Евгений Викторович

Установить:

- PostgreSQL
- PgAdmin

Лекция 1

Реляционная БД:

- Данные распределены по смыслу по таблицам
- Между таблицами есть отношения

Одна таблица - одна сущность предметной области

Столбцы - характеристики сущности (фиксированное кол-во)
Строки - непосредственно экземпляры (неограниченное кол-во)

Столбцы обязательно имеют **тип данных**:

- Числовые
- Символьные
- Для работы с датами
- Бинарные
- Логические
- Прочие
  Также можно создавать свои типы данных

`-- комментарий`

Конструкция SELECT

```
-- Основное
select список_столбцов
from имя_таблицы
-- Доп.
where условия_на_выборку
order by список_столбцов -- Сортировка
```

```
-- Example
select id_comp, name from company
```

Проекция отношений

Необязательно название, можно выражение (в пределах запроса):

```
select Price * ProductCount from Products;
```

Также можно изменить название выходного столбца или задать псевдоним с помощью `AS`

```
select ProductCount as Title, Price * ProductCount as totalsum
```

Условие WHERE

```
select * from Products where Price > 100
```

Функции `like` и `ilike`

- `like 'text'` - с учётом регистра
- `ilike 'text` - без учёта регистра
- `_` - один произвольный символ
- `%` - произвольный набор символов, в том числе пустой

Пустое значение `NULL`

> Значение в ячейке отсутствует

Для проверки на пустое значение используется `IS`:

```
is null
is not null -- не пустая
```

> Любые операции со значением `NULL` возвращают `NULL` !!!

`a = NULL` - всегда возвращает False

Сортировка ORDER BY

```
select * from company order by name
```

Для обратной сортировки `desc` (действует на поле)

```
select * from company order by name, age desc
-- name в правильном порядке, age в обратном
```

Ограничение вывода `limit`

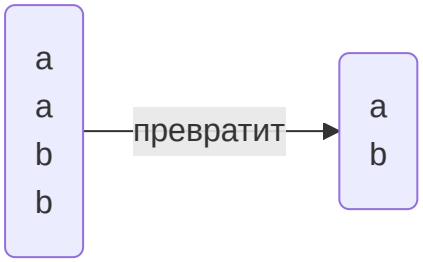
```
select * from company limit 3
-- будет выведено только 3 строки
```

Уникальные записи

Для выборки уникальных записей используется ключевое слово `distinct`

```
select distinct place from pass_in_trip order by 1 -- 1 - порядковый номер столбца в select (то есть place)
```

`order by 1` - плохая практика



Объединение запросов `union`

```
select name from company
union
select name from passenger
```

Просто `union` удаляет повторяющиеся записи (накладывает `distinct`)
Для вывода всех записей `union all`

```
select name from company
union all
select name from passenger
```

Оба запроса должны выдавать результаты в одинаковой формате (количество и типы полей)

Практика 2

PgAdmin обращается к серверу только во время запросов

Лекция 2

Создание/удаление объектов

- DDL - язык определения данных
- DML - язык обработки данных

Создание БД:

```
create database <name>
```

Удаление БД (необратимый процесс):

```
drop database <name>
```

Эти конструкции выполняются на уровне кластера, все остальные(ниже) на уровне БД

Схема - некоторая возможность логически разделить объекты в БД, одноуровневые(схема в схеме невозможна). Схема по умолчанию: `public`

```
create schema <name>
drop schema <name>
```

Создание/удаление таблиц

Для создания:

```
create table <name_table>
(name_columnn_1 type_data attributes_column_1,
 имя_столбца_2 тип_данных аттрибуты_столбца_2,
 );
```

Пример:

```
create table customers
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(30),
    LastNAme CHARACTER VARYING(30),
    Age INTEGER
);
```

Удаление:

```
drop table <name>
```

PRIMARY KEY

`PRIMARY KEY` - **первичный ключ**, уникально идентифицируют строку в таблице
Варианты создания первичного ключа

```
create table Cusomers
(
    Id SERIAL PRIMARY KEY,
    ...
)
```

```
create table Cusomers
(
    Id SERIAL,
    ...,
    PRIMARY KEY(Id)
)
```

В качестве первичного ключа может быть любой тип данных, главное чтобы было уникально.

`SERIAL` - создаёт поле int и генерирует последовательные номера (создаёт объект последовательности)1

Составной первичный ключ

```
create table Cusomers
(
    Id SERIAL,
    Num INTEGER
    ...,
    PRIMARY KEY (Id, Num)
)
```

`UNIQUE` - также уникальные значения, но не возможно связывать с другими таблицами

NOT NULL

Чтобы указать, что обязательно нужны некие значения: `NOT NULL`

```
create table Customers
(
    ...,
    Name CHARACTER VARYING(52) NOT NULL
```

```
    ...
)
```

По умолчанию NULL возможен, если не указать обратного

DEFAULT

`DEFAULT` - значение по умолчанию, когда при добавлении не определяем значение этого атрибута

```
create table customers
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(30),
    LastNAme CHARACTER VARYING(30),
    Age INTEGER DEFAULT 18
);
```

CHECK

`CHECK` - задаёт условие для проверки значений

```
Age INTEGER CHECK(Age > 0 AND Age < 100)
Email CHARACTER VARYING(30) UNIQUE CHECK(Email != '')
```

CHECK можно наложить на всю таблицу

CONSTRAINT

`CONSTRAINT` - задаёт имя для ограничений

Изменение таблиц

```
ALTER TABLE <name>
ADD name_column type_date [attr] |
DROP COLUMN name_column |
ALTER COLUMN name_column параметры |
ADD [CONSTRAINT] определение ограничения |
DROP [CONSTRAINT] имя_ограничения
;
```

С помощью `TYPE` можно поменять тип колонки (но не всегда работает)
`SET` добавляет ограничение
`RENAME` - переименовать столбец

Редактирование данных

Добавление записей в таблицу

```
INSERT INTO имя таблицы (столбец1, столбец 2, столбец 3) VALUES (значение 1, значение 2, значение 3)
```

Можно не указывать перечень столбцов, но тогда нужно указать значения для всех столбцов в определённом порядке

Также можно добавить несколько строк указывая после VALUES через запятую в скобках сами строки

```
UPDATE имя_таблицы SET столбец1 = значение1, столбец 2 = значение 2 WHERE условие(для каких записей обновить)
```

```
DELETE FROM имя_таблицы WHERE условие (что конкретно удалить)
```

Удаляется строка!

Практика 2

> Сортировка у `union` всегда одна и объявляется в конце

```
select 3+4
```

Приведение типов:

```
select 3::real
```

Функция random():

```
select random();
select random() * 10;
```

`round()` - округляет число
`trunc()` - количество знаков после запятой

Date

```
select '2025-02-18'::date
```

now()

`now()` - вернёт текущую дату и время в формате `timestamp with time zone`
В пределах запроса время не изменяется

age()

`age()` - перегруженная функция
1. Передаём дату, считает возраст до текущего дня

```
select age('2006-11-30'::date)`
```

```
18 years 2 mons 18 days
```

extract()

`extract()` - извлечь

Пример: количество полных лет

```
select extract(years from age('2006-11-30'::date))
```

Пример:

```
select time_out from trip
where time_out::time <= '12:00:00'::time
```

```
select * from pass_in_trip where extract(month from date) = 4 and
extract(year from date) = 2003;
select * from pass_in_trip where date>='2003-04-01'::date and date<='2003-
04-30'::date;
select * from pass_in_trip where date::text like '2003-04-%'
```

Практика 3

substring(from, start, count)

Срез по строке

> Индексы начинаются с 1

lenght()

Возвращает длину строки

```sql
-- task1
select * from pass_in_trip where extract(year from date) = 2003;
-- where date::text like '2003-%'

-- task2
select distinct id_psg from pass_in_trip
-- where place like '1%' or place like '2%' or place like '3%';
where substring(place, 1, length(place) - 1)::int in (1, 2, 3);

-- task3
select name from company where country like '%an%' order by name --desc;

-- task4
select trip_no from pass_in_trip where extract(day from date) % 2 = 0;

-- task5
select *, extract(year from age(date)) from pass_in_trip;

-- task6
select * from pass_in_trip where extract(year from age(date)) > 20;

-- task7
select * from trip where time_out::time <= '12:00:00'::time;

-- task8
select * from trip where extract(hour from time_out) % 2 = 1;

-- task9
select * from trip where time_out > time_in;

-- task10
select * from trip where days like '%2%4%';

-- task11
select town_to from trip where days like '%3%5%'
```

count()

Считает количество...

split_part(string, delimiter(символ разделения), num)

Делит строку по символу и возвращает num часть

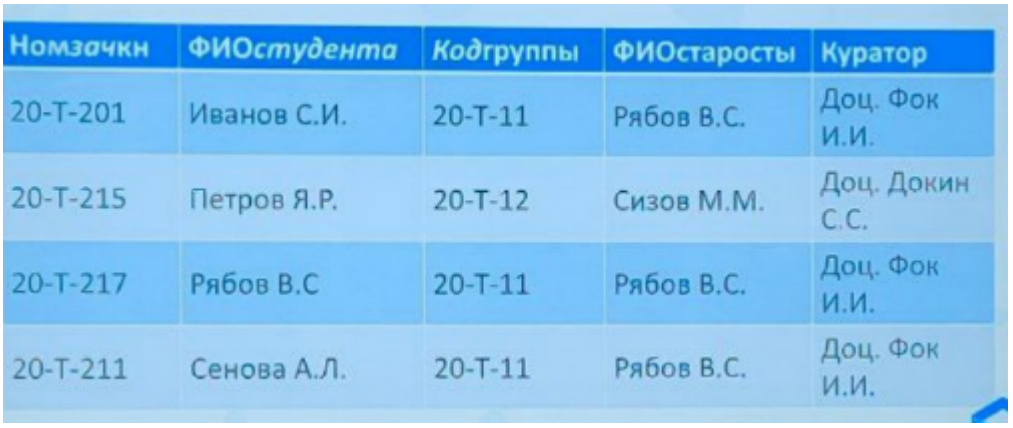
Лекция 3

Аномалии, нормальные формы

> Придумать тему для индивидуального проекта. Индивидуальный проект - маленькая БД, где изначально есть четыре сущности. В известной области.

Атрибут - свойство некоторой сущности
Домен атрибута - множество допустимых значений
Кортеж - конечное множество взаимосвязанных допустимых значений атрибутов
Отношение - конечное множество кортежей (таблица)
Схема отношений - конечное множество атрибутов, определяющее некоторую сущность
Проекция - отношение, полученное путём удаления и(или) перестановки некоторых атрибутов

Аномалия - ситуация в БД, которая приводит к противоречию в БД либо существенно усложняет обработку БД. Причиной является излишнее дублирование данных в таблице.
Аномалии:



Номзачкн	ФИОстудента	Кодгруппы	ФИОстаросты	Куратор
20-Т-201	Иванов С.И.	20-Т-11	Рябов В.С.	Доц. Фок И.И.
20-Т-215	Петров Я.Р.	20-Т-12	Сизов М.М.	Доц. Докин С.С.
20-Т-217	Рябов В.С	20-Т-11	Рябов В.С.	Доц. Фок И.И.
20-Т-211	Сенова А.Л.	20-Т-11	Рябов В.С.	Доц. Фок И.И.

Пример

- Добавления

  > Не можем добавить новую группу, так как добавление группы связано с добавлением студента.

- Редактирования
- Удаления

Нормализация данных

Нормализация - способ организации данных. В нормализованной базе нет повторяющихся данных, с ней проще работать и можно менять её структуру для разных задач. В процессе нормализации данные преобразуют, чтобы они занимали меньше места, а поиск по элементам был быстрым и результативным.
В процессе нормализации создают дополнительные атрибуты или таблицы.

Всего 7 (чаще всего используются первые три)

Первая нормальная форма

Требование первой нормальной формы:

1. Значения атрибута должны быть простыми (атомарными - типо атом как неделимая частица)

Неправильно:

Фирма	Модели
BMW	M5, X5M, M1
Nissan	GT-R

Правильно:

Фирма	Модели
BMW	M5
BMW	X5M
BMW	M1
Nissan	GT-R

2. Каждая запись должна иметь уникальную идентификацию (*Сложные ключи*(состоящие из нескольких атрибутов) и *простые ключи*)

> **Простые ключи** - новые данные, но проще работать
> **Сложные ключи** - когда используется связь, то весь ключ переносится в в дочернюю таблицу (и будет так расти)

Вторая нормальная форма

Включает в себя первую нормальную форму, имеет составной ключ и все остальные неключевые атрибуты зависят от *всего ключа*

Модель	Фирма	Цена	Скидка
M5	BMW	5500000	5%
X5M	BMW	6000000	5%
M1	BMW	2500000	5%
GT-R	Nissan	5000000	10%

Цена зависит от `модели` и `фирмы`, но `скидка` зависит только от `фирмы`. Это не удовлетворяет второй нормальной форме.

Эта ситуация решается путём декомпозиции, делением на два или больше отношений.

Модель	Фирма	Цена
M5	BMW	5500000
X5M	BMW	6000000
M1	BMW	2500000
GT-R	Nissan	5000000
Фирма	Скидка	
BMW	5%	
Nissan	10%	

Вторая нормальная форма рассматривается только для сложных ключей.

Третья нормальная группа

Находится во второй нормальной форме и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Чтобы нормализовать БД до 3НФ, необходимо сделать так, чтобы в таблице отсутствовали неключевые столбцы, которые зависят от других неключевых столбцов.
Неправильно:

Модель	Магазин	Телефон
BMW	Риал-авто	87-33-98
Audi	Риал-авто	87-33-98
Nissan	Некст-Авто	94-54-12

Снова делить.

Магазин	Телефон
Риал-авто	87-33-98
Некст-Авто	94-54-12
Модель	Магазин
BMW	Риал-авто
Audi	Риал-авто
Nissan	Некст-Авто

Практика 4

```
insert into students values (1, 'Ivan', 'Ivanov')
```

В том порядке значение в каком они идут в таблице, и ровно такое же количество

ВАЖНО!!!
Перед выполнение команд, таких как `UPDATE` или `DELETE`, всегда писать и проверять условие `WHERE` с помощью `SELECT` и уже потом выполняем `UPDATE` или `DELETE`

Практика 5

between

> Проверяет значение в диапазоне, включает левое и правое значение

```
select * from trip
where time_out::time between '10:00'::time and '16:00'::time
```

case

```
select
    case
        when time_in > time_out then time_in - time_out
        else time_in - time_out + '24:00'::interval
    end,
    time_in - time_out
from trip
```

▪

Serial - не тип столбца, а инструкция, которая сначала создаёт `последовательность`, а столбец имеет тип `integer` и имеет значение по умолчанию, которое берётся из ранее созданной последовательности.

> Если есть таблица с `Serial`, то вручную значения этого столбца не выставляем (за малым исключением, но тогда нужно изменить `Serial)

Для изменения `Serial`:

```
alter sequence student_id_seq restart with 100;
```

Практика 6

Добавление первичного ключа: (ограничение на всю таблицу)

```
alter table student add primary key(id);
```

Ограничение на конкретный столбец:

```
alter table student
alter COLUMN fname set not null,
alter COLUMN lname set not null;
```

`IF EXISTS` - если существует. Если не существует, то предупреждение, но не ошибка

```
drop table if exists student
```

> `default` срабатывает только, когда поле не заполняем. Если мы явно указываем `null`, то `default` не сработает.

Лекция 4

Агрегирующие функции

> Агрегирующие функции выполняют операции **над набором строк**. Результатом такой функции является **одно** значение, а набор строк, для которых она выполнялась, сворачиваются в **одну строку.**

count(*)

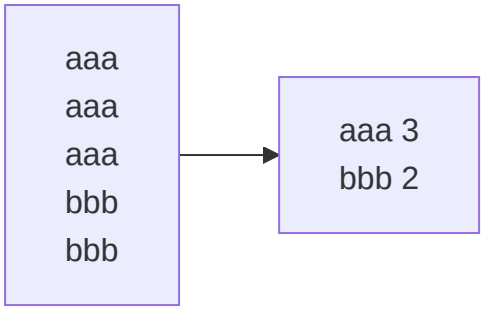
> Количество записей

```
select count(*) from company
```

group by

> Если в запросе присутствует агрегирующая функция и неагргирующие атрибуты, то все эти атрибуты обязательно перечисляются в `group by`.

```
select name, count(*)
from passenger
group by name
```



`group by` - Группировка по уникальным атрибутам

having()

> Ограничивает набор записей выдающих в результат (тот же where, только работает после группировки)

```
select name, count(*)
from company
group by name
having count(*) > 1
```

```
select S_group, avg(rating)
from company
1) where S_group like 'K0709-%'
group by S_group
2) having S_group like 'K0709-%'
```

Результат одинаковый, но для сервера выполнение по разному. `where` *будет быстрее.*
`having` - используется для сортировки результата агрегирующий функций. На этапе `where` нет ещё результата агрегирующих функций.

Связи

Связь - ассоциация(отношение) между сущностями разных типов.
Выделяют **главную**(родительская) и **зависимую**(дочерняя). Родительская спокойно может обойтись без дочерней.

> Например: студент и группа. Студент без группы не может быть. Группа без студента может существовать.

Связь один к одному (1-1)

Объекту одной сущности можно сопоставить только один объект другой сущности

> Пример: человек и СНИЛС

Связь один ко многим (1-*)

Несколько строк из дочерней таблицы зависят от одной строки в родительской

> Пример: У одного человека может быть несколько паспортов. Но один паспорт имеет владельца только в виде одного человека.

Связь многие ко многим (*-*)

Одна строка из таблицы А может быть связана с множеством строк из таблицы Б. В свою очередь строка из таблицы Б может быть связана со множеством строк таблицы А.

> Пример: статьи и тэги. Статья имеет множество тегов. Тэги имеют множества статьей.

Создание связей

Используются внешние ключи.
Внешний ключ - один или несколько атрибутов, которые привязывают сущности одной таблицы к сущности другой.
Для создания внешних ключей используется `references`.

При создании таблицы

...

Изменение существующей таблицы

```
alter table public."ArticleTags"
    add foreign key ('ArticleId') references public."Articles"
("ArticleId")
```

- В базе данных есть только тип один ко многим.
- Один к одному реализуется с помощью признака уникальности `UNIQUE`
- Для многие ко многим создаётся специальная таблица отношения - таблица связки.

Параметры связей (атрибуты)

`Foreign key` может иметь атрибуты, определяющие действия БД при удалении (`ON DELETE`) или редактировании (`ON UPDATE`) записи в родительской таблице.

- `SET NULL` - при удалении записи из родительской таблицы, то в дочерней пропишится `NULL`
- `SET DEFAULT` - ставится значение по умолчанию
- `NO ACTION` или `RESTRICT` - не позволяет удалить запись в родительской таблице.

- `CASCADE` - Если удаляется в родительской, то всё связанное удаляется и в дочерней

```
alter table public."ArticleTags"
    add foreign key ('ArticleId')
    references public."Articles" ("ArticleId")
        on update no action
        on delete no action;
```

`NO ACTION` - позволяет отсрочить проверку. `RESTRICT` - ставит жёсткий запрет.

Практика 7

`trim()` - убирает пробелы
Третий вариант `insert into`:

```
insert into country (name)
select distinct country from company
```

> Вставляет результат `select` в таблицу `country`

`||` - конкатенация строк
`''` - экранирование `'`

```
-- Изменения passenger
select * from passenger;
alter table passenger add column lname text;
update passenger set lname = split_part(name, ' ', -1);
select *, trim(split_part(name, lname, 1)) from passenger;
update passenger set name = trim(split_part(name, lname, 1));
alter table passenger alter column lname set not null;

alter table passenger
add constraint passenger_sex_check
    check (sex in ('m', 'f'));

-- Изменение company
select * from company;

create table country (
    id_country Serial PRIMARY KEY,
    name text not null
);

select * from country;

select distinct replace(country, '
```

```
', '') from company;

update company set country = 'Russia' where country like 'Russia%';
select distinct country from company;

insert into country (name)
select distinct country from company;

alter table company add id_country integer;

select *, 'update company set id_country='||
    id_country::text||' where country = ''' || name || ''';' from country;

update company set id_country=1 where country = 'Russia';
update company set id_country=2 where country = 'England';
update company set id_country=3 where country = 'France';

alter table company drop column country;

select * from company;
```

Практика 8

Увеличение количества ключей в первичном ключе - уменьшает производительность базы.
`string_to_table(days, ',')` - табличная функция, то есть возвращает таблицу. Разделяет значение по `,` и выносит каждое значение на отдельную строчку.

Самолёты в отдельных отношениях, planes, код в trip, тоже самое в town_to и town_in через union. planes, towns

```
-- pass_in_trip
alter table pass_in_trip alter column date type date;
select * from pass_in_trip;
-- trip
select * from trip;
alter table trip alter column time_out type time;
alter table trip alter column time_in type time;

create table trip_days (
    trip_no int not null,
    day int not null check (day between 1 and 7),
    primary key (trip_no, day)
)
select * from trip_days;

select trip_no, string_to_table
from trip, string_to_table(days, ',');
```

```sql
insert into trip_days
select trip_no, string_to_table::int
from trip, string_to_table(days, ',');

select * from trip_days;
alter table trip drop column days;

select * from trip;
-- homework
create table planes (
    id_plane Serial primary key,
    name text not null
);

insert into planes (name)
select distinct plane from trip;

select * from planes;

alter table trip add id_plane int;
select * from trip;

select *, 'update trip set id_plane = '|| id_plane::text ||' where plane =
'''|| name ||''';' from planes;

update trip set id_plane = 1 where plane = 'Boeing';
update trip set id_plane = 2 where plane = 'IL-86';
update trip set id_plane = 3 where plane = 'TU-134';
update trip set id_plane = 4 where plane = 'TU-154';

select * from trip;
select * from planes;

alter table trip drop plane;

create table towns (
    id_town Serial primary key,
    name text not null
)

alter table trip alter id_plane set not null;

select * from towns;

insert into towns (name)
select town_from from trip
union
select town_to from trip;
```

```
select *, 'update trip set town_from = ' || id_town::text ||' where
town_from ='''|| name ||''';' from towns;
update trip set town_from = 1 where town_from ='London';
update trip set town_from = 2 where town_from ='Paris';
update trip set town_from = 3 where town_from ='Singapore';
update trip set town_from = 4 where town_from ='Moscow';
update trip set town_from = 5 where town_from ='Vladivostok';
update trip set town_from = 6 where town_from ='Rostov';

select * from trip;

select *, 'update trip set town_to = ' || id_town::text ||' where town_to
='''|| name ||''';' from towns;
update trip set town_to = 1 where town_to ='London';
update trip set town_to = 2 where town_to ='Paris';
update trip set town_to = 3 where town_to ='Singapore';
update trip set town_to = 4 where town_to ='Moscow';
update trip set town_to = 5 where town_to ='Vladivostok';
update trip set town_to = 6 where town_to ='Rostov';

select * from trip;
```

Практика 9

Добавление связи происходит в дочернюю таблицу

```
alter table student
    add foreign key ("id_grp") references s_group (id);
```

Проверяет все ли значения соответствуют тем, что есть в s_group

> Чтобы изменить `foreign key` , нужно удалить и потом создать новый

`set null` и `set default` почти никогда не используется
Чаще всего `no action` и `cascade` (удобно, но опасно)

```
create table s_group (
    id serial primary key,
    name text not null
);
create table student (
    id serial primary key,
    fio text not null,
    id_grp integer
);
insert into s_group (name) values ('K0709-23-1'), ('K0709-23-2'), ('K0709-
23-3');
select * from s_group;
```

```sql
insert into student (fio, id_grp) values
    ('student 001', 2),
    ('student 011', 1),
    ('student 021', 3),
    ('student 052', 3);

alter table student
    add foreign key (id_grp) references s_group (id);

delete from s_group where id = 3; --ошибка, нарушение целостности

-- Вариант 1.1
alter table student
    drop constraint student_id_grp_fkey,
    add foreign key (id_grp) references s_group (id)
    on delete set null;
-- Вариант 1.2
alter table student
    drop constraint student_id_grp_fkey,
    add foreign key (id_grp) references s_group (id)
    on delete set default;
-- Вариант 2.1 (по умолчанию)
alter table student
    drop constraint student_id_grp_fkey,
    add foreign key (id_grp) references s_group (id)
    on delete no action;
-- Вариант 2.2
alter table student
    drop constraint student_id_grp_fkey,
    add foreign key (id_grp) references s_group (id)
    on delete restrict;
-- Вариант 3
alter table student
    drop constraint student_id_grp_fkey,
    add foreign key (id_grp) references s_group (id)
    on delete cascade;

delete from s_group where id = 2;

select * from student;
select * from s_group;
```

Лекция 5

Реляционная алгебра, сложные запросы

Реляционная алгебра базируется на теории множеств и является основой логики работы баз данных.

Результатом любой операции реляционной алгебры является новое отношение

Проекция

> Операция, при которой из отношения выделяется часть атрибутов

Выбор

> Из отношения выбираем только определённые записи

Объединение

> Состыковываем отношения

Пересечение

> Отношение, в которое входят кортежи в состав первого отношения и добавляются кортежи из второго.

Вычитание

> Берём отношение 1, кроме кортежей, которые существуют в отношении 2

Умножение (Декартово произведение)

> Список атрибутов - сумма атрибутов из первого и второго отношения. Кортежи результата будут составляться как произведение каждого кортежа первого отношения на все кортежи второго отношения

```
select * from passenger, pass_in_trip
```

Когда в `from` находится несколько таблиц

Соединение

> Произведение с условием на соединение записей

```
select * from passenger, pass_in_trip
where passenger.id_psg = pass_in_trip.id_psg
```

Варианты соединения:

Представленные таблицы:

id	name	d.id
1	Alice	10

id	name	d.id
2	Bob	20
3	Charlie	30
4	David	NULL

id	name
10	HR
20	IT
40	Finance

INNER JOIN (или просто JOIN)

Внутреннее соединение, в результате только те кортежи, для которых нашлось соответствие, то есть

> Только те записи, которые существуют в обоих таблицах

```
select e.id, e.name, e.department_id, d.department_name
from employees e -- e - это alias
inner join departments d on e.department.id = d.department.id;
```

id	name	d.id	d.name
1	Alice	10	HR
2	Bob	20	IT

LEFT JOIN (LEFT OUTER JOIN)

> Все кортежи из первой таблицы + совпадающие записи из второй таблицы

```
select e.id, e.name, e.department_id, d.department_name
from employees e -- e - это alias
left join departments d on e.department.id = d.department.id;
```

id	name	d.id	d.name
1	Alice	10	HR
2	Bob	20	IT
3	Charlie	30	NULL
4	David	NULL	NULL

RIGHT JOIN (RIGHT OUTER JOIN)

> Все кортежи из второй таблицы + совпадающие записи из первой таблицы

id	name	d.id	d.name
1	Alice	10	HR
2	Bob	20	IT
NULL	NULL	NULL	Finance

FULL JOIN

> Все кортежи из первой таблицы и из второй. Записи состыкованные и оствшиеся

```
select e.id, e.name, e.department_id, d.department_name
from employees e -- e - это alias
full join departments d on e.department.id = d.department.id;
```

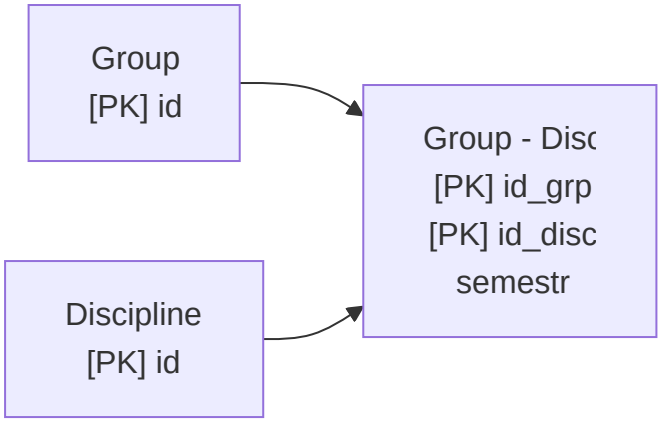
id	name	d.id	d.name
1	Alice	10	HR
2	Bob	20	IT
3	Charlie	30	NULL
4	David	NULL	NULL
NULL	NULL	NULL	Finance

Общие рекомендации по построению запросов

1. Выбираем таблицу, являющейся основной в контексте запроса. Например, если нужно выбрать сведения о полетах пассажира – этой таблицей будет `pass_in_trip`, начинаем с нее формировать предложение `from`
2. Добавляем к ней с использованием `left join` таблицы, уточняющие сведения в запросе в порядке их следования в иерархии структуры БД
3. Формируем предложение `where`
4. Формируем текст предложения `select`
5. Формируем предложение `order by`

Практика 10

Для создания связи `многие ко многим` используется дополнительная таблица, в которой первичный ключ состоит из двух полей




```
create table s_group (
    id Serial primary key,
    name text not null
);
create table discipline (
    id serial primary key,
    name text not null
);

select * from s_group;

select * from discipline;

insert into s_group (name) values ('K0709-23-1'), ('K0709-23-2'), ('K0709-
23-3');
insert into discipline (name) values ('Databases'), ('Literature');

create table group_discipline (
    id_group int not null,
    id_discipline int not null,
    primary key(id_group, id_discipline)
);

alter table group_discipline
    add foreign key (id_group) references s_group (id),
    add foreign key (id_discipline) references discipline (id);


insert into group_discipline (id_group, id_discipline) values
    (1, 1), (1, 2),
    (2, 1), (2, 2),
    (3, 2);

select * from group_discipline;

select * from group_discipline where id_group = 1; -- Какие дисциплины у
группы 1
```

```sql
select * from group_discipline where id_discipline = 1; -- У каких групп дисциплина 1
```

Практика 10-11

Задание из мудла:

СТО

- Сделать первичный ключ
- Вынести режим работы в другую таблицу:
- Разбить адрес (город, улица, дом), город - отдельная сущность
- Добавить номер телефона

> Лучше сделать сразу так, чтобы впоследствии ничего не делать

```sql
-- ## Города
create table towns (
    town_id serial primary key,
    name text not null
);

-- # СТО
create table station (
    id serial primary key,
    name text not null,
    town_id int not null references towns (town_id), -- поэтому сначала towns, либо alter
    address text not null,
    director text not null,
    phone char (11) not null,
);

-- ## Расписание
create table schedule (
    station_id int not null references station (id),
    day int not null check (day between 1 and 7),
    time_start time not null,
    time_end time not null,
    primary key (station_id, day) -- обеды не сделаем, для них (station_id, day, time_start)
);
```

Прайс лист

- Id
- СТО
- Стоимость помимо названия зависит от категории и сто, поэтому убираем стоимость в другую таблицу, также все три поля, от которых зависит, тоже в ту таблицу (останется id, название и описание)
- Категория тоже отдельная сущность
  Таблицы:
- Работа
  - id
  - name
  - description
- Тип ТС
  - id
  - name
- Стоимость
  - id
  - категория
  - СТО_id
  - Стоимость

ТС

Марка зависит от модели
Таблички:

- ТС
  - id
  - регистрационный номер
  - модель
  - владелец
- Модель
  - id
  - name
  - марка
  - тип тс
- Марка
  - id
  - name
- Person
  - id
  - ФИО

- phone

Если первичный ключ в другой таблице сложный, то ссылаемся мы на весь ключ, то есть указываем все атрибуты первичного ключа

Посещения

Список - нарушение, выносим в отдельную таблицу работ
Таблицы:

- Выполненная работа
  - id_посещения PK
  - id_работы PK
  - master
  - %
  - price - выделяем это поле, потому что в будущем цена может измениться, но мы должны помнить сколько заплатили в тот день
- Посещения
  - id
  - data
  - id_ts
  - id_station
  - ...
  - Итоговую стоимость считаем при надобности, не выделяем как поле, дублирование информации (иногда осознанное нарушение 3НФ, если нужно считать данные со многих таблиц, но берём ответственность за то, что эти данные совпадали)

person - переделать

```
-- ## Города
create table towns (
    town_id serial primary key,
    name text not null
);

-- # СТО
create table station (
    id serial primary key,
    name text not null,
    town_id int not null references towns (town_id), -- поэтому сначала
towns, либо alter
    address text not null,
    director text not null,
    phone char (11) not null
```

```
);

-- ## Расписание
create table schedule (
    station_id int not null references station (id),
    day int not null check (day between 1 and 7),
    time_start time not null,
    time_end time not null,
    primary key (station_id, day) -- обеды не сделаем, для них
(station_id, day, time_start)
);

-- # Стоимость
-- ## Тип ТС
create table type_ts (
    id serial primary key,
    name text not null
);

-- ## Работа
create table service (
    id serial primary key,
    name text not null,
    description text not null
);

-- ## Цены
create table price (
    id_type_ts int not null references type_ts (id),
    id_service int not null references service (id),
    id_station int not null references station (id),
    price int not null,
    primary key (id_type_ts, id_service, id_station)
);

-- # ТС
-- ## Марка
create table brand (
    id serial primary key,
    name text not null
);

-- ## Модель
create table model (
    id serial primary key,
    name text not null,
    id_brand int not null references brand (id),
    id_type_ts int not null references type_ts (id)
);
```

```sql
-- ## Человек-владелец
create table person (
    id serial primary key,
    fname text not null,
    lname text not null,
    phone char (11) not null
);

-- ## ТС
create table ts (
    id serial primary key,
    reg_number char (9) not null,
    id_model int references model (id),
    id_person int references person (id)
);

-- # Посещения
```

Лекция 6

Подзапросы

Подзапросы - это запросы, которые вкладываются внутрь другого запроса. Подзапросы могут возвращать одно значение, несколько значений или целую таблицу, в зависимости от контекста.

Скалярный подзапрос

> Возвращает одно значение (одну строку или столбец). Используется в `SELECT, WHERE, HAVING`

```sql
select name, (select avg(salary) from employees) as avg_salary
from employees;
```

Подзапрос, возвращающий несколько значений

> Используется операторами `IN, ANY, ALL, EXISTS`

```sql
select name
from employees
where department_id in (select id from department where location = 'New
York');
```

- `ANY` - используется для сравнения значения с хотя бы одним

- `ALL` - используется для сравнения значения со всеми значениями из результата подзапроса

```
-- salary больше минимального
select name, salary
from employees
where salary > any (select salary from employees where department_id =
10);
-- лучше min в данном запросе вместо any
```

```
-- salary больше всех
select name, salary
from employees
where salary > all (select salary from employees where department_id =
10);
```

- `EXISTS` - проверяет, возвращает ли подзапрос хотя бы одну строку

```
select name
from employees e
where exists(
    select 1
    from departments d
    where d.id = e.department_id and d.location = 'New York'
);
```

Подзапрос, который возвращает таблицу

> Может использоваться в `FROM` как временная таблица

```
select e.name, d.department
from employees e
    join (select id, name as department_name from department) d
    on e.department_id = d.id;
```

Использование подзапросов

- В `SELECT` : для вычисления значений.
- В `FROM` : для создания временных таблиц.
- В `WHERE` : для фильтрации данных.
- В `HAVING` : для фильтрации групп.
- В `INSERT, UPDATE, DELETE` : для работы с данными на основе результатов подзапроса

Связанные подзапросы

> Если подзапрос связан с основным и не может быть выполнен отдельно.
> Связанные подзапросы почти не переписываются через `JOIN`

В условии WHERE

```
-- Выводит всех сотрудников, у которых зарплата больше среднего в ЕГО
отделе
select name, salary
from employees e
where salary > (select avg(salary))
                from employees e2
                where e2.department_id = e1.department_id);
```

В условии `where` есть ссылка на e1, поэтому не получится выполнить отдельно. Подзапросы довольно медленные, используются по мере необходимости, в отсутствии других возможностей.

Представления

Представления (`View`) - ещё один объект в БД - это виртуальная таблица, которая представляет результат выполнения запроса. Хранится только скрипт, во время обращения - выполняется этот скрипт и возвращается его результат. Не работает `ORDER BY`
Основные характеристики:

- Виртуальность
- Динамичность
- Абстракция
- Безопасность

Создание представлений

```
create view <name> as
select <column1>, ...
from <table1>
where <conditon>
```

Изменение представления

```
create or replace view_name as
select <new_column1>, ...
```

```
from <new_table1>
where <newconditon>
```

Количество атрибутов и их типы должны совпадать и названия тоже

Удаление представления

```
drop view view_name
```

Материализованные представления

> Представляют собой сохранённый запрос и закешированные данные

Создаются с помощью `CREATE MATERIALIZED VIEW`
Хранят результат выполнения запроса на момент создания или обновления.
Требуют явного обновления (`REFRESH MATERIALIZED VIEW`)

```
create materialized view mat_view_name as
select column1, ...
from table_name
where condition;
```

```
refresh materialized view mat_view_name
```

Оператор with

> Позволяет в пределах запроса создать виртуальную таблицу

```
with active_employees as (
    select id, name, department
    from employees
    where active = true
)
select * from active_employees;
```

что-то

Рекурсия

```
with RECURSIVE employee_hierarchy as (
    select id, name, manager_id
    from employees
    where manager_id не дописано
)
```

Практика 13-14

```sql
select *
from company
    join country on country.id_country = company.id_country;

select *
from company
    left join country on country.id_country = company.id_country;

select *
from company
    right join country on country.id_country = company.id_country;

-- from moodle
-- 1
select t.trip_no, c.name
from trip t
    left join company c on c.id_comp = t.id_comp;
-- 2
select psg.lname, psg.name, pit.place
from pass_in_trip pit
    join passenger psg on psg.id_psg = pit.id_psg;
-- 3
select distinct psg.lname, psg.name
from pass_in_trip pit
    left join passenger psg on psg.id_psg = pit.id_psg
where pit.date between '2003-04-01'::date and '2003-04-30'::date;
-- 4
select distinct psg.lname, psg.name
from pass_in_trip pit
    join passenger psg on psg.id_psg = pit.id_psg
where substring(pit.place, 1, length(pit.place) - 1)::int = 4
-- 5
select t.name
from towns t
    left join trip tr on tr.id_town_to = t.id_town
    left join pass_in_trip pt on pt.trip_no = tr.trip_no
    left join passenger p on p.id_psg = pt.id_psg
where p.name like 'Bruce' -- and p.lname like 'Willis'

select tt.name
from passenger p
    left join pass_in_trip pit on pit.id_psg = p.id_psg
    left join trip t on t.trip_no = pit.trip_no
    left join towns tt on tt.id_town = t.id_town_to
where p.name = 'Bruce'
-- 6
-- my:
```

```sql
select c.name
from company c
    left join trip t on t.id_comp = c.id_comp
    left join pass_in_trip pt on pt.trip_no = t.trip_no
    left join passenger p on p.id_psg = pt.id_psg
where p.name like 'Bruce' and p.lname like 'Willis'

-- correct:
select c.name
from pass_in_trip pit
    left join trip t on t.trip_no = pit.trip_no
    left join company c on c.id_comp = t.id_comp
    left join passenger p on p.id_psg = pit.id_psg
where p.name like 'Bruce' and p.lname like 'Willis'
-- 7
select p.name, p.lname
from pass_in_trip pit
    join trip t on t.trip_no = pit.trip_no
    join towns tt on tt.id_town = t.id_town_to
    join company c on c.id_comp = t.id_comp
    join country cc on cc.id_country = c.id_country
    join passenger p on p.id_psg = pit.id_psg
where tt.name = 'Paris' and cc.name = 'France'
-- 8
select t.trip_no, c.name, cc.name
from trip t
    join trip_days td on td.trip_no = t.trip_no
    join company c on c.id_comp = t.id_comp
    join country cc on c.id_country = cc.id_country
where td.day = 1
-- 9
select p.name, p.lname
from pass_in_trip pit
    join passenger p on p.id_psg = pit.id_psg
    join trip t on t.trip_no = pit.trip_no
    join towns t1 on t.id_town_from = t1.id_town
    join towns t2 on t.id_town_to = t2.id_town
where t1.name = 'Moscow' and t2.name = 'Rostov'
-- 10
select t.trip_no, town_from.name, town_to.name, t.price
from trip t
    join company c on c.id_comp = t.id_comp
    join country cc on c.id_country = cc.id_country
    join towns town_from on t.id_town_from = town_from.id_town
    join towns town_to on t.id_town_to = town_to.id_town
where cc.name = 'Russia'
-- 11
select p.name, p.lname, pit.place
from pass_in_trip pit
    join passenger p on p.id_psg = pit.id_psg
```

```
    join trip t on t.trip_no = pit.trip_no
    join trip_days td on t.trip_no = td.trip_no
where td.day = 3
```

Практика 15

Агрегирующие функции

count

```
select count(*) from passenger; -- количество строк, которые вернули
select count(sex) from passenger; -- количество не NULL значений
select count(distinct sex) from passenger; -- количество уникальных (кроме
NULL) значений
```

min и max

```
select min(price), max(price) from trip
```

group by

```
select sex, count(*) from passenger -- ошибка
```

```
select sex, count(*) from passenger
group by sex -- группирует и выполняет агрегирующую функцию для каждой
группы
```

1. `from`
2. `where`
3. `group by`
   В `GROUP BY` должны участвовать все атрибуты, которые не являются агрегирующими функциями
   `GROUP BY` не используется если в `SELECT` только агрегирующие функции

Лекция 7

Транзакции и уровни изоляции

Транзакция - последовательность операция, выполняемых как единое целое. Транзакция либо завершается успешно (все изменения сохраняются), либо откатываются (и все изменения отменяются).

> *Пример*: перевод денег с одного счёта на другой

Если один из шагов **не выполняются**, то вся операция должна быть **отменена**.

Свойства транзакций (ACID)

- **Atomicity (Атомарность)** - транзакция выполняется как единое целое, все операции выступают как один неделимый объект
- **Consistency (Согласованность)** - Транзакция переводит состояние базы из одного согласованного(все данные не нарушают ограничения, которые существуют в базе) состояния в другое согласованное состояние
- **Isolation (Изоляция)** - параллельные транзакции не должны влиять друг на друга
- **Durability (Долговечность)** - После завершения транзакции изменения сохраняются, даже в случае сбоя в системе, *изменения не отменяются*

Синтаксис

`BEGIN` - начало транзакции
`COMMIT` - завершение транзакции
`ROLLBACK` - отменяет всё, что сделано в рамках текущей транзакции. Но не всегда, иногда транзакция выходит за свои пределы.

> Например, если мы добавляем запись в таблицу, в которой есть последовательность `SERIAL`, то добавления отменится, а номер последовательности не уменьшится

```
begin;
insert into ...;
commit;
```

```
begin;
insert into ...
rollback;
```

`SAVEPOINT` - создание точки сохранения внутри транзакции

```
begin;
insert into ...;

savepoint my_save;

update ...;

rollback to my_save;
```

```
commit;
```

База данных может сама закрывать транзакции и только в одном случае она закрывает транзакцию корректно (`COMMIT`):

> Когда пользователь корректно отключился и изменения, которые прописаны в транзакции, успешно выполняются.

Во всех остальных случаях `ROLLBACK`
База данных автоматически создаёт транзакцию для `INSERT, DELETE, UPDATE`

Счётчик транзакций

> Внутренний механизм, который используется для управления транзакциями и их изоляциями. Каждой транзакции присваивается уникальный номер `TRANSACTION ID` или `XID` . Счётчик работает как объект кластера, а не в рамках конкретной базы данных.

`XID` - 32-разрядное число. При создании строки с `XID=100` , то такая строчка будет видна только транзакциям, у которых `XID > 100` . В случае достижения максимального числа транзакций `XID` начинается с начала.

Уровни изоляции

Проблемы параллельного выполнения операций

- **Потерянное обновление** - две транзакции одновременно изменяют одни и те же данные, одно из изменений потеряется
- **Грязное чтение** - ...
- **Неповторяемое чтение** - транзакции читают одни и те же данные. но получают различные значения из-за изменения внесённых другой транзакции
- **Фантомное чтение** - разный набор строк в результате из-за изменения внесённых другой транзакции

Виды изоляций

1. `Read Uncommitted` - Чтение незафиксированных данных
   - Разрешает грязное чтение
   - Самый низкий уровень изоляции
   - Используется, когда важна производительность, а целостность данных не критична
2. `Read Committed` - Чтение зафиксированных данных
   - Запрещает грязное чтение
   - ...

    - ...
3. `Repeatable Read`
    - ...
    - ...
    - ...
4. `Serializable`
    - Самый строгий уровень изоляции
    - Запрещает грязное, неповторяемое и фантомное чтение
    - Транзакции выполняются так, как если бы они выполнялись последовательно

Синтаксис

Для конкретной транзакции:

```
begin transaction isolation level *уровень изоляции*
```

Для всей сессии:

```
set transaction isolation level *уровень изоляции*
```

Реализация

- **Блокировка (Locks)** - Транзакции блокируют данные, чтобы другие транзакции не могли ...
  - Блокировка на уровне таблиц
  - Блокировка на уровне строк
  Блокировки доступны в **автоматическом режиме**, а также можно в **ручном режиме**, но надо позаботиться о минимизации времени блокировки.

Практика 17

Любой подзапрос, используемый в `WHERE`, должен иметь только один атрибут `p.*` - выводит все столбцы таблицы `p`

```
-- select ...
-- from ...
-- where ... (select ...)

--Part 1
--1
select * from company
where id_comp not in (
    select distinct id_comp from trip
);
```

```sql
--2
select *
from passenger
where id_psg not in (
    select distinct id_psg from pass_in_trip
);
-- Найти рейсы, билет на которые дороже средней стоимость
select trip_no from trip
where price > (select avg(price) from trip);

--3
select * from passenger
where id_psg in (select distinct id_psg from pass_in_trip
    where trip_no in (select trip_no from trip
        where price > (select avg(price) from trip)
    )
);

--4
select * from passenger
where id_psg not in (
    select distinct id_psg
    from pass_in_trip pit
        join trip t on t.trip_no = pit.trip_no
        join towns tw on tw.id_town = t.id_town_to
    where tw.name = 'Paris'
);

--5
select * from country
where id_country in (
    select distinct comp.id_country
    from company comp
        join trip t on t.id_comp = comp.id_comp
        join towns tw on tw.id_town = t.id_town_to
    where tw.name = 'Moscow'
);

--6
select * from company
where id_comp in (
    select distinct id_comp from trip
    group by id_comp
    having count(*) > (
        select count(*) from trip
        where id_comp in (
            select id_comp from company
            where name = 'Aeroflot'
        )
```

```
    )
);

-- Part 2
-- 1
select * from passenger
where id_psg not in (
    select distinct id_psg from pass_in_trip
    where trip_no in (
        select distinct trip_no from trip
        where id_comp in (
            select distinct id_comp from company
            where name = 'Aeroflot'
        )
    )
);
```

Практика 18

Практика 19

```
-- Part 2
-- 1
select * from passenger
where id_psg not in (
    select distinct id_psg from pass_in_trip
    where trip_no in (
        select distinct trip_no from trip
        where id_comp in (
            select distinct id_comp from company
            where name = 'Aeroflot'
        )
    )
);

-- 2
select * from towns
where id_town in (
    select distinct id_town_to from trip
    where price > (
        select avg(price) from trip
        where id_town_to in (
            select id_town from towns
            where name = 'Moscow'
        )
    )
);
```

```sql
-- 3
select * from passenger
where id_psg in (
    select distinct id_psg from pass_in_trip
    where trip_no in (
        select trip_no from trip
        where price > (select avg(price) from trip)
    )
);

-- 4
select distinct trip_no from trip_days
group by trip_no
having string_agg(day::text, ',') = (
    select string_agg(day::text, ',') from trip_days
    where trip_no = 1101
);


-- 5
select * from towns
where id_town in (
    select id_town_from from trip
    group by id_town_from
    having count(*) > (
        select avg(cnt) from (
            select count(*) as cnt from trip
            group by id_town_from
        )
    )
);

-- 6
select * from company
where id_comp in (
    select distinct id_comp from trip
    group by id_comp
    having count(*) > (
        select avg(cnt) from (
            select count(*) as cnt from trip
            group by id_comp
        )
    )
)
```

Практика 20

Связанные подзапросы

> `NULL` в БД такое же исключение, как и `0` в математике. Любая операция с `NULL` будет `NULL`. Любое сравнение с `NULL` возвращает `FALSE`

coalesce

`coalesce()` - возвращает первый ненулевой аргумент (не `NULL`)

```
-- Part 3
-- 1
select *
from trip t
where t.price > (
    select avg(price) from trip t2
    where t2.id_comp = t.id_comp
);
-- 2
select pit.* from pass_in_trip pit
    join trip t on t.trip_no = pit.trip_no
    join passenger p on p.id_psg = pit.id_psg
where t.price > (
    select avg(price) from trip t2
        join pass_in_trip pit2 on pit2.trip_no = t2.trip_no
        join passenger p2 on p2.id_psg = pit2.id_psg
    where p2.sex = p.sex
);

-- Example (подзапрос в select) и про null
select pit.*, (
    select avg(price) from trip t2
        join pass_in_trip pit2 on pit2.trip_no = t2.trip_no
        join passenger p2 on p2.id_psg = pit2.id_psg
    where coalesce(p2.sex, '-') = coalesce(p.sex, '-') -- NULL поменяется
на '-', всё отсальное без изменений
)
from pass_in_trip pit
    join trip t on t.trip_no = pit.trip_no
    join passenger p on p.id_psg = pit.id_psg
where t.price > (
    select avg(price) from trip t2
        join pass_in_trip pit2 on pit2.trip_no = t2.trip_no
        join passenger p2 on p2.id_psg = pit2.id_psg
    where p2.sex = p.sex
);

-- 3
select * from trip t
    join company comp on comp.id_comp = t.id_comp
```

```sql
where t.price > (
    select avg(price) from trip t2
        join company comp2 on comp2.id_comp = t2.id_comp
    where comp.id_country = comp2.id_country
);

-- 4
select distinct p.* from passenger p
    join pass_in_trip pit on pit.id_psg = p.id_psg
    join trip t on t.trip_no = pit.trip_no
where t.price > (
    select avg(price) from trip t2
    where t2.id_comp = t.id_comp
);

-- 5
select distinct trip_no from trip_days
where trip_no != 1101
group by trip_no
having string_agg(day::text, ',' order by day) = (
    select string_agg(day::text, ',' order by day) from trip_days
    where trip_no = 1101
);

-- 6
select p.* from passenger p
    join pass_in_trip pit on pit.id_psg = p.id_psg
    join trip t on t.trip_no = pit.trip_no
group by p.id_psg
having string_agg(t.trip_no::text, ',' order by t.trip_no) = (
    select string_agg(t2.trip_no::text, ',' order by t2.trip_no) from trip
t2
        join company comp on comp.id_comp = t2.id_comp
    group by t2.id_comp
    where comp.name = 'Aeroflot'
);
```

Практика 21

Метаданные

Информация о структурах, свойствах и отношениях между объектами в БД, но не содержат пользовательские данные.
Эти сведения лежат в двух схемах:

- **information_schema** - стандарт SQL-схема, которая представляет единый интерфейс для доступа к метаданным. (Поддерживается всеми базами данных)
- **pg_catalog** - только в PostgreSQL

Основные задачи:

- Универсальность
- Безопасность
- Удобство

Представления - сохранённый SQL запрос
Основные представления:

- **tables** - информации о таблицах
- **columns** - сведения о столбцах
- **views** - данные о представлениях
- **schemata** - информация о схемах
- **table_constraints** - ограничения таблиц

```
-- example
select * from information_schema.views
where table_schema = 'information_schema' and table_name ilike
'%constraint%';

select * from information_schema.constraint_column_usage
where constraint_name in (
    select constraint_name from information_schema.table_constraints
    where table_name = 'trip' and constraint_type = 'FOREIGN KEY'
)


-- task
select 'create table trip ('
union all
select column_name || ' ' || data_type ||
    case
        when is_nullable = 'NO' then ' not null'
        else ''
    end || ','
from information_schema.columns
where table_name = 'trip'
union all
select 'primary key (' ||
    (select string_agg(column_name, ',') from
information_schema.constraint_column_usage
    where constraint_name in (
        select constraint_name from information_schema.table_constraints
```

```
        where table_name = 'trip' and constraint_type = 'PRIMARY KEY'
    )) || '),'
union all
select 'foreign key (' || key_usg.column_name || ') ' || 'references ' ||
usg.table_name || ' (' || usg.column_name || '),' from
information_schema.constraint_column_usage usg
    join information_schema.key_column_usage key_usg on
key_usg.constraint_name = usg.constraint_name
where usg.constraint_name in (
    select constraint_name from information_schema.table_constraints
    where table_name = 'trip' and constraint_type = 'FOREIGN KEY'
)
union all
select ')'
--
```

Практика 23

Представления

```
create or replace view pass1 as
select id_psg,
    trim(coalesce(name, '') || ' ' || coalesce(lname, '')) as name, --
coalesce - первое не-NUll значение, trim - удаление пробелов
    sex
from passenger;

select * from pass1
order by id_psg;

update passenger set lname = 'Willis-1' where id_psg = 1;

create materialized view pass2 as
select id_psg,
    trim(coalesce(name, '') || ' ' || coalesce(lname, '')) as name, --
coalesce - первое не-NUll значение, trim - удаление пробелов
    sex
from passenger;


update passenger set lname = 'Willis' where id_psg = 1;

select * from pass1 order by id_psg;
select * from pass2 order by id_psg;

refresh materialized view pass2;
```

```
select * from pass2 order by id_psg;
```

Временные таблицы

Только в рамках одной сессии

```
create temporary table temTable(
    id int not null,
    name text
);

insert into temTable values (1, 'abc');
select * from temTable; -- только в этой сессии, если ещё раз
подключиться, то там уже не будет
```

WITH

Когда создание представления особо не нужно, но подобное хотелось бы

```
with pass3 as (
    select id_psg,
    trim(coalesce(name, '') || ' ' || coalesce(lname, '')) as name, --
coalesce - первое не-NUll значение, trim - удаление пробелов
    sex
    from passenger
)
select * from pass3;
```

Лекция 8 (Last Lection)

Индексы и производительность

Индекс - отдельная структура, отдельный файл, в котором определены специальным образом данные

- **B-Tree** - структура данных, самобалансирующегося дерева.
- **Хэш-Фукнция**
- **GIST (Обобщённое поисковое дерево)** - каждый узел описывает диапазон или множество значений и связан с **предикативной функцией**(типо возвращает "да" или "нет")
- **GIN (Обратный индекс)** - Он работает с типами данных, которые состоят из нескольких элементов, неатомарных.
- **BRIN** - все данные разбиваются на блоки, а потом по каждому блоку строятся характеристики

В PostgreSQL используются B-Tree(по умолчанию) и hash-функция

Создание индексов

> Создание индексов - важный шаг для оптимизации производительности запросов. Неправильное или избыточное создание индексов может привести к замедлению операций записи (INSERT, UPDATE, DELETE) и увеличение объёма данных

```
create index idx_name on table_name (column_name);
-- явно указывая тип индекса
create index idx_name on table_name (column_name) using btree;
```

Анализ запросов

Для анализа запросов используются конструкции

```
EXPLAIN и EXPLAIN ANALYZE
```

Пример:

```
explain select * from table2 where c_isso in (select c_isso in from
table1)
```

ДОПИСАТЬ

Практика 24

В большой БД, тестовые запросы ограничиваем с помощью `limit`:

```
select * from temp_table
limit 10;
```

Выполняет запрос и выводит результаты анализа

```
explain analyze select * from table_name
```

Практика 25

- Классическая SQL-инъекция

  > Изменяется основной запрос

- Union-атаки

> Извлечение данных

- Слепые SQL-инъекции

> Нет видимой реакции