещения слоев обработки данных на клиенте и сервере. Эти варианты включают тонкий и толстый клиент. Тонкий клиент переносит большую часть задач по обработке информации на сервер, тогда как толстый клиент выполняет большую часть работы на стороне клиента.

Слои клиент-серверной архитектуры:

Слой представления данных: Отвечает за представление данных пользователю. Может располагаться как на клиенте, так и на сервере.

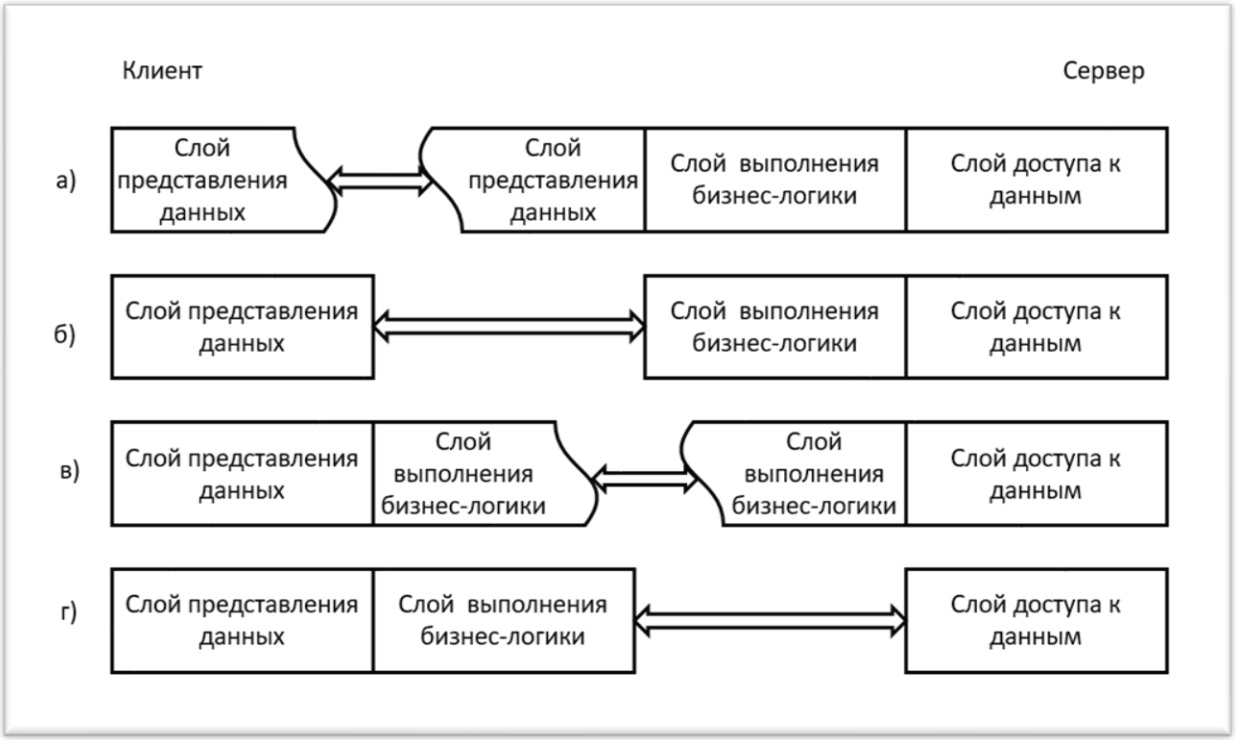
Слой обработки бизнес-логики: Занимается выполнением бизнес-логики приложения. Располагается как на клиенте, так и на сервере.

Слой доступа к данным: Отвечает за доступ к данным. Обязательно располагается на сервере, так как управляет хранилищем данных.

Толстый клиент: Приложение, которое обеспечивает расширенную функциональность и переносит большую часть обработки данных на клиентский компьютер. Сервер в основном служит хранилищем данных.

Тонкий клиент: Компьютер и программа-клиент, переносящая большую часть обработки данных на сервер. Клиент осуществляет лишь отображение данных и взаимодействие с пользователем.

Клиент-серверная архитектура имеет преимущества, такие как снижение требований к пропускной способности сети и упрощение поддержки, но также сопряжена с недостатками, такими как зависимость от работоспособности сервера и сложность масштабирования.



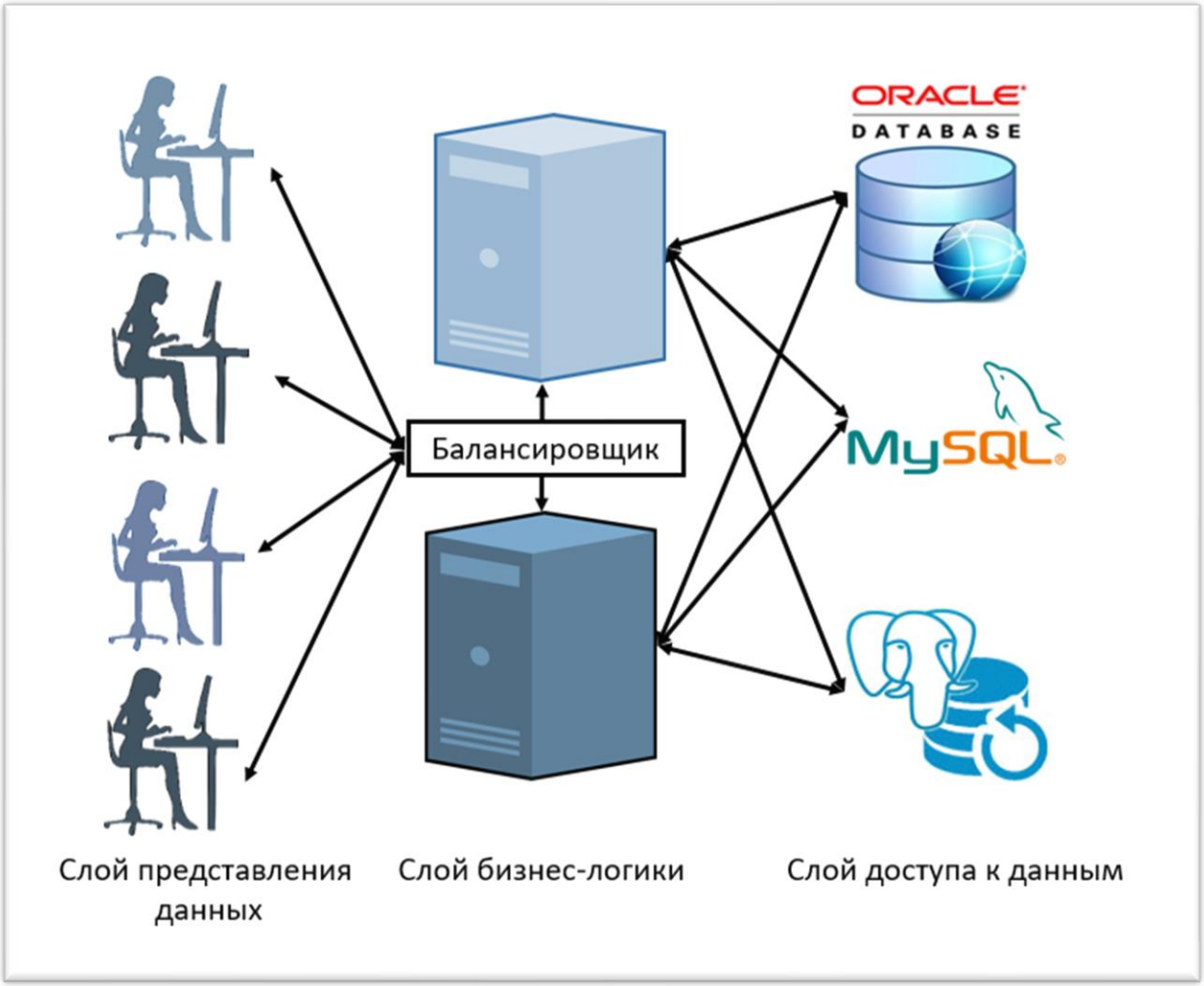
Постепенно в среде разработчиков IT-решений родилось понимание, что три слоя обработки данных можно разнести на три физических уровня – так родилась трехзвенная архитектура.

Трехзвенная по сути подвид клиент-сервер

Трехзвенная — устраняет недостатки двухзвенной(клиент-сервер) архитектуры, располагая каждый из слоев на отдельном узле. Теперь на клиенте находится только пользовательский интерфейс со средствами вывода данных и ввода команд. По сути, клиент превращается в терминал и называется «тонкий клиент». За выполнение вычислений и формирование запросов к СУБД отвечает сервер приложений, а слой доступа к данным в виде СУБД и БД находится на отдельном сервере данных. Таким образом, трехзвенная архитектура устраняет почти все недостатки файл-серверной и клиент-серверной на 2-х звеньях ценой увеличения расходов на администрирование и разработку серверных частей.

На самом деле полную мощь трехзвенной архитектуры иллюстрирует рисунок 2.5, на котором явно определены слои обработки данных. Видно, что при необходимости масштабирования слоя бизнес-логики необходимо просто добавить сервера, подключив их к балансировщику.

Балансировщик нагрузки серверов — аппаратная схема или один сервер, распределяющий запросы к серверам приложений. Использование балансировщика повышает отказоустойчивость и позволяет равномерно распределять нагрузку на сервера.



10Преимущества трезвенной архитектуры и область ее применения.

Преимущества трехзвенной архитектуры весьма значительны и оказывают положительное воздействие на различные аспекты разработки и эксплуатации информационных систем.

1. **Простота модификации:**

Разделение на три слоя облегчает внесение изменений в систему, так как каждый слой может быть модифицирован независимо от других. Это упрощает обновление бизнес-правил, пользовательского интерфейса или слоя доступа к данным.

2. **Простота расширения:**

Добавление новых функциональных возможностей часто требует расширения бизнес-логики. В трехзвенной архитектуре это можно сделать, не затрагивая другие слои, что ускоряет процесс развития системы.

3. **Простота интеграции:**

Каждый из слоев предоставляется в виде сервиса, что упрощает интеграцию с другими системами. Различные слои могут взаимодействовать через стандартизированные интерфейсы.

4. **Повышение безопасности:**

Физическое и логическое разделение слоев повышает уровень безопасности. Бизнес-логика, например, может быть скрыта от прямого доступа извне, что снижает риски.

5. **Низкая стоимость внедрения тонкого клиента:**

Использование тонкого клиента уменьшает затраты на внедрение, поддержку и обновление клиентской стороны системы.

6. **Снижение требований к поддержке клиента:**

Тонкий клиент обычно требует меньше ресурсов на клиентской стороне, что упрощает задачи по его поддержке.

7. **Независимость от операционной системы:**

Тонкий клиент может работать независимо от операционной системы, что улучшает гибкость системы.

8. **Доступность из любой точки мира:**

Использование тонкого клиента обеспечивает доступность системы из любой точки мира, что становится все более важным в современном мире.

Недостатки трехзвенной архитектуры скорее являются платой за преимущества:

- трудность проектирования;
- трудность отладки;
- эксплуатация требует высококвалифицированного персонала

Трехзвенная архитектура широко применяется в различных областях, где требуется высокая гибкость, масштабируемость и безопасность информационных систем.

11Использование индексов и основные сведения о индексах.

Индекс — это вспомогательная таблица, которая формируется из значений одного или нескольких полей индексируемой таблицы и указателей на соответствующие записи. Простейший индекс показан на рис 4.16.

Рис. 4.16. Простейший индекс

Есть разные типы индексов. Простейший - это таблица с копиями полей и указателями. Более распространенными являются индексы, построенные на основе балансированных деревьев (B-деревья). Они оптимизированы для поиска и обеспечивают эффективность при операциях поиска и сортировки.

B-дерево - это структура, в которой каждый элемент содержит диапазон значений и ссылку на следующий элемент. Поиск начинается с корня, и база данных ищет нужный диапазон, используя ссылки и переходя к следующему элементу, пока не найдет нужное значение.

Ваша задача — спроектировать и создать индексы, которые лучше всего подходят для работы с конкретной БД, чтобы оптимизатор запросов мог выбирать из нескольких индексов наиболее эффективный для конкретного запроса.

Необходимо запомнить ряд правил, которые обязательны к исполнению при работе с индексами:

- Индексы создаются для ускорения поиска и выборки.
- Индексы замедляют все операции кроме выборки.

Все операции с таблицами, кроме выборки, изменяют информацию и, следовательно, ведут к перестроению всех индексов, связанных с этими таблицами, что может занять достаточно большое время.

- Индексы занимают место.

Для больших таблиц индексы могут занимать много памяти, а если индексов много, то они могут занять места больше, чем сама таблица.

– Для маленьких таблиц индексы не нужны.

Простой перебор для небольших таблиц (менее 1000 записей) эффективней, чем работа с индексом, а значит индексы для таких таблиц не нужны.

– Для таблиц, которые часто обновляются используйте как можно меньше индексов.

Частое обновление таблицы ведет к частому перестроению индексов и замедляет работу системы.

– Для временных таблиц индексы не нужны.

Временные таблицы часто изменяются и часто очищаются – это ведет к огромной работе по обновлению индексов, которая сводит на нет все преимущества индексации.

– Для составного индекса НЕ важен порядок полей в индексе.

В СУБД начала века, действительно, можно было получить выигрыш при правильной комбинации полей в составном индексе и это упоминается в разной литературе. В настоящее время все СУБД оптимизированы так, что порядок полей в составном индексе не важен.

12Пользователи, роли и разграничение прав доступа. Предопределенные роли пользователей СУБД.

Любые действия в СУБД производятся от имени какого-нибудь пользователя и во время входа система должна его аутентифицировать и предоставить возможность работы в соответствии с выделенными правами.

Дискреционное управление доступом (DAC): Это механизм, который разграничивает доступ между пользователями и объектами базы данных. В случае СУБД Oracle, это может включать:

Роли пользователей: Назначение определенных ролей пользователям, определяющих их права доступа.

Привилегии: Назначение конкретных прав доступа к определенным объектам базы данных.

Группы пользователей: Группировка пользователей с общими правами доступа.

При создании **пользователя** в СУБД ему автоматически **создается схема** – пространство имен для объектов и данных, принадлежащих этому пользователю. Владелец схемы (хозяин) или администратор СУБД могут управлять правами доступа к объектам в схеме.

Пример: Если пользователь "BOOK" создает таблицу "СОТРУДНИКИ", ее полное имя будет "BOOK.СОТРУДНИКИ".

Роль - это набор прав на конкретные объекты БД, предназначенный для выполнения определенных бизнес-процессов. Роли упрощают процедуру раздачи прав, объединяя их в наборы. Например, роль "преподаватель" может включать права на доступ к таблицам и процедурам, необходимым для преподавательской деятельности.

В СУБД ORACLE, сразу после ее развертывания автоматически создаются три предопределённые роли:

- CONNECT — работает с данными (не может создавать и изменять объекты БД).
- RESOURCE — работает с объектами БД (может их создавать, модифицировать, удалять и раздавать на них права).
- DBA — работает с объектами СУБД (пользователями, их схемами, табличными пространствами, устройствами хранения данных и т.д.).

Конечный пользователь ИС, работающий с данными и имеющий все права на эти данные, обладает правами роли **CONNECT**, эта роль и дается ему при создании.

Пользователь, обладающий правами **RESOURCE**, это обычно владелец схемы БД, разработчик ИС, имеет все права на все объекты БД. Он также обладает всеми правами роли CONNECT, хотя никогда не правит данные в работающей системе, работать с данными он может только при отладке системы и при устранении сбойных ситуаций.

Пользователь, обладающий правами **DBA**, это владелец всей схемы СУБД, и имеет все права на все объекты всех схем всех пользователей и, соответственно обладает всеми правами ролей CONNECT и RESOURCE. Пользователь DBA работает в основном с объектами СУБД – создает пользователей, схемы, рабочие пространства и их копии, следит за аппаратурой на, которой работает СУБД. Обычно пользователь DBA не пользуется своими правами, для действий, связанных с производственными задачами других пользователей, единственная ситуация, когда это возможно – устранение последствий сбоев системы.

13Удаление записей и целостность базы данных.

из БД ничего удалять нельзя. Любое удаление данных из БД приводит к невозможности получения аналитических отчетов за период, в котором были удалены данные. Если из базы данных кафедры ВУЗА удалить, например, отчисленных студентов, то нельзя будет получить списки групп, статистику посещаемости и успеваемости за прошлые годы. Исходя из невозможности удаления данных, надо предусмотреть методы поддержания изменений состояния сущностей, атрибутов и даже связей между сущностями в процессе жизни и развития ИС (см. раздел «Поддержка историчности»), а также методы ускорения работы БД при постоянном росте ее объемов (см. разделы «Секционирование таблиц» и «Кластеризация таблиц»). В редких случаях в процессе эксплуатации ИС

приходится удалять некоторые данные, но это делается вне ИС, администратором БД и последствия этих действий целиком ложатся на его плечи.

Иногда записи таблицы удаляют, если они были вставлены по ошибке. К примеру, сведения о заказчике могут быть по ошибке введены дважды. Такая ситуация типична в крупных компаниях, где вводом данных о заказах одновременно занимается множество операторов. Удаление таких записей, чаще всего возникающих из-за нарушения технологии ввода данных, должны выполнять лица занимающие должности начальников операторов ввода, призванные не допускать нарушения технологии и разбирающиеся в каждом отдельном случае нарушений.

Конечно, правило «из БД ничего удалять нельзя» не относится к процессу разработки и отладки ИС, в эти периоды жизненного цикла удаление данных из БД – обычное дело, только надо помнить о согласованности данных.

Согласованность (целостность) базы данных - соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам. Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, называется ограничением согласованности. Такие ограничения должны быть формально объявлены в СУБД, после чего СУБД должна контролировать их выполнение

Примеры правил: каждый студент должен состоять в учебной группе; количество знаков в телефонном номере не должно превышать 15; баланс банка должен сходится; число проданных билетов не превышает число посадочных мест и т. д.

14Триггеры и их использование в СУБД

Триггер - хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой инициируется СУБД при попытке изменения данных в таблице, с которой он связан. Триггеры применяются для обеспечения согласованности данных и реализации сложной бизнес-логики. Все модификации данных производимые в триггере рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера, соответственно при возникновении ошибки в триггере все вызвавшие его транзакции завершаются неудачно. Надо отметить, что триггеры, также, как и другие встроенные процедуры, пишутся на процедурном расширении SQL принятом в конкретной СУБД или на внешних языках, таких как Java или C#.

Использование триггеров:

- Обеспечение целостности данных: Триггеры могут применяться для проверки и обеспечения целостности данных в таблицах.

- Журналирование (Logging): Используются для записи изменений в журнал, что полезно для отслеживания истории изменений.
- Автоматизация бизнес-логики: Триггеры позволяют автоматизировать выполнение действий при определенных событиях.\

Преимущества и недостатки:

- Преимущества: Обеспечение целостности данных, автоматизация процессов, управление безопасностью.
- Недостатки: Возможность злоупотребления, сложность отладки, возможные проблемы с производительностью.

15Ключи и атрибуты. Суррогатные ключи как идеальные первичные ключи.

Атрибуты таблицы: Каждый столбец в таблице представляет атрибут. Например, если у вас есть таблица "Студенты", то атрибутами могут быть "Имя", "Фамилия", "Возраст" и т.д.

Ключи, существующие в таблицах (а каждая таблица соответствует сущности), однозначно определяют экземпляры сущностей:

- **первичный ключ** - это поле таблицы (атрибут сущности) однозначно определяющий запись таблицы (экземпляр сущности), напомним, что первичным ключом может быть только суррогатный ключ;
- **вторичный ключ** - это поле таблицы однозначно определяющее запись таблицы, но не являющееся первичным ключом, например, студента можно однозначно определить по номеру паспорта, номеру зачетки, номеру студенческого билета, номеру пропуска - все эти поля являются вторичными ключами (заметьте, все эти поля уникальны);
- **внешний ключ** - это поле дочерней таблицы содержащее значение первичного ключа внешней, родительской таблицы и, следовательно, однозначно определяющее запись связанной родительской таблицы.

Суррогатные ключи: Это ключи, созданные искусственно и назначаемые системой управления базами данных (СУБД) для уникальной идентификации записей.

Преимущества суррогатных ключей:

- Гарантируют уникальность.
- Постоянны и независимы от изменений данных.
- Упрощают процессы слияния и разделения данных.

16BI системы. Путь данных от получения до анализа.

BI-системы — программные продукты, которые собирают информацию из разных источников, обрабатывают её и представляют в виде удобных отчётов. Они упрощают анализ данных, с их помощью пользователи могут принимать решения быстрее и эффективнее.

В BI-системах работает процесс, называемый ETL: extract, transform, load — извлечение, преобразование и загрузка. Вот как можно описать этапы работы BI-систем:

Сбор данных. BI-системы способны извлекать их из множества разных источников — CRM-систем, баз данных, ERP-систем и даже таблиц в Excel и Google Sheets. Они могут делать это автоматически, могут — по запросу пользователя.

Обработка данных. Все извлечённые данные системы преобразуют в формат, с которым могут работать. Потом структурируют и изучают данные — определяют зависимости и закономерности.

Визуализация данных. BI-системы строят отчёты и визуализируют их: создают презентации, графики, дашборды. Дашборд — это интерактивная информационная панель, на которую выводят важные для пользователей системы показатели.

Готовыми отчётами могут пользоваться аналитики, генеральный директор, руководитель отдела маркетинга, продакт-менеджер и другие специалисты. Одна BI-система может предоставлять отчёты для разных целей — например, выводить дашборды с данными, нужными разным специалистам.

Благодаря BI-системам специалисты быстрее анализируют данные. Не нужно изучать разные отчёты из разных систем, сопоставлять их, вручную считать показатели — всё видно в одном окне. Система BI-аналитики сама обрабатывает данные и визуализирует их.

BI-системы могут работать с любыми данными, нужными бизнесу. Поэтому число задач, которые можно решать с их помощью, велико.

Вот несколько примеров таких задач:

- найти слабые места в бизнес-процессах и оптимизировать их;
- понять потребности клиентов и разработать продукты, которые удовлетворят их;
- проанализировать продажи и найти закономерности — например, в сезонных колебаниях продаж некоторых товаров;
- выяснить, почему снижается чистая прибыль бизнеса и как можно это исправить;
- управлять ресурсами — планировать закупки и поставки на основе данных о предыдущих проектах.

С помощью BI компании могут решать эти и другие задачи намного быстрее, чем компании, в которых нет BI-систем. Поэтому можно сказать, что BI-системы помогают бизнесу стать более адаптивным, эффективным и конкурентоспособным.

17Хранимые (встроенные) процедуры в СУБД. Типы хранимых процедур.

Хранимая процедура — объект БД, представляющий собой набор инструкций, который компилируется один раз и хранится на сервере в словаре БД. Хранимые процедуры пишутся на SQL или на расширении языка SQL специфичного для каждой отдельной СУБД. Эти процедуры могут иметь входные и выходные параметры, локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных (как DDL, так и DML). Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения. Все последние версии СУБД для создания таких модулей допускают использование языков программирования общего назначения, таких как Java или C#.

Хранимые процедуры похожи на определяемые пользователем функции. Основное различие заключается в том, что пользовательские функции можно использовать, как и любое другое выражение в SQL запросе, в то время как хранимые процедуры должны быть вызваны с помощью функции CALL или EXECUTE.

Хранимые процедуры используются для обработки ошибок, обеспечения согласованности БД, реализации особо сложных запросов. **Особой разновидностью** хранимых процедур являются **триггеры**, отличающиеся способом вызова. Более подробно триггеры рассмотрены в следующем разделе.

18Обеспечение живучести и отказоустойчивости. Копирование и репликация.

Копирование данных в контексте баз данных — это процесс создания и сохранения копии информации из базы данных для последующего использования в случае потери данных или необходимости клонирования базы. Это важная практика для обеспечения безопасности данных, восстановления после сбоев и обеспечения тестирования и разработки.

Репликация (replication) это процесс поддержание в согласованном состоянии копий одних и тех же данных на нескольких машинах (узлах, нодах), соединенных с помощью сети.

Копирование предназначено преимущественно для создания резервных копий и обеспечения безопасности данных, тогда как репликация нацелена на распределенный доступ и повышение производительности системы. Разные объемы данных копируется обычно вся БД, репликации для части.

Виды репликаций (не знаю куда засунуть будут здесь)

Поблочная репликация. Журнал упреждающей записи (Write-Ahead Log), также известный как журнал повтора (Redo Log), содержит сведения о серии событий, каждое из которых сопоставляется с транзакцией или записью. В журнале перечислены все блоки, измененные на дисках в результате такого события. В таких системах, как PostgreSQL, где используется этот метод, журнал изменений отправляется непосредственно всем приложениям-получателям, которые и выполняют необходимые изменения на дисках реплик.

Построчная репликация. При построчной репликации операции записи заносятся в журналы репликации ведущего узла как события, указывающие на то, как изменяются отдельные записи таблицы. Указываются поля с новыми данными; данные приводятся как до, так и после изменений, а также отмечаются удаленные записи. В репликах эти данные применяются не для выполнения исходного оператора, а для непосредственного изменения записей.

Разница заключается в том, как упаковываются и передаются изменения. В поблочной репликации изменения группируются и отправляются блоками, тогда как в построчное каждое изменение передается индивидуально. Отправляются именно изменения, а не транзакции.

Логическая репликация (репликация транзакций). При логической репликации каждая транзакция (атомарный блок SQL-операторов) выполняющая изменения в БД, записывается и отправляется от ведущего узла к подписчикам. Это означает, что на каждом из подписчиков будет выполняться весь набор операторов транзакции. При таком способе передачи изменений значительно уменьшается нагрузка на сеть, ведь не надо передавать всех блоков данных – один запрос (несколько килобайт) может вызвать изменения гигабайтов данных. Кроме того, есть некоторые другие интересные возможности. Поскольку на реплики передаются не сами данные, а запросы, вызывающие их изменения, мы можем использовать различную структуру таблиц на мастере и репликах. В частности, может отличаться тип таблицы, сегментирование, кластеризация или набор индексов.

19Что такое биткойн?

Биткойн—это способ передачи стоимости от одного лица к другому без участия третьей стороны, посредника (банка). Он сам по себе не имеет денежного эквивалента стоимости; он обладает не внутренней стоимостью, а очень высокой полезностью. По сути—это инструмент (метод, протокол) для осуществления транзакций—передачи некоторой стоимости от одного субъекта к другому. Это своего рода финансовый интернет

20Какая БД используется в биткойне

!!!

21Где и почему выгодно применение технологии блокчейн

!!!

22Критерии распределенности (по К. Дейту)

Распределенная база данных - это совокупность логически взаимосвязанных баз данных, распределенных в компьютерной сети. Распределенная база данных может объединять базы данных, поддерживающие любые модели (иерархические, сетевые, реляционные и объектно-ориентированные базы данных) в рамках единой глобальной схемы. Подобная конфигурация должна обеспечивать для всех приложений прозрачный доступ к любым данным независимо от их местоположения и формата.

СУРБД – это программный комплекс (СУБД), предназначенный для управления РБД, и позволяющий сделать распределённость прозрачной для конечного пользователя. Прозрачность РБД заключается в том, что с точки зрения конечного пользователя она должна вести себя точно также, как централизованная.

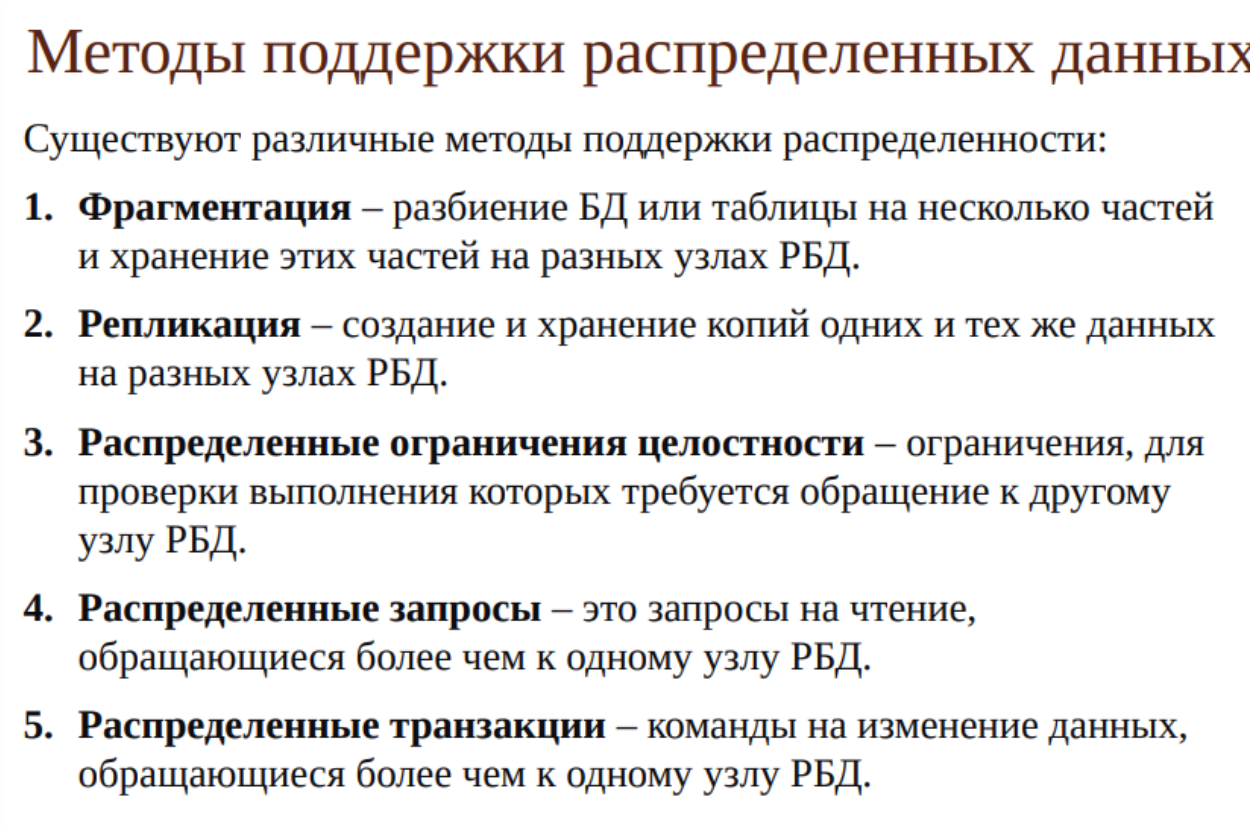
Критерии распределенности (по К. Дейту)

Локальная автономность. Локальные данные принадлежат локальным узлам и управляется администраторами локальных БД.
Локальные процессы в РБД остаются локальными.
Все процессы на локальном узле контролируются только этим узлом.
Отсутствие опоры на центральный узел.
В системе не должно быть узла, без которого система не может функционировать, т.е. не должно быть центральных служб.
Непрерывное функционирование.
Удаление или добавление узла не должно требовать остановки системы в целом.
Независимость от местоположения.
Пользователь должен получать доступ к любым данным в системе, независимо от того, являются эти данные локальными или удалёнными.
Независимость от фрагментации.
Доступ к данным не должен зависеть от наличия или отсутствия фрагментации и от типа фрагментации.
Независимость от репликации.
Доступ к данным не должен зависеть от наличия или отсутствия реплик данных.

Критерии распределенности (по К. Дейту)

Обработка распределенных запросов.
Система должна автоматически определять методы выполнения соединения (объединения) данных.
Обработка распределенных транзакций.
Протокол обработки распределённой транзакции должен обеспечивать выполнение четырёх основных свойств транзакции: атомарность, согласованность, изолированность и продолжительность.
Независимость от типа оборудования.
СУРБД должна функционировать на оборудовании с различными вычислительными платформами.
Независимость от операционной системы.
СУРБД должна функционировать под управлением различных ОС.
Независимость от сетевой архитектуры.
СУРБД должна быть способной функционировать в сетях с различной архитектурой и типами носителя.
Независимость от типа СУБД.
СУРБД должна быть способной функционировать поверх различных локальных СУБД, возможно, с различными моделями данных (требование гетерогенности).

23Методы поддержки распределенных данных



24Типы табличной фрагментации

В реляционных базах данных (РБД) фрагментация относится к процессу разделения таблицы на более мелкие фрагменты или части. Это может быть полезным для улучшения производительности, распределения данных или обеспечения отказоустойчивости. Есть несколько видов фрагментации:

Горизонтальная фрагментация: Данные таблицы разделяются по строкам. Каждый фрагмент содержит некоторые строки таблицы, и эти фрагменты могут храниться на разных серверах или узлах.

Вертикальная фрагментация: Данные таблицы разделяются по столбцам. Каждый фрагмент содержит определенные столбцы таблицы. Это может быть полезно, если различные столбцы используются в различных запросах, и разные фрагменты могут храниться на разных серверах.

Смешанная фрагментация: Данные таблицы разделяются как по строкам, так и по столбцам. Каждый фрагмент содержит определенные строки и выборку столбцов из исходной таблицы. Эти фрагменты могут физически распределяться по разным серверам или узлам, обеспечивая более гибкое управление доступом к данным и оптимизацию выполнения запросов, исходя из конкретных потребностей системы.

25Репликация БД (общая схема)

Репликация (replication) это процесс поддержания в согласованном состоянии копий одних и тех же данных на нескольких машинах (узлах, нодах), соединенных с помощью сети. Существует несколько причин репликации данных:

- для горизонтального масштабирования количества машин, обслуживающих запросы на чтение (и повышения, таким образом, пропускной способности по чтению);
- отказоустойчивость - часто репликация необходима для того, чтобы система могла продолжать работать при отказе некоторых ее частей (и по¬вышения, таким образом, надежности и доступности) т.е. при отказе одного из узлов все запросы переназначаются на реплики;
- ради хранения данных географически близко к пользователям (и сокращения, таким образом, трафика сети и задержек).

На основном сервере формируются журналы изменения данных, которые передаются на репликационный сервер и распространяются на сервера реплики. На серверах репликах

производятся действия по применению журналов для изменения информации точно так же, как и на основном сервере. Обратите внимание этот механизм работает независимо от типа репликации. Все чаще в литературе используется терминология, принятая в СУБД MS SQL SERVER – издатель, дистрибьютор, подписчик

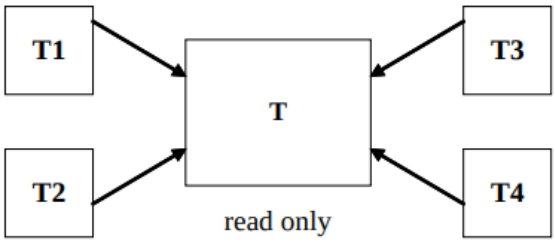
26Репликация с основной копией и её варианты

Есть одна главная копия данных, запросы на запись поступают только туда, а затем реплицируются на все остальные копии. За согласованность изменений отвечает главная копия. Если изменения приводят к несогласованному состоянию, то транзакция откатывается. Запросы на чтение могут быть обработаны любой репликой. Такая реализация эффективна, если количество запросов на запись сильно меньше количества запросов чтения.

Репликация с основной копией

Существуют следующие варианты:

1. **Классический подход** заключается в наличии одной основной копии, в которую можно вносить изменения; остальные копии создаются с определением read only.
2. **Асимметричная репликация**: основная копия фрагментирована и распределена по разным узлам РБД, и другие узлы могут являться подписчиками отдельных фрагментов (read only).
3. **Рабочий поток**. При использовании этого подхода право обновления не принадлежит постоянно одной копии, а переходит от одной копии в другой в соответствии с потоком операций. В каждый момент времени обновляться может только одна копия.
4. **Консолидация данных:**



Один из таких сложных вариантов репликации — это «консолидация», показанная на рис 2.27. Все изменения, происходящие в БД структурных подразделение корпорации, в виде журналов поступают на репликационный сервер, где происходит слияние потоков. Далее все журналы попадают на «Сервер хранилища», где выполнятся (здесь применяется «логическая репликация»). Структуры всех БД структурных подразделений, как и схема

«хранилища» одинаковы, но данные в них разные, при этом **первичные ключи всех таблиц на различных серверах подразделений не пересекаются** – каждая БД имеет свой разрешенный диапазон. После выполнения всех транзакций на «Сервере хранилища» мы получаем единую БД с информацией по всем структурным подразделениям.

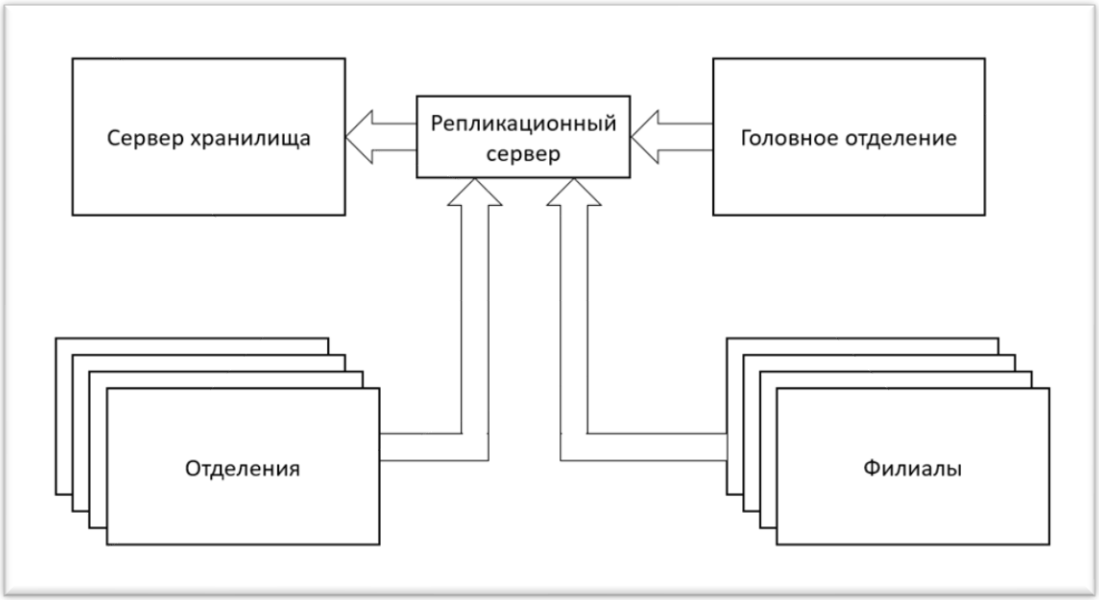


Рис. 2.27. Консолидация БД

Из такого хранилища удобно получать всю отчетность по корпорации в целом и аналитические отчеты для высшего руководства.

27Репликация без основной копии и её варианты?

Репликация без основной копии

Симметричная репликация (без основной копии). Все копии реплицируемого набора могут обновляться одновременно и независимо друг от друга, но все изменения одной копии должны попасть во все остальные копии.

Существует два основных механизма распространения изменений при симметричной репликации:

- **синхронный:** изменения во все копии вносятся в рамках одной транзакции;
- **асинхронный:** подразумевает отложенный характер внесения изменений в удаленные копии.

Достоинство синхронного распространения изменений – полная согласованность копий и отсутствие конфликтов обновления.

Недостатки:

– трудоемкость и большая длительность модификации данных,

– низкая надежность работы системы.

Конфликтные ситуации:

- Добавление двух записей с одинаковыми первичными или уникальными ключами. Для предотвращения таких ситуаций обычно каждому узлу РБД выделяется свой диапазон значений ключевых (уникальных) полей.
- Конфликты удаления: одна транзакция пытается удалить запись, которая в другой копии уже удалена другой транзакцией. Если такая ситуация считается конфликтом, то она разрешаются вручную.
- Конфликты обновления: две транзакции в разных копиях обновили одну и ту же запись, возможно, по-разному, и пытаются распространить свои изменения. Для идентификации конфликтов обновления необходимо передавать с транзакцией дополнительную информацию: старое и новое содержимое записи. Если старая запись не может быть обнаружена, налицо конфликт обновления.

Методы разрешения конфликтов обновления:

1. Разрешение **по приоритету узлов**: для каждого узла назначается приоритет, и к записи применяется обновление, поступившее с узла с максимальным приоритетом.
2. Разрешение **по временной отметке**: все транзакции имеют временную отметку, и к записи применяется обновление с минимальной или максимальной отметкой. Использовать ли для этого минимальную или максимальную отметку – зависит от предметной области и, обычно, может регулироваться.
3. **Аддитивный метод** (add – добавить): может применяться в тех случаях, когда изменения основаны на предыдущем значении поля, например, salary = salary + X. При этом к значению поля последовательно применяются все обновления.
4. Использование **пользовательских процедур**.
5. Разрешение конфликтов **вручную**. Сведения о конфликте записываются в журнал ошибок для последующего анализа и устранения администратором.

Способы реализации распространения изменений:

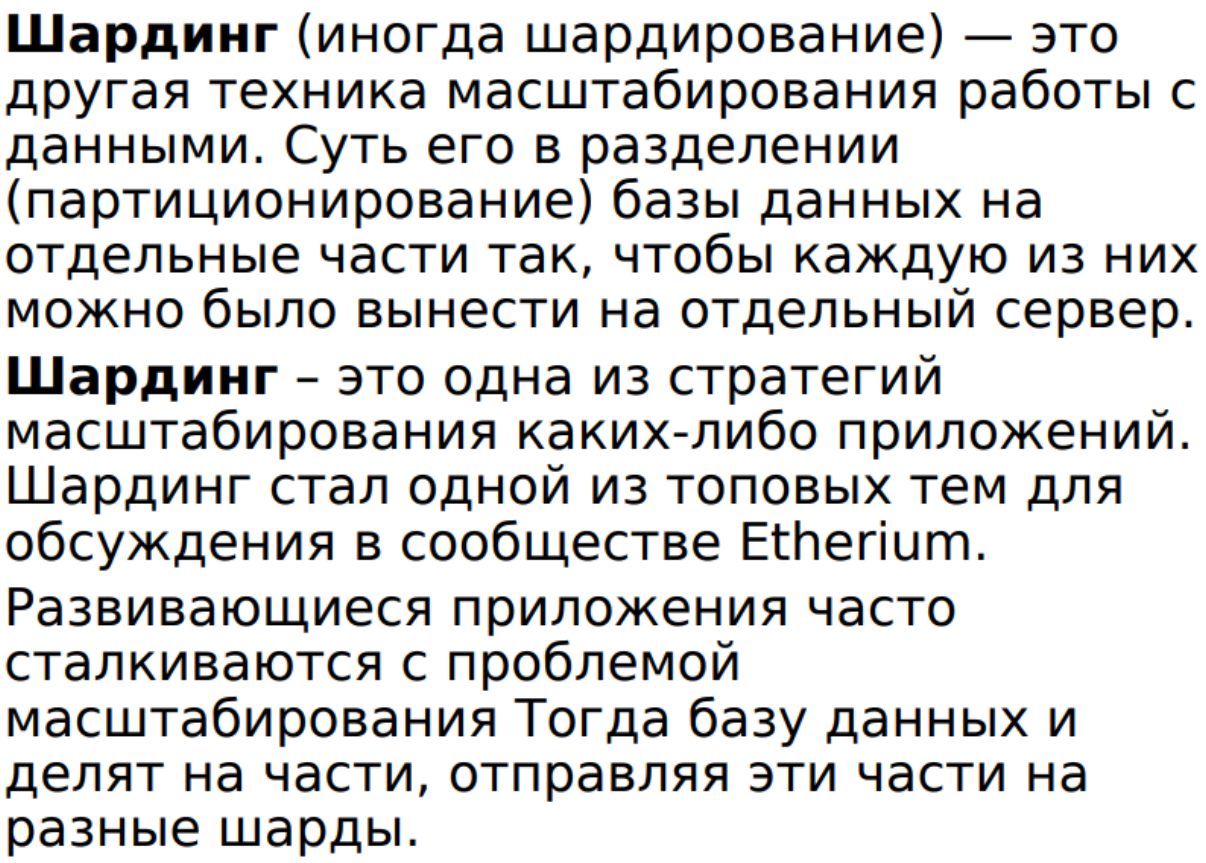
1. Использование триггеров.
   Внутрь триггера помещаются команды, проводящие на других копиях обновления, аналогичные тем, которые вызвали выполнение триггера. Этот подход достаточно гибкий, но он обладает рядом недостатков:

- триггеры создают дополнительную нагрузку на систему;
- триггеры не могут выполняться по графику (время срабатывания триггера не определено);
- с помощью триггеров сложнее организовать групповое обновление связанных таблиц (из-за проблемы мутирующих таблиц).

2. Поддержка журналов изменений для реплицируемых данных. Рассылка этих изменений входит в задачу сервера СУБД или сервера тиражирования (входящего в состав СУБД). Основные принципы, которых необходимо придерживаться при этом:

- Для сохранения согласованности данных должен соблюдаться порядок внесения изменений.
- Информация об изменениях должна сохраняться в журнале до тех пор, пока не будут обновлены все копии этих данных.

28Поддержка распределенных БД. Шардинг?



В лекции ток слайд и какие-то непонятные схемы

Шардинг, или шардирование, представляет собой стратегию масштабирования базы данных, которая заключается в разделении базы данных на отдельные части, называемые шардами или партициями. Каждый шард может быть вынесен на отдельный сервер или узел, что позволяет распределить нагрузку, обеспечить более эффективное масштабирование системы и параллельную обработку данных.

Эээээ по сути просто разбиение БД на непересекающиеся части

Существует несколько основных методов шардирования данных.

Хешированное — данные разбиваются на шарды на основе хеш-функции, которая принимает входные данные и возвращает хеш-значение. Это значение определяет, в какой шард будет помещена каждая запись данных. Метод позволяет достичь высокой производительности и отсутствия единой точки отказа, однако усложняет поиск данных.

Диапазонное — данные разбиваются на шарды на основе диапазона значений. Значения могут присваиваться с помощью ключей (ключевое шардирование) и других атрибутов. Метод прост в реализации и позволяет быстрее находить информацию, чем при хешировании, однако может привести к несбалансированности базы.

Круговое — шарды упорядочиваются в виде кольца и каждый из них ответственен за определенный диапазон значений. Запросы на данные маршрутизируются в соответствии

с позицией шарда в кольце. Запросы распределяются равномерно, но при добавлении и удалении шардов требуется перераспределение данных.

Динамическое — позволяет автоматически масштабировать хранилище в зависимости от текущей производительности и объема данных. Высокая гибкость такого хранилища требует надежную систему мониторинга и балансировки нагрузки, а также хорошо продуманную архитектуру базы данных.

29Протокол двухфазной фиксации

Протокол двухфазной фиксации (2PC) является алгоритмом управления транзакциями в распределенных базах данных. Его основная цель - обеспечение согласованности данных в условиях, когда транзакция взаимодействует с несколькими узлами (серверами) базы данных. Протокол состоит из двух основных фаз и обеспечивает атомарность и долговечность транзакций.

Вот как работает протокол двухфазной фиксации:

Фаза подготовки (Prepare Phase):

- Когда транзакция готова к завершению (все её локальные изменения выполнены), координатор (главный узел) отправляет запрос на подготовку (PREPARE) всем узлам, участвующим в транзакции.
- Каждый узел, получив запрос, фиксирует все свои изменения и готовится к завершению транзакции, но ещё не подтверждает фиксацию.

Фаза фиксации (Commit Phase):

- Если все узлы успешно подготовились, координатор отправляет команду фиксации (COMMIT) всем узлам.
- Каждый узел, получив команду фиксации, окончательно фиксирует изменения и подтверждает успешное завершение транзакции.
- Если хотя бы один узел не может завершить транзакцию (например, из-за сбоя), то координатор отправляет команду отката (ROLLBACK) всем узлам, чтобы отменить изменения.

Протокол двухфазной фиксации обеспечивает атомарность транзакций: если транзакция фиксируется на одном узле, она фиксируется на всех, и наоборот. Он также обеспечивает долговечность транзакций: если транзакция была успешно завершена, её изменения сохраняются даже в случае сбоев.

Далее с его лекций

Действия координатора транзакции

Координатор выполняет протокол 2ФФ по следующему алгоритму:
I. Фаза 1 (голосование).
Занести запись *begin_commit* в системный журнал и обеспечить ее перенос из буфера в ОП на ВЗУ. Отправить всем участникам команду PREPARE.
Ожидать ответов всех участников в пределах установленного тайм-аута.
II. Фаза 2 (принятие решения).
При поступлении сообщения ABORT: занести в системный журнал запись *abort* и обеспечить ее перенос из буфера в ОП на ВЗУ; отправить всем участникам сообщение GLOBAL_ABORT и ждать ответов участников (тайм-аут).
Если участник не отвечает в течение установленного тайм-аута, координатор считает, что данный участник откатит свою часть транзакции и запускает протокол ликвидации.
Если все участники прислали COMMIT, поместить в системный журнал запись *commit* и обеспечить ее перенос из буфера в ОП на ВЗУ. Отправить всем участникам сообщение GLOBAL_COMMIT и ждать ответов всех участников.
После поступления подтверждений о фиксации от всех участников: поместить в системный журнал запись *end_transaction* и обеспечить ее перенос из буфера в ОП на ВЗУ.
Если некоторые узлы не прислали подтверждения фиксации, координатор заново направляет им сообщения о принятом решении и поступает по этой схеме до получения всех требуемых подтверждений.

Протоколы ликвидации

Протокол ликвидации для координатора:

1. Тайм-аут в состоянии WAITING: координатор не может зафиксировать транзакцию, потому что не получены все подтверждения от участников о фиксации. Ликвидация заключается в откате транзакции.
2. Тайм-аут в состоянии DECIDED: координатор повторно рассылает сведения и принятом глобальном решении и ждет ответов от участников. Простейший протокол ликвидации для участника заключается в блокировании процесса до тех пор, пока сеанс связи с координатором не будет восстановлен. Но в целях повышения производительности (и автономности) узлов могут быть предприняты и другие действия:
    - Тайм-аут в состоянии INITIAL: участник не может сообщить о своем решении координатору и не может зафиксировать транзакцию. Но может откатить свою часть транзакции. Если он позднее получит команду PREPARE, он может проигнорировать ее или отправить координатору сообщение ABORT.
    - Тайм-аут в состоянии PREPARED: участник уже известил координатор о решении COMMIT, то он не может его изменить. Участник оказывается заблокированным.

Протоколы восстановления

Действия, которые выполняются на отказавшем узле после его перезагрузки, называются *протоколом восстановления*.

Они зависят от того, в каком состоянии находился узел, когда произошел сбой, и какую роль выполнял этот узел в момент отказа: координатора или участника.

При отказе координатора:

- ✓ В состоянии INITIAL: процедура 2ФФ еще не запускалась, поэтому после перезагрузки следует ее запустить.
- ✓ В состоянии WAITING: координатор уже направил команду PREPARE, но еще не получил всех ответов и не получил ни одного сообщения ABORT. В этом случае он перезапускает процедуру 2ФФ.
- ✓ В состоянии DECIDED: координатор уже направил участникам глобальное решение. Если после перезапуска он получит все подтверждения, то транзакция считается успешно зафиксированной. В противном случае он должен прибегнуть к протоколу ликвидации.

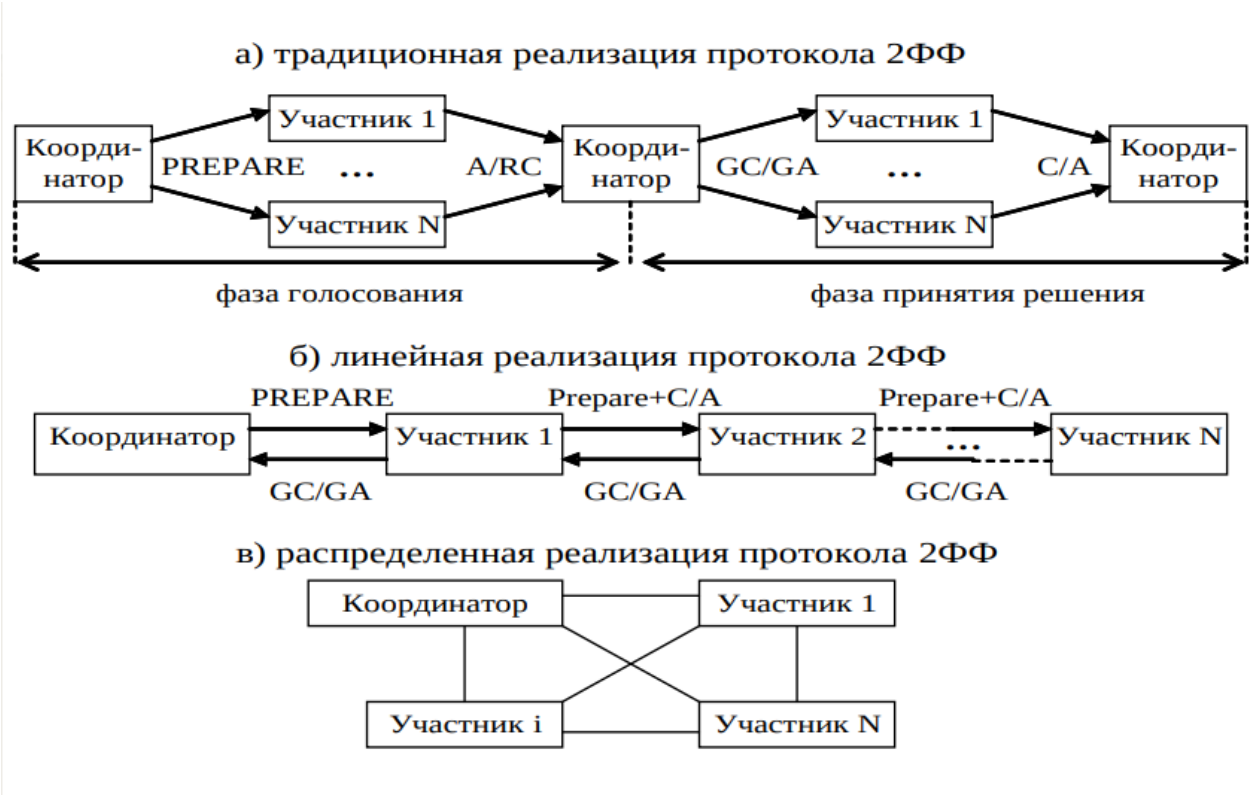
Протоколы восстановления

При отказе участника цель протокола восстановления – гарантировать, что после восстановления узел выполнит в отношении транзакции то же действие, которое выполнили другие участники, и сделает это независимо от координатора, т.е. по возможности без дополнительных подтверждений.

Рассмотрим три возможных момента возникновения отказа:

- ✓ В состоянии INITIAL: участник еще не успел сообщит о своем решении координатору, поэтому он может выполнить откат, т.к. координатор не мог принять решение о глобальной фиксации транзакции без голоса этого участника.
- ✓ В состоянии PREPARED: участник уже направил сведения о своем решении координатору, поэтому он должен запустить свой протокол ликвидации.
- ✓ В состоянии ABORTED/COMMITED: участник уже завершил обработку своей части транзакции, поэтому никаких дополнительных действий не требуется.

30Варианты реализации протокола 2ФФ



Традиционная схема реализации протокола 2ФФ: подразумевает передачу 4N сообщений (N- кол-во участников)

Линейная схема реализации протокола 2ФФ: согласно этой схеме координатор отправляет команду PREPARE вместе со списком участников только одному участнику. Участник 1 передает след участнику полученный пакет + COMMIT/ABORT. Последний участник отправляет обратное сообщение об общем решении. Кол-во передаваемых сообщений 2N. Снижается параллельность выполнения транзакции. Данная схема применима для сети с небольшим количеством узлов. Для повышения эффективности необходимо, чтобы последний участник посылал широковещательное сообщение о глобальном решении.

Распределенная схема реализации протокола 2ФФ: все узлы, получив от координатора команду PREPARE, рассылают всем участникам сведения о своем решении. Как только узел получает все подтверждения готовности к фиксации, он фиксирует свою часть транзакции. Если он получает хотя бы одно сообщений ABORT, то он откатывает транзакцию, фаза принятия решения при этом отсутствует.

31Технология Oracle Streams?

Технология Oracle Streams

Oracle Streams – универсальный гибкий механизм обмена информацией между серверами в много серверной архитектуре (MTS). Позволяет одновременно реализовать репликацию, обмен сообщениями, загрузку хранилищ данных, работу с событиями, поддержку резервной БД. Данные следуют по определенным пользователем маршрутам и доставляются к месту назначения. В результате получается механизм, который обеспечивает большую функциональность и гибкость, чем традиционные решения для хранения и распространения данных, а также совместного их использования с другими базами данных и приложениями.
Oracle Streams –отдельная информационная инфраструктура, которая состоит из процессов
capture, propagation и apply.

Термины Oracle Streams

LCR, CR
В контексте Oracle Streams информационное представление любого изменения, сделанного в базе данных, называется ***LCR*** (logical change record). ***CR*** (change record) –запись изменения, используется для того, чтобы обозначить конкретное изменение в базе.

Capture, Propagation and Apply –три основных процесса Oracle Streams.
Основные задачи процесса ***capture***:

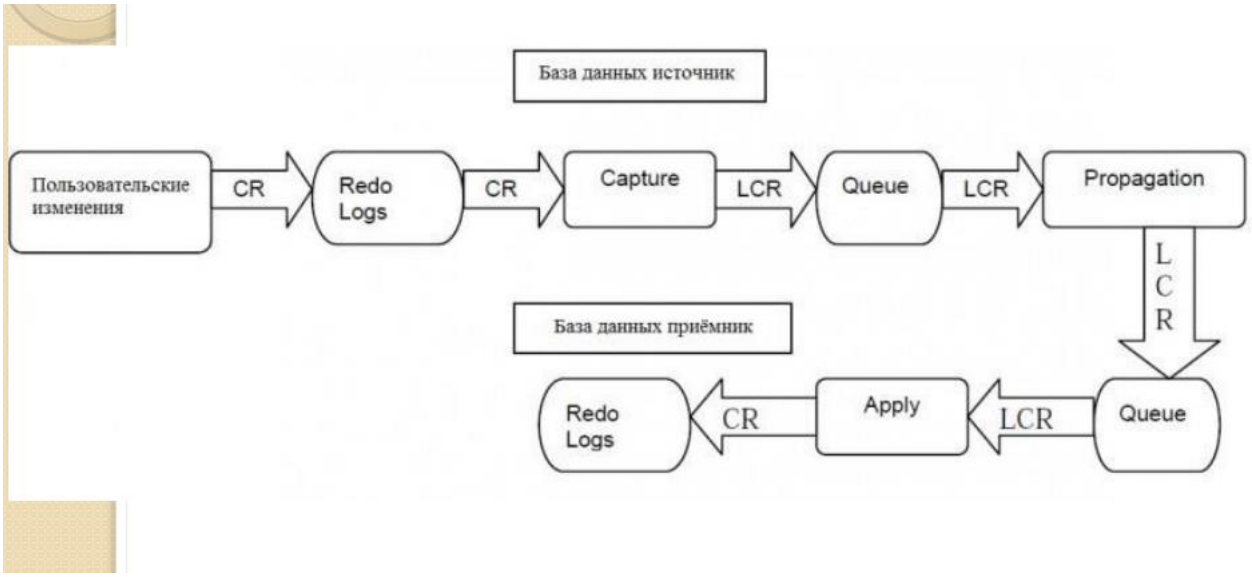
- считывание изменений, содержащихся в журналах транзакций;
- преобразование CR в LCR;
- постановка LCR в очередь.

Так же как и *capture*, процесс ***propagation*** выполняет 3 основных задачи:

- просмотр LCR;
- передача LCR из одной очереди в другую, причем очереди могут находится как на одной базе данных, так и на разных;
- удаление LCR.

Apply процесс:

- извлекает принятые LCR из очереди;
- производит изменения с базой данных в соответствии с LCR;
- удаляет LCR из очереди.



32Стихийные угрозы информационной безопасности!!!

Стихийные угрозы информационной безопасности

- Отказ оборудования
- Стихийное бедствие
- Техногенная катастрофа
- Неумышленные действия персонала

33Злоумышленные угрозы информационной безопасности!!!

Злоумышленные угрозы информационной безопасности

- Несанкционированное считывание информации
- Неразрешенная модификация информации и ее уничтожение
- Изменение штатного функциони- рования вычислительной системы (сети)

Злоумышленные угрозы информационной безопасности (Предполагается наличие злоумышленника):

1-Несанкционированное считывание информации

- считывание паролей и отождествление их с конкретными пользователями;
- получение конфиденциальной информации;
- идентификация информации, запрашиваемой пользователями;
- подмена паролей с целью доступа к информации;
- контроль активности абонентов сети для получения косвенной информации о взаимодействии пользователей и характере информации, которой обмениваются абоненты сети.

2-Неразрешенная модификация информации и ее уничтожение

- разрушение данных и кодов исполняемых программ путем внесения тонких, трудно обнаруживаемых изменений в информационные массивы;
- внедрение программных закладок в другие программы и подпрограммы (вирусный механизм воздействий);
- искажение или уничтожение собственной информации сервера, и тем самым нарушение работы сети;
- модификация пакетов сообщений.

3-Изменение функционирования вычислительной системы (сети)

- уменьшение скорости работы вычислительной системы (сети); частичное или полное блокирование работы системы (сети);
- имитация физических (аппаратных) сбоев работы вычислительных средств и периферийных устройств;
- переадресация сообщений;
- обход программно-аппаратных средств криптографического преобразования информации;
- обеспечение доступа в систему с непредусмотренных периферийных устройств.

34Простые действия для обеспечения безопасности IT-систем!!

Lec.9 sli.12

-Настройка поддерживаемых систем
-Обновление поддерживаемых систем
-Обучение персонала (пользователей)
-Распределение ролей и ограничение доступа к данным
-Создание программ проверки целостности
-Создание всеохватывающей системы аудита