Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

А. А. Сидоров
Г. А. Волокитин

СОВРЕМЕННЫЕ СУБД

Методические указания к лабораторным работам,
и организации самостоятельной работы для студентов направления
«Программная инженерия»
(уровень бакалавриата)

Томск
2018

Сидоров, Анатолий Анатольевич

С34 Современные СУБД: методические указания к лабораторным работам и организации самостоятельной работы для студентов направления «Программная инженерия» (уровень бакалавриата) / А. А. Сидоров, Г. А. Волокитин. – Томск : Томск. гос. ун-т систем упр. и радиоэлектроники, 2018. – 34 с.

Методические указания предназначены для сопровождения образовательного процесса по дисциплине «Современные СУБД» в части подготовки обучающихся к лабораторным ра-ботам и организации их самостоятельной работы.

Для студентов высших учебных заведений, обучающихся по направлению «Программ-ная инженерия» (уровень бакалавриата), а также иным направлениям и специальностям, предусматривающим освоение цифровых компетенций в области создания программных про-дуктов на основе баз данных.

© Сидоров А. А., Волокитин Г. А., 2018
© Томск. гос. ун-т систем упр.
и радиоэлектроники, 2018

Оглавление

ВВЕДЕНИЕ 4
1 МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ 5
1.1 Лабораторная работа «Основы работы в клиент-серверной СУБД MS SQL SERVER» 5
1.2 Лабораторная работа «Основы работы в NoSQL СУБД MongoDB» 11
1.3 Лабораторная работа «Работа с многозвенной архитектурой» 16
1.4 Лабораторная работа «Разработка ORM средствами языка программирования» 18
1.5 Лабораторная работа «Разработка пользовательских процедур в PL/pgSQL» 21
1.6 Лабораторная работа «Реализация многоуровневых запросов SQL и их оптимизация» 25
1.7 Лабораторная работа «Разработка программных систем для базы данных» 29
2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ 32
2.1 Общие положения 32
2.2 Проработка лекционного материала и подготовка к лабораторным работам 32
2.3 Подготовка к промежуточной аттестации 33
СПИСОК ЛИТЕРАТУРЫ 34

ВВЕДЕНИЕ

Выполнение лабораторных работ и самостоятельная деятельность обучающихся направлены на приобретение навыков работы с современными системами управления базами данных, а также проектирования баз данных разных архитектур и расширения функционала систем управления базами данных и информационных систем, построенных на основе баз данных.

В результате выполнения лабораторных работ и осуществления самостоятельной работы обучающийся должен достичь следующих результатов:

– знать историю развития систем управления базами данных; теорию реляционных баз данных; теорию нереляционных баз данных; варианты использования процедурных расширений языка SQL; преимущества и недостатки использования систем управления базами данных для различных сфер; основы DDL, DML, DCL; виды архитектур СУБД; методы управления базами данных, манипуляции данных и осуществление доступа к данным в MongoDB; теорию распределительных систем без совместно используемых ресурсов; этапы создания и проектирования БД; теорию объектно-реляционных отображений; преимущества и недостатки ORM; основные операторы процедурных расширений; правила вложенных запросов; правила оптимизации запросов; виды систем управления базами данных по назначению;

– уметь производить настройку клиент-серверных СУБД; разрабатывать нереляционные СУБД; разрабатывать хранимые процедуры и триггеры; разрабатывать модели данных с помощью объектно-реляционного отображения; разрабатывать многоуровневые и оптимизированные запросы SQL; производить многозвенную архитектуру; производить манипуляцию данных с помощью курсоров, средствами языков программирования; разрабатывать информационные системы для работы с базами данных и их корректное тестирование;

– владеть методикой проектирования баз данных на основе нереляционной модели; программными средствами управления базами данных; навыками альтернативного использования объектно-реляционных отображений; методикой осуществления миграции данных.

В ходе освоения дисциплины у студента должны быть сформированы цифровые компетенции и профессиональное мышление.

1 МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРОВЕДЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

1.1 Лабораторная работа «Основы работы в клиент-серверной СУБД MS SQL SERVER»

Цель работы

Познакомиться с клиент-серверными СУБД, установить и настроить MS SQL SERVER, разработать структуру базы данных (БД) для выбранной предметной области, в MS SQL SERVER MANAGEMENT STUDIO содержащую не менее пяти взаимосвязанных таблиц.

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

На проверку должна быть представлена БД, содержащая не менее 5 взаимосвязанных таблиц, в каждой из которых должно быть не менее 3-х записей, а также код языка T-SQL.

Теоретические основы

SQL Server является одной из наиболее популярных систем управления базами данных (СУБД) в мире. Данная СУБД подходит для самых различных проектов: от небольших приложений до больших высоконагруженных проектов.

SQL Server Management Studio (SSMS) — это интегрированная среда для управления любой инфраструктурой SQL, от SQL Server до баз данных SQL Azure. SSMS предоставляет средства для настройки, наблюдения и администрирования экземпляров SQL Server и баз данных. С помощью SSMS можно развертывать, отслеживать и обновлять компоненты уровня данных, используемые вашими приложениями, а также создавать запросы и скрипты.

База данных в SSMS имеет следующую структуру, представленную на рисунке 1.

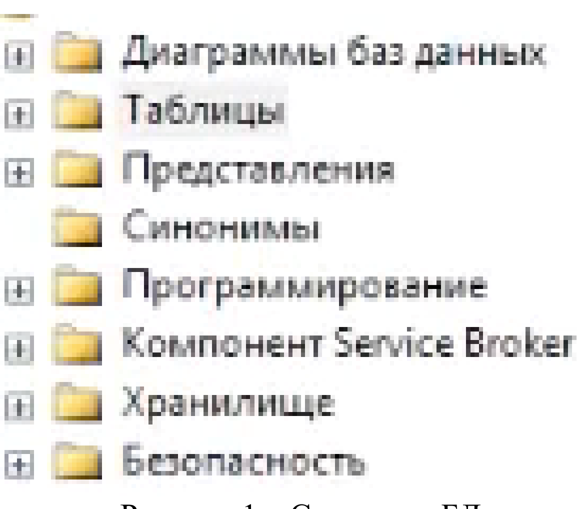

Рисунок 1 – Структура БД

Вся информация в базе данных хранится в таблицах. Таблицы состоят из записей. Запись – это строка в таблице. Вся информация обрабатывается по записям. Каждая запись состоит из полей. Поле – это столбец таблицы. Каждое поле имеет три характеристики:

- ***Имя поля*** – используется для обращения к полю;
- ***Значение поля*** – определяет информацию, хранимую в поле;
- ***Тип данных поля*** – определяет, какой вид информации можно хранить в поле.

В SQL сервер используется следующие типы данных:

Битовые типы данных, которые содержат последовательности нулей и единиц: ***Binary(n)*** и ***Varbinary(n)***, где *n* длина. Содержимое полей типа ***Binary*** всегда равно *n*, разница заполняется пробелами. ***Varbinary*** размер поля равен *n* или большему;

Целочисленные типы данных – типы данных для хранения целых чисел (в скобках указан диапазон значений типа данных): ***Tinyint*** (0- 255), ***Smallint*** (±32000), ***Int*** (±2000000000), ***Bigint*** (±263);

Типы данных для хранения дробных чисел: ***Real*** семь знаков после запятой, ***Float(m)*** может хранить числа из *m* знаков, максимальное *m*=38, ***Decimal(m n)*** дробные числа с *m* знаков до запятой и *n* после;

Специальные типы данных: ***Bit*** – логический тип данных является заменой логическому типу ***Boolean*** в Visual Basic, ***Text*** – тип для хранения больших объемов текста, одно поле может хранить до 2 Гб текста, ***Image*** – тип данных для хранения до 2Гб рисунков, *RowGUID* – уникальный идентификатор строки таблицы, ***SQL_Variant*** – аналогичен типу ***Variant*** в Visual Basic;

Типы данных даты и времени: ***Datetime*** (от 1.01.1953 до 3.12. 1999). ***SmallDatetime*** (от 1.01.19 до 6.07 2079);

Денежные типы данных для хранения финансовой информации: ***Money*** (±1015 и 4 знака после нуля), ***Smallmoney*** (± 20000,0000);

Автоматически обновляемые типы данных - аналоги счетчиков, но в данной роли они не используются: ***RowVersion*** уникальный идентификатор строки. ***TimeStamp*** – закодированное дата и время создания строки.

Для проверки подключения к MS SQL SERVER необходимо открыть приложение MS SQL Server Management Studio, нажать на кнопку Создать запрос и в открывшемся окне ввести команду:

```
SELECT @@VERSION
```

При успешном выполнении в нижней части экрана выведет сообщение, представленное на рисунке 2.

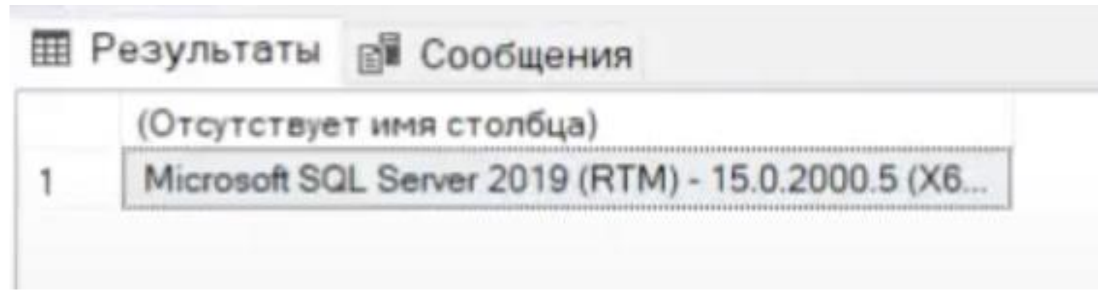

Рисунок 2 – Сообщение об успешном выполнении запроса

Новую БД можно создать, используя стандартные команды языка T-SQL. Все команды языка T-SQL набираются на вкладке нового запроса (SQLQuery). Для того чтобы создать новый запрос на панели инструментов необходимо нажать кнопку Создать запрос. Для выполнения команд языка T-SQL на панели инструментов необходимо нажать кнопку Выполнить или на вкладке нового запроса набрать команду GO. Для создания нового файла данных используется команда CREATE DATABASE, которая имеет следующий синтаксис:

```
CREATE DATABASE [Имя БД] ON PRIMARY
(
   NAME = <Логическое имя>,
    FILENAME = <Имя файла>,
    SIZE = <Нач.размер>,
    MAXSIZE = <Макс.размер>,
    FILEGROWTH = <Шаг>)
LOG ON (
   NAME = <Логическое имя>,
    FILENAME = <Имя файла>,
    SIZE = <Нач.размер>,
    MAXSIZE = <Макс.размер>,
    FILEGROWTH = <Шаг>)
```

Для создания таблиц в SQL Server в первую очередь необходимо сделать активной ту БД, в которой создается таблица. Для этого в новом запросе можно набрать команду: USE <Имя БД>, либо на панели инструментов необходимо выбрать в выпадающем списке рабочую БД. После выбора БД можно создавать таблицы.

Таблицы создаются командой:

```
CREATE TABLE <Имя таблицы>
(
    <Имя поля1> <Тип1> IDENTITY(NULL|NOTNULL),
    <Имя поля1> <Тип1> NOT NULL,
    …
);
```

Здесь:

- <Имя таблицы> – имя создаваемой таблицы;
- <Имя поля> – имена столбцов таблицы;
- <Тип>–типы полей;
- <IDENTITY NULL|NOT NULL> – поле счётчик.

Если имя поля содержит пробел, то оно заключается в квадратные скобки.

Для указания внешнего ключа таблицы нужно добавить конструкцию FOREIGN KEY в конструкцию CREATE TABLE:

```
FOREIGN KEY <имя внешнего ключа> REFERENCES <таблица первичного ключа>
    ON UPDATE <операция обновления>
    ON DELETE <операция удаления>
```

Имена столбцов внешних ключей указываются после ключевых слов FOREIGN KEY. В конструкции REFERENCES содержится имя таблицы того первичного ключа, на который производится ссылка. Если столбцы первичного ключа заданы в конструкции PRIMARY KEY своей таблицы, нет необходимости перечислять их имена. Если же имена столбцов не являются частью конструкции PRIMARY KEY, необходимо перечислить столбцы первичного ключа в конструкции REFERENCES.

В SQL Server 20XX заполнение таблиц производится при помощи следующей команды:

```
INSERT <Имя таблицы> (<Список полей>)
VALUES (<Значения полей>)
```

Здесь:

– <Имя таблицы> – таблица, куда вводим данные;

– <Список полей> – список полей, куда вводим данные, если не указываем, то подразумевается заполнение всех полей, в списке полей поля указываются через запятую;

– <Значения полей> – значение полей через запятую.

В качестве значений можно указать константу DEFAULT, то есть будет поставлено значение по умолчанию, либо можно подставить оператор SELECT.

Из таблицы можно удалить все столбцы, либо отдельные записи. Это осуществляется командой:

```
DELETE <Имя таблицы>
WHERE <Условие>
```

Если условие указано, то удаляются записи поля, которых соответствуют условию.

Для изменения данных в таблице используется следующая команда:

```
UPDATE <Имя таблицы>
SET
<Имя поля1> = <Выражение1>,
[<Имя поля2> = <Выражение2>,]
...
[WHERE <Условие>]
```

Здесь:

– <Имя поля1>, <Имя поля2> - имена изменяемых полей;

– <Выражение1>, <Выражение2> - либо конкретные значения, либо NULL, либо операторы SELECT. Здесь SELECT применяется как функция;

– <Условие> – условие, которым должны соответствовать записи, поля которых изменяем.

Порядок выполнения работы

Для примера будем использовать диаграмму базу данных успеваемости студента, приведенную на рисунке 3.

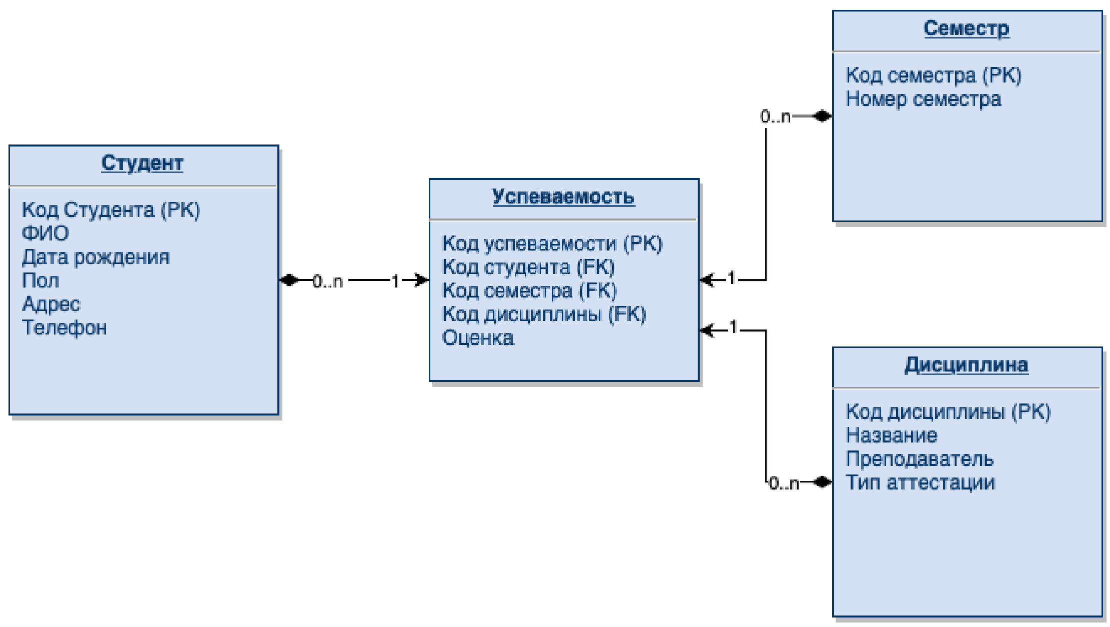

Рисунок 3 –Диаграмма базы данных успеваемости студента

Создаем БД в MS SQL SERVER MANAGEMENT STUDIO (рисунок 4)

```
SQLQuery1.sql - DE...G2FADT\prog1 (52))*
CREATE DATABASE УспеваемостьСтудента;
```

Рисунок 4 – Создание БД

Выбираем БД и создаем таблицу «Студент» (рисунок 5).

```
SQLQuery1.sql - DE...G2FADT\prog1 (52))*
USE УспеваемостьСтудента;
CREATE TABLE Студент(
    КодСтудента int NOT NULL PRIMARY KEY,
    ФИО text,
    ДатаРождения date,
    Пол varchar(1),
    Телефон text,
    Адрес text
);
```

Рисунок 5 – Создание таблицы «Студент»

Создаем таблицы «Семестр» и «Дисциплина» (рисунок 6).

```
SQLQuery1.sql - DE...G2FADT\prog1 (52))*
USE УспеваемостьСтудента
CREATE TABLE Семестр(
    КодСеместра int NOT NULL IDENTITY PRIMARY KEY,
    НомерСеместра int,
);

CREATE TABLE Дисциплина(
    КодДисциплины int NOT NULL IDENTITY PRIMARY KEY,
    Название text,
    Преподаватель text,
    ТипАттестации text
);
```

Рисунок 6 – Создание таблицы «Дисциплина»

Создаем связующую таблицу “Успеваемость” с применением Foreign Key (рисунок 7).

```
SQLQuery1.sql - DE...G2FADT\prog1 (52))*
USE УспеваемостьСтудента;
CREATE TABLE Успеваемость(
    КодУспеваемости int IDENTITY NOT NULL PRIMARY KEY,
    КодСтудента int FOREIGN KEY
        REFERENCES Студент(КодСтудента),
    КодСеместра int FOREIGN KEY
        REFERENCES Семестр(КодСеместра),
    КодДисциплины int FOREIGN KEY
        REFERENCES Дисциплина(КодДисциплины),
    Оценка text
);
```

Рисунок 7 – Создание таблицы «Успеваемость» со связующими таблицами

Варианты заданий

1. Автосалон
2. Агентство недвижимости
3. Аэропорт
4. Банк
5. Библиотека
6. Гостиница
7. Деканат
8. Документооборот предприятия
9. Магазин продовольственных товаров
10. Музей
11. Научная организация
12. Отдел кадров
13. Поликлиника

14. Развлекательный центр
15. Ресторан
16. Сервисный центр
17. Спортивный клуб
18. Супермаркет
19. Турфирма
20. Университет

Контрольные вопросы
1. Какие основные типы данных присутствуют в MS SQL SERVER?
2. Что такое T-SQL?
3. К какому виду сетевых СУБД относится MS SQL SERVER?

1.2 Лабораторная работа «Основы работы в NoSQL СУБД MongoDB»

Цель работы
Познакомиться с нереляционными СУБД, в частности MongoDB и сформировать навыки работы в них.

Форма проведения
Выполнение индивидуального задания.

Форма отчетности
На проверку должна быть представлена БД в MongoDB состоящая не менее чем из 5 таблиц и содержащая не менее 3-х записей.

Теоретические основы
MongoDB реализует новый подход к построению баз данных, где нет таблиц, схем, запросов SQL, внешних ключей и многих других вещей, которые присущи объектно-реляционным базам данных.

Если реляционные базы данных хранят строки, то MongoDB хранит ***документы***. В отличие от строк документы могут хранить сложную по структуре информацию. Документ можно представить как хранилище ключей и значений.

Ключ представляет простую метку, с которой ассоциирована определенная часть данных.

Однако при всех различиях есть одна особенность, которая сближает MongoDB и реляционные базы данных. В реляционных СУБД встречается такое понятие как ***первичный ключ***. Это понятие описывает некий столбец, который имеет уникальные значения. В MongoDB для каждого документа имеется уникальный идентификатор, который называется _id. И если явным образом не указать его значение, то MongoDB автоматически сгенерирует для него значение.

Каждому ключу сопоставляется определенное значение. Но здесь также надо учитывать одну особенность: если в реляционных базах есть четко очерченная структура, где есть поля и, если какое-то поле не имеет значение, ему (в зависимости от настроек конкретной БД) можно присвоить значение NULL. В MongoDB все иначе. Если какому-то ключу не сопоставлено значение, то этот ключ просто опускается в документе и не употребляется.

Коллекции. Если в традиционном мире SQL есть таблицы, то в мире MongoDB есть коллекции. И если в реляционных БД таблицы хранят однотипные жестко структурированные объекты, то в коллекции могут содержать самые разные объекты, имеющие различную структуру и различный набор свойств.

Репликация. Система хранения данных в MongoDB представляет набор реплик. В этом наборе есть основной узел, а также может быть набор вторичных узлов. Все вторичные узлы сохраняют целостность и автоматически обновляются вместе с обновлением главного узла. И если основной узел по каким-то причинам выходит из строя, то один из вторичных узлов становится главным.

Отсутствие жесткой схемы базы данных и, в связи с этим потребности при малейшем изменении концепции хранения данных пересоздавать эту схему значительно облегчают работу с базами данных MongoDB и дальнейшим их масштабированием. Кроме того, экономится время разработчиков. Им больше не надо думать о пересоздании базы данных и тратить время на построение сложных запросов.

Выбор существующей или создание новой базы данных:

```
use <Имя БД>
```

Узнать текущую базу данных:

```
db
```

Удалить базу данных:

```
db.dropDatabase()
```

Создать коллекцию (вообще создается автоматически при добавлении документов, но можно создать явно):

```
db.createCollection("<Имя коллекции>")
```

Удаление коллекции:

```
db.<Имя коллекции>.drop()
```

Переименовать коллекцию:

```
db.<Имя коллекции>.renameCollection("<Новое имя коллекции>")
```

Добавить один документ в коллекцию:

```
db.<Имя коллекции>.insertOne({"<Ключ1>": "<Значение1>",  "<Ключ2>": "<Значение2>"})
```

Добавить несколько документов в коллекцию:

```
db.<Имя коллекции>.insertMany({"<Ключ1>":"<Значение1>", "<Ключ2>": "<Значение2>"}, {"<Ключ1>": "<Значение1>", "<Ключ2>": "<Значение2>"})
```

Добавить любое количество документов в коллекцию:

```
db.<Имя коллекции>.insert({"<Ключ1>":"<Значение1>", "<Ключ2>": "<Значение2>"})
```

Выборка из коллекции уникальных значений поля "name":

```
db.<Имя коллекции>.distinct("<Имя ключа>")
```

Выборка по одному и нескольким условиям:

```
db.<Имя коллекции>.find({name: "<Имя ключа>"})
```

Выборка по вложенному элементу:

```
db.<Имя коллекции>.find({"<Ключ коллекции>.<Ключ вложенной коллекции>":
"<Значение вложенной коллекции>"})
```

Выборка с сортировкой по полю (по возрастанию 1, по убыванию -1):

```
db.<Имя коллекции>.find().sort({<Имя ключа>: 1})
```

Изменение данных:

```
db.<Имя коллекции>.update( { <Имя искомого ключа>: "<Значение искомого
ключа>" }, {<Ключ 1>: "<Значение1>", <Ключ 2>: "<Значение2>" }, { upsert:
true } )
```

Удаление поля, укажем оператор $unset:

```
db.<Имя коллекции>.update({<Имя искомого ключа> : "<Значение искомого
ключа>"}, {$unset: {<Имя ключа>: "<Значение ключа>"}})
```

Удаление документа:

```
db.<Имя коллекции>.deleteOne({<Имя искомого ключа> : "<Значение искомого
ключа>"})
```

Удаление множества документов:

```
db.<Имя коллекции>.remove({<Имя искомого ключа> : "<Значение искомого
ключа>"})
```

Удаление всех элементов коллекции:

```
db.<Имя коллекции>.remove({})
```

Для обеспечения связи между коллекциями необходимо во вложенной коллекции добавить «_id», а в коллекции, которой будет использоваться вставить названия поля:

```
db.<Коллекция 1>.insert({"_id" : "<Значение 1>", <Ключ 2>: <Значение2>})
db.<Коллекция 2>.insert({<Ключ 1>: "<Значение>", <Ключ 2>: "<Значение
1>"})
```

Порядок выполнения работы

Для выполнения данной лабораторной работы необходимо изменить адаптировать диаграмму под NoSQL формат. Для этого необходимо вынести «Семестр» и «Дисциплина» в отдельные коллекции. Коллекция «Студент» будет содержать в себе список успеваемости, каждая из которой будет содержать дисциплину и семестр (рисунок 8).

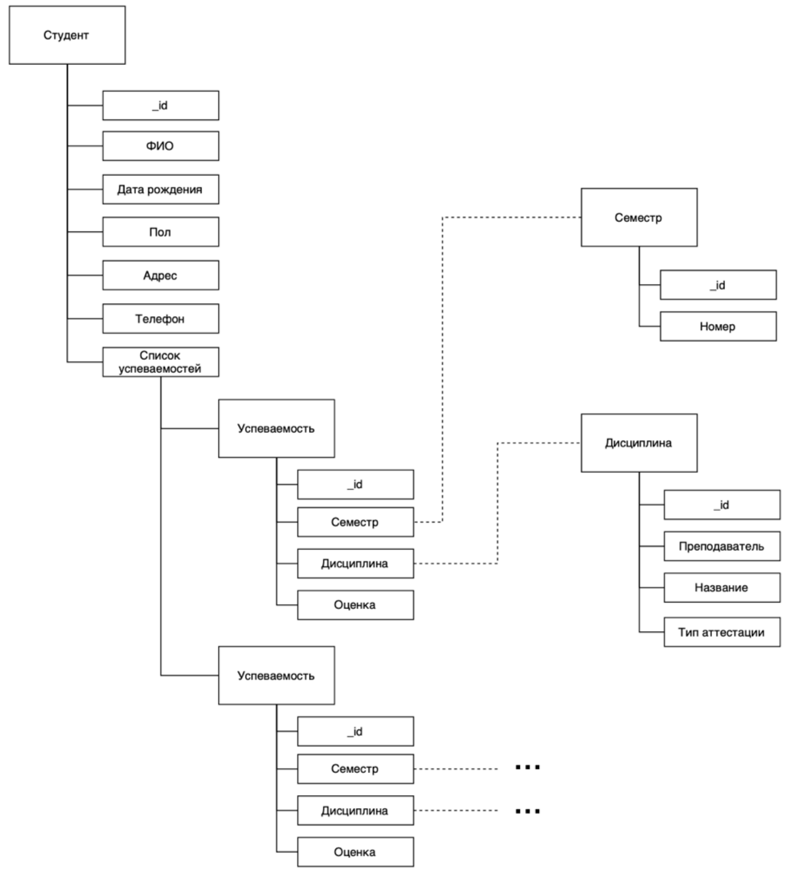

Рисунок 8 – Схема представления данных в формате NoSQL

Открываем терминал и запускаем mongoDB (рисунок 9).

```
raggab@MacBook-Pro-Gennady ~ % mongo
MongoDB shell version v5.0.2
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("f243d5ce-2df8-48a2-9957-11ba89a4f326") }
MongoDB server version: 5.0.2
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
an upcoming release.
We recommend you begin using "mongosh".
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
================
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
        https://community.mongodb.com
---
The server generated these startup warnings when booting:
        2021-08-22T18:37:18.538+07:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
        2021-08-22T18:37:18.538+07:00: Soft rlimits for open file descriptors too low
        2021-08-22T18:37:18.538+07:00:         currentValue: 256
        2021-08-22T18:37:18.538+07:00:         recommendedMinimum: 64000
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> 
```

Рисунок 9 – Запуск mongoDB

Создаем базу данных и даем ей имя student_grade (рисунок 10).

```
> use student_grade
switched to db student_grade
>
```

Рисунок 10 – Создание БД

Создаем коллекции «Семестр» и «Дисциплина» и записываем в них документы (рисунок 11).

```
> db.createCollection("semestr")
{ "ok" : 1 }
> db.semestr.insert({"number": 1})
WriteResult({ "nInserted" : 1 })
> db.createCollection("discipline")
{ "ok" : 1 }
> db.discipline.insert({"name": "Современные СУБД", "teacher": "Сенченко П.В.",})
WriteResult({ "nInserted" : 1 })
```

Рисунок 11 – Запись документов в коллекции

В коллекцию «Студент» добавляем документ с данными о студенте, в ключе «grades» добавляем список успеваемости, со связями на объекты семестра и дисциплины (рисунок 12).

```
> db.student.insert({"fio": "Иванов Иван Иванович",
... "birthday": "10.02.2000", "phone": "89999999999",
... "sex": "male", "address": "Ф Лыткина 10", "grades": [{
... "semestr": ObjectId("612240da3ac7c18b098d9434"),
... "discipline": ObjectId("612242223ac7c18b098d9437"),
... "grade": "5"},]})
WriteResult({ "nInserted" : 1 })
```

Рисунок 12 – Запись документов в коллекции со связями на объекты

Проверяем результат работы (рисунок 13).

```
> db.student.find().pretty()
{
        "_id" : ObjectId("6124d7d2093e6db7f3bc6315"),
        "fio" : "Иванов Иван Иванович",
        "birthday" : "10.02.2000",
        "phone" : "89999999999",
        "sex" : "male",
        "address" : "Ф Лыткина 10",
        "grades" : [
                {
                        "semestr" : ObjectId("612240da3ac7c18b098d9434"),
                        "discipline" : ObjectId("612242223ac7c18b098d9437"),
                        "grade" : "5"
                }
        ]
}
```

Рисунок 13 – Просмотр коллекции

Контрольные вопросы

1. Что такое документ в MongoDB?
2. В чем отличие реляционных от нереляционных БД?
3. Что такое коллекции в MongoDB?
4. Что такое репликация?
5. Что такое шардинг?

1.3 Лабораторная работа «Работа с многозвенной архитектурой»

Цель работы

Познакомиться с многозвенной архитектурой, разработать многозвенную архитектуру с применением сокетов на любом языке программирования.

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

На проверку должен быть представлен программный код, в котором реализовано общение с БД с помощью сокетов.

Теоретические основы

Многозвенная архитектура представляет собой дальнейшее совершенствование технологии «клиент – сервер». Рассмотрев архитектуру «клиент – сервер», можно сделать вывод, что она является двухзвенной: первое звено – клиентское приложение, второе звено – сервер БД + сама БД. В многозвенной архитектуре вся бизнес-логика, ранее входившая в клиентские приложения, выделяется в отдельное звено, называемое сервером приложений. При этом клиентским приложениям остается лишь пользовательский интерфейс.

Сокеты (англ. socket – разъём) – название программного интерфейса для обеспечения обмена данными между процессами. Процессы при таком обмене могут исполняться как на одной ЭВМ, так и на различных ЭВМ, связанных между собой сетью. Сокет – абстрактный объект, представляющий конечную точку соединения.

Сокет состоит из IP-адреса и порта.

IP-адрес – уникальный сетевой адрес узла в компьютерной сети, построенной по протоколу IP. В версии протокола IPv4 IP-адрес имеет длину 4 байта (например, 192.168.0.3), а в версии протокола IPv6. IP-адрес имеет длину 16 байт (например, 2001:0db8:85a3:0000:0000:8a2e:0370:7334). IP-адрес должен быть уникален.

Порт – натуральное число, записываемое в заголовках протоколов транспортного уровня (TCP, UDP и др.). Порт используется для определения процесса-получателя пакета в пределах одного хоста.

Курсор – это поименованная область памяти, содержащая результирующий набор select запроса.

Порядок выполнения работы

Для реализации работы с БД при помощи сокетов необходимо импортировать библиотеки:

```
import socket
import psycopg2
```

Задаем хост и порт для сервера, для клиента должны использоваться те же значения:

```
HOST = '127.0.0.1'
PORT = 65432
```

Подключаемся к сокету и начинаем его использовать:

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

Подключаемся к БД при помощи метода *connect()*:

```
db_connect = psycopg2.connect(
     dbname="wxnwioxo",
     user = "wxnwioxo",
     password="MkITccNXZXQMlPijxFXv_XLtiO9ovHV",
     host="hattie.db.elephantsql.com")
```

Создаем курсор для управления БД и выполняем запрос SQL:

```
cursor = db_connect.cursor()
cursor.execute("SELECT * from userdata_users")
```

Заносим данные из курсора в переменную, курсор возвращает нам список, поэтому конвертируем его в строку:

```
record = str(cursor.fetchall())
```

Закрываем курсор и подключение к БД:

```
cursor.close()
db_connect.close()
```

Устанавливаем сокету параметры подключения:

```
s.bind((HOST, PORT))
s.listen()
conn, addr = s.accept()
```

Подключаемся к сокету и начинаем слушать сообщения, при получении данных от клиента возвращаем данные, которые пришли из сокета:

```
with conn:
   while True:
       data = conn.recv(1024)
       if not data:
           break
       conn.send(str.encode(record))
```

Пишем клиент, который подключиться к серверу, отправит ему любое сообщение и вернет данные с БД:

```
import socket
HOST = '127.0.0.1'
PORT = 65432
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
   s.connect((HOST, PORT))
   s.sendall(b'run')
   data = s.recv(1024)
print('Received', data.decode())
```

Контрольные вопросы

1. Как организованна централизованная архитектура?
2. Как организованна архитектура файл-сервер?
3. Как организованна архитектура клиент-сервер?
4. В чем отличие многозвенной архитектуры от архитектуры клиент-сервер?

1.4 Лабораторная работа «Разработка ORM средствами языка программирования»

Цель работы

Познакомиться с объектно-реляционными отображениями, разработать отображения с помощью ORM средствами любого языка программирования.

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

На проверку должен быть представлен программный код, в котором содержится описание моделей БД, файлы миграции, БД с принятыми миграциями.

Теоретические основы

ORM или **Object-relational mapping** (рус. объектно-реляционное отображение) – это технология программирования, которая позволяет преобразовывать несовместимые типы моделей в ООП, в частности, между хранилищем данных и объектами программирования. ORM используется для упрощения процесса сохранения объектов в реляционную базу данных и их извлечения, при этом ORM сама заботится о преобразовании данных между двумя несовместимыми состояниями. Большинство ORM-инструментов в значительной мере полагаются на метаданные базы данных и объектов, так что объектам ничего не нужно знать о структуре базы данных, а базе данных – ничего о том, как данные организованы в приложении. ORM обеспечивает полное разделение задач в хорошо спроектированных приложениях, при котором и база данных, и приложение могут работать с данными каждый в своей исходной форме.

Использование ORM решает проблему так называемой парадигмы «несоответствия», которая гласит о том, что объектные и реляционные модели не очень хорошо работают вместе. Реляционные базы представляют данные в табличном формате, в то время как объектно-ориентированные языки представляют их как связанный граф объектов. Основные проблемы и несоответствия возникают во время сохранения этого графа объектов в реляционную базу или его загрузки:

– реляционная модель может быть намного детальнее, чем объектная, т.е. для хранения одного объекта в реляционной базе данных используется несколько таблиц;

– реляционные СУБД не имеют ничего похожего на наследование – естественную парадигму объектно-ориентированных языков программирования;

– в СУБД определен только один параметр для сравнения записей – первичный ключ. В то время как ООП предоставляет как проверку идентичности объектов (a==b), так и их равенства (a.equals(b));

– для связи объектов СУБД использует понятие внешних ключей, в объектно-ориентированных языках связь между объектами может быть только однонаправленной. Если же нужно организовать двунаправленные отношения, то придется определить две однонаправленные ассоциации. Кроме того, нет возможности определить кратность отношения, глядя на модель предметной области;

– принцип доступа к данным в ООП кардинально отличается от доступа к данным в БД. Для доступа к данным в ООП используются последовательные переходы от родительского объекта к свойствам дочерних элементов и инициализации объектов по необходимости. Такой

подход считается не эффективным способом извлечения данных из реляционных баз данных. Как правило, количество запросов к БД должно быть сведено к минимуму, необходимые сущности должны по возможности загружаться сразу с использованием JOIN-ов.

Ключевой особенностью ORM является отображение, которое используется для привязки объекта к его данным в БД. ORM как бы создает «виртуальную» схему базы данных в памяти и позволяет манипулировать данными уже на уровне объектов. Отображение показывает, как объект и его свойства связанны с одной или несколькими таблицами и их полями в базе данных. ORM использует информацию этого отображения для управления процессом преобразования данных между базой и формами объектов, а также для создания SQL-запросов для вставки, обновления и удаления данных в ответ на изменения, которые приложение вносит в эти объекты.

Порядок выполнения работы

Создание модели студента представлено на рисунке 14.

```
class StudentModel(models.Model):

    sex_choice = [
        (1,  "male"),
        (2, "female")
    ]

    fio = models.CharField(max_length=255, default="")
    birthday = models.CharField(max_length=10, default="")
    sex = models.IntegerField(choices=sex_choice, default=1)
    address = models.CharField(max_length=255, default="")
    phone = models.CharField(max_length=11, default="")

    def __str__(self):
        return self.fio
```

Рисунок 14 – Модель студента

Создание модели семестра представлено на рисунке 15.

```
class SemesterModel(models.Model):
    number = models.IntegerField(max_length=2, default=1)

    def __str__(self):
        return f'{self.number}'
```

Рисунок 15 – Модель семестра

Создание модели дисциплины представлено на рисунке 16.

```
class DisciplineModel(models.Model):

    type_choice = [
        (1, "Экзамен"),
        (2, "Зачет"),
        (3, "Диф. зачет"),
        (4, "Курсовая работа")
    ]

    name = models.CharField(max_length=255, default="")
    teacher = models.CharField(max_length=255, default="")
    type = models.IntegerField(choices=type_choice, default=1)

    def __str__(self):
        return self.name
```

Рисунок 16 – Модель дисциплины

Создание модели оценок представлено на рисунке 17.

```
class GradesModel(models.Model):
    student = models.ForeignKey(StudentModel, on_delete=models.CASCADE)
    semester = models.ForeignKey(SemesterModel, on_delete=models.CASCADE)
    discipline = models.ForeignKey(DisciplineModel, on_delete=models.CASCADE)
    grade = models.CharField(max_length=255, default="")

    def __str__(self):
        return f'{self.student.fio}'
```

Рисунок 17 – Модель оценок

Выполнение миграции представлено на рисунках 18 и 19.

```
(venv) raggab@MacBook-Pro-Gennady lab_example % python manage.py makemigrations
System check identified some issues:

WARNINGS:
api.SemesterModel.number: (fields.W122) 'max_length' is ignored when used with IntegerField.
        HINT: Remove 'max_length' from field
Migrations for 'api':
  api/migrations/0001_initial.py
    - Create model DisciplineModel
    - Create model SemesterModel
    - Create model StudentModel
    - Create model GradesModel
```

Рисунок 18 – Сохранение миграций

```
(venv) raggab@MacBook-Pro-Gennady lab_example % python manage.py migrate
System check identified some issues:

WARNINGS:
api.SemesterModel.number: (fields.W122) 'max_length' is ignored when used with IntegerField.
        HINT: Remove 'max_length' from field
Operations to perform:
  Apply all migrations: admin, api, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying api.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

Рисунок 19 – Применение миграций

После выполнения миграций можно увидеть, что модели преобразовались в таблицы (рисунок 20).

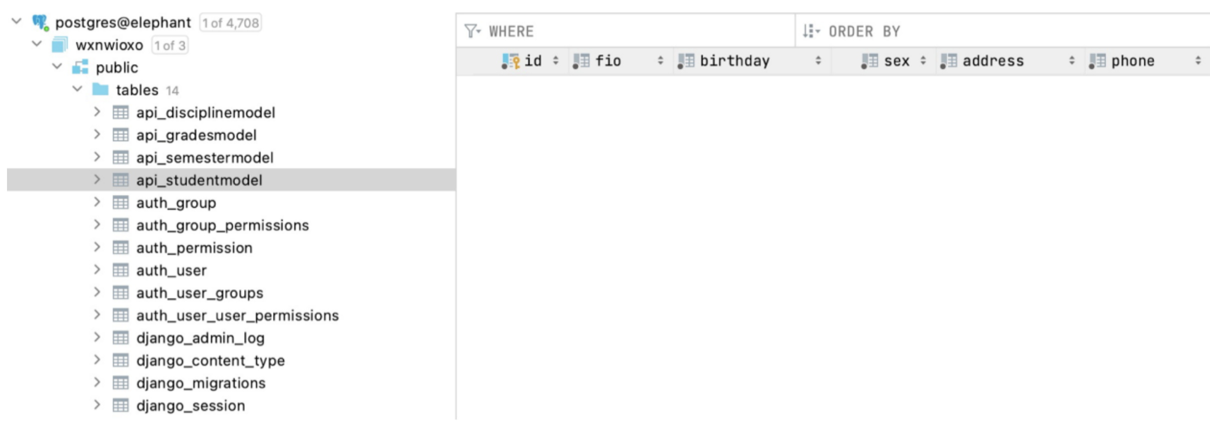

Рисунок 20 – Результат выполнения миграций

Контрольные вопросы

1. Что такое объектно-реляционные отображения?
2. Для чего нужны файлы миграции?
3. Какие преимущества и недостатки имеют объектно-реляционные отображения?

1.5 Лабораторная работа «Разработка пользовательских процедур в PL/pgSQL»

Цель работы

Познакомиться с процедурными расширениями языка SQL, разработать процедуры и триггеры для PostgreSQL с применением процедурного языка PL/pgSQL.

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

На проверку должен быть представлен триггер, реализованный с помощью процедурного расширения языка SQL.

Теоретические основы

PL/pgSQL – это процедурный язык для СУБД PostgreSQL. Целью проектирования PL/pgSQL было создание загружаемого процедурного языка, который позволяет реализовывать следующий функционал и обладает необходимыми свойствами:

– используется для создания функций и триггеров;
– добавляет управляющие структуры к языку SQL;
– может выполнять сложные вычисления;
– наследует все пользовательские типы, функции и операторы;
– может быть определён как доверенный язык;
– прост в использовании.

Функции PL/pgSQL могут использоваться везде, где допустимы встроенные функции. Например, можно создать функции со сложными вычислениями и условной логикой, а затем использовать их при определении операторов или в индексных выражениях.

В версии PostgreSQL 9.0 и выше, PL/pgSQL устанавливается по умолчанию. Тем не менее, это, по-прежнему, загружаемый модуль и администраторы, особо заботящиеся о безопасности, могут удалить его при необходимости.

PostgreSQL и большинство других СУБД используют SQL в качестве языка запросов. SQL хорошо переносим и прост в изучении. Однако каждый оператор SQL выполняется индивидуально на сервере базы данных.

Каждое объявление и каждый оператор в блоке должны завершаться символом «;» (точка с запятой). Блок, вложенный в другой блок, должен иметь точку с запятой после END, как показано выше. Однако финальный END, завершающий тело функции, не требует точки с запятой.

Метка требуется только тогда, когда нужно идентифицировать блок в операторе EXIT, или дополнить имена переменных, объявленных в этом блоке. Если метка указана после END, то она должна совпадать с меткой в начале блока.

Ключевые слова не чувствительны к регистру символов. Как и в обычных SQL-командах, идентификаторы неявно преобразуются к нижнему регистру, если они не взяты в двойные кавычки.

Это значит, что ваше клиентское приложение должно каждый запрос отправлять на сервер, ждать пока он будет обработан, получать результат, делать некоторые вычисления, затем отправлять последующие запросы на сервер. Всё это требует межпроцессного взаимодействия, а также несёт нагрузку на сеть, если клиент и сервер базы данных расположены на разных компьютерах.

В PL/pgSQL можно создавать триггерные процедуры, которые будут вызываться при изменениях данных или событиях в базе данных. Триггерная процедура создаётся командой CREATE FUNCTION, при этом у функции не должно быть аргументов, а типом возвращаемого значения должен быть trigger (для триггеров, срабатывающих при изменениях данных) или event_trigger (для триггеров, срабатывающих при событиях в базе). Для триггеров автоматически определяются специальные локальные переменные с именами вида TG_имя, описывающие условие, повлёкшее вызов триггера.

Триггер при изменении данных объявляется как функция без аргументов и с типом результата trigger. Заметьте, что эта функция должна объявляться без аргументов, даже если

ожидается, что она будет получать аргументы, заданные в команде CREATE TRIGGER – такие аргументы передаются через TG_ARGV, как описано ниже.

Когда функция на PL/pgSQL срабатывает как триггер, в блоке верхнего уровня автоматически создаются несколько специальных переменных:

NEW – тип данных RECORD. Переменная содержит новую строку базы данных для команд INSERT/UPDATE в триггерах уровня строки. В триггерах уровня оператора и для команды DELETE этой переменной значение не присваивается.

OLD – тип данных RECORD. Переменная содержит старую строку базы данных для команд UPDATE/DELETE в триггерах уровня строки. В триггерах уровня оператора и для команды INSERT этой переменной значение не присваивается.

TG_NAME – тип данных name. Переменная содержит имя сработавшего триггера.

TG_WHEN – тип данных text. Строка, содержащая BEFORE, AFTER или INSTEAD OF, в зависимости от определения триггера.

TG_LEVEL – тип данных text. Строка, содержащая ROW или STATEMENT, в зависимости от определения триггера.

TG_OP – тип данных text. Строка, содержащая INSERT, UPDATE, DELETE или TRUNCATE, в зависимости от того, для какой операции сработал триггер.

TG_RELID – тип данных oid. OID таблицы, для которой сработал триггер.

TG_RELNAME – тип данных name. Имя таблицы, для которой сработал триггер. Эта переменная устарела и может стать недоступной в будущих релизах. Вместо неё нужно использовать TG_TABLE_NAME.

TG_TABLE_NAME – тип данных name. Имя таблицы, для которой сработал триггер.

TG_TABLE_SCHEMA – тип данных name. Имя схемы, содержащей таблицу, для которой сработал триггер.

TG_NARGS – тип данных integer. Число аргументов в команде CREATE TRIGGER, которые передаются в триггерную процедуру.

TG_ARGV[] – тип данных массив text. Аргументы от оператора CREATE TRIGGER. Индекс массива начинается с 0. Для недопустимых значений индекса (< 0 или >= tg_nargs) возвращается NULL.

Триггерная функция должна вернуть либо NULL, либо запись/строку, соответствующую структуре таблице, для которой сработал триггер.

Порядок выполнения работы

Для примера напишем триггерную процедуру, которая будет записывать изменения в таблице «Студенты» в новую таблицу.

Для начала необходимо создать таблицу, в которую будут записываться данные логгирования об изменениях. В поле «text» будем записывать произведенную операцию и ФИО студента, а в поле «added» время операции (рисунок 21).

```
create table logs
(
    id bigserial not null
        constraint logs_pk
            primary key,
    text text,
    added timestamp without time zone
);
```

Рисунок 21 – Создание таблицы для логгирования

Затем необходимо написать функцию логгирования. При помощи переменной TG_OP, можно узнать какой запрос был выполнен, при помощи переменных NEW вновь созданную либо отредактированную запись, OLD удаленную либо запись до редактирования (рисунок 22).

```
CREATE OR REPLACE FUNCTION add_to_log() RETURNS TRIGGER AS $$
DECLARE
    mstr varchar(30);
    astr varchar(100);
    retstr varchar(254);
BEGIN
    IF    TG_OP = 'INSERT' THEN
        astr = NEW.fio;
        mstr := 'Add new user ';
        retstr := mstr || astr;
        INSERT INTO logs(text,added) values (retstr,NOW());
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        astr = NEW.fio;
        mstr := 'Update user ';
        retstr := mstr || astr;
        INSERT INTO logs(text,added) values (retstr,NOW());
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        astr = OLD.fio;
        mstr := 'Remove user ';
        retstr := mstr || astr;
        INSERT INTO logs(text,added) values (retstr,NOW());
        RETURN OLD;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Рисунок 22 – Процедура логгирования

Далее необходимо написать сам триггер, который будет работать на сервере базы данных (рисунок 23).

```
CREATE TRIGGER u_logger
    AFTER INSERT OR UPDATE OR DELETE ON api_studentmodel FOR EACH ROW
    EXECUTE PROCEDURE add_to_log()
```

Рисунок 23 – Триггер

Далее необходимо выполнить триггер, после чего он будет записывать изменения в новую таблицу. Получим мы примерно следующее (рисунок 24).

	id	text	added
1	1	Remove user Петров Петр Петрович	2021-11-01 15:57:38.530227
2	2	Add new user Петров Петр Петрович	2021-11-01 15:59:19.414506

Рисунок 24 – Результат работы

Контрольные вопросы

1. Какие виды процедурных расширений бывают?
2. Основные операторы процедурных расширений?

3. Что такое хранимые процедуры?
4. Что такое триггер?
5. Какие переменные хранит триггер?

1.6 Лабораторная работа «Реализация многоуровневых запросов SQL и их оптимизация»

Цель работы

Познакомиться с многоуровневыми SQL-запросами и правилами оптимизации SQL-запросов, применить на практике полученные знания.

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

На проверку должны быть представлены два многоуровневых запросов на выборку и добавление данных, которые должны соответствовать правилам оптимизации.

Теоретические основы

Вложенный запрос – это запрос, который находится внутри другого SQL запроса и встроен внутри условного оператора WHERE.

Данный вид запросов используется для возвращения данных, которые будут использоваться в основном запросе, как условие для ограничения получаемых данных.

Вложенные запросы должны следовать следующим правилам:

– вложенный запрос должен быть заключён в родительский запрос;
– вложенный запрос может содержать только одну колонку в операторе SELECT;
– оператор ORDER BY не может быть использован во вложенном запросе. Для обеспечения функционала ORDER BY, во вложенном запросе может быть использован GROUP BY;
– вложенные запросы, возвращающие более одной записи могут использоваться с операторами нескольких значений, как оператор IN;
– вложенный запрос не может заканчиваться в функции;
– SELECT не может включать никаких ссылок на значения BLOB, ARRAY, CLOB и NCLOB;
– оператор BETWEEN не может быть использован вместе с вложенным запросом.

Примеры:

Вложенный запрос имеет следующий вид:

```
SELECT имя_колонки [, имя_колонки2 ]
FROM   таблица1 [, таблица2 ]
WHERE  имя_колонки ОПЕРАТОР
       (SELECT имя_колонки [, имя_колонки2 ]
       FROM таблица1 [, таблица2 ]
       [WHERE])
```

Предположим, что имеется таблица ***Программисты***, которая содержит следующие записи:

ИД	Имя	Специальность	Опыт	Стоимость часа
1	Программист 1	Java	2	2500
2	Программист 2	Java	3	3500
3	Программист 3	C++	3	2500
4	Программист 4	C#	2	2000
5	Программист 5	Python	2	1800
6	Программист 6	Python	2	1800
7	Программист 7	C#	1	900

Предположим, что имеется есть клон таблицы ***Программисты***, который имеет имя ***Программисты_Копия*** и имеет структуру как у таблицы ***Программисты*** и не содержит данные.

Теперь попробуем выполнить для этой же таблицы следующий запрос:

```
INSERT INTO Программисты_Копия
               SELECT * FROM developers
               WHERE ИД IN (SELECT ИД
                            FROM Программисты);
```

В результате выполнения данного запроса таблица ***Программисты_Копия*** будет содержать следующие данные:

ИД	Имя	Специальность	Опыт	Оклад
1	Программист 1	Java	2	2500
2	Программист 2	Java	3	3500
3	Программист 3	C++	3	2500
4	Программист 4	C#	2	2000
5	Программист 5	Python	2	1800
6	Программист 6	Python	2	1800
7	Программист 7	C#	1	900

Другими словами, мы скопировали все данные из таблицы ***Программисты*** в таблицу ***Программисты_Копия***.

Теперь изменим данные в таблице ***Программисты***, воспользовавшись данными из таблицы ***Программисты_Копия*** с помощью следующего запроса:

```
UPDATE Программисты
       SET Оклад = Оклад * 1.25
       WHERE Опыт IN (SELECT Опыт
                            FROM Программисты_Копия
                            WHERE Опыт >= 2);
```

В результате этого наша таблица, содержащая изначальные данные, будет хранить следующие данные:

ИД	Имя	Специальность	Опыт	Стоимость часа
1	Программист 1	Java	2	3125
2	Программист 2	Java	3	4375
3	Программист 3	C++	3	3125
4	Программист 4	C#	2	2500
5	Программист 5	Python	2	2250
6	Программист 6	Python	2	2250
7	Программист 7	C#	1	900

И наконец, попробуем выполнить удаление данных из таблицы с помощью вложенного запроса:

```
 DELETE FROM Программисты
 WHERE  Опыт ERIENCE IN
     (SELECT Опыт FROM Программисты_Копия
                             WHERE Опыт >= 2);
```

В результате таблица ***Программисты*** содержит следующие записи:

ИД	Имя	Специальность	Опыт	Стоимость часа
7	Программист 7	C#	1	900

Оптимизация SQL-запросов

1. Используйте конкретные имена столбцов после оператора select, вместо «*» – это позволит увеличить быстроту отработки запроса и уменьшению сетевого трафика.

Сведите к минимуму использование подзапросов.

2. Например, запрос:

```
Select Column_A
From Table_1
Where Column_B = (Select max (Column_B From Table_2)
And Column_C = (Select max (Column_C From Table_2)
And Column_D = ‘position_2’
```

выглядит значительно хуже на фоне аналогичного запроса:

```
Select Column_A
From Table_1
Where (Column_B, Column_C) = (Select max (Column_B), max (Column_C)
From Table_2)
```

3. Используйте оператор IN аккуратно, поскольку на практике он имеет низкую производительность и может быть эффективен только при использовании критериев фильтрации в подзапросе.

4. Соединение таблиц в запросе также является критичным: в случае, когда соединение таблиц происходит в правильном порядке, то общее число строк, необходимых к обработке, значительно сократится.

При соединении основной и уточняющей таблиц убедитесь, что первой будет основная таблица, в противном случае вы рискуете получить обработку гораздо большего числа строк, чем необходимо.

5. При соединении таблиц EXIST предпочтительнее distinct (таблицы отношения «один-ко-многим»).

6. Избыточность при работе с SQL – это критичная необходимость, используйте в разделе WHERE как можно больше ограничивающих условий.

Например, если указан:

```
WHERE Column_A=Column_B and Column_A=425
```

Вы сможете вывести результат, где Column_B=425, однако при задании условий:

```
WHERE Column_A=Column_B and Column_B=Column_C
```

оператор не сможет определить, что Column_A=Column_C.

7. Пишите простые запросы. Больше упрощайте. Оптимизатор может не справиться со слишком сложными операторами. Кроме того, иногда выполнение нескольких простых до невозможности операторов дает лучший результат по сравнению со сложными и позволяет добиться лучшей эффективности.

8. Помните, что одного и того же результата можно добиться разными способами. Например, оператор MINUS выполняется гораздо быстрее, чем запросы с оператором WHERE NOT EXIST. Запрос с данным оператором в самом общем виде выглядит следующим образом:

```
Select worker_id
From workers
MINUS
Select worker_id
From orders
```

Этот пример показывает все значения worker_id, которые содержаться в таблице workers, не в таблице orders. Другими словами, если бы значение worker_id одновременно присутствовало в таблицах workers и orders, то значение worker_id не вывелось в результат, поскольку нет конкретики, содержание какой именно таблицы вывести как результат отработки запроса.

9. Оформляйте повторяющиеся коды в пользовательскую процедуру. Это может значительно ускорить работу, уменьшить сетевой трафик.

Таким образом, рассмотренные нами моменты работы с SQL операторами и запросами значительно ускоряют работу с СУБД.

В заключение хочется отметить, что очень важно при работе с SQL – мыслить шире, чем границы поставленной перед вами задачи

Всегда старайтесь оптимизировать запрос, какой бы не была мощной инфраструктура, даже ее производительность может снизиться при выполнении неоптимизированных запросов. Учитывайте все необходимые условия при работе с операторами таким образом, чтобы нагрузка на базу была минимальной.

Контрольные вопросы

1. Какие правила вложенных запросов используют?
2. Какие правила оптимизации запросов используют?

1.7 Лабораторная работа «Разработка программных систем для базы данных»

Цель работы

Разработать многозвенную архитектуру, в которой клиентская часть имеет графический интерфейс и была реализована реляционная база данных с применением процедурных расширений.

Форма проведения

Выполнение индивидуального задания.

Форма отчетности

На проверку должна быть представлена программная система, использующая БД и имеющая графический интерфейс.

Теоретические основы

СУБД по типу назначения делятся на 3 вида.

Промышленные универсального назначения. Универсальные рассчитаны «на все случаи жизни» и, как следствие, либо очень сложны в использовании и требуют от пользователя специальных знаний, либо просты, но ограничены в возможностях.

Примеры:

– Access;
– FoxPro;
– Oracle;
– DB2.

Промышленные специализированные. Специализированные направлены на выполнения узких задач и поэтому создаются так, чтобы они были просты в использовании для профессионалов в своей области.

Примеры:

– 1С Предприятие;
– БЭСТ;
– Правовая система Гарант.

Разрабатываемые под конкретного заказчика. Максимально учитывают нужды потребителя, его ситуацию и не требуют дополнительных знаний от пользователя. Они весьма дороги и требуют времени для создания, отладки и внедрения.

В настоящее время все чаще и чаще люди приходят за разработкой ПО для работы с какими-либо данными.

Порядок выполнения работы

С предыдущих работ у нас остались модели данных ORM, созданные с использованием Django-Rest-Framework. Используя модели, напишем сериалайзеры (рисунок 25). Они необходимы для преобразования данных из json в понятный языку программирования формат.

```
class StudentSerializer(serializers.ModelSerializer):

    class Meta:
        model = StudentModel
        fields = '__all__'


class DisciplineSerializer(serializers.ModelSerializer):

    class Meta:
        model = DisciplineModel
        fields = '__all__'


class SemestrSerializer(serializers.ModelSerializer):

    class Meta:
        model = SemesterModel
        fields = '__all__'


class GradeSerializer(serializers.ModelSerializer):
    class Meta:
        model = GradesModel
        fields = '__all__'
```

Рисунок 25 – Сериалайзеры для БД

В классе «Meta» выбираем модель данных и поля которые собираемся использовать. Далее, вспоминая паттерн MVC, добавляем представления (рисунок 26).

```
class StudentViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = StudentModel.objects.all()
    serializer_class = StudentSerializer


class DisciplineViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = DisciplineModel.objects.all()
    serializer_class = DisciplineSerializer


class SemestrViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = SemesterModel.objects.all()
    serializer_class = SemestrSerializer


class GradeViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = GradesModel.objects.all()
    serializer_class = GradeSerializer
```

Рисунок 26 – Представления для БД

Далее необходимо прописать адреса url для конкретных представлений (рисунок 27).

```
from django.contrib import admin
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from api.views import *

router = DefaultRouter()
router.register(r'students', StudentViewSet)
router.register(r'discipline', DisciplineViewSet)
router.register(r'semestrs', SemestrViewSet)
router.register(r'grades', GradeViewSet)

urlpatterns = [
    path('', include(router.urls)),
    path('admin/', admin.site.urls),
]
```

Рисунок 27 – URL-адреса

Далее необходимо проверить результаты работы. Для этого необходимо запустить сервер и перейти по адресу /students (рисунок 28).

```
Student List

GET /students/

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "fio": "Иванов Иван Иванович",
        "birthday": "24.04.1997",
        "sex": 1,
        "address": "Лыткина 10",
        "phone": "89999999999"
    },
    {
        "id": 3,
        "fio": "Петров Петр Петрович",
        "birthday": "11.11.2000",
        "sex": 1,
        "address": "Лыткина 10",
        "phone": "87777777777"
    },
    {
        "id": 4,
        "fio": "Петров Илья Петрович",
        "birthday": "11.11.2000",
        "sex": 1,
        "address": "Лыткина 10",
        "phone": "89991112233"
    }
]
```

Рисунок 28 – Результаты работы

В результате были получены данные с базы данных и развернули полноценное API с помощью Django-Rest-Framework.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

2.1 Общие положения

Целями самостоятельной работы являются систематизация, расширение и закрепление теоретических знаний в области различных аспектов современных СУБД.

Самостоятельная работа студента по дисциплине «Современные СУБД» включает следующие виды деятельности:

1) проработка лекционного материала, в том числе подготовка к тестированию;

2) подготовка к лабораторным работам;

3) подготовка к промежуточной аттестации.

В ходе самостоятельной работы студент, ориентируясь на изложенные рекомендации, планирует свое время и перечень необходимых работ в зависимости от индивидуальных психофизических особенностей. Формат самостоятельной работы студентов может отличаться в зависимости от формы обучения и объема аудиторной работы.

2.2 Проработка лекционного материала и подготовка к лабораторным работам

Для качественного усвоения учебного материала целесообразно осуществлять проработку лекционного материала, которая направлена как на систематизацию имеющегося материала, так и на подготовку к освоению практических аспектов, связанных с содержанием дисциплины.

Проработка лекционного материала включает деятельность, связанную с изучением рекомендуемых преподавателем источников, в которых отражены основные моменты, затрагиваемые в ходе лекций. Кроме того, важное место отведено работе с собственноручно составленным конспектом лекций. При конспектировании во время лекции помните, что не следует записывать все, что говорит и/или демонстрирует лектор: старайтесь выявить главное и записать только это. Цель конспекта – формирование целостного логически выстроенного взгляда на круг вопросов, затрагиваемых в ходе изучения соответствующей темы, а не механическая фиксация текстовой и графической информации.

Во внеаудиторное время проработка лекционного материала может быть выстроена в двух основных форматах:

а) отработка прослушанной лекции (прочтение конспекта и рекомендованных преподавателем источников с сопоставлением записей) и восполнение пробелов, если они имелись (например, если студент не понял чего-то, не успел записать);

б) прочтение перед каждой последующей лекцией предыдущей, дабы не тратилось много времени на восстановление контекста изучения дисциплины при продолжающейся или связанной теме.

В ходе проработки лекционного материала обращайте внимание на контрольные вопросы, которые, как правило, имеются в конце каждой темы учебника (учебного пособия). Отвечая на них, можно сделать вывод о степени понимания материала. Если ответы на какие-то вопросы вызвали затруднения, то следует предпринять еще одну попытку изучения отдельных вопросов.

При подготовке к лабораторным работам необходимо заранее изучить методические рекомендации по его проведению, обратить внимание на цель, формат и содержание занятия.

Если какие-то моменты вызвали дополнительные вопросы, целесообразно обратиться к содержанию лекционного материала, рекомендациям преподавателя по изучению теоретической части курса (рекомендуемым источникам) или за личной консультацией. В ходе подготовки к лабораторным работам может потребоваться обращение к различным источникам. Проявляйте инициативу и самостоятельность в данном вопросе. При этом следует пользоваться только авторитетными изданиями, как печатными, так и электронными.

2.3 Подготовка к промежуточной аттестации

Подготовка к экзамену (зачету) включает в себя изучение теоретического материала, представляющего в интегративном виде содержание дисциплины. Экзаменационный (зачетный) билет содержит по 2 теоретических вопроса.

СПИСОК ЛИТЕРАТУРЫ

1. Гордеев, С. И. Организация баз данных в 2 ч. Часть 1 : учебник для вузов /С. И. Гордеев, В. Н. Волошина. – 2-е изд., испр. и доп. – Москва : Издательство Юрайт, 2018. – 311 с. – URL: https://urait.ru/bcode/421030.

2. Гордеев, С. И. Организация баз данных в 2 ч. Часть 2 : учебник для вузов /С. И. Гордеев, В. Н. Волошина. – 2-е изд., испр. и доп. – Москва : Издательство Юрайт, 2018. – 501 с. – URL: https://urait.ru/bcode/421577.

3. Стасышин, В. М. Базы данных: технологии доступа : учебное пособие для вузов / В. М. Стасышин, Т. Л. Стасышина. – 2-е изд., испр. и доп. – Москва : Издательство Юрайт, 2018. – 164 с. – URL: https://urait.ru/bcode/426121 .