# Лекция 2
# БД и ORM

Проектирование систем и продуктовая веб-разработка

Канев Антон Игоревич

# Повторение БД

- Курс по Postgre в Jupyter ноутбуках https://aiintro.wiki.iu5edu.ru/docs/db/
- Курс PostgreSQL с 4 семестра
- https://github.com/iu5git/Database
- Индексы – специальная структура в БД, которая позволяет уменьшить время чтения (поиска) данных. Одно из доп. заданий в ДЗ

```
%%sql
INSERT INTO people (person_id, first_name, last_name)
VALUES (9223372036854775807, 'Johnathan', 'Smith')
```

```
%%sql
INSERT INTO people (first_name, last_name)
VALUES ('John', 'Smith')
```

Выше возникла ошибка `database or disk is full`.

## Индексы

О задаче проектирования индексов здесь.

**Индексы** – это специальные таблицы, которые могут быть использованы поисковым двигателем базы дан

В реляционных базах данных таблица представляет собой список строк. В то же время каждая строка им Каждая строка также имеет последовательный порядковый номер `rowid`, используемый для идентифик таблицу как список пар: (`rowid`, `row`).

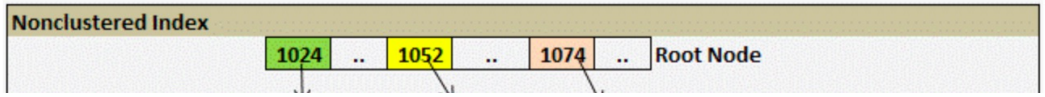
В отличие от таблицы, индекс имеет противоположное отношение: (`row`, `rowid`). **Индекс** - это дополнит производительность запроса.

Каждый индекс должен быть связан с определенной таблицей. Индекс состоит из одного или нескольких одной таблице. Таблица может иметь несколько индексов.

Всякий раз, когда вы создаете индекс, SQLite создает структуру B-дерева (по-русски произносится как Б

Индекс содержит данные из столбцов, указанных в индексе, и соответствующее `rowid`. Это помогает SQ индексированных столбцов.

Представьте себе индекс в базе данных, как индекс книги. Глядя на индекс, вы можете быстро определи

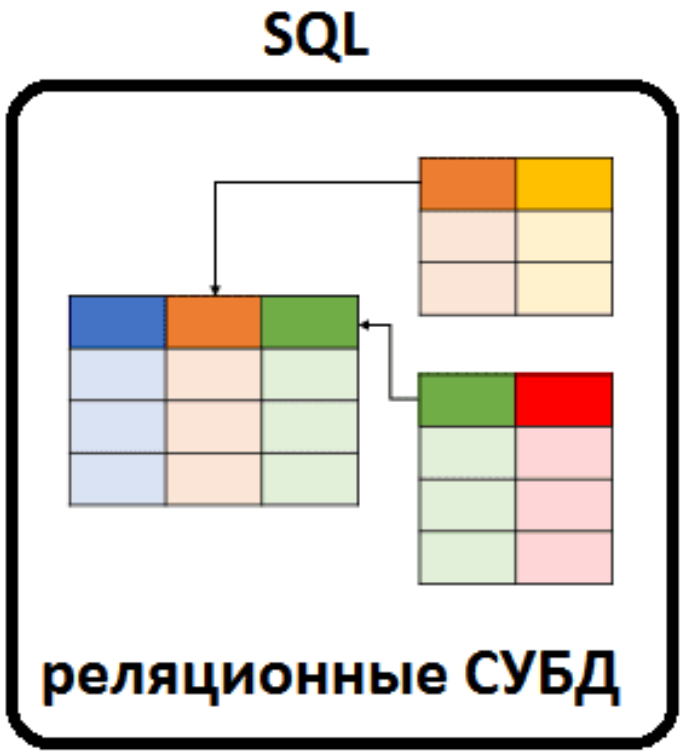


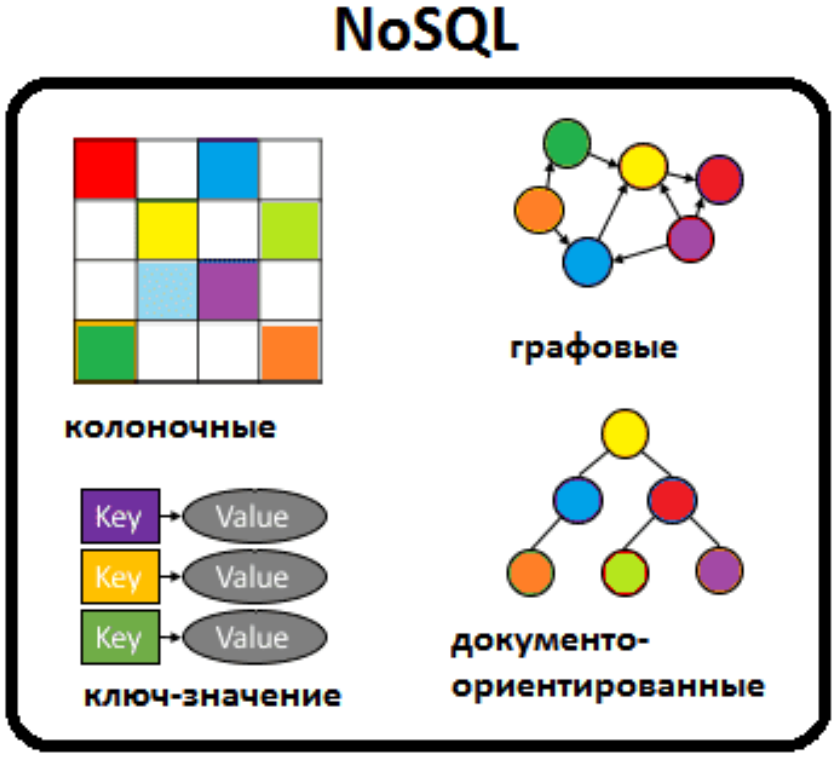
# *Виды баз данных

- Реляционные **Универсальность+SQL**

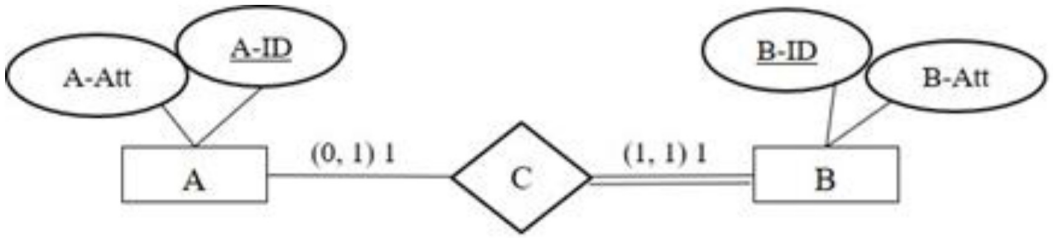
Специализированные:

- Документо-ориентированные **просто как JSON**
- in memory **Redis, скорость**
- Графовые
- Ключ-значение **Redis, как индекс**
- Колоночные **По колонкам**



# Виды баз данных




Tables:
A (A-ID, A-Att)
B (B-ID, B-Att, A-ID)

a. Relational Database ER and Tables

<CF> A

<row key> #A-ID

+A-Att

<CF> B

<row key> #B-ID
+B-Att
+A-ID

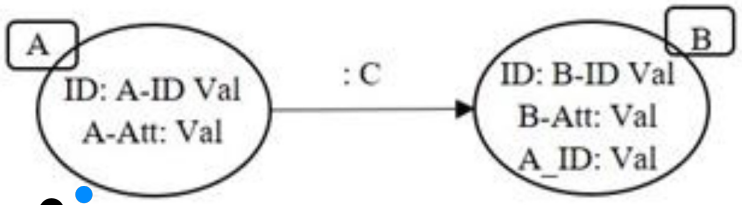
b. Column-based NoSQL Columns

```
A Collection
{
   _id: < A-ID val>,
   A-Att : "       ",
   B {
        B-ID :<     >
        B-Att : "       "
      }
}
```

```
B Collection
{
   _id : <B-ID val>,
   B-Att : "       "
}
A Collection
 {
   _id: < A-ID val>,
   A-Att : "       "
   B-ID : <       >,
}
```

1-Embedded document 2-Referencing document

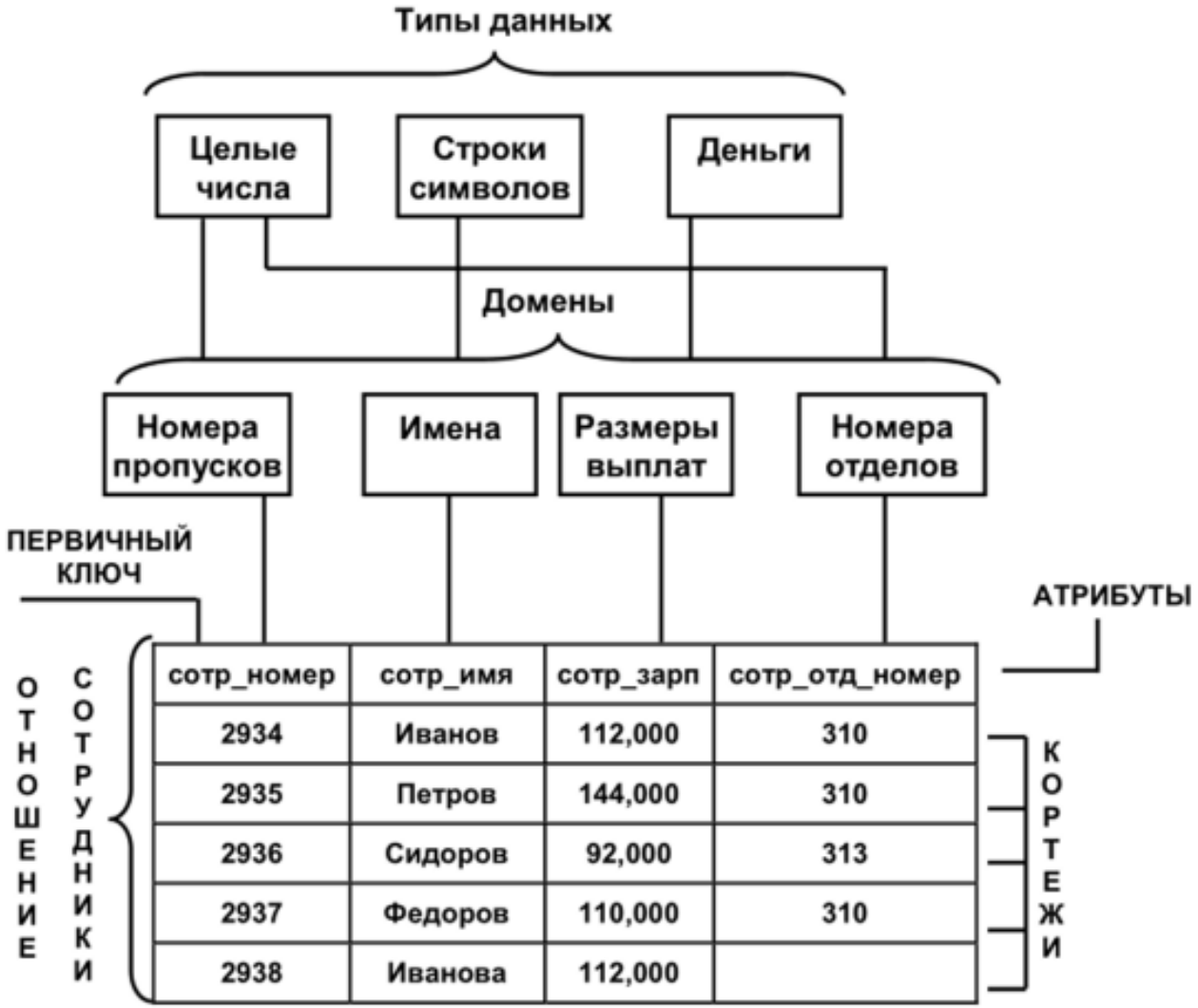
c. Document-based NoSQL Collections




d. Graph-based NoSQL Nodes

# *Реляционные базы данных

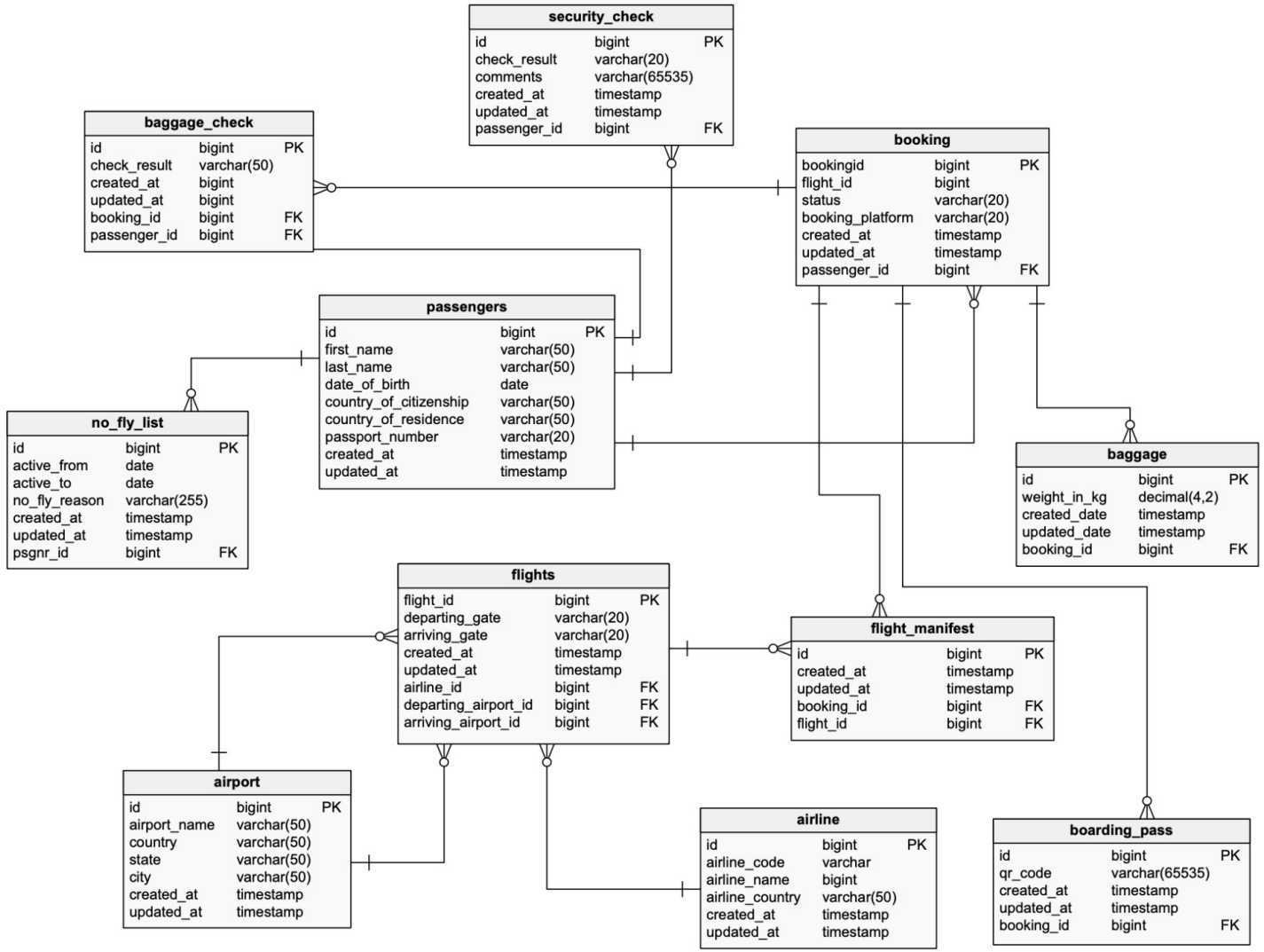
- Универсальные, мощный язык SQL
- Relation – отношение, связь. В ней домены, кортежи, атрибуты
- Исходно имеем одну большую таблицу как в Excel, ее разделяем, т.е. нормализуем
- Из Реляционной алгебры: нормализация данных для устранения аномалий и избыточности.



# ER диаграмма

- Отношения - таблицы
- Атрибуты и типы данных
- Первичные и внешние ключи

- В этом примере центральная таблица flight_manifest
- мы можем дописать к ней колонки почти из всех таблиц



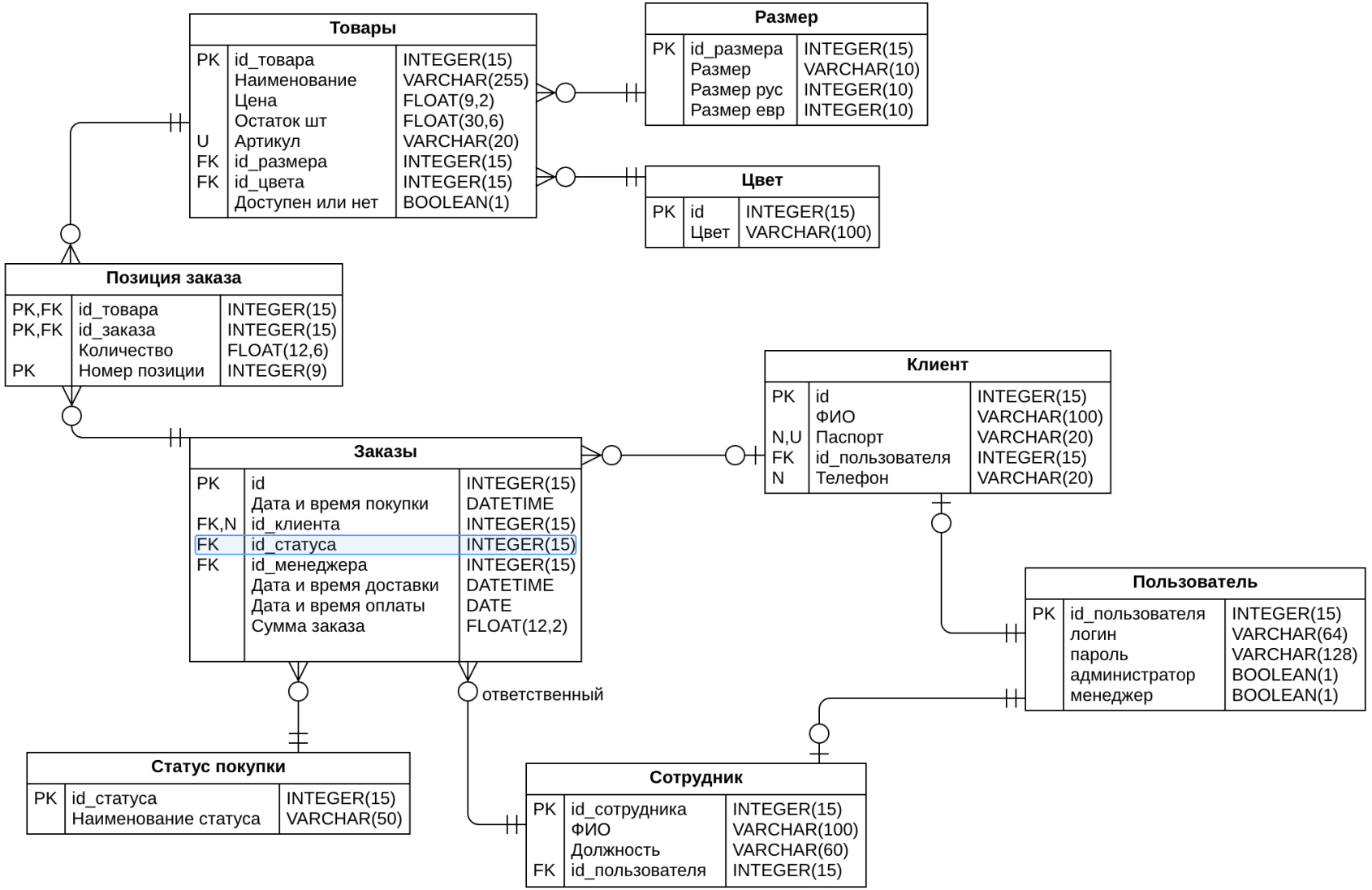
# *ER в StarUML

- Логическое удаление soft delete

- Товар или заказ может быть **0** позиций, либо их **много**
- У позиции **один** заказ и **один** товар

Более сложный пример чем в курсе

- В курсе не делаем справочники: размер, цвет, статус



# Типы данных

- Дата
- Целочисленный
- Вещественный
- Строковый

- В каждой СУБД **свои!**
- **Длина поля** в битах, символах

| | db2 | mysql | openbase | oracle |
|---|---|---|---|---|
| **:binary** | blob(32768) | blob | object | blob |
| **:boolean** | decimal(1) | tinyint(1) | boolean | number(1) |
| **:date** | date | date | date | date |
| **:datetime** | timestamp | datetime | datetime | date |
| **:decimal** | decimal | decimal | decimal | decimal |
| **:float** | float | float | float | number |
| **:integer** | int | int(11) | integer | number(38) |
| **:string** | varchar(255) | varchar(255) | char(4096) | varchar2(255) |
| **:text** | clob(32768) | text | text | clob |
| **:time** | time | time | time | date |
| **:timestamp** | timestamp | datetime | timestamp | date |

# Типы данных

```
<event>
  <data name="SessionID">
    <value>CDE3D5</value>
  </data>
  <data name="NTUserName">
    <value>xyz</value>
  </data>
  <data name="NTDomainName">
    <value>ABC</value>
  </data>
  <data name="DatabaseName">
    <value>TestCube1</value>
  </data>
  <data name="ApplicationName">
    <value>Microsoft SQL Server Management Studio</value>
  </data>
  <data name="ServerName">
    <value>SERXYZ</value>
  </data>
  <data name="RequestID">
    <value>A737</value>
  </data>
  <action name="attach_activity_id_xfer" package="package0">
    <value>AE23</value>
  </action>
  <action name="attach_activity_id" package="package0">
    <value>CCCCB490-</value>
  </action>
</event>
```

| CustomerId | FirstName | LastName | Picture varbinary(max) |
|---|---|---|---|
| 235 | 'John' | 'Doe' | 0x3B0E95AE3B292F0B… |
| 236 | 'Sally' | 'Smith' | 0xF3000EEF2932002C… |

- BLOB – бинарные файлы (если храним pdf, картинку в БД)
- CLOB – очень большой текст
- XML – разновидность CLOB

# Форматы дат

- Множество форматов дат и времени
- Можно только дату или дату и время (3 даты у нас в заявке это **datetime**)
- Разные сокращения для месяцев

- Ошибки в обработке даты в разных системах
- На фронтенде форма может быть один, в бэке другой в БД третий. Это разные ОС и компьютеры

- Но есть готовые функции. Прибавить месяц к 31 января?

| | | |
|---|---|---|
| MON | Yes | Abbreviated name of month. |
| MONTH | Yes | Name of month. |
| PM P.M. | Yes | Meridian indicator with or without periods. |
| Q | | Quarter of year (1, 2, 3, 4; January - March = 1). |
| RM | Yes | Roman numeral month (I-XII; January = I). |
| RR | Yes | Lets you store 20th century dates in the 21st century using only two digits.<br>**See Also:** "The RR Datetime Format Element" |
| RRRR | Yes | Round year. Accepts either 4-digit or 2-digit input. If 2-digit, provides the same return as RR. If you do not want this functionality, then enter the 4-digit year. |
| SS | Yes | Second (0-59). |
| SSSSS | Yes | Seconds past midnight (0-86399). |
| TS | Yes | Returns a value in the short time format. Makes the appearance of the time components (hour, minutes, and s |

# *SQL

- DDL – Data Definition Language

CREATE, ALTER, DROP

- DML – Data Manipulation Language

SELECT, INSERT, UPDATE, DELETE

- DCL – Data Control Language

GRANT, REVOKE

- TCL – Transaction Control Language

COMMIT, ROLLBACK, SAVEPOINT

```
SELECT DeptID, SUM(SaleAmount)
 FROM Sales
WHERE SaleDate = '01-Jan-2000'
GROUP BY DeptID
HAVING SUM(SaleAmount) > 1000
```

Примеры **транзакций** в курсе:

- Нужно поменять местами две позиции в заявке: меняем только две строки **вместе**!
- В услуге хранится количество товара на складе и с каждой заявкой нужно это количество уменьшать – в двух таблицах **вместе**

# *Достоинства и недостатки

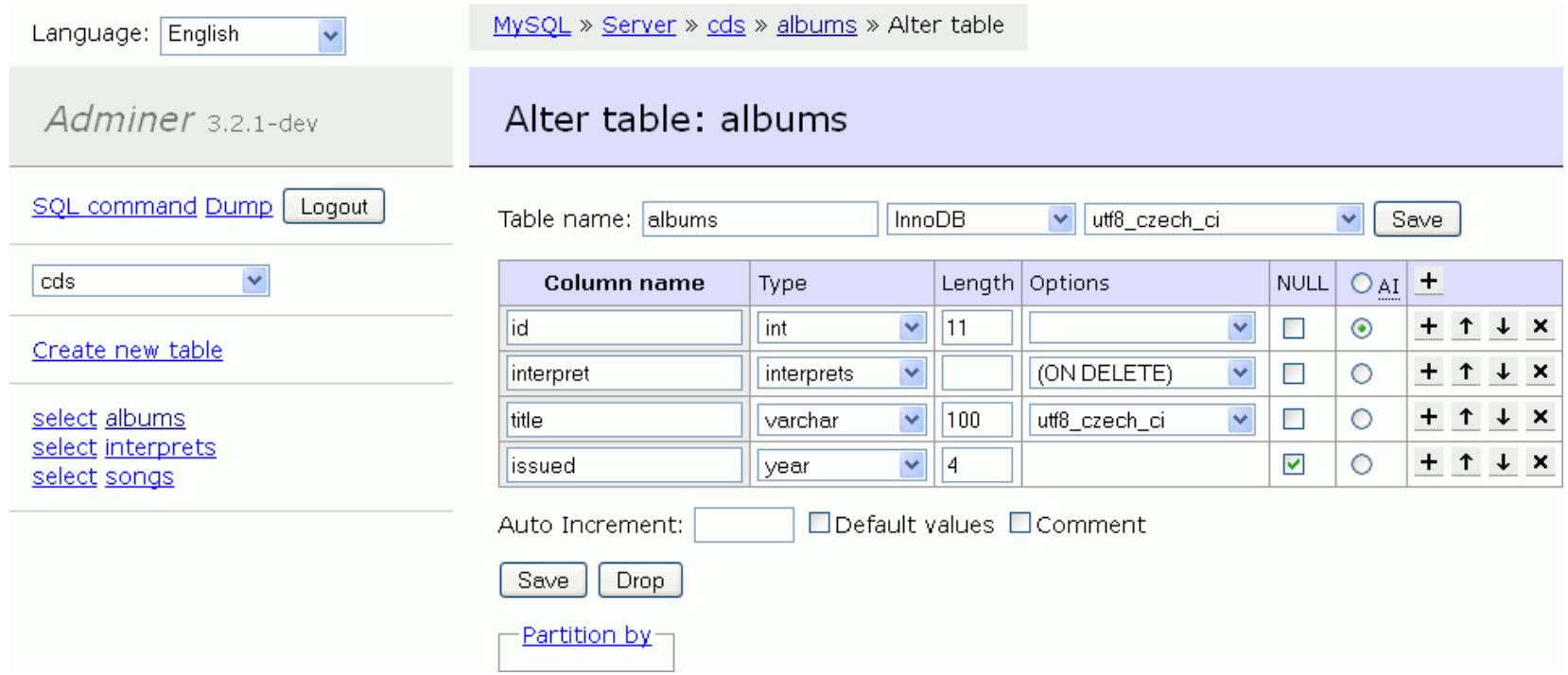
- Борьба с избыточностью и аномалиями
- Высокоуровневый язык запросов SQL

- Необходимость операций соединения (JOIN), которые замедляют скорость
- В реальности не нормализованные данные: если JSON в поле БД

- Теорема CAP – согласованность, доступность, устойчивость к разделению.
- Как забронировать билеты на олимпиаду? Продавать не все сразу

# Интерфейс для администрирования БД

- Adminer – web
- ~~Data Grip - десктоп~~




- Это интерфейс для администратора системы: удалить пользователя, назначить **модератора/менеджера**
- Обычные пользователи тут не работают и у них **нет даже доступа**

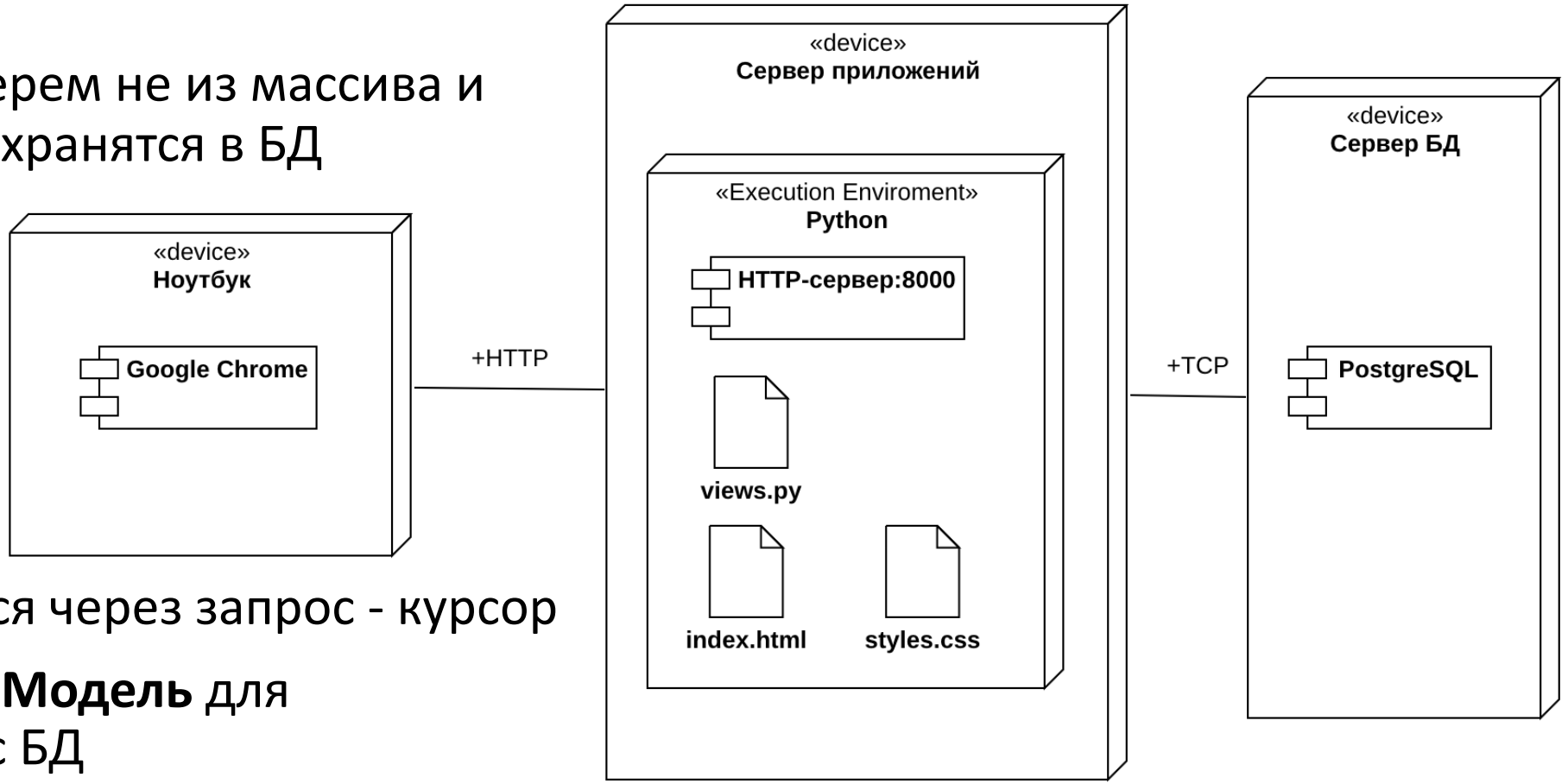
# *Миграции данных (DML)

- Часто требуется создать БД, где часть таблиц уже заполнены данными – типы, виды и другие статичные данные.
- Чтобы перенести данные из одной БД в другую можно использовать разные инструменты
- Не путать с миграциями ORM – есть некоторые отличия

- Например goose (миграции данных, не ORM) для Go
- Или миграции Django ORM для Python (встроено в Django)

# *Трехзвенная архитектура

- Традиционная SSR трехзвенная архитектура
- Теперь данные берем не из массива и не из файла. Они хранятся в БД




- Можем обратиться через запрос - курсор
- Можно добавить **Модель** для взаимодействия с БД

# *Подключение к БД ORM

- Для использования ORM или курсоров нам нужно **сначала подключиться** к БД

```
DB_HOST=0.0.0.0
DB_NAME=bmstu
DB_PORT=5432
DB_USER=bmstu_user
DB_PASS=bmstu_password
```

- **Go, gorm**

```
func FromEnv() string {
        host := os.Getenv("DB_HOST")
        if host == "" {
                return ""
        }

        port := os.Getenv("DB_PORT")
        user := os.Getenv("DB_USER")
        pass := os.Getenv("DB_PASS")
        dbname := os.Getenv("DB_NAME")

        return fmt.Sprintf("host=%s port=%s
}
```

# *Курсоры (DML)

- Использование обычных SQL запросов в коде бэкенда
- Описать текст запроса, выполнить, зафиксировать, закрыть

```
var enough bool
// Query for a value based on a single row.
if err := db.QueryRow("SELECT (quantity >= ?) from album where id = ?",
    quantity, id).Scan(&enough); err != nil {
    if err == sql.ErrNoRows {
        return false, fmt.Errorf("canPurchase %d: unknown album", id)
    }
    return false, fmt.Errorf("canPurchase %d: %v", id)
}
return enough, nil
```

- **Go**

# *Когда выгодно использовать курсоры?

- Если быстрее выполнить select сразу в БД:

```
insert into names (id, name) values (1, 'Лиза');
insert into names (id, name) values (2, 'Андрей');
insert into names (id, name) values (3, 'Наташа');
insert into names (id, name) values (4, 'Анатоль');
insert into names (id, name) values (5, 'Элен');
insert into names (id, name) values (6, 'Пьер');
```

```
insert into matches (first, second) values (1, 2);
insert into matches (first, second) values (2, 1);
insert into matches (first, second) values (3, 2);
insert into matches (first, second) values (2, 3);
insert into matches (first, second) values (3, 4);
insert into matches (first, second) values (6, 5);
insert into matches (first, second) values (3, 6);
insert into matches (first, second) values (6, 3);
```

```
select f_name.name,
       s_name.name
  from matches m
  join names f_name
    on m.first = f_name.id
  join names s_name
    on m.second = s_name.id;
```

| name | name |
|---|---|
| Лиза | Андрей |
| Андрей | Лиза |
| Наташа | Андрей |
| Андрей | Наташа |
| Наташа | Анатоль |
| Пьер | Элен |
| Пьер | Наташа |
| Наташа | Пьер |

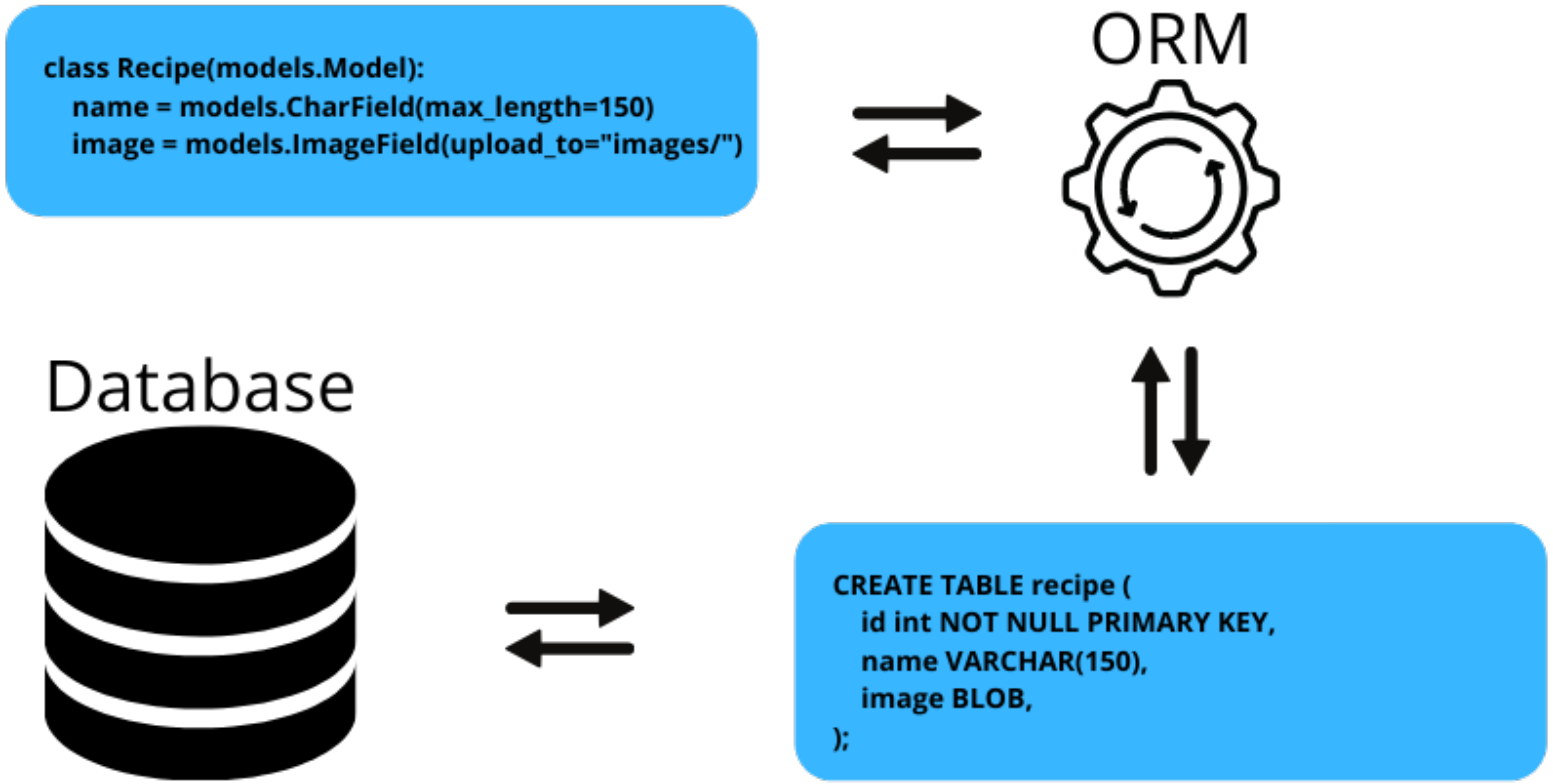
- Если пользователей **много**, ваш алгоритм должен быть **быстрым**

```
select f_name.name,
       s_name.name
  from matches f
  join matches s
    on f.second = s.first
   and f.first < s.first
  join names f_name
    on f.first = f_name.id
  join names s_name
    on f.second = s_name.id
where f.first = s.second;
```

| name | name |
|---|---|
| Лиза | Андрей |
| Андрей | Наташа |
| Наташа | Пьер |

# *ORM

- **Object Relational Mapping**
- Соответствие между Таблицами (на диске) и Моделями (переменные, массивы в ОП)




- Создаете таблицы/объекте в одном месте, миграция сама создает в другом
- Не нужно самим писать запросы

# *Модель и Миграции ORM в gorm (DDL)

- Сначала необходимо создать **классы** - модели для наших таблиц
- Миграции ORM Внесение изменений в модель (**добавить таблицу, колонку и тд**):
- Для создания таблиц в БД (DDL) на основе классов ORM нам требуется выполнить миграции ORM

```
_ = godotenv.Load()
db, err := gorm.Open(postgres.Open(dsn.FromEnv()), &gorm.Config{})
if err != nil {
        panic("failed to connect database")
}

// Migrate the schema
err = db.AutoMigrate(&ds.Product{})
if err != nil {
        panic("cant migrate db")
}
```

```
package ds

import (
        "gorm.io/gorm"
)

type Product struct {
        ID        uint `gorm:"primarykey"`
        Code  string
        Price uint
}
```

# *QuerySet!!! (DML)

- Получить список Услуг – это **не миграция**!
- Для добавления/редактирования/удаления или просто получения данных из БД (DML) **через ORM** нам требуется **QuerySet**
- Аналог **курсора**

```go
product := &ds.Product{}

err := r.db.First(product, "id = ?", "1").Error
if err != nil {
        return nil, err
}

return product, nil
```

- **Go**

# Мастер-класс Golang. Шаблонизация, ORM

В ролике на VK/YouTube рассмотрены:

- Примеры моделей и миграций
- Пример структуры проекта с чистой архитектурой 1, 3 уровней
- Реализация функций просмотра услуг и размера корзины
- Обращение к БД через ORM / SQL

# *Чистая архитектура

- Уровень Обработчиков: представления и контроллеры (обработчики) для нужного потребителя (HTML, JSON/API, куки)

- Уровень Бизнес-логики, набор use-case’ов (сценариев): оплата, регистрация (+отправка e-mail), добавление в корзину, расчеты и тд
- Пример use-case – при оплате заказа заставить гостя пройти регистрацию

- Уровень Сущностей: взаимодействие с БД (обязательные поля при insert, логическое удаление)
- Каждый уровень может меняться отдельно