Содержание

1 Лекция 1. Введение в реляционные базы данных

1.1 Понятие базы данных и СУБД

Definition 1.1. Данные

Данные – факты, текст, графики, картинки, звуки, аналоговые или цифровые видеосегменты, представленные в форме, пригодной для хранения, передачи и обработки

Notation 1.1. Задачи технологий работы с данными

- Загрузить
  - Получить
  - Передать между системами
  - Собрать в одном месте
- Сохранить
  - Обеспечить эффективное безопасное хранение
  - Предоставить доступ
  - Обеспечить быстрый поиск
- Обработать
  - Объединить в одной структуре
  - Рассчитать показатели
  - Обработать модель
- Принять решение
  - Отчеты
  - Дашборды
  - Визуализация
  - Предписание в операционном процессе

Notation 1.2. Что должны обеспечивать системы обработки данных?

- Поддержку необходимых требований к данными
  - Сохранность и достоверность данных в соответствии со спецификой предметной области
  - Защиту от искажения, уничтожения данных и несанкционированного доступа
  - Простоту и легкость использования данных
  - Требуемую скорость доступа к данным
- Независимость прикладных программ от данных
  - Возможность использования одних и тех же данных различными приложениями
  - Изменение логического представления данных (добавление/удаление новых элементов данных) не должно приводить к изменению прикладных программ обработки
  - Изменение параметров физической организации данных (характеристик носителя, длины блока, и др.) или методов доступа к ним не должны приводить к изменению прикладных программ

Definition 1.2. Независимость (от) данных

- **Логическая**
  Изменение логического представления данных (добавление/удаление новых элементов данных, разделение на несколько логических сегментов, изменение порядка хранения) не должно приводить к изменению прикладных программ обработки
- **Физическая**
  Изменение параметров физической организации данных (характеристик носителя, длины блока, и др.) или методов доступа к ним не должны приводить к изменению логической структуры данных или прикладных программ их обработки

Notation 1.3. Концепции обработки данных

- **Файловая обработка данных**
  До середины 60-х годов прошлого века – основная концепция построения программного обеспечения обработки данных
- **Базы данных**
  Независимость прикладных программ от данных
- **Объектно-ориентированные базы данных**
  Независимость прикладных программ от данных вместе с процедурами их обработки (объектно-ориентированный подход в программировании)

Remark 1.1. Файловая обработка данных

- Совмещение логического и физического представления данных
  - Физически данные ИС размещаются в файлах операционной системы
  - Прикладные программы работают напрямую с файлами
  - Структура файла зависит от требований конкретной прикладной программы, с ним работающей, и определяется разработчиком приложения
- Зависимость программ обработки от организации данных
  - Возможная избыточность и противоречивость данных
  - Для каждой задачи – свой алгоритм получения данных
- Разграничение доступа и защита данных – на уровне средств ОС

Она не обеспечивает:

- Поддержание логически согласованного набора файлов
- Единого языка манипулирования данными
- Восстановление информации после разного рода сбоев
- Реально параллельной работы нескольких пользователей

Remark 1.2. Базы данных

- Разделение логического и физического представления данных
  - Физически данные как правило размещаются в файлах операционной системы
  - Разрабатывается определённая логическая структура данных, с которой «общаются» прикладные программы
  - Логическая структура данных основана на физической, но не меняется при изменении последней
- Независимость программ обработки от организации данных
  - Сокращение избыточности и противоречивости данных
  - Единые языки для получения и изменения данных
  - Дополнительные средства разграничения доступа и защиты данных, учитывающие их логическую структуру

Remark 1.3. Объектно-ориентированные БД

- Разделение логического и физического представления данных
  - Физические данные как правило размещаются в файлах операционной системы
  - На логическом уровне данные представляют собой объекты
    - Данные $\rightarrow$ свойства объекта (статическая часть)
    - Обработка данных $\rightarrow$ методы объекта (динамическая часть)
- Независимость прикладных программ от объектов
- Концепция повторного использования программного кода
  - Разные прикладные программы используют унифицированные методы обработки одних и тех же данных

1.2 Базы данных и СУБД

Definition 1.3. База данных

База данных – логически структурированная совокупность постоянно хранимых в памяти компьютера данных, характеризующих актуальное состояние некоторой предметной области и используемых прикладными программными системами какого-либо предприятия
База данных (БД) – совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных (ГОСТ)

Notation 1.4. Основные характеристики БД

- Компьютерная система – БД хранится и обрабатываемя в вычислительной системе
- Содержит структурированную информацию – данные в БД логически структурированы (систематизированы) с целью обеспечения возможности их эффективного поиска и обработки в вычислительной системе
  - Структурированность БД оценивается не на уровне физического хранения, а на уровне некоторой логической модели данных
- Поддерживает определенный набор операций над данными
  - Структурированность определяет семантику и допустимые операции

Definition 1.4. Система баз данных (банк данных)

База данных является составной частью системы баз данных
Система базы данных – компьютеризированная система обработки данных, хранящихся в БД
Основные функции системы баз данных:

- Добавление новых структур данных (таблиц) в базу данных
- Изменение существующих структур данных (таблиц)
- Добавление новых элементов данных (записей в таблицы)
- Выборка необходимых элементов данных (записей из таблиц)
- Обновление элементов данных (записей в таблицах)
- Удаление элементов данных (записей из таблиц)
- Удаление структур данных (таблиц) из базы данных

Definition 1.5. Системы управления базами данных

СУБД – совокупность программных средств, предназначенная для модификации и извлечения из БД необходимых пользователю (прикладной программе) данных, а также для создания БД, поддержания их в работоспособном состоянии, обеспечения безопасности БД и решения других задач администрирования
Цель: Обеспечивать совместное безопасное использование данных различными приложениями и пользователями
Основные функции СУБД:

- Обеспечение физической и логической независимости данных
- Поддержка связи между логической и физической организацией данных
- Предоставление интерфейса доступа к данным пользователям и прикладным программам
- Обеспечение целостности и непротиворечивости данных при совместной работе нескольких пользователей
- Предоставление механизмов защиты данных
- Предоставление механизмов восстановления данных

Notation 1.5. Обеспечение логической и физической независимости данных

СУБД обеспечивает необходимые структуры внешней памяти как для хранения пользовательских данных БД, так и для служебных целей
СУБД скрывает от пользователей как используется файловая система и как организованы файлы
Для связи логической и физической структур данных СУБД использует служебную (мета) информация, хранящуюся в словаре данных

Notation 1.6. Управление буферами оперативной памяти

Для увеличения скорости обработки данных используется буферизация данных в оперативной памяти
СУБД поддерживает собственный набор буферов оперативной памяти с собственным алгоритмом замены буферов

Notation 1.7. Предоставление интерфейса доступа к данным

Для работы с базами данных СУБД предоставляет поддержку специальных языков, называемых языками баз данных

- Язык описания данных (DDL) – позволяет создавать и изменять структуру объектов базы данных
- Язык манипулирования данными (DML) – позволяет заносить данные в БД, удалять, модифицировать или выбирать существующие данные

Использование данных языков позволяет пользователям сохранять, извлекать и обновлять данные в БД без необходимости понимания физической реализации системы

Notation 1.8. Обеспечение целостности

Целостность БД – свойство БД, означающее, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную область информация
Целостность данных предполагает:

- Отсутствие неточно введенных данных или двух одинаковых записей об одном и том же факте
- Защиту от ошибок при обновлении данных в БД
- Невозможность удаления (или каскадное удаление) записей, которые связаны с другими записями
- Неискажение данных при работе в многопользовательском режиме
- Сохранность данных при сбоях техники (восстановление данных)

СУБД должна обладать инструментами контроля за тем, чтобы данные и их изменения соответствовали заданным правилам

Notation 1.9. Управление параллельной работой пользователей

Для управления параллельной работой пользователей и поддержки целостности данных в СУБД реализован механизм изоляции транзакций

Definition 1.6. Транзакция

Транзакция – некоторая неделимая последовательность операций над данными БД, которая отслеживается СУБД от начала и до завершения
Если по каким-либо причинам (сбои и отказы оборудования, ошибки в программном обеспечении) транзакция остается незавершенной или при выполнении транзакции нарушается целостность данных, то она отменяется

Definition 1.7. Журнализация

СУБД должна иметь возможность восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя
Для восстановления БД нужно располагать некоторой дополнительной информацией, которая должна храниться особо надежно
Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД – журнала транзакций

Notation 1.10. Обеспечение безопасности БД

СУБД должна иметь механизм, гарантирующий возможность доступа к базе данных только санкционированных пользователей
Термин безопасность относится к защите БД от преднамеренного или случайного несанкционированного доступа
Защита данных от несанкционированного доступа может достигаться:

- Созданием ролей и ввдеением системы паролей
- Настройкой разрешений на доступ к данным и выполнение операций с данными
- Шифрованием соединения с прикладными программами
- Шифрованием данных

Notation 1.11. Выбор СУБД

Выбор СУБД – важный шаг при создании ИС, влияющий на эффективность, как проектирования, так и функционирования системы

Учитывая тенденции развития ИС, СУБД должна отвечать следующим требованиям:

- Обеспечивать работу в гетерогенной сетевой среде, включая возможность эффективной работы в Интернете
- Легко переноситься с платформы на платформу
- Обеспечивать работу с большими объемами разнотипных данных
- Быть надежной и эффективной

1.3 Принципы организации базы данных

Notation 1.12. Требования к БД и СУБД

- Высокое быстродействие (малое время отклика на запрос)
- Простота обновления данных
- Защита данных от преднамеренного или непреднамеренного нарушения секретности, искажения или разрушения
- Поддержка целостности данных
- Независимость данных – возможность изменения логической и физической структуры БД без изменения предсталвений пользователей
- Совместное использование данных многими пользователями
- Стандартизация построения и эксплуатации БД
- Адекватность отображения данных соответствующей предметной области

Notation 1.13. Принципы организации БД

- Разделение различных видов данных
  - Данные пользовательские (приложений)
  - Вспомогательные данные (индексы)
  - Метаданные (словарь данных БД)
  - Служебная информация
- Позволяет повысить безопасность данных и быстродействие работы с ними
- Проектирование логической структуры данных
  Выбор модели данных (в зависимости от требований доступа к данным)
- Позволяет обеспечить:
  - Логическую и физическую независимость данных
  - Адекватность отображения данных соответствующей предметной области
  - Удобство работы и гибкость
- Определение ограничений данных
  - Уменьшение ошибок ввода/изменения данных
  - Зависимость от семантики данных
- Позволяет поддержать целостность данных

1.4 Реляционная модель данных

Definition 1.8. Модель данных

Модель данных – формальное описание представления и обработки данных в системе управления базами данных

Notation 1.14. Компоненты реляционной модели данных

- Структурный аспект
  - Какие структуры данных рассматриваются реляционной моделью
  - Постулируется, что единственной структурой данных, используемой в реляционной модели, являются нормализованные n-арные отношения
- Аспект целостности
  - Ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных базах данных
  - Целостность сущностей реального мира
  - Ссылочная целостность
- Аспект обработки
  - Способы манипулирования реляционными данными
  - Реляционная алгебра – базируется на теории множеств
  - Реляционное исчисление – базируется на логическом аппарате исчисления предикатов первого порядка

1.5 Структурный аспект реляционной модели

Definition 1.9. Отношение

Отношение (relation) – класс объектов реального мира, каждый из которых должен быть уникально идентифицирован

- Подмножество R декартового произведения множеств элементов доменов: (не обязательно различных)
- Или множество кортежей, соответствующих одной схеме отношения
- Мощность отношения – количество кортежей в отношении
- На бытовом уровне – тело таблицы

Definition 1.10. Схема отношения

- Именованное конечное множество пар вида {Имя_атрибута: Имя_домена}
- Степень или -арность (схемы) отношения – количество атрибутов
- На бытовом уровне – заголовок таблицы

Definition 1.11. Атрибут

Атрибут (поле, столбец) – характеристика объекта, принимающая значения определенного типа данных

- Подмножество домена (атрибут A_n определен на домене D_n, который содержит множество возможных значений атрибута)
  $A_1(D_1), A_2(D_2), \dots A_n(D_n)$
- На бытовом уровне – столбец таблицы
- Имена атрибутов должны быть уникальны в пределах отношения

Definition 1.12. Тип данных

Тип данных – реляционная модель данных допускает использование только простых типов

- Соответствует понятию типа данных в языках программирования
- Реляционная модель данных допускает использование только простых типов (логические, символьные, числовые ...), т.к. в реляционных операциях не должна учитываться внутренняя структура данных

Definition 1.13. Домен

Домен – произвольное логическое выражение (опционально), определяющее набор допустимых значений атрибута

- Базовый тип данных + произвольное логическое выражение (опционально)
- Если вычисление заданного логического выражения для элемента данных заданного типа дает результат "истина" , то он является элементом домена
- Домены ограничивают сравнения: некорректно, с логической точки зрения, сравнивать значения из различных доменов, даже если они имеют одинаковый тип данных (например, возраст сотрудника и количество его детей)

Definition 1.14. Кортеж

Кортеж (запись, строка) – набор связанных значений атрибутов, относящихся к одному объекту (сущности)

- Множества пар вида { Имя_атрибута: Значение_атрибута }
- На бытовом уровне – строка таблицы
- Значение атрибута должно быть в пределах заданного домена
- Каждый кортеж отношения должен быть уникально идентифицирован

Definition 1.15. Схема БД

Схема БД (в структурном смысле) – набор именованных схем отношений

Definition 1.16. Реляционная база данных

Реляционная база данных – набор отношений, имена которых совпадают с именами схем отношений в схеме БД

Notation 1.15. Свойства отношений

- В отношении нет кортежей-дубликатов
  Т.к. отношение – это множество кортежей, а каждое множество (в классической теории множеств) состоит из различных элементов
  Во многих РСУБД может нарушаться для отношений, являющихся результатами запросов
- Кортежи не упорядочены
  Множество не упорядочено
- Атрибуты не упорядочены
  Каждый атрибут имеет уникальное имя в пределах отношения, поэтому порядок атрибутов не имеет значения
- Значение атрибута должно быть атомарным (неразделяемым на несколько значений)
  В современных РСУБД в ячейки таблиц можно поместить что угодно – массивы, структуры и даже другие таблицы
- Домены ограничивают сравнения
  Некорректно, с логической точки зрения, сравнивать значения из различных доменов, даже если они имеют одинаковый тип данных

1.6 Аспект целостности реляционной модели

Definition 1.17. Целостность базы данных

Целостность базы данных (database integrity) – соответствие имеющейся в базе данных информации ее внутренней логике, структуре и всем явно заданным правилам
Каждое такое правило, налагающее ограничение на возможное состояние базы данных, называется ограничением целостности (integrity constraint)
Задача аналитика и проектировщика БД – возможно более полно выявить все имеющиеся ограничения целостности и задать их в базе данных
СУБД может (и должна) контролировать целостность БД

Notation 1.16. Сущностная целостность

Каждый кортеж отношения должен быть уникально идентфиицирован по значениям его атрибутов
Потенциальный ключ обладает следующими свойствами:

- Свойством уникальности – в отношении не моежт быть двух различных кортежей с одинаковым значением потенциального ключа
- Свойством неизбыточности – никакое подмножество в потенциальном ключе не обладает свойством уникальности, т.к. если из потенциального ключа убрать любой атрибут, он утратит свойство уникальности

Отношение может иметь несколько потенциальных ключей:

- Один из потенциальных ключей объявляется первичным – Primary Key, а остальные – альтернативными – Alternate Key
- С точки зрения реляционной модели данных, нет оснований выделять таким образом один из потенциальных ключей

Notation 1.17. Правила для поддержки сущностной целостности

Потенциальный ключ может быть:

- Простым – состоит из одного атрибута
- Составным – состоит из нескольких атрибутов

Потенциальные ключи фактически являются идентификаторами. Если бы идентификаторы могли содержать NULL значения, невозможно было бы дать ответ "да" или "нет" на вопрос, совпдаают ли два идентификатора
Это определяет следующее правило: значения атрибутов, входящих в состав некоторого потенциального ключа не могут быть NULL (во многих СУБД выполняется только для первичного ключа)

Definition 1.18. NULL

Для того чтобы обойти проблему неполных или неизвестных данных, каждый тип данных в БД может быть дополнен NULL
NULL – это не значение, а некий маркер, показывающий, что значение неизвестно
В ситуации, когда возможно появление неизвестных или неполных данных, разработчик имеет на выбор два варианта:

- Ограничиться использованием обычных типов данных и не использовать NULL, а вместо неизвестных данных вводить либо нулевые значения, либо значения специального вида
- Использовать NULL вместо неизвестных данных

Наличие NULL приводит к использованию трехзначной логики:

- Три возможных значений выражений: TRUE (T), FALSE (F), UNKNOWN (U)
- NULL = NULL → U (NOT NULL = NULL → U)
- NULL !+ NULL → U (NOT NULL != NULL → U)
- F OR NULL → U (T OR NULL → T)
- T AND NULL → U (F AND NULL → F)

Definition 1.19. Внешние ключи

Различные объекты предметной области, информация о которых хранится в базе данных, взаимосвязаны друг с другом
Для реализации взаимосвязи между родительским и дочерним отношениями в реляционных БД используются внешние ключи – foreign key (FK)

Notation 1.18. Требования к FK

Подмножество атрибутов FK отношения R будем называть внешним ключом, если

- Существует отношения S (R и S не обязательно различны) с потенциальным ключом K
- Каждое значение FK в отношении R всегда совпадает со значением K для некоторого кортежа из S, либо является NULL

Remark 1.4. Замечания относительно FK

- Отноешние S называется родиетльским отношением, отношение R называется дочерним отношением
- FK, также как и потенциальный, может быть простым и Составным
- FK должен быть определен на тех же доменах, что и соответствующих потенциальный ключ родительского отношения
- FK, как правило, не обладает свойством уникальности (тип связи – один ко многим)
- Для FK не требуется, чтобы он был компонентом некоторого потенциального ключа
- NULL для значений атрибутов FK допустимы только в том случае, когда атрибуты FK не входят в состав никакого потенциального ключа

Definition 1.20. Связь

Связь – ассоциирование двух или более сущностей (или копий одной и той же сущности)
Одно из основным требований к организации БД – это обеспечение возможности поиска одних сущностей по значениям других, для чего необходимо установить между ними определенные связи
Типы связей:

- Связь 1:1. Один экземпляр сущности одного класса связан с одним экземпляром сущности другого класса
- Связь 1:M. Один экземпляр сущности одного класса связан со многими экземплярами сущности другого класса
- Связь M:N. Несколько экземпляров сущности одного класса связаны с несколькими экземплярами сущности другого класса

Notation 1.19. Допустимая кратность связей

В реляционных БД допустимыми являются связи типа 1:M и 1:1 (значения внешнего ключа – уникальны)
Механизм реализации допустимых взаимосвязей состоит в том, что на дочернее отношение добавляются атрибуты, являющиеся ссылками на ключевые атрибуты родительского отношения
Невозможно ссылаться на несуществующие объекты $\rightarrow$ значения атрибута внешнего ключа дочернего отношения должны иметь соответствие среди значений атрибутов отношения потенциального ключа родительского
Взаимосвязи типа M:N реализуются использованием нескольких взаимосвязей типа 1:M

1.7 Знакомство с PostgreSQL (реклама)

Definition 1.21. PostgreSQL

PostgreSQL – это открытая, BSD-лицензированная система управления объектно-ориентированными реляционными базами данных

Notation 1.20. История PostgreSQL

- 1986 – старт проекта PostgreSQL на факультете компьютерных наук Калифорнийсокго университета в Беркли
  Первоначальное название проекта – POSTGRES (развитие старой БД Ingres)
- 1996 – проект POSTGRES переименован в PostgreSQL, для отражения поддержки SQL
  Global Development Group PostgreSQL, специализированное сообщество участников, продолжает выпускать релизы проекта с открытым исходным кодом
- Первоначально PostgreSQL был разарботан для работы на UNIX-подобных платформах
  Сейчас PostgreSQL поддерживает различны платформы, такие как Windows, macOS и Solaris

Notation 1.21. Преимущества и особенности СУБД PostgreSQL

- Надежность
- Производительность
- Расширяемость
- Поддержка SQL
- Поддержка многочисленных типов данных

Notation 1.22. Расширяемость

PostgreSQL спроектирован с рассчетом на расширяемость
Прикладные программисты могут:

- Создавать собственные типы данных на основе уже имеющихся (составные типы, диапазоны, масисвы, перечисления)
- Писать хранимые процедуры и функции для обработки данных в БД (в том числе триггеры)
- Писать расширения (языке программирования Си), которые добавляют необходимый функционал и, обычно, могут подключаться даже к работающему серверу

Если вам не нравится какая-либо часть системы, вы всегда можете разработать собственный плагин

Remark 1.5. Важный факт

Подавляющее большинство СУБД:

- Представляет собой сервис (демон в *nix-системах), который взаимодействует с внешним миром по специальным протоколам (чаще всего, построенным поверх TCP/IP)
- Не имеет никакого человеческого интерфейса
- Общение осуществляется на специализированном языке через специальные библиотеки

MySQL Workbench, Microsoft SQL Serber Management Studio, Oracle SQL Developer и им подобные – это не СУБД, это лишь клиентское программное обеспечение, позволяющее нам взаимодействовать с СУБД

Notation 1.23. Упрощенная архитектура PostgreSQL

PostgreSQL – это СУБД клиент-серверного типа с многопроцессной архитектурой, работающая на одном хосте

Сбой в одном из процессов не повлияет на остальные и система продолжит функционировать

Набор нескольких процессов, совместно управляющих одним кластером БД, называется "сервером PostgreSQL"

Один сервер PostgreSQL может управлять несколькими конкурентными клиенсткими подключениями

Notation 1.24. Основные процессы

- FrontEnd процессы – клиентские приложения:
  - Используют PostgreSQL в качестве менеджера баз данных
  - Соединение может происходить через TCP/IP или локальные сокеты
- Демон postres (postmaster) – это основной процесс PostgreSQL:
  - Прослушивание через порт/сокет входящих клиенстких подключений
  - Создание BackEnd процессов и выделение им ресурсов
- BackEnd процессы:
  - Аутентификация клиенстких подключений
  - Управление запросами и отправка результатов клиенстким приложениям
  - Выполнение внутренних задач (служебные процессы)

Notation 1.25. Процесс взаимодействия с БД

1. Клиент (FrontEnd)
   (a) Используя язык БД формулирует требования к результату
   (b) Опирается на знание логической структуры БД
2. СУБД (postmaster и BackEnd)
   (a) Получает запрос от клиента
   (b) Анализирует разрешения
   (c) Анализирует и оптимизирует запрос
   (d) Создает план выполнения запроса, опираясь на знание физической структуры данных
   (e) Выполняет запрос

Notation 1.26. Взаимодействие с БД

Для работы с реляционной СУБД существует два основных подхода:

- Работа с библиотекой, которая соответствует конкретной СУБД и позволяет использовать для работы с БД язык БД
- Работа с ORM, которая использует объектно-ориентированный подход для работы с БД и автоматически генерирует код на языке БД

Notation 1.27. Подключение к СУБД с использованием клиентской библиотеки

Строка подключения включает:

- Имя сервера БД (или IP адрес и порт)
- Имя базы данных
- Учетную запись пользователям
- И другие параметры, необходимые для установки исходного подключения

1.8 SQL и NoSQL

Notation 1.28. Модели данных

- Иерархическая (файловая система)
- Сетевая (социальные сети)
- Документ-ориентированная (системах управления содержимым)
- Реляционная (банковские системы)
- Объектно-ориентированная (естественное отображение ООП кода на БД, уменьшающее impedance mismatch)
- Многомерная (аналитические системы)

Notation 1.29. Преимущества РБД

- Совместное использование данных
  - Улучшенное управление паралелльной работой
  - Повышенная безопасность
  - Контроль доступа к данным
- Поддержка целостности данных
  - Контроль за избыточностью данных и их непротиворечивостью
  - Обеспечение поддержки бизнес-правил
- Эффективное управление
  - Упрощение сопровождения системы за счет неазависимости от данных
  - Эффективное резервное копирование и восстановление данных
- Применение стандартов

Notation 1.30. Недостатки РБД

- Сложность
  Затраты на преобразование данных на входе и выходе
- Уязвимость
  Централизация ресурсов повышает уязвимость системы
- Высокие финансовые затраты
  - Стоимость СУБД
  - Стоимость сопровождения
  - Дополнительные затраты на аппаратное обеспечение

Definition 1.22. Базы данных NoSQL

Базы данных NoSQL хорошо подходят для приложений, которые должны быстро, с низкой временной задержкой (low latency) обрабатывать большой объем данных с разной структурой

- Гибкость. Благодаря использованию гибких моделей данных БД NoSQL хорошо подходят для частично структурированных и неструктурированных данных. Эффективность работы с разреженными данными
- Масштабируемость. БД NoSQL рассчитаны на масштабирование с использованием распределенных кластеров аппаратного обеспечения. Широко используются в облачных решениях в качестве полностью управляемых сервисов
- Высокая производительность. БД NoSQL оптмизированы для конкретных моделей данных и шаблонов доступа, что позволяет достичь более высокой производительности по сравнению с реляционными базами данных
- Широкие функциональные возможности. БД NoSQL предоставляют API и типы данных с широкой функциональностью, которые специально разработаны для соответствующих моделей данных

Definition 1.23. Нереляционные БД

- Хранилища ключей и значений
  - Поддерживают высокую разделяемость и обеспечивают беспрецендентное горизонтальное масштабирование
  - Игровые, рекламные прилоежния и приложения IoT (Amazon DynamoDB, Redis, Riak)
- Колоночные
  - Данные хранятся не по строкам, а по столбцам
  - Хорошо подходят для BigData (Hbase, Clickhouse, Vertica)
- Документоориентированные
  - Хранение коллекций документов с произвольным набором атрибутов (полей)
  - Каталоги, пользовательские профили и системы управления контентом, где каждый документ уникален и изменяется со временем (CouchDB, Couchbase, MongoDB)
- Графовые
  - Упор на установление произвольных связей между данными
  - Социальные сети, сервисы рекомендаций, системы выявления мошенничества и графы знаний (OrientDB, Neo4j)

Параметр	Реляционные (SQL)	NoSQL
Подходящие рабочие нагрузки	OLTP и OLAP	Приложения с низкой задержкой доступа к данным
Модель данных	Нормализованная реляционная модель обеспечивает целостность ссылочных данных в отношениях между таблицами	Предоставляют разнообразные модели данных, оптимизированные для высокой производительности и масштабируемости
Струткура	Жесткая схема: таблицы (строки, столбцы)	Гибкие модели: документы, ключ-значение, графы, колоночные
Изменение схемы	Требует ALTER TABLE, миграции	Гибкость (документы без фиксированной схемы)
Типы данных	Строгая типизация	Диинамические (JSON, BLOB и другие)
Связи	JOIN, внешние ключи, ACID-транзакции	Часто денормализация, ссылки или вложенные данные
Производительность	Зависит от дисковой подсистемы. Требуется оптимизация запросов, индексов и струткур таблицы	Зависит от размера кластера базового аппаратного обеспечения, задержки сети и вызывающего приложения
Чтение/запись	Быстрые сложные запросы (OLTP), но JOIN могут замедлять	Высокая скорость для простых операция (ключ-значение)
Оптимизация	Индексы, нормализация	Денормализация, распределенные вычисления
Масштабирование	Масштабируются путем увеличения вычислительных возможностей аппаратного обеспечения или добавления отдельных копий для рабочих нагрузок чтения	Поддерживают высокую разделяемость благодаря шаблонам доступа с возможностью масштабирования на основе распределенной архитектуры
Горизонтальное	Сложно (шардинг требует усилий)	Оптмиизировано (например, Cassandra, DynamoDB)
Вертикальное	Стандартный подход (увеличение сервера)	Возможно, но реже используется

1.9 Классификация баз данных

Notation 1.31. Классификация БД

- Классификация БД по характеру организации данных
  - Неструктурированные
    БД, хранят данные в виде обычного текста или гипертекстовой разметки
    * рпоще зафиксировать (как есть)
    * Очень трудно искать конкретные данные, поскольку они не структурированы
    * Очень трудно анализировать, поскольку данных как правило качественные (семантические)
  - Структурированные
    БД, хранящие данные в организованном виде в отформатированном хранилище
    * Требуют предварительного проектирования и описания структуры БД
    * Только после этого БД такого типа могут быть заполнены данными
    * Очень простой поиск и нахождение данных в базе данных или наборе данных
    * Очень легко анализировать данные, поскольку они как правило количественные
- Классификация БД по характеру хранимой информации
  - Документальные
    * Предназначены для хранения слабо структурированных данных. Единицей хранения является документ, заданный конечным (но не фиксированным) набором полей в общем случае произвольной длины. Значение поля может иметь сложную структуру и зависеть от контекста использования
    * Пользователю в ответ на его запрос выдается либо ссылка на документ, либо сам документ, в котором он может найти интересующую информацию
    * Использование: гипертекстовые документы в сети Интернет
    * Информационно-справочные или информационно-поисковые системы
  - Фактографические
    * Ориентированы на хранение хорошо структурированных данных. Единицей информации служит описание факта конечным, четко определенным множеством свойств. Каждое свойство факта (объекта) имеет атомарное значение, которое не зависит от контекста использования
    * Использование: БД оперативной обработки транзакций (OLTP) – операционные БД, БД оперативной аналитической обработки (OLAP) – хранилища данных (Data Warehouse)
- Классификация БД и СУБД по структуре организации данных
  Структурированные БД различаются по типу используемой модели представления данных
  - Сетевые
  - Иерархические
  - Реляционные
  - Многомерные
  - Объектно-ориентированные

Definition 1.24. Модель представления данных

Модель данных – интегрированный набор понятий для описания и обработки данных, связей между ними и ограничений, накладываемых на данные в рамках предметной области

Модель данных можно рассматривать как сочетание трех компонентов:

- Структурная часть – набор правил, по которым может быть построена БД
- Управляющая часть – определяет типы допустимых операций с данными: для обновления и извллечения данных, для изменения структуры данных
- Набор ограничений (необязательный) для поддержки целостности данных, гарантирующих корректность используемых данных (полноту, непротиворечивость и адекватное отражение предметной области)

Definition 1.25. Иерархическая модель данных

Иерархическая модель данных – это модель данных, где используется представление БД в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней – родителей-потомков

В иерархической модели узел может иметь только одного родителя

- Самый верхний узел называется корневым узлом
- Все узлы дерева, за исключением корневого, должны иметь родительский узел
- Связи между отдельными узлами дерева отражаются с помощью направленных ребер графа – от родителя к ребенку

Definition 1.26. Сетевая модель данных

Для описания сетевой модели данных используют понятия "запись" и "связь"

Связь определяется для двух записей: предка и потомка

В сетевой модели данных запись-потомок может иметь произвольное число записей-предков

В сетевой структуре каждый элемент может быть связан с любым другим элементом

Сетевые базы данных подобны иерархическим, за исключением того, что в них имеются указатели в обоих направлениях, которые соединяют родственную информацию

Несмотря на то, что эта модель решает некоторые проблемы, связанные с иерархической моделью, выполнение простых запросов остается достаточно сложным процессом

Также, поскольку логика процедуры выборки данных зависит от физической организации этих данных, то эта модель не является полностью независимой от приложения. Другими словаи, если необходимо изменить структуру данных, то нужно изменить и приложение

Definition 1.27. Реляционная модель данных

Эта модель данных основана на понятии математических отошений (relation)

В реляционной модели данные представлены в виде плоских таблиц, связанных между собой. Необходимо помнить, что таблица есть понятие нестрогое и часто означает не отношение как абстрактное понятие, а визуальное представление отношения на бумаге или экране. В частности – таблицы обычно предполагают упорядоченное хранение данных, в то время как отношения – не обладают этой характеристикой

Между отношениями поддерживаются связи один-к-одному или один-ко-многим

Definition 1.28. Многомерная модель данных

Данные представлены в виде многомерного куба (массива). Измерение (атрибут в реляционной модели) – размерность куба. Факт (агрегированная числовая характеристика) – содержимое ячейки
Агрегаты по всем срезам куба высчитываются один раз и хранятся в базе

Definition 1.29. Объектно-ориентированная модель данных

Данные представлены в виде классов и относящихся к ним объектов

- Класс – тип объекта
- Атрибут – свойство объекта
- Метод – операция над объектом

Инкапсуляция структурного и функционального описания объектов
Наследуемость внешних свойств объектов на основе соотношения "класс-подкласс"

1.10 Классификация СУБД

Definition 1.30. Словарь данных

Словарь данных – набор доступных для выборки всем пользователям базы данных системных таблиц, в которых хранятся метаданные (данные о данных)

Notation 1.32. Классификация СУБД

- Классификация по количеству пользователей
  - Однопользовательские
    - Реализуются на автономном ПК без использования сетей связи
    - Рассчитаны на работу одного пользователя или группы пользователей, разделяющих по времени одно рабочее место
    - Настольные или локальные СУБД
  - Многопользовательские
    - Ориентированы на коллективное использование информации
    - Строятся на базе локальной вычислительной сети
    - Могут быть распределены по нескольким узлам (хостам)
- Классификация по степени распределенности
  - Локальные СУБД – все части размещаются на одном компьютере
  - Распределенные СУБД – части СУБД могут размещаться не только на одном, но на двух и более компьютерах
    - Файл-серверные
    - Клиент-серверные
      - Двухзвенные (СУБД и БД, клиентские приложения)
      - Многозвенные (СУБД и БД, сервер приложений, клиентские приложения)
- Классификация по способу доступа к БД

Notation 1.33. Недостатки файл-серверной архитектуры

СУБД не располагает информацией о том, что происходит на компьютере где хранятся данные:

- Невозможно считать из БД только ту часть данных, которые запрашивает пользователь – считывается файл целиком (блокировка файла)
- Большой объем сетевого трафика (передача по сети множества блоков и файлов, необходимых приложению)
- Узкий спект операций манипулирования с данными, определяемый только файловыми командами
- Отсутствие адекватных средств безопасности доступа к данным (защита только на уровне файловой системы)
- Недостаточно развитый аппарат транзакций служит потенциальным источником ошибок в плане нарушения смысловой и ссылочной целостности информации при одновременном внесении изменений в одну и ту же запись

Definition 1.31. Архитектура клиент-сервер (двухзвенная)

На сервере: база данных, серверная часть СУБД – взаимодействует с БД, обеспечивая выполнение запросов клиентской части
На клиенте: прикладные программы, клиентская часть СУБД – обеспечивает взаимодействие с пользователем и формирование запросов к БД и передача их на сервер
Преимущества:

- СУБД располагает информацией о наборе файлов БД
- Обеспечение разграничения доступа к данным нескольких пользователей
- Считывание только необходимой пользователю информации из файла (блокировка блока данных)
- Обеспечивает корректную параллельную раоту всех пользователей с единой БД

Недостатки:

- Очень большая загрузка на сервер, так как он обслуживает множество клиентов и выполняет всю основную обработку данных
- Нагрузка с обработкой полученных данных дублируется на клиентские хосты

Definition 1.32. Архитектура клиент-сервер (трехзвенная)

Схема: тонкий клиент $\rightarrow$ сервер приложений $\rightarrow$ сервер базы данных
В функции клиентской части (тонкий клиент) входи только интерактивное взаимодействие с пользователем
Вся логика обработки данных (прикладные программы) вынесена на сервер приложений, который и обеспечивает формирование запросов к БД, передаваемых на выполнение серверу БД
Сервер приложений может являться специализированной программой или обычным web-сервером
Преимущества:

- Снижается нагрузка на сервер БД – он занимается исключительно функциями СУБД
- При изменении бизнес-логики нет необходимости изменять клиенсткие приложения
- Максимально снижаются требования к аппаратуре пользователей
- Данная модель обладает большей гибкостью, чем двухуровневые модели

Недостатки:

- Более высокие затраты ресурсов компьютеров на обмен информацией между компонентами приложений по сравнению с двухуровневыми моделями

Definition 1.33. Встраиваемые СУБД

Встаиваемая СУБД – поставляется как составная часть некоторого программного продукта, не требующая процедуры самостоятельной установки
Предназначена для локального хранения данных своего приложения и не рассчитана на коллективное использование в сети
Физически чаще всего реализуется в виде подключаемой библиотеки
Доступ к данным со стороны приложения может происходить через язык запросов либо через специальные программные интерфейсы

1.11 Аспект обработки реляционной модели

Notation 1.34. Математические аппараты для манипулирования данными

Виды:

- Реляционная алгебра – основана на теории множеств. Описывает порядок выполнения операций, позволяющих из исходных выражений получить результат
- Реляционное исчисление – основано на логике предикатор первого порядка. Описывает результат в терминах исходных отношений

Свойство замкнутости операций на множестве отношений. Выражения реляционной алгебры и формулы реляционного исчисления определяются над отношениями реляционных БД и результатом вычисления также являются отношения

В современных РСУБД не используется в чистом виде ни реляционная алгебра, ни реляционное исчисление. Фактическим стандартом доступа к реляционным данным стал язык SQL (Structured Query Language), который представляет собой смесь операторов реляционной алгебры и выражений реляционного исчисления, использующий синтакси, близкий к фразам английского языка и расширенный лополнительными отсутствующими в упомянутых аппаратах

Definition 1.34. Реляционная алгебра (РА)

Реляционная алгебра – это формальный язык операций над отношениями (таблицами), включающий SQL – декларативный язык запросов, который включает не только операции РА, но и дополнительные конструкции (агрегацию, рекурсию, модификацию данных и др.)

Вывод: классическая реляционная алгебра и базовый SQL (без агрегации, рекурсии, оконных функци итд) эквивалентны по выразительной силе в рамках запросов к базе данных. Однако SQL строго мощнее, если учитывать все его возможности (например, рекурсивные запросы, агрегацию, модификацию данных)

Операции реляционной алгебры:

- Теоретико-множественные
  - Объединение отношений
  - Пересечение отношений
  - Вычитание отношений
  - Декартово произведение отношений
- Специальный
  - Выборка (ограничение) отношений
  - Проекция отношения
  - Соединение отношений
  - Деление отношения
- Дополнительные
  - Присваивание (сохранение результатов вычисления)
  - Переименование атрибутов отношения

Notation 1.35. Совместимость по типу

Некоторые реляционные операции требуют, чтобы отношения были совместимы по типу
Отношения являются совместимыми по типу, если их схемы идентичны
Отношения имеют одно и то же множество имен атрибутов, т.е. для любого атрибута в одном отношении найдется атрибут с таким же именем в другом отношении
Атрибуты с одинаковыми именами определены на одних и тех же доменах
Степени схем отношений (количество атрибутов) совпадают

Definition 1.35. Объединение отношений

Объединением двух совместимых по типу отношений A и B называется отношение S с той же схемой, что и у отношений A и B, и состоящее из кортежей, принадлежащих или A, или B, или обоим отношениями
Синтаксис: A UNION B или A $\cup$ B

Remark 1.6.

Объединение, как и любое отношение, не может содержать одинаковых кортежей
Если некоторый кортеж входит и в отношение A, и в отношение B, то в объединение он входит один раз

Definition 1.36. Вычитание отношений

Вычитанием двух совместимых по типу отношений A и B называется отношение S с той же схемой, что и у отношений A и B, и состоящее из кортежей, принадлежащих отношению A и не принадлежащих отношению B
Синтаксис: A EXCEPT B или A $\setminus$ B

Definition 1.37. Пересечение отношений

Пересечением двух совместимых по типу отношений A и B называется отношение S с той же схемой, что и у отношений A и B, и состоящее из кортежей, принадлежащих одновременно обоим отношениям
Синтаксис: A INTERSECT B или A $\cap$ B

Remark 1.7.

Пересечение может быть выражено через операцию вычитания: $A \cap B = A \setminus (A \setminus B)$

Definition 1.38. Декартово произведение отношений

Декартовым произведением двух отношений $A = (A_1, A_2, \dots A_n)$ и $B = (B_1, B_2 \dots B_m)$ называется отношение S, со схемой, состоящей из атрибутов отношений A и B: $(A_1, A_2 \dots A_n, B_1, B_2 \dots B_m)$ и являющееся результатом конкатенации (сцепления) каждого кортежа из отношения A с каждый кортежем из отношения B. В результате получаем набор кортежей $(a_1, a_2 \dots a_n, b_1, b_2 \dots b_m)$, таких, что $(a_1 \dots a_n) \in A$, а $(b_1 \dots b_m) \in B$
Синтаксис: A CROSS JOIN B или $A * B$

Remark 1.8.

Мощность произведения равен произведению мощностей отношений A и B
Если в отношениях A и B имеются атрибуты с одинаковыми наименованиями, то перед выполнением операции декартового произвдеения такие атрибуты необходимо переименовать

Definition 1.39. Выборка (ограничение) отношений

Выборкой (ограничением) называется подмножество кортежей отношения R, удовлетворяющих определенному условию (предикату)
Результат выборки – горизонтальный срез отношения по некоторому условию
Предикат – логическое выражение, в которое могут входить атрибуты отношения R и/или скалярные выражения
Синтаксис: R WHERE или σ R

Definition 1.40. Проекция отношения

Проекцией называется вертикальное подмножество кортежей отношения R, создаваемое посредством извлечения значений указанных атрибутов $A_1 \dots A_n$ отношения
Результат проекции – вертикальный срез отношения, в котором удалены все возникшие при этом дубликаты кортежей
Синтаксис: $R[A_1 \dots A_n]$ или $\prod_{A_1 \dots A_n} R$

Definition 1.41. Соединение отношений

Θ-соединение (тэта-соединение) – определяет отношение S, которое содержит кортежи из декартового произведения отношений A и B, удовлетворяющих предикату Θ
Синтаксис: (A JOIN B) ON Θ

Remark 1.9. Частные случаи

- Экви-соединение – предикат содержит только оператор равенства
- Естественное соединение – эквисоединение отношений A и B, выполненное по всем общим атрибутам, из результатов которого исключается по одному экземпляру каждого общего атрибута
- Левое внешнее соединение – соединение, при котором кортежи отношения A, не имеющие совпадающих значений в общих атрибутах отношения B, также включаются в общее отношение

Definition 1.42. Деление

Пусть даны отношения $A(X_1 \dots X_n, Y_1 \dots Y_p)$ и $B(Y_1 \dots Y_p)$, причем атрибуты $(Y_1 \dots Y_p)$ – общие для двух отношений
Результатом деления отношения A на B является отношение со схемой $S(X_1 \dots X_n)$, содержащее множество кортежей $(x_1 \dots x_n)$, таких, что для всех кортежей $(y_1 \dots y_p) \in B$ в отношении A найдется кортеж $(x_1 \dots x_n, y_1 \dots y_p)$
Отношение A выступает в роли делимого, отношение B выступает в роли делителя
Все атрибуты отношения B должны входить в состав схемы отношения A
Синтаксис: A DIVIDE BY B или A ÷ B

Remark 1.10.

Деление может быть выражено через операции декартова произведения и вычитания
Типичные запросы, реализуемые с помощью операции деления, обычно в своей формулировке имеют слово все

Example 1.1.

1. Получить имена поставщиков, поставляющих деталь номер 2:
   ((DP JOIN P) WHERE DNUM=2)[PNAME]
2. Получить имена поставщиков, поставляющих по крайней мере одну гайку
   (((D JOIN DP) JOIN P) WHERE DNAME=Гайка)[PNAME] (((D WHERE DNAME=Гайка) JOIN DP) JOIN P)[PNAME]
3. Получить имена поставщиков, поставляющих все детали
   ((DP[PNUM,DNUM] DIVIDE BY D[DNUM]) JOIN P)[PNAME]

1.12 Типы данных

Definition 1.43. PostgreSQL и типы данных

SQL – язык со строгой типизацией. Каждый элемент данных имеет некоторый тип, определяющий его поведение и допустимое использование
PostgreSQL наделен расширяемой системой типов, более универсальной и гибкой по сравнению с другими реализациями SQL

Notation 1.36. Типы данных

- Символьные
- Числовые
- Дата и время
- Логические
- Двоичные
- Специальные

Definition 1.44. Символьные данные

varchar(n), char(n), text
Константные зачения. Последовательность символов, заключенная в апострофы. Две строковые константы, разделенные пробельными символами и минимум одним переводом строки, объединяются в одну

- Константы со спецпоследовательностями в стиле C
  Начинаются с буквы E (заглавной или строчной)
- Строковые константы со спецпоследовательностями Unicode
  Позволяют включать в строки символы Unicode по их кодам
  Начинается с U& (строчная или заглавная U и амперсанд)
  Символы Unicode можно записывать двумя способами:
  - \ и код символа из четырех шестнадцатеричных цифр
  - \+ и код символа из шести шестнадцатеричных цифр
- Строковые константы, заключенные в доллары
  Используются для работы со строками, содержащими много апострофов или обратных косых черт. Позволяют избежать необходимости зеркалирования служебных символов. Делают строки более читабельными. Обрамляются $[тэг]$

Definition 1.45. Точные числовые данные

Целочисленные типы – smallint (int2), integer (int4), bigint (int8)
Числа фиксированной точности – numeric (precision, scale) и decimal (precision, scale)

Definition 1.46. Числовые данные с плавающей точкой

real, double precision и float(p)
Поддерживают специальные значения Infinity, -Infinity и NaN
Если точность вводимого числа выше допустимой – будет выполняться округление значения. При вводе слишком большого или очень маленького значения будет генерироваться ошибка
Внимание: сравнение двух чисел с плавающей точкой на предмет равенства их значений может привести к неожиданным результатам

Definition 1.47. опследовательные типы

serial (int4), bigserial (int8) и msallserial (int2)
Реализованы как удобная замена целой группы SQL-команд: создание объекта SEQUENCE – генератор уникальных целых чисел, генерация и получение значений последовательности
Часто используются в качестве значений суррогатного первичного ключа (Primary Key)
Нет неоюходимости указывать явное значение для вставки в поле PK

Notation 1.37. Функции для работы с последовательностями

Функция	Тип результата	Описание
currval('name')	bigint	Возвращает последнее сгенерированное значение указанной последовательности (которое было возвращено при последнем вызове функции nextval)
lastval()	bigint	Возвращает последнее сгенерированное значение любой последовательности (которое было возвращено при последнем вызове функции nextval)
nextval('name')	bigint	Генерит и возвращает новое значение последовательности
setval('name', bigint)	bigint	Устанавливает текущее значение последовательности
setval('name', bigint, boolean)	bigint	Устанавливает текущее значение последовательности и флаг is-called, указывающий на то, что это значение уже использовалось

Definition 1.48. Дата и время

date, time и time with time zone (timetz)
Даты обрабатываются в соответствии с григорианским календарем
time хранит время внутри суток. time with time zone хранит время с учетом смещения, соответствующего часовому поясу
При вводе значений их нужно заключать в одинарные кавычки, как и текстовые строки

Definition 1.49. Временная метка (интегральный тип)

timestamp, timestamp with time zone (timestamptz)
Получается в результате объединения типов даты и времени
Оба типа занимают 8 байтов
Значения типа timestamptz хранятся приведенными к нулевому часовому поясу (UTC), а перед выводом приводятся к часовому поясу пользователя

Definition 1.50. Тип interval

Представляет продолжительность отрезка времени
Формат: quantity unit [quantity unit ...] direction
Стандарт ISO 8601: P[yyyy-mm-dd][Thh:mm:ss]
Значение типа interval можно получить при вычитании одной временной метки из другой

Notation 1.38. Операторы даты/времени

- date +/- integer – добавляет/вычитает к дате заданное число дней
- date +/- interval – добавляет/вычитает к дате интервал
- date +/- time – добавляет/вычитает к дате время
- interval +/- interval – складывает/вычисляет интервалы
- timestamp +/- interval – добавляет/вычитает к метке времени интервал
- date - date – возвращает разницу между датами в днях
- timestamp - timestamp – вычитает из одной отметки времени другую (преобразуя 24-часовые интервалы в дни)

Definition 1.51. Логический тип

boolean
Может иметь три состояния: true, false, NULL. Реализует трехзначную логику

Definition 1.52. Двоичные типы данных

bytea
Позволяют хранить байты с кодом 0 и другими непечатаемыми значениями (значения вне десятичного диапазона 32..126)
В операциях с двоичными строками обрабатываются байты в чистом виде
Поддерживает два формата ввода и вывода (параметр bytea-output):

- hex (шестнадцатеричный) – '\x коды символов в 16-ой системе'
- escape (спецпоследовательностей) – '\коды символов в 8-ой системе'

Notation 1.39. Приведение типов

Приведение типов в PostgreSQL – это осуществление преобразования одного типа информации в другой
Для приведения типов данных в PostgreSQL используется: функция CAST, выражение::тип, тип выражения
Неявные преобразования, производимые PostgreSQL, могут влиять на результат запроса

2 Лекция 2. Введение в язык

2.1 Язык SQL

Definition 2.1. Язык SQL

SQL (Structured Query Language) – язык структурированных запросов
Разработан IBM в 1970-х годах. Принят органами стандартизации ANSI и ISO. Первоначальное название – SEQUEL (Structured English QUEry Language). Широко используется в промышленности (диалекты)
Язык запросов SQL:

- Является декларативным
- Определяет требования к результату, а не алгоритм выполнения
- Регистронезависим

Notation 2.1. История версий стандарта SQL

Название	Изменения
SQL-86	Первый вариант стандарта, принятый институтом ANSI и одобренный ISO в 1987 году
SQL-89	Немного доработанный вариант предыдущего стандарта
SQL-92	Значительные изменения (ISO 9075); уровень Entry Level стандарта SQL-92 был принят как стандарт FIPS 127-2
SQL:1999	Добавлена поддержка регулярных выражений, рекурсивных запросов, поддержка триггеров, базовые процедурные расширения, не скалярные типы данных и некоторое объектно-ориентированные возможности
SQL:2003	Введены расширения для работы с XML-данными, оконные функции, генераторы последовательностей и основанные на них типы данных
SQL:2006	Функциональность работы с XML-данными значительно расширена. Появилась возможность совместно использовать в запросах SQL и XQuery
SQL:2008	Улучшены возможности оконных функций, устранены некоторое неоднозначности стандарта SQL:2003
SQL:2011	Добавлена поддержка temporal баз данных. Расширены возможности использования оконных функций и выражение FETCH
SQL:2016	Добавлен поиск на основе шаблона строк, полиморфические табличные функции, JSON
SQL:2019	Добавлена часть 15, многомерные массивы (MDarray type and operatorrs)

Notation 2.2. Команды языка SQL

SQL command:

- DDL
  - Create
  - Alter
  - Drop
  - **Truncate Table**
- DCL
  - Grant
  - Revoke
- DML
  - Insert
  - Update
  - Delete
- TCL
  - **Begin**
  - Commit
  - Rollback
  - Savepoint
- DQL
  - Select

Remark 2.1. DDL – Data Definition Language

Используются для определения структур базы данных

- CREATE – создание объектов
- ALTER – модификация структуры объектов
- DROP – удаление объектов
- TRUNCATE TABLE – пересоздание таблицы с удалением всех записей и освобождением места

Remark 2.2. DCL – Data Control Language

Используются для управления привилегиями пользователей на доступ к объектам в базе данных

- GRANT – предоставление привилегии
- REVOKE – отзыв ранее предоставленной привилегии (неявный запрет)

Remark 2.3. DML – Data Manipulation Language

Команды языка манипулирования данными используются для выполнения всех типов модификации данных в базе данных

- INSERT – добавление записей в таблицу
- UPDATE – изменение значений в полях таблицы
- DELETE – удаление записей из таблицы

Remark 2.4. TCL – Transaction Control Language

Используются для управления выполнением транзакций

- BEGIN – открытие явной транзакции
- COMMIT – сохранение (фиксация) изменений, выполненных транзакцией
- ROLLBACK – отмена (откат) изменений, выполненных транзакцией
- SAVEPOINT – задание точки сохранения

Remark 2.5. DQL – Data Query Language

Используется для извлечения данных из таблиц базы данных

2.2 Написание простых запросов

Notation 2.3. Синтаксис оператора SELECT

Элемент	Выражение	Описание
SELECT	Список столбцов через запятую	Определяет, какие столбцы должна содержать результирующая таблица
FROM	Определение таблиц-источников строк	Определяет таблицы-источники для извлечения данных
WHERE	Условие отбора исходных строк	Фильтрует данные из таблиц-источников с помощью предиката
GROUP BY	Группировка по списку столбцов	Упорядочивает строки по группам
HAVING	Условие отбора групп	Фильтрует группы с помощью предиката
ORDER BY	Сортировка по списку столбцов	Сортирует строки результирующей таблицы

Remark 2.6. Логическая последовательность выполнения оператора SELECT

Порядок, в котором запрос записывается отличается от порядка в котором запрос выполняется сервером БД

5. SELECT <select list>
7. [INTO new_table_name]
1. FROM <table source>
2. WHERE <search condition>
3. GROUP BY <group by list>
4. HAVING <search condition>
6. ORDER BY <order by list>[ASC|DESC]

Example 2.1. Применение логического порядка операций к написанию SELECT

```
SELECT empid,
    extract('year' from orderdate) AS OrderYear
FROM "Sales"."Orders"
WHERE custid = 71
GROUP BY empid, extract('year' from orderdate)
HAVING COUNT(*) > 2
ORDER BY empid, OrderYear;
```

Remark 2.7. SELECT безо всего

Используется для:

- Инициализации переменных
- Возврата результата выражений и функций

Example 2.2. Извлечение данных из таблицы

Извлечение из всех столбцов таблицы

```
SELECT *
FROM "Sales"."Customers";
```

Извлечение из отдельных столбцов таблицы

```
SELECT companyname, country
FROM "Sales"."Customers";
```

Notation 2.4. Элементы языка

Элементы языка	Предикаты и операторы
Предикаты	BETWEEN, IN, LIKE, IS, ALL, ANY, SOME
Операторы сравнения	=, >, <, >=, <=, <> (!=)
Логические операторы	AND, OR, NOT
Арифметические операторы	*, /, %, +, -, - (унарный)
Конкатенация (*зависит от диалекта языка)	||, *(&, +)

Example 2.3. Вычисляемые столбцы и псевдонимы столбцов

Создание вычисляемых столбцов

```
SELECT unitprice, qty, (qty * unitprice)
FROM "Sales"."OrderDetails";
```

Псевдонимы заключаются в двойные качки, если содержат пробелы, специальные символы или необходимо различать регистры символов

```
SELECT unitprice, qty Quantity, (qty * unitprice) AS Total
FROM "Sales"."OrderDetails";
```

Example 2.4. Псевдонимы таблиц

Создаются в предложении FROM. Полезны при выборке данных из нескольких таблиц

```
SELECT custid, orderdate
FROM "Sales"."Orders" AS SO;
```

Ссылка на столбцы таблицы с использованием псевдонима таблицы

```
SELECT SO.custid, SO.orderdate
FROM "Sales"."Orders" AS SO;
```

Notation 2.5. Влияние логического порядка выполнения запроса на псевдонимы

Предложения FROM, WHERE и HAVING обрабатываются до SELECT
Псевдонимы столбцов создаются в SELECT и видны только в ORDER BY
Выражения, для которых в предложении SELECT определены псевдонимы, должны быть повторно использованы в остальных предложениях запроса

Notation 2.6. Использование выражения CASE в предложении SELECT

Выражение CASE возвращает скалярное значение
CASE может использоваться для:

- Создания вычисляемого столбца в SELECT
- Формирования условия в WHERE или HAVING
- Задания порядка сортировки в ORDER BY

CASE возвращает результат вычисления выражения

Notation 2.7. Формы выражений CASE

- Simple CASE
  - Сравнивает одно выражение со списком возможных значений
  - Возвращает первое совпадение
  - Если совпадений не обнаружено, возвращает значение, основываясь на выражении ELSE
  - Если не найдено совпадений и не определено выражение ELSE, возвращает NULL
- Searched CASE
  - Проверяет набор предикатов или логических выражений
  - Возвращает значение указанное в выражении THEN первого выражения, которое возвращает TRUE

Example 2.5. Simple CASE

```
SELECT contactname,
    CASE contacttitle
        WHEN 'Owner' THEN 'Yes'
        ELSE 'No'
    END AS Owner
FROM "Sales"."Customers"
```

Example 2.6. Searched CASE

```
SELECT productname, unitprice,
    CASE
        WHEN discontinued = 1::bit THEN 'withdrawn from sale'
        WHEN unitprice < 25::money THEN 'low price'
        WHEN unitprice BETWEEN 25::money AND 30::money THEN 'medium-price'
        WHEN unitprice BETWEEN 31::money AND 50::money THEN 'high-price'
    ELSE 'VIP-price'
END AS "Price category"
FROM "Production"."Products"
```

Notation 2.8. Использование функций

Функции форматирования и преобразования	Поддержка приведения и преобразования типов данных	CAST, TO_CHAR, TO_DATE, TO_NUMBER, TO_TIMESTAMP
Логические функции	Выполнение логических операций	NULLIF, GREATEST, LEAST
Функции даты и времени	Выполняют операции над значениями даты и времени	AGE, NOW, CURRENT_DATE, CURRENT_TIME, LOCALTIME, DATE_PART, DATE_TRUNC, MAKE_DATE, EXTRACT
Строковые функции	Выполняют операции со строковыми (char и varchar) значениями	CONCAT, CONCAT_WS, FORMAT, LEFT, LENGTH, LOWER, LTRIM, REPLACE, REGEXP_REPLACE, REVERSE, RIGHT, RTRIM, SUBSTRING, TRIM, UPPER
Математические функции	Выполняют вычисления, основанные на числовых значениях	ABS, CEILING, FLOOR, POWER, ROUND, SQRT, TRUNC
Функции для перечислений	Используются для работы с типами перечислений (ENUM)	ENUM_FIRST, ENUM_LAST, ENUM_RANGE

2.3 Сортировка данных

Notation 2.9. Использование ORDER BY

ORDER BY сортирует записи в результирующем наборе. Без ORDER BY порядок записей результирующей выборки не гарантируется. Сортирует все NULL значения вместе
ORDER BY может ссылаться на:

- Имя столбца, псевдоним или позицию столбца в результирующей выборке (не рекоммендуется)
- Результат выражения
- Столбцы, не используемые в результирующей выборке (если не используется DISTINCT)

ORDER BY не поддерживается в инструкциях SELECT/INTO

Example 2.7. Пример использования ORDER BY

```
SELECT companyname, contactname
FROM "Sales"."Customers" c
ORDER BY country ASC, city desc;

SELECT custid, orderdate
FROM "Sales"."Orders" o
ORDER BY DATE_PART('year', orderdate) DESC;
```

Notation 2.10. Фильтрация с помощью LIMIT*

Ограничивает число строк, возвращаемых в результирующем наборе

- integer_expression – число или числовое выражение, определяющее количество возвращаемых строк
- ALL – равносильно отсутствию указания LIMIT
- OFFSET – указывает число строк, которые необходимо пропустить, прежде чем начать выдавать строки
- Для получения предсказуемого и согласованного результата необходимо использовать фильтрацию отсортированого набора ORDER BY

Example 2.8. Фильтрация с помощью LIMIT

```
SELECT productname, unitprice
FROM "Production"."Products"
WHERE unitprice <= 40::money
ORDER BY unitprice DESC
limit 3;
```

Notation 2.11. Фильтрация в ORDER BY с помощью OFFSET-FETCH

OFFSET-FETCH – это расширение ORDER BY. Позволяет отфильтровать требуемый диапазон строк. Предоставляет механизм для разбиения результирующего набора на страницы

Определяет количество строк, которые необходимо

- Пропустить – OFFSET (может быть ноль, если не нужно пропускать строки)
- Вернуть – FETCH (должно быть больше или равно единице)

Если FETCH опущено – возвращаются все записи до конца набора

WITH TIES – вернуть дополнительные строки, с точки зрения ORDER BY совпадающие с последней строкой набора результатов

Example 2.9. Фильтрация с помощью OFFSET-FETCH

```
SELECT orderid, custid, orderdate
FROM "Sales"."Orders" o
ORDER BY orderdate DESC
OFFSET 50 ROWS FETCH FIRST 50 ROWS ONLY;
```

Notation 2.12. Фильтрация дубликатов

SELECT DISTINCT используется для возврата только уникальных записей: удаляет дубликаты, базируясь на результирующем списке столбцов (не на основе таблицы-источника); работает с записями уже обработанными в выражениях WHERE, HAVING и GROUP BY; NULL значения уникальны

Definition 2.2. DISTINCT ON

DISTINCT ON отличается от DISTINCT тем, что определяет уникальность записей не по всем полям, а только указанным. Строки с одинаковыми значениями выражений считаются дублирующимися и возвращается только первая строка. Обработка DISTINCT ON производится после сортировки ORDER BY

2.4 Фильтрация записей

Definition 2.3. Фильтрация исходных записей – WHERE

Содержит логическое условие: записи, для которых условие возвращает TRUE – попадают в результирующую выборку; FALSE и UNKNOWN – отфильтровываются
В предложении WHERE не доступны псевдонимы столбцов
Данные фильтруются на стороне сервера. Оптмизиация за счет использования индексов, снижение нагрузки на сеть и использование памяти на клиенте

Notation 2.13. Операторы сравнения

Оператор	Описание	Пример
< > = <= >= <> !< !> !=	Операторы сравнения	Name != 'Vasia' или Name <> 'Vasia'
~ и !~	Проверка соответствия строки регулярному выражению POSIX с учетом регистра	'Thomas' ~ '.*thomas.*' → False
~* и !~*	Проверка соответствия строки регулярному выражению POSIX без учета регистра	'Thomas' ~* '.*thomas' → True

Notation 2.14. Предикативные операторы

Оператор	Описание	Пример
BETWEEN/NOT BETWEEN	Проверка по диапазону	orderdate BETWEEN '2006-07-05' AND '2006-07-25'
IN/NOT IN	Проверка на основе списка	Price IN (50, 125, 253, 264)
LIKE/NOT LIKE	Сравнение строк по маске с учетом регистра	City LIKE 'London'
ILIKE/NOT ILIKE	Сравнение строк по маске без учета регистра	City ILIKE 'london'
SIMILAR TO/NOT SIMILAR TO	Сравнение строк по шаблону на основе регулярных выражений в стандарте SQL	'abc' SIMILAR TO '%(b|d)%'
IS NULL/IS NOT NULL	Проверка на наличие/отсутствие NULL значений	region IS NOT NULL
IS DISTINCT FROM/IS NOT DISTINCT FROM	Проверка на неравенство/равенство заданному значению. При этом NULL воспринимается как обычное значение	region IS DISTINCT FROM 'WA'

Example 2.10. Использование операторов сравнения

```
SELECT productname, unitprice
FROM "Production"."Products" p
WHERE discontinued != 1::bit;
```

Example 2.11. Фильтрация с использованием логических операторов

Приоритет логических операторов – NOT, AND, OR
Если два оператора в выражении имеют один и тот же уровень приоритета, они вычисляются в порядке слева направо по мере их появления в выражении
Чтобы изменить приоритет операторов в выражении, следует использовать скобки

```
SELECT *
FROM "Production"."Products" p
WHERE categoryid = 1 OR categoryid = 2 AND unitprice >= 40::money;
```

Example 2.12. Фильтрация NULL значений

NULL значения используются для маркировки отсутствующих значений (missing values)
Для корректно обработки необходимо использовать предикаты:

- IS NULL (ISNULL) или IS NOT NULL (NOTNULL)
- IS DISTINCT FROM (!=) или IS NOT DISTINCT FROM (=)

```
SELECT companyname, region
FROM "Sales"."Customers" c
WHERE region IS NULL;

SELECT companyname, region
FROM "Sales"."Customers" c
WHERE region IS DISTINCT FROM 'WA';
```

Example 2.13. Проверка на принадлежность диапазону

Границы диапазона включены. Для задания исключающего диапазона используйте операторы больше (>) и меньше (<)
Если любой параметр предиката BETWEEN или NOT BETWEEN имеет значение NULL, результат не определен (UNKNOWN)

```
SELECT productname, unitprice
FROM "Production"."Products" p
WHERE unitprice BETWEEN 30::money AND 38::money;
```

Example 2.14. Проверка на принадлежность множеству

Определеяет, совпадает ли указанное значение с одним из значений, содержащихся во вложенном запросе или списке
Использование значений NULL с предикатами IN/NOT IN может привести к непредвиденным результатам

```
SELECT productname, unitprice
FROM "Production"."Products" p
WHERE unitprice IN (22::money, 30::money, 32::money, 38::money);
```

Example 2.15. Проверка на соответствие шаблону

```
match_expression [NOT] LIKE pattern [ESCAPE escape_character]
match_expression [NOT] ILIKE pattern [ESCAPE escape_character]
```

Example 2.16. Фильтрация по шаблону

```
SELECT custid, contactname, contacttitle
FROM "Sales"."Customers" c
WHERE contacttitle LIKE '%Manager';

SELECT custid, contactname, contacttitle
FROM "Sales"."Customers" c
WHERE contactname LIKE 'C%' OR contactname LIKE 'L%';
```

Example 2.17. Сопоставление с началом строки

```
SELECT custid, contactname, contacttitle
FROM "Sales"."Customers" c
WHERE start_with(contactname, 'L');

SELECT custid, contactname, contacttitle
FROM "Sales"."Customers" c
WHERE contactname ^@ 'C' OR contactname ^@ 'L';
```

Notation 2.15. Символы шаблона POSIX

.	любой один символ
[...]	любой один символ в диапазоне или наборе
[^ ...]	любой один символ, не входящий в диапазон или набор
*	повторение предыдущего элемента 0 и более раз
+	повторение предыдущего элемента 1 и более раз
?	вхождение предыдущего элемента 0 или 1 раз
$\{m\}$	ровно m вхождений предыдущего элемента
$\{m,\}$	m и более вхождений предыдущего элемента
$\{m,n\}$	от m до n вхождений предыдущего элемента
()	объединение нескольких элементов в одну логическую группу
|	выбор (одного из двух вариантов)
	привязывает шаблон к началу строки
$	привязывает шаблон к концу строки

Notation 2.16. Проверка на соответствие шаблону POSIX

Возвращает true или false в зависимости от того, соответствует ли данная строка шаблону или нет. Шаблоны соответствуют определению регулярных выражений в стандарте SQL. Регулярные выражения SQL представляют собой гибрид синтаксиса LIKE с синтаксисом обычных регулярных выражений POSIX
Условие SIMILAR TO истинно, только если шаблон соответствует всей строке. Это отличается от условий с регулярными выражениями, в которых шаблон может соответствовать любой части строки

Example 2.18. Фильтрация с помощью SIMILAR TO

```
SELECT custid, contactname, contacttitle
FROM "Sales"."Customers" c
WHERE contactname SIMILAR to 'S(i|m)%';

SELECT custid, contactname, contacttitle
FROM "Sales"."Customers" c
WHERE contactname NOT SIMILAR to '[^B-R]%';
```

Notation 2.17. Операторы регуярных выражений POSIX

Оператор	Описание	Пример
~	Проверка соответствия строки регулярному выражению с учетом регистра	'thomas' ~ 't.*ma'
~*	Проверка соответствия строки регулярному выражению без учета регистра	'thomas' ~* 'T.*ma'
!~	Проверка несоответствия строки регулярному выражению с учетом регистра	'thomas' !~ 't.*max'
!~*	Проверка несоответствия строки регулярному выражению без учета регистра	'thomas' !~* 'T.*ma'

Example 2.19. Фильтрация с использованием операторов регулярных выражений

```
SELECT region
FROM "Sales"."Customers" c
WHERE region ~ '.*ra$';

SELECT companyname, contactname, city
FROM "Sales"."Customers" c
WHERE contactname ~ '^(B|K|S).*(e|k)$';
```

2.5 Массивы

Definition 2.4. Массивы

Многомерные массивы переменной длины
Элементами массивов могут быть:

- Любые встроенные или определенные пользователями базовые типы
- Перечисления, составные типы, типы-диапазоны или домены

Для объявления типа массива

- К названию типа элементов добавляются квадратные скобки
- Запись с ключевым словом ARRAY

Example 2.20.

```
integer[3]
integer ARRAY[3]
integer ARRAY
```

Example 2.21. Определение массива

```
SELECT '{1,2,3,4,5}'::integer[], ARRAY[1,2,3]::integer[];

SELECT '{{1,2},{3,4}}'::integer[], ARRAY[[1,2],[3,4]]::integer[];
```

Example 2.22. Добавление элементов в массив

Добавление элементов в массив:

- Оператор || (конкатенация)
- Функция array_append(массив, элемент)
- Функция array_prepend(элемент, массив)

```
SELECT '{1,2,3,4}'::integer[] || 5,
    array_append('{1,2,3,4}'::integer[],5),
    array_prepend(5,'{1,2,3,4}'::integer[]);
```

Example 2.23. Удаление и изменение элементов массива

Удаление элементов из массива: функция array_remove(массив, значение элемента)
Изменение элемента массива: функция array_replace(массив, старое значение, новое значение)

```
SELECT array_remove('{1,2,3,4}::integer[],3),
    array_replace('{1,2,3,4}'::integer[],2,5);
```

Example 2.24. Извлечение элементов массива

Индексация элементов массива начинается с 1
Для извлечения одного элемента массива – необходимо указать его номер в []
Для извлечения нескольких элементов из массива – необходимо в [] указать номер первого элемента и номер последнего извлекаемого элемента разделенные символом двоеточие

```
select ('{"(921)-745-8965","(908)-567-78234",
        "(911)-745-8512","(812)-750-8965"}'::text[])[2];
```

Example 2.25. Проверка на вхождение в массив

Для проверки вложенности элементов одного массива в состав элементов другого массива используются операторы вложенности массивов (<@, @>)
Один массив считается вложенным в другой, если каждый элемент первого встречается во втором. Повторяющиеся элементы рассматриваются на общих основаниях

```
select '{(408)-567-78234}'::text[] <@ '{(408)-745-8965","(408)-567-78234"}'::
```

Для проверки вхождения литерала в массив используется оператор ANY (SOME)

```
SELECT 10 = SOME (ARRAY[192, 168, 10, 10]);
```

Example 2.26. Разворачивание массива в набор записей

Чтобы представить элементы массива в виде значений некоторого столбца необходимо воспользоваться функцией UNNEST (ANYARRAY)

```
SELECT UNNEST(ARRAY[100, 110, 153, 500]) as prodid,
        50000::money as price,
        '2022-10-20' as change_date;
```

Example 2.27. Пример использования массива

```
select contactname,
        split_part(contactname,',',1) as "Lname",
        trim((string_to_array(contactname,','))[2]) as "Fname"
from "Sales"."Customers";
```

2.6 Типы JSON

Definition 2.5. JSON

JSON (JavaScript Object Notation) – текстовый формат обмена данным, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми. ФОрмат JSON был разработан Дугласом Крокфордом
Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается независимым от яызка и может использоваться практически с любым языком программирования
Для многих языков существует готовый код для создания и обработки данных в формате JSON

Notation 2.18. Структура JSON-данных

JSON – текст, имеющий одну из двух структур:

- Ключ – может быть только строка (регистрозависимость не регулируется стандартом, это остается на усмотрение программного обеспечения)
- Значение – любая допустимая форма

Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность

Notation 2.19. Значения JSON

В качестве json значений могут выступать:

- Запись – неупорядоченное множество пар ключ:значение, заключенное в фигурные скобки. Пары ключ-значение отделяются друг от друга запятыми
- Массив (одномерный) – это упорядоченное множество значений. Массив заключается в квадратные скобки. Значения разделяются запятыми. Массив может быть пустым. Значение в пределах одного массива могут иметь разный тип
- Число (целое или вещественное)
- Литералы (в нижнем регистре) true, false и null
- Строка – это упорядоченное множество из нуля или более символов юникода, заключенное в двойные кавычки. Поддерживаются строковые константы со спецпоследовательностями в стиле C и строковые константы со спецпоследовательностями Unicode

Notation 2.20. Типы данных JSON

Предназначены для сохранения и обработки данных в формате JSON в БД PostgreSQL
Существует два типа данных для работы с JSON, которые отличаются способом хранения данных и способом работы с ними: json и jsonb

Remark 2.8. json

Использует текстовый формат хранения JSON-данных, который позволяет хранить только простые типы данных JSON: строки, числа, логические значения, null и массивы или объекты, состоящие только из этих типов
При хранении данных типа json не происходит никакой оптимизации – высокая скорость записи

- Сохраняет порядок следования ключей и повторяющиеся значения ключей, при этом функции обработки будут считать действительной последнюю пару
- Любые изменения в данных приводят к перезаписи всей строки JSON
- При использовании выполняется полный разбор – низкая скорость обработки

Remark 2.9. jsonb

Использует бинарный формат хранения JSON-данных, который позволяет хранить все типы данных JSON, включая вложенные объекты и массивы, а также дополнительные типы данных, такие как булевы значения, даты и времена итд
Бинарный формат данных jsonb позволяет эффективно хранить, индексировать и быстро выполнять запросы к данным в формате JSON:

- Разбор производится однократно при сохранении – низкая скорость сохранения
- Ключи не дублируются
- Отсортированы по длине и ключу
- При изменении данных не требует перезаписи всей строки JSON, а лишь тех частей, которые изменились
- При использовании разбор не выполняется – высокая скорость обработки

Remark 2.10. ВАЖНО!

PostgreSQL позволяет использовать только одну кодировку символов в базе данных. Если кодировка базы данных не UTF-8:

- Данные JSON не будут полностью соответствовать спецификации
- Нельзя будет вставить символы, непредставимые в кодировке сервера
- Допустимыми будут символы, представимые в кодировке сервера, но не в UTF-8

Example 2.28. Примеры json данных

- Числа, строки в двойных кавычках, true и false (в нижнем регистре) или null

```
SELECT '5'::json, '"Post"'::json, 'true'::json, 'null'::json;
```

- Массив из нуля и более элементов (элементы могут быть разных типов)

```
SELECT '[1, 2, "foo", null]'::json;
```

- Объект, содержащий пары ключей и значений (ключи объектов – всегда строки в двойных кавычках)

```
SELECT '{"cheef":"Ivan", "Empl":["Svetlana","Eugen"]}'::jsonb;
```

Definition 2.6. Функции-конструкторы

Для формирования json можно использовать специализированные функции:

- to_json(anyelement) и to_jsonb(anyelement)
- jsonb_build_object(VARIADIC "any") и jsonb_build_array(VARIADIC "any")
- array_to_json(anyarray [, pretty_bool])
- row_to_json(record [, pretty_bool])
- json_object(keys text[], values text[]) и jsonb_object(keys text[], values text[])

Example 2.29. Извлечение значений по индексу и имени ключа

Получение элементов массива по индексу (индексация элементов начинается с 0. Отрицательные числа задают позиции с конца)

```
SELECT '[{"a":"foo"},{"b":"bar"}]'::json -> 2;
SELECT '[{"a":"foo"},{"b":"bar"}]'::jsonb ->> -1;
```

Получение значения по имени ключа

```
SELECT '{"cheef":"Ivan", "Empl":["Svetlana","Eugen"]}'::json -> 'cheef';
SELECT '{"cheef":"Ivan", "Empl":["Svetlana","Eugen"]}'::jsonb ->> 'Empl';
```

Example 2.30. Проверка наличия ключа верхний уровень (jsonb)

Типы данных json или jsonb не требуют задавать структуру объектов, т.е. конкретные имена ключей и типы значений. Может потребоваться выполнить проверку наличия соответствующего ключа. Сделать это можно только для типа данных jsonb

```
SELECT '[{"a":"foo"},{"b":"bar"}]'::jsonb ? 'b';
```

Example 2.31. Извлечение элементов по пути

Для извлечения элементов можно использовать указание пути

- #> – результат json/jsonb
- #» – результат текст

Example 2.32. Проверка вхождения по пути (jsonb)

Для проверки наличия пути/значения используется оператор @> (<@)

```
SELECT '{"a":1, "b":2}'::jsonb @> '{"b":2}'; --true
```

Notation 2.21. Функции

- json_extract_path – возвращает значение JSON по пути, заданному элементами пути (#>)
- json_extract_path_text – возвращает значение JSON по пути, заданному элементами пути, как text (#»)
- jsonb_path_exists – выполняет поиск ключа как на заданном уровне, так и на произвольном
- jsonb_path_match – выполняет проверку предиката пути JSON для заданного значения JSON
- jsonb_strip_nulls – удаления ключей с NULL-значениями

Notation 2.22. Изменение json и jsonb

- Добавление/изменение элементов (jsonb_set)
- Удаление элементов (-)

2.7 Встроенные функции

Notation 2.23. Типы функций

- Обзор встроенных функций
- Функции для работы с текстом
- Функции для работы с датой/временем
- Функции для работы с числами
- Функции преобразования и форматирования
- Функции для работы с NULL
- Системные информаицонные функции

Notation 2.24. Обзор встроенных функций

PostgreSQL предоставляет широкий набор встроенных функций, работающих со встроенными типами данных
Все встроенные функции разделяются на стандартные функции SQL92 и функции в стиле PostgreSQL

- В функциях SQL92 аргументы разделяются специальными ключевыми словами SQL (такими, как FROM, FOR, USING)
- Функции в стиле PostgreSQL получают аргументы, разделенные запятыми

Внимание: круглые скобки не обязательны только для ряда функци SQL92

Notation 2.25. Использование вложенных функций

Вызовы функций могут быть вложенными: тип данных, возвращаемый внутренней функцией должен быть совместим с типом соответствующего аргумента внешней функции
Допускается вложение вызовов на произвольную глубину

Definition 2.7. Строковые функции

В PostgreSQL существует множество разнообразных строковых функций, предназначенных для форматирования, анализа и сравнения строк. Под строками в данном контексте подразумеваются значения типов character, character varying и text

Notation 2.26. Строковые функции

- char_length('string')
- character_length('string')
- length('string')
- lower('string')
- upper('string')
- strpos('string', 'substring')
- position('substring' in 'string')
- subst('string', from_int[, count_int])
- substring('string' [from int] [for int])
- substring('string' from 'шаблон POSIX')
- substring('string' from 'шаблон SQL' for 'спецсимвол')
- regexp_matches('string', 'шаблон POSIX')
- repeat('string', count_int)
- reverse('string')
- trim([leading|trailing|both]['characters'] from 'string')
- ltrim('string'[, 'characters'])
- rtrim('string'[, 'characters'])
- right('string', count_int)
- left('string', count_int)
- concat(arg1 [,arg2[,...]])
- concat_ws('separator',arg1[,arg2[,...]])
- replace('string','old text','new text')
- translate('string,'old text','new text')
- overlay('string' placing 'substring' from int [for int])
- regexp_replace('string','шаблон POSIX','replacement')
- split_part('string' text, 'delimiter', item_int)
- regexp_split_to_array('string','шаблон POSIX')

Notation 2.27. Использование шаблонов

Для определения шаблонов в PostgreSQL поддерживается два типа регулярных выражений:

- Регулярные выражения в стиле SQL
- Регулярные выражения POSIX

Notation 2.28. Регулярные выражения в стиле SQL

Для определения шаблона в стиле SQL используются:

- _ – любой один символ – '_етров' => 'Ветров', 'Петров' и т.д.
- % – любая строка, содержащая ноль или более символов – 'компьютер%' => 'компьютер', 'компьютеры', 'компьютерный' и т.д.

Definition 2.8. Функция format

Функция format выдает текст, отформатированный в соответствии со строкой формата

```
format(formatstr text [, formatarg "any" [, ...]])
```

formatstr – спецификатор формата

- позиция – строка вида n$, где n – индекс выводимого аргумента. Если позиция опускается, по умолчанию используется следующий аргумент по поярдку
- Флаги – поддерживается только знак минус, который выравнивает результат спецификатора по левому краю если определена ширина
- Ширина – минимальное число символов, которое будет занимать результат данного спецификатора
- Тип спецификатора – определяет преобразование соответствующего выводимого значения
  - S – строка
  - I – SQL-идентификатор, при необходимости заключается в кавычки
  - L – значение аргумента заключается в апострофы, как строка SQL

Notation 2.29. Функции даты/времени

- age(timestamp)
- age(timestamp, timestamp)
- current_date
- current_time
- current_time(integer)
- current_timestamp
- current_timestamp(integer)
- clock_timestamp()
- now()
- localtime
- localtimestamp
- date_trunc('part', timestamp)
- date_trunc(text, interval)
- date_trunc('part', timestamp with time zone, 'time_zone_name')
- date_part('part', timestamp)
- date_part('part', interval)
- extract(part from timestamp)
- extract (part from interval)
- make_date(year int, month int, day int)
- make_time(hour int, min int, sec double precision)
- make_timestamp(year int, month int, day int, hour int, min int, sec double precision)

Example 2.33. Часовые пояса (timezone names)

```
select name, abbrev, uct_offset
from pg_timezone_names;

SELECT TIMESTAMP '2022-02-16 20:38:40' AT
    TIME ZONE 'America/Denver';

select date_trunc('day', timestamptz '2022-10-16 20:38:40+00', 'Australia/Syd
    date_trunc('day', timestamptz '2022-10-16 20:38:40+00', 'US/Samoa');
```

Example 2.34. Функции даты/времени

```
SELECT age('2022-06-25 12:34'::timestamp),
    clock_timestamp( ),
    clock_timestamp( );

SELECT extract(hour from timestamp '2001-02-16 20:38:40'),
    date_part('hour', timestamp '2001-02-16 20:38:40');
```

Notation 2.30. Математические функции

- random()
- ceil(dp/numeric)
- ceiling(dp/numeric)
- floor(dp/numeric)
- round(dp/numeric)
- round(v numeric, s integer)
- trunc(dp/numeric)
- trunc(v numeric, s integer)
- power(a dp, b dp)
- power(a numeric, b numeric)
- abs(x)
- div(y numeric, x numeric)
- mod(y, x)
- sqrt(dp/numeric)
- cbrt(dp)

Definition 2.9. Преобразование типов

Для явного преобразования типов используется

- Функция CAST (стандарт SQL) – CAST(выражение AS type)
- Конструкция :: (PostgreSQL) – выражение::type
- Синтаксис функций приведения к типу – typename(выражение). Будет работать только для типов, имена которых являются допустимыми именами функций!

Remark 2.11. Внимание

Приведение будет успешным, только если определен подходящий оператор преобразования типов
Явно приведение типа можно опустить, если возможно однозначно определить, какой тип должно иметь выражение (неявное преобразование)
Запись typename 'string' можно использовать только для указания типа простой текстовой константы; она не работает для массивов

Notation 2.31. Функции форматирования

Функция	Описание	Пример
to_char(timestamp, text)	Преобразует время в текст	to_char(current_timestamp, 'HH12:MI:SS')
to_char(interval, text)	Преобразует интервал в текст	to_char(interval '15h 2m 12s', 'HH24:MI:SS')
to_char(int, text)	Преобразует целое число в текст	to_char(125, '999')
to_char(double precision, text)	Преобразует плавающее одинарной/двойной точности в текст	to_char(125.8::real, '999D9')
to_char(numeric, text)	Преобразует числовое значение в текст	to_char(-125.8, '999D99S')
to_date(text, text)	Преобразует текст в дату	to_date('05 Dec 2000', 'DD Mon YYYY')
to_number(text, text)	Преобразует текст в число	to_number('12,454.8-', '99G999D9S')
to_timestamp(text, text)	Преобразует строку во время	to_timestamp('05 Dec 2000', 'DD Mon YYYY')

Definition 2.10. Функции для работы с NULL

Функция COALESCE возвращает первый попавшийся аргумент, отличный от NULL. Если же все аргументы равны NULL, результатом тоже будет NULL
Функция NULLIF выдает значение NULL, если занчение1 равно значение2; в противном случае она возвращает значение1. Это может быть полезно для реализации обратной операции к COALESCE

Notation 2.32. Функции получения информации о сеансе

Имя	Тип рез.	Описание
current_catalog current_database()	name	Имя текущей базы данных (в стандарте SQL она называется каталогом)
current_user current_role	name	Имя пользователя в текущем контексте выполнения
session_user	name	Имя пользователя сеанса
current_schema[()]	name	Имя текущей схемы
pg_backend_pid()	int	Код серверного процесса, обслуживающего текущий сеанс
pg_blocking_pids(int)	int[]	Идентификаторы процессов, не дающих серверному процессу с определенным ID получить блокировку
version()	text	Информация о версии PostgreSQL

2.8 Физическое хранение БД. Низкий уровень

Notation 2.33. Физическое хранение базы данных

Для каждой БД существует подкаталог внутри каталога ТП по умолчанию (PGData/base) и каталогов ТП, в которых размещены объекты БД: име подкаталога = OID БД (pg_database). Этот подкаталог – по умолчанию место хранения файлов БД
Все файлы объектов одной БД, принадлежащих одному табличному пространству, будут помещены в один каталог

Notation 2.34. Идентификация файлов связанных с таблицами и индексами

Каждый объект БД, хранящий данные (таблица, индекс, последовательность, материализованное представление), представляет собой набор файлов, расположенных в каталоге БД
Каждый объект имеет уникальный идентификатор объекта БД – OID
Файлы идентифицируются с помощью relfilenode. Значение relfilenode таблиц и индексов изначально совпадают с соответствующими OID. Выполнение команд TRUNCATE, REINDEX, CLUSTER – изменяет значения relfilenode таблиц и индексов

Notation 2.35. Типы файлов (слои)

Каждой объекту БД, хранящему данные, соответствует несколько типов файлов (слоев):

- Основной слой (main) – данные
- Слой инициализации (init)
- Карта свободного пространства (fsm – free space map)
- Карта видимости (vm – visibility map)

Каждый тип файла содержат определенный вид данных
Каждый слой состоит из одного или нескольких файлов-сегментов размером до 1Гб. Когда размер файла превышает 1Гб, PostgreSQL создает новый файл с именем refilenode.1.
Изменение размера сегмета (–with-segsize) – только при сборке

Notation 2.36. Слои данных

Основной слой (main) – это данные: версии строк таблиц или строки индексов. Существует для всех объектов, хранящих данные. Имена файлов состоят только из идентификатора – refilenode, refilenode.1 ...
Слой инициализации (init) – существует только для нежурналируемых таблиц (UNLOGGED) и их индексов:

- Действия над такими таблицами не записываются в WAL-log
- При восстановлении (recovery) PostgreSQL удаляет все слои таких объектов и записывает слой инициализации на место основного слоя (получается пустая таблица)
- Имена файлов имеют окончание _init – refilenode_init

Notation 2.37. Организация данных в куче

Файлы разделены на страницы (блоки). Страница по умолчанию имеет размер 8Кб. Изменение размера блока (–with-blocksize) – только при сборке (16Кб или 32Кб). Собранный и запущенный кластер может работать со страницами только одного размера
Табличные данные на страницах файла хранятся неупорядоченно – куча (HEAP). Записи (Tuple) кучи идентифицируются с помощью TID – tuple id. TID состоит из номера файла, номера блока в файле и позиции строки внутри блока
Для получения доступа к записям используется система указателей, размещаемых в начале страниц. Указатели имеют фиксированный размер (4 байта). Наличие указателей позволяет перемещать строку внутри страницы, не ломая индексные ссылки.
Кортеж идентифицируется внутренним идентификатором кортежа (TID): номер страницы, номер указателя строки

Notation 2.38. Структура страницы

Страница разделена на области. Границы областей страницы записаны в ее заголовке:
- 0 – начало заголовка страницы
- 24 – начало указателей на версии строк
- lower – начало свободного места
- upper – начало данных (версий строк)
- special – начало специальных данных (только для индексов)
- pagesize – конец страницы

Notation 2.39. Структура версии строки

Все версии строк таблицы имеют одинаковую структуру:
- Заголовок фиксированного размера – 23 байта
- Необязательная битовая карта пустых (NULL) значений
- Необязательное поле идентификатора объекта
- Пользовательские данные

Заголовок содержит служебну информация:
- t_xmin – идентификатор транзакции (XID), создавшей кортеж
- t_xmax – идентификатор транзакции, удалившей кортеж
- t_cid – идентификатор команды внутри транзакции (CID), создавшей и/или удалившей кортеж
- infomask – набор служебных битов (флагов), определяющих свойства версии
- ctid – внутренний казатель на более новую версию кортежа. У самой новой (актуальной) версии строки citd ссылается на саму эту версию

Notation 2.40. Инспектирование страиц

Для получения информаици о структуре и содержании страниц используется стандартное расширение pageinspect
При установке расширения становятся доступны функции, позволяющие исследовать страницы баз данных на низком уровне:

- get_raw_page(relname text, blkno int) – возвращает содержимое указанной страницы из основного слоя в bytea
- page_header(page bytea) – возвращает набор полей заголовка указанной страницы
- heap_page_items(page bytea) – возвращает кортежи, их заголовки и содержимое ItemIdData указанной страницы
- page_checksum(page bytea, blkno int4) – возвращает контрольную сумму указанной страницы

Notation 2.41. Просмотр содержимого заголовка

Получение информации из заголовка нулевой страницы из таблицы pg_class
Получение контрольной суммы 0-й страницы, которая должна была бы находиться в заданном блоке

Notation 2.42. Служебные слои

Карта свободного пространства (fsm) – free space map

- Хранит информацию об объеме свободного места на страницах после очистки
- Используется при вставке новых версий строк
- Имена файлов имеют окончание _fsm – relfilenode_fsm, relfilenode_fsm.1
- Существует для всех объектов

Notation 2.43. Служебные слои

Карта видимости (vm) – битовая карта видимости. Появляется только после выполнения очистки. Отмечает чистые страницы, на которых все версии строк видны во всех снимках. Существует только для таблиц
Используется для оптимизации работы процесса очисти (отмеченные страницы не нуждаются в очистке); ускорения индексного доступа
Имена файлов имеют окончание _vm – relfilenode_vm. Начиная с версии 9.6 в этом же слое хранится и карта заморозки

2.9 Хранение больших данных

Notation 2.44. Хранение длинных строк

Физический размер одной записи не может занимать больше одной страницы данных
Для длинных версий строк, размер которых превышает 2Кб, автоматически применяется технология TOAST – The Oversize Attributes Storage Technique: PostgreSQL сжимает и/или разбивает на несколько физических строк и выносит значения поля за пределы таблицы. Это происходит незаметно для пользователя и на большую часть кода сервера влияет незначительно
Для каждой таблицы с потенциально большими полями автоматически создается отдельная TOAST-таблица (и к ней специальный индекс)

Notation 2.45. TOAST-таблица

Каждая TOAST-таблица содержит разделенные на части (chunk) длинные атрибуты записи основной таблицы. Запись в основной таблице будет содержать указатель (chunk_id), указывающий на нужный фрагмент в TOAST-таблице. Одному chunk_id может соответствовать несколько строк TOAST-таблице
TOAST-таблица состоит из 3 столбцов:

- chunk_id – номер чанка, на который ссылается куча
- chunk_seq – номер кжадого сегмента чанка
- chunk_data – часть данных каждого сегмента

TOAST-таблица читается только при обращении к длинному атрибуту. Собственная версионность: если обновление данных не затрагивает длинное значение, новая версия строки будет ссылаться на то же самое значение в TOAST-таблице
TOAST-таблица для базовой таблицы – в схеме pg_toast, для временной – в схеме pg_toast_temp

Notation 2.46. Параметры TOAST

КОд обработки TOAST срабатывает, когда размер записи превышает toast_tuple_threshold байт (по умолчанию 2Кб)
Параметр toast_tuple_target указывает до какого размера в байтах TOAST должен пытаться сжать запись (по умолчанию 2Кб). Значение данного параметра можно переопределить на уровне таблицы
Ни toast_tuple_threshold, ни toast_tuple_target не являются жестким ограничением на максимальный размер кортежа

Notation 2.47. Стратегия обработки больших данных

Атрибут STORAGE столбца таблицы определяет стратегию обработки:

- plain – TOAST не применяется (тип имеет фиксированную длину)
- main – приоритет сжатия. Отдельное хранение – как крайняя мера
- extended – допускается как сжатие, так и отдельное хранение
- external – только отдельное хранение без сжатия. Этот режим можно использовать для повышения производительности (избегая сжатия/распаковки) за счет более высокого потребления дискового пространства

Notation 2.48. TOAST-able типы данных

Стратегия хранения конкретного типа данных определяет стратегию обработки
Информаиця о типе данных и его характеристики хранится в таблице pg_catalog.pg_type
Стратегии хранения:

- p = PLAIN
- x = EXTENDED
- e = EXTERNAL (по умолчанию эта стратегия не используется ни для каких типов данных)
- m = MAIN

Notation 2.49. TOAST

PostgreSQL стремится к тому, чтобы на странице помещалось хотя бы 4 строки
Если размер строки превышает $\frac{1}{4}$ страницы с учетом заголовка, к части значений необходимо применить TOAST:

1. Анализ атрибутов со стратегиями external и extended
   - Extended-атрибуты пытается сжать и, если занчение превосходит $\frac{1}{4}$ страницы, оно отправляется в TOAST-таблицу
   - External-атрибуты обрабатываются так же, но не сжимаются
2. Если это не помогло, осуществляется попытка сжать атрибуты со стратегией main, оставив их на табличной странице
3. Если строка все равно недостаточно коротка, main-атрибуты отправляются в TOAST-таблицу

Remark 2.12. Особенность

Если при определении столбцов таблицы PostgreSQL видит, что размер строки не может превысить TOAST_TUPLE_THRESHOLD, TOAST-таблица не создается
Если вы смените тип какого-либо столбца, в результате чего размер записи увеличится – будет создана TOAST-таблица
Если снова сменить тип – СУБД удалит TOAST-таблицу

Notation 2.50. Алгоритмы сжатия

Для сжатия строк поддерживается два алгоритма:

- PGLZ – используется по умолчанию
- LZ4 – доступен, если PostgreSQL был собран с флагом –with-lz4

LZ4 сжимает данные хуже других алгоритмов, но он очень быстрый
Особенность PGLZ – если ему не удается сжать первые 1024 байта входных даннхы, алгоритм сдается и гвоорит, что эти данные несжимаемые. Если у вас какие-то особые данные, первые байты которых плохо сжимаются, иметь смысл дописать к ним в начале нулевых байт

2.10 Планировщик запросов

Definition 2.11. Планировщик запросов

В PostgreSQL используется планировщик запросов, основанный на стоимости
Основная задача – построение плана выполнения запроса с наименьшей оценочной стоимостью выполнения

- Объем ресурсов ОП
- Объем ресурсов ЦП
- Для приложений работающих с OLTP БД скорость выполнения запроса не должна превышать десятки миллисекунд (в худшем случае сотни миллисекунд)

Планировщик при построении плана выполнения запроса:

- Использу статистические данные оценивает селективность предикатов
- Выполняет анализ различных методов доступа к данным
- Для многотабличных запросов выполняет анализ способов соединений наборов записей

Notation 2.51. Селективность предикатов

Предикаты – выражения, которые оцениваются как истинные или ложные. Используются в предложениях JOIN, WHERE, HAVING
Избирательность (селективность) предикатов

- Процент строк таблицы, соответствующих предикату
- Высокая избирательность = низкий процент возвращаемых строк
- Низкая избирательность = высокий процент возвращаемых строк

Селективность при оптимизации запросов используется для выбора метода доступа к данным и операций соединения в многотабличных запросах

Definition 2.12. EXPLAIN

Чтобы узнать, какой план был выбран планировщиком для запроса, используется команда EXPLAIN

EXLAIN query

- Позволяет посмотреть на план выполнения без фактического выполнения запроса (оценочный план)
- cost – сумма затрат на работу с диском (чтение/запись) и оценка CPU: первое значение – затраты на получение первой строки; второе – затраты на получение всех строк
- rows – приблизительное количество возвращаемых строк
- width – средний размер одной строки в байтах

EXPLAIN (ANALYZE) query

- Фактически выполняет запрос и отображает результат, полученный на реальных данных
- actual time – реальное время в миллисекундах, затраченное для получения первой строки и всех строк соответственно
- rows – реальное количество полученных строк
- loops – сколько раз пришлось выполнить операцию
- Execution Time – общее время выполнения запроса

EXPLAIN (ANALYZE, BUFFERS) query

- Позволяет посмотреть что происходит на физическом уровне при выполнении запроса
- Buffers: shared read – количество блоков, считанное с диска
- Buffers: shared hit – количество блоков, считанных из кэша PostgreSQL
- Объем кэша определяется параметром shared_buffers
- Sort Method: external merge Disk – при сортировке используется временный файл на диске
- Sort Method: quicksort memory – сортировка целиков проведена в оперативной памяти

EXPLAIN (SETTINGS) query

- Добавляет в выходные данные EXPLAIN параметры конфигурации, отличные от параметров по умолчанию (которые плияют на планирование запросов)
- Добавлен в PostgreSQL v12 (2019) и по умолчанию отключен

2.11 Методы доступа к данным

Notation 2.52. Методы доступа к данным

Метод доступа характеризует способ, используемый для прсомотра таблиц с целью извлечения только тех записей, которые соответствуют условиям запроса

- Последовательное сканирование (Seq Scan) (без использования индекса)
- Индексное сканирование (Index Scan)
- Сканирование битовой карты (Bitmap Scan)
- Исключительно индексное сканирование (Index Only Scan)

Notation 2.53. Доступ к данным в куче

При выполнении запроса к данным в куче выполняется последовательное сканирование таблицы целиков. Полностью читаются файлы основного слоя таблицы. На каждой прочитанной странице:

- Проверяется видимость каждой версии строки
- Версии, не удовлетворяющие условиям запроса, отбрасывается

Чтение происходит через буферный кеш. При последовательном сканировании больших таблиц используется буферное кольцо небольшого размера:

- Несколько процессов, одновременно сканирующих таблицу, могут присоединяться к кольцу и тем самым экономить дисковые чтения
- Поэтому в общем случае сканирование может начаться не с начала файла

Definition 2.13. Последовательное сканирование

Последовательное сканирование – самый эффективный способ прочитать всю таблицу или значительную ее часть. Хорошо работает при низкой селективности запроса. Последовательное чтение выполняется быстрее, чем чтение страниц вразнобой. Это особенно верно для жестких дисков, где механическая операция подведения головки к дорожке занимает существенно больше времени, чем само чтение данных; в случае дисков SSD этот эффект менее выражен
Для поиска поднабора записей данный метод малоэффективен
Решение: нужно создать индекс

Definition 2.14. Что такое индексы?

Индексы в PostgreSQL – специальные объекты базы данных, предназначенные в основном для ускорения доступа к данным. Это вспомогательные структуры, сохраняемые в отдельных файлах. Их можно удалить и восстановить заново по информации в таблице. Индексы также служат для поддержки некоторых ограничений целостности
Индекс устанавливает соответствие между ключом индекса (значением проиндексированного столбца) и TID записей, в которых это значение встречается. С помощью ключа можно быстро найти интересующие нас строки, без необходимости полного сканирования всей таблицы. Индекс, созданный по столбцу, участвующему в соединении двух таблиц, может позволить ускорить процесс выборки записей из таблиц

Remark 2.13. Издержки

Замедление операций модификации данных в таблицах

- При выполнении операций модификации над проиндексированными данными индексы должны быть перестроены в рамках той же транзакции
- Обновление полей таблицы, не входящих в состав ключа индекса, не приводит к перестроению индексов – HOT (Heap-Only Tuple)
- Чем больше смещение профиля нагрузки в сторону увеличения записей по отношению к чтению, тем это более критично

Дополнительные объемы дискового пространства

- Размер совокупных индексных данных не должен превышать 0,5 размера таблицы

Усложненное техническое обслуживание

- Добавление новых версий строк при модификации данных – Vacuum решает задачу для основных записей, но не для индексных
- Индексы могут распухать (BLOAD) на часто изменяемых данных и их необходимо пересоздавать. Это требует тонкой работы, чтобы не устроить даун-тайм для всего сервиса

Поддержкой индексов в актуальном состоянии занимается СУБД – тратя ресурсы

Definition 2.15. Индексное сканирование (Index scan)

Индексное сканирование (Index Scan)

- На основе ключа индекса определяется TID
- Происходит обращение к странице таблицы, на которую указывает TID, для получения версии строки
- Проверяется ее видимость в соответствии с правилами многоверсионности, и возвращаются полученные данные
- Процесс повторяется для каждого TID

Индексное сканирование хорошо работает, если версии строк в таблице коррелированы с порядком, в котором метод доступа выдает идентификаторы

Используется:

- Для быстрого поиска по ключу
- Для выдачи остортированного результата в том случае, когда порядок сортировки запроса совпдаает с порядком сортировки индексных записей
- Для получения данных, которые присутствуют в индексе без обращения к таблице

Definition 2.16. Bitmap Scan

При увеличении выборки возрастают шансы, что придется возвращаться к одной и той же табличной странице несколько раз
Сканирование битовой карты

- Возвращаются все TID, соответствующие условию (узел Bitmap Index Scan)
- По ним строится битовая карта версий строк (exact)
- Затем версии строк читаются из таблицы (узел Bitmap Heap Scan) – при этом каждая страница будет прочитана только один раз

Размер битовой карты не может превышать размер work_mem. Если выборка слишком велика – строится грубая битовая карта страниц (lossy), содержащих хотя бы одну подходящую версию строки. Для создания более точной битовой карты – увеличить размер work_mem. При чтении страницы приходится перепроверять условия для каждой хранящеймя там строки (узел Recheck Cond)
При выполнении фильтрации по нескольким проиндексированным полям таблицы, сканирование битовой карты позволяет (если оптимизатор сочтет это выгодным) использовать несколько индексов одновременно:

- Для каждого индекса строятся битовые карты версий строк
- Если выражения соединены условием AND – битовые карты логически побитово умножаются
- Если выражения соединены условием OR – битовые карты логически побитово складываются

Definition 2.17. Index Only Scan

Если все необходимые для запроса данныые находятся на индексных страницах, такой индекс называется покрывающим (covering). В этом случае оптимизатор может применить исключительно индексное сканирование (Index Only Scan)
Для прокешированных данных результаты практически не отличаются от Index Scan
Индексы в PostgreSQL не содержат информации, позволяющей судить о видимости строк

- Метод доступа возвращает все версии строк, попадающие под условие поиска, независимо от того, видны они текущей транзакции или нет
- Если страница, на которую указывает TID отмечена в карте видимости – видимость не проверяется
- Если страница, на которую указывает TID, не отмечена в карте видимости выполняется проверка видимости строки на основании табличных данных

Требуется регулярное выполнение очистки VACUUM. Оптимизатор учитывает число мертвых строк и может отказаться от использования исключительно индексного сканирования. Число вынужденных обращений к таблице (Heap Fetches) можно узнать, используя команду explain analyze

Definition 2.18. Parallel Index Scan

Индексный доступ может выполняться в параллельном режиме:

- Сначала ведущий процесс (gather) спускается от корня дерева к листовой странице
- Затем рабочие процессы (parallel_workers) выполняют параллельное чтение листовых страниц индекса, двигаясь по указателям

Издержки:

- Процесс, прочитавший индексную страницу, выполняет и чтение необходимых табличных страниц
- Если несколько индексных страниц, прочитанных разными процессами, содержат ссылку на одну и ту же табличную страницу – ее прочитают несколько процессов (страница будет находиться в буферном кеше в одном экземпляре)

Notation 2.54. Число рабочих процессов

Равно нулю (параллельный план не строится), если размер выборки $<$ min_parallel_index_scan_size (=512kB)
Фиксировано, если для таблицы указан параметр хранения parallel_workers
Вычисляется по формуле 1 + [log 3 (размер выборки / min_parallel_index_scan_size)], но не больше, чем max_parallel_workers_per_gather

Notation 2.55. Сравнения стоимости методов доступа

Каждый из методов доступа имеет свои преимущества и недостатки:

- Индексное сканирование зависит от корреляции между физически расположением версий строк и порядком, в котором индексный метод доступа выдает идентификаторы
- Сканирование только индекса сильно зависит от карты видимости
- Сканирование по битовой карте при слабой корреляции существенно превосходит индексное сканирование
- Последовательное сканирование не зависит от селективности и, начиная с некоторой доли выбираемых строк, обычно работает лучше остальных методов

Планировщик выполняет серьезную работу по оценке эффективности каждого из методов в каждом конкретном случае. Чтобы эти оценки были близки к реальности, очень важна актуальная статистика

Notation 2.56. Параметры сервера

- seq_page_cost (floating point) – стоимость получения страницы при последовательном чтении. По умолчанию 1.0
- random_page_cost – стоимость получения случайной страницы. По умолчанию 4.0
- Оба параметра могут быть изменены для tablespace (SSD) или когда данные целиком умещаются в памяти. Таким образом, можно для активной части базы в памяти иметь random_page_cost близкий к единице, а для архивных данных – оставить значение по умолчанию
- cpu_tuple_cost – стоимость получения страницы данных из памяти. По умолчанию 0.01
- cpu_index_tuple_cost – то же самое, только для индекса. По умолчанию 0.005
- cpu_operator_cost – стоимость обработки оператора или функции. По умолчанию 0.0025

Значение параметра effective_cache_size не влияет на выделение памяти, но подсказывает PostgreSQL, на какой общий размер кэша рассчитывать, включая кэш операционной системы. Чем выше это значение, тем большее предпочтение отдается индексам. Начать можно с 50-75% от объема ОЗУ

Параметры, включающие или отключающие использование планировщиком запросов соответствующих методов доступа:

- enable_seqscan
- enable_indexscan
- enable_indexonlyscan
- enable_bitmapscan

Значение по умолчанию – on. Если off – препятствует планировщику использовать соответствующий метод доступа, если есть другие доступные методы

2.12 Способы соединения

Notation 2.57. Соединения

Очень часто запросы обращаются к данным из нескольких таблиц:

- Недостаточно получать данные с помощью соответствующих методов доступа
- Необходимо соединять два и более наборов строк
- Проверять наличие связных записей в соединяемых наборах

При написании запроса используются логические операции соединения таблиц (JOIN и их производные). Выбор типа логической операции зависит от решаемой задачи. Порядок соединения таблиц не важен с точки зрения логики запроса

Для реализации соединения таблиц PostgreSQL использует способы соединения наборов записей, учитывающие вопросы производительности. Наборы записей всегда соединяются попарно. Порядок соединения наборов записей может влиять на производительность

Notation 2.58. Типы соединений наборов записей

- Соединение вложенными циклами (Nested Loop)
- Соединение хешированием (Hash Join)
- Соединение слиянием (Merge Join)

Definition 2.19. Nested Loop

Реализован в виде двух вложенных циклов. Для каждой записи первого (внешнего – outer) набора проверяется условие соединения во втором (внутреннем – inner) наборе. Процесс повторяется до тех пор, пока внешний набор не исчерпает все строки. Обращение к записям второго (внутреннего) набора осуществляется столько раз, сколько записей в первом (внешнем) наборе. Если отсутствует эффективный метод доступа для поиска соответствующих записей во втором наборе (отсуттсвует индекс) эффективность сильно снизится, так как придется просматривать большое количество ненужных записей

Данный тип объединения оптимизатор обычно предпочитает для небольших выборок

Единственный способ для соединений не по =

Подходит для OUTER JOIN, EXISTS, NOT EXISTS. задает порядок соединения

1. Nested Loop Left Join – возвращает строки, даже если для левого набора записей не нашлось соответствия в правом наборе записей (LEFT JOIN)
2. Nested Loop Anti Join – возвращает записи одного набора в том случае, если для них не нашлось соответствия в другом наборе (NOT EXISTS, LEFT|RIGHT JOIN ... WHERE поле IS NULL)
3. Nested Loop Semi Join – возвращает строки одного набора, если для них нашлось хотя бы одно соответствие в другом наборе (EXISTS) (т.е. достаточно получить всего одну строку)

Модификаций алгоритма вложенного цикла для правого (RIGHT) и полного (FULL) соединений не существует. Это связано с тем, что полный проход по второму набору строк может не выполняться

Соединение вложенным циклом может использоваться вв параллельных планах

Definition 2.20. Соединение с индексированным вложенным циклом

Если для внутренней таблицы существует индекс, который может ускорить поиск записей, удовлетворяющих условию соединения, планировщик рассматривает возможность использования этого индекса для прямого поиска записей внутренней таблицы вместо последовательного сканирования – indexed nested loop join
Если есть индекс внешней таблицы и его атрибуты участвуют в условии соединения, то его можно использовать для сканирования индекса вместо последовательного сканирования внешней таблицы. В частности, при наличии индекса, атрибутом которого может быть предикат доступа в предложении WHERE, диапазон поиска во внешней таблице сужается; следовательно, стоимость соединения с вложенным циклом может быть значительно снижена

Definition 2.21. Hash join

Реализация:

- По одному набору записей строится хеш-таблица на ключах соединения (узел Hash). Функция хеширования равномерно распределяет значения по ограниченному числу корзин хеш-таблицы – по значению хеш-функции определяется номер корзины. Если равномерности не будет – в одной корзине может образоваться список значений, что приведет к снижению производительности
- Затем Postgres выполняет итерацию по второму набору записей:
  - Вычисляет хеш-функцию от значения полей, участвующих в условии соединения
  - Проверяет наличие соответствующего хеш-значения в хеш-таблице hash(table1.val)=hash(table2.val)
  - Дополнительно проверяет значения полей, подходящие под условие соединения

Используется для соединений по =
Для большой выборки оптимизатор предпочитает соединение хешированием. Зависит от порядка соединения – внутренний набор должен быть меньше внешнего, чтобы минимизировать хеш-таблицу
Размер хеш-таблицы в памяти ограничен значением work_mem x hash_mem_multiplier (начиная с 13 версии). В корзину помещаются хеш-код и все поля, которые входят в условие соединения или используются в запросе. Не следует использовать в запросе лишние поля, в том числе звездочку
Если таблица не умещается в памяти:

1. Разбивает наборы записей на пакеты (Batches)
2. Строит хеш-таблицу для одного пакета (work_mem x hash_mem_multiplier)
3. Остальные пакеты сохраняет во временные файлы (temp written)
4. Выполняет сопоставление с записями второго набора
5. Если запись соответствует данному пакету – она возвращается. Если строка принадлежит другому пакету, она сбрасывается на диск – в свой временный файл
6. Процедура повторяется для всех пакетов

Notation 2.59. Группировка и уникальные значения

Для выполнения группировки (GROUP BY) и устранения дубликатов (DISTINCT и операции со множествами без ALL) используются методы, схожие с методами соединения: один из способов выполнения состоит в том, чтобы построить хеш-таблицу по нужным полям и получить из нее уникальные значения

Definition 2.22. Merge join

Для использования данного способа основным условием является соединение упорядоченных наборов записей

- Если наборы записей не отсортированы – выполняет сортировку (дорогая операция)
- Сначала берет первые записи обоих наборов и сравнивает их между собой
- Если значения совпадают – читает следующую строку второго набора
- Если значения не совпадают – читает следующую строку того набора, для которого значение поля, по которому происходит соединение, меньше (один набор догоняет другой)
- Главное отличие – нет правого и левого набора записей

Алгоритм слияния возвращает результат соединения в отсортированном виде. ПОлученный набор записей может быть использован для соединения слиянием со следующим набором записей без дополнительной сортировки

Подходит для outer join-ов. Не определяет жестко порядок соединения таблиц. Чуть ли не единственный способ реализовать full outer join

Если соединяемые наборы записей уже остортированы (например определны индексы) PostgreSQL может предпочесть данный способ соединению хешированием. Не требуются накладные расходы на простроение хеш-таблицы

Если требуется выполнение сортировки, то в зависимости от количества записей и объема work_mem

- Может быть выполнена быстрая сортировка (quick sort)
- Может быть применена частичная пирамидальная сортировка (top-N heapsort) если нужно отсортировать не весь набор данных (LIMIT)
- Если данные уже отсортированы, но не по всем ключам, может быть использована инкрементальная сортировка
- Либо может быть отдано предпочтение соединению хешированием

Remark 2.14. Другие варианты

Подобно объединению с вложенным циклом, соединение слиянием в PostgreSQL fnкже имеет варианты, на основе которых может выполняться сканирование индекса внешней таблицы

Notation 2.60. Параметры сервера связанные со способами объединения

Параметры, включающие или отключающие использование планировщиком запросов соответствующих способов объединения:

- enable_nestloop
- enable_hashjoin
- enable_mergejoin

Значение по умолчанию – on. Если off – препятствует планировщику использовать соответствующий способ объединения, если есть другие доступные способы

3 Лекция 3. Запросы к нескольким таблицам

3.1 Запросы к нескольким таблицам

Definition 3.1. Предложение FROM

В предложении FROM перечисляются один или несколько источников данных, используемых для формирования результирующей выборки
При использовании нескольких источников необходимо указать тип их соединения:

- CROSS JOIN
- [INNER] JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN
- NATURAL JOIN
- USING

Definition 3.2. CROSS JOIN – перекрестное соединение

CROSS JOIN – возвращает декартово произведение таблиц. Может быть заменено списком таблиц через запятую в разделе FROM

```
FROM tableA CROSS JOIN tableB
```

Notation 3.1. Соединение посредством предиката

```
FROM tableA [type_of_join] JOIN tableB ON predicate
```

Тип соединения	Описание
INNER	Используется для соединения строк из обеих таблиц на основе заданного предиката. Используется по умолчанию, когда тип соединения явно не задан
LEFT [OUTER]	"Левое (внешнее)" Включает в себя все строки из левой таблицы A и те строки из правой таблицы B, для которых выполняется условие предиката. Для строк из таблицы A, для которых не найдено соответствия в таблице B, в столбцы, извлекаемые из таблицы B, заносятся значения NULL
RIGHT [OUTER]	"Правое (внешнее)" Включает в себя все строки из правой таблицы B и те строки из левой таблицы A, для которых выполняется условие предиката. Для строк из таблицы B, для которых не найдено соответствия в таблице A, в столбцы, извлекаемые из таблицы A, заносятся значения NULL
FULL [OUTER]	"Полное (внешнее)" Это комбинация левого и правого соединений. В полное соединение включаются все строки из обеих таблиц. Для совпадающих строк поля заполняются реальными значениями, для несовпадающих строк поля заполняются в соответствии с правилами левого и правого соединений

Remark 3.1.

Вместо ON можно использовать USING при условии совпадения названий столбцов, используемых для соединения

Definition 3.3. SELF JOIN

Для объединения записей внутри одной таблицы необходимо использовать объединение таблицы со своей копией. Для создания копии используются псевдонимы таблиц

3.2 Использование операторов наборов записей

Notation 3.2. Требования к наборам записей

Оба набора должны иметь одинаковое количество столбцов, совместивых по типу данных
ORDER BY не допускается во входных запросах, но может использоваться для сортировки результирующего набора
NULL значения считаются равными при сравнении наборов

Definition 3.4. UNION и UNION ALL

UNION возвращает результирующий набор уникальных строк, объединенных из двух входных наборов. Дубликаты удаляются при обработке запроса (влияет на производительность)
UNION ALL возвращает результирующий набор со всеми строками из двух входных наборов. Чтобы избежать потери производительности, вызванной фильтрацией дубликатов, используйте UNION ALL вместо UNION всякий раз, когда это позволяют требования

Definition 3.5. INTERSECT

INTERSECT возвращает уникальный результирующий набор строк, которые присутствуют в обоих входных наборах

Definition 3.6. EXCEPT (MINUS)

EXCEPT возвращает только уникальные строки, которые присутствуют в левом наборе, но нет в правом. Порядок, в котором указаны наборы, имеет значение

4 Лекция 4.

4.1 Запросы с группировкой

Definition 4.1. Агрегатные функции

Работают с набором значений столбца, возвращают скалярное значение (без имени столбца), игнорируют NULL-значения, кроме COUNT(*)
Могут использоваться в:

1. Предложениях SELECT, HAVING и ORDER BY
2. Часто используются с предложением GROUP BY

Notation 4.1. Категории агрегатных функций

- Общие
  - sum
  - min
  - max
  - avg
  - count
  - string_agg
- Статистические
  - corr
  - covar_
  - regr_
  - stddev_
  - var_
- Дополнительные
  - grouping
  - array_agg
  - bit_and
  - bit_or
  - bool_and
  - bool_or
  - json_agg
  - json_object_agg

Definition 4.2. Агрегатные выражения

Агрегатное выражение представляет собой применение агрегатной функции к строкам, выбранным запросом
Агрегатное выражение может записываться следующим образом:

```
aggr ([ALL] expr [, ...] [order_by]) [FILTER (WHERE predicate)]
aggr (DISTINCT expr [, ...] [order_by]) [FILTER (WHERE predicate)]
aggr (*) [FILTER (WHERE predicate)]
```

Notation 4.2. Использование DISTINCT в агрегатных функциях

Используйте DISTINCT с агрегатными функциями, чтобы агрегировать только уникальные значения. Агрегаты с DISTINCT исключают повторяющиеся значения, а не строки (в отличие от SELECT DISTINCT)

Notation 4.3. Обработка NULL-значений

Большинство агрегатных функций игнорирует NULL
- COUNT(<column>) – игнорирует NULL-значения
- COUNT(*) – подсчитывает все строки

NULL может давать неправильные результаты (например, при использовании AVG). Используйте COALESCE для замены NULL-значений перед агрегированием

Definition 4.3. Предложение FILTER

Если добавлено предложение FILTER, агрегатной функции подаются только те входные строки, для которых условие фильтра вычисляется как истинное; другие строки отбрасываются

Definition 4.4. Предложение ORDER_BY

Результат выполнения некоторых агрегатных функций (таких как array_agg и string_agg) зависит от порядка данных
Для задания нужного порядка необходимо добавить предложение order_by. Данное предложение выполняет сортировку на основе исходных столбцов. Нельзя использовать имена результирующих столбцов или их числовые индексы

Notation 4.4. Использование GROUP BY

GROUP BY создает группы из исходных записей в соответствии с уникальной комбинацией значений, указанных в предложении GROUP BY. Детальные строки теряются после обработки предложения GROUP BY. Если запрос использует GROUP BY, все последующие этапы работают с группами, а не с исходными данными
HAVING, SELECT и ORDER BY должны возвращать одно значение для каждой группы. Все столбцы в SELECT, HAVING и ORDER BY должны появляться в предложении GROUP BY или быть входными данными для агрегатных выражений

Definition 4.5. GROUPING SETS

Оператор GROUPING SETS – это расширение предложения GROUP BY
GROUPING SETS группирует записи на основе задаваемых пользователем наборов для группировки. Наборы для группировки – должны включать в себя столбцы, по которым вы группируете с помощью предложения GROUP BY. Набор для группировки обозначается списком столбцов, разделенных запятыми, заключенных в круглые скобки

Definition 4.6. CUBE

CUBE – это расширение предложения GROUP BY
CUBE позволяет создавать все возможные группы на основе указанных столбцов измерений

Definition 4.7. ROLLUP

В отличие от CUBE, ROLLUP не генерирует все возможные группы на основе указанных столбцов
ROLLUP предполагает иерархию входных столбцов и генерирует все группы, которые имеют смысл с учетом иерархии. По этой причине ROLLUP часто используется для создания промежуточных и общих итогов отчетов

Definition 4.8. Функция GROUPING

Функция GROUPING возвращает целочисленную битовую маску, показывающую, какие аргументы не вошли в текущий набор группирования: бит равен 0 – если аргумент является членом текущего набора группирования, и 1 – если аргумент не является членом текущего набора группирования
Аргументы функции GROUPING должны в точности соответствовать выражениям, заданным в предложении GROUP BY

4.2 Использование оконных функций

Remark 4.1. Наборы или курсоры?

Часто решения для запроса данных на основе SQL делят на два вида: основанные на наборах записей и основанные на курсорах с итеративным проходом по записям
Наиболее эффективно – обрабатывать наборы записей: таблица представляет собой набор неупорядоченных данных, сохраняются реляционные принципы обработки
Язык SQL никак не связан с конкретной физической реализацией в ядре СУБД. Задача физического уровня – определить, как выполнить логический запрос, и максимально быстро вернуть правильный результат

Notation 4.5. Зачем нужны оконные функции

Оконные функции – это мощнейший инструмент для анализа данных, который с легкостью помогает решать множество задач
Отличие от агрегатных функций:

- Групповые запросы формируют информацию в виде агрегатов, но при этом теряются детали (уменьшают количество строк)
- Оконные функции позволяют использование как детальных данных, так и агрегатов (не уменьшают количество строк)
- Результаты работы оконных функций просто добавляются к результирующей выборке в виде нового столбца

Этот функционал очень полезен для построения аналитических отчетов, расчета скользящего среднего и нарастающих итогов, а также для расчетов различных моделей атрибуции

Notation 4.6. Задачи решаемые с помощью оконных функций

- Ранжирование (всевозможные рейтинги)
- Сравнение со смещением (соседние элементы и границы)
- Агрегация (сумма и среднее)
- Скользящие агрегаты (сумма и среднее в динамике)

Definition 4.9. OVER

При использовании инструкции OVER() без предложений окном будет весь набор данных

Definition 4.10. Предложение PARTITION BY

Предложение PARTITION BY делит строки на несколько групп или разделов, к которым применяется оконная функция. Определяет столбец (столбцы), по которому будет производиться разделение набора строк на окна
Предложение PARTITION BY является необязательным
Если вы пропустите предложение PARTITION BY, оконная функция будет рассматривать весь результирующий набор как один раздел

Definition 4.11. Предложение ORDER BY

Предложение ORDER BY указывает порядок строк в каждом разделе, к которому применяется оконная функция. Сортирует строки внутри окна для вычисления нарастающего итога
Предложение ORDER BY использует параметр NULLS FIRST или NULLS LAST, чтобы указать, должны ли значения, допускающие значение NULL, быть первыми или последними в результирующем наборе. По умолчанию используется опция NULLS LAST
Предложение ORDER BY является обязательным для некоторых функций

Definition 4.12. FILTER

Предложение FILTER определяет какие строки будут подаваться на вход оконной функции. Подаются только те входные строки, для которых условие фильтра вычисляется как истинное; другие строки отбрасываются. Предложение FILTER допускается только для агрегирующих оконных функций

Definition 4.13. Именованное окно

Предложение WINDOW определяет подмножество строк в текущем разделе, к которому применяется оконная функция. Это подмножество тсрок называется именованное окно

Notation 4.7. Определение фрэйма (рамки)

Фрэйм определяет рамку окна, ограничивающую подмножество строк текущего раздела. Оконная функция работает с рамкой, а не со всем разделом
Определение фрэйма (режимы):

- ROWS позволяет ограничить строки в окне, указывая фиксированное количество строк, предшествующих или следующих за текущей
- RANGE работает с набором строк имеющих одинаковый ранг в инструкции ORDER BY

Необходимо наличие ORDER BY

Notation 4.8. Задание границ фрэйма

Для ограничения строк ROWS или RANGE можно использовать следующие ключевые слова:

- UNBOUNDED PRECEDING – указывает, что фрэйм начинается с первой строки группы
- UNBOUNDED FOLLOWING – с помощью данной инструкции можно указать, что фрэйм заканчивается на последней строке группы
- CURRENT ROW – инструкция указывает, что фрэйм начинается или заканчивается на текущей строке
- ЧИСЛО PRECEDING – определяет число строк перед текущей строкой
- ЧИСЛО FOLLOWING – определяет число строк после текущей строки

Если конец рамки не задан, подразумевается CURRENT ROW

Notation 4.9. Исключение рамки

Исключение рамки позволяет исключить из рамки строки, которые окружают текущую строку, даже если они должны быть включены согласно указаниям, определяющим начало и конец рамки

Исключение рамки может быть следующим:

- EXCLUDE CURRENT ROW – исключает из рамки текущую строку
- EXCLUDE GROUP – исключает из рамки текущую строку и родственные ей согласно порядку сортировки
- EXCLUDE TIES – исключает из рамки все родственные строки для текущей, но не собственно текущую строку
- EXCLUDE NO OTHERS – явно выражает поведение по умолчанию – не исключает ни текущую строку, ни родственные ей

Notation 4.10. Обработка оконных функций

Если запрос содержит оконные функции, эти функции вычисляются после каждой группировки, агрегатных выражений и фильтрации HAVING: оконные функции видят не исходные строки, полученные из FROM/WHERE, а сгруппированные

При использовании нескольких оконных функций с одинаковым определением окна (PARTITION BY и ORDER BY) все функции обрабатывают данные за один проход. Они увидят один порядок сортировки, даже если ORDER BY не определяет порядок однозначно

Оконные функции требуют предварительно отсортированных данных, так что результат запроса будет отсортирован согласно тому или иному предложению PARTITION BY/ORDER BY оконных фукнций. Чтобы результаты сортировались определенным образом, необходимо явно добавить предложение ORDER BY на верхнем уровне запроса

Notation 4.11. Порядок выполнения запроса

1. Взять нужные таблицы (FROM) и соединить их при необходимости (JOIN)
2. Отфильтровать строки (WHERE)
3. Сгруппировать строки (GROUP BY)
4. Отфильтровать результат группировки (HAVING)
5. Сформировать конкретные столбцы результата (SELECT)
6. Рассчитать значения оконных функций (FUNCTION() OVER WINDOW)
7. Отсортировать то, что получилось (ORDER BY)

4.3 Оконные функции

Notation 4.12. Типы оконных функций

- Агрегатные функции
  - SUM – возвращает сумму значений в столбце
  - COUNT – возвращает количество значений в столбце
  - AVG – возвращает среднее значение в столбце
  - MAX – возвращает максимальное значение в столбце
  - MIN – возвращает минимальное значение в столбце

  Значения NULL не учитываются. Использование ключевого слова DISTINCT не допускается с предложением OVER
- Ранжирующие функции
  - ROW_NUMBER – возвращает номер строки в окне
  - RANK() – возвращает ранг каждой строки. Строки которые имеют одинаковые значения в столбцах, по которым выполняется сортировка, получают одинаковый ранг с пропуском следующего значения
  - DENSE_RANK() – возвращает ранг каждой строки. В отличии от функции RANK, она для одинаковых значений возвращает ранг, не пропуская следующий
  - NTILE(n) – позволяет разделить упорядоченные строки в секции на заданное количество ранжированных групп. Возвращает для каждой строки номер группы

  Наличие ORDER BY обязательно
- Функции смещения
  - LAG или LEAD – функция LAG обращается к данным из предыдущей строки окна, а LEAD – к данным из следующей строки. Имеют трип параметра: столбце, значение которого необходимо вернуть; количество строк для смещения (по умолчанию 1); значение, которое необходимо вернуть если после смещения возвращается значение NULL
  - FIRST_VALUE или LAST_VALUE – позволяют получить превое и последнее значение в окне. В качестве параметра принимают столбец, значение которого необходимо вернуть
  - NTH_VALUE – возвращает значение из n-й строки указанного столбца в упорядоченном разделе результирующего набора
- Аналитические функции

  Возвращают ифнормацию о распределении данных и используются для статистического анализа
  - CUME_DIST – возвращает относительный ранг текущей строки в окне
  - PERCENT_RANK – возвращает относительный ранг текущей строки в окне

  Функции распределения рангов вычисляют относительный ранг текущей строки в секции окна, который выражается числом от 0 до 1 (процент)

Definition 4.14. ROW_NUMBER

Позволяет:

- Задать нумерацию, которая будет отличаться от порядка сортировки строк результирующего набора
- Создать несквозную нумерацию, т.е. выделить группы из общего множества строк и пронумеровать их отдельно для каждой группы
- Использовать одновременно несколько способов нумерации, так как нумерация не зависит от сортировки строк основного запроса

4.4 Использование подзапросов

Notation 4.13. Запросы по типу возвращаемого результата

- Скалярные (Scalar Subqueries) – возвращают одно значение
- Многозначные (Multi-Valued Subqueries) – возвращают список значений
- Табличные (Table-Valued Subqueries) – возвращают таблицу

Notation 4.14. Использование подзапросов

Предложение	Scalar Subquery	Multi-Valued Subquery	Table-Valued Subquery
SELECT	Вычисляемый столбец	-	-
FROM	-	Derived Table (именованная таблица из одного столбца)	Derived Table (именованная таблица), CTE (временное представление)
WHERE	Фильтрация записей на основе сравнения со скалярной величиной	Фильтрация записей с использованием предикатов [NOT] IN, SOME, ALL, ANY, [NOT] EXISTS*	Фильтрация записей с использованием предиката [NOT] EXISTS*
GROUP BY	-	-	-
HAVING	Фильтрация групп на основе сравнения со скалярной величиной	-	-
ORDER BY	-	-	-

Example 4.1. Использование скалярного вложенного подзапроса

```
SELECT orderid, productid, unitprice, qty
FROM "Sales"."OrderDetails"
WHERE orderid =
    (SELECT MAX(orderid) FROM "Sales"."Orders");
```

Если вложенный подзапрос возвращает пустую выборку (empty set), реузультат конвертируется в NULL

Example 4.2. Использование многозначного вложенного запроса с одним столбцом

Список заказов, сделанных клиентами из 'Mexico':

```
SELECT custid, orderid
FROM "Sales"."Orders"
WHERE custid IN (SELECT custid
                 FROM "Sales"."Customers"
                 WHERE country = 'Mexico');
```

Example 4.3. Использование многозначного вложенного запроса с несколькими столбцами

Вы можете использовать IN (ANY, ALL, SOME) для сравнения по нескольким столбцам. В предложении WHERE необходимо сгруппировать имена стоолбцов в скобках. Количество сгруппированных столбцов должно соответствовать целевому списку подзапроса, и иметь те же типы данных

```
SELECT custid, orderid, shipcity
FROM "Sales"."Orders"
WHERE (custid, shipcity) != ALL (SELECT custid, city
                                 FROM "Sales"."Customers")
AND custid = 76;
```

Example 4.4. Использование скалярного коррелированного запроса

Список заказов с общей суммой каждого заказа

```
SELECT orderid, o.custid,
       (SELECT SUM(unitprice * qty)
        FROM "Sales"."OrderDetails" od
        WHERE od.orderid = o.orderid) AS OrderTotal
FROM "Sales"."Orders" o;
```

Example 4.5. Использование предиката EXISTS

Если вложенный запрос возвращает хотя бы одну запись – EXISTS возвращает TRUE, иначе FALSE

```
SELECT custid, companyname
FROM "Sales"."Customers" AS c
WHERE [NOT] EXISTS (
    SELECT 1
    FROM "Sales"."Orders" AS o
    WHERE o.custid = c.custid);
```

4.5 Использование табличных выражений

Definition 4.15. Производные таблицы (Derived Tables)

Производные таблицы – именованные выражения запроса, созданные во внешнем операторе SELECT. Не хранится в БД – представляет собой виртуальную реляционную таблицу. Область видимости – запрос, в котором она определена
Производная таблица должна:

- Иметь псевдоним
- Иметь уникальные имена для всех столбцов
- Не ссылаться несколько раз в одном запросе

Производная таблица может:

- Использовать внутренние или внешние псевдонимы столбцов
- Быть вложенной в другие производные таблицы

Definition 4.16. Общие табличные выражения (CTE)

CTE (Common Table Expression) – именованные табличные выражения, используемые для разбиения сложных запросов на простые части. Можно представить как определения временных таблиц, существующих только для одного запроса
CTE определяется в предложении WITH. Все возвращаемые столбцы должны иметь уникальные имена. CTE поддерживают несколько определений. На CTE допустимо ссылаться несколько раз только в одном запросе

Definition 4.17. Рекурсивный запрос WITH

Для определния рекурсивного запроса WITH используется указание RECURSIVE
Рекурсивный запрос WITH включает:

- Не рекурсивную часть
- Оператор UNION или UNION ALL
- Рекурсивную часть

Только в рекурсивной части можно обратиться к результату запроса

Definition 4.18. Подзапросы LATERAL

При необходимости объединить записи исходной таблицы с записями, которые возвращает коррелированный табличный подзапрос или табличная функция необходимо использовать операторы LATERAL. CROSS JOIN LATERAL – аналог INNER JOIN, а LEFT JOIN LATERAL – LEFT JOIN
Коррелированный табличный подзапрос должен быть указан справа от оператора LATERAL

5 Лекция 5. Проектирование

5.1 Основные подходы к проектированию ИС

Notation 5.1. Основные этапы жизненного цикла ИС

- Определение требований (анализ). Концептуальные модели данных и бизнес проектов. Определение качественных характеристик
- Проектирование. Логические модели приложений и баз данных для реализации требований. Определение архитектуры
- Реализация. Физические модели приложений и баз данных
- Тестирование и внедрение. Установка технических средств, настройка системы, обучение персонала, пробное использование итд
- Функционирование. Обслуживание и администрирование, в т.ч. внесение изменений (доработка)

Notation 5.2. Использование моделей в процессе проектирования

Модель – это по существу упрощенное абстрактное представление сложного реального объекта (процесса, явления или системы), призванное ответить на вопросы относительно него
Охватить сложный объект целиком – представить его с помощью единой модели – вряд ли получится, поэтому каждая модель фокусируется на отдельных, важных для конкретной задачи, свойствах исследуемого объекта

Notation 5.3. Этапы и модели

Проектирование системы на всех этапах разработки должно быть привязано к процессам предметной области (технологическому, бизнес-процессу), особенно на этапе разработки концептуальной модели

Концептуальная модель	Структурно-функциональное моделирование (IDEF0, IDEF3, DFD)	$\Leftrightarrow$	Объектно-ориентированное моделирование (UML)
		$\Updownarrow$	
Логическая модель		Информационное моделирование (IDEF1X)	
		$\Updownarrow$	
Физическая модель		Система управления БД	

Notation 5.4. Основные этапы и методы

Этап	Методы
Разработка концептуальной модели ИС	Структурно-функциональное и/или объектно-ориентированное моделирование
Разработка логической модели ИС	Информационное моделирование (создание диаграммы сущность-связь)
Разработка физической модели и программного обеспечения ИС	Реализация объектов логической модели, разработка программного кода
Тестировка и отладка ИС	Корректировка программного обеспечения
Эксплуатация ИС	Поддержка ИС после ввода в эксплуатацию

Notation 5.5. Подходы к проектированию

Функционально-ориентированный	Объектно-ориентированный
Функциональная декомпозиция системы с последующим уточнением структуры данных	Представление системы в терминах взаимодействия объектов (классов объектов)
Первичны – процессы (функции). Менее стабильны	Первичны – объекты (данные). Более стабильны
Связывание динамической и статической проекций системы затруднено	Изначально связанные динамическая и статическая проекции
Методы создания моделей (отсутствие свободной трактовки/интерпретации)	Рекомендации, лучшие практики итп
Восприятие экспертами предметной области – лучше	Восприятие экспертами предметной области – хуже

5.2 Основные этапы проектирования баз данных

Notation 5.6. Важность

От того, насколько успешно будет спроектирована БД, зависит
- Эффективность функционирования проектируемой информационной системы в целом
- Ее жизнеспособность
- Возможность расширения и дальнейшего развития

Поэтому вопрос проектирования баз данных выделяют как отдельное, самостоятельное направление работ при разработке информационных систем

Definition 5.1. Методология проектирования БД

Методология проектирования БД – структурированный подход, предусматривающий использование специализированных процедур, технических приемов, инструментов, документации и ориентированный на поддержку и упрощение процесса проектирования
Методология проектирования предусматривает разбиение всего процесса на несколько стадий, каждая из которых, в свою очередь, состоит из нескольких этапов. На каждом этапе разработчику предлагается набор технических приемов, позволяющих решать задачи, стоящие перед ним на данной стадии разработки

Definition 5.2. Проектирование базы данных

Цель проектирования базы данных – это корректное отображение выбранной для автоматизации предметной области
Выделяют два основных подхода к данному процессу:
- Нисходящий (сверху вниз)
  Сначала происходит изучение целого, или описание предметной области, а затем уже осуществляется разделение целого на составные части, каждая из которых после этого также последовательно изучается
- Восходящий (снизу вверх)
  Сначала происходит описание составных частей, полсе чего собирается общая картина

Definition 5.3. Нисходящее проектирование

- Анализ предметной области. Описание внешнего уровня базы данных. Результат – внешняя модель предметной области
- Концептуальное (инфологическое) проектирование. Анализ требований к данным. Результат – обобщенная концептуальная модель предметной области
- Логическое проектирование. Преобразование требований к данным в СУБД ориентированную структуру БД. Реузльтат – логическая модель данных
- Нормализация. Корректировка логической модели для устранения возможных аномалий данных. Результат – нормализованная логическая модель данных
- Формирование физической (внутренней) модели базы данных. Определение особенностей хранения данных, методов доступа итд. Создание схемы БД для конкретной СУБД и программно-аппаратной платформы. Результат – внутренняя модель БД

5.3 Анализ предметной области

Notation 5.7. Анализ предметной области

Проводится анализ предметной области в целом с последующей декомпозицией на отдельные элементы
Результат – описание внешнего уровня базы данных (внешняя модель) для пользователей, которое абстрагируется от особенностей реализации

- Назначение ИС и границы проекта: абстрагирование от неважного, сосредоточение на важном
- Определение основных пользователей системы (действующих лиц) и их целей
- Определение бизнес-правил и ограничений (функциональные требования). Отношения между информационными объектами. Ограничения на значения определнных характеристик
- Определение требований обработки данных (нефункциональные требования): защита, время отклика, темпы увеличения объема данных итд

Notation 5.8. Анализ требований

Включает в себя три типа деятельности:

- Сбор требований: общение с клиентами и пользователями, чтобы определить, каковы их требования
- Анализ требований: определение, являются ли собранные требования неясными, неполными, неоднозначными, или противоречащими, и затем решение этих проблем
- Документирование требований: требования могут быть задокументированы в различных формах, таких как простое описание, сценарии использования, пользовательские истории, или спецификации процессов

Example 5.1.

Пример начального описания предметной области:
Предприятие общественного питания предлагает своим посетителям набор блюд, изготавливаемых по авторским рецептам. Каждое блюдо имеет уникальное название, относится к единственному виду и приготавливается по единственному рецепту. Каждое блюдо состоит из набора продуктов, в соответствии с рецептом
Компания имеет широкую сеть организаций поставщиков продуктов, расположенных в различных городах
Пример определения действующих лиц:

Действующие лица	Бизнес-цели
Сотрудник кухни	Изготовление блюда: описание рецепта, информация о количестве порций, сведения о доступности продуктов
Сотрудник по работе с поставщиками	Взаимодействие с поставщиками продуктов: информация о наличии продуктов, информация о поставщиках конкретных продуктов
Сотрудник по реализации готовой продукции	Планирование и реализация готовых продукции (блюд): формирование заявки на порции согласно меню конкретного дня, реализация порций блюд в соответствии с заказом

Определение бизнес-правил и ограничений:
- Каждое блюдо должно быть изготовлено строго по одному рецепту
- При изготовлении блюда в конкретный день каждый продукт может быть взят только из одной поставки
- Поставки продукта могут использоваться в нескольких блюдах в разные дни
- Поставщик может осуществлять в один и тот же день только одну поставку каждого продукта
- Каждый поставщик осуществляет поставку только определенных продуктов
- В одной поставке могут присутствовать продукты поставляемые разными поставщиками
- ИТД

5.4 Концептуальное (инфологическое) проектирование

Definition 5.4. Концептуальное проектирование

Концуптуальная модель – это формализованное описание предметной области на самом высоком уровне абстракции (без привязки к физическому представлению и хранению данных), выполненное без жесктйо ориентации на программные и технические средства
Концептуальная модель отражает специфику предметной области, а не структуру БД
Чтобы БД адекватно отражала предметную область, проектировщик должен хорошо представлять себе все нюансы, присущие ей, и уметь отобразить их в БД
Цель – построение независмой от СУБД информационной структуры путем объединения информационных требований пользователей
Результат – концептуальная (инфологическая) модель (семантическая, ER-модель)

Notation 5.9. Реализация

Обобщение полученных на этапе анализа предметной области сведений
Моделирование частных представлений пользователей с использованием диаграмм сущность-связь (Entity-Relationship Diagrams, ER-диаграмм):

- Определение сущностей
- Определение атрибутов сущностей
- Идентификация ключевых атрибутов сущностей
- Определение связей между сущностями

Интеграция частных представлений: концептуальные требования отдельных пользователей, определенные в стандартной форме (диграммы сущность-связь), сливаются в единое глобальное представление. При этом должны быть выявлены и ликвидированы противоречивые и избыточные данные
Уточнение требований: устранение противоречивости и избыточности данных. Принцип бритвы Оккама: не следует множить сущности без необходимости

Notation 5.10. Требования предъявляемые к инфологической модели

- Адекватное отображение предметной области
- Недопущение неоднозначной трактовки модели
- Представление данных с минимальным дублированием
- Легкая расширяемость, обеспечивающая ввод новых данных без изменения ранее определенных, то же относят и к удалению данных
- Легкое восприятие различными категориями пользователей. Желательно чтобы инфологическую модель строил (или хотя бы участвовал в ее создании) специалист, работающий в данной предметной области, а не только проектировщик систем машинной обработки данных

5.5 Формирование даталогической модели базы данных

Definition 5.5. Формирование даталогической модели базы данных

Даталогическая модель БД – модель, отражающая логические взаимосвязи между элементами данных (отношениями) безотносительно их содержания и физической организации. Интегрированный набор понятий для описания и обработки данных, связей между ними и ограничений, накладываемых на данные в рамках предметной области
Даталогическая модель (модель данных) разрабатывается на основе концептуальной (информационно-логической) модели предметной области с учетом: конкретной реализации СУБД, специфики конкретной предметной области – возможных запросов к БД на основе ранее выявленных потребностей пользователей системы

5.6 Нормализация

Definition 5.6. Нормализация

Нормализация – формальная процедура, проверки и реорганизации отношений, в ходе которой создается оптимизированная структура БД, позволяющая избежать различных видов аномалий
Нормализация подразумевает декомпозицию (разделение) отношения на два или более, обладающих лучшими свойствами при добавлении, изменении и удалении данных

Notation 5.11. Задачи нормализации

Исключение необходимости хранения повторяющейся информации
Создание структуры, в которой предусмотрена возможность ее будущих изменений и с минимальным влиянием на приложения. Реализация логической и физической независимости

5.7 Физическое проектирование

Definition 5.7. Физическое проектирование

Цель – создание физической структуры БД (и ее объектов), а также набора реализуемых алгоритмов обработки данных с целью эффективного использования вычислительных ресурсов (дисковой памяти, времени центрального процессора)
Физическая структура БД соответствует понятию внутренней модели и учитывает специфику конкретной СУБД:

- Ограничения на именование объектов базы данных
- Ограничения на поддерживаемые типы данных
- Требования к индексным структурам
- Выбор методов управления дисковой памятью
- Разделение БД по файлам и устройствам
- Методы доступа к данным

Результат – полностью готовая к внедрению структура базы данных и набор реализуемых алгоритмов по ее использованию

Notation 5.12. Показатели эффективности функционирования физической БД

- время ввода-вывода – время, затрачиваемое на выборку данных с внешней памяти в оперативную, включая время передачи данных
- Время доступа – промежуток времени между выдачей команды, содержащей обращение к некоторым данным, и фактически получением данных для обработки
- Время отклика – промежуток времени между вводом запроса к базе данных в компьютер и завершением обработки запроса с представлением результатов

Notation 5.13. файловые струткуры для хранения информации в БД

В каждой СУБД по-разному организованы зранение и доступ к данным, однако существуют некоторые файловые структуры, которые имеют общепринятые способы организации и широко применяются праткически во всех СУБД

Remark 5.1. Внутренние (физические) модели данных

Зависят от особенностей конкретной СУБД и определяют:

- специальные структур данных, поддерживаемые СУБД: индексно-организованные таблицы, кластеры, секционированные таблицы итд
- Взаимосвязь между логическими и физическими структурами хранения данных: логические (например, табличное пространство, сегмент, экстент, блок данных БД) и физические (файл данных, страница данных)
- Способы распрпеделения данных на физических устройствах хранения: один/несколько независимых дисков, ленточные устройства, RAID-массивы итд
- Способы доступа к данным: полное чтене таблицы, чтение индекса с пропусками, чтение кластера

Notation 5.14. Файловая струткура БД

Таблица – логический элемент БД, физически его нет
Все данные хранятся в файлах, а фалйы организуются в группы
Файлы подразделяются на файлы данных и файлы журнала транзакций. Файлы могут лежать на разных дисках при нехватке места или необходимости распределенного хранения
Файлы разделяются на блоки – страницы. Размер блока зависит от СУБД. В блоке может быть одна или несколько записей. За распределение записей по файлам отвечает СУБД

Notation 5.15. Формирование реляционной БД из ER-диаграмм

1. Каждая сущность превращается в таблицу. Имя сущности становится именем таблицы
2. Каждый атрибут становится столбцом с тем же именем
3. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы
4. Связи многие-к-одному (и один-к-одному) становятся внешними ключами

5.8 Автоматизация проектирования баз данных

Definition 5.8. CASE-технология

CASE-технология (Computer-Aided Software/System Engineering) – это инструментарий для системных аналитиков, разработчиков и программистов, позволяющий автоматизировать процесс проектирования и разработки ПО. Совокупность методологий анализа, проектирования, разработки и сопровождения сложных систем, поддерживаемая комплексом взаимосвязанных средств автоматизации
Современные CASE-средства охватывают обширную область поддержки многочисленных технологий проектирования ИС: от простых средств анализа и документирования до полномасштабных средств автоматизации, покрывающих весь жизненный цикл ПО

Definition 5.9. CASE-средства

Современный рынок программных средств насчитывает около 300 различных CASE-средств
CASE-средства можно классифицировать по:

- Применяемым методологиям и моделям систем и БД
- Степени интегрированности с СУБД
- Доступным платформам

Notation 5.16. Классификация по типа

- Средства анализа – предназначены для построения и анализа моделей предметной области
- Средства анализа и проектирования – поддерживают наиболее распространенные методологии проектирования и используются для создания проектных спецификаций на компоненты и интерфейсы системы, архитектуру системы, алгоритмы и структуры данных
- Средства проектирования баз данных – обеспечивают моделирование данных и генерацию схем баз данных (как правило, на языке SQL) для наиболее распространенных СУБД (прямое проеткирование)
- Средства разработки приложений – средства реинжиниринга, обеспечивают анализ программных кодок и схем баз данных и формируют на их основе различные модели и проектные спецификации (обратное проектирование)

Notation 5.17. CASE-средства проектирования БД

- Независимые от СУБД – поддерживают различные платформы, позволяя генерировать структуру БД для различных СУБД
- Встроенные – специализированы и ориентированы на конкретную СУБД в состав которой входят

Notation 5.18. Состав CASE-средств разработки БД

Графические средства создания и редактирования ER-модели
Средства автоматического отображения ER-модели в реляционную
Средства создания на основе полученной реляционной модели SQL-скриптов для создания СУБД-ориентированной БД: таблиц БД и связей между ними
Реальная БД может быть создана непосредственно системой – целевая СУБД должна быть доступна для системы – или с помощью сгенерированного сценария – разработчик получает файл сценария на диалекте выбранной СУБД и выполняет этот сценарий на целевом сервере

Remark 5.2. Преимущества и недостатки

Преимущества:

- CASE-средства незаменимы при создании больших проектов группой проектировщиков
- Использование CASE-средства позволяет сократить срок разработки
- Использование CASE-средства позволяет снизить число возможных ошибок

Недостатки:

- Позволяют моделировать только бинарные связи между сущностями
- Генерируют отношения по упрощенным правилам, не всегда учитывая класс принадлежности сущности связи

5.9 Диаграммы сущность-связь

Definition 5.10. ER-модель

Модель сущность-связь впервые была предложена в 1976 г. Питером Ченом
ER-модель представляет собой описание предметной области в терминах сущность (объект) – атрибут (свойство) – связь
Для графического представления ER-модели используется ERD: детализация хранилищ данных проектируемой системы, документирование сущностей системы и способов их взаимодействия

Notation 5.19. Основные понятия ER-модели

- Сущность
- Экземпляр сущности
- Атрибут
- Ключ
- Связь

5.10 Сущность

Definition 5.11. Сущность и экземпляр сущности

Сущность – именованное множество подобных уникальных экземпляров – объектов, событий, концепций – информация о которых должна сохраняться в базе данных. Имя сущности – существительное в единственном числе
Экземпляр сущности – конкретный представитель данной сущности

Notation 5.20. Выделение сущностей

Для формирования начального списка сущностей:

- В списке имен исходного материала выбираем все существительные
- Отвечаем на вопросы, которые касаются каждой возможной сущности:
  - Может ли она быть описана, т.е. обладает ли она набором характеристик?
  - Существует ли более одного экземпляра этой сущности?
  - Может ли один экземпляр этой сущности быть отделен от другого, т.е. идентифицирован?
  - Называет или описает это что-либо? (Если этот ответ положительный, то это скорее атрибут, чем сущность)

Анализ является субъективным процессом, поэтому различные разработчики могут создавать разные, но вполне допустимые интерпретации одного и того же факта

Notation 5.21. Типы сущностей

- Независимая (сильная, родительская, основная). Каждый экземпляр сущности может быть однозначно идентифицирован без определения его отношений с другими сущностями. Уникальность экземпляра определяется только собственными атрибутами
- Зависимая сущность (слабая, дочерняя, подчиненная). Однозначная идентификация экземпляра сущности зависит от его отношения к другой сущности. Существование экземпляра сущности зависит от существования экземпляра родительской сущности

5.11 Атрибуты

Definition 5.12. Атрибут

Атрибут – именованное и неотъемлемое свойство всех экземпляров сущности, предназначенное для классификации, идентификации, количественной характеристики и выражения состояния сущности
Атрибут выражает некоторое отдельное свойство сущности, которое характеризует ее экземпляр

Notation 5.22. Правила атрибутов

Атрибуты неотъемлемая бизнес-характеристика сущности
Сущность может обладать любым количеством атрибутов

- Каждый атрибут идентифицируется уникальным в пределах сущности именем
- Для каждого экземпляра сущности должно существовать значение кжадого его обязательного атрибута
- Ни один экземпляр сущности не может обладать более чем одним значением для связанного с ней атрибута
- Значение атрибутов не должны слишком часто меняться

Notation 5.23. Определение состава атрибутов

На первом этапе необходимо выделить только те атрибуты, которые изначально имеются у сущности (если ее рассматривать отдельно от всех остальных)
Самый простой способ определения атрибутов – после идентификации сущности или связи, задать себе вопрос: какую информацию требуется хранить о . . . ? Если возникает необходимость добавить атрибут(ы) для связи, это указывает на тот факт, что связь является сущностью. После установки связей у сущности могут появиться атрибуты характеризующие связь
Существенно помоьч в определении атрибутов могут различные бумажные и электронные формы и документы, используемые в организации при решении задачи

Notation 5.24. Виды атрибутов

- Составные/простые (атомарные). Составной – состоит из нескольких компонентов (ФИО, адрес), простой – состоит из одного компонента с независимым существованием (цена, дата рождения). Степень атомарности атрибутов определяется разработчиком и зависит от характера работы с данными (задач)
- Обязательные/необязательные. Значение обязательного атрибута должно быть определено для любого возможного экземпляра сущности, необязательные атрибуты допускают отстуствие значения
- Однозначные/многозначные. Однозначный – содержит только одно значение для одного экземпляра сущности, многозначный – может содержать несколько значений
- Базовый/производный (вычисляемые). Значение атрибута может быть определено по значениям других атрибутов
- Ключевые/неключевые. Ключевой (идентфиицирующий) – служит для уникальной идентфикации экземпляра сущности, не ключевой (описательный) – используется для определения интересующих пользователя свойств
- Изначально присущие сущности (бизнес атрибутов)/появившиеся в результате проектирования (суррогатные ключи, связи)

Notation 5.25. Типы данных и домены

Для каждого атрибута должны быть определены:

- Тип данных атрибута – характеризует вид хранящихся данных (числовой, текстовый, логический, временной и др.)
- Домен – множество допустимых значений атрибута, отвечающих бизнес-правилам предметной области

5.12 Ключи

Definition 5.13. Определение ключей

На основании выделенного множества атрибутов для сущности определяется набор ключей
Ключ – один или несколько атрибутов сущности, служащих для однозначной идентификации ее экземпляров или для их быстрого поиска
Выдялеют следующие типы ключей:

- Потенциальный ключ (candidate key)
- Первичный ключ (primary key)
- Альтернативный ключ (alternate key)
- Суррогатный ключ (surrogate key)

Definition 5.14. Ключи

Потенциальный ключ – минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся. Значение ключа уникально для каждого экземпляра сущности. Сущность может иметь несколько потенциальных ключей. Если ключ состоит из нескольких атрибутов, то она называется составным ключом
Первичный ключ – потенциальный ключ, который выбран в качестве основного идентфикатора, уникально идентифицирующего экземпляр сущности
Альтернативный ключ – потенциальный ключ, не ставший первичным
Суррогатный ключ – дополнительный служебный атрибут, добавленный к имеющимся бизнес-атрибутам, единственное предназначаение которого – служить первичным ключом

Notation 5.26. Характеристики хорошего первичного ключа

- Должен быть уникальным
- Должен быть обязательным (иметь значения для всех возможных экземпляров сущности)
- Должен быть минимальным не только по количеству атрибутов, но и по размеру (суммарной длине значения)
- Должен быть постоянным – значения такого атрибута не должны меняться в течение всего времени существования экземпляра сущности
- Должен обеспечивать высокую скорость поиска нужного экземпляра сущности (не должен быть текстовым)

Remark 5.3. Определение ключей

Каждая сущность должна обладать первичным ключом
Каждая сущность может обладать любым числом альтернативных ключей
Ключ может состоять из одого (простой ключ) или нескольких атрибутов (составной ключ)
Отдельный атрибут может быть частью более чем одного ключа (первичного или альтернативного)

5.13 Связь

Definition 5.15. Связь

Связь – ассоциирование экземпляров двух или более сущностей (или различных экземпляров одной и той же сущности). Одно из основных требований к организации базы данных – это обеспечение возможности поиска экземпляров одних сущностей по значениям других, для чего необходимо установить между ними определенные связи
Типы связей:

- Связь 1:1. Один экземпляр сущности одного класса связан с одним экземпляром сущности другого класса
- Связь 1:M. Один экземпляр сущности одного класса связан со многими экземплярами сущности другого класса
- Связь M:N. Несколько экземпляров сущности одного класса связаны с несколькими экземплярами сущности другого класса

Definition 5.16. Внешний ключ

Внешние ключи (Foreign Key) – это атрибуты, предназначенные для определения связи между сущностями. Сущность может обладать любым количеством внешних (наследуемых) атрибутов
Связь всегда реализуется между двумя сущностями. Одна сущность выступает в роли родительской, а другая в роли дочерней
Для поддержки связей обе сущности должны содержать наборы атрибутов, по которым они связаны: в родительской сущности – первичный ключ или альтернативный ключ. В дочерней сущности – внешний ключ. Внешний ключ должен по составу и типу атрибутов соответствовать первичному ключу, родительской сущности

Notation 5.27. Характеристики связи

- Имя связи – как правило определяется глагольной фразой. Определяет семантику связи и выражает некоторое бизнес правило. Связь может быть поименована от родителя (обязательно) и от потомка
- Мощность (кардинальность) связи – служит для обозначения количества экземпляров (значений) дочерних сущностей связанных с соответствующим значением родительской сущности
- Модальность связи – описывает ее обязательность: может и должен. Может – указывает, что экземпляр сущности может не связываться с экземпляром другой сущности. Должен – обязан связаться минимум 1 раз
- Мощность и модальность связи выражают существующие ограничения предметной области

Notation 5.28. Тип связи

- Идентифицирующая (сильная) связь. Определяет такое отношение между двумя сущностями, в котором для идентификации каждого экземпляра подчиненной сущности требуются значения атрибутов родительской сущности. Экземпляр подчиненной сущности не может существовать или не имеет смысла без экземпляра родительской сущности. Атрибуты, составляющие первичный ключ родительской сущности, входят в состав первичного ключа дочерней сущности
- Не идентифицирущая (слабая) связь. Определяет такое отношение между двумя сущностями, в котором каждый экземпляр подчиненной сущности не зависит от значений атрибутов родительской сущности. Экземпляр подчиненной сущности может существовать (имеет смысл) без экземпляра родительской сущности. Атрибуты первичного ключа родительской сущности являются неключевыми атрибутами дочерней сущности

Notation 5.29. Категориальная связь

Если две и более сущностей имеют общие по смыслу атрибуты, либо когда сущности имеют общие по смыслу связи, можно применять в модели иерархию наследования (категорий), включающую в себя суперклассы и подклассы. Не существует в реляционной модели, поэтому в ходе дальнейшего проектирования должна быть разрешена
Суперкласс (родительская сущность, супертип) – сущность, включающая в себя подклассы (категории, подтипы)
В роидетльской сущности вводится атрибут-дискриминатор, позволяющий распределять ее экземпляры по категориям

Notation 5.30. Виды категорий

- Эксклюзивная – каждому родительскому экземпляру может соответствовать экземпляр только в одном из потомков
- Не эксклюзивная – каждому родительскому экземпляру могут соответствовать экземпляры нескольких потомков
- Полная – выделены все возможные подтипы: одному родительскому экземпляру обязательно соответствует экземпляр в каком-либо потомке
- Неполная – определены только некоторые из возможных подтипов: могут существовать родительские экземпляры, которые не имеют соответствия экземпляров в потомках

Definition 5.17. Рекурсивная связь

Связь между различными экземплярами одной и той же сущности. Всегда не идентифицирующая. Атрибут внешнего ключа должен быть необязательным (может содержать NULL)

Remark 5.4. Требования предъявляемые к ИЛМ

- Адекватное отображение предметной области
- Недопущение неоднозначной трактовки модели
- Представление данных с минимальным дублированием
- Легкая расширяемость, обеспечивающая ввод новых данных без изменения ранее определенных, а также удаление одних элементов данных без потери других
- Легкое восприятие различными категориями пользователей
- Применимость языка описания модели как при ручном, так и при автоматизированном проектировании информационных систем

Definition 5.18. Концептуальное проектирование

Цель – построение независимой от СУБД информационной структуры путем объединения информационных требований пользователей
Результат – концептуальная (инфологическая) модель (семантическая, ER-модель)

5.14 Обеспечение корректности и целостности данных

Definition 5.19. Обеспечение корректности и целостности данных

Корректность данных – соответствие определенным ограничениям предметной области. Обеспечивается правилами валидации значений атрибутов (простые условия) или триггерами СУБД (сложные условия, нереализуемые в СУБД через заданные правила для атрибута, т.е. на уровне ограничения столбца таблицы)
Полнота данных – значение для всех обязательных атрибутов определены. Может обеспечиваться заданием значений атрибутов по умолчанию
Целостность данных – согласованное представление информации для связанных сущностей. Значений одинаковых атрибутов связанных сущностей совпадают. Обеспечивается правилами ссылочной целостности или на уровне триггеров СУБД (для правил, неподдерживаемых СУБД на уровне ограничений внешнего ключа)

Notation 5.31. Общий алгоритм разработки ER-модели

1. Выделение сущностей
2. Определение атрибутов сущностей
3. Определение первичных ключей
4. Определение связей между сущностями
5. Определение правил поддержки ссылочной целостности, правил проверки значений атрибутов и значений по умолчанию для атрибутов

Данный алгоритм является итерационным (указанная последовательность шагов может выполняться несколько раз в процессе разработки, постепенно уточняя модель). Пункты 2-3 и 4 на практике можно поменять местами

5.15 Нормализация реляционной базы данных

Definition 5.20. Нормализация

Нормализация – формальная процедура, проверки и реорганизации отношений, в хоже которой создается оптимизированная структура БД, позволяющая избежать различных видов аномалий
Нормализация подразумевает декомпозицию (разделение) отношения на два или более, обладающих лучшими свойствами при добавлении, изменении и удалении данных

Notation 5.32. Задачи нормализации

- Исключение из таблиц повторяющейся информации
- Создание структуры, в которой предусмотрена возможность ее будущих изменений и с минимальным влиянием на приложения. Реализация логической и физической независимости

Definition 5.21. Дублирование данных

Дублирование заключается в хранении идентичных данных
- Необходимое. Важен сам факт идентичности данных – для поддержки ссылочной целостности (связей между сущностями)
- Избыточное. Приводит к проблемам при обработке данных (аномалиям), которые не являются неразрешимыми в принципе, но существенно усложняют обработку. При избыточном дублировании могут возникать противоречия данных (возможность вывода двух взаимоисключающих утверждений на основе имеющихся данных)

Definition 5.22. Аномалии

Аномалия – такая ситуация в БД, которая приводит к противоречиям в данных, либо существенно усложняет их обработку
- Аномалия модификации данных. Необходимость изменения одного значения приводит к необходимости просмотра всей таблицы (для исключения противоречий)
- Аномалия удаления данных. Состоит в том, что удаление данных из таблицы приводит к удалению информации не связанной напрямую с удаляемыми данными
- Аномалия вставки данных. Нельзя добавить одни данные без наличия других или же для вставки данных необходимо просмотреть всю таблицу (для исключения противоречий)

Основная причина аномалий – хранение в одном отношении разнородной информации

Тут важно посмотреть презентацию: лекция 5. Нормализация (до следующей лекции)

Example 5.2. Примеры избыточности и аномалий различных видов

- Избыточность: для каждого сотрудника отдела повторяются данные об отделе и должности
- Аномалия модификации: если какой-либо отдел решает переименовать необходимо изменить значение атрибута Наим_отдела для многих экземпляров сущности сотрудник
- Аномалия удаления: если все сотрудники какого-либо отдела будут уволены, информация об отделе также будет утеряна
- Аномалия вставки: если при внесении данных о каком-либо сотруднике оператор совершит ошибку и внесет неверное наименование отдела (оставив верный номер), будет не ясно, какая из строк БД содержит правильную информацию (в базе будут содержаться противоречивые сведения). Вставка сведений о новом отделе компании невозможна без наличии сведений о хотя бы однои его сотруднике

Notation 5.33. Описание предметной области

Используется далее для иллюстрации проведения нормализации

- Каждая группа на кафедре учится по определенной специальности
- Нумерация групп – сквозная: не может быть групп с одинаковыми номерами, обучающихся на разных специальностях
- Учебный план специальности определяет дисциплины, изучаемые в группах этой специальности
- Одна и та же дисциплина может встречаться в учебных планах групп разных специальностей
- Каждую дисциплину может вести как один преподаватель, так и несколько (даже в рамках одной группы)
- Каждый преподаватель может вести несколько дисциплин, а также одну дисциплину у нескольких групп

Notation 5.34. Имеющиеся аномалии

- Если уволить преподавателя, может быть потеряна информация о дисциплине и даже группе (аномалия удаления)
- При открытии новой специальности, информацию о ней невозможно будет добавить, не определив какие дисциплины на ней будут читаться и кто будет их вести (аномалия вставки)
- При необходимости изменения названия дисциплины придется просматривать все отношение (аномалия модификации)
- При добавлении новой группы необходимо будет измерять структуру таблицы – добавлять новый столбец – разреженность данных

5.16 Первая, вторая и третья нормальные формы (1НФ, 2НФ и 3НФ)

Definition 5.23. Определение 1НФ

Отношение находится в первой нормальной форме тогда и только тогда, когда

1. Определен его первичный ключ
2. Все его атрибуты содержат атомарные значения
3. Нет многозначных атрибутов (повторяющихся групп атрибутов)

Для приведения отношения к 1НФ следует:

- Разделить сложные атрибуты на атомарные
- Повторяющиеся атрибуты совместить
- Пересмотреть состав атрибутов РК

Definition 5.24. Определение 2НФ

Отношение находится во второй нормальной форме (2НФ), тогда и только тогда, когда оно находится в 1НФ и каждый его неключевой атрибут функционально полно зависит от первичного ключа. Если первичный ключ отношения состоит из одного атрибута, оно автоматически находится во 2НФ

Для приведения отношения к 2НФ следует: выделить неключевые атрибуты, которые зависят только от части первичного ключа и перенести их в новое отошение вместе с той частью ключа, от которой они зависят

Definition 5.25. Определение 3НФ

Отношение находится в тертьей нормальной форме (3НФ) тогда и только тогда, когда оно находится во 2НФ и не содержит транзитивных функциональных зависимостей между неключевыми атрибутами и первичным ключом. Ни один неключевой атрибут не должен функционально зависеть от другого неключевого атрибута

Для приведения отношения к 3НФ следует: создать новое отношение и перенести в него атрибуты с одной и той же зависимостью от неключевого атрибута

Тут есть примеры приведения к 2НФ и 3НФ, но для их понимания следует посмотреть презентацию