

Реляционные базы данных

Модуль 7

Бот «Расписание»

- Пользователь должен иметь возможность запросить расписание поездов. Для этого он формирует запрос в следующем виде:
Покажи расписание со станции Вольный поселок до станции Свободные лаврики (здесь подчеркнутое меняется, остальная часть фразы постоянна)
 - Разбираемся как работать с реляционной базой данных на примере Postgre SQL

Определение базы данных

- База данных – это файл специального формата, содержащий информацию, структурированную заданным образом
- База данных – это совокупность массивов и файлов данных, организованная по определённым правилам, предусматривающим стандартные принципы описания, хранения и обработки данных независимо от их вида

Определение базы данных

- Одно из основных свойств БД – независимость данных от программы, использующих эти данные. Работа с базой данных требует решения различных задач, основные из них следующие:
 - создание базы;
 - запись данных в базу;
 - корректировка данных;
 - выборка данных из базы по запросам пользователя.

Требования к информации, содержащейся в базе данных

- информация, содержащаяся в базах данных, должна быть:
 - непротиворечивой (не должно быть данных, противоречащих друг другу);
 - неизбыточной (следует избегать ненужного дублирования информации в базе, избыточность может привести к противоречивости – например, если какие – то данные изменяют, а их копию в другой части базы забыли изменить);
 - целостной (все данные должны быть связаны, не должно быть ссылок на несуществующие в базе данные)

Системы управления базами данных

- Система управления базами данных, (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.
- СУБД — комплекс программ, позволяющих создать базу данных и манипулировать данными
- Примеры СУБД:
 - Microsoft SQL Server
 - Postgre SQL
 - Mongo DB

Реляционная модель данных

- база данных представляет собой набор таблиц, связанных друг с другом отношениями
- Сами таблицы ничего не знают друг о друге
- Отношение между таблицами поддерживается реляционной СУБД

Таблица в реляционной модели данных

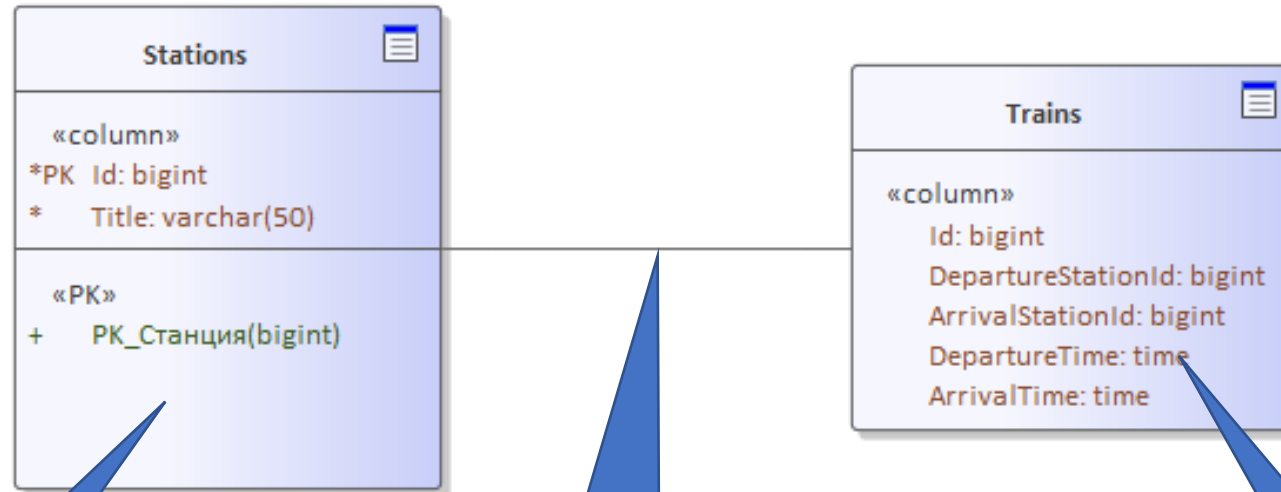
- Таблица представляет собой двумерный массив, в котором хранятся данные
- Столбцы таблицы называются полями, строки – записями
- Количество полей таблицы фиксировано, количество записей – нет
- Фактически таблица – нефиксированный массив записей с одинаковой структурой полей в каждой записи

Таблица в реляционной модели данных

| Номер | Фамилия | Имя |
|-------|---------|---------|
| 1 | Иванов | Петр |
| 2 | Петров | Иван |
| 3 | Сидоров | Николай |

| Номер | Телефон | Местоположение |
|-------|-------------|-----------------|
| 1 | 111-222-333 | Офис 1010 |
| 1 | 333-444-111 | Дом |
| 3 | 555-333-333 | Дом |
| 2 | 111-000-222 | Кабинет 20 |
| 3 | 333-000-111 | Загородная дача |

Проектируем таблицы для задачи «Расписание»



Таблица, которая хранит список станций

Связь между таблицами, указывает, что из Trains есть ссылка на таблицу Stations

Таблица, которая хранит список поездов

Как наполнить таблицы данными и как получить данные из таблиц?

- Для работы с таблицами используется специальный язык структурированных запросов (Structured Query Language)
- SQL поддерживает следующие типы команд:
 - Команды по управлению таблицами базы данных (CREATE, DROP,...)
 - SELECT- выборка записей из таблиц в соответствии с указанными критериями
 - INSERT – вставка одной или несколько записей в таблицу
 - UPDATE – обновление записей таблицы, удовлетворяющих указанным критериям
 - DELETE – удаление записей таблицы, удовлетворяющих указанным критериям

Как работать с таблицами?

- Для изменения состава и структура таблиц используются следующие команды SQL:
 - CREATE TABLE – создание таблицы с указанными полями
 - DROP TABLE – удаление таблицы
 - ALTER TABLE – изменение в таблице состава полей

Создание таблиц для задачи «Расписание»

- Изучите нижеследующие команды по созданию таблиц:

```
CREATE TABLE public.stations (  
  id bigint NOT NULL GENERATED ALWAYS AS IDENTITY,  
  title varchar(50) NOT NULL  
);
```

```
CREATE TABLE public.trains (  
  id bigint NOT NULL GENERATED ALWAYS AS IDENTITY,  
  departurestation bigint NULL,  
  arrivalstation bigint NULL,  
  departuretime time(0) NULL,  
  arrivaltime time(0) NULL  
);
```

Подключение к БД Postgre

- Для подключения к БД необходимо сформировать словарь с конфигурацией

```
connection_param = {  
    'dbname': 'train-schedule',  
    'user': 'bot',  
    'password': '*****',  
    'host': '127.0.0.1'  
}
```

Указывается имя
базы данных

Указывается имя
пользователя СУБД

Указывается пароль
пользователя СУБД

Указывается адрес
сервера СУБД

- Данные для конфигурации узнайте у сетевого администратора
- Этот словарь необходимо разместить в файле config.py

Подключение к БД Postgre

```
import config
import psycopg2
from contextlib import closing

with closing(psycopg2.connect(**config.connection_param)) as conn:
    ...
```

Устанавливается
подключение к БД

Если подключение успешно, то в
этом блоке будут дальнейшие
действия с БД

Использование оператора **with** позволяет автоматически закрыть подключение к БД, когда оно станет не нужным!

Взаимодействие с БД через курсор

- Курсор – это виртуальный указатель на базу данных
- После подключения к БД необходимо получить курсор

```
import config
import psycopg2
from contextlib import closing

with closing(psycopg2.connect(**config.connection_param)) as conn:
    with conn.cursor() as cursor:
        ...
```

В этом блоке будут выполняться
дальнейшие шаги по работе с данными
БД

Получение курсора

Использование оператора **with** позволяет автоматически закрыть курсор, когда он станет не нужным!

Создание таблицы через курсор

```
import config
import psycopg2
from contextlib import closing

with closing(psycopg2.connect(**config.connection_param)) as conn:
    with conn.cursor() as cursor:
        sql_cmd = """CREATE TABLE public.stations (
id bigint NOT NULL GENERATED ALWAYS AS IDENTITY,
title varchar(50) NOT NULL
);"""
        cursor.execute(sql_cmd)
        conn.commit()
```

Команда SQL
располагается в
обычной строке

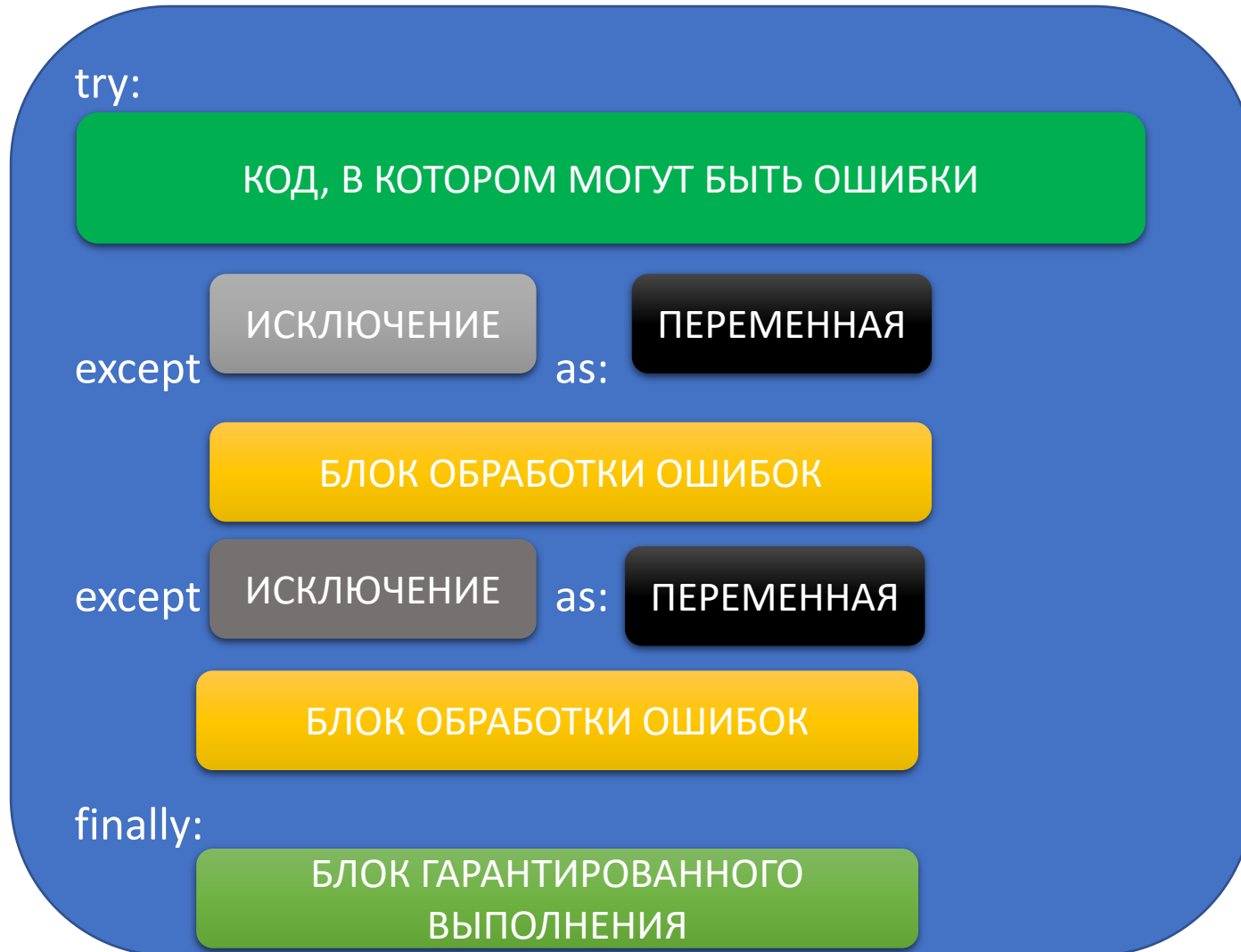
Подготовка команды

Запуск подготовленных
команд на выполнение

Создание таблиц для задачи «Расписание»

- Дополните вашу программу, которая при старте будет создавать таблицы БД
- Примените команду `DROP TABLE STATIONS` для удаление таблицы, если она уже была создана на предыдущем шаге. Используйте курсор и его метод `execute`
- Примените последовательно команды по созданию таблиц, согласно предыдущему упражнению и после этого вызовите метод `commit()` подключения к БД

Обработка ошибок при работе с БД



- Имя исключения, связанного с той или иной ошибкой нужно смотреть в документации библиотеки `psycopg2`!

Опасность SQL-инъекции при формировании SQL-запросов

- Необходимо использовать специальный модуль sql для формирования строки запроса, чтобы в запрос нельзя было передать SQL-инъекцию
- SQL-инъекция – специальным образом сформированная команда призванная нанести вред при выполнении обычной команды SQL

Обычная команда, которая удаляет запись, в которой поле ID содержит значение 2

Дополнение, внедренное хакером приведет к стиранию всех записей таблицы.

DELETE FROM TABLE WHERE ID=2 **OR ID <> 0**

Команда удаляет все записи, в которых поле ID содержит значение 2, а также все поля в которых значение ID не содержит ноль

Вставка данных

```
INSERT INTO ИМЯ ТАБЛИЦЫ ( ( КОЛ1, КОЛ2, ... ) VALUES ( ЗНАЧКОЛ1, ЗНАЧКОЛ2, ... )
```

Вставка данных

```
import config
import psycopg2
from psycopg2 import sql
from contextlib import closing

with closing(psycopg2.connect(**config.connection_param)) as conn:
    with conn.cursor() as cursor:
        columns = ("title",)
        values = ("Вольный поселок",)
        cmd = sql.SQL('INSERT INTO public."stations" ({})) VALUES ({})) RETURNING id').
            format(
                sql.SQL(',').join(map(sql.Identifier, columns)),
                sql.SQL(',').join(map(sql.Literal, values)))
        cursor.execute(cmd)
        conn.commit()
        id = cursor.fetchone()[0]
```

Указываются в
кортеже все колонки, в
которые вставляются
данные

Указываются в
кортеже все значения
колонок, в которые
вставляются данные

RETURNING id
позволит получить
суррогатный ключ
только что
вставленной записи

Непосредственное
получение значения
суррогатного ключа только
что вставленной записи

Вставка данных с временем и датой

Для формирования значения даты или времени модулем datetime

```
from datetime import datetime
now = datetime.now() # текущее время
year = now.strftime("%Y")
print("year:", year)

month = now.strftime("%m")
print("month:", month)

day = now.strftime("%d")
print("day:", day)

time = now.strftime("%H:%M:%S")
print("time:", time)

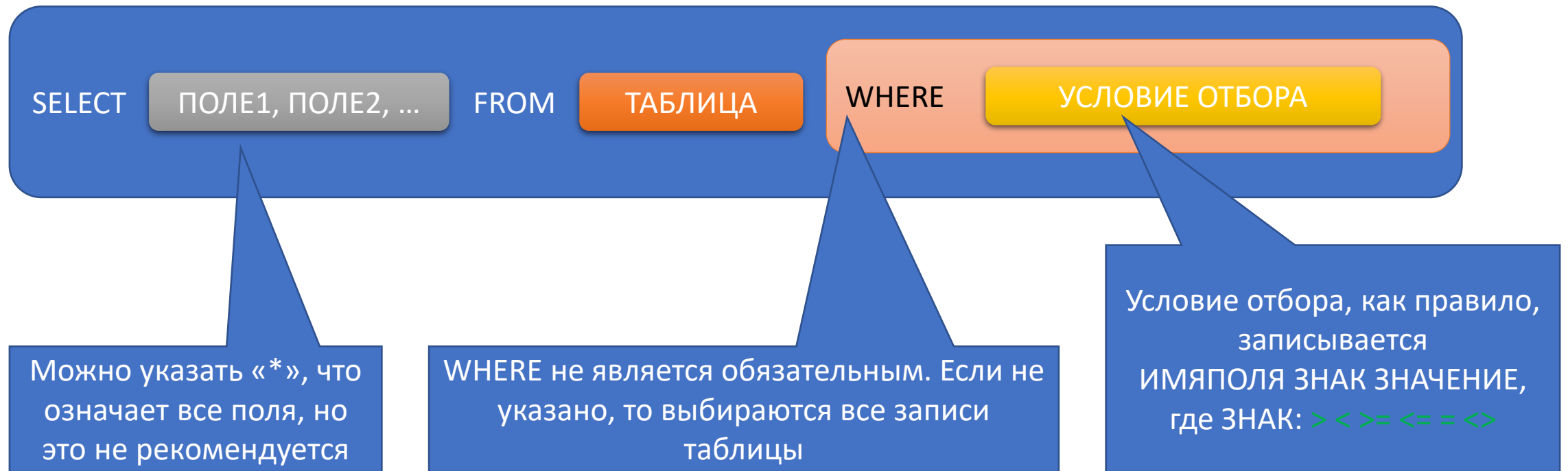
date_time = now.strftime("%m/%d/%Y, %H:%M:%S")
print("date and time:", date_time)
```

Вставка данных для задачи «Расписание»

- Удалите таблицы с помощью команды DROP TABLE, которую передайте аналогичным образом через курсор
- Измените тип полей «departuretime» и «arrivaltime» на тип «datetime» для чего внесите исправления в примеры в каталоге Module7\Live
- Разработайте программу, которая выполняется отдельно от бота и формирует расписание по вашим станциям на несколько дней
- Для формирования даты используйте функции из пакета datetime, создающие строки, содержащие дату

Выборка данных

- Для выборки данных используют запрос SQL по одной или нескольким таблицам



Обработка полученных данных от SQL-запроса

- После выполнения запроса вы можете воспользоваться следующими методами объекта курсор:
 - `fetchone()` – возвращает кортеж, соответствующей одной текущей записи из базы данных, выбранных командой `SELECT`
 - `fetchall()` – возвращает кортеж кортежей, которые соответствуют всем выбранным записям из базы данных командой `SELECT`

Обработка полученных данных от SQL-запроса

```
import psycopg2
from psycopg2 import sql
from contextlib import closing

with closing(psycopg2.connect(**config.connection_param)) as conn:
    with conn.cursor() as cursor:
        columns = ("id", "title")
        where = "Поселение хоббитов"
        cmd = sql.SQL('SELECT {} from public."stations" where {}'.format(
            sql.SQL(',').join(map(sql.Identifier, columns)),
            sql.SQL('title = {}'.format(sql.Literal(where))
        )
        cursor.execute(cmd)
        recordset = cursor.fetchall()
        for c in recordset:
            print(f"id={c[0]} title={c[1]}")
```

Получаем записи,
выбранные командой
SELECT

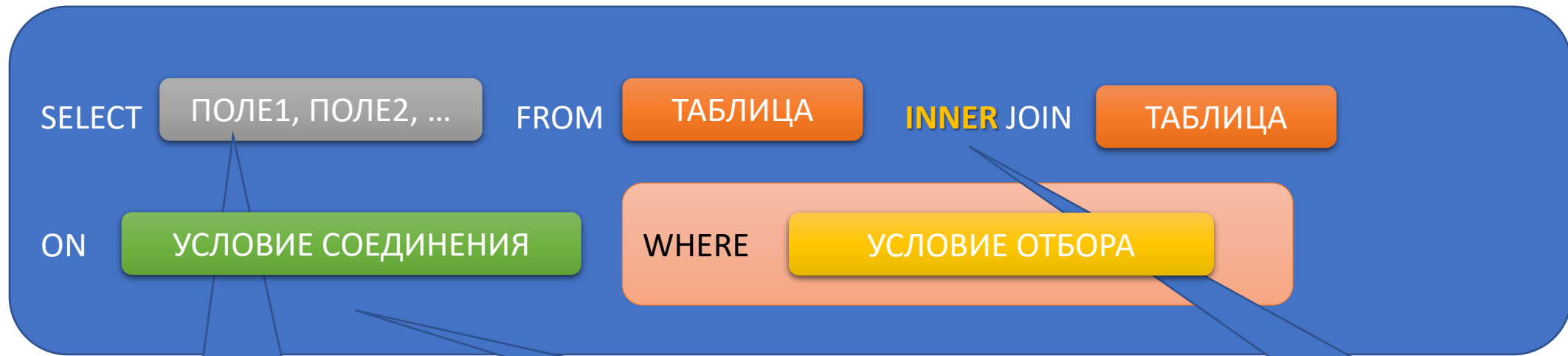
Обрабатываем
каждую запись

Выборка из нескольких таблиц

| Номер | Фамилия | Имя |
|-------|---------|---------|
| 1 | Иванов | Петр |
| 2 | Петров | Иван |
| 3 | Сидоров | Николай |

| Номер | Телефон | Местоположение |
|-------|-------------|-----------------|
| 1 | 111-222-333 | Офис 1010 |
| 1 | 333-444-111 | Дом |
| 3 | 555-333-333 | Дом |
| 2 | 111-000-222 | Кабинет 20 |
| 3 | 333-000-111 | Загородная дача |

Выборка из нескольких таблиц



Здесь ПОЛЕ указывается в формате ТАБЛИЦА.ИМЯПОЛЯ, чтобы можно было отображать поля из нескольких таблиц

Условие соединения, как правило указывает, что поле одной таблицы, равно полю другой таблицы

- Таких INNER JOIN может быть несколько...
- Может быть LEFT JOIN и т.д.

Выборка из нескольких таблиц (INNER JOIN)

```
import psycopg2
from psycopg2 import sql
from contextlib import closing
import config

with closing(psycopg2.connect(**config.connection_param)) as conn:
    with conn.cursor() as cursor:
        columns = ("departuretime", "arrivaltime", "title")

        cmd = sql.SQL('SELECT {} from public."trains" inner join public."stations"'
on trains.departurestation = stations .id ').format(
            sql.SQL(',').join(map(sql.Identifier, columns)),
        )
        cursor.execute(cmd)
        recordset = cursor.fetchall()
        for c in recordset:
            print(c[0], c[1], c[2])
```

Выборка из нескольких таблиц

- Используя INNER JOIN мы можем объединить несколько таблиц по условиям
- Для корректного отображения расписания нам необходимо вместо идентификаторов станций в таблице «trains» подставить их имена
- Для этого следует использовать, в нашем случае, LEFT JOIN

Выборка из нескольких таблиц (LEFT JOIN)

```
import psycopg2
from psycopg2 import sql
from contextlib import closing
import config

with closing(psycopg2.connect(**config.connection_param)) as conn:
    with conn.cursor() as cursor:
        columns = ("source.title", "destination.title", "trains.departuretime", "trains.arrivaltime")

        cmd = sql.SQL("""
        select source.title ,destination.title , departuretime, arrivaltime from trains
left join stations as source on source.id = trains.departurestation
left join stations as destination on destination.id = trains.arrivalstation """)
        cursor.execute(cmd)
        recordset = cursor.fetchall()
        for c in recordset:
            print(c[0],c[1],c[2],c[3])
```


Выполнение SQL запроса на выборку данных для задачи «Расписание»

- Напишите скрипт, который показывает расписание поездов по станции отправления
- Напишите скрипт, который показывает расписание поездов по станции отправления
- Напишите скрипт, который показывает расписание поездов в определенный интервал времени. Вам необходимо будет задать сложное логическое условие вида **ПОЛЕ >= ЗНАЧ AND ПОЛЕ <= ЗНАЧ**

Самостоятельное изучение

- Попробуйте самостоятельно поменять расписание по станции, написав для этого соответствующий скрипт отдельно от бота
- Для этого изучите формат запроса UPDATE, который может обновлять записи
- Для UPDATE использование условия WHERE необходимо, иначе все записи будут обновлены одним и тем же значением
- Команда UPDATE имеет следующий формат:
- **UPDATE** ИМЯТАБЛИЦЫ **SET** КОЛОНКА1 = ЗНАЧЕНИЕ1, КОЛОНКА2 = ЗНАЧЕНИЕ2, ... **WHERE** УСЛОВИЕ

Завершение бота «Расписание»

- Напишите бота, который будет по запросу пользователя возвращать расписание поездов по станции
- Вам необходимо сформировать SELECT запрос, полученные данные превратить в строку и вернуть как результат функции, вызванной ботом

NoSQL базы данных

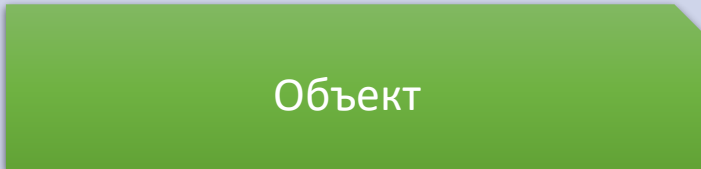
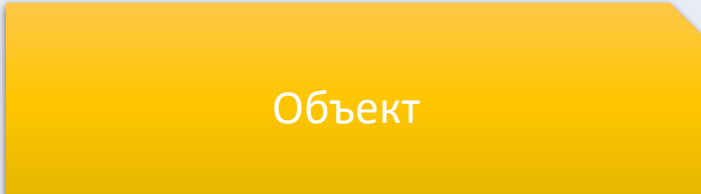
Модуль 8

Задача «Бот-эксперт»

- Пользователь пишет некоторый вопрос. По ключевым словам, которые, выделяет бот, производится поиск в базе данных и выдается ответ пользователю. Если ни по одному ключевому слову нет информации, бот должен сообщить об этом пользователю
 - Узнаем что такое реляционные базы данных
 - Научимся работать с базой данных средствами Python

NoSQL базы данных

- NoSQL базы данных не используют реляционную модель таблиц
- NoSQL базы данных хранят документы – произвольные структурированные объекты
- NoSQL базы данных используют таблицы «Ключ-значение» для хранения и быстрого доступа к данным

| Ключ | Значение |
|--|--|
| {8AA263F5-C273-49B2-B61F-4812230B1816} |  Объект |
| {8025F5FD-D417-4EA6-B5DD-D3D3BA0ABB8F} |  Объект |

Ключ

- Ключ должен быть уникальным
- Ключ может быть:
 - Строкой
 - Числом
 - Специальным идентификатором (GUID)

NoSQL СУБД

- Документоориентированные СУБД
 - CouchDB
 - Couchbase
 - MongoDB
 - eXist
 - Berkeley DB XML

MongoDB

- MongoDB — документоориентированная система управления базами данных, не требующая описания схемы таблиц.
- Считается одним из классических примеров NoSQL-систем
- Использует JSON-подобные документы и схему базы данных

MongoDB

- Система поддерживает запросы:
 - они могут возвращать конкретные поля документов и пользовательские JavaScript-функции
 - Поддерживается поиск по регулярным выражениям.
 - Можно настроить запрос на возвращение случайного набора результатов
- Система может быть использована в качестве файлового хранилища с балансировкой нагрузки и репликацией данных
- Может работать в соответствии с парадигмой MapReduce
- Поддерживается JavaScript в запросах, функциях агрегации (например, в MapReduce).

Соответствие терминов реляционных баз данных и MongoDB

| Реляционные базы данных | MongoDB |
|-------------------------|-------------------|
| Database | Database |
| Table | Collection |
| Column | Field |
| Primary Key | Primary Key |
| Table Join | Embedded Document |

Подключение к MongoDB

```
import pymongo
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')
db = client['bot']
```

Создание объекта для связи с MongoDB. Узнайте параметры подключения у вашего администратора

Получения доступа к объекту базы данных. Если такой базы данных нет, она будет создана

Вставка документа

- Для формирования объекта с данными используется словарь
- Объекты, могут быть сколь угодно сложными
- После сохранения, каждый документ получает атрибут «_id», в котором хранится уникальный ключ объекта

Вставка одного документа

```
import pymongo
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')
db = client['bot']
article = {"author": "Derrick Mwiti",
          "about": "Introduction to MongoDB and Python",
          "tags":
            ["mongodb", "python", "pymongo"]}
articles = db.articles
result = articles.insert_one(article)
print("First article key is: {}".format(result.inserted_id))
```

Словарь-документ для
сохранения в базе
данных

Получение доступа к
коллекции. Если
коллекции нет, она
будет создана

Вставка документа

Уникальный ключ
сохраненного документа

Вставка нескольких документов

```
import pymongo
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')
db = client['bot']
article = {"author": "Derrick Mwiti",
          "about": "Introduction to MongoDB and Python",
          "tags":
              ["mongodb", "python", "pymongo"]}
articles = db.articles
article1 = {"author": "Emmanuel Kens",
           "about": "Knn and Python",
           "tags":
               ["Knn", "pymongo"]}
article2 = {"author": "Daniel Kimeli",
           "about": "Web Development and Python",
           "tags":
               ["web", "design", "HTML"]}
new_articles = articles.insert_many([article1, article2])
print("The new article IDs are {}".format(new_articles.inserted_ids))
```

Получение документа

```
import pymongo
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')
db = client['bot']
article = {"author": "Derrick Mwiti",
          "about": "Introduction to MongoDB and Python",
          "tags":
              ["mongodb", "python", "pymongo"]}
articles = db.articles
for article in articles.find():
    print(article)
```

Метод find без параметров
возвращает всю коллекцию
документов в виде контейнера
словарей

Получение документа по ключу

```
import pymongo
from pymongo import MongoClient
from bson.objectid import ObjectId
client = MongoClient('mongodb://localhost:27017/')
db = client['bot']
article = {"author": "Derrick Mwiti",
          "about": "Introduction to MongoDB and Python",
          "tags":
              ["mongodb", "python", "pymongo"]}
articles = db.articles
result = articles.insert_one(article)

document = articles.find_one({'_id': ObjectId(result.inserted_id)})
print(document)
```

Импорт функции для
создания ключа

Создание ключа для
поиска

Получение отдельных полей документа

- Первый параметр методов `find_one()` или `find()` указывает условия поиска
- Второй параметр этих методов указывает какие поля требуется отображать, а какие нет

```
document = articles.find_one({'_id': ObjectId(result.inserted_id)},  
                               { "_id": 0, "author": 1, "about": 1 })  
print(document)
```

Условия поиска документов

- MongoDB поддерживает язык запросов для получения документов по конкретным параметрам
- Объект запроса может включать несколько условий

{ **ИМЯ ПОЛЯ** : { **ОПЕРАТОР** : **ЗНАЧЕНИЕ** } }

| Оператор | Условие |
|----------|-------------------------------------|
| \$lt | < |
| \$lte | <= |
| \$gt | > |
| \$gte | >= |
| \$ne | <> |
| \$eq | = |
| \$in | Значение из перечисленного в списке |

Поиск документов по условию

```
import pymongo
from pymongo import MongoClient
from bson.objectid import ObjectId
client = MongoClient('mongodb://localhost:27017/')
db = client['bot']
article = {"author": "Derrick Mwiti",
          "about": "Introduction to MongoDB and Python",
          "tags":
              ["mongodb", "python", "pymongo"]}
articles = db.articles
result = articles.find({"tags" : {"$in": ["mongodb"]}})
for c in result:
    print(c)
document = articles.find_one({"author" : {"$eq" : "Derrick Mwiti"}})
print(document)
```

Возможные варианты для оператора `$in` перечисляются в списке

Оператор `$eq` требует точного совпадения

Изменение документа в коллекции

- Для изменения объекта достаточно вызвать методы `update()` или `update_one()`

```
import pymongo
from pymongo import MongoClient
from bson.objectid import ObjectId
client = MongoClient('mongodb://localhost:27017/')
db = client['bot']
article = {"author": "Derrick Mwiti",
          "about": "Introduction to MongoDB and Python",
          "tags":
              ["mongodb", "python", "pymongo"]}
articles = db.articles
query = { "author": "Derrick Mwiti" }
new_author = { "$set": { "author": "John David" } }

articles.update_one(query, new_author)

for article in articles.find():
    print(article)
```

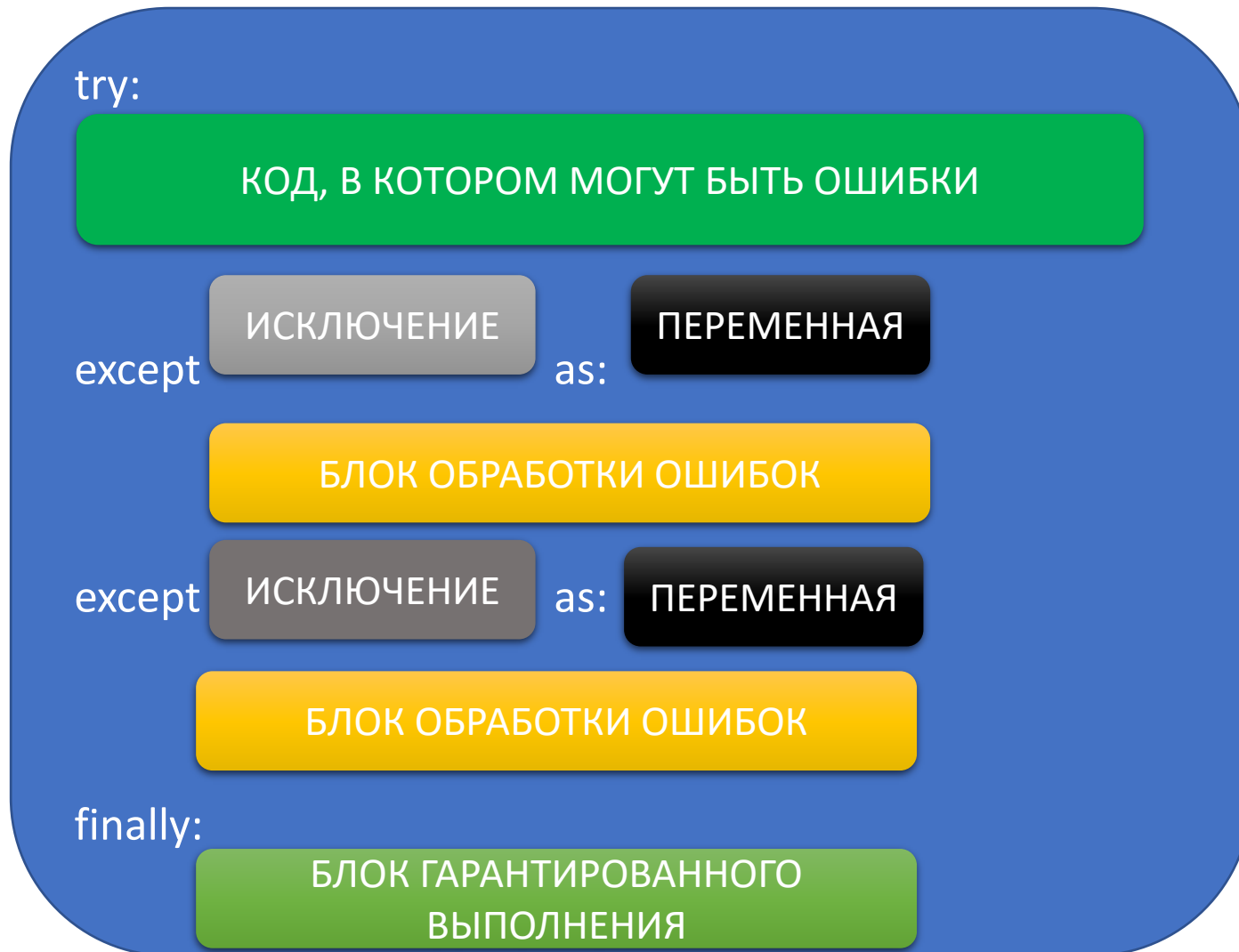
Оператор `$set` в качестве ключа указывает на поля, которые будут изменены

Удаление документа

- Для удаления документа используем методы коллекции `delete()` или `delete_one()`

```
import pymongo
from pymongo import MongoClient
from bson.objectid import ObjectId
client = MongoClient('mongodb://localhost:27017/')
db = client['bot']
article = {"author": "Derrick Mwiti",
          "about": "Introduction to MongoDB and Python",
          "tags":
              ["mongodb", "python", "pymongo"]}
articles = db.articles
result = articles.insert_one(article)
db.articles.delete_one({"_id": ObjectId(result.inserted_id)})
```

Обработка ошибок при работе с MongoDB



- Имя исключения, связанного с той или иной ошибкой нужно смотреть в документации библиотеки pymongo!