03.09.2020

МДК 11.01 Технология разработки и защиты баз данных

ПМ.11 Разработка, администрирование и защиты баз данных

Итоговая оценка будет!

Литература: Г. Н. Федорова – Основы проектирования баз данных (2017);

Г. Н. Федорова – Разработка, администрирование и защита баз данных (2017)

sql-ex.ru - сайт для отработки всяких штучек

mysql.ru – сайт документации

**Введение в базы данных**

Система баз данных – это компьютеризированная система, основная задача которой – хранение информации и предоставление доступа к ней по требованию.

Система баз данных включает в себя:

* Данные, непосредственно сохраняемые в базе данных;
* Аппаратное обеспечение;
* Программное обеспечение;
* Пользователей:
* Прикладные программисты;
* Конечные пользователи;
* Администраторы баз данных;

СУБД – это специализированное программное обеспечение, предоставляющее пользователю базы данных возможность работать с ней, не вникая в детали хранения информации на уровне программного обеспечения.

Существует 3 уровня архитектуры СУБД:

* Внутренний уровень – наиболее близкий к физическому хранению
* Внешний уровень – наиболее близкий к пользователям
* Концептуальный уровень – является промежуточным между двумя первыми

Функции СУБД: определение данных; обработка данных; безопасность и целостность данных; восстановление данных и дублирование; словарь данных; производительность. СУБД должная выполнять своим функции с максимальной производительностью.

**Классификация баз данных**

Разделяются:

* По модели представления данных
* По организации хранения данных и обращения к ним
* По типу хранимой информации

Классификация БД по модели представления данных:

* Иерархическая БД – база данных, в которой связь между элементами осуществляется по типу подчинения и схематично изображается в виде дерева. Иерархия начинается с корневого узла. Каждый узел имеет только одного «предка» и N «потомков»

(+) Простота и однозначность представления, легкость адресации

(-) существенная зависимость от программно-аппаратных средств

Пример: дерево папок Windows, каталог товаров Интернет.

* Сетевая БД – возможно существование любых связей между объектами. Если изобразить эту модель графически, что получится набор узлов на плоскости, связанных линиями со стрелками

(+) теоретически возможны сколько угодно сложные связи между объектами

(-) сложность реализации, существенная зависимость от программно-аппаратных средств

Пример: служба WWW – документы, произвольно связанные ссылками.

* Реляционная БД – представление данных в виде системы взаимосвязанных таблиц. Каждый объект системы описывается в виде таблицы с набором свойств, а взаимосвязь между объектами – связями между таблицами

(+) простота; относительная независимость от программных и аппаратных средств

(-) существенная зависимость скорости обработки от объема БД

Использование: все существующие СУБД

Классификация БД по организации хранения данных и обращения к ним:

* Локальные – БД и СУБД находятся на одном компьютере и не может работать по сети;
* Сетевые – БД и СУБД находится на сервере сети, а приложение на компьютере пользователя посылает запрос и получает результат
* Распределенные базы данных – БД находятся на множестве компьютеров в сети, а СУБД осуществляет контроль над ними, а приложение на компьютере пользователя посылает запрос к СУБД и получает результат.

Классификация БД по типу хранимой информации:

Документальные, фактографические, лексикографические

В базе данных не должно быть избыточной информации

*Избыточность информации* – превышение количества информации, используемой для передачи или хранения сообщения, над его информационной энтропией.

*Логическая независимость данных* – общая логическая структура данных может быть изменена без изменения прикладных программ.

*Физическая независимость данных* – физическое расположение и организация данных могут изменяться, на вызывая при этом изменений ни общей логической структуры данных, ни прикладных программ.

Компоненты СУБД

*Система управления ими данных* – это комплекс языковых и программных средств, предназначенный для создания, ведения и совместного использования БД многими пользователями

1. Место хранения данных и метаданные
2. Менеджер памяти
3. Менеджер транзакций

**Реляционная модель данных**

*Отношение это* – фундаментальное понятие реляционной модели

Каждая таблица содержит информацию об объектах одного типа, а совокупность всех таблиц образует единую базу данных. Строка таблицы называется записью, столбец таблицы – поле.

*Тип данных* реляционной модели полностью совпадает с понятием типа данных в программировании.

*Домен* определяется путем задания некоторого базового типа данных и описания их ограничения.

*Атрибут* – это свойство объекта предметной области, информация о котором хранится в базе данных. Атрибут характеризуется именем и значением, которое должно принадлежать некоторому типу.

*Схема отношения* – это именованное множество пар {имя атрибута, имя домена (или типа, если понятие домена не поддерживается}

*Картеж*, соответствующий данной схеме отношения, - это множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения.

*Эквиваленты*, используемые в теории БД:

Отношение – Таблица, Картеж – строка, Атрибут – столбец, поле.

**Первичный ключ – *минимальный* набор атрибутов, значения которых *однозначно* определяют картеж отношений**

Ограничения:

1. Значения в ячейках таблицы должны быть одиночными – ни повторяющиеся группы, ни массивы не допускаются
2. Строки имеют фиксированное число полей и значений. Иначе говоря, в каждой позиции таблицы на пересечении строки и столбца всегда имеется в точности одно значение или ничего
3. Строки таблицы обязательно отличаются друг от друга хотя бы единственным значением, что позволяет однозначно идентифицировать любую строку такой таблицы.
4. Все записи в столбце должны быть одного типа. Например, если третий столбец первой строки таблицы содержит номер сотрудника, что и во всех остальных строках таблицы третий столбец также должен содержать номер сотрудника. Каждый столбец имеет уникальное имя; порядок столбцов в таблице несущественен. Наконец, в отношении не может быть двух одинаковых строк, и порядок строк не имеет значения.
5. Полное информационное содержание базы данных представляется в виде явных значений данных и такой метод представления является единственным
6. При выполнении операции с таблицей ее строки и столбцы можно обрабатывать в любом порядке безотносительно к их информативному содержанию

08.09.2020

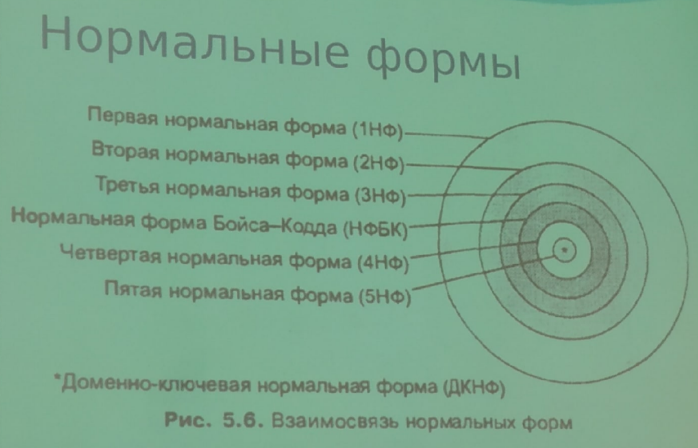
**Базы и банки данных**

**Нормализация схем базы данных**

Нормализация:

1. Борется с избыточностью
2. Потенциальная противоречивость (аномалии обновления)
3. Аномалии включения
4. Аномалии удаления

**Нормальные формы**



Основные свойства нормальных форм:

* Каждая следующая нормальная форма лучше предыдущей
* При переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются

В основе процесса проектирования лежит метод нормализации, декомпозиция отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиями следующей нормальной формы

*Нормализация* – процесс преобразования отношения, имеющего некоторые недостатки, в отношение, которое этих недостатков не имеет.

Первая нормальная форма (1НФ):

* Ячейки таблицы должны содержать одиночные значения и в качестве значений не допускаются ни повторяющиеся группы, ни массивы
* Все записи в одном столбце должны иметь и тот же тип
* Каждый столбец должен иметь уникальное имя, но порядок следования столбцов в таблице несущественен
* В таблице не может быть более двух одинаковых строк, и порядок следования строк несущественен

Вторая нормальная форма (2НФ):

Функциональная зависимость – это связь между атрибутами, атрибут y функционально зависит от атрибута x, если значения x однозначно определяет значение y.

Отношения находятся во *второй нормальной форме* тогда и только тогда, когда оно находится в первой нормальной форме и все его не ключевые атрибуты функционально полно зависят от первичного ключа, то есть зависят целиком от первичного ключа и не зависят от его частей

Задачка 1.

{N\_сотр, ФИО\_сотр, тел\_сотр

код\_изд, назв\_изд, дата\_изг, кол-во\_изд}

N\_сотр - ФИО\_сотр, тел\_сотр

код\_изд - назв\_изд

N\_сотр, код\_изд, дата\_изг - кол-во\_изд

PK: N\_сотр; ФИО\_сотр; Тел\_сотр

PK: Код\_изд; назв\_изд

(ВСТАВИТЬ ПОТОМ КАРТИНКУ)

Замечания: если в отношении первичный ключ простой, то отношение уже находится в второй нормальной форме

Алгоритм приведения отношения ко второй нормальной форме:

Шаг 1: определить первичный ключ. Если первичный ключ простой, то отношение уже находится во второй нормальной форме, иначе перейти к шагу 2.

Шаг 2: выписать функциональные зависимости от частей первичного ключа

Шаг 3: каждая функциональная зависимость образует новое отношение

Шаг 4: в каждом новом отношении определить первичный ключ

Шаг 5: установить связи один ко многим от первичного ключа на част первичного ключа

Третья нормальная форма (3НФ):

Отношение находится в 3НФ тогда и только тогда, когда оно находится во второй нормальной форме и в нем отсутствуют транзитивные зависимости, то есть зависимости между не ключевыми атрибутами.

Алгоритм приведения отношения к третьей нормальной форме:

Шаг 1: привести отношение ко второй нормальной форме

Шаг 2: проверить, нет ли в отношении транзитивных зависимостей. Если транзитивных зависимостей нет, то отношение уже находится в третье нормальной форме. Если транзитивные зависимости есть, то переходим к шагу 3

Шаг 3: Вынести транзитивные зависимости в отдельные отношения. В каждой транзитивной зависимости выделить первичный ключ

Шаг 4: Продублировать первичный ключ нового отношения в то отношение, откуда оно было вынесено в качестве внешнего ключа (FK)

Шаг 5: установить связи один ко многим от первичного ключа к внешнему ключу

16.09.2020

**Модель сущность-связь**

Требования, предъявляемые к инфологической модели:

Основным требованием к информационно-логической модели является требование адекватного отображения предметной области

**Построение модели** “объект – свойство – отношение”. Классом объекта называют совокупность, обладающих одинаковым набором свойств. Каждый объект представляется своим идентификатором. Каждый объект обладает определенным набором свойств.

Сущность – это реальный или представляемый объект, информация о котором должная сохраняется и быть доступна. В диаграммах ER-модели сущность представляется в виде прямоугольника.

16.10.2020

**Транзакции**

*Транзакция* – неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации) такая, что:

* либо результаты всех операторов, входящих в транзакцию, отображаются в БД;
* либо воздействие всех этих операторов полностью отсутствует.

**Выполнение и откат транзакции**

**Исходное**

**состояние**

**Исходное**

**состояние**

**Измененная БД**

**Нарушение целостности**

COMMIT ROLLBACK

**Модели транзакций**

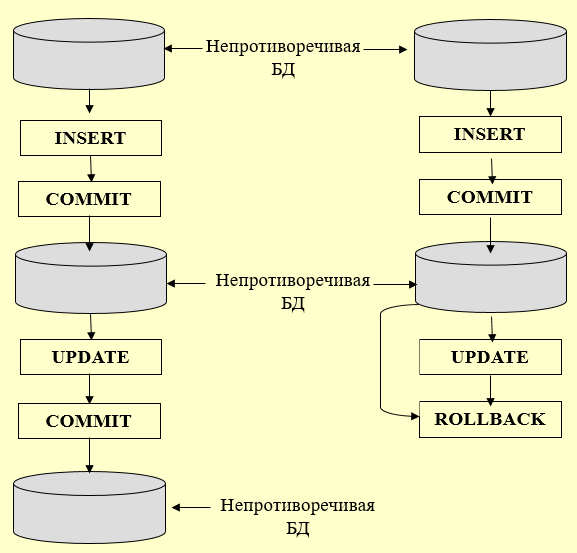
Стандарт ANSI/ISO (СУБД DB2) - автоматическое выполнение транзакций

* COMMIT
* ROLLBACK

Модель транзакций СУБД Sybase (Transact-SQL) - управляемое выполнение транзакций

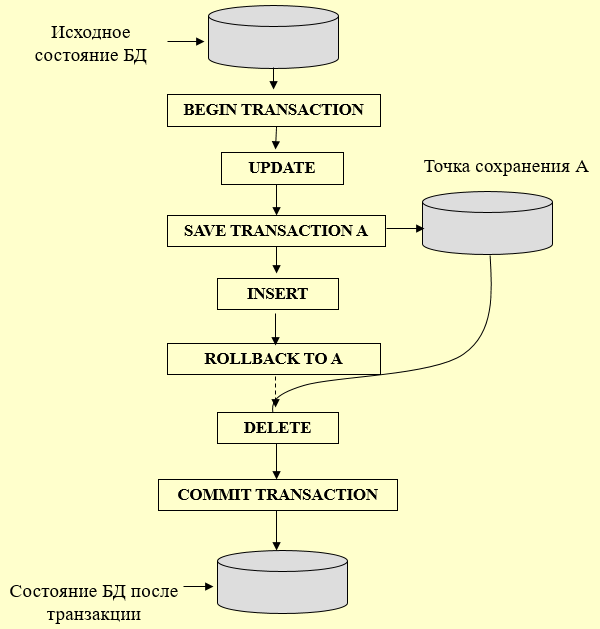
* BEGIN TRANSACTION
* COMMIT TRANSACTION
* SAVE TRANSACTION
* ROLLBACK

**Модель автоматического выполнения транзакций**

Транзакция автоматически начинается с выполнения пользователем или программой первой инструкции SQL. Завершается последовательное выполнение инструкций одним из двух способов:

* инструкцией COMMIT (изменения, внесенные в БД, становятся постоянными, а новая транзакция начинается сразу после инструкции COMMIT);
* инструкцией ROLLBACK (отменяет выполнение текущей транзакции и возвращает БД к состоянию начала транзакции, новая транзакция начинается сразу после инструкции ROLLBACK).

**Модель управляемого выполнения транзакции**

* BEGIN TRANSACTION сообщает о начале транзакции;
* COMMIT TRANSACTION сообщает об успешном выполнении транзакции (новая транзакция не начинается автоматически);
* SAVE TRANSACTION позволяет создать внутри транзакции *точку сохранения*;
* ROLLBACK отменяет выполнение текущей транзакции и возвращает БД к состоянию SAVE TRANSACTION или к состоянию начала транзакции.

**Проблемы при одновременном обращении двух транзакций к одним данным**

* Потерянное обновления
* Грязное чтение
* Неповторяемое чтение
* Чтение

**Потерянные обновления** появляются при выборе одной записи двумя или более транзакциями, которые затем обновляют эту запись на основе ее первоначального значения

**Грязное чтение** возникает в том случае, когда первая транзакция читает данные которые обновила, но еще не зафиксировала вторая транзакция

**Неповторяемое чтение** возникает, когда первая транзакция несколько раз обращается к одним и тем же данным, однако данные меняются вследствие того, что между обращениями вторая транзакция обновляет данные и фиксирует.

**Чтение фантомов** появляется в том случае, когда записи из диапазона записей, читаемого в первой транзакции, добавляются или удаляются второй транзакцией

**Уровень изоляции транзакции** - какие блокировки накладываются на данные, обрабатываемые в рамках транзакций

Уровни изоляции транзакции

1. READ UNCOMMITTED (неподтвержденное чтение);
2. READ COMMITTED (подтвержденное чтение);
3. REPEATABLE READ (повторяемое чтение);
4. SERIALIZABLE (упорядочение).

**Импорт CSV-файла в таблицу MySQL**

Оператор LOAD DATA INFILE позволяет считывать данные из текстового файла.

Для этого необходимо:

* Создать таблицу данных, в которую будут импортированы данные из файла.
* Создать CSV файл с соответствующим числом столбцом и с соответствующим форматом данных в каждом столбце.
* Создать учетную запись пользователя, который подключается к серверу базы данных MySQL и имеет привилегии FILE и INSERT. (можно делать с root)

Функция str\_to\_date преобразует дату к нужному формату. Select str\_to\_date(‘may/09/2003’,’%M,%d,%Y’) (если месяц слово – то, M-большая, если цифра – маленькая)

**Экспорт результатов запросов в файл**

mysql> SELECT \* FROM passwd INTO OUTFILE '/tmp/passwd.txt'  
-> FIELDS TERMINATED BY ',' ENCLOSED BY '"'  
-> LINES TERMINATED BY '\r\n';

**Администрирование СУБД PostgreSQL**

СУБД PostgreSQL является свободной объектно-реляционной.

**Архитектура PostgreSQL**

В основе PostgreSQL серверный процесс базы данных, выполняемый на одном сервере.

Преимущества и функциональные возможности PostgreSQL:

* Надежность – проверена и доказана. Она соответствует принципам OSI. Реализована репликация. Поддержка целостности данных на уровне схемы
* Производительность – основана на применении индексов, наличии гибкой системы блокировок и интеллектуального планировщика запросов.
* Расширяемость – создание пользовательских функций, типов и так далее.
* Поддержка SQL
* Поддержка многочисленных типов данных (численные, булевые, символьные, дата, время, массивы, JSON, можно создавать свои типы данных)

**Ограничение БД:**

* Размер БД – нет ограничения
* Размер таблицы – до 32TB
* Размер строки – до 1,6TB
* Размер поля – до 1GB
* Количество строк в таблице – нет ограничения
* Количество столбцов в таблице – до 250…1600
* Количество индексов – нет ограничения
* Длина идентификатора – до 63 байт включительно

**Объекты в БД**

* Таблицы
* Индексы
* Пользователи и группы (роли)
* Языки (для создания функций)
* Функции (FUNCTION)
* Триггеры (TRIGGER)
* Правила (RULE)
* Представления (VIEW)
* Правила преобразования типов (CAST)
* Типы данных (TYPE)
* Последовательности (SEQUENCE)

**Политика защиты строк**

**Политика ограничивает видимость строк в таблице**

видимость определяется предикатами, которые вычисляются для каждой строки с правами вызывающего клиенту доступны только те строки, для которых предикаты истинны

**Предикат для существующих строк**

используется операторами SELECT, UPDATE, DELETE при нарушении политики не возникает ошибка (если только не сброшен параметр row\_security)

**Предикат для новых строк**

используется операторами INSERT, UPDATE если не задан, используется первый предикат при нарушении политики возникает ошибка

**Политика применяется**

* к таблице, для которой включена защита
* для указанных ролей и для указанных операторов (SELECT, INSERT, UPDATE, DELETE)

**Политика не применяется**

* при проверке ограничений целостности
* для суперпользователей и ролей с атрибутом BYPASSRLS
* для владельца (если не включить принудительно)

**Разрешительные политики** видимость должна предоставить хотя бы одна разрешительная политика если не определена ни одна политика, строка не видима

**Ограничительные политики** видимость должны предоставить все ограничительные политики, если они определены

**Алгоритм создания политик**

1) Создать политику.

Create policy (название политики) on (имя таблицы, к которой привязан) Using (запрос)

Alter table (имя таблицы, к которой привязана) enable row level security

**Повышение производительности**

Тут была инфа но ее надо найти

При поиске по индексу, даже если поиск осуществляется по индексированному полю, требуется время, чтобы найти в индексе первую строку в соответствии с требуемым порядком.

set enable\_indexscan=off; для отключения индексации

explain select \* from bookings order by book\_ref desc;

**Репликация**

**Репликация** – это процесс синхронизации нескольких копий кластера без данных на разных серверах.

Базовый механизм для решения задач

* Отказоустойчивость
* Масштабируемость

**Мастер реплика**

Записи WAL передаются на реплику и применяются

* Поток данных только в одну сторону
* Реплицируется кластер целиком, выборочная реплика невозможна

Реплика – точная копия мастера

* Одна и та же основная версия сервера
* Полностью совместимые архитектуры и платформы

Реплика доступа только для чтения

Мастер передает на реплику журнальные записи, а реплика применяет эти записи к своим файлам данных.

Сервера должны быть на одной и той же аппаратно-программной платформе. И на них должны быть запущены одинаковые основные версии PostgreSQL.

Реплицировать можно только кластер целиком, а не отдельные базы данных или таблицы. Возможность «отфильтровать» журнальные записи отсутствует.

Реплика не может генерировать собственных журнальных записей, она лишь применяет записи мастера. Поэтому при таком подходе реплика может быть доступна только для чтения – никакие операции, изменяющие данные, на реплике не допустимы.

**Использования реплики**

Допускаются:

* Запросы на чтение данных (select, copy to, курсорсы)
* Установка параметров сервера (set, reset)
* Управление транзакциями (begin, commit, rollback…)
* Создание резервной копии (pg\_basebackup)

Не допускаются:

* Любые изменения (insert, update, delete, truncate, nextval…)
* Блокировки, предполагающие изменение (select for update…)
* Команды DDL (create, drop…), в том числе создание временных таблиц
* Команды сопровождения (vacuum, analyze, reindex…)
* Управление доступом (grant, revoke…)

**Триггеры и блокировки**

* Триггеры и пользовательские блокировки не работают на реплике

**Резервное копирование с реплики**

* Параметр full\_page\_writes должен быть заранее включен на мастере
* Задержки из-за ожидания контрольной точки и переключения сегментов.

**Режимы репликации**

* Асинхронный
* Синхронный - Фиксация с ожиданием подтверждения от синхронной реплики обеспечивает надежность, но не согласованность

**Итоги**

Механизм репликации основан на передаче журнальных записей на реплику и их применении

* Основной режим: потоковая репликация. Может быть поддержана архивом журнальных записей

Реплика – точна копия мастера

* Не генерирует собственные WAL, доступна только для чтения

Сложный механизм, требующий настройки и мониторинга

* Настройка существенно зависит от решаемых задач

**Переключение на реплику**

Причины

* Плановое переключение. Остановка основного сервера для проведения технических работ
* Аварийное переключение. Переход на реплику из-за сбоя основного сервера

Процедура

* Убедиться, что мастер остановлен
* Переключение вручную: «продвижение» или триггерный файл
* Автоматическое переключение отсутствует

**Логическая репликация**

* Публикация-подписчики: в сервера нет выделенной роли
* Трансляция изменений табличных строк
* Необходим уровень журнала logical
* Требуется совместимость на уровне протокола
* Возможны выборочная репликация отдельных таблиц

**Модель**

Публикация

* Объект базы данных
* Выдает измерения данных построчно
* Изменения в порядке фиксации транзакций

Подписчики

* Подписываются, получают и применяют изменения
* Таблицы и столбцы сопоставляются по полным именам
* Поддерживается «бесшовная» синхронизация данных при создании подписки
* Могут возникать конфликты с локальными данными

Логическая репликация использует модель «публикация-подписчики».

На одном сервере создается *публикация*, которая может включать ряд таблиц одной базы данных. Для репликации из нескольких баз данных потребуется создать несколько репликаций.

Публикация включает в себя изменения, происходящие с таблицами: эти изменения, передаются на уровне строк («в таблице такой-то такая-то строка изменилась таким-то образом»).

Другие серверы могут создавать подписку на публикацию, получать и применять изменения.

Применение изменений всегда происходит построчно. Хотя каждое отдельное изменение не требует накладных расходов на разбор и планирование запроса, массивные изменения из-за этого будут выполняться медленнее.

Таблицы идентифицируются по полным именам (включая схему), столбцы также идентифицируются по именам. Это позволяет подписчику использовать отличающуюся схему данных (например, иметь в таблице дополнительные столбцы).

**Ограничения**

Реплицируются не все изменения

* Только команды INSERT, UPDATE, DELETE
* Только базовые таблицы
* (не реплицируются последовательности, материализованные представления, секционированные таблицы)

Подписку можно создать только на основном сервере

* Не работает на физических репликах

Циклы в репликации не обрабатываются

* Нельзя реплицировать одну и ту же таблицу
* С одного сервера на другой и обратно.

Также не реплицируются другие объекты, объединяемые термином relation – последовательности, материализованные представления, секционирование таблиц.

**Итоги**

* Логическая репликация: модель «публикация-подписчики»
* Передаются изменения табличных строк
* Возможна выборочная репликация отдельных таблицы
* Не требуется двоичная совместимость серверов
* Пока доступны только базовые возможности, но функционал активно развивается

**Блокировки**

Задача: упорядочение конкурентного доступа к разделяемым ресурсам

Механизм

* Перед обращением к данным процесс захватывает блокировку, после обращения – освобождает
* Блокировки приводят к очереди

Альтернативы

* Много версионность – несколько версий данных
* Оптимистические блокировки – процессы не блокируются, но при неудачном стечении обстоятельств возникает ошибка.

Блокировки используются, чтобы упорядочить конкурентный доступ к разделяемым ресурсам

Под конкурентным доступом понимается одновременные доступ нескольких процессов. Сами процессы могут выполняться как параллельно (если позволяет аппаратура), так и последовательно в режиме разделения времени.

**Ресурсы**

Ресурс

* Все, что можно идентифицировать

Примеры ресурсов

* Реальные хранимые объекты: страница, таблица, строки и т.п., структуры данных в общей памяти (хеш-таблицы, буферы…), абстрактные ресурсы (число).

**Время жизни**

Долговременные блокировки

* Обычно захватываются до конца транзакции и относятся к хранимым данным
* Большое число режимов
* Развитая «тяжеловесная» инфраструктура, мониторинг

Краткосрочные блокировки

* Обычно захватываются на долг секунды и относятся к структурам в оперативной памяти
* Минимум режимов
* «легковесная» инфраструктура, мониторинг может отсутствовать.

В PostgreSQL информация о том, что строка заблокирована, хранится только и исключительно в версии строки внутри страницы данных. Там она представлена номером блокирующей транзакции и дополнительными информационными битами.

За счет этого блокировок уровня строки может быть неограниченное количество. Это не приводит к потреблению каких-либо ресурсов и не снижает производительность системы.

Всего существует 4 режима, в которых можно заблокировать строку.

Два режима представляют исключительные блокировки, которые одновременно может удерживать только одна транзакция. Режим UPDATE предполагает полное изменение (или удаление) строки, а режим NO KEY UPDATE – изменение только тех полей, которые не входят в уникальные индексы

Команда UPDATE сама выбирает минимальный подходящий режим блокировки; обычно строк блокируются в режиме NO KEY UODATE.

**Разделяемый режим**

Еще два режима представляют разделяемые блокировки, которые могут удерживаться несколькими транзакциям.

Режим SHARE применяется, когда нужно прочитать строку, но при это нельзя допустить, чтобы она как-либо изменилась другой транзакцией.

Режим KEY SHARE допускает изменение строки, но только не ключевых полей. Этот режим, в частности, автоматически используется PosgreSQL при проверке внешних ключей.

Замечания:

* Исключительные режимы конфликтует между собой
* Разделяемые режимы совместимы между собой
* Разделяемый режим KEY SHARE совместим с исключительным режимом NO KEY UPDATE

Возможны ситуации взаимоблокировки, когда одна транзакция пытается захватить ресурс, уже захваченный другой транзакцией, в тое время, как другая транзакция пытается захватить ресурс, захваченный первой.

**Итоги**

Блокировки строк хранятся в страницах данных

* Из-за потенциально большого количества

Очереди и обнаружение взаимоблокировок обеспечиваются блокировками объектов

* Приходится прибегать к сложным схемам блокирования.

**Блокировка в памяти**

**Спин-блокировки**

Устанавливаются на очень короткое время

* Несколько инструкций процессора

Единственный режим – исключительный

Нет возможности мониторинга

Нет обнаружения взаимоблокировок

Цикл активного ожидания

* Используются атомарные инструкции процессор

Спин-блокировки или спинлоки предназначены для захвата на очень короткое время и защищают отдельные поля от одновременного изменения.

Если блокировка занята, выполняется цикл активного ожидания – команда повторяется до тех пор, пока не выполнится успешно. Это имеет смысл, поскольку спин-блокировки применяются только в тех случаях, когда вероятность конфликта очень мала.

**Легкие блокировки**

Устанавливаются на короткое время

* Обычно доли секунды

Исключительный и разделяемый режимы

Есть мониторинг

Нет обнаружения взаимоблокировок

Пассивное ожидание

* При освобождении ресурса возникает состояние гонки, выигрывает случайный процесс.

**Закрепление буфера**

Устанавливается на время работы с буфером

* Возможно длительное

Исключительный и разделяемые режимы

Есть мониторинг

Есть обнаружение взаимоблокировок

Пассивное ожидание

* Но обычно закрепленный буфер пропускается.

1 и 2 надо записать как-то))А))А)А)А))А

**3. Согласованность**

Если мастер основан на схеме мастер-реплика с одним пишущим узлом и транзакция выполняется локально на одном из узлов гарантии и согласованности могут оказаться очень слабыми.

Например, один узел могу успеть применить запись о фиксации транзакции и ее результат уже доступен клиенту, а другой узел еще нет.

В таком случае у клиента нет гарантии, что он не прочитает старое значение на одном узле уже после того, как прочитал новое. Обработка таких ситуаций ложится на приложение.

Если кластер предоставляет глобальные (распределенные) транзакции, то можно говорить о согласованности в смысле ACID: глобальная транзакция должна переводить базу данных из одного согласованного состояния в другое согласованное при условии, что транзакция полностью выполняется на всех узлах при отсутствии помех со стороны других конкурентных транзакций.

В этом случае требуется обеспечить атомарность транзакций, все узлы кластеры должны выполнять одинаковое действие, либо применить, либо отменить транзакцию

**4. Плановая работа без прерывания обслуживания**

Время простоя может быть вызвано не только возникновением неполадок, но и плановыми работами, которые невозможно выполнить без прерывания обслуживания.

Основы обеспечения отказоустойчивости – это дублирование всех узлов системы.

При отказе узла, этот узел перестает функционировать как часть кластера. Отказ может быть вызван остановкой или сбоем сервера СУБД, сбоем ОС, выключением сервера и т.д.

Для обнаружения сбоев узлы кластера периодически обмениваются короткими сообщениями. Если какой-либо из узлов не отвечает в течении определенного времени, фиксируется и отрабатывается сбой этого узла.

Процедура отработки отказа зависит от архитектуры кластера.

Например, в кластере, построенном на схеме мастер-реплика, при сбое мастера, какая-то из реплик должна занять его место.

**Создание пул соединений**

Когда клиента подключается к серверу, для него создается отдельный процесс, с которым происходит вся дальнейшая работа этого клиента.

При большом количестве клиентов это вызывает проблемы:

* Создание соединений обходится дорого, в основном из-за того, что установка соединения требует аутентификации (особенно в случае шифрованных соединений) и наполнения кэшей (например, кэша системного каталога)
* Большое количество процессов замедляет работу сервера. Основная причина в том, что для создания снимка данных (а это частая операция) приходится просматривать список всех имеющихся процессов. И такой просмотри требует блокировки внутренних структур.
* Каждый процесс занимает определенную часть памяти сервера.

ТУТ МНОГО ИНФОРМАЦИИ КОТОРУЮ Я ПРОПУСТИЛ УВЫ УВЫ УВЫ

**Особенности полутранзакций**

Режим «пула транзакций» имеет для разработчика приложений особенности, связанные с тем, что две транзакции, выполняемые в одном клиентском сеансе, могу выполняться в разных сеансах сервера БД. Поэтому не будут работать (точнее, будут работать не та, ка задумано) любые команды, действие которых распространяется на время сеанса, а не транзакции.

**Итоги**

* На каждое клиентское соединение создается обслуживающий процесс на сервере
* Большое количество соединений приводит к проблемам
* Пул соединений позволяет одновременно работать многим клиентам, ограничивая количество соединений с сервером
* Транзакционный режим пула имеет особенности с точки зрения разработки приложений.

**Типы для больших значений**

Для хранения и обработки предусмотрены типы: cloob – символьные данные, bloob – двоичные данные. В PosgreSQL они отсутствуют, вместо clob – text, вместо bloob – bytea (bytearray).

ТУТ ОПЯТЬ МНОГО ПРОПУЩЕННОГО ТЕКСТА

Инструкция добавления значения типа bytea:

Insert into test(img) values (pg\_read\_binary\_file(`ПУТЬ`));

Смотрим размер файла, который нем содержится:

select lenghth(img) from test;

Этот размер совпадает с размером файла

Итоги

Для работы с двоичными данными представлены

* тип bytea, использующий технологию TOAST
* подсистема «больших объектов»

Использование файловой системы позволяет добиться высокой скорости за счет отказа от возможностей СУБД.

ТУТ СНОВА КОНСПЕКТ, МЕНЯ НЕ БЫЛО

**Слабоструктурированные данные**

**Применение**

Интерфейсный формат

* Между приложением и внешней системой
* Между клиентской и серверной частями приложения

Внутри базы данных как атомарное значение

* Только хранение и извлечение
* Не нарушает 1NF, достаточно обычных средств SQL

Внутри базы данных как неатомарное значение

* Операция с отдельными частями документа
* SQL не достаточно; требуется специальный язык запросов
* Гибкость, когда данные плохо укладываются в реляционную модель

Документы XML и JSON могут применяться как удобный интерфейсный формат между разным компонентами, независимый от конкретной платформы или языка программирования. Например, для взаимодействия с внешними системами, или для обмена данными между клиентской и серверной частями системы.

**Операции**

Преобразование документа к реляционному виду

* Сохранение транспортного документа в базу данных
* Использование средств SQL для обработки

Преобразование реляционных данных к виду документа

* Выгрузка данных в транспортной форме

Выделение части документа

* Специализированный язык запросов

Индексирование документов

* Поддержка операций специализированного языка запросов

**Формат XML**

eXtensible Markup Language

* универсальный язык разметки, первый стандарт 1998 года
* форматы на основе XML: XHTML, FB2, RSS и Atom, SVG, SOAP

Структура

* Вложенные элементы
* Отделяются тегами, могу иметь атрибуты, содержат текст

Инструментарий

* Описание схемы (DTD, XML Schema), языки запросов XPath и XQuery, PosgreSQL: XPath 1.0

**Формат JSON**

JavaScript Object Notation

* Простой язык разметки, стандарт 1999 года
* Появился в JavaScript, но распространен повсеместно

Структура

* Объекты (пары «ключ; значение»), массивы значений
* Значения текстовые, числовые, даты, логические

Инструментарий

* PostgreSQL; язык запросов JSONPath
* Неполная пока поддержка стандарта SQL/JSON
* Индексирование

**Типы данных для JSON**

Json

* Хранение в виде обычного текста + синтаксическая проверка
* Сохраняется исходное форматирование
* Необходимость повторного разбора при каждом запросе

Jsonb

* Внутренний формат, исключающий повторный разбор
* Поддержка SQL/JSON (язык запросов JSONPath)
* Поддержка индексирования

Метод доступа GIN

Инвертированный индекс

* Метод применяется для сложносоставных значений (массив, текст)
* API метода доступа определяет, как выделять элементы значения

Идея метода доступа GIN основана на том, что для сложносоставных значений имеет смысл индексировать элементы значений, а не все значения целиком. Для хранения элементов в GIN используется обычное B-дерево.

**Фоновые процессы**

**Фоновые процессы**

Фиксированный набор служебных процессов

* Postmaster
* Walwriter
* Checkpointer
* Autovacuum
* и другие

Динамически порождаемые фоновые процессы

* контролируются процессом postmaster
* имеют доступ к разделяемой памяти
* могут устанавливать внутреннее соединение с базами данных
* реализуются на языке C

**Использование**

Параллельное выполнение запросов

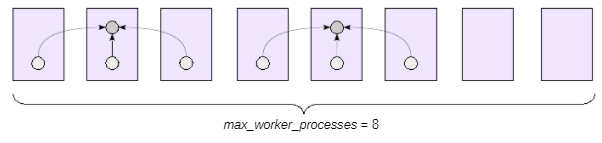
* max\_parallel\_workers = 8

Параллельное выполнение служебных команд

* max\_parallel\_maintenance\_workers = 2

Логическая репликация

* max\_logical\_replication\_workers = 4



Вот простой наглядный пример использования фоновых процессов для распараллеливания запросов. Если требуется посчитать количество строк в большой таблице, основной (лидирующий) процесс запускает несколько рабочих процессов (в данном случае 2):

**EXPLAIN (analyze, costs off, timing off)**

**SELECT count(\*) FROM mail\_messages;**

**Прикладные задачи**

Некоторые идеи

* планировщик заданий внутри СУБД
* асинхронная обработка событий
* распараллеливание сложной обработки данных
* автономные транзакции
* и другое

**Расширение dblink**

Назначение

* выполнение произвольных SQL-команд на удаленном сервере
* в качестве удаленного сервера может выступать и локальный

Плюсы

* стандартное расширение

Минусы

* фоновые процессы не используются,
* устанавливается новое соединение

**Расширение pg\_background**

Назначение

* возможность выполнить команду SQL в фоновом процессе
* команда может вызвать подпрограмму на любом серверном языке

Плюсы

* используются фоновые процессы

Минусы

* стороннее расширение

**Итоги**

Сервер предоставляет механизм фоновых процессов

Фоновые процессы используются для внутренних целей,

но могут быть использованы и для прикладных задач

Расширение pg\_background позволяет реализовывать фоновые процессы на процедурных языках

**Полнотекстовый поиск**

**Зачем текстовый поиск?**

Средства SQL — LIKE, регулярные выражения

* нет морфологического поиска
* нет возможности ранжирования результатов
* нет индексной поддержки

Внешние поисковые системы

* сложно синхронизировать с базой данных
* отсутствие транзакционности
* нет доступа к метаданным
* сложности с разграничением доступа

Документ, по которому нужен поиск, должен быть предварительно переведен в специальное представление — тип данных tsvector.

Исходный документ может быть произвольным текстом, который можно разобрать на отдельные слова (точнее, лексемы — разницу подробнее рассмотрим дальше). По двоичным документам тоже можно искать, если предварительно перевести их в текстовый вид (с помощью сторонних библиотек).

Поисковый запрос также должен быть представлен значением специального типа — tsquery. Запрос может состоять либо из одной лексемы, либо из нескольких лексем, связанных логическими операторами «и», «или», «не». Начиная с версии 9.6 также поддерживается оператор предшествования, который обеспечивает фразовый поиск: можно найти документ, содержащий заданные слова не в любом месте, а стоящие рядом (или на определенном расстоянии друг от друга).

Чтобы проверить, соответствует ли документ (точнее, его представление в виде tsvector) запросу (точнее, его представлению tsquery), надо использовать оператор @@.

Результаты запроса можно ранжировать по «релевантности», чтобы понять, какие из найденных документов больше соответствуют запросу, а какие — меньше.

В PostgreSQL есть две встроенные функции: ts\_rank (учитывает, насколько часто лексемы из запроса встречаются в документе) и ts\_rank\_cd (дополнительно учитывает близость найденных лексем). Обе функции позволяют отмасштабировать свой результат с учетом размера документа.

**Анализатор**

Выделяет в тексте документа фрагменты и назначает фрагментам типы

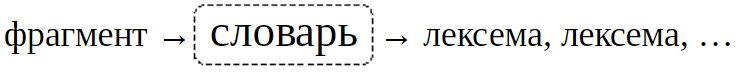
Штатный анализатор default

* 23 типа фрагментов (слова, числа, адреса, теги XML и т. п.)

Другие анализаторы

* Расширения
* набор функций на языке Си

Вначале документ пропускается через анализатор (parser), который выделяет в нем фрагменты (tokens). Более того, каждому фрагменту анализатор сопоставляет тип, который используется для дальнейшей обработки. Например, стандартный анализатор (единственный встроенный в PostgreSQL) выделяет 23 типа: слова, числа, адреса urlи email, XML-теги и другие. За счет этого он довольно универсален

**Словари**

Словарь превращает фрагмент в лексему

* убрать стоп-слова
* привести буквы к одному регистру
* привести словоформы к общему виду
* привести синонимы к одному варианту
* и т. п.

Штатные словари

* simple — нижний регистр и стоп-слова
* стемминг для 21 языка

Полученный от анализатора фрагмент пропускается через словарь и превращается в лексему (lexeme) или в несколько лексем.

Смысл такого преобразования в том, чтобы «нормализовать» слова, привести их к такому виду, чтобы их было легко найти.

Например:

* из текста можно убрать стоп-слова (которые встречаются почти в каждом документе, так что искать их бессмысленно)
* можно привести слова к одному регистру
* можно отрезать от слов изменяющиеся части (окончания), чтобы находить любые словоформы (стемминг)
* Можно свести все синонимы к одному

**Шаблоны словарей**

Штатные шаблоны

* стеммер snowball
* словарь ispell
* синонимы: приведение синонимов к одному виду
* тезаурус: приведение фраз к одному виду

**Возможности шифрования**

Postgres Pro обеспечивает шифрование на разных уровнях и даёт гибкость в выборе средств защиты данных в случае кражи сервера, от недобросовестных администраторов или в небезопасных сетях. Шифрование может также требоваться для защиты конфиденциальных данных, например, медицинских сведений или финансовых транзакций.

**Шифрование хранимых паролей**

По умолчанию, пароли пользователей базы данных хранятся в виде хешей MD5, так что даже администратор не может определить, какой именно пароль имеет пользователь. Если шифрование MD5 также применяется при проверке подлинности, пароли не присутствуют на сервере в открытом виде даже кратковременно, так как клиент вычисляет и передаёт по сети только хеши паролей.

**Шифрование избранных столбцов**

Модуль pgcrypto позволяет хранить в зашифрованном виде избранные поля. Это полезно, если ценность представляют только некоторые данные. Чтобы прочитать эти поля, клиент передаёт дешифрующий ключ, сервер расшифровывает данные и выдаёт их клиенту.

Расшифрованные данные и ключ дешифрования находятся на сервере в процессе расшифровывания и передачи данных. Именно в этот момент данные и ключи могут быть перехвачены тем, кто имеет полный доступ к серверу баз данных, например, системным администратором.

**Шифрование раздела данных**

Шифрование хранилища данных можно реализовать на уровне файловой системы или на уровне блоков. В Linux можно воспользоваться шифрованными файловыми системами eCryptfs и EncFS, а во FreeBSD есть PEFS. Шифрование всего диска на блочном уровне в Linux можно организовать, используя dm-crypt + LUKS, а во FreeBSD — модули GEOM, geli и gbde. Подобные возможности есть и во многих других операционных системах, включая Windows.

Этот механизм не позволяет читать незашифрованные данные с дисков в случае кражи дисков или всего компьютера. При этом он не защищает данные от чтения, когда эта файловая система смонтирована, так как на смонтированном устройстве операционная система видит все данные в незашифрованном виде. Однако, чтобы смонтировать файловую систему, нужно передать операционной системе ключ (иногда он хранится где-то на компьютере, который выполняет монтирование).

**Шифрование паролей при передаче по сети**

Метод проверки подлинности с MD5 дважды шифрует пароль на стороне клиента, прежде чем передать его серверу. Сначала вычисляется хеш MD5 пароля вместе с именем пользователя, а затем этот хеш обрабатывается ещё раз с добавлением случайного числа, переданного сервером при попытке подключиться к нему. Затем это дважды хешированное значение передаётся серверу по сети. Двойное хеширование не только позволяет защититься от вычисления исходного пароля, но и не даёт использовать тот же зашифрованный пароль при следующем подключении.

**Шифрование данных при передаче по сети**

SSL-соединения шифруют все данные, передаваемые по сети: пароль, запросы и возвращаемые данные. Файл pg\_hba.conf позволяет администраторам указать, для каких узлов будут разрешены незашифрованные соединения (host), а для каких будет требоваться SSL (hostssl). Кроме того, и на стороне клиента можно разрешить подключения к серверам только с SSL. Для шифрования трафика также можно применять stunnel и SSH.

**Проверка подлинности сервера SSL**

И клиент, и сервер могут проверять подлинность друг друга по сертификатам SSL. Это требует дополнительной настройки на каждой стороне, но даёт более надёжную гарантию подлинности, чем обычные пароли. С такой защитой подставной компьютер не сможет представлять из себя сервер с целью получить пароли клиентов. Она также предотвращает атаки с посредником («man in the middle»), когда компьютер между клиентом и сервером представляется сервером и незаметно передаёт все запросы и данные между клиентом и подлинным сервером.

**Защита баз данных. Повторение**

**Защита данных –** это организационные, программные и технические методы и средства, направленные на удовлетворение ограничений, установленных для типов данных или экземпляров типов данных в системах обработки данных (ГОСТ 20886-85)

Защита данных выключает предупреждение случайного или несанкционированного доступа к данным, их изменения или разрушения со стороны пользователей или при сбоях аппаратуры. Реализация защиты включает:

Контроль достоверности данных с помощью ограничений целостности;

Обеспечение безопасности данных (физической целостности данных);

Обеспечение секретности данных.

Типы ограничений целостности в языке SQL:

1. Уникальность значения первичного ключа (PRIMARY KEY)
2. Уникальность ключевого поля или комбинации значений ключевых полей: UNIQUE(A), где A – один или несколько атрибутов, указанных через запятую (1,2 – явные структурные ограничения целостности)
3. Обязательность/необязательность значения (NOT NULL/NULL)
4. Задание диапазона значений атрибута Field:

CHEK(field BETWEEN min\_value AND max\_value)

1. Задание взаимоотношений между значениями атрибутов Field1 и Field2:

CHEK(field1 @ field2), где @ - оператор отношения (например, знак “>”)

1. Ограничение на обновление данных (например, каждое следующее значение атрибута должно быть больше предыдущего). В SQL напрямую не реализуется, требует использования специальных возможностей СУБД (триггеров).
2. Ограничения на параллельное выполнение операций (механизм транзакций) и проверка ограничений целостности после окончания внесения взаимосвязанных изменений)

**Обеспечение безопасности данных**

Под функцией безопасности (или физической защиты) данных подразумевается предотвращение разрушения или искажения данных в результате программного или аппаратного сбоя.

Обеспечение безопасности является внутренней задачей СУБД, поскольку связано с ее нормальным функционированием, и решается на уровне СУБД

Цель **восстановления базы данных** после сбоя – обеспечить, чтобы результаты всех подтвержденных транзакций были отражены в восстановленной БД, и вернуться к нормальному продолжению работы как можно быстрее, изолируя пользователей от проблем, вызванных сбоем.

**Типичные сбои и способы защиты от них**

**1. Сбой предложения**

СУБД автоматически откатывает результаты этого предложения, генерирует сообщение об ошибке и возвращает управление пользователю (приложение пользователя)

**2. Сбой пользовательского процесса**

Система автоматически откатывает неподтвержденные транзакции сбившегося пользовательского процесса и освобождает все ресурсы, занятые этим процессом.

**3. Сбой процесса сервера**

Восстановление после сбоя процессора сервера может потребовать перезагрузки БД, при этом автоматически происходит откат все незавершенных транзакций.

**4. Сбой процессора операционной системы**

В этой ситуации сервер БД не может продолжать работу, и для восстановления базы данных требуется участие человека (обычно, администратора базы данных, АБД)

**5. Сбой носителя (диска)**

В этой ситуации сервер БД не может продолжать работу, и для восстановления базы данных требуется участие человека (обычно, администратора базы данных, АБД)

**6. Ошибка пользователя**

Ошибки пользователей могут потребовать участия человека (АБД) для восстановления базы данных в состояние на момент возникновения ошибки

**Средства физической защиты данных**

**Резервное копирование**

Резервное копирование означает периодическое сохранение файлов БД на внешнем запоминающем устройстве. Оно выполняется тогда, когда состояние файлов БД является непротиворечивым. В случае сбоя (или аварии диска) БД восстанавливается на основе последней копии.

**Полня резервная копия** включает всю базу данных (все файлы БД, в том числе вспомогательные, состав которых зависит от СУБД)

**Частичная резервная копия** включает часть БД, определенную пользователем. Резервная копия может быть инкрементной: она состоит только их тех блоков (страниц памяти), которые изменились со времени последнего резервного копирования.

Создание частичной и инкрементной РК выполняется средства СУБД, а создание полной РК – средствами СУБД или ОС (например, с помощью команды **copy**). В резервную копию, созданную средствами СУБД, обычно включаются только те блоки памяти, которые реально содержат данные (т.е. пустые блоки, выделенные под объекты БД, в резервную копию на входят).

**Резервное копирование**

Периодичность резервного копирования определяется администратором системы и зависит от многих факторов: Объем БД, Интенсивность запросов к БД, Интенсивность обновления данных и др.

Как правило, технология проведения резервного копирования такова:

* Раз в неделю (день, месяц) осуществляется полное копирование;
* Раз в день (час, неделю) частичное или инкрементное копирование.

**Восстановление базы данных**

В том случае, если нельзя восстановить БД после сбоя автоматически, восстановление БД выполняется в два этапа:

Перенос на рабочий диск резервной копии базы данных (или той ее части, которая была повреждена)

Перезапуск сервера БД с повторным проведение всех транзакций, зафиксированных после создания резервной копии и до момента возникновения сбоя.

**Восстановление базы данных**

Последовательность восстановления БД:

1. Устранение проблем с диском (если они были)

2. Запись резервной копии на место основной базы данных

3. Запуск сервера БД в режиме восстановления (recover)

4. Определение параметров восстановления при наличии архива журналов транзакций

**Мониторинг активности базы данных**

**Мониторинг активности базы данных (DAM)**, это технология безопасности базы данных для мониторинга и анализа активности, которая работает независимо от [системы управления базами данных (СУБД)](https://ru.wikipedia.org/wiki/%D0%A1%D0%A3%D0%91%D0%94) и не зависит от какой-либо формы внутреннего (СУБД-резидентного) аудита или собственных журналов, таких как трассировка или Журналы транзакций. DAM обычно выполняется непрерывно и в режиме реального времени.

**Привилегированный мониторинг пользователей:** мониторинг привилегированных пользователей (или суперпользователей), таких как администраторы баз данных, системные администраторы, разработчики, справочные службы и внешний персонал, которые обычно имеют неограниченный доступ к корпоративным базам данных, необходимы для защиты от внешних и внутренних угроз. Привилегированный мониторинг пользователей включает аудит всех видов деятельности и транзакций, выявление аномальных действий (например, просмотр конфиденциальных данных или создание новых учётных записей с привилегиями суперпользователя), а также согласование наблюдаемых действий (например, добавление или удаление таблиц) с разрешёнными запросами на изменение.

**Мониторинг активности приложений.** Основной целью мониторинга активности приложений является обеспечение более высокого уровня подотчетности конечного пользователя и обнаружение мошенничества (и других злоупотреблений законным доступом), которое происходит через корпоративные приложения, а не через прямой доступ к базе данных.

**Защита от кибер атак:** [Внедрение SQL-кода](https://ru.wikipedia.org/wiki/%D0%92%D0%BD%D0%B5%D0%B4%D1%80%D0%B5%D0%BD%D0%B8%D0%B5_SQL-%D0%BA%D0%BE%D0%B4%D0%B0) - это тип атаки, для использования некачественных методов кодирования в приложениях, использующих реляционные базы данных. Злоумышленник использует приложение для отправки инструкции [SQL](https://ru.wikipedia.org/wiki/SQL), которая состоит из заявления приложения, объединённого с дополнительным заявлением, которое вводит злоумышленник.

**Создание bash-сценария**

Это обычный текстовый файл с расширением .sh

Первая строка должна быть следующей: #!/bin/bash

Вывести фразу: echo `Hello world!`