# Лекция 2
# Базы данных и основы SQL

Tinkoff.ru

# Оглавление

# Откуда берутся данные?

Каждая система в рамках бизнес процесса выполняет свою роль и как следствие собирает свои собственные данные. При аналитике может потребоваться обращаться к данным с разных систем, для этого нужно понимать откуда их взять. А для ускорения работы и уменьшения нагрузки на эти системы, лучше их скопировать и положить в хранилище данных.

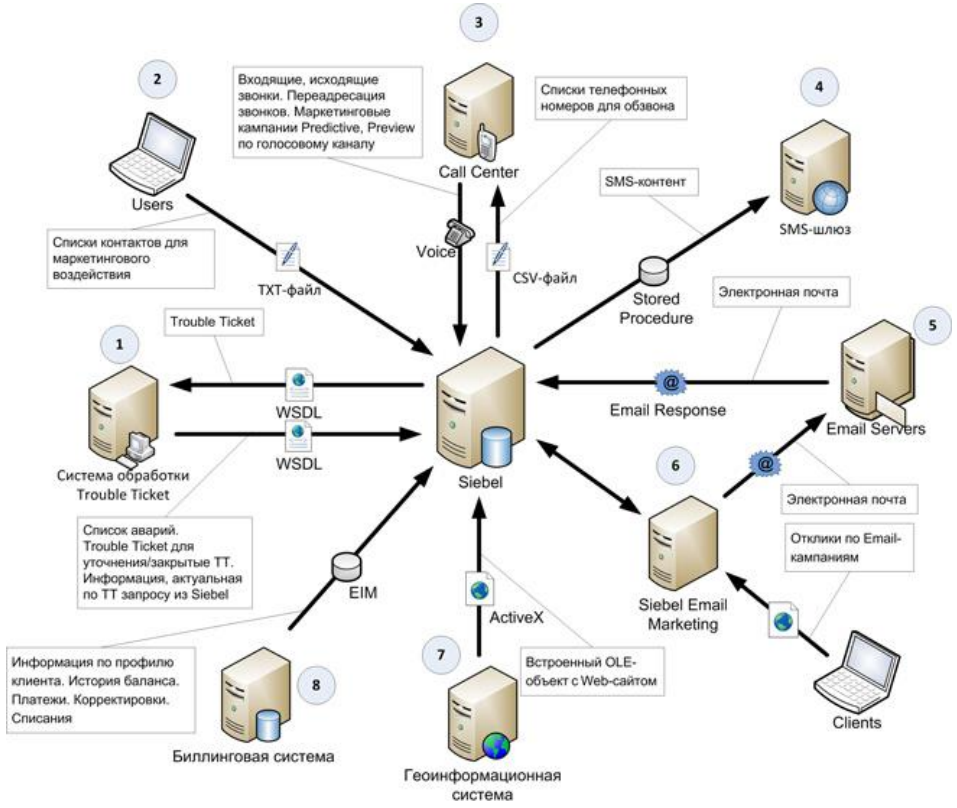


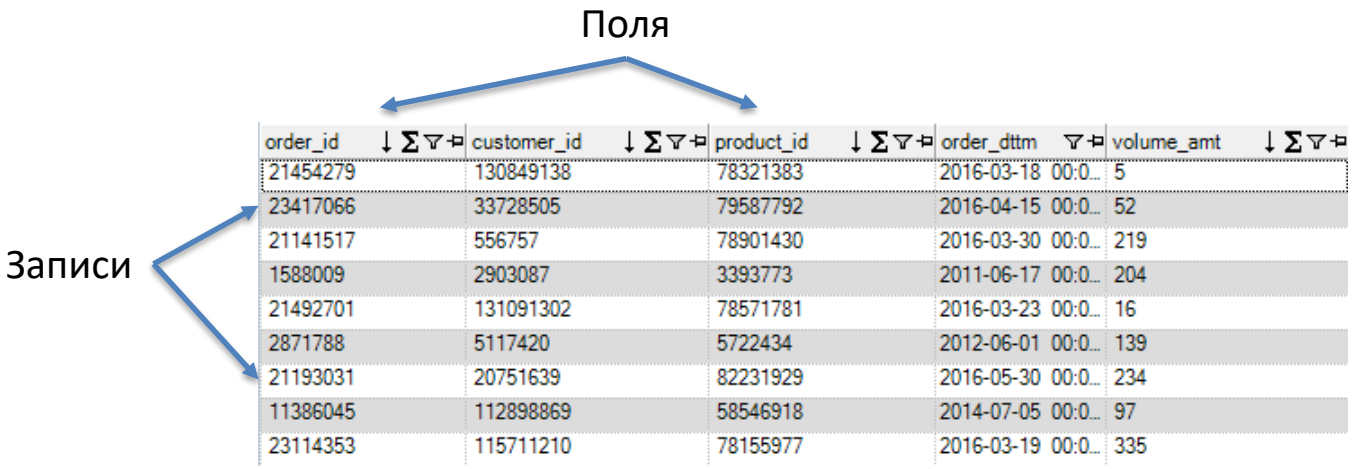

Схема с сайта [http://www.tadviser.ru/](http://www.tadviser.ru/)

# Хранение данных

Данные удобно воспринимать в формате таблиц – это достаточно наглядно, так как в рамках процесса зачастую происходят схожие “явления”, просто каждая со своим набором параметров.

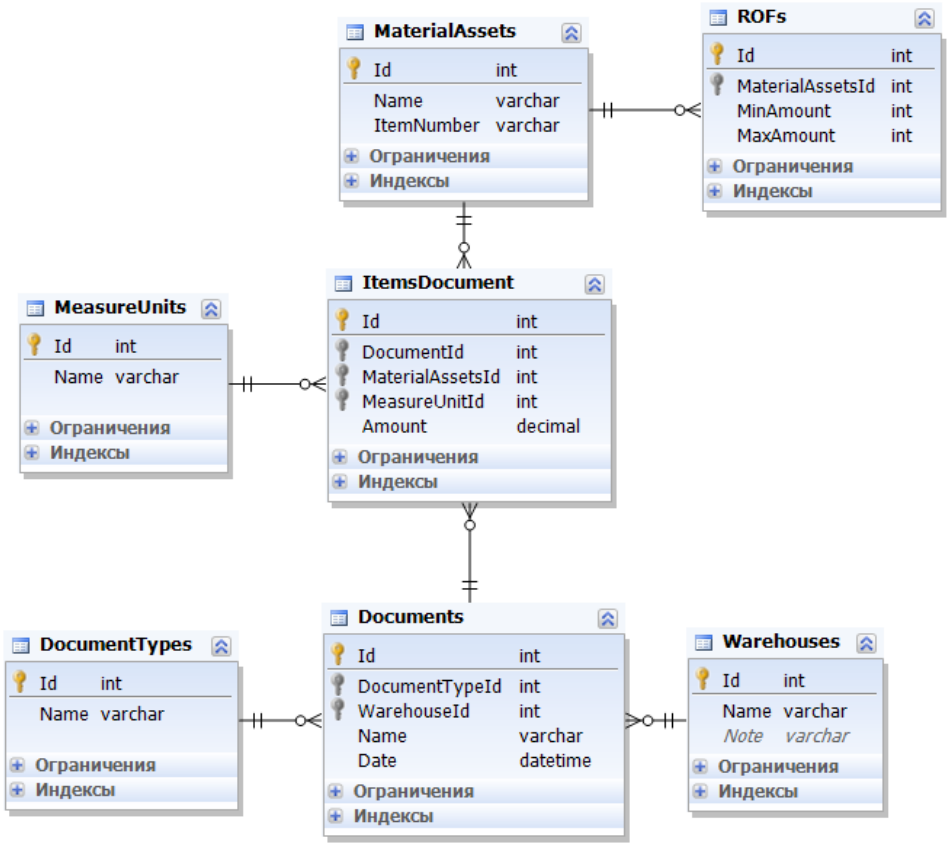
Сроки таблицы – они же **записи**, разделяют эти “явления”

Столбцы таблицы – они же **поля**, служат описанием параметров этого “явления”



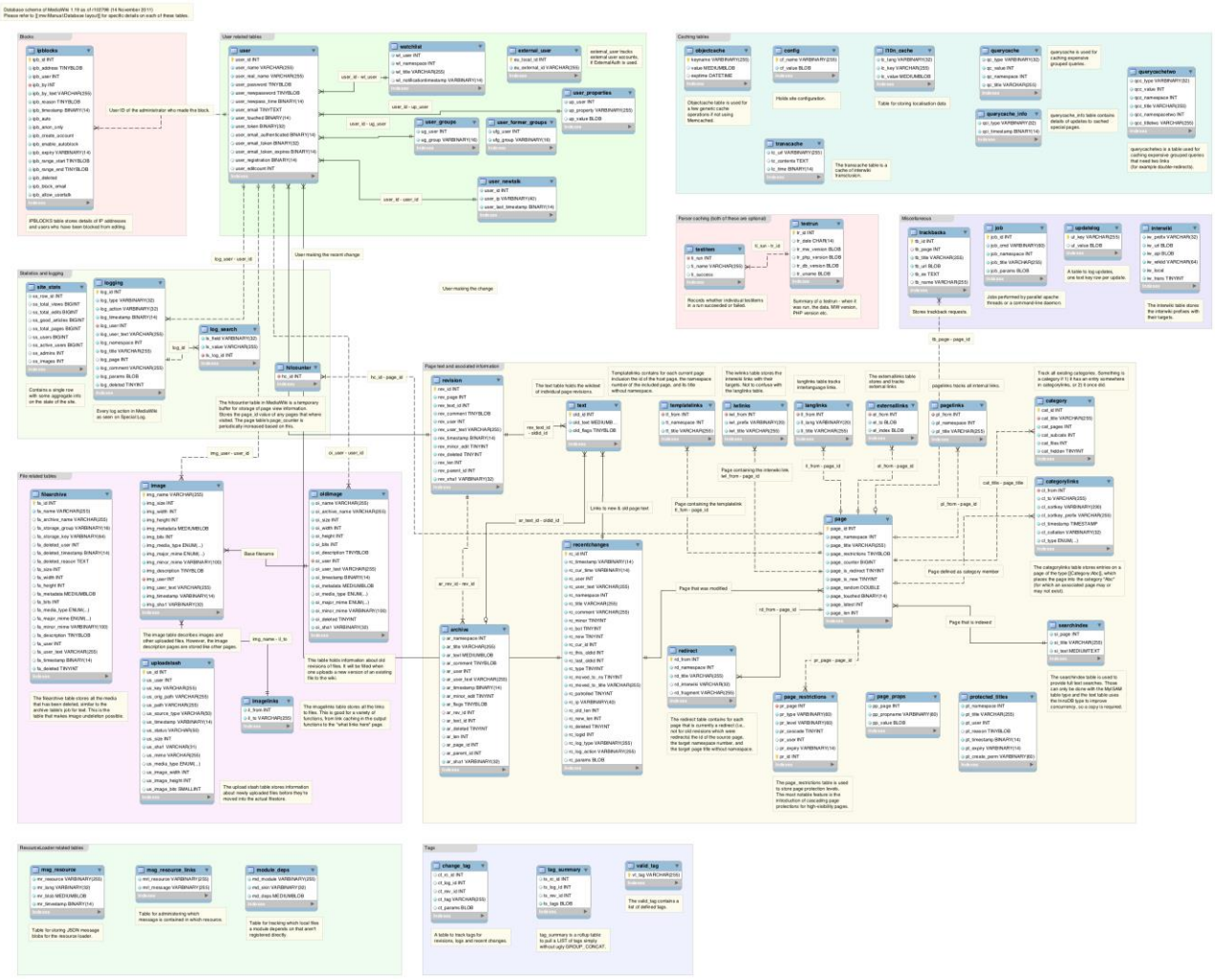
# Модель данных

**Модель данных -** модель «сущность-связь» (ER-модель), описывающая набор взаимосвязанных сущностей, которые отражают потребности бизнеса в аналитике и отчетности.



# Модель данных

А еще модель данных может выглядеть вот так

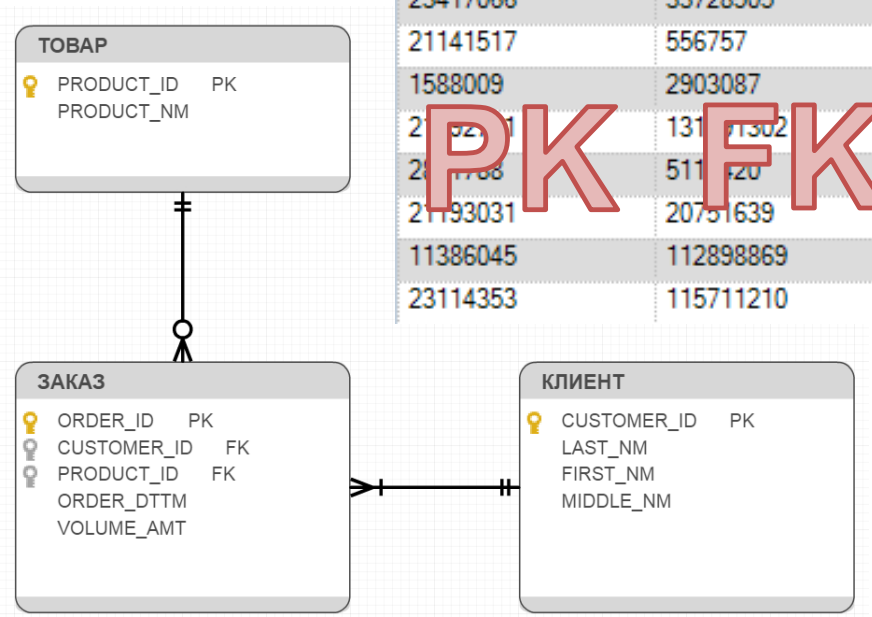


# Первичные и вторичные ключи

- Первичный ключ – поле или набор полей, идентифицирующих строку в таблице:
  - недопустимо отсутствие значения;
  - все значения уникальны.
- Внешний ключ – поле или набор полей, устанавливающих связь между данными в двух таблицах. В общем случае не имеет логических ограничений, как первичный ключ.

| order_id | customer_id | product_id | order_dttm | volume_amt |
|---|---|---|---|---|
| 21454279 | 130849138 | 78321383 | 2016-03-18 00:0... | 5 |
| 23417066 | 33728505 | 79587792 | 2016-04-15 00:0... | 52 |
| 21141517 | 556757 | 78901430 | 2016-03-30 00:0... | 219 |
| 1588009 | 2903087 | 3393773 | 2011-06-17 00:0... | 204 |
| 2[illegible] | 131[illegible] | 78[illegible] | 2016-03-23 00:0... | 16 |
| 2[illegible] | 511[illegible] | 57[illegible] | 2012-06-01 00:0... | 139 |
| 21193031 | 20751639 | 82231929 | 2016-05-30 00:0... | 234 |
| 11386045 | 112898869 | 58546918 | 2014-07-05 00:0... | 97 |
| 23114353 | 115711210 | 78155977 | 2016-03-19 00:0... | 335 |

PK FK FK



# Почти случайные логотипы


# Почти случайные логотипы






# Почти случайные логотипы







# Тинькофф Квест (легенда)

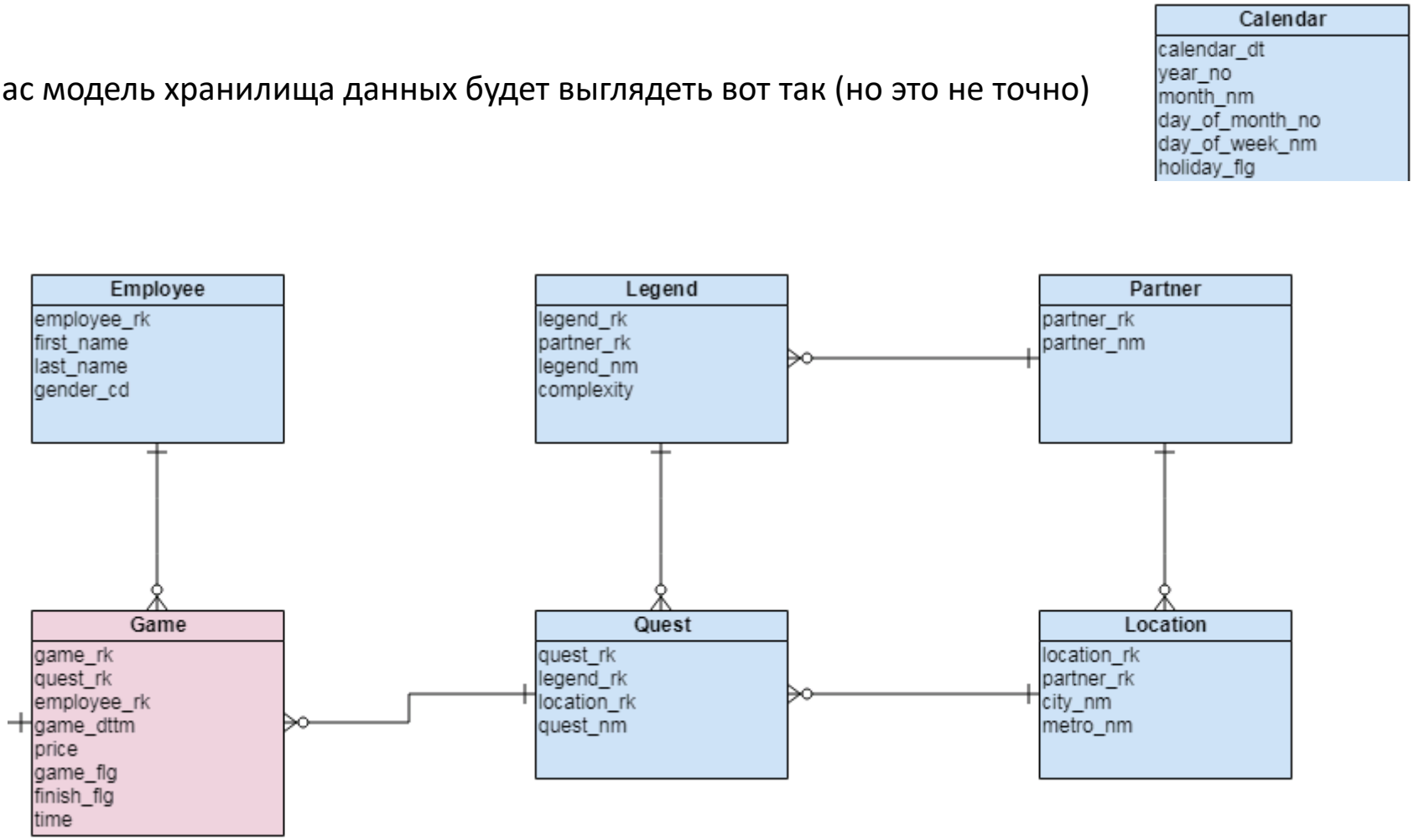
Давайте предположим, что мы запускаем проект “Тинькофф Квест”.

- Открываем квесты во всей стране по франшизной системе, то есть, каждой локацией по факту владеем не мы, а наши партнеры. Каждая локация принадлежит только одному партнеру.
- Локация – это по сути помещение, при этом на одной локации может проводиться несколько квестов, если наш партнер сможет их все уместить, естественно с соблюдением всех норм и правил безопасности.
- В рамках нашего проекта, партнеры в праве продавать легенды (сценарии) своих квестов другим участникам проекта. Мы даже наняли несколько креативных агентств, которые нам написали несколько базовых легенд для квестов.
- Игрой будем называть слот в расписании. Так вот, в рамках каждой отдельной игры, команде участников будет предложено пройти квест за 50 минут, кто-то справляется быстрее, а кто-то наоборот не успевает пройти. Бывает и такое, что игра может не состояться, если на нее никто не записался.
- В ходе прохождения игры, участникам будет помогать оператор.

P.S. Данная история целиком и полностью вымышленная, все совпадения с реальными личностями случайны ☺

# Тинькофф Квест (модель)

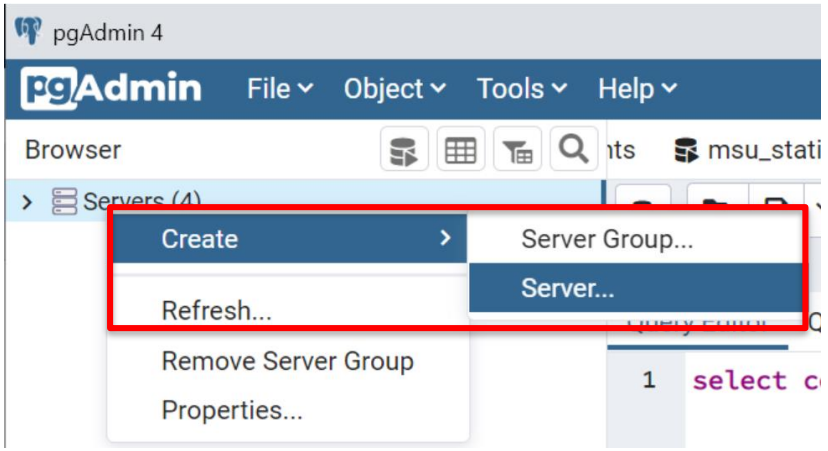
У нас модель хранилища данных будет выглядеть вот так (но это не точно)

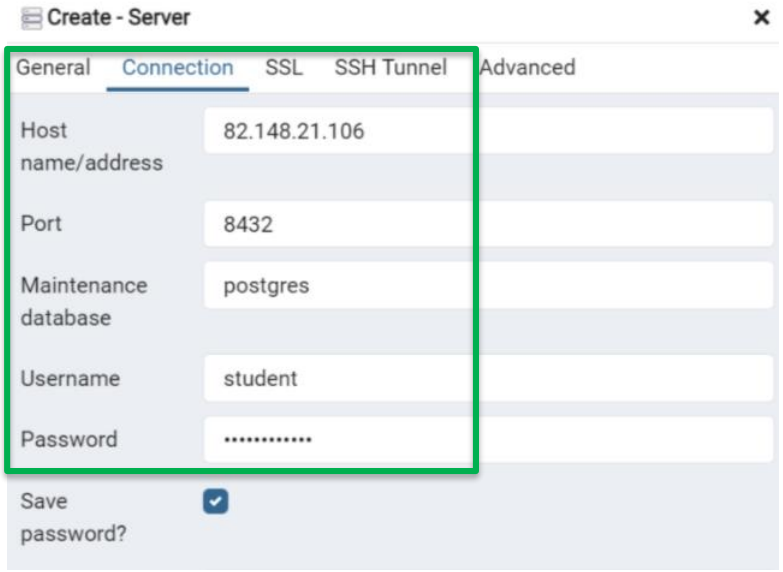


# Оглавление

# Подключаемся к базе

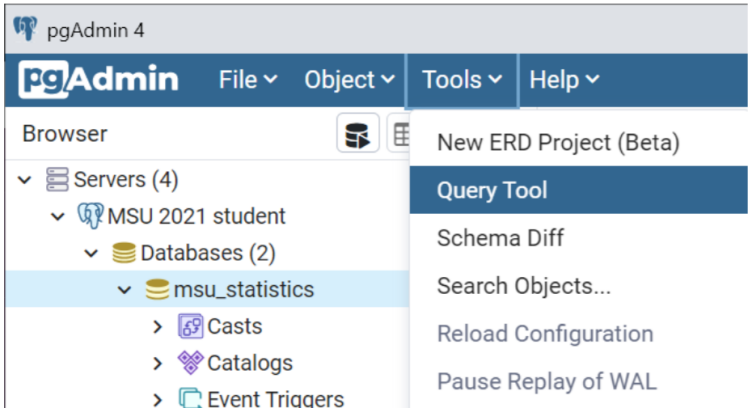
- Устанавливаем pgAdmin4 https://www.pgadmin.org/download/
- Создаем новое подключение к серверу
- Задаем любое имя на вкладке **General**
- Переходим на вкладку **Connection**
- Хост **82.148.21.106**
- Порт **8432**
- Database **postgres**
- Пользователь **student**
- Пароль **Bahz3loDieta**

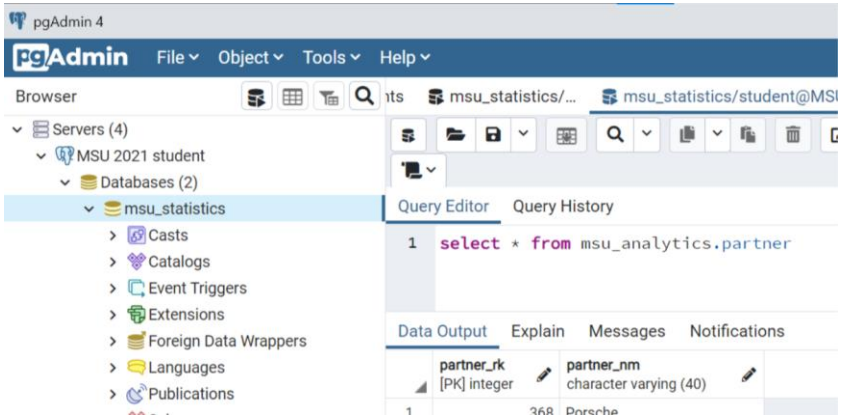


# Подключаемся к базе

После этого, у вас будет доступ к данным, которые будут лежать в схеме ***msu_analytics***.
Открываем окно для написания запросов и вперед.



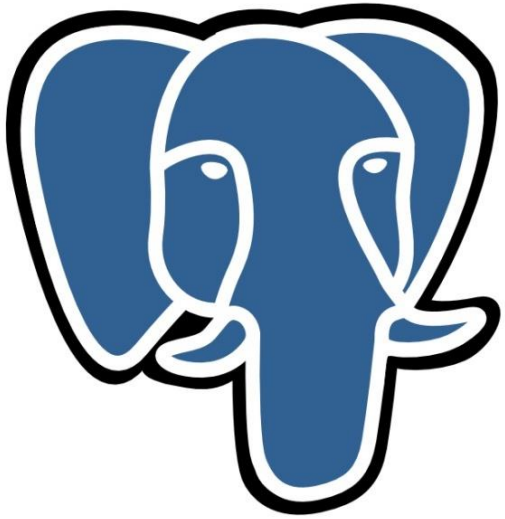
# SQL - запрос

Первый закон аналитики: “прежде чем начать анализировать данные, их нужно собрать”.

С этим легко можно справиться при помощи SQL запроса.

Структура SQL запроса проста:

- Select
- From
- Where
- Group by
- Having
- Order by
- Limit



PostgreSQL

# Операторы Select и From

Оператор Select отвечает за то, какие поля мы выводим в результат, в то время как оператор From – какие таблицы мы используем. Они должны быть в каждом запросе.

```
Select distinct
    quest_rk
From
    cource_analytics.game
```

```
Select
    *
From
    cource_analytics.employee
```

Если в операторе Select написать “*”, то будут выведены все поля, а если приписать distinct, то из всех одинаковых записей останется только одна.

```
Select
    count(*) as cnt
    ,avg(time) as avg_time
From
    cource_analytics.game
```

Если в операторе Select писать только агрегирующие функции, то он выдаст единственную строку с агрегатом по всей таблице. В данном случае число строк таблицы и среднее значение времени (по тем строкам, где оно заполнено)

Приписка as позволяет поля переименовывать.

# Оператор Where

Оператор Where позволяет оставить только нужные записи.

```
Select
        count(*) as cnt
From
        cource_analytics.employee
Where
        gender_cd = ‘f’
```

Теперь мы считаем количество не всех строк, а только тех, в которых *gender_cd = ‘f’*, что в нашем случае значит – посчитать количество девушек сотрудниц

В операторе Where можно использовать любые логические связки с использованием and, or, not

```
Select
        count(*) as cnt
From
        cource_analytics.employee
Where
        gender_cd = ‘m’
        and first_name = ‘Том’
```

# Операторы Group by и Having

Оператор Group by позволяет данные группировать

```
Select
        gender_cd
        ,count(*) as cnt
From
        cource_analytics.employee
Group by
        gender_cd
```

Теперь в выводе мы увидим не 1 запись как ранее, а 2. Одна будет говорить сколько у нас сотрудников мужчин, а вторая - сколько сотрудниц женщин.

Having – это фильтрация после группировки. В данном случае, мы оставим только те записи, в которых значение поля ***cnt*** будет больше 35.

```
Select
        gender_cd
        ,count(*) as cnt
From
        cource_analytics.employee
Group by
        gender_cd
Having
        count(*) > 35
```

# Оператор Order by

Оператор Order by отвечает за сортировку выводимого результата

```
Select
        gender_cd
        ,count(*) as cnt
From
        cource_analytics.employee
Group by
        1
Order by
        2 desc, 1
```

Для начала заметим, что в операторе Group by мы использовали число – это порядковый номер поля в операторе Select, то есть сгруппировали мы по полю *gender_cd*.

Сортировка же отработает по второму полю, то есть по полю *cnt*, приписка desc же означает, что сортировка будет произведена в обратном порядке (от большего к меньшему)

В операторах Group by и Order by можно использовать несколько полей, в этом случае их нужно перечислить через запятую.

# Join

Join позволяет нам связывать записи из разных таблиц между собой

```
Select
        a.employee_rk
        ,count(*) as cnt
From
        cource_analytics.employee a
        inner join cource_analytics.game b
                on a.employee_rk = b.employee_rk
Group by
        1
Order by
        2 desc
```

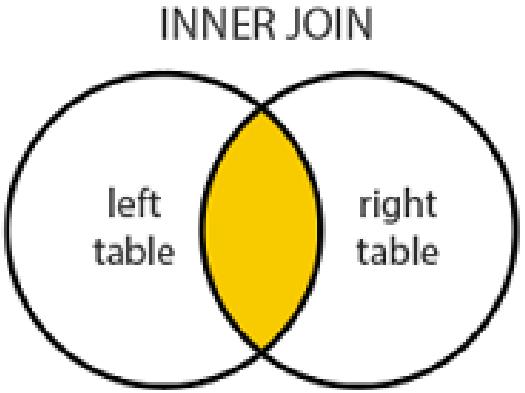
Inner join к каждой записи первой таблицы, которую мы обозначили за ***a***, подставляет записи из таблицы, которую мы обозначили за ***b*** подходящие под условия после on, при этом, на 1 запись первой таблицы может прийтись несколько записей из второй, в этом случае создастся запись на каждую комбинацию.

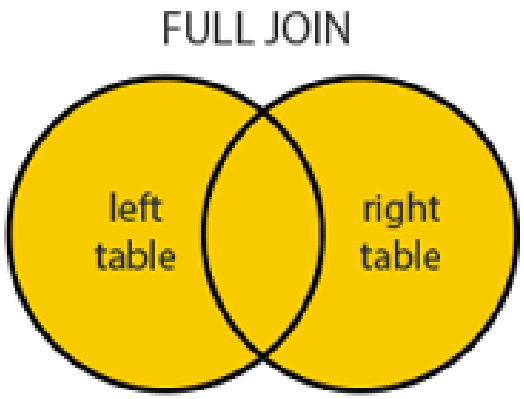
Если на запись из первой таблицы не нашлось записи из второй, то запись первой таблицы выфильтровывается.

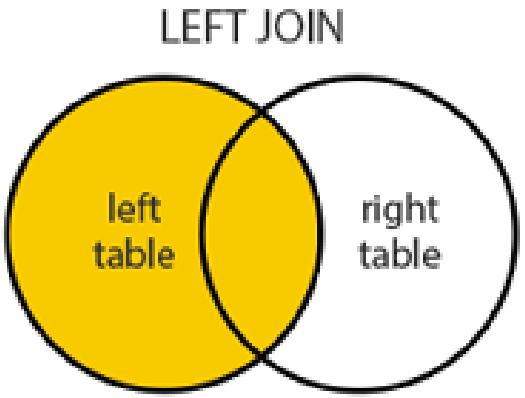
Если бы мы использовали left join, то такая запись первой таблицы дополнилась бы пустой строкой из второй таблицы.

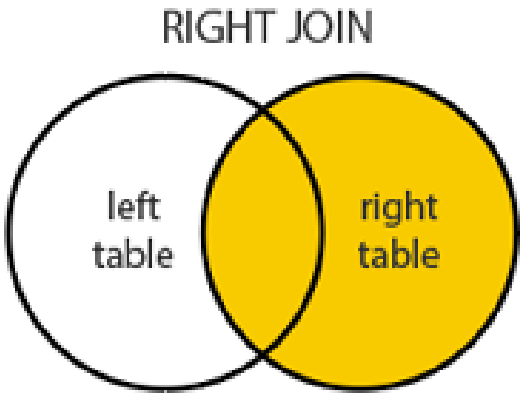
Как результат мы получили таблицу, в которой на каждого сотрудника указано на сколько игр он провел, а результат отсортирован от большего числа игр к меньшему.

# Join


INNER JOIN
left table
right table


FULL JOIN
left table
right table


LEFT JOIN
left table
right table


RIGHT JOIN
left table
right table

# Join

Можно использовать несколько Join подряд.

```
Select
        a.partner_rk
        ,a.partner_nm
        ,b.partner_nm as b_name
        ,c.partner_nm as c_name
From
        cource_analytics.partner a
        inner join cource_analytics.partner b
                on a.partner_rk = b.cpartner_rk
        left join cource_analytics.partner c
                on a.partner_rk = c.partner_rk
                and c.partner_rk % 2 = 1
```

# Подзапросы

Подзапросы позволяют нам “создавать” таблицы и сразу к ним обращаться.

```
Select
        date_trunc(‘month’, a.dt):: date as month
        ,avg(a.cnt) as avg
From
        (
        Select
                game_dttm::date as dt
                ,count(*) as cnt
        From
                cource_analytics.game
        Group by 1
        ) a
Group by 1
```

В такой реализации мы сначала для каждого дня посчитали количество игр в расписании этого дня.

А затем усреднили это число в рамках месяца.

Достаточно часто такая конструкция используется, если нам нужно проводить несколько агрегаций подряд.

# Промежуточные таблицы

Иногда удобнее создать таблицы иным способом

```
With test_table as
(
        Select *
        From cource_analytics.employee
        Where gender_cd = ‘f’
)

Select *
From test_table
```

Сейчас мы создали таблицу test_table и теперь можем к ней обращаться неограниченное количество раз.

# Промежуточные таблицы

Их тоже можно создавать несколько

```
With test_table as
(
        Select *
        From cource_analytics.employee
        Where gender_cd = ‘f’
),
test_table_2 as
(
        Select *
        From cource_analytics.employee
        Where gender_cd = ‘m’
)
Select *
From test_table
```

# Оконные функции

Подзапросы позволяют нам “создавать” таблицы и сразу к ним обращаться.

```
Select
        partner_rk
        ,location_rk
        ,row_number() over (partition by partner_rk order by location_rk) as num
From cource_analytics.location
```

Оконные функции не изменяют количество строк в select-е, но все равно может обогатить записи информацией об агрегатах.

Over – создает оконную функцию

В рамках этого окна, он группирует значения по partition by

И сортирует по order by

В нашем случае мы пронумеровали для каждого партнера все его локации.

Но так можно использовать и обычные агрегирующие функции вроде sum(). Или более хитрые.

# Пара полезных советов

```
Select *
From cource_analytics.game
Limit 10
```

Limit позволяет выводить только часть результата, пишется он в самом конце

Аккуратно сравнивайте поля в разных форматах.

Особенно это касается даты и даты - времени

Game_dttm <= ‘2018-02-01’ Эквивалентно Game_dttm <= ‘2018-02-01:00:00:00’

# Домашнее задание

1) Со сколькими креативными агентствами мы работаем? Креативное агентство – это партнер без единой локации, но имеющий патент на хотя бы одну легенду.

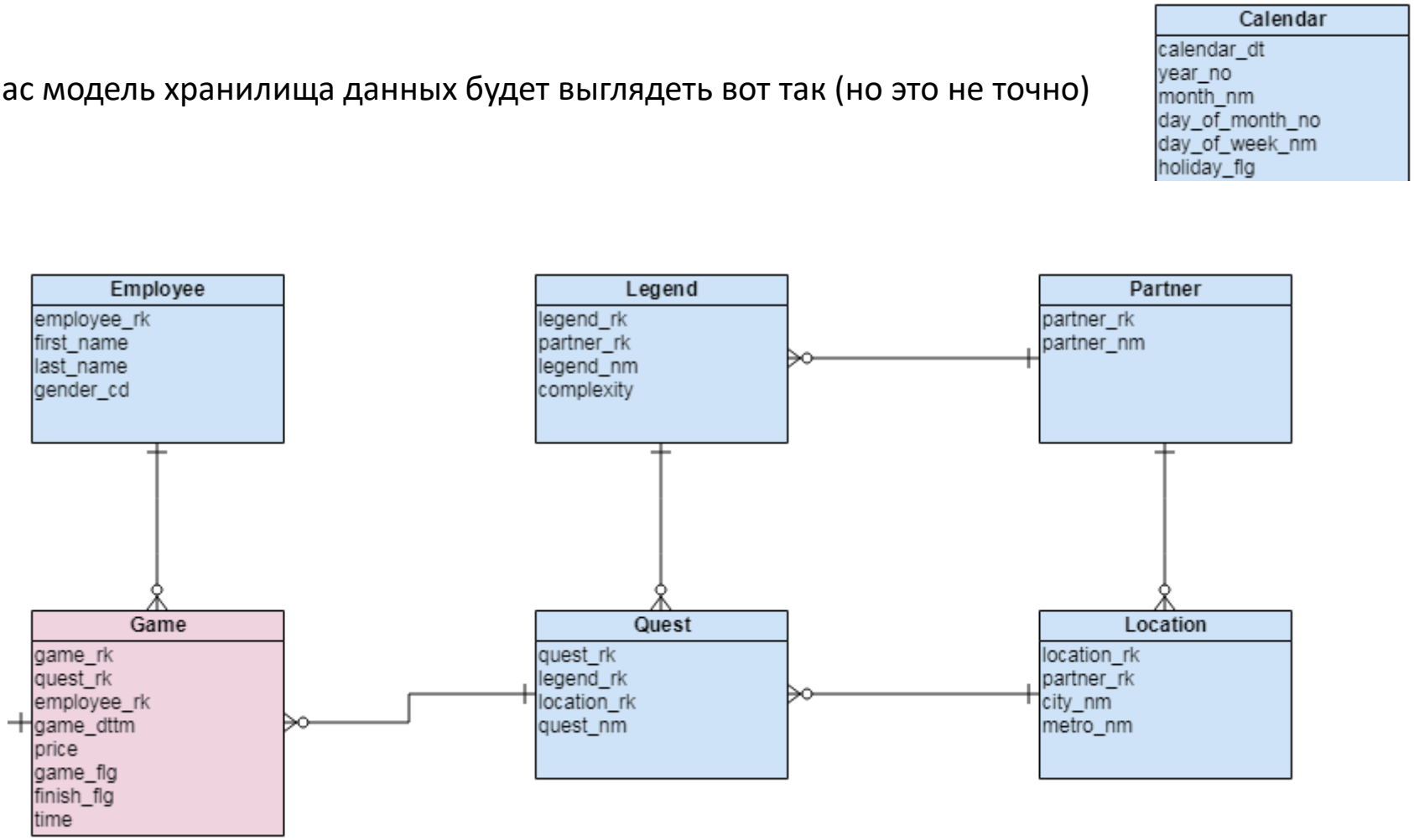
2) У какого квеста (выпишите его quest_nm) разница доли состоявшихся квестов в январе и в феврале наибольшая по модулю? Долей считать количество состоявшихся квестов деленное на количество заявленных. В случае наличия нескольких квестов, подходящих под условие, требуется вывести тот, у которого значение quest_rk больше.

3) В каждом городе провели награждение 3х сотрудников, в чью смену среди пройденных игр оказалось самое низкое среднее время прохождения. Выпишите имена и фамилии награжденных серебряными медалями девушек.

# Оглавление

# Тинькофф Квест (модель)

У нас модель хранилища данных будет выглядеть вот так (но это не точно)



# Partner

Витрина содержит информацию о наших бизнес партнерах

| Название поля | Описание |
|---|---|
| Partner_rk | Ключ партнера в хранилище данных |
| Partner_nm | Название партнера |

# Location

Витрина содержит информацию о тех локациях, на которых проходят квесты нашей франшизы

| Название поля | Описание |
|---|---|
| Location_rk | Ключ локации в хранилище данных |
| Partner_rk | Ключ партнера, которому принадлежит эта локация |
| City_nm | Название города, в котором расположена локация |
| Metro_nm | Название ближайшей станции метро к локации |

# Legend

Витрина содержит информацию о легендах (сценариях/сюжетах) конкретных квестов.

| Название поля | Описание |
|---|---|
| Legend_rk | Ключ легенды в хранилище данных |
| Partner_rk | Ключ партнера, которому принадлежит авторское право на эту легенду |
| Legend_nm | Запатентованное название сюжета |
| Complexity | Сложность квеста, идущего по данному сюжету |

# Quest

Витрина содержит информацию о квестах, в которые могут играть наши клиенты.

| Название поля | Описание |
|---|---|
| Quest_rk | Ключ квеста в хранилище данных |
| Legend_rk | Ключ легенды, в рамках которой играется квест |
| Location_rk | Ключ локации, на которой квест располагается |
| Quest_nm | Название квеста |

# Employee

Витрина содержит информацию о сотрудниках, которые проводят игры и помогают командам.

| Название поля | Описание |
|---|---|
| Employee_rk | Ключ сотрудника в хранилище данных |
| First_name | Имя сотрудника |
| Last_name | Фамилия сотрудника |
| Gender_cd | Пол сотрудника |

# Game

Витрина с расписанием запланированных и состоявшихся игр (отдельных прохождений и просто слотов в расписании по играм)

| Название поля | Описание |
|---|---|
| Game_rk | Ключ отдельной игры в хранилище данных |
| Quest_rk | Ключ квест, в рамках которого проходила игра |
| Employee_rk | Ключ сотрудника, который проводил игру |
| Game_dttm | Дата-время запланированного начала игры |
| Price | Стоимость игры |
| Game_flg | Флаг того, что игра состоялась |
| Finish_flg | Флаг того, что состоявшуюся игру удалось пройти |
| Time | Время прохождения игры |

# Calendar

Витрина с календарем

| Название поля | Описание |
|---|---|
| Calendar_dt | День календаря |
| Year_no | Год |
| Month_nm | Название месяца |
| Day_of_month_no | Номер дня внутри месяца |
| Day_of_week_nm | Название дня недели |
| Holiday_flg | Флаг выходного или праздничного дня |