**Теория БД**

**Транзакция –** Единица работы, состоящая из множества операций, которые переводят БД из одного согласованного состояния в другое.

**ACID –** свойство транзакций, направленные на обеспечение надежности и целостности данных.

*Atomicity* (Атомарность) – Транзакция не может быть зафиксирована частично, то есть не может иметь промежуточных состояний, она либо выполнена полностью, либо не выполнена вовсе.

*Consistency* (Согласованность) - свойство, гарантирующее, что каждая успешная транзакция зафиксирует только допустимые результаты. То есть, это гарантия того, что при успешной транзакции будут выполнены все правила, ограничения, которые предъявляет система к конкретным данным, иначе транзакция не будет выполнена и данные в системе вернутся к прежнему состоянию.

*Isolation* (Изолированность) – Параллельные транзакции (транзакции, работающие с одними и теми же данными) не должны влиять на результаты выполнения друг друга.

Durability (Надежность) – Если клиент получил подтверждение от системы подтверждение завершения транзакции, то эти изменения не будут отменены из-за сбоя.

***Уровни изоляции транзакций:***

***Read uncommitted –*** Самый низкий уровень изоляции. Изменения видны транзакциям, даже если они еще не были зафиксированы. В PostgreSQL не реализован (можно указать, но он работает как Read committed).

Возможные эффекты: грязные чтения, неповторяющиеся чтения и фантомные чтения.

***Read committed*** – Транзакции видят только подтвержденные изменения. Вторая транзакция может увидеть изменения первой во время своей транзакции, если использует те же данные. По умолчанию в PostgreSQL.

Возможные эффекты: неповторяющиеся чтения и фантомные чтения.

***Repeatable read*** – Транзакция не увидит Update и Delete других транзакций до завершения своей транзакции, но Insert возможны. По умолчанию в MySQL.

Возможные эффекты: фантомные чтения.

***Serializable*** – Максимальный уровень изоляции. Блокирует чтение. Не содержит эффектов, но ухудшает параллелизм и производительность.

\*Уровни изоляции и соответствующие им аномалии не соответствуют таковым в PostgreSQL, актуальную для него информацию см. на картинке ниже\*

Уровень изоляции устанавливается с помощью **Set local/session transaction isolation level** <уровень изоляции> - local по умолчанию, нет параметра global для всей БД.

Так же локальный уровень изоляции можно задать во время начала транзакции

**Begin transaction isolation level** <уровень изоляции>;

**Show transaction\_isolation** – показывает текущий уровень изоляции.

****

**Аномалии:**

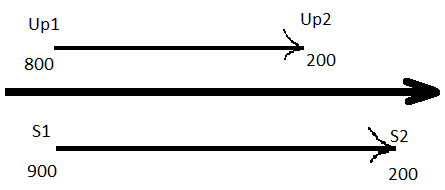
*Потерянное обновление* – Работает как race condition (при вычитывании 2мя транзакциями значения, будет видимо то изменение, которое будет наложено последним). Не допускается ни на одном уровне транзакций.

*Грязное чтение* – Возможность транзакции считать, обновить или добавить данные, изменения которых впоследствии не подтвердится. Не допускается ни на одном уровне изоляции из-за протокола изоляции на основе снимков.

*\*Несогласованное чтение* – чтение данных, которые находятся в несогласованном состоянии из-за изменений в другой транзакции, то есть совокупность одних и тех же данных из двух транзакций дает разный результат.

Пример: Пусть счет 1 =900, счет 2 = 100;

Транзакция 1 переводит средства со счета 1 на счет 2, то есть делает update -100 и update +100, вторая транзакция начинает чтение счета 1 до commit изменений транзакции 1 и видит неизмененное значение (900), затем транзакция 1 фиксирует изменения (счет 1 = 800, счет 2 = 200), после чего транзакция 2 читает счет 2 (200) и общая сумма получается 1100, а не 1000.



Эта аномалия возможна только на уровне Read commited или при использовании функций с изменчивостью volatile – что значит, что результат функции зависит от внешнего контекста.

*Неповторяющееся чтение* – При повторном чтении в рамках одной транзакции данные изменяются.

*Фантомное чтение* – При повторном чтении изменяется кол-во строк в результирующей выборке за счет изменения другой транзакцией критериев выборки или добавление/удаление строк выборки.

На уровне изоляции repeatable read и serializable отсутствуют аномалии неповторяющегося чтения и фантомного чтения.

Пример для запуска: Первая транзакция вставляет новую строку и изменяет данные в таблице, не фиксируя изменения. Вторая транзакция читает таблицу (изменения не будут видны). Commit 1 транзакции и запуск select в транзакции 2 – изменения не будут видны в транзакции 2 до ее завершения даже после коммита транзакции 1.

|  |  |
| --- | --- |
| Код транзакция 1:  begin;  CREATE TABLE accounts(  id integer PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,  number text UNIQUE,  client text,  amount numeric  );  INSERT INTO accounts VALUES  (1, '1001', 'alice', 1000.00),  (2, '2001', 'bob', 100.00),  (3, '2002', 'bob', 900.00);  commit;  BEGIN;  UPDATE accounts SET amount = 200.00 WHERE id = 2;  UPDATE accounts SET amount = 800.00 WHERE id = 3;  INSERT INTO accounts VALUES  (4, '3001', 'charlie', 100.00);  SELECT \* FROM accounts ORDER BY id;  --запустить после второй транзакции  commit; | Код транзакция 2:  begin isolation level repeatable read;  select \* from accounts order by id; |

\*

После снятия блокировки UPDATE перечитывает строку, которую пытается обновить

В read commited нельзя опираться на результаты запроса в следующием запросе так как данные между ними могут меняться

Read commited не может потерять изменения, поэтому он перечитывает заблокированную строку и изменяет ее состояние. То есть он работает с набором снимков.

Потерянные изменения возможны, если разбить запрос на select и записать состояние в переменную, затем на основании анализа этой переменной произвести изменения. В этом случае результат переменной перезапишется поверх первой транзакции.

Repeatable read – работает только с 1 снимком, поэтому он не может перечитывать строки, что ведет невозможности предотвращения потерянных изменений и транзакция завершается с ошибкой сериализации.

Несогласованная запись –возможна на repeatable read

Serializable требует, чтобы все транзакции были того же уровня изоляции при несогласованной записи вылетит ошибка сериализации

Есть стандартные аномалии, которые не могут быть допущены на определенном уровне изоляции и выбор делается в сторону нестандартных

То есть на read commited между потерянными изменениями и несогласованным чтением (посредством перечитывания блокируемой строки) выбор делается в пользу последнего, т.к потерянные изменения запрещены, в repeatable read запрещено и перечитывать строки и терять изменения, поэтому в таких ситуациях ошибка сериализации

Аномалия только читающей транзакции – есть на repeatable, при serializable оператор select блокируется.

\*

[Про уровни изоляции и аномалии](https://habr.com/ru/companies/postgrespro/articles/442804/)

**Физическая организация БД**

Объекты БД в целом имеют схожую структуру – большинство из них хранят в себе данные и имеют строчную организацию (последовательность – однострочная таблица, материализованные представления – таблица, которая помнит запрос и т.д.).

Состоят из слоев: 1) Основной слой – хранит в себе данные, индексные или табличные строки. Имеется у всех отношений, кроме представлений, т.к. они не хранят данных.

2) Слой инициализации – существует только у нежурналируемых таблиц и индексов, записи которых не попадают в журнал предзаписи (работа с ними быстрее, но потеряют данные в случае сбоя).

3) Карта свободного пространства (free space map) – Слой, в котором отмечено наличие пустого места внутри страниц. Помогает избегать фрагментации данных.

4) Карта видимости (visibility map) – слой в виде битовой карты, в котором отмечены страницы, содержащие только актуальные версии строк (то есть на данный момент не редактируются ни в одной транзакции).

Каждый слой представлен файлом до 1Гб, после заполнения этого пространства добавляется новый файл (сегмент). Каждый файл разделен на страницы по 8Кб (стандарт, может быть 16, 32, но нужно пересобрать базу), страница содержит различные версии строк и массив указателей на них.

**Версии строк и снимки**

Каждая версия строки должна храниться в рамках одной страницы, если этого не происходит, то используется либо стратегия сжатия до допустимого размера, либо использование стратегии TOAST – создается служебная таблица и версия строки «режется» на страницы допустимого размера, к этой стратегии также применимо сжатие.

Каждая версия строки содержит служебную информацию – номер транзакции, которая создала строку – поле xmin, транзакция, удалившая строку – поле xmax, помимо номеров транзакций эти поля хранят информацию о состоянии этих транзакций (завершена или продолжается), которая дублируется из журнала статусов транзакций.

Записанная в xmax транзакция со статусом незавершенности – признак блокировки, то есть построчные блокировки располагаются в самих данных, что не ведет к перерасходу памяти

Снимки данных – состояние данных доступное на момент начала транзакции. Состоит из зафиксированных версий строк, то есть транзакции, изменяющие эти строки должны быть завершены, и снимок должен содержать только одну версию определенной строки. То есть снимок данных дает согласованную картину данных на данный момент времени, которая лежит в основе MVCC (Multi-Version Concurrency Control) – метод управления параллельным доступом в БД за счет многоверсионности строк без блокировок.

*Read committed* – снимок данных создается в начале каждого оператора транзакции, *repeatable read и serializable* – снимок создается в начале первого оператора.

xmax – номер следующей (еще не существующей транзакции) xip\_list – список активных транзакций, xmin – номер самой ранней активной транзакции – для оптимизации и определения горизонта событий (По нему определяются версии, которые гарантировано находятся в прошлом, то есть они точно не будут использованы и точно не участвуют в активных транзакциях, что говорит о том, что их можно подвергнуть vacuum). За счет этих параметров можно определить, какие версии доступны, а какие нет.

**Очистка –** удаление невидимых для активных транзакций записей. Необходимость очистки тесно связана с операциями delete и update и зависит от частоты использования этих операций, т.к. update включает в себя delete строки (отметка в xmax) и insert строки с новым значением. Про этом строки продолжают существовать в БД и их физическое удаление происходит только при очистке.

Внутристраничная очистка – работает в рамках одной табличной страницы, не затрагивает индексы, не может быть запущен вручную - активируется, если подразумевается, что место на странице заканчивается (если при update не хватило места для вставки или страница заполнена больше, чем на fillfactor). При этом удаляются все версии строки, не видимые ни в одном снимке (находятся за горизонтом событий).

Vacuum – Выполняется на уровне таблицы, просматриваются только страницы таблицы не отмеченные в visibility map (очистке подлежат только те строки, которые не используются ни в одной транзакции), неотмеченные страницы пропускаются и помечаются для обработки при следующем запуске. Актуализирует информацию о просмотренных страницах в free space map и в visibility map. не блокирует операции чтения и изменения таблицы. Помимо очистки записей таблицы, также очищает и индексные записи. Освобожденное пространство помечается как свободное, но не возвращается операционной системе.

Помимо очистки, vacuum также занимается «заморозкой» строк – помечает строки, существующие достаточно давно, как устаревшие. Механизм нужен из-за наличия цикличной нумерации транзакций – нумерация транзакций заключена в 32 бита, что является оптимальным для хранения в заголовках строк, что хоть и является достаточно большим числом (более 4 млрд), но конечным, поэтому со временем транзакциям приходится использовать уже использованный ранее номер строки. Из-за этого может возникнуть конфликтная ситуация, когда есть достаточно старая строка с неким номером транзакции, который используется повторно уже другой транзакцией. Заморозка строк и позволяет избегать конфликта номеров.

Vacuum full – заново пересобирает таблицу, видимые записи копируются в новый файл и он становится основным при этом «лишнее» свободное место возвращается операционной системе (применяется, если таблица сильно выросла в размерах из-за накопившихся старых записей). Также пересобираются и индексы. Блокирует таблицу и все ее индексы.

Reindex – полностью пересоздает индекс без перестройки таблиц. Блокирует индекс и запрещает запись в таблицу, reindex concurrently – не блокирующий, но перестраивается дольше (с 12 версии).

Vacuum analyze – вакуум вместе со сбором статистики для оптимизации запроса.

**Буферный кэш и журналирование**

Буферный кэш – Разделяемая область в оперативной памяти, используемая для временного хранения данных в целях ускорения доступа к ним. Снижает кол-во обращений к диску за счет того, что запрашиваемые данные сначала ищутся в буфере и только в случае их отсутствия она будут вычитаны с диска сначала в буфер, а затем переданы в запрос.

Когда буфер заполняется, то подключается механизм вытеснения, организованный на основании счетчика обращений (каждое чтение страницы из буфера прибавляет 1 (5 максимум), а каждый проход вытесняющего потока -1 для каждой страницы), вытесняется та страница, счетчик обращений которой равен 0, и она не будет заблокирована другими процессами. Если вытесняемая страница была грязной (была изменена в кэше, но изменения не были сохранены на диск), то подключается поток предзаписи, который запишет данные на диск, а затем страница вытесняется.

Журнал предзаписи – хранит в себе записи, содержащие действия, произведенные над страницами в оперативной памяти (изменения страниц, фиксация или отмена транзакций). Подразумевает, что данные из журнала предзаписи должны попасть на диск до самих данных.

Процесс журналирования так же контролируется отдельным процессом (checkpointer), который занимается постепенным очищением журнала предзаписи путем отправки изменений на диск (контрольная точка).

При возникновении сбоя, записи из журнала вычитываются, и данные, утраченные в оперативной памяти, восстанавливаются посредством повторения действий из журнала. То есть возникновение журнала перезаписи напрямую связано с использованием буферного кэша.

*Вся теория в серии* [*статей*](https://habr.com/ru/companies/postgrespro/articles/442804/) *и* [*видео*](https://www.youtube.com/watch?v=g3NH55nQgX4&list=PLaFqU3KCWw6K3AyBVcZGdXMtM2xvjHA1N&index=2&ab_channel=PostgresProfessional)

**Обработка и исполнение запроса**

*Разбор (Parsing) -* Синтаксический разбор - СУБД анализирует SQL-запрос, чтобы удостовериться, что он соответствует правилам языка SQL. Семантический разбор - проверка существования таблиц и полей, а также правильность их использования. Для проверки используется системный каталог.

*Трансформация –* замена представлений на таблицы, преобразование SEARCH и CYCLE для рекурсивных запросов.

*Планирование (Query Planning) -* Планировщик запросов создает дерево выполнения запроса и выбирает вариант исполнения с наименьшей стоимостью, используя статистику из системного каталога.

*Выполнение* - СУБД выполняет запрос в соответствии с выбранным планом выполнения.

Данные этапы выполняются для каждого запроса.

Prepare - Подготовленные запросы в SQL позволяют СУБД сохранять оптимизированный план выполнения запроса и повторно использовать его при последующих выполнениях запроса с разными параметрами. Это позволяет уменьшить накладные расходы на планирование и оптимизацию запроса каждый раз при его выполнении. (отсекается часть работы – не выполняется повторно).

**База данных и схема**

База данных представляет набор связанных данных (в виде отношений), хранящиеся в структурированном виде (в виде таблиц). Помимо таблиц обычно содержит другие объекты – индексы, процедуры, представления. То есть это хранилище для всех объектов.

Схема данных логически сгруппированные объекты БД, т.е. схема обеспечивает не организацию данных в каждой отдельной таблице, а объединяет сущности БД в отдельную логическую группу – схему. Так же позволяет обеспечивать контроль доступа к объектам и безопасность.

Т.е. БД — это хранилище данных для всех объектов, а схема — это набор данных внутри БД, сгруппированных логически.

Схема – для разграничения пространства имен и ограничения прав доступа.

Проектирование БД – [1](https://www.youtube.com/watch?v=HnRXzrg3Sd4&list=PLf30vI0hEi1v435cBmZSHkr1QAJdOk9mb&index=5&ab_channel=RclassTech), [2](https://www.youtube.com/watch?v=tNQlFW3BRUc&list=PLLACapFZr4XvyQgxZaIOK-MJwblGVCL41&index=7&ab_channel=ListenIT)