Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Уфимский государственный нефтяной технический университет»
Кафедра вычислительной техники и инженерной кибернетики

Аналитика данных

Часть 1. Введение в SQL и работа с базами данных

Учебно-методическое пособие

Уфа, 2024

Учебно-методическое пособие к выполнению практических и лабораторных работ по дисциплине «Аналитика данных» предназначено для студентов очной и заочной форм обучения.

Допускается использование данного учебно-методического пособия для проведения занятий по смежным дисциплинам.

В учебно-методическом пособии приведена теория, разбираются примеры и задаются упражнения для самостоятельного выполнение, все это позволяет закрепить на практике теоретические знания.

С помощью данного пособия можно научиться:

- создавать простые запросы;
- создавать отчёты и анализировать данные с помощью SQL;
- извлекать и исследовать данные с помощью Python;
- создавать интерактивные дашборды в Excel и DataLens.

Составители: Яковлева Е.Э., ассистент
Тулупова О.П., к.т.н. доцент

Содержание

1 Модели баз данных

1.1 Базы данных

Базы данных (БД) – это структурная совокупность взаимосвязанных данных определённой предметной области: реальных объектов, процессов, явлений и т. д.

Существует множество определений термина «База данных», и из всех этих определений можно выделить ключевые моменты:

- БД хранятся и обрабатываются в вычислительной системе;
- данные в БД логически структурированы;
- БД включает схему или метаданные, описывающие её структуру.

Первый момент является строгим, а другие допускают различные трактовки и различные степени оценки.

Основная функция базы данных – предоставление единого хранилища для всей информации, относящейся к определённой теме или предметной области.

База данных может содержать любую информацию:

- оценки успеваемости учащихся;
- данные интернет-магазина;
- бюджет страны.

А можно создать небольшую базу и хранить в ней список дел с учётом затраченного времени, что позволит в дальнейшем анализировать собственную эффективность.

1.2 Системы управления базами данных

СУБД – это программа для хранения, обработки, управления и поиска информации в базе данных.

СУБД используются для выполнения различных операций с данными:

- хранения;
- манипулирования;
- обработки запросов к БД;
- выборки;
- сортировки;

- обновления;
- поиска;
- защиты данных от несанкционированного доступа или потери.

Модель данных - это способ организации данных в БД, который определяет структуру данных, их типы, хранение и обработку.

Модели данных и СУБД, поддерживающие эти модели можно разделить на следующие типы:

- иерархические;
- сетевые;
- реляционные;
- объектно-ориентированные;
- noSQL (документно-ориентированные, ключ-значение, колоночные, графовые).

1.2.1 Иерархические

Иерархическая модель базы данных — одна из самых ранних моделей, организующая данные в древовидную структуру с одним корневым узлом, соединенным с несколькими дочерними узлами.

В иерархической модели данные хранятся в виде набора полей, где каждое поле содержит только одно значение. Записи связаны друг с другом через связи в отношениях родитель-потомок.

В иерархической модели базы данных каждая дочерняя запись имеет только одного родителя. Родитель может иметь несколько детей.

Пример графического представления иерархической модели данных БД можно увидеть на рисунке 1.1.

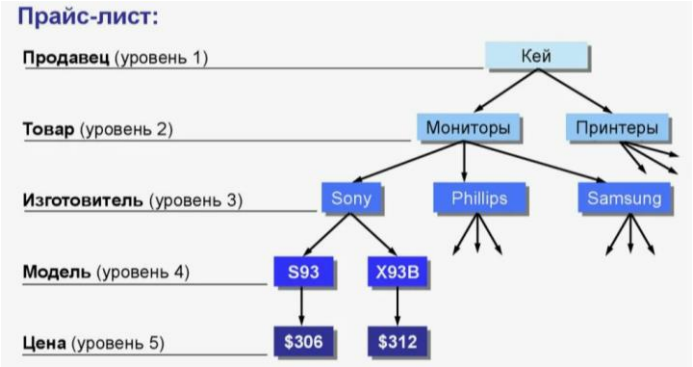

Рисунок 1.1 – Иерархическая модель данных БД[1]

В качестве примера иерархической СУБД можно привести: Information Management System.

1.2.2 Сетевые

Сетевые СУБД используют сетевую структуру для создания отношений между объектами. Сетевые базы данных имеют иерархическую структуру, но в отличие от иерархических баз данных, где у одного дочернего элемента может быть только один родитель, сетевой узел может иметь отношения с несколькими объектами. Сетевая база данных больше похожа на паутину.

Пример графического представления сетевой модели данных можно увидеть на рисунке 1.2.

Рисунок 1.2 – Сетевая модель данных БД[2]

[1]Источник рисунка: https://fairysoft.ru/ierarxicheskaya-baza-danniyx
[2] Источник рисунка: https://domgadalki.ru/foto/setevaya-model-bd

В качестве примеров сетевых СУБД можно привести: IDS, IDMS.

1.2.3 Реляционные

Реляционная СУБД (система управления базами данных) - это вид базы данных, организованной в соответствии с моделью реляционной модели данных. В такой базе данных данные хранятся в виде двухмерных таблиц, которые состоят из строк и столбцов. Каждая строка представляет собой запись, а каждый столбец - атрибуты.

Основные составляющие реляционной СУБД:

1. Таблицы данных - основной способ организации информации в реляционной базе данных.
2. Ключи - уникальные идентификаторы для строк и связей между таблицами.
3. Ограничения целостности - правила, которые гарантируют целостность и консистентность данных.
4. Язык SQL (Structured Query Language — «язык структурированных запросов») - язык программирования для работы с реляционными базами данных.

Отличие реляционной СУБД от других видов:

1. Структура данных - данные хранятся в виде таблиц, что упрощает структурирование и организацию информации.
2. Операции JOIN - возможность объединения данных из разных таблиц для получения нужной информации.
3. Нормализация - процесс организации данных для исключения избыточности и повышения эффективности запросов.
4. Поддержка ACID-свойств - гарантия атомарности, согласованности, изолированности и устойчивости данных.

Пример двумерной таблицы реляционной модели данных можно увидеть на рисунке 1.3.

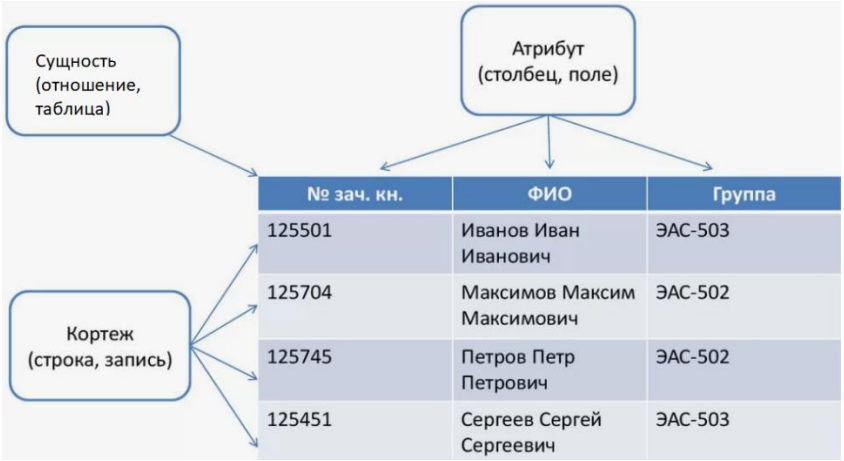

Рисунок 1.3 – Пример таблицы в реляционной модели данных БД[3]

Наиболее популярными реляционными СУБД являются:

1. PostgreSQL от компании PostgreSQL Global Development Group
    a) Postgres Pro от российской компании Postgres Professional.
2. MS Access, MS SQL Server от компании Microsoft Corporation.
3. Oracle, MySQL от компании Oracle Corporation.

1.2.4 Объектно-ориентированные

Объектно-ориентированная СУБД (ООСУБД) - это тип базы данных, который основан на принципах объектно-ориентированного программирования (ООП), где данные представлены в виде объектов с атрибутами и методами.

Составляющие объектно-ориентированной СУБД:

1. Объекты - представление данных в виде объектов с атрибутами и методами.
2. Классы - описание структуры и поведения объектов.

[3] Источник картинки: https://fairysoft.ru/baziy-danniyx-ponyatie-relyatsionnoy-modeli

3. Наследование - возможность создания новых классов на основе уже существующих с использованием их свойств и методов.
4. Полиморфизм - возможность одинаково обращаться к объектам различных классов через общий интерфейс.

Плюсы объектно-ориентированной СУБД:

1. Более наглядное и удобное представление данных.
2. Более гибкая и простая модель данных.
3. Улучшенная переиспользование кода через наследование.
4. Упрощенная разработка и поддержка сложных приложений.

Минусы объектно-ориентированных СУБД:

1. Требуется более высокая квалификация для работы с ООСУБД.
2. Может потребоваться дополнительное время на обучение и разработку приложений.
3. Не подходит для всех типов приложений и задач, например, для простых приложений с небольшим объемом данных.

В сравнении с реляционной СУБД, объектно-ориентированная СУБД, предоставляет более гибкую модель данных, улучшает процесс разработки и поддержки приложений, однако требует большей квалификации и может быть неэффективна для определенных типов приложений.

Пример графического представления объектно-ориентированной модели данных можно увидеть на рисунке 1.4.

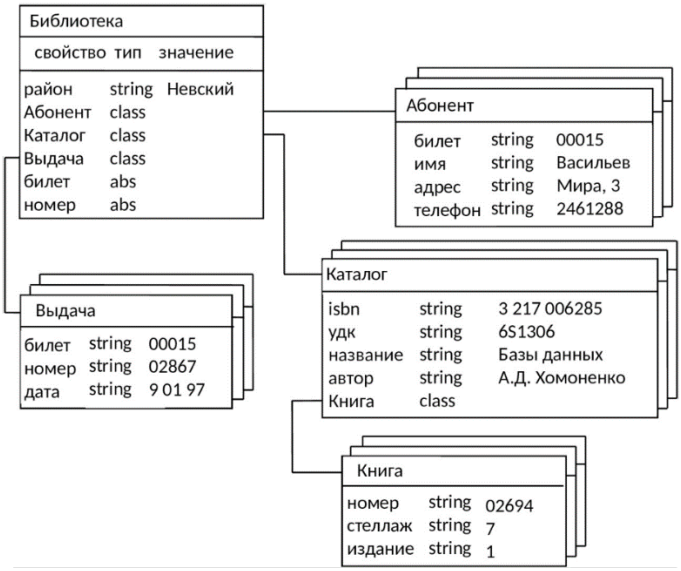

Рисунок 1.4 – Объектно-ориентированная модель данных БД

В качестве примеров, объектно-ориентированных СУБД можно привести следующие: db4o, ORION.

1.2.5 NoSQL

NoSQL (Not only SQL) - это подход к хранению и обработке данных, который позволяет хранить и обрабатывать данные, не используя традиционные реляционные базы данных. Основное отличие от реляционных баз данных в том, что NoSQL базы данных не требуют строгой схемы данных и могут обрабатывать огромные объемы данных.

В NoSQL моделях данных можно выделить несколько основных типов:

1. Графовая модель данных - данные представляются в виде графа, состоящего из вершин и ребер, что позволяет эффективно работать с связанными данными и выполнять сложные запросы.
2. Документно-ориентированная модель данных - данные хранятся в виде документов (например, JSON или BSON), что упрощает работу с иерархическими данными.
3. Ключ-значение - данные хранятся в виде пар "ключ-значение", что делает доступ к данным быстрым и эффективным.

4. Колоночная модель данных - данные хранятся в виде колонок, что позволяет эффективно работать с большими объемами данных и производить аналитические запросы.

Каждая из этих моделей данных имеет свои особенности и применяется в зависимости от конкретных потребностей проекта. Например, колоночные базы данных подходят для хранения больших объемов данных, а графовые базы данных - для анализа связей между данными.

На рисунке 1.5 представлены основные модели данных NoSQL.

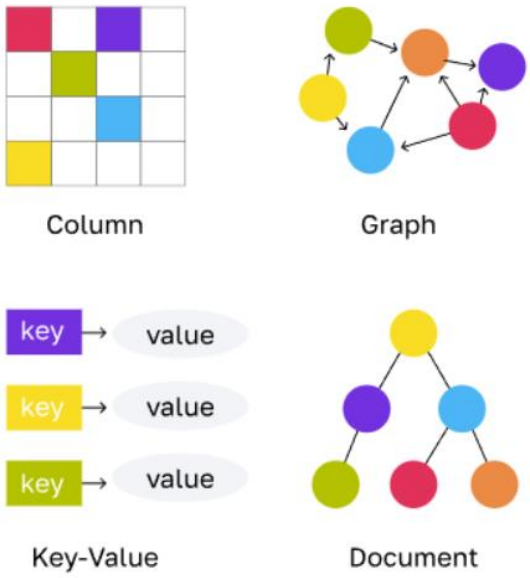

Рисунок 1.5 – Модели данных NoSQL

1.2.5.1 Графовые

Графовая модель данных – это способ организации и хранения информации, который использует графы для представления данных и их связей. В графовых моделях данные представлены в виде узлов (вершин) и связей между ними (ребер).

Графовые СУБД (системы управления базами данных) – это специализированные программные средства, предназначенные для работы с

графовыми моделями данных. Они позволяют эффективно хранить, индексировать и осуществлять запросы к данным, представленным в виде графов.

Основными составляющими графовых моделей данных являются:

1. Вершины (узлы) - представляют собой объекты или сущности, которые содержат информацию.
2. Ребра (связи) - представляют отношения между объектами или сущностями и содержат дополнительную информацию о связях.

Плюсы от использования графовых моделей данных:

1. Гибкость и удобство для моделирования сложных структур данных и их взаимосвязей.
2. Эффективность при выполнении запросов, связанных с анализом связей и путей между данными.
3. Поддержка повышенной степени декларативности для запросов к данным.

Минусы от использования графовых моделей данных:

1. Сложность при работе с большими объемами данных и сложными запросами.
2. Необходимость обеспечения высокой производительности и масштабируемости системы.

Назначение графовых моделей данных:

1. Анализ социальных сетей и связей между людьми или сущностями.
2. Маршрутизация сетей и оптимизация путей передачи данных.
3. Рекомендательные системы и анализ взаимодействия с пользователями.
4. Биоинформатика и исследование генетических и биологических данных.

5. Анализ данных и выявление зависимостей для принятия решений в различных областях деятельности.

Графическое представление графовой модели данных можно увидеть на рисунке 1.6.

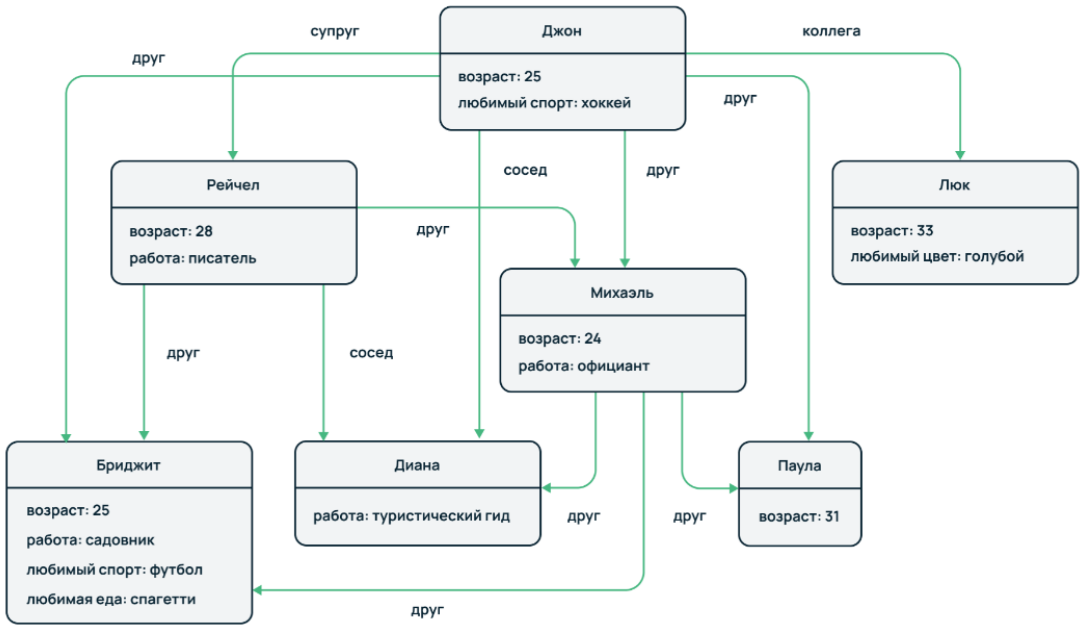

Рисунок 1.6 – Графовая модель данных[4]

В качестве примеров СУБД, поддерживающих графовую модель данных можно привести следующие: Amazon Neptune, Neo4j и InfoGrid.

1.2.5.2 Документо-ориентированные

Документно-ориентированная модель данных — это способ организации данных в базе данных, основанный на хранении документов вместо таблиц. В такой модели каждый документ представляет собой запись, содержащую структурированную информацию в формате, удобном для манипулирования и обработки.

[4] Источник картинки: https://fairysoft.ru/grafoviye-strukturiy-danniyx

Документно-ориентированные системы управления базами данных (NoSQL) отличаются от реляционных баз данных тем, что они не требуют строгой схемы данных и могут хранить различные типы данных в одном документе. Основными составляющими документа являются ключ, значение и метаданные.

Плюсы документно-ориентированной модели данных:

1. Гибкость и расширяемость: документы могут содержать различные типы данных и быть легко расширяемыми без необходимости изменения схемы базы данных.
2. Простота использования: документы легко манипулировать и обрабатывать, что делает их удобными для разработки и поддержки приложений.
3. Высокая производительность: документы хранятся в компактном виде, что обеспечивает быстрый доступ к данным.

Минусы документно-ориентированной модели данных:

1. Ограниченная поддержка запросов: из-за отсутствия языка запросов типа SQL, доступ к данным может быть ограничен.
2. Большие объемы данных: при хранении больших объемов данных может возникнуть проблема масштабируемости и производительности.
3. Сложности в поддержке транзакций: документно-ориентированные базы данных могут иметь ограниченную поддержку транзакций и консистентности данных.

Назначение документно-ориентированной модели данных заключается в создании эффективных и гибких решений для хранения и обработки данных, включая управление контентом, каталогизацию информации, хранение сенсорных данных и многое другое. Она может быть использована в различных

областях, таких как веб-разработка, аналитика данных, хранение и обработка больших объемов информации.

Пример структуры документо-ориентированной модели данных можно увидеть на рисунке 1.7.

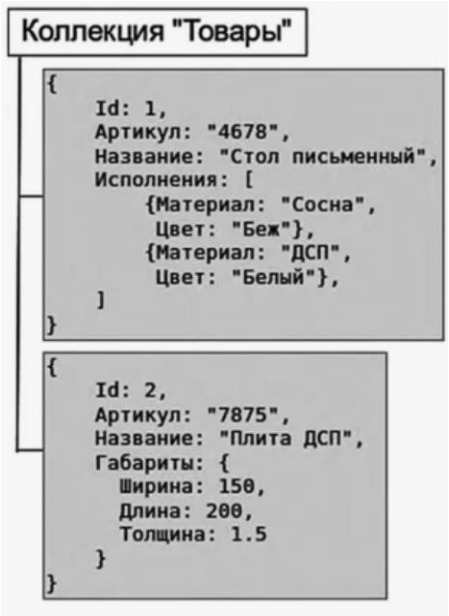

Рисунок 1.7 – Коллекция со структурой документов в документо-ориентированной модели данных БД[5]

В качестве примеров СУБД, поддерживающих документно-ориентированную модель данных можно привести следующие: Amazon DocumentDB, CouchDB, MongoDB.

1.2.5.3 Ключ-значение (Key-value)

Модель данных ключ-значение является одной из самых простых и популярных структур данных. Она представляет собой хранилище, в котором данные организованы в пары ключ-значение. Каждый элемент данных хранится с помощью уникального ключа, по которому он затем может быть извлечен.

[5] Источник картинки: https://fairysoft.ru/sostav-sistem-upravleniya-bazami-danniyx

СУБД ключ-значение (key-value store) представляет собой базу данных, в которой данные организованы и хранятся в виде пар ключ-значение. Подобные базы данных отличаются от реляционных баз данных, поскольку они не имеют строгой схемы таблиц и отношений между данными.

Составляющими модели данных ключ-значение являются:

1. Ключ: уникальный идентификатор, по которому хранится и извлекается данные.
2. Значение: данные, которые хранятся под определенным ключом.

Плюсы модели данных ключ-значение:

1. Простота и эффективность при хранении и извлечении данных.
2. Высокая скорость доступа к данным благодаря уникальному ключу.
3. Гибкость и возможность хранения различных типов данных.

Минусы модели данных ключ-значение:

1. Ограниченные возможности для сложных запросов и аналитики данных.
2. Отсутствие стандартизации при хранении структурированных данных.
3. Трудности с масштабированием и обеспечением надежности при больших объемах данных.

Назначение модели данных ключ-значение включает хранение и доступ к простым данным, кэширование данных, сессионное хранение и временное хранение данных. Кроме того, такая модель может применяться в качестве инструмента для реализации распределенных систем хранения данных.

Пример структуры модели данных ключ-значение можно увидеть на рисунке 1.8.

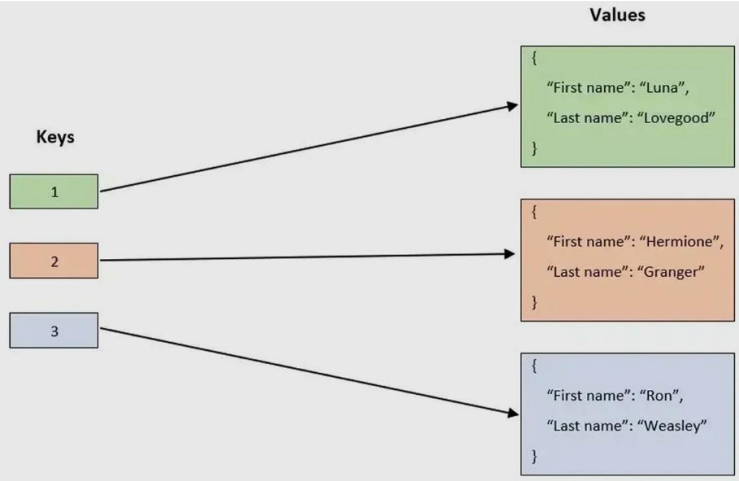

Рисунок 1.8 – Модель данных ключ-значение

В качестве примеров СУБД, поддерживающих модель данных ключ-значение можно привести следующие: Tarantool (VK), Redis (Redis Labs), Oracle NOSQL Datebase (Oracle), Amazon DynamoDB (Amazon).

1.2.5.4 Колоночная (Column-oriented)

В таких системах данные хранятся в виде матрицы, строки и столбцы которой используются как ключи.

Типичным применением этого типа СУБД является веб-индексирование и задачи, связанные с большими данными, с пониженными требованиями к согласованности. Каждая строка имеет свой набор столбцов. Данный тип систем управления базами данных обычно используется в аналитике.

Пример структуры колоночной модели данных можно увидеть на рисунке 1.9.

Рисунок 1.9 – Справа колоночная модель данных, а слева таблица, соответствующая реляционной модели данных

В качестве примеров СУБД, поддерживающих колоночную модель данных можно привести следующие: Cassandra (Apache Software Foundation), ClickHouse (Яндекс).

1.3 BASE VS ACID

Существуют следующие две модели транзакций в БД: BASE, ACID. И каждая из моделей данных БД поддерживает один из этих подходов.

Подход ACID поддерживается реляционной моделью данных.

Подход BASE поддерживают noSQL моделями данных.

1.3.1 Подход BASE

Подход BASE в моделях данных NoSQL включает:

BA – basically availability (базовая доступность) – деградация части узлов ведёт к деградации части сессий, исключая полную деградацию системы. Система отвечает на любой запрос, но в ответе могут быть неверные данные.

S – soft state (неустойчивое состояние) – уменьшение времени хранения сессий и фиксация обновлений только критичных операций.

E – eventually consistent (конечная согласованность) – изменение состояния в конечном итоге применится на все системы.

Подход BASE позволяет проектировать высокопроизводительные системы.

1.3.2 Подход ACID

Требования ACID в реляционной модели данных:

Атомарность – означает, что при фиксации выполняются все операции, составляющие транзакцию, при откате не выполняется ни одна.

Согласованность – поддержание целостности данных. Транзакция начинает работу в согласованном (целостном) состоянии и по окончании своей работы также оставляет данные согласованными.

Изоляция – означает, что на результат работы транзакции не влияют другие одновременно с ней выполняющиеся транзакции. По стандарту изоляция может иметь несколько различных уровней, в различной степени защищающих транзакции от внешних воздействий.

Долговечность – подразумевает возможность восстановить базу данных после сбоя в согласованном состоянии.

Подход ACID позволяет проектировать высоконадёжные системы.

1.3.3 BASE VS ACID

Каждый тип СУБД должен соответствовать требованиям или BASE, или ACID. Так как эти требования противоречат друг другу, то в рамках одной СУБД они сочетаться не могут.

Есть некоторые СУБД, например, MongoDB, что удовлетворяет требованиям BASE, но при определённых конфигурациях пытается частично соблюдать и требования ACID.

Если речь идёт про инфраструктуру баз данных, где нужно сочетать как надёжность, так и производительность, то могут использоваться разные типы моделей данных.

1.4 Теоремы CAP и PACELC

Теорема CAP (теорема Брюера) - это теорема в теории баз данных, которая утверждает, что в распределенной СУБД невозможно одновременно обеспечить Consistency (Согласованность), Availability (Доступность) и Partition tolerance (Толерантность к разделению) в любой сети компьютеров.

Согласованность (Consistency) означает, что все узлы в базе данных видят одинаковые данные одновременно.

Доступность (Availability) означает, что каждый запрос к базе данных должен быть успешно выполнен, даже если некоторые узлы в сети недоступны.

Толерантность к разделению (Partition tolerance) означает, что система продолжает функционировать, даже если сеть разделена на несколько частей.

Согласно теореме CAP, в распределенной системе данных можно обеспечить только два из трех свойств CAP, но не все три одновременно. Например, в сети, где происходит разделение (Partitioning), возможно соблюсти либо согласованность и доступность (CA), либо доступность и толерантность к разделению (AP), но нельзя одновременно гарантировать согласованность и толерантность к разделению (CP).

Примеры систем, которые соблюдают различные комбинации CAP, включают в себя:

1. RDBMS, такие как MySQL или PostgreSQL, обеспечивают CA, но не AP.
2. Системы NoSQL, такие как Cassandra или Couchbase, обеспечивают AP, но не CA.
3. Google Spanner обеспечивает CP, за счет репликации данных через географически разделенные регионы.

Таким образом, выбор системы баз данных должен зависеть от конкретных требований проекта к согласованности данных, доступности системы и толерантности к разделению сети.

На рисунке 1.10 приведены примеры систем, относящихся к CA, CP, PA.

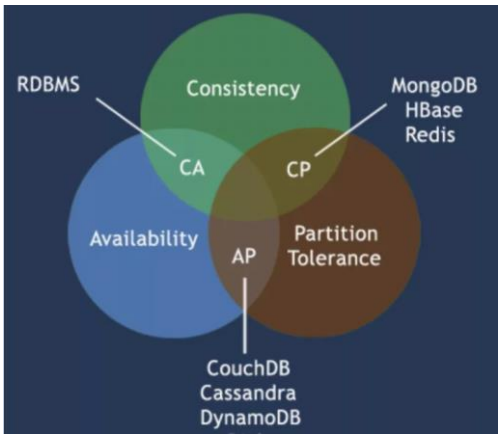

Рисунок 1.10 – Примеры систем, относящихся к CA, CP, PA[6]

Проблемы теоремы CAP:

1. Свойства далеки от реального мира.
2. В рамках разработки, выбор в основном лежит между CP и AP.
3. Множество систем — просто P.

Далее появилось расширение CAP-теоремы с добавлением понятия Latency – время, за которое клиент получит ответ, и которое регулируется каким-либо уровнем согласованности.

Учитывая данное расширение, диаграмма принимает вид, показанный на рисунке 1.11.

[6] Источник картинки: https://kuhnianasha.ru/sxema/cap

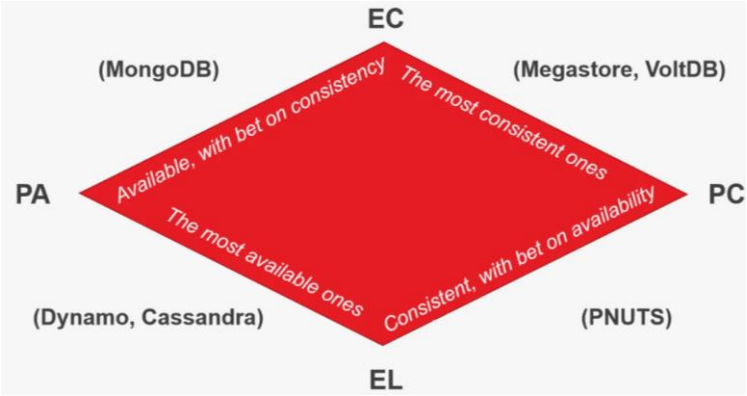

Рисунок 1.11 – Примеры PACELC

1.5 Вопросы для закрепления материала

1. В реляционных СУБД данные хранятся в:

- объектах;
- таблицах;
- узлах;
- ячейках.

2. Данные восходящего генеалогическое дерево лучше хранить в БД поддерживаемой следующей моделью данных:

- графовый;
- иерархический;
- реляционный;
- сетевой.

3. В рамках одной СУБД могут соблюдаться требования BASE и ACID?

- нет;
- да.

4. При асинхронной записи на два разных кластера, когда система устойчива к разделению она будет удовлетворять требованиям по CAP теореме:

- CA;
- CP;
- PA;
- PC.

5.Требования BASE были разработаны для реляционных СУБД?

- нет;
- да.

6. MongoDB соответствует следующей модели данных:

- реляционная;
- ключ-значение;
- документо-ориентированная;
- объектно-ориентированная.

7. PostgreSQL относится к СУБД, основанной на модели данных:

- сетевой;
- реляционный;
- табличный;
- графовый;

2 Работа с реляционной базой данных

2.1 Реляционная модель

Реляционная модель данных – это способ организации данных в базе данных, предложенный Эдгаром Коддом в 1970 году. В этой модели данные представлены в виде отношений или таблиц, где каждая строка представляет кортеж данных, а каждый столбец - атрибуты данных. Связи между отношениями устанавливаются посредством ключей.

Достоинства реляционной модели данных:

1. Простота и понятность структуры данных.
2. Легкость в поиске и обработке данных.
3. Безопасность и целостность данных благодаря использованию различных ограничений (constraints).
4. Гибкость в изменении структуры данных без необходимости изменения программного кода.

Недостатки реляционной модели данных:

1. Сложность в организации сложных и иерархических данных.
2. Некоторые операции, такие как соединения и агрегирование данных, могут быть затратными по ресурсам.
3. Ограничения на производительность при работе с большими объемами данных.
4. Не всегда удобно описывать динамические связи между данными.

Несмотря на недостатки, реляционная модель данных остается одной из самых популярных и широко используемых моделей данных в современных информационных системах благодаря своей надежности и простоте использования.

2.2 Ограничение первичных и внешних ключей

2.2.1 Первичные ключи

При создании таблицы могут быть использованы различные «ограничения» (CONSTRAINTS), которые содержат правила, указывающие, какие данные представлены в ней.

Одним из самых используемых ограничений является первичный ключ (PRIMARY KEY), который гарантирует, что каждая строка таблицы содержит уникальный идентификатор.

Правильным считается наличие первичного ключа во всех таблицах базы данных: PRIMARY KEY = UNIQUE + NOT NULL + INDEX

Первичный ключ должен удовлетворять трем условиям:

1. UNIQUE – данные в столбце/ах должны быть уникальными.

2. NOT NULL – отсутствие пустых значений.

3. INDEX – сущность, которая позволяет производить ускоренный поиск по данным.

Первичные ключи делятся на простые (состоят из одного столбца) и составные (состоят из нескольких столбцов).

Первичные ключи также делятся на натуральные (естественные) и искусственные (суррогатные).

Натуральные первичные ключи представляют собой данные, которые уже присутствуют в описываемой предметной области. Например, почтовые индексы могут быть использованы как естественные первичные ключи без дополнительной обработки. Их использование, если оно возможно, считается более правильным, чем искусственных.

В справочнике стран натуральным первичным ключом может быть ISO код стран. В справочнике граждан РФ натуральным составным первичным ключом может быть серия и номер паспорта.

Суррогатные первичные ключи, как правило, представляют собой целочисленный идентификатор. Применяется там, где нет возможности использовать натуральный первичный ключ. Позволяют решать те же практические задачи, что и естественные: улучшение производительности памяти и индексов при операциях обновления.

2.2.2 Внешние ключи

В то время как одна таблица имеет первичный ключ, другая таблица может иметь ограничение, описывающее, что её значения ссылаются на гарантированно существующие значения в первой таблице.

Это реализуется через создание в «дочерней» таблице столбца (может быть несколько столбцов), значениями которого являются значения первичного ключа (или иные уникальные значения) из «родительской» таблицы.

Вместе наборы этих столбцов составляют внешний ключ (`FOREIGN KEY`), который является механизмом базы данных, гарантирующим, что значения в «дочерних» столбцах присутствуют как первичные ключи (или иные уникальные значения) в «родительских».

Это ограничение контролирует все операции на этих таблицах:

- добавление / изменение данных в «дочерней» таблице;
- удаление / изменение данных в «родительской» таблице.

Внешний ключ проверяет, чтобы данные корректно присутствовали в обеих таблицах. Иначе операции будут отменены.

Внешние ключи могут быть составными.

Как правило «родителем» во внешнем ключе является первичный ключ, но при необходимости «родителем» может быть любой столбец, который имеет ограничение уникальности (`UNIQUE`).

2.3 Типы данных в СУБД PostgreSQL

2.3.1 Числовые типы

- integer: хранит числа от -2147483648 до +2147483647. Занимает 4 байта. Имеет псевдонимы int и int4;

- bigint: хранит числа от -9223372036854775808 до +9223372036854775807. Занимает 8 байт. Имеет псевдоним int8;

- real: хранит числа с плавающей точкой из диапазона от 1E-37 до 1E+37. Занимает 4 байта. Имеет псевдоним float4;

- double precision: хранит числа с плавающей точкой из диапазона от 1E-307 до 1E+308. Занимает 8 байт. Имеет псевдоним float8. Округление значений не соответствует правилам математики;

- numeric: хранит числа с плавающей точкой, имеет точность и масштаб, используется там, где требуется точность округления.

2.3.2 Символьные типы

- character(n): представляет строку из фиксированного количества символов. С помощью параметра задаётся количество символов в строке. Имеет псевдоним char(n);

- character varying(n): представляет строку из нефиксированного количества символов. С помощью параметра задаётся максимальное количество символов в строке. Имеет псевдоним varchar(n);

- text: представляет текст произвольной длины, устаревший тип данных.

2.3.3 Типы для работы с датами и временем

- timestamp: хранит дату и время. Занимает 8 байт. Для дат самое нижнее значение - 4713 г. до н. э., самое верхнее значение - 294276 г. н. э.;

- timestamp with time zone: то же самое, что и timestamp, только добавляет данные о часовом поясе;

- date: представляет дату от 4713 г. до н. э. до 5874897 г. н. э. Занимает 4 байта;

- time: хранит время с точностью до 1 микросекунды без указания часового пояса. Принимает значения от 00:00:00 до 24:00:00. Занимает 8 байт;

- time with time zone: хранит время с точностью до 1 микросекунды с указанием часового пояса. Принимает значения от 00:00:00+1459 до 24:00:00-1459. Занимает 12 байт;

- interval: представляет временной интервал. Занимает 16 байт.

2.3.4 Логический тип

Тип boolean может хранить одно из двух значений: true или false.

Вместо true можно указывать следующие значения:

TRUE, 't', 'true', 'y', 'yes', 'on', '1'.

Вместо false можно указывать следующие значения:

FALSE, 'f', 'false', 'n', 'no', 'off', '0'.

2.3.5 Специальные типы данных

- json: хранит данные json в текстовом виде;

- jsonb: хранит данные json в бинарном формате;

- xml: хранит данные в формате XML;

- массивы:

    - text[] – массив, элементами которого являются строки;

    - int[] – массив, элементами которого являются числа.

2.3.6 NULL

NULL – это значение, которое означает, что значение отсутствует.

Для сравнения с NULL используется конструкция:

```
SELECT *
FROM table_name
WHERE column_name IS NULL
```

Для сравнения, что значение не является NULL, используется конструкция:

```
SELECT *
FROM table_name
WHERE column_name IS NOT NULL
```

2.4 Логический порядок инструкции SELECT

При написании запроса необходимо использовать операторы в определенной логической последовательности. Последовательность операторов при написании запроса:

1. `SELECT DISTINCT` – выбор данных.
2. `FROM` – получение данных из таблицы.
3. `JOIN ON` – указание на присоединение таблицы.
4. `WHERE` – условие к данным.
5. `GROUP BY` – группировка данных.
6. `WITH CUBE или WITH ROLLUP` – сложные виды группировок.
7. `HAVING` – условие к сгруппированным данным.
8. `ORDER BY` – сортировка данных.
9. `LIMIT OFFSET` – указывается при необходимости.

Последовательность инструкций при написании отличается от последовательности при выполнении:

```
1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUP BY
6. WITH CUBE или WITH ROLLUP
7. HAVING
8. SELECT
9. DISTINCT
10. ORDER BY
11. OFFSET
12. LIMIT
```

Оператор `LIMIT` в PostgreSQL позволяет извлечь определённое количество строк.

Например, нужно извлечь четыре первых названия продуктов из таблицы `Product`:

```
SELECT *
FROM Product
ORDER BY ProductName LIMIT 4;
```

Оператор `OFFSET` позволяет указать, с какой строки надо начинать выборку.

Например, нужно извлечь три названия продукта, начиная со второй строки таблицы `Product`:

```
SELECT *
FROM Product
ORDER BY ProductName LIMIT 3 OFFSET 2;
```

2.5 Типовые операции с разными видами данных в PostgreSQL

2.5.1 Типизация

При работе с разными типами данных и совершении над этими типами операций может происходить явное и не явное преобразование типа данных.

В большинстве случаев пользователю не нужно понимать детали механизма преобразования типов. Однако неявные преобразования, выполняемые PostgreSQL, могут повлиять на результаты запроса. Когда это необходимо, эти результаты можно адаптировать с помощью явных преобразований типов.

Для определения текущего типа данных в PostgreSQL используется функция `PG_TYPEOF()`.

Запрос для выяснения типа данных у значения 100 будет записан следующим образом: `SELECT PG_TYPEOF(100)`

Данный запрос выведет результат integer.

Теперь используем одинарные кавычки, чтобы сделать из цифры 100 строку '100':

```
SELECT PG_TYPEOF('100')
```

Данный запрос вернет результат `UNKNOWN`, так как тип данных явно не задан. Тип данных зависит от того, какие действия со значением будут производиться.

Явные преобразования типов позволяют явно указывать, как должен быть преобразован конкретный тип данных. Это полезно, когда неявное преобразование приводит к неожиданным результатам.

Неявные преобразования типов выполняются автоматически PostgreSQL при необходимости. Они могут применяться, например, когда значение одного типа должно быть использовано в контексте другого типа.

2.5.2 Явное преобразование типов

Для явного преобразования одного типа данных к другому используются операторы CAST или ::

`SELECT CAST(100 AS VARCHAR(3))` – преобразует числовое значение в символьное.

`SELECT '01/01/2021'::DATE` – преобразует строку к дате.

2.5.3 Работа со строками. LIKE / ILIKE

Выражение `LIKE` возвращает `TRUE`, если строка соответствует заданному шаблону.

Выражение `NOT LIKE` возвращает `FALSE`, когда `LIKE` возвращает `TRUE` и наоборот.

Если шаблон не содержит знаков процента и подчёркиваний, тогда шаблон представляет в точности строку и `LIKE` работает как оператор сравнения.

Подчёркивание (_) в шаблоне подменяет любой символ.

Знак процента (%) подменяет любую (в том числе и пустую) последовательность символов.

`LIKE` – регистрозависимый поиск

`ILIKE` – регистронезависимый поиск.

Примеры:

`SELECT 'abc' LIKE 'abc'` – вернет true

`SELECT 'abc' LIKE 'a%'` – вернет true

`SELECT 'abc' LIKE '_b_'` – вернет true

`SELECT 'abc' LIKE 'c'` – вернет false

`SELECT 'abc' ILIKE '%c%'` – вернет true

2.5.4 Функции для работы со строками

Функция `STRPOS` возвращает положение указанной подстроки:

`SELECT STRPOS('Hello world!', 'world')` – вернет значение 7

Функция `CHARACTER_LENGTH` возвращает длину строки:

`SELECT CHARACTER_LENGTH('Hello world!')` – вернет значение12

Функция `OVERLAY` заменяет подстроку:

`SELECT OVERLAY('Hello world!' PLACING 'Max' FROM 7 FOR 5)` – вернет строку Hello Max!

Функция `SUBSTRING` извлекает подстроку:

`SELECT SUBSTRING('Hello world!' FROM 7 FOR 5)` – вернет строку world

2.5.5 Функции для работы с датами

Функция `EXTRACT` позволяет получить значение года:

`SELECT EXTRACT(YEAR FROM '28.02.2020'::DATE)` – вернет значение 2020

Функция `EXTRACT` позволяет получить номер месяца:

`SELECT EXTRACT(MONTH FROM '28.02.2020'::DATE)` – вернет значение 2 (номер месяца)

Функция `EXTRACT` позволяет получить день:

`SELECT EXTRACT(DAY FROM '28.02.2020'::DATE)` – вернет значение 28

Функция `NOW()` для получения текущей дата и времени:

`SELECT NOW()` – вернет текущие дата и время

2.5.6 Операции над типами данных

Еще больше функций можно найти пройдя по ссылкам:

- математические https://www.postgresql.org/docs/12/functions-math.html
- строковые https://www.postgresql.org/docs/12/functions-string.html
- логические https://www.postgresql.org/docs/12/functions-logical.html

2.6 Практическая часть

Лабораторная работа №1. Подключение к базе данных и создание простейших запросов.

Необходимо подключиться к уже созданной базе данных, заполнить ее данными и написать запросы для получения информации.

Для этого необходимо:

1. Скачать СУБД Postgres и pgAdmin – клиентского приложения для управления СУБД, которое дает возможность администрировать структуру БД, выполнять запросы на SQL, получать результаты запросов, импортировать и экспортировать данные и т. д.

Сделать это можно по ссылкам:

https://www.postgresql.org/download/

https://www.pgadmin.org/download/

2. Установить скачанный файл.

- принимаем лицензионное соглашение и нажимаем “Next” (рисунок 2.1);

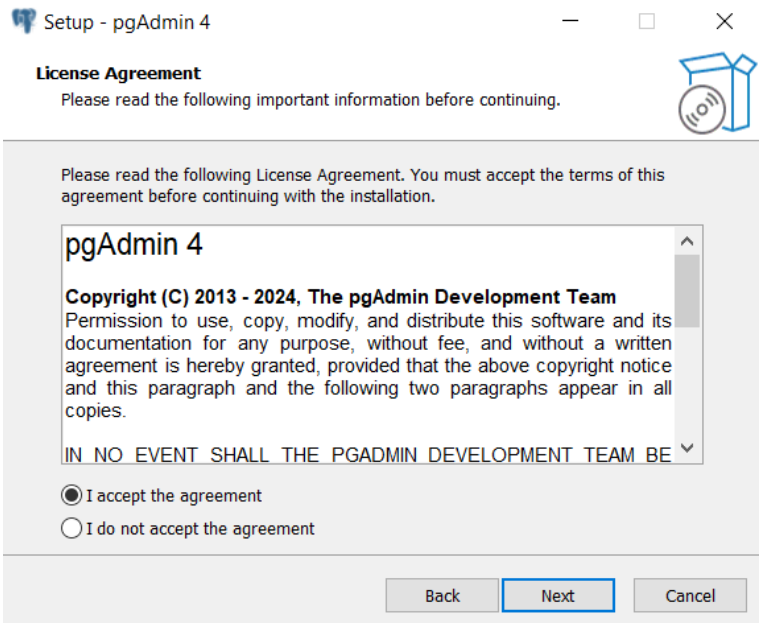

Рисунок 2.1 – Окно с лицензионным соглашением

- выбираем путь установки и нажимаем “Next” (рисунок 2.2);

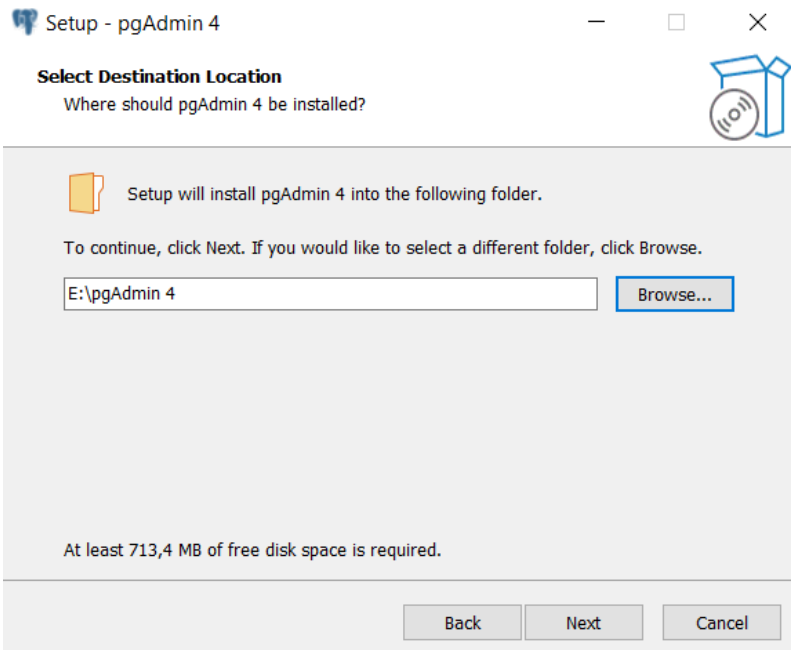

Рисунок 2.2 – Окно выбора пути установки

- проверяем параметры установки. Если все верно, то нажимаем “Install” (рисунок 2.3).

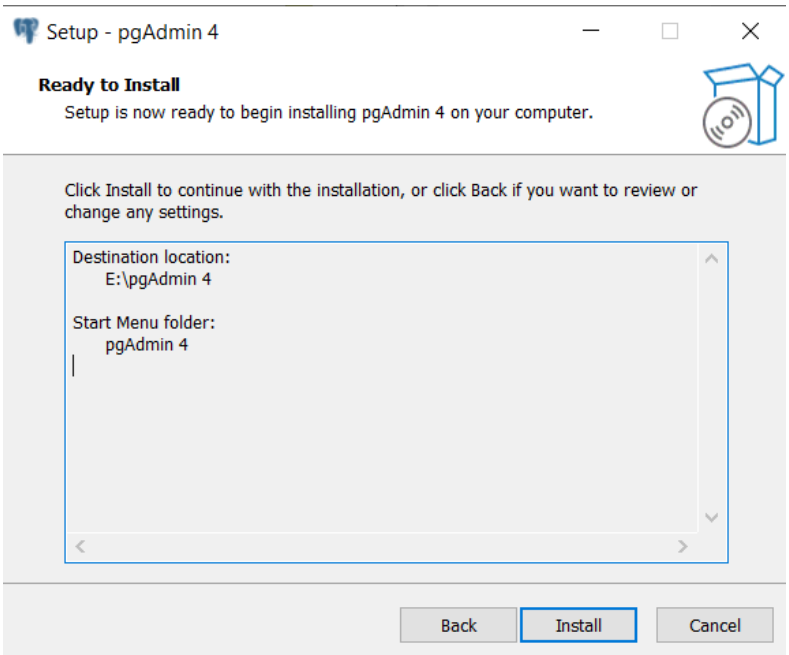

Рисунок 2.3 – Окно установки

3. Подключение к облачной базе данных

Архив для загрузки учебной БД расположен:

https://code.google.com/archive/p/northwindextended/downloads

Выберите вариант northwind.postgre.sql и загрузите на компьютер.

Этапы подключения:

3.1. Для начала необходимо добавить локальный сервер. Для этого на главном экране выбираем пункт «Добавить новый сервер» (рисунок 2.4).

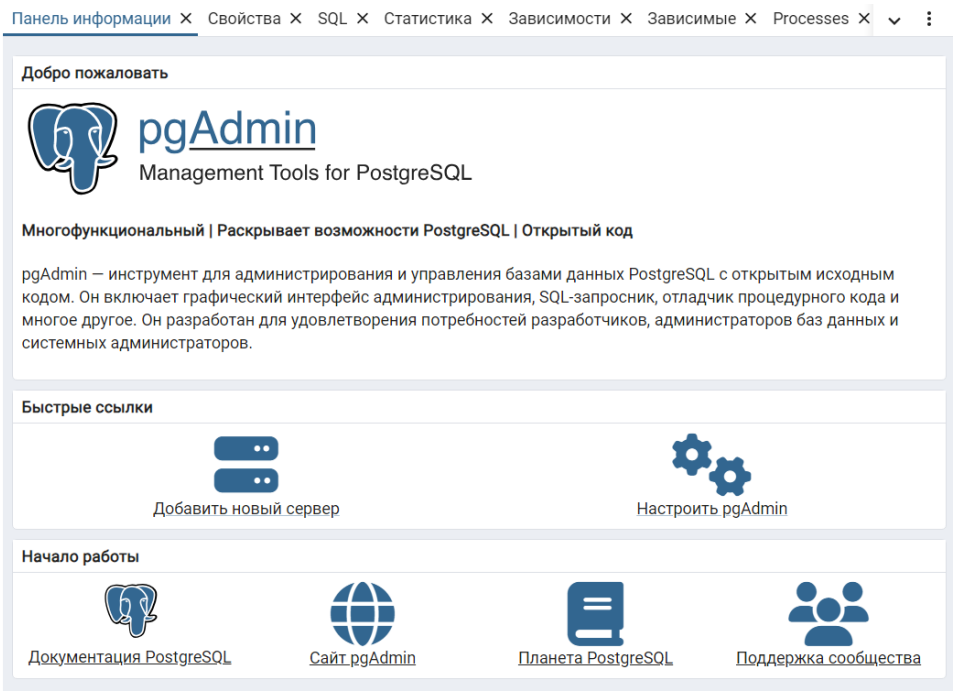

Рисунок 2.4 – Приветственное окно

После открывается окно, в котором необходимо ввести имя сервера (рисунок 2.5).

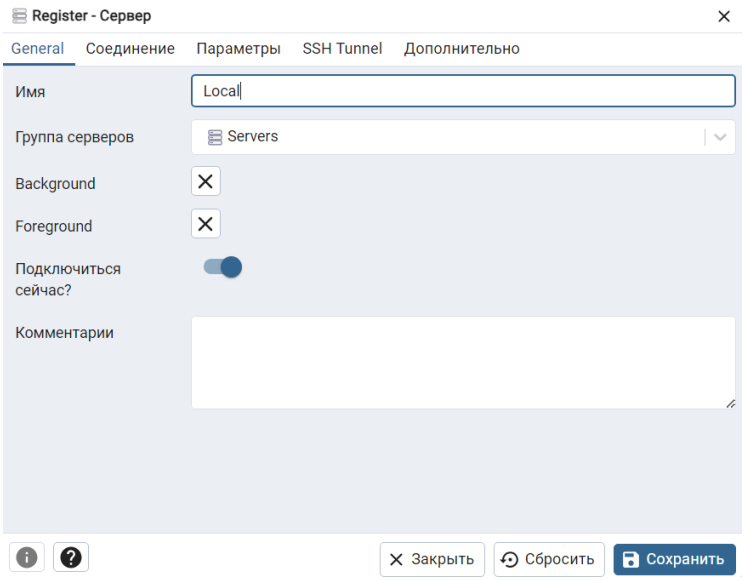

Рисунок 2.5 – Окно добавления сервера (часть 1)

Далее переходим на вкладку «Соединение» и вводим необходимые данные (рисунок 2.6).

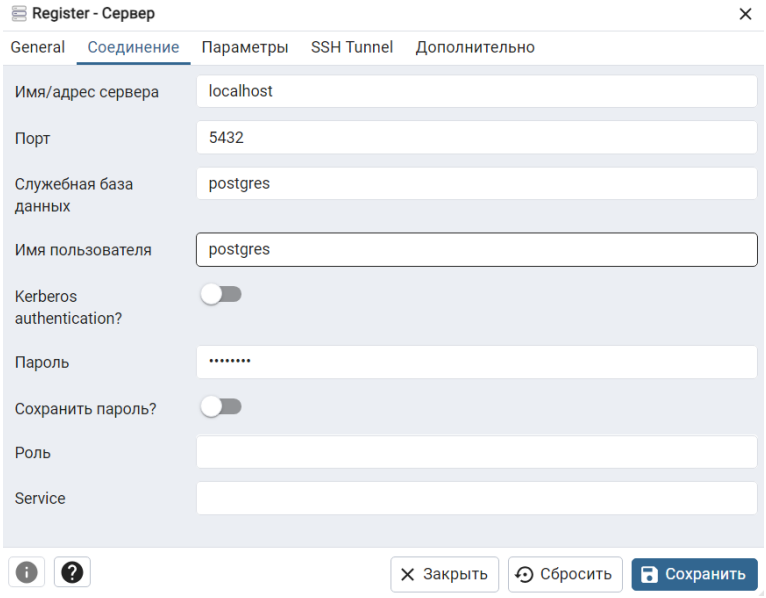

Рисунок 2.6 – Окно добавления сервера (часть 2)

3.2. Создайте базу данных с именем Northwind_фамилия (укажите свою фамилию) (рисунок 2.7).

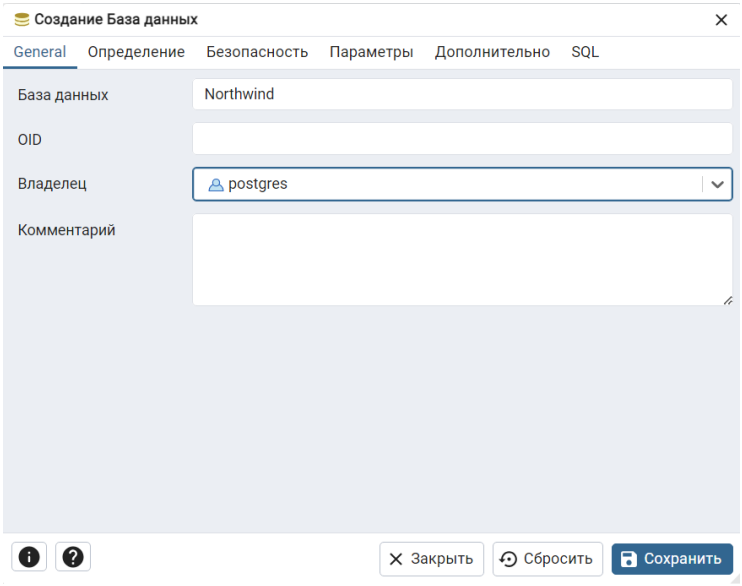

Рисунок 2.7 – Создание базы данных

3.3. Кликните по созданной базе данных и выберите в контекстном меню «Запросник» (рисунок 2.8).

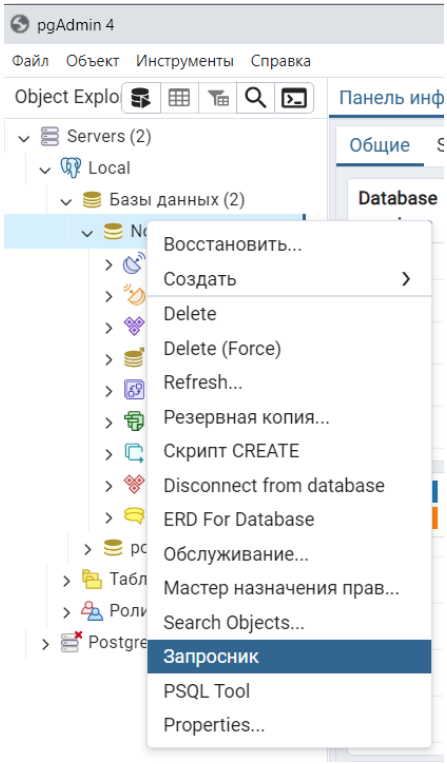

Рисунок 2.8 – Контекстное меню

3.4. Укажите путь к папке куда сохранили файл northwind.postgre.sql. Результат должен быть похож на рисунок 2.9.

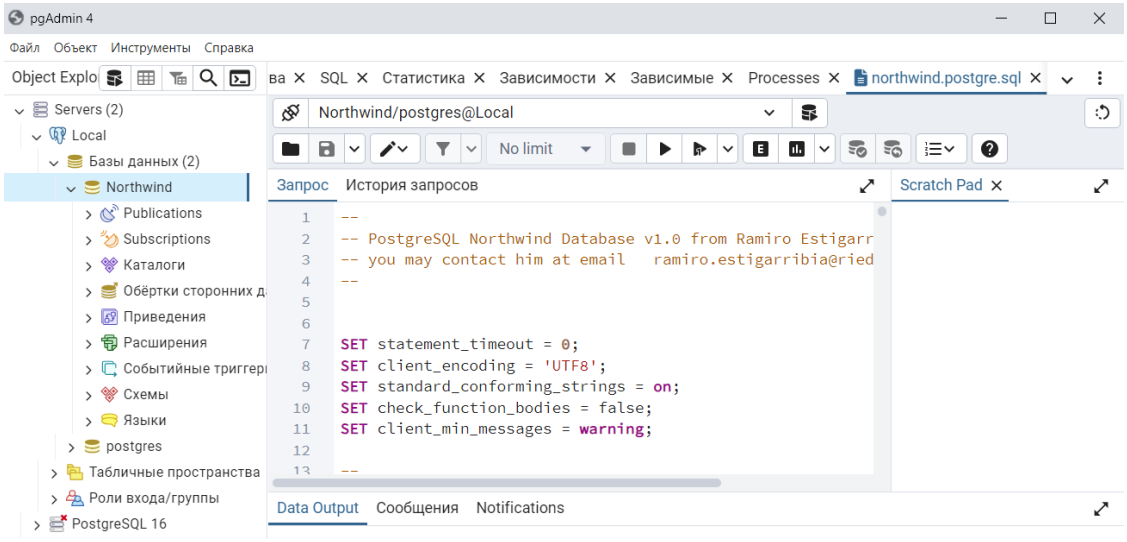

Рисунок 2.9 – Открытый файл northwind.postgre.sql

3.5. Выполните скрипт файла. Если скрипт выполнился успешно, то должно появится соответствующее сообщение (рисунок 2.10).

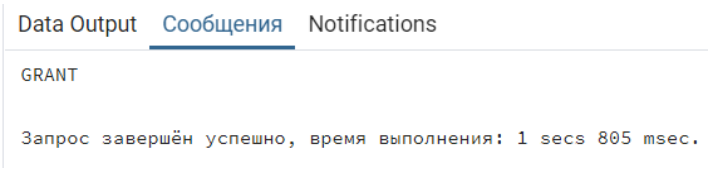

Рисунок 2.10 – Сообщение об успешном выполнение скрипта

Диаграмму схемы данных можно найти, пройдя по ссылке: https://support.microsoft.com/ru-ru/office/схема-базы-данных-northwind-cd422d47-e4e3-4819-8100-cdae6aaa0857

Описание схемы БД Northwind можно найти, пройдя по ссылке: https://docs.yugabyte.com/preview/sample-data/northwind/

4. Запросы

Необходимо написать следующие запросы:

1. Запрос, с помощью которого можно получить пользователей из таблицы `Customers`, имена которых начинаются на «B».

2. Запрос, с помощью которого можно получить товары из таблицы `Products`, название которого начинается на «M» и заканчивается на «i».

3. Запрос, который заменяет первую букву имени на «X» у пользователей из таблицы `Employees` с именами на «A».

4. Запрос, получающий год из даты 27.06.2024.

5. Запрос, получающий число из даты 27.06.2024.

6. Запрос, получающий месяц и год из даты 27.06.2024.

7. Запрос, получающий текущую дату.

2.7 Вопросы для закрепления материала

1. Верно ли утверждение, что в таблице может быть более одного ограничения первичного ключа?

- нет;

- да.

2. Ограничение внешнего ключа может быть наложено на несколько столбцов?

- нет;

- да.

3. Может ли отсутствовать значение в столбце, на который наложено ограничение первичного ключа?

- нет;

- да.

4. Можно ли создавать собственные типы данных?

- нет;

- да.

5. Какой будет результат при выполнении следующего запроса?

```
SELECT DISTINCT PG_TYPEOF(delivery_date::timestamp)
FROM delivery
```

- date;

- varchar;

- timestamp;

- integer.

6. Чтобы найти отсутствующие значения, нужно написать условие `WHERE column = null`?

- нет;

- да.

7. Какой запрос вернет результатом следующую строку 'Hello Max!'

-
```
SELECT OVERLAY('Hello world' PLACING 'Max'
FROM STRPOS('Hello world!', 'world')
FOR CHARACTER_LENGTH('world'));
```

-
```
SELECT OVERLAY ('Hello world' PLACING 'Max'
FROM STRPOS('Hello world!', 'Max')
FOR CHARACTER_LENGTH('world'));
```

-
```
SELECT OVERLAY ('Hello world!' PLACING 'Max'
FROM STRPOS('Hello world!', 'world')
FOR CHARACTER_LENGTH('world'));
```

-
```
SELECT OVERLAY ('Hello world' PLACING 'Max'
FROM 8
FOR 5).
```

8. Сколько таблиц в учебной базе данных Northwind?

- 10;

- 15;

- 20;

- 17.

3 Создание таблиц и простые запросы в SQL

3.1 DDL

Data Definition Language (DDL) – это группа операторов определения данных. С помощью этих операторов определяется структура базы данных и происходит работа с объектами базы данных.

В эту группу входят следующие операторы:

`CREATE` – используется для создания объектов базы данных;

`ALTER` – используется для изменения объектов базы данных;

`DROP` – используется для удаления объектов базы данных.

Примеры использования оператора CREATE:

Создание базы данных с именем example:

```
CREATE DATABASE example;
```

Создание таблицы customer с автоинкрементным полем первичного ключа, значение которого не подразумевает ручной ввод:

```
CREATE TABLE customer(
id INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
last_name VARCHAR(50),
birthday DATE);
```

Создание таблицы customer с автоинкрементным полем первичного ключа, значение которого позволяет использовать ручной ввод:

```
CREATE TABLE customer(
id INT PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
last_name VARCHAR(50),
birthday DATE);
```

Создание представления underage для вывода покупателей от 18-ти лет:

```
CREATE VIEW underage AS
SELECT last_name
FROM customer
WHERE DATE_PART('MONTH', AGE(NOW(), birthday)) >= 18;
```

Примеры использования оператора ALTER:

Изменение имени базы данных:

```
ALTER DATABASE old_name RENAME TO new_name;
```

Изменение имени таблицы:

```
ALTER TABLE customer RENAME TO persons;
```

Добавление столбца в таблицу:

```
ALTER TABLE persons ADD password VARCHAR(20) NOT NULL;
```

Изменение типа данных атрибута:

```
ALTER TABLE persons MODIFY last_name VARCHAR(100);
```

Примеры использования оператора DROP:

Удаление базы данных:

```
DROP DATABASE postgres;
```

Удаление таблицы:

```
DROP TABLE product;
```

Удаление представления:

```
DROP VIEW some_view;
```

3.2 DML

Data Manipulation Language (DML) – это группа операторов для манипуляции данными. С помощью этих операторов можно добавлять, изменять, удалять и получать данные из базы данных.

В эту группу входят следующие операторы:

`SELECT` – осуществляет выборку данных;

`INSERT` – добавляет новые данные в таблицу;

`UPDATE` – изменяет существующие данные;

`DELETE` – удаляет данные.

Примеры использования оператора SELECT:

Вывести все записи таблицы:

```
SELECT *
FROM persons;
```

Вывести количество записей в таблице:

```
SELECT COUNT(*)
FROM persons;
```

Вывести определённые столбцы из таблицы:

```
SELECT last_name, birthday
FROM persons;
```

Вывести данные по родившемя в январе 2000-го года:

```
SELECT last_name, birthday
FROM persons
WHERE birthday BETWEEN ‘01/01/2000’ AND ‘31/01/2000’;
```

Примеры использования оператора INSERT:

Вставить данные в таблицу (при использовании данного варианта необходимо вводить значения для всех столбцов (в том числе и идентификатора)) (возможно при использовании конструкции `GENERATED BY DEFAULT AS IDENTITY`):

```
INSERT INTO persons
VALUES (4, 'Иванов', '1985-07-12');
```

Если ограничения столбца позволяют вставлять значение `NULL`, то некоторые вставляемые значения можно заменить на значение `NULL` (в примере ниже в поле birthday устанавливается значение `NULL`):

```
INSERT INTO persons
VALUES (5, 'Петров', NULL);
```

Вставить данные в таблицу с указанием столбца или столбцов (при использовании данного варианта можно пропустить значение первичного ключа, если он задается автоинкрементом):

```
INSERT INTO persons (last_name)
VALUES ('Петров');
```

Вставить данные в таблицу из другой таблицы (добавление значения `last_name` из таблицы `customers` в таблицу `persons`):

```
INSERT INTO persons (last_name)
SELECT last_name
FROM customers;
```

Примеры использования оператора UPDATE:

Важно!

Если не указывать конкретную строку через оператор `WHERE`, то изменения произойдут во всех строках таблицы!

Изменить значение в конкретной строке:

```
UPDATE persons
SET last_name = 'Сидоров'
WHERE id = 1;
```

Изменить значения в поле `last_name` во всех строках таблицы, то есть задать всем фамилию Сидоров:

```
UPDATE persons
SET last_name = 'Сидоров';
```

Примеры использования оператора DELETE:

Важно!

Если не указывать конкретную строку через оператор `WHERE`, то изменения произойдут во всех строках таблицы!

Удалить все данные в таблице:

```
DELETE FROM persons;
```

Удалить конкретную строку или набор строк, то есть удалить строки с идентификаторами 1,3 и 7:

```
DELETE FROM persons
WHERE id IN (1, 3, 7);
```

3.3 Практическая часть

Лабораторная работа №2. DDL и DML.

Задание на лабораторную работу:

1. Необходимо, используя операторы DDL и DML, добавить ограничения внешних ключей для таблиц из учебной базы данных Northwind (для этого необходимо открыть схему БД (Схема-public-ERD For Schema), найти одноименные столбцы и связать их).

2. Создать таблицы (2-3 штуки), подходящие по контексту к данной базе данных. Добавить к созданным таблицам необходимые ограничения первичных и внешних ключей.

3. Изменить данные в таблице `customers` (одну строчку) на свои собственные (с использованием команды `UPDATE`).

4. Проверить, что будет при попытке удаления одной записи из таблицы `products` (с использованием команды `DELETE`).

5. Запрос, получающий заказы, сделанные в 1997-м году (по полю `OrderDate` в таблице `Orders`).

6. Вывести имена всех сотрудников, родившихся в январе.

3.4 Соединение данных. JOIN

Ключевое слово `JOIN` в SQL используется при построении запросов на выборку, обновление и удаление. `JOIN` позволяет соединить поля из нескольких таблиц в одну таблицу вывода. Соединение временное и целостность таблиц не нарушает.

Существует следующие типы оператора `JOIN`:

- `INNER JOIN` (слова `INNER` обычно опускается);
- `LEFT OUTER JOIN` (слова `OUTER` обычно опускается);
- `RIGHT OUTER JOIN` (слова `OUTER` обычно опускается);
- `FULL OUTER JOIN` (слова `OUTER` обычно опускается);
- `CROSS JOIN`.

Синтаксис:

```
JOIN:
<левая_таблица> JOIN <правая_таблица>
ON <условия_соединения>

LEFT JOIN:
<левая_таблица> LEFT JOIN <правая_таблица>
ON <условия_соединения>

RIGHT JOIN:
<левая_таблица> RIGHT JOIN <правая_таблица>
ON <условия_соединения>

FULL JOIN:
<левая_таблица> FULL JOIN <правая_таблица>
ON <условия_соединения>

CROSS JOIN:
<левая_таблица> CROSS JOIN <правая_таблица>
```

Рассмотрим каждый тип соединения, для этого создадим следующие таблицы:

```
CREATE TABLE table_one (
id INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
name_one VARCHAR(25) NOT NULL
);

CREATE TABLE table_two (
id INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
name_two VARCHAR(25) NOT NULL
);

INSERT INTO table_one (name_one)
VALUES ('one'), ('two'), ('three'), ('four'), ('five');
```

```
INSERT INTO table_two (name_two)
VALUES ('four'), ('five'), ('six'), ('seven'), ('eight');
```

3.4.1 INNER JOIN

INNER JOIN возвращает данные по строкам, содержащим одинаковые значения. Если смотреть на таблицы как на множества строк, то результат их выполнения можно представить на следующей диаграмме Венна (рисунок 3.1).

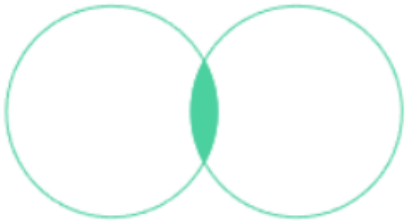

Рисунок 3.1 – INNER JOIN

Пример:

```
SELECT t1.name_one, t2.name_two
FROM table_one t1
INNER JOIN table_two t2 ON t1.name_one = t2.name_two
```

Данный запрос при исходных данных выдаст следующий результат:

```
name_one | name_two |
---------+----------+
four     |four      |
five     |five      |
```

3.4.2 LEFT JOIN

LEFT JOIN возвращает все строки из левой таблицы и совпадающие строки из правой таблицы. Если совпадений нет, то в столбцах правой таблицы в соответствующих строках, автоматически запишется значение `NULL` (рисунок 3.2).

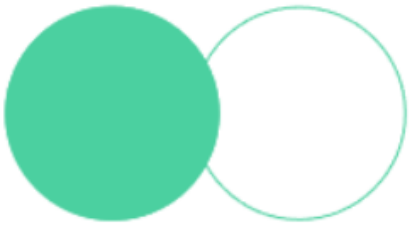

Рисунок 3.2 – LEFT JOIN

Пример:

```
SELECT t1.name_one, t2.name_two
FROM table_one t1
LEFT JOIN table_two t2 ON t1.name_one = t2.name_two
```

Данный запрос при исходных данных выдаст следующий результат:

```
name_one | name_two |
---------+----------+
one      |          |
two      |          |
three    |          |
four     |four      |
five     |five      |
```

3.4.3 RIGHT JOIN

RIGHT JOIN возвращает все строки из правой таблицы и совпадающие строки из левой таблицы. Если совпадений нет, то в столбцах левой таблицы в соответствующих строках, автоматически запишется значение `NULL` (рисунок 3.3).

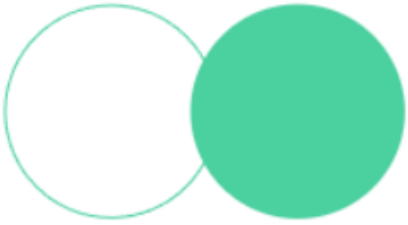

Рисунок 3.3 – RIGHT JOIN

Пример:

```
SELECT t1.name_one, t2.name_two
FROM table_one t1
RIGHT JOIN table_two t2 ON t1.name_one = t2.name_two
```

Данный запрос при исходных данных выдаст следующий результат:

```
name_one | name_two |
---------+----------+
four     |four      |
five     |five      |
         |six       |
         |seven     |
         |eight     |
```

3.4.4 FULL JOIN

FULL JOIN позволяет получить сопоставление по всем строкам в обеих таблицах. Мы получаем все данные из левой и правой таблиц, а там, где сопоставлений нет, добавляются значения `NULL` (рисунок 3.4).

Рисунок 3.4 – FULL JOIN

Пример:

```
SELECT t1.name_one, t2.name_two
FROM table_one t1
FULL JOIN table_two t2 ON t1.name_one = t2.name_two
```

Данный запрос при исходных данных выдаст следующий результат:

```
name_one | name_two |
---------+----------+
one      |          |
two      |          |
three    |          |
four     |four      |
five     |five      |
         |six       |
         |seven     |
         |eight     |
```

Чтобы получить список уникальных строк из обеих таблиц, можно также воспользоваться оператором `WHERE` (рисунок 3.5).

Рисунок 3.5 – FULL JOIN с оператором WHERE

Пример:

```
SELECT t1.name_one, t2.name_two
FROM table_one t1
FULL JOIN table_two t2 ON t1.name_one = t2.name_two
WHERE t1.name_one IS NULL OR t2.name_two IS NULL
```

Данный запрос при исходных данных выдаст следующий результат:

```
name_one | name_two |
---------+----------+
one      |          |
two      |          |
three    |          |
         |six       |
         |seven     |
         |eight     |
```

3.4.5 CROSS JOIN

CROSS JOIN – это Декартово произведение, когда каждая строка левой таблицы сопоставляется с каждой строкой правой таблицы. В результате получается таблица со всеми возможными сочетаниями строк обеих таблиц.

Пример:

```
SELECT t1.name_one, t2.name_two
FROM table_one t1
CROSS JOIN table_two t2
```

Данный запрос при исходных данных выдаст следующий результат:

```
name_one | name_two |
---------+----------+
one      |four      |
one      |five      |
one      |six       |
one      |seven     |
one      |eight     |
two      |four      |
two      |five      |
two      |six       |
two      |seven     |
two      |eight     |
three    |four      |
three    |five      |
three    |six       |
three    |seven     |
three    |eight     |
four     |four      |
```

```
four     |five      |
four     |six       |
four     |seven     |
four     |eight     |
five     |four      |
five     |five      |
five     |six       |
five     |seven     |
five     |eight     |
```

3.5 Агрегатные функции и группировка

Агрегатные функции выполняют вычисления над значениями группы строк.

Агрегатные функции:

`SUM(поле таблицы)` – возвращает общую сумму значений;

`COUNT(поле таблицы)` – возвращает количество строк, соответствующих заданному критерию;

`COUNT(*)` – возвращает общее количество строк;

`AVG(поле таблицы)` – возвращает среднее значение;

`MIN(поле таблицы)` – возвращает наименьшее значение выбранного столбца;

`MAX(поле таблицы)` – возвращает наибольшее значение выбранного столбца.

Напишем запрос для получения информации по количеству грузов, общему весу, среднему весу груза, максимальному и минимальному весу груза по каждому пользователю:

```
SELECT "CustomerID", COUNT("OrderID"), SUM("Freight"),
AVG("Freight"), MAX("Freight"), MIN("Freight")
FROM orders
GROUP BY "CustomerID";
```

GROUP BY – оператор группировки, с помощью которого можно формировать данные по группам и уже в рамках этих групп получать значения с помощью агрегатных функций.

Группировать можно как по одному столбцу, так и по нескольким. При этом важно помнить, что все значения, указанные в SELECT и неуказанные внутри агрегатных функций, должны быть указаны в операторе GROUP BY.

Дополним предыдущий запрос, чтобы получить данные по каждому пользователю, у которого вес груза более 100, а суммарный вес всех грузов более 1000:

```
    SELECT "CustomerID", COUNT("OrderID"), SUM("Freight"),
AVG("Freight"), MAX("Freight"), MIN("Freight")
    FROM orders
    WHERE "Freight" > 100
    GROUP BY "CustomerID"
    HAVING SUM("Freight") > 1000
```

Для работы с агрегатными функциями вместо WHERE необходимо использовать HAVING для задания условий.

3.6 Практическая часть

Лабораторная работа №3. Соединение данных.

Задание на лабораторную работу:

1. Требуется создать запрос, с помощью которого вывести три столбца:

- фамилия сотрудника;

- имя сотрудника;

- номер заказа, за который он отвечает, также рассмотреть вариант вывода сотрудников, у которых пока нет заказов, за которые он отвечает (новенький сотрудник).

Необходимо использовать оператор `JOIN`.

2. Требуется написать запрос для получения всех возможных комбинаций фамилий сотрудников, исключив повторения. Необходимо использовать оператор `JOIN`.

3. Создать запрос, который выводит название продукта и города, куда данный продукт доставляли. Необходимо использовать оператор `JOIN`.

Лабораторная работа №4. Запросы.

Задание на лабораторную работу:

Требуется написать запросы, чтобы получить ответ на следующие задачи:

1. Какое количество товаров отправлено в регион «RJ»?

2. В какой город отправлено больше всего грузов?

3. Сколько доставок было совершено в "Island Trading" сотрудником Laura Callahan?

3.7 Вопросы для закрепления материала

1. Напишите запрос к таблице `orders` и по результату ответьте на вопрос: «Какое количество грузов было отправлено?»

- 465;
- 946;
- 0;
- 830.

2. Напишите запрос и по результату ответьте на вопрос: «Какое количество грузов отправлено в город «Bern»?»

- 117;
- 10;
- 8;
- 16.

3. Напишите запрос и по результату ответьте на вопрос: «В какой регион отправлено больше всего товаров?»

- SP;
- RJ;
- OR;
- ID.

4 Создание отчетов и аналитика с помощью SQL

4.1 Подзапросы

Подзапрос – это SELECT, результаты которого используются в другом SELECT. Подзапросы нужны для разделения логики в основном запросе.

Подзапрос может использоваться:

- в инструкции SELECT;

- в инструкции FROM;

- в условии WHERE.

Подзапрос может быть вложен в инструкции SELECT, INSERT, UPDATE или DELETE, а также в другой подзапрос.

Подзапросы можно использовать в любой части запроса, в зависимости от этой части запроса подзапросы могут возвращать:

- отдельное значение;

- таблицу;

- одномерный массив.

Синтаксис вложенного запроса:

```
SELECT <поле или список полей>
FROM <таблица или список таблиц>
WHERE [поле]|[значение] оператор_сравнения|логический_оператор
     (SELECT <поле>
     FROM <таблица>)
```

Пример 1 (отдельное значение):

Найдём процентное отношение суммы веса грузов каждого пользователя к общему весу грузов всех пользователей:

```
SELECT "CustomerID", SUM("Freight") / (SELECT SUM("Freight")
                                       FROM orders)
FROM orders
GROUP BY "CustomerID"
```

Пример 2 (использование JOIN и подзапрос в предложении FROM):

Найдём общий вес грузов, отправленных сотрудниками, чья фамилия начинается на букву “D”:

```
SELECT e.fio, SUM("Freight")
FROM orders o JOIN (SELECT "EmployeeID",CONCAT("LastName",' ',
                                          "FirstName") AS "fio"
                    FROM employees
                    WHERE LEFT("LastName", 1) = 'D') e
     ON e."EmployeeID" = o."EmployeeID"
GROUP BY o."EmployeeID", e.fio
```

Пример 3 (использование IN и подзапрос в предложении WHERE):

Найдём общий вес грузов, отправленных сотрудниками, чья фамилия начинается на букву “D”:

```
SELECT "EmployeeID", SUM("Freight")
FROM orders
WHERE "EmployeeID" IN (
                    SELECT "EmployeeID"
                    FROM employees
                    WHERE LEFT("LastName", 1) = 'D')
GROUP BY "EmployeeID"
```

4.2 Оконные функции

По документации PostgreSQL оконная функция выполняет вычисления для набора строк, некоторым образом связанных с текущей строкой.

Можно сравнить её с агрегатной функцией, но при использовании оконной функции несколько строк не группируются в одну, а продолжают существовать отдельно.

Внутри же оконная функция, как и агрегатная, может обращаться не только к текущей строке результата запроса.

Разберем пример работы оконной функции `ROW_NUMBER`, которая нумерует строки в рамках окна. Пронумеруем грузы по каждому пользователю в порядке возрастания их веса:

```
SELECT "OrderID", "CustomerID", "Freight",
          ROW_NUMBER() OVER (PARTITION BY "CustomerID"
                                ORDER BY "Freight")
FROM orders
```

`ROW_NUMBER` – функция, которую применяется к окну.

`OVER` – описание окна.

Описание окна содержит:

`PARTITION BY` – поле или список полей, которые описывают группу строк для применения оконной функции. Работает по аналогии с `GROUP BY`, но не влияет на количество строк в запросе.

`ORDER BY` – поле, которое задаёт порядок записей внутри окна. Для полей внутри `ORDER BY` можно применять стандартные модификаторы `DESC` и `ASC`.

При работе с окнами можно использовать знакомые агрегатные функции:

- `SUM();`
- `AVG();`
- `COUNT();`
- `MIN();`

- MAX().

Также есть другие оконные функции, назначение которых можно посмотреть в документации:

- LEAD();
- LAG();
- DENSE_RANK();
- RANK();
- FIRST_VALUE() и т.д.

Полный список оконных функций, их синтаксис и назначение можно посмотреть в документации: https://www.postgresql.org/docs/current/tutorial-window.html

Так как оконные функции работают с результатом текущего запроса, то задать напрямую условие к результату оконной функции в условии WHERE текущего запроса нельзя. Для этого необходимо использовать подзапросы.

Пример:

Напишем запрос для получения каждого 10-го заказа пользователя в порядке следования идентификаторов заказов:

```
SELECT *
FROM (SELECT "OrderID", "CustomerID", "Freight",
             ROW_NUMBER() OVER (PARTITION BY "CustomerID"
                                ORDER BY "OrderID")
      FROM orders)
WHERE row_number % 10 = 0
```

4.3 Представления

VIEW – это именованные запросы, которые помогают сделать представление (именно вид) данных, лежащих в таблицах PostgreSQL.

VIEW основывается на одной или нескольких базовых таблицах. Представления удобны для часто используемых запросов.

VIEW, кроме MATERIALIZED VIEW, не хранят данные.

Создание представления:

```
CREATE VIEW view_name AS query;
```

Пример:

Получим информацию по ФИО сотрудника, общему весу грузов и количеству грузов по каждому региону и разместим этот запрос в представлении:

```
CREATE VIEW employees_sum_avg AS
     SELECT e."LastName", e."FirstName", o."ShipRegion",
                         SUM(o."Feight"), COUNT(o."OrderID")
     FROM orders o JOIN employees e
                    ON e."EmployeeID" = o."EmployeeID"
     GROUP BY e."LastName", e."FirstName", o."EmployeeID",
                                           o."ShipRegion"
     ORDER BY o."EmployeeID"
```

Теперь нам достаточно выполнить запрос:

```
SELECT *
FROM employees_sum_avg
```

VIEW от MATERIALIZED VIEW различаются тем, что при обращении к обычному представлению выполнится запрос, который лежит внутри этого представления, а при обращении к материализованному представлению будет считываться результат логики, который хранится на жёстком диске.

Пример:

Получим информацию по ФИО сотрудника, общему весу грузов и количеству грузов по каждому региону и разместим этот запрос в материализованном представлении:

```
CREATE MATERIALIZED VIEW employees_sum_avg_mat AS
    SELECT e."LastName", e."FirstName", o."ShipRegion",
                        SUM(o."Feight"), COUNT(o."OrderID")
    FROM orders o JOIN employees e
                    ON e."EmployeeID" = o."EmployeeID"
    GROUP BY e."LastName", e."FirstName", o."EmployeeID",
                                            o."ShipRegion"
    ORDER BY o."EmployeeID"
```

Если выполним запрос:

```
SELECT *
FROM employees_sum_avg_mat
```

То получим такой же результат, как и при обращении к представлению `employees_sum_avg`.

При использовании материализованных представлений результат выполнения логики будет хранится на жёстком диске.

Если нужно актуализировать и обновить данные, то нужно обновить материализованное представление:

```
REFRESH MATERIALIZED VIEW employees_sum_avg_mat
```

Денормализация – это процесс ухода от правил нормализации там, где это необходимо.

Обычные представления используются там, где нужно упростить работу с данными и использовать определённую логику многократно. Запрос, находящийся в обычном представлении, должен отрабатываться быстро.

Материализованные представления используются в денормализации и формировании тяжёлых отчётов. Важно следить за актуальностью данных, их

достоверность нужно проверять на момент составления отчёта. Настраивать планировщик задач на обновление материализованных представлений ночью, а днём работать с подготовленными данными.

4.4 Вопросы для закрепления материала

1. Для того, чтобы сгруппировать данные в рамках оконной функции применяется оператор?

- GROUP BY;
- ORDER BY;
- OVER;
- PARTITION BY.

2. Результат работы оконной функции добавляется к общему результату в виде дополнительных строк?

- нет;
- да.

3. Материализованные представления являются неотъемлемой частью нормализации базы данных?

- нет;
- да.

4. Скорость получения данных при обращении к обычному и материализованному представлениям одинаковая?

- нет;
- да.

5. Предложение partition by в оконной функции группирует данные запроса и влияет на количество строк в результате?

- нет;
- да.

5 Вопросы для закрепления материала

1. SQL это:

- строительный язык запросов;

- хранилище данных;

- структурированный язык запросов;

- модная вещь.

2. Для получения данных используется следующая команда:

- INSERT;

- UPDATE;

- DELETE;

- SELECT.

3. Для фильтрации сгруппированных данных используется оператор:

- GROUP BY;

- PARTITION BY;

- HAVING;

- ORDER BY.

4. Для работы с финансовыми значениями используется следующий тип данных:

- NUMERIC;

- VARCHAR;

- FLOAT;

- INTEGER.

5. Для обновления одного значения в кортеже используется следующая конструкция:

- UPDATE название_таблицы
  SET название_столбца = значение;
- UPDATE название_столбца
  SET значение
  WHERE название_столбца = значение;
- INSERT INTO название_таблицы (название_столбца)
  VALUES (значение);
- UPDATE название_таблицы
  SET название_столбца = значение
  WHERE название_столбца = значение.

6. Для группировки данных в рамках оконной функции используется оператор:

- GROUP BY;
- PARTITION BY;
- PARTITION;
- ORDER BY.

7. Какие типы данных поддерживаются в PostgreSQL?

- JSON;
- DATE;
- CHAR;
- DATETIME.

8. Результатом вычитания даты из даты будет:

- DATE;
- INTERVAL;
- INTEGER;
- CHAR.

9. Выберите агрегатные функции:

- SUM();
- AVG();
- COUNT();
- MINIMUM();
- MAXIMUM().

10. Результат подзапроса хранится на жестком диске и его можно использовать в разных запросах.

- да;
- нет.

11. Если на столбец наложено ограничение внешнего ключа, то это значение нельзя удалить. Верно ли утверждение?

- да;
- нет.

12. Ограничение первичного ключа, это сочетание следующих операторов:

- INDEX + NOT NULL + UNIQ;
- INDEX + SERIAL + NOT NULL;
- NOT NULL + UNIQUE + INDEX;
- NULL + INDEX + UNIQUE.

13. Вызов оконной функции происходит с помощью оператора:

- WINDOW;
- OVER;
- ОТКРОЙСЯ!;
- ROW_NUMBER().

Словарь терминов и профессиональных понятий

Агрегация

Способ преобразования набора данных в одно результирующее значение.

Атрибут

Столбец таблицы.

База данных

Совокупность данных, которые структурированы таким образом, чтобы их можно было быстро найти и обработать с помощью компьютера. Данные могут включать информацию из различных источников, таких как статьи, расчёты, нормативные акты, судебные решения и другие материалы.

База данных в PostgreSQL

Верхний уровень структуры СУБД, где хранятся структурированные данные в виде схем, таблиц, представлений, функций и иных сущностей. База данных в PostgreSQL имеет свои настройки прав доступа, автоматизации и другие свойства.

Время запроса (time)

Фактическое время выполнения узла или всего запроса, измеряется в миллисекундах.

Графический интерфейс пользователя (GUI)

Система средств для взаимодействия пользователя с электронными устройствами. Она основана на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана.

Группировка

Процесс, который используется для объединения строк, имеющих одинаковые данные, в таблице. Позволяет получать результат агрегации в рамках определённых групп данных.

Диаграмма отношений сущностей (ER-диаграмма)

Визуальное представление различных сущностей внутри системы и их взаимосвязи друг с другом.

Домен

Тип данных с дополнительными условиями, которые ограничивают допустимый набор значений. Когда нужно задать дополнительные ограничения на базовый тип данных, создаётся домен.

Индекс

Специальный объект базы данных, предназначенный в основном для ускорения доступа к данным. Данные можно получить быстрее, если в запросе указано условие по индексированному столбцу или производится сортировка по нему же.

Интегрированная среда разработки (IDE)

Комплекс программных средств, используемый для разработки программного обеспечения.

Кодировка

Система соответствия между визуальными символами и числовыми значениями. Кодировка определяет, как будут отображаться символы в приложении.

Кортеж

Строка таблицы.

Локаль (локализация)

Языковой стандарт в операционной системе, приложении или СУБД. От настройки локали зависит то, как будут отображаться символы алфавита, форматы даты и времени и другие параметры, связанные с языком и региональными настройками.

Локальная база данных

Локальный сервер, на котором установлено всё необходимое программное обеспечение, включая СУБД с данными. С ним пользователь работает без доступа к интернету.

Метаданные (pg_catalog, information_schema)

Набор таблиц и представлений, содержащих информацию об объектах, определённых в текущей базе данных. Метаданные описывают все сущности и структуру БД.

Модель данных

Совокупность структур данных и операций над ними, поддерживаемая в рамках системы управления базами данных (СУБД). Она определяет логическую структуру базы данных, а также динамическое моделирование состояний предметной области, то есть описывает, как данные могут изменяться со временем и как эти изменения могут быть отражены в базе данных.

Облачная база данных (облако)

Удалённый сервер, на котором установлено всё необходимое программное обеспечение, включая СУБД с данными. К нему пользователь имеет доступ только через интернет.

Ограничения

Дополнительные параметры, которые позволяют управлять данными в таблицах, контролируя, какие значения можно вносить, а какие нет. С помощью ограничений достигается корректность и целостность данных.

Оконная функция (окно, over, рамки окна)

Функция, которая выполняет вычисления для набора строк, некоторым образом связанных с текущей строкой. Её действие можно сравнить с вычислением, производимым агрегатной функцией.

Однако при использовании оконных функций строки не группируются в одну выходную строку, а остаются отдельными сущностями. Внутри же оконная функция, как и агрегатная, может обращаться не только к текущей строке результата запроса.

Оператор

Зарезервированное слово или символ, используемый СУБД для выполнения операций сравнения, арифметических или логических операции.

Отношение

Базовая единица СУБД, представляемая в виде классической таблицы.

План запроса (plan query, explain analyze)

Последовательность операций, согласно которой будет выполняться запрос. Выбирается планировщиком как алгоритм для обработки данных, который будет создавать меньшую нагрузку на сервер и быстрее возвращать результат.

Подзапрос (сложный запрос, вложенный запрос)

Инструкция `SELECT` в инструкции `SELECT`. То есть, когда запрос используется внутри другого запроса. Другими словами, это запрос, который используется внутри другого запроса.

Представление (view, вьюха, вьюшка)

Интерфейс, который позволяет присвоить запросу имя и в дальнейшем работать с ним, как с обычной таблицей. Разделяются на обычные, которые содержат в себе логику, и материализованные, которые хранят результат логики на жёстком диске.

Система управления базами данных (СУБД)

Комплекс программ, позволяющих создать базу данных (БД) и манипулировать этими данными: вставлять, обновлять, удалять и выбирать их. СУБД обеспечивает безопасность, надёжность хранения и целостность данных, а также предоставляет средства для администрирования БД.

Стоимость запроса (cost)

Измеряется в произвольных единицах, определяемых параметрами планировщика. Традиционно единицей стоимости считается операция чтения страницы с диска.

Схема

Пространство имён, которое содержит именованные объекты базы данных, такие как таблицы, представления, индексы, типы данных, функции и операторы. Условно говоря, она может быть представлена как «папка на диске», которая позволяет разделить данные по предметным областям.

Тип данных

Диапазон значений, которые могут храниться в столбце. Он определяет, сколько данные будут занимать места в памяти. SQL предлагает встроенные типы данных, такие как числовые, символьные, дата и время, сложные. Также SQL позволяет создавать собственные типы данных.

Транзакция

Группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще, и тогда она не произведёт никакого эффекта.

Обработка транзакций осуществляется транзакционными системами, которые создают историю выполненных транзакций.

Функциональная зависимость

Концепция в реляционной модели баз данных, которая описывает связь между атрибутами.

Функция

То же, что и оператор, только с иной синтаксической конструкцией. После названия функции используются скобки, в которые передаются аргументы функции.

Check (или `CHECK`)

Наиболее общий тип ограничений, который позволяет указать, что значение этого столбца должно удовлетворять логическому выражению.

DBeaver

Клиентское программное приложение SQL и инструмент администрирования баз данных. Для реляционных баз данных он использует интерфейс прикладного программирования Java Database Connectivity (JBDC). Это API позволяет взаимодействовать с базами данных через драйвер JDBC и обеспечивает целостность и безопасность данных.

Foreign key (внешний ключ)

Ограничение, которое указывает, что значения столбца или группы столбцов должны соответствовать значениям в некоторой строке другой таблицы. Внешний ключ формирует ссылочную целостность двух связанных таблиц.

Join

Механизм, который позволяет соединять данные из разных таблиц в одну общую промежуточную таблицу.

NoSQL

Обозначение широкого класса разнородных систем управления базами данных, появившихся в конце 2000-х – начале 2010-х годов и существенно отличающихся от традиционных реляционных СУБД с доступом к данным средствами языка SQL. NoSQL применяется к системам для решения проблемы масштабируемости и доступности за счёт полного или частичного отказа от требований атомарности и согласованности данных.

Not null (или `NOT NULL`)

Ограничение, которое указывает, что столбцу нельзя присваивать значение `NULL`.

Null (или `NULL`)

Значение, которое обозначает, что значение отсутствует.

PostgreSQL

Объектно-реляционная система управления базами данных (ОРСУБД, ORDBMS). Эта система основана на программе Postgres, Version 4.2, разработанной на факультете компьютерных наук Калифорнийского университета в Беркли. В Postgres было реализовано множество новшеств,

которые впоследствии были использованы в коммерческих системах управления базами данных (СУБД).

Primary key (первичный ключ)

Ограничение, которое означает, что образующий его столбец или группа столбцов может быть уникальным идентификатором строк в таблице. Для этого требуется, чтобы значения были одновременно уникальными и отличными от `NULL`. Первичный ключ формирует функциональную зависимость между атрибутами.

SQL

Декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных.

Unique (или `UNIQUE`)

Ограничение, которое гарантирует, что данные в определённом столбце или группе столбцов уникальны среди всех строк таблицы.