Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский Авиационный Институт»

(Национальный Исследовательский Университет)

Факультет прикладной математики и информатики

Курсовой проект

по курсу «Вычислительные системы»

1 семестр

Реферат по теме

«Базы данных»

|  |
| --- |
| Руководитель |
| Довженко А.А. |
|  |
| (подпись) (дата) |
| Студент |
| Группа М8О-113Б-21 |
| Соломатина С.В. |
|  |
| (подпись) (дата) |
|  |
| (оценка) |

Москва, 2021г

Содержание

[Введение 3](#_Toc5433)

[Модели данных 4](#_Toc809)

[ACID и проблемы реляционных БД 6](#_Toc3355)

[Нормализация и денормализация 8](#_Toc28031)

Индексы и хэш-таблицы в БД 9

Хранение БД 10

Структура БД 12

Конфликт транзакций 15

Язык SQL 16

Заключение 21

[Список литературы](#_Toc22754) 22

### Введение

В наши дни существует несчётное количество сайтов, онлайн-магазинов, интернет-сервисов. Трудно представить, чтобы все потоки информации, проходящие через них ежедневно, размещались на бумажных носителях —  в таком случае архивы занимали бы здания, а обработка данных —  недели, месяцы. В нашу эпоху интернета для работы с информацией используются базы данных (БД), и, конечно, спектр их применения не ограничивается только интернетом с его веб-технологиями: это и производство, и промышленность, и научная деятельность, и маркетинг, и мобильные устройства, финансовая и банковская деятельность, телевидение, телекоммуникации, реклама.

Базы данных выступают как средство для сбора и организации информации. Они могут хранят структурированную информацию о людях, продуктах, заказах и т.д. Многие базы данных начинаются с программ обработки текста или листа таблицы; с ростом этого листа появляются дубликаты и несоответствия. В такой форме хранения данные становится трудно понимать и анализировать, кроме того, лист ограничивает пользователя в возможностях поиска обработки лишь части данных. Когда подобные проблемы возникают, лучшей идеей становится перенос данных в базу данных под управлением систем управления.

Компьютеризованная база данных —  это контейнер объектов. Одна база данных может содержать больше одной таблицы. Например,

система инвентарного контроля, использующая 3 таблицы —  это не 3 базы данных, а одна, содержащая три таблицы.

Таким образом, база данных —  это совокупность данных, хранимых в соответствии со схемой данных (метаданными), манипулирование которыми выполняют в соответствии с правилами средств моделирования данных. Однако помимо хранения, возникает потребность в обработке информации. За это, так же как и за сохранность, возможность изменять (обновлять, дополнять, удалять), искать и отбирать данные, выводить результаты их обработки, отвечают системы управления базами данных (СУБД) —  комплекс программ, позволяющих манипулировать базами данных. Из этого исходят проблемы, которые должны решать СУБД —  это многопоточность, способы структуризации, хранения, уточнения данных.

### Модели данных

Из-за огромного спектра применения баз данных существует множество моделей БД, различающихся представлением, обработкой и скоростью обработки, способом хранения информации, одновременности доступа к информации, наличию взаимосвязей внутри данных, безопасности проведения изменения данных и т.д. Чаще всего БД классифицируют по используемой модели данных (абстрактная модель, организующая элементы данных):

* **Иерархическая**, следующая принципу один-многим (хранит данные в виде B-tree, для каждого отдельного объекта существует один путь, у каждого объекта один «родитель»),
* **Сетевая**, следующая принципу многие-многим (это расширение иерархической модели данных: объекты такой БД могут быть связаны не единственным путём),
* **Реляционная**, в которой данные хранятся в виде таблиц, с независимыми друг от друга столбцами и строками уникальными рядами и именами столбцов; таблицы связаны между собой общими столбцами. Они наглядно показывают отношение между объектами и обеспечивают целостность. Их недостаток —  требование чёткой структуры. В случае с типизированными объектами —  это преимущество, но когда требуется хранить лог операций или описание объектов с различными характеристиками реляционная БД может стать неоправданно большой.
* **Нереляционная** —  это структура БД, в отличие от реляционных баз вовсе не или слабо типизированна, что позволяет добавлять в строке или документе не задекларированное ранее поле, т.е. эффективно работать с разреженными данными. В нереляционных БД работа с сущностями —  таблицами в реляционных БД —  ведётся как с целостными объектами —  данные здесь представлены в виде агрегатов или объектов.  
  Семейство нереляционных моделей данных разделяют на четыре вида: ключ-значение, документоориентированные, столбцовые и графовые базы данных.   
  Базы *ключ-значение* —  это БД, в которых данные хранятся по ключам без ограничения на структуру как ключа, так и значения. Такая база хороша тем, что она, подобно индексу, очень быстро ищет данные. Именно поэтому ключ-значение очень часто используется для кэша. Эта модель данных без определённой схемы, структуры значений. Благодаря тому, что это очень простая структура, она имеет высокую скорость и легко масштабируется. У нас уже есть ключи, и мы можем очень легко их разделять, сделать их хэши. Это одна из самых высокомасштабируемых баз данных.  
  *Документоориентированные* базы данных очень похожи на key-value в какой-то из областей их применения. Но у них единицей является документ. Это такая сложная структура, по которой мы можем выбирать определённые данные, обновлять данные с диска. Каждый документ может хранить в себе XML, JSON или BSON — бинарно-сохраняемый JSON —  своеобразную пару ключ-значение. *Документоориентированные* БД обладают очень высокой доступностью и гибкостью данных. В любой документ, в любую структуру можно записать абсолютно любой набор данных. И они очень часто применяются — например, когда нужно сделать какой-нибудь каталог и когда каждый продукт в каталоге может иметь разные характеристики.  
  Другая модель, в которой удобно хранить данные, — *столбцовые* БД. Такая база данных подразумевает, что мы храним данные на диске не строчками, а столбцами. Используется для очень быстрого поиска по огромному объёму данных. Как правило — для аналитики, когда нужно выбрать значения только из определённых столбцов. Преимущество таких БД —  высокая скорость обработки запросов и большая гибкость данных, потому что мы можем добавлять любое количество столбцов, не меняя структуру.  
  *Графовые* БД содержат узлы и ребра, используемые, чтобы показать отношения между узлами. Такая БД очень гибкая, у неё высокая скорость поиска, поскольку у каждого узла есть рёбра, отражающие все отношения.

Как и говорилось выше, нет лучшей или худшей модели данных —  надо рассматривать конкретную задачу, используемые в ней данные и находить оптимальный подход к вариант. В этой исследовательской работе рассмотрим реляционную модель данных, как популярнейшую и предлагающую большой функционал работы с данными, притом не вызывающую сложностей в понимании и контроле структуры данных при параллельной работе с базой разных проектов из-за отсутствия каких-либо словарей на стороне базы.

ACID и проблемы реляционных БД

Прежде всего, скажем о трудностях, с которыми сталкиваются базы данных. Это валидация, синхронизация данных, одновременность запроса данных, из которой возникает так называемая проблема race conditions, когда над одними и теми же данными работает больше одного пользователя. Этим обусловлены ситуации, когда процесс получает неполные, временные, отменённые или неактуальные данные, не соответствующие последнему сохранению БД. Для решения подобных проблем пришли транзакции —  группа последовательности операций с БД (запросов к БД; любая операция имеет 2 свойства время вызова и время завершения, строго большее времени вызова), логически представляющая собой единую операцию.

Каждая реляционная БД в той или иной степени удовлетворяет условиям стандарта ACID (Atomicity, Consistency, Isolation, Durability). Он устанавливает гарантии, предоставляемые БД при работе с транзакциями. Рассмотрим эти гарантии.

Атомарность: это условие, при котором транзакция либо полностью выполняется, либо не выполняется вовсе; в случае неудачного выполнения транзакции система возвращается в состояние «до», то есть транзакция откатывается. БД с принципом атомарности преследуют надёжность (пример —  оплата в интернет-магазине с потерей соединения, деньги не спишутся со счёта покупателя и не зачислятся на счёт продавца), но требуют быстрого исполнения транзакций, то есть возникает ограничение на количество операций, выполняемых одновременно (в таком случае может быть ошибка, связанная с разницей времени выполнения транзакций: начались одновременно, одна закончилась, другая нет, данные разошлись). Быстрое исполнение транзакций также невозможно, если сервера БД расположены далеко от конечного пользователя (например, при заказе авиабилетов в другой стране или бронировании отеля).

Консистентность —  это гарантия соблюдения правил, применяемых к сохраняемым данным, например, касающаяся первичных или вторичных ключей, индексов, default-значений, типов столбцов и т.д. Различают физическую и логическую консистентность: физическая гарантирует корректную запись изменённых данных (операция будет либо полностью выполнена, либо не выполнена совсем), логическая обеспечивает корректное чтение данных (если две или более транзакции конфликтуют по данным, сначала выполняется (или не) выполняется первая, после чего (или не) вносятся изменения в БД, и только потом чтение этих данных доступно другой транзакции).

Изолированность транзакций показывает на то, как сильно друг на друга влияют параллельно проводящиеся транзакции. Полностью изолированные транзакции - те транзакции, что не видят промежуточных значений других транзакций, БД выполняет одновременные операции словно бы друг за другом. Изолированность помогает предотвратить нежелательное состояние БД, чтение, запись некорректных данных. Притом есть понятие уровня изоляции транзакций, выбирая который, мы приходим к консенсусу между скоростью выполнения отдельных транзакций и их согласованностью между собой. Не все уровни изоляции общего стандарта реализованы в некоторых базах данных.

* Read uncommitted. Каждая транзакция видит незафиксированные изменения другой транзакции. Феномен грязного чтения.
* Read committed. Параллельно исполняемые транзакции видят только зафиксированные изменения из других транзакций. Феномен неповторяющегося чтения, когда мы видим удалённые и обновлённые строки.
* Repeatable read. Мы не видим удалённые, обновлённые строки, но видим новые данные из другой транзакции. Это называется фантомным чтением.
* Serializable. Уровень, при котором транзакции ведут себя независимо; пока одна не завершила свою работу, мы не можем проводить какие-либо операции.

Многие БД используют свои уровни изоляции или не включают в себя все приведённые выше уровни. Пользователь или разработчик может переопределить уровень изоляции по умолчанию.

Надёжность: гарантия того, что при любом сбое не будет потери результатов транзакции или содержимого БД. Достигается за счёт фиксации данных в энергонезависимом носителе памяти, например, на жёстком диске. Притом, что очевидно, не гарантирует моментальную фиксацию информации.

### Нормализация и денормализация

Теперь представим, что у нас есть компьютерный клуб, и мы записываем клиентов, их заказ и время заказа. Мы можем записать все данные в одну большую таблицу, широкую, где будет много дублирующихся данных —  кто, когда, на сколько забронировал тот или иной ПК. А можем всё записать в различные таблицы, а между ними установить связи, то есть нормализовать наши данные.

Нормализация  — это разбиение данных на таблицы с целью, чтобы эти данные стали более структурированы. Нормализованная структура данных обеспечивает целостность данных: в базу данных будут записаны только заказы тех клиентов, которые мы знаем, бронирование известных нам компьютеров. Мы обеспечиваем факт того, что всегда сможем собрать общую картину целиком.При этом в определённую таблицу попадают соответствующие данные: только ID, только даты, имена и т.д. Таким образом, мы сокращаем размер базы данных, соответственно, упрощаем запись в в эту БД. Итак, нормализация способствует

* упрощению системы: из нормализованной структуры проще видна логическая связь между таблицами
* уменьшению количества ошибок и устранению избыточности данных: например, если мы записываем заказ на пользователя, которого в системе нет, то нам придётся его завести. В дополнение, при записи заказа на того же клиента будет создана лишь соответствующая запись о заказе, но не добавлен новый пользователь. Значит, итоговые данные останутся целостными.
* уменьшению размера БД: нам не приходится каждый раз писать все данные о клиенте, чтобы посмотреть информацию о нём достаточно искать в соответствующей таблице
* устранению несогласованности зависимостей: достигаемому с помощью уникальных идентификаторов или primary keys
* защите данных от случайного удаления: мы не можем удалить пользователя, потому что, например, у него есть запись о заказе

Однако для упрощения анализа данных часто применяется денормализация —  намеренное нарушение нормальных норм. Например, мы захотим посмотреть общее количество посещений клуба всеми клиентами за долгосрочный период. В таком случае вместо подсчёта каждого отдельного посещения можно рассчитывать сначала количество посещений за день, неделю, месяц —  выборка таких данных будет производится быстрее. При этом важно сохранить и детальную информацию, на случай, если потребуется посмотреть количество посещений определённого клиента или в определённый день. Сложность, возникающая при денормализации, состоит в том, что добавляя данные, нужно обновлять эти столбцы, то есть увеличивается вес операций обновления и изменения данных. По этой причине денормализация применяется только там, где она действительно нужна.

### Индексы и хэш-таблицы в БД

Что, если компьютерный клуб перерастёт в сеть? Таблица данных будет постоянно расти, чтобы искать в ней данные, нужно перебирать все заказы, проверять тем ли клиентом они сделаны, в то ли время, по тому ли адресу, и только тогда выводить результаты анализа. Решение, которое позволяет быстро искать нужную информацию в таблице —  это индексы (зачастую —  primary key).

Индексы в БД —  это отдельная структура, независимая от таблиц. Они помогают искать и выделять часть данных таблицы. Поиск по индексам в БД обеспечивается различными алгоритмами. Наиболее распространёнными являются B-дерево и B+-дерево, работающие по принципу разделения и балансирования значений индексов справа и слева от корневого узла и далее. B+-дерево отличается от B-дерева тем, что информацию хранят лишь самые нижние узлы, а все остальные узлы предназначены для оптимизации поиска по дереву. Важно упомянуть и хэш-таблицы, реализующие поиск на основании информации о ключе для каждого элемента таблицы, хэш-функции для вычисления хэша (логического блока) по ключу, функции сравнения ключей внутри хэша. При правильно составленном запросе к такой хэш-таблице, поиск осуществляется со сложностью О(1).

### Хранение БД

Как хранятся базы данных? В одном «экземпляре» хранить её явно небезопасно: если с ней что-то случается, мы можем потерять все данные или, в случае с поломкой сети или узла, столкнёмся с простоем системы. Разрешается эта проблема репликацией БД, то есть полным копированием данной БД на другие узлы и серверы.

Помимо увеличения сохранности данных, репликация, во-первых, позволяет распределить нагрузку на сервер: одна и та же информация в один момент времени может быть доступна с разных баз, так как мы можем масштабировать количество реплик на чтение. Во-вторых, копирование БД даёт возможность покрыть максимальное количество пользователей при установке новых серверов с копией этой БД в разных географических точках. Кроме того, при проведении долгой аналитики, когда важно, чтобы данные в транзакции были заблокированы, можно задействовать дублированную БД, чтобы дать возможность клиентам в то же время работать с базой, впоследствии обновив изменённые ими данные и в реплицированной базе.

Для грамотного распределения и обновления копий БД, вводится понятие ведущего и ведомого (реплика) узлов, Master и Slave. В узел Master пользователь вносит изменения; этот узел их принимает, записывает в журнал изменений, отправляет этот журнал на узел Slave. Slave меняет только те данные, изменение которых отражено в журнале. Примечательно, что и Master, и Slave должны иметь одну и ту же конфигурацию, потому что разные версии базы могут иметь и разный формат журнала операций. Рассмотрим также разрешение проблемы синхронности запросов. Если запрос синхронен, то Master отправляет запрос на синхронный Slave, и ждёт подтверждения изменений, после чего сам возвращает подтверждение пользователю. Если же реплика асинхронная, то Master отправляет запрос на реплику, но возвращает подтверждение пользователю.

Несколько синхронных реплик могут затруднить прохождение транзакций: если хотя бы с одной не придёт подтверждение об изменённых данных, транзакция полностью откатывается. По этой причине обычно ограничиваются одной синхронной репликой, гарантирующей, что данные сохранились куда-то помимо Master, а другие делают асинхронными.

Ведущих узлов может быть несколько —  в них удобно делать запись. Читать же удобно с синхронной реплики, поскольку она с максимальной точностью гарантирует, что в ней корректные данные. Однако если важна абсолютная точность данных, следует направлять запрос в Master: в синхронный Slave данные придут с опозданием. В случае, когда актуальность данных не важна, можно считывать данные и из асинхронной реплики.

Такая репликация часто применяется в офлайн-приложениях, для совместного редактирования онлайн-документов, либо когда очень велика вероятность потери сети.

Что, если данных настолько много, что сервер не справляется? Первое решение — вертикальное масштабирование —  купить более дорогую машину с большим количеством памяти, с большим CPU, с большим диском. Однако здесь мы сталкиваемся с проблемой дороговизны железа. Однажды покупать новую машину станет слишком дорого, либо будет уже просто некуда расти. Есть огромное количество данных, которые просто физически невозможно разместить на одной машине. В таких случаях можно использовать горизонтальное масштабирование —  увеличение количества машин.

Чтобы разбить данные по машинам, используют секционирование. Данные разделяются по определённому признаку и отправляются на разные машины. Таким образом, оборудование может стать менее производительным, но система по-прежнему сможет функционировать и получать данные уже с разных машин. Проблема поиска нужных данных разрешается по-разному, чаще всего это таблица с соответствиями секций, копий и данных либо программная прослойка, хранящая знания о диапазонах значений в каждой секции. Но бывают случаи, когда не используется специальное хранилище данных и клиент проверяет каждую секцию на наличие данных, соответствующих его запросу. «Разбивая» данные, важно правильно подобрать ключ, по которому можно будет их прочесть, определив те данные, которые потенциально будут использоваться и понимая, насколько сбалансирована будет система машин.

Что лучше —  разбиение на секции или индексы? Ответ —  индексы. Он обусловлен тем, что секции —  это отдельные машины, и система так или иначе получается куда как более ресурсоёмкой. Так что горизонтальное масштабирование производится только при действительно большом количестве данных.

### Структура БД

Итак, БД представляет собой совокупность информации, к которой можно легко получить доступ и модифицировать. По сути, это группа файлов, но чтобы использовать транзакции для обеспечения сохранности и связанности данных и быстро обрабатывать данные вне зависимости от их объёма важно, как именно организованы эти файлы и как связанны между собой.

В целом структура БД выглядит так:

Основные компоненты БД (Core Components):

* Диспетчер процессов. Во многих БД имеется пул процессов/потоков, которыми нужно управлять. Причём в погоне за производительностью некоторые БД используют свои собственные потоки, а не предоставляемые операционной системой.
* Диспетчер сети. Пропускная способность сети имеет большое значение, особенно для распределённых БД.
* Диспетчер файловой системы. Очень важно иметь диспетчер, который идеально работает с файловой системой ОС или даже заменяет её.
* Диспетчер памяти. Чтобы не упереться в невысокую производительность дисковой подсистемы, нужно иметь много оперативной памяти. А значит, нужно эффективно ею управлять, что и делает данный диспетчер. Особенно это важно, когда имеется много одновременных запросов, использующих память.
* Диспетчер безопасности. Управляет аутентификацией и авторизацией пользователей.
* Диспетчер клиентов. Используется для управления соединениями с клиентами — веб-серверами или конечными пользователями/приложениями. Сначала проводит аутентификацию, потом авторизует для использования БД. Далее диспетчер проверяет, есть ли доступный процесс (или поток), который может обработать ваш запрос, сверяет, не находится ли база в состоянии высокой нагрузки. Диспетчер может подождать, пока не высвободятся нужные ресурсы. Если период ожидания закончился, а они не высвободились, диспетчер закрывает соединение с сообщением об ошибке. Если же ресурсы имеются, то диспетчер перенаправляет запрос диспетчеру запросов. Поскольку диспетчер клиентов передаёт и получает данные от диспетчера запросов, то он сохраняет их в буфере частями и отправляет вам. При возникновении проблемы диспетчер прерывает соединение и высвобождает ресурсы.

Инструменты (Tools):

* Диспетчер резервного копирования.
* Диспетчер восстановления. Используется для восстановления согласованного состояния БД после сбоя.
* Диспетчер мониторинга. Занимается протоколированием всех активностей внутри БД и используется для наблюдения за её состоянием.
* Диспетчер общего управления. Хранит метаданные (вроде наименований и структуры таблиц) и используется для управления базами, схемами, табличными пространствами и т.д.

Диспетчер запросов (Query Manager):

* Парсер запросов. Проверяет их валидность.

Каждый SQL-оператор отправляется в парсер и проверяется на правильность синтаксиса. Отклоняет запрос, если в нём содержится ошибка проверяет порядок использования ключевых слов, анализирует таблицы и их поля внутри запроса. Парсер использует метаданные БД для проверки самого факта существования таблицы, наличия в ней полей, возможности осуществления операций с полями имеющихся типов. Проверяет права доступа пользователя. В ходе всех этих проверок запрос трансформируется во внутреннее представление (чаще всего в дерево). Если всё в порядке, то оно отправляется в рерайтер запросов.

* Рерайтер запросов. В его задачи входит:
* Предварительная оптимизация запроса.
* Исключение ненужных операций.
* Помощь оптимизатору в поиске наилучшего возможного решения.

Рерайтер применяет к запросу список известных правил, Например, исключение ненужных операций, сведение подзапросов или предотвращение ненужных объединений. Если запрос всем им удовлетворяет, то он перезаписывается.  
После применения политик запрос отправляется в оптимизатор, и там начинается самое веселье.

* Оптимизатор запросов.  
  Ищет оптимальный план выполнения запросов. Все современные БД используют CBO (Cost Based Optimization), стоимостную оптимизацию. Суть её заключается в том, что для каждой операции определяется её «стоимость», а затем общая стоимость запроса уменьшается с помощью использования наиболее «дешёвых» цепочек операций.
* Исполнитель запросов. Компилирует запрос и исполняет запросы.На данном этапе план выполнения уже оптимизирован. Он перекомпилируется в исполняемый код и, если ресурсов достаточно, исполняется. Операторы, содержащиеся в плане могут обрабатываться как последовательно, так и параллельно, решение принимает исполнитель. Для получения и записи данных он взаимодействует с диспетчером данных.

Диспетчер данных (Data Manager):

Диспетчер данных сильно привязан к диспетчеру запросов: он нуждается в данных из таблиц,индексов и запрашивает их у диспетчера данных. В реляционных БД, использующих транзакционную модель возникают некоторые трудности. Первая —  это невозможность в любой момент времени получить желаемые данные, поскольку они могут кем-то обрабатываться или изменяться, а вторая связана с тем, что извлечение данных —  самая медленная операция в БД, откуда следует, что диспетчеру данных приходится прогнозировать свою работу, чтобы своевременно заполнять буфер памяти. Диспетчер данных состоит из:

* Диспетчер транзакций. Занимается обработкой данных. В его обязанности входит отслеживание, чтобы каждый запрос исполнялся с помощью собственной транзакции.
* Диспетчер кэша. Используется для отправки данных перед использованием и перед записью на диск. Увеличивает производительность системы.
* Диспетчер доступа к данным. Управляет доступом к данным в дисковой подсистеме.
* Диспетчер транзакций. В его обязанности входит отслеживание, чтобы каждый запрос исполнялся с помощью собственной транзакции. Включает в себя диспетчер блокировок, изолирующий данные для одной транзакции при конфликтной ситуации и логов. В диспетчер логов записывается информация о произведённых транзакциях, так что в случае отката или ошибки в системе БД знает, как откатить или удалить транзакцию.

### Конфликт транзакций

Возвращаясь к обсуждению стандарта ACID, заметим, что атомарность, изолированность и надёжность обеспечивают возможность безопасно осуществлять операции записи над одними и теми же данными. Если у нас есть две транзакции, производящие лишь чтение данных —  процессы идут параллельно. Дилемма возникает, когда хотя бы одна из этих транзакций изменяет данные. И поочерёдное выполнение процессов здесь неэффективно, поскольку будут использоваться ресурсы лишь одного ядра, теряется возможность масштабирования. Тогда идеальным способом решения проблемы будет мониторинг всех операций каждой транзакции; а в случае возникновения конфликта, управление очерёдностью операций внутри транзакций, то есть выполнение конфликтующих частей транзакций в определённом порядке, остальных —  параллельно. Кроме того, надо иметь в виду, что транзакции могут быть отменены. Однако просчёт оптимальных вариантов работы с данными для нескольких транзакций слишком ресурсоёмок. Потому чаще используются менее совершенные подходы, где на разрешение конфликтов тратится больше времени, чем в идеальной модели.

Одна из идей в таких подходов —  это диспетчер блокировок, блокирующий данные на время, пока транзакция с ними работает; другим транзакциям для доступа к данным приходится ждать, пока первая не снимет блокировку; доступ к тем же данным есть лишь в случае, если все работающие с этими данными транзакции лишь производят чтение. Здесь конфликтная ситуация возникает только при взаимной блокировке данных, когда одна транзакция эксклюзивно заблокировала данные1 и ожидает освобождения данных 2, и в то же время другая эксклюзивно заблокировала данные 2 и ожидает освобождения данных 1. Потому в БД, использующих диспетчер блокировок, применяется особый протокол, согласно которому транзакция делится на две фазы: фазу подъёма, когда транзакция может только применять блокировки, но не снимать их и фазу спада, когда транзакция может только снимать блокировки (с данных, которые уже обработаны и не будут обрабатываться снова), но не применять новые. Благодаря этому протоколу, снимаются  блокировки (кроме эксклюзивных, которые могут быть сняты только по завершении транзакции), больше не нужные, чтобы уменьшить время ожидания других транзакций и предотвращаются случаи, когда транзакция получают данные, модифицированные ранее запущенной транзакцией.

Ещё один способ решения проблемы конфликта транзакций — использование версионности данных. Согласно этому принципу, все транзакции могут модифицировать одни и те же данные в одно и то же время, каждая транзакция работает с собственной копией (версией) данных, и если две транзакции изменяют одни и те же данные, то принимается только одна из модификаций, другая будет отклонена, а сделавшая её транзакция будет откачена (и, может быть, перезапущена).Это позволяет увеличить производительность, но вместе с тем ставит новую проблему, возникающую при одновременной записи одних и тех же данных, а так же может давать ошибки, связанные с индексами (дубликаты в уникальном индексе, в индексе больше записей, чем строк в таблице и т.д.).

### **Язык SQL**

Поскольку в этой работе мы уделили большое внимание реляционным базам данных, предлагаем рассмотреть инструмент для взаимодействия с ними -- SQL, язык структурированных запросов, позволяющий хранить, манипулировать и извлекать данные из реляционных баз данных.

При выполнении любой SQL-команды в любой RDBMS (Relational Database Management System — система управления РБД, СУБД, например, PostgreSQL, MySQL, MSSQL, SQLite и др.) система определяет наилучший способ выполнения запроса, а движок SQL определяет способ интерпретации задачи.

Стандартными командами для взаимодействия с РБД являются CREATE, SELECT, INSERT, UPDATE, DELETE и DROP. Эти команды могут быть классифицированы следующим образом:

* DDL — язык определения данных (Data Definition Language)

| **N** | **Команда** | **Описание** |
| --- | --- | --- |
| 1 | CREATE | Создает новую таблицу, представление таблицы или другой объект в БД |
| 2 | ALTER | Модифицирует существующий в БД объект, такой как таблица |
| 3 | DROP | Удаляет существующую таблицу, представление таблицы или другой объект в БД |

* DML — язык изменения данных (Data Manipulation Language)

| **N** | **Команда** | **Описание** |
| --- | --- | --- |
| 1 | SELECT | Извлекает записи из одной или нескольких таблиц |
| 2 | INSERT | Создает записи |
| 3 | UPDATE | Модифицирует записи |
| 4 | DELETE | Удаляет записи |

* DCL — язык управления данными (Data Control Language)

| **N** | **Команда** | **Описание** |
| --- | --- | --- |
| 1 | GRANT | Наделяет пользователя правами |
| 1 | REVOKE | Отменяет права пользователя |

Теперь рассмотрим применение команд на тестовой БД.

Пусть схема БД состоит из четырёх таблиц:

|  |  |  |  |
| --- | --- | --- | --- |
| Product(maker, model, type) | PC(code, model, speed, ram, hd, cd, price) | Laptop(code, model, speed, ram, hd, price, screen) | Printer(code, model, color, type, price) |
| |  |  |  | | --- | --- | --- | | **maker** | **model** | **type** | | A | 1232 | PC | | A | 1233 | PC | | A | 1276 | Printer | | A | 1298 | Laptop | | A | 1401 | Printer | | A | 1408 | Printer | | A | 1752 | Laptop | | B | 1121 | PC | | B | 1750 | Laptop | | C | 1321 | Laptop | | D | 1288 | Printer | | D | 1433 | Printer | | E | 1260 | PC | | E | 1434 | Printer | | E | 2112 | PC | | E | 2113 | PC | | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **code** | **model** | **speed** | **ram** | **hd** | **cd** | **price** | | 1 | 1232 | 500 | 64 | 5.0 | 12x | 600.0000 | | 10 | 1260 | 500 | 32 | 10.0 | 12x | 350.0000 | | 11 | 1233 | 900 | 128 | 40.0 | 40x | 980.0000 | | 12 | 1233 | 800 | 128 | 20.0 | 50x | 970.0000 | | 2 | 1121 | 750 | 128 | 14.0 | 40x | 850.0000 | | 3 | 1233 | 500 | 64 | 5.0 | 12x | 600.0000 | | 4 | 1121 | 600 | 128 | 14.0 | 40x | 850.0000 | | 5 | 1121 | 600 | 128 | 8.0 | 40x | 850.0000 | | 6 | 1233 | 750 | 128 | 20.0 | 50x | 950.0000 | | 7 | 1232 | 500 | 32 | 10.0 | 12x | 400.0000 | | 8 | 1232 | 450 | 64 | 8.0 | 24x | 350.0000 | | 9 | 1232 | 450 | 32 | 10.0 | 24x | 350.0000 | | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | **code** | **model** | **speed** | **ram** | **hd** | **price** | **screen** | | 1 | 1298 | 350 | 32 | 4.0 | 700.0000 | 11 | | 2 | 1321 | 500 | 64 | 8.0 | 970.0000 | 12 | | 3 | 1750 | 750 | 128 | 12.0 | 1200.0000 | 14 | | 4 | 1298 | 600 | 64 | 10.0 | 1050.0000 | 15 | | 5 | 1752 | 750 | 128 | 10.0 | 1150.0000 | 14 | | 6 | 1298 | 450 | 64 | 10.0 | 950.0000 | 12 | | |  |  |  |  | | --- | --- | --- | --- | | **code** | **model** | **color** | **price** | | 1 | 1276 | n | 400.0000 | | 2 | 1433 | y | 270.0000 | | 3 | 1434 | y | 290.0000 | | 4 | 1401 | n | 150.0000 | | 5 | 1408 | n | 270.0000 | | 6 | 1288 | n | 400.0000 | |

Таблица Product представляет производителя (maker), номер модели (model) и тип ('PC' - ПК, 'Laptop' - ноутбук или 'Printer' - принтер). Предполагается, что номера моделей в таблице Product уникальны для всех производителей и типов продуктов. В таблице PC для каждого ПК, однозначно определяемого уникальным кодом – code, указаны модель – model (внешний ключ к таблице Product), скорость - speed (процессора в мегагерцах), объем памяти - ram (в мегабайтах), размер диска - hd (в гигабайтах), скорость считывающего устройства - cd (например, '4x') и цена - price. Таблица Laptop аналогична таблице РС за исключением того, что вместо скорости CD содержит размер экрана -screen (в дюймах). В таблице Printer для каждой модели принтера указывается, является ли он цветным - color ('y', если цветной), тип принтера - type (лазерный – 'Laser', струйный – 'Jet' или матричный – 'Matrix') и цена - price.

Выполним несколько практических задач по извлечению данных с помощью языка SQL.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Задание | Запрос | Результат |
| 1 | Для каждого производителя, выпускающего ноутбуки c объёмом жесткого диска не менее 10 Гбайт, найти скорости таких ноутбуков. Вывод: производитель, скорость. | SELECT p.maker, l.speed  FROM Product p JOIN Laptop l  ON p.model = l.model  WHERE l.hd >= 10  GROUP BY p.maker, l.speed; | |  |  | | --- | --- | | maker | speed | | A | 450 | | A | 600 | | A | 750 | | B | 750 | |
| 2 | Найдите номера моделей и цены всех имеющихся в продаже продуктов (любого типа) производителя B (латинская буква). | SELECT Product.model, price  FROM Product JOIN PC on Product.model = PC.model  WHERE Product.maker = 'B'  UNION  SELECT Product.model, price  FROM Product JOIN Laptop ON Product.model = Laptop.model  WHERE Product.maker = 'B'  UNION  SELECT Product.model, price  FROM Product JOIN Printer ON Product.model = Printer.model  WHERE Product.maker = 'B' | |  |  | | --- | --- | | model | price | | 1121 | 850.0000 | | 1750 | 1200.0000 | |
| 3 | Найдите производителя, выпускающего ПК, но не ноутбуки. | SELECT DISTINCT maker  FROM Product  WHERE type = 'PC'  EXCEPT  SELECT maker  FROM Product  WHERE type = 'Laptop' | |  | | --- | | maker | | E | |
| 4 | Найдите пары моделей PC, имеющих одинаковые скорость и RAM. В результате каждая пара указывается только один раз, т.е. (i,j), но не (j,i), Порядок вывода: модель с большим номером, модель с меньшим номером, скорость и RAM. | SELECT DISTINCT P.model, L.model, P.speed, P.ram  FROM PC P JOIN PC L ON P.model > L.model  WHERE P.speed = L.speed AND P.ram = L.ram | |  |  |  |  | | --- | --- | --- | --- | | model | model | speed | ram | | 1233 | 1121 | 750 | 128 | | 1233 | 1232 | 500 | 64 | | 1260 | 1232 | 500 | 32 | |
| 5 | Найдите модели ноутбуков, скорость которых меньше скорости каждого из ПК. Вывести: type, model, speed | SELECT DISTINCT type, Laptop.model, speed  FROM Laptop JOIN Product ON Laptop.model = Product.model  WHERE speed < (SELECT MIN(speed) from PC) | |  |  |  | | --- | --- | --- | | type | model | speed | | Laptop | 1298 | 350 | |
| 6 | Найдите производителей самых дешёвых цветных принтеров. Вывести: maker, price | SELECT DISTINCT Product.maker, Printer.price  FROM Product join Printer ON Product.model = Printer.model  WHERE price = (SELECT MIN(price) FROM Printer WHERE color = 'y') AND color = 'y' | |  |  | | --- | --- | | maker | price | | D | 270.0000 | |
| 7 | Перечислите номера моделей любых типов, имеющих самую высокую цену по всей имеющейся в базе данных продукции. | WITH MAX  AS (  SELECT model, price  FROM PC  UNION  SELECT model, price  FROM Laptop  UNION  SELECT model, price  FROM printer  )  SELECT model FROM MAX  WHERE price = (  SELECT MAX(price)  FROM MAX  ) | |  | | --- | | model | | 1750 | |
| 8 | Найдите производителей принтеров, которые производят ПК с наименьшим объемом RAM и с самым быстрым процессором среди всех ПК, имеющих наименьший объем RAM. Вывести: Maker | SELECT DISTINCT maker  FROM Product JOIN PC  ON Product.model = PC.model  WHERE ram = (  SELECT MIN(ram)  FROM PC  )  AND speed = (  SELECT MAX(speed)  FROM PC  WHERE ram = (  SELECT MIN(ram)  FROM PC)  )  AND maker IN (  SELECT maker  FROM Product  WHERE TYPE='Printer'  ) | |  | | --- | | maker | | A | | E | |
| 9 | Найдите среднюю цену ПК и ноутбуков, выпущенных производителем A (латинская буква). Вывести: одна общая средняя цена. | SELECT AVG(price)  FROM (  SELECT price  FROM PC JOIN Product ON PC.model = Product.model  WHERE maker = 'A'  UNION ALL  SELECT price  FROM Laptop JOIN Product ON Laptop.model = Product.model  WHERE maker = 'A') AS priceList | |  | | --- | |  | | 754.1666 | |

### Заключение

Наш быстроразвивающийся мир невозможно представить без баз данных. В сравнении с любым бумажным носителем они выгодно отличаются компактностью, скоростью обработки информации, низкими трудозатратами на обслуживание и широкой применимостью. Благодаря базам данных люди по всему миру могут получать информационные ресурсы в нужный им момент, и спектр применения этих хранилищ знаний действительно огромен.

Повсеместная потребность в сборе, хранении, аналитике, обработке информации делает специалистов по базам данных востребованными. Это направление активно развивается, ищет и находит новые технологии, совершенствует имеющиеся.

В этом реферате мы рассмотрели лишь малую часть того, что есть базы данных, но провели их классификацию, узнали основные принципы работы, структуру, проблемы, с которыми сталкиваются разработчики, и методы их решения, а также выполнили несколько практических задач по работе с базами данных на языке SQL.

### Список литературы

https://en.sodiummedia.com/4111010-data-models-features-classification-and-description модели данных

https://habr.com/ru/post/152477/ нереляционные БД

https://habr.com/ru/company/vk/blog/266811/ реляционные БД

[https://cstack.github.io/db\_tutorial/parts/part1.html](https://cstack.github.io/db_tutorial/parts/part1.html" \t "_blank),

https://dane-bulat.medium.com/working-with-databases-in-c-creating-a-social-following-app-56a813778fb7 основы и структура БД

https://sql-ex.ru/?Lang=1 задания на извлечение информации из базы данных