

Базы данных

Лекция 2.
Основы SQL. DDL. DML.

МФТИ, 2024

Игорь Шевченко

[@igorshvch](https://twitter.com/igorshvch)

I. Дополнительная информация о реляционных операциях

Дополнительно:
обозначения операций реляционной алгебры

Операции		Символьная запись	Полная запись
Теоретико- множественные	Объединение	$R_1 \cup R_2$	$R_1 \text{ UNION } R_2$
	Пересечение	$R_1 \cap R_2$	$R_1 \text{ INTERSECT } R_2$
	Разность	$R_1 - R_2$	$R_1 \text{ MINUS } R_2$
	Декартово произведение	$R_1 \times R_2$	$R_1 \text{ TIMES } R_2$
Реляционные	Выборка (ограничение)	$\sigma_{\text{condition}}(R)$	$R \text{ WHERE condition}$
	Проекция	$\pi_{\text{attr}_1, \dots, \text{attr}_N}(R)$	$R[\text{attr}_1, \dots, \text{attr}_N]$
	Соединение	$R_1 \bowtie R_2$	$R_1 \text{ JOIN } R_2$
	Деление	$R_1 \div R_2$	$R_1 \text{ DIVIDE } R_2$

Дополнительно:
формальное определение деления

$$R_1 \div R_2 = \pi_{R_1-R_2}(R_1) \overset{3}{-} \pi_{R_1-R_2} \left(\left(\pi_{R_1-R_2}(R_1) \overset{1}{\times} R_2 \right) \overset{2}{-} R_1 \right)$$

- Если игнорировать операции проекции (которые в формуле приведены с дополнительным условием разности отношений R_1 и R_2), то операция деления состоит из трех последовательных шагов:
 - Построить декартово произведение
 - Исключить уже существующие кортежи из этого произведения
 - Исключить из кортежей, являющихся потенциальными ответами, кортежи, которые такими ответами не являются

Дополнительно:
Пример использования операции деления

- Допустим, у нас есть два отношения – поставщики и товары.
- Нам нужно найти всех поставщиков, цена товаров у которых больше 1000 рублей

Поставщики

p_id	t_id	p_имя	p_регион
П_1	T_1	ООО "Ромашка"	Москва
П_2	T_3	ООО "Подсолнух"	Хабаровск
П_2	T_5	ООО "Подсолнух"	Хабаровск
П_3	T_3	ООО "Василек"	Йошкарولا
П_3	T_4	ООО "Василек"	Йошкарولا
П_4	T_6	ООО "Ландыш"	Дербент
П_5	T_1	ООО "Астра"	Тверь
П_5	T_2	ООО "Астра"	Тверь
П_5	T_3	ООО "Астра"	Тверь
П_5	T_4	ООО "Астра"	Тверь
П_5	T_5	ООО "Астра"	Тверь
П_5	T_6	ООО "Астра"	Тверь

Товары

t_id	t_имя	t_цена
T_1	Доски	800,00 ₽
T_2	Кирпич	1 200,00 ₽
T_3	Цемент	1 100,00 ₽
T_4	Гвозди	600,00 ₽
T_5	Щебень	500,00 ₽
T_6	Песок	300,00 ₽

Дополнительно:
Пример использования операции деления

- Для ответа нам нужны только правильные идентификаторы поставщиков, поэтому отсечем лишнюю информацию, построив две проекции:
- $R_1 = \pi_{p_{id}, t_{id}}(\text{Поставщики})$
- $R_2 = \pi_{t_{id}}(\sigma_{\text{Т}_{\text{цена}} > 1000}(\text{Товары}))$

Поставщики	
p_id	t_id
П_1	Т_1
П_2	Т_3
П_2	Т_5
П_3	Т_3
П_3	Т_4
П_4	Т_6
П_5	Т_1
П_5	Т_2
П_5	Т_3
П_5	Т_4
П_5	Т_5
П_5	Т_6

Товары	
t_id	
Т_2	
Т_3	

Дополнительно:
Пример использования операции деления

- Построим проекцию $\pi_{R_1 - R_2}(R_1)$:

Поставщики

п_id
п_1
п_2
п_3
п_4
п_5

Дополнительно:
Пример использования операции деления

- Построим декартово произведение $\pi_{R_1 - R_2}(R_1) \times R_2$:

$\pi_{R_1 - R_2}(R_1) \times R_2$	
п_id	т_id
п_1	т_2
п_1	т_3
п_2	т_2
п_2	т_3
п_3	т_2
п_3	т_3
п_4	т_2
п_4	т_3
п_5	т_2
п_5	т_3

Дополнительно:
Пример использования операции деления

- Произведем вычитание:
 $(\pi_{R_1 - R_2}(R_1) \times R_2) - R_1$

$\pi_{R_1 - R_2}(R_1) \times R_2$

п_id	т_id
п_1	т_2
п_1	т_3
п_2	т_2
п_2	т_3
п_3	т_2
п_3	т_3
п_4	т_2
п_4	т_3
п_5	т_2
п_5	т_3

—

Поставщики

п_id	т_id
п_1	т_1
п_2	т_3
п_2	т_5
п_3	т_3
п_3	т_4
п_4	т_6
п_5	т_1
п_5	т_2
п_5	т_3
п_5	т_4
п_5	т_5
п_5	т_6

Дополнительно:
Пример использования операции деления

$\pi_{R_1 - R_2}(R_1) \times R_2$

n_id	t_id
П_1	Т_2
П_1	Т_3
П_2	Т_2
П_2	Т_3
П_3	Т_2
П_3	Т_3
П_4	Т_2
П_4	Т_3
П_5	Т_2
П_5	Т_3

—

Поставщики

n_id	t_id
П_1	Т_1
П_2	Т_3
П_2	Т_5
П_3	Т_3
П_3	Т_4
П_4	Т_6
П_5	Т_1
П_5	Т_2
П_5	Т_3
П_5	Т_4
П_5	Т_5
П_5	Т_6

=

$(\pi_{R_1 - R_2}(R_1) \times R_2) - R_1$

n_id	t_id
П_1	Т_2
П_1	Т_3
П_2	Т_2
П_3	Т_2
П_4	Т_2
П_4	Т_3

Дополнительно:
Пример использования операции деления

- Построим проекцию:

$$\pi_{R_1 - R_2} \left(\left(\pi_{R_1 - R_2}(R_1) \times R_2 \right) - R_1 \right)$$

$$\pi_{R_1 - R_2} \left(\left(\pi_{R_1 - R_2}(R_1) \times R_2 \right) - R_1 \right)$$

n_id
п_1
п_2
п_3
п_4

Дополнительно:
Пример использования операции деления

- Вычислим последнюю разность:

$$\pi_{R_1-R_2}(R_1) - \pi_{R_1-R_2}\left(\left(\pi_{R_1-R_2}(R_1) \times R_2\right) - R_1\right)$$

Поставщики

п_id
п_1
п_2
п_3
п_4
п_5

—

$\pi_{R_1-R_2}((\pi_{R_1-R_2}(R_1) \times R_2) - R_1)$

п_id
п_1
п_2
п_3
п_4

=

п_id
п_5

Дополнительно:
Еще немного о делении

- Деление в реляционной алгебре похоже на арифметическое деление: при построении декартова произведения «частного» и «делителя» мы должны получить подмножество отношения-«делимого». При этом операция не всегда «обратима» - в примере выше исходного отношения-«делителя» при обращении операции мы не получим. Иными словами:

$$A \div B = C, \quad C \times B \subseteq A$$

- Деление также можно описать следующим образом:

Пусть имеются отношения $A(X, Y)$ и $B(Y)$, где атрибуты X, Y определены на одном и том же домене. Результатом деления $A \div B$ будет отношение с заголовком из атрибута X и телом, в которое входят кортежи $\langle x:X \rangle$ такие, что существует кортеж $\langle x:X, y:Y \rangle$, который принадлежит отношению A для всех кортежей $\langle y:Y \rangle$ из отношения B .

II. Что дальше читать про PostgreSQL?

Основные ресурсы по PostgreSQL

- [PostgreSQL: Documentation: 16: PostgreSQL 16.2 Documentation](#) – официальная документация, международная версия
- [PostgreSQL : Документация: 16: Документация к PostgreSQL 16.1 : Компания Postgres Professional](#) – официальная русскоязычная документация от Postgres Professional (компания Postgres Professional поставляет свои решения на основе PostgreSQL и вносит вклад в развитие Opensource-составляющей базовой версии PostgreSQL)
- На сайте Postgres Professional обратите внимание на раздел «образование» - [Образование \(postgrespro.ru\)](#). В нем много полезной информации, есть демо-база с данными для экспериментов, а также книги в открытом доступе
- Для общего развития: [Как Postgres Pro вытеснил Oracle? Выясняем вместе с Иваном Панченко – YouTube](#) – интервью от 06.02.2024 с Иваном Панченко (заместитель генерального директора Postgres Professional)

II. Structured query language - основная информация



Немного истории...

- В общем виде разработан инженерами IBM в 1973-1974 гг. Синтаксис представлен в статье «**SEQUEL: a structured English query language**» (Donald D. Chamberlin, Raymond F. Boyce, 1974). На момент «первого появления» формальное описание синтаксиса занимало всего две (!) страницы (объем всей статьи – 16 стр.), включал в себя аспекты определения данных и манипуляции данными (DDL, DML)...
- «***We wanted the language to be simple enough that ordinary people could “walk up and use it” with a minimum of training***» - Donald Chamberlin, Early history of SQL, 2012. Сегодня полное описание стандарта занимает несколько тысяч страниц...

Дополнительно: И еще немного истории

- 1970 год – консорциум CODASYLC (COncference on DAta SYstems Language - разработчики COBOL и первых стандартов для сетевых БД, один из участников – General Electric) предложил свой язык для работы с БД. Если коротко – он не был гораздо ближе к «железу», чем впоследствии SQL
- 1972, 1973 годы – Эдгар Кодд предложил два варианта нотации для работы с данными в рамках своей реляционной модели – реляционную алгебру и реляционное исчисление
- В 1973-1974 году Реймонд Бойс на основе наработок Кодда разрабатывает язык Square (статья выйдет только в 1975 году)
- 1974 год – программная статья Д. Чемберлина и Р. Бойса. В это время они работали над СУБД, в которой должна была быть реализована реляционная модель и язык запросов – IBM System R (стала первой РСУБД, в которой был реализован предок современного SQL, позднее «превратилась» в IBM DB2)
- 1977 год – наименование Sequel по юридическим причинам (смешение с другими товарными знаками) было сокращено до SQL



Дональд Чемберлин

Стандартизация SQL

- 1986 г – первый ANSI/ISO (1987) стандарт SQL. К тому времени SQL уже использовался в некотором количестве СУБД от различных производителей и его диалект разнился от вендора к вендору и от продукта к продукту.
- С 1986 года – доработки стандарта в среднем \approx раз в пять лет (на сайте ISO есть «опубликованные» стандарты).
- Действующая (с июля 2023) версия стандарта – SQL:2023

Стандарты по годам выхода

Год	Название	Описание
1986	SQL-86	Первая попытка формализации
1989	SQL-89	SQL-86 + ограничение целостности
1992	SQL-92	Очень много изменений
1999	SQL:1999	Согласование регулярных выражений, рекурсивные запросы, триггеры, поддержка процедурных и контрольных операций, не скалярные типы и объектно-ориентированные фичи. Поддержка внедрения SQL в Java и наоборот
2003	SQL:2003	Связанные с XML функции, оконные функции, стандартизованные сиквенсы и столбцы с автоматически генерируемыми значениями
2006	SQL:2006	Определен способ работы SQL с XML: способы импорта и хранения, публикация XML и обычных данных в формате XML
2008	SQL:2008	TRUNCATE, INSTEAD OF триггеры
2011	SQL:2011	Улучшены оконные функции
2016	SQL:2016	Добавляет сопоставление шаблонов строк, функции полиморфных таблиц, JSON

Стандарт SQL:2023

- Полноценная поддержка JSON – теперь это не просто строка, а самостоятельный тип данных (здесь стандарт отстал от практики, во многих РСУБД JSON уже давно поддерживается на уровне самостоятельного типа);
- Появился целый новый раздел (№ 16) - Property Graph Queries. Вне документируются правила, следование которым при реализации РСУБД и ее использовании позволяет строить запросы к табличным данным так, как если бы они были в графовой БД:

```
CREATE TABLE person (...);
CREATE TABLE company (...);
CREATE TABLE ownerof (...);
CREATE TABLE transaction (...);
CREATE TABLE account (...);

CREATE PROPERTY GRAPH financial_transactions
  VERTEX TABLES (person, company, account)
  EDGE TABLES (ownerof, transaction);

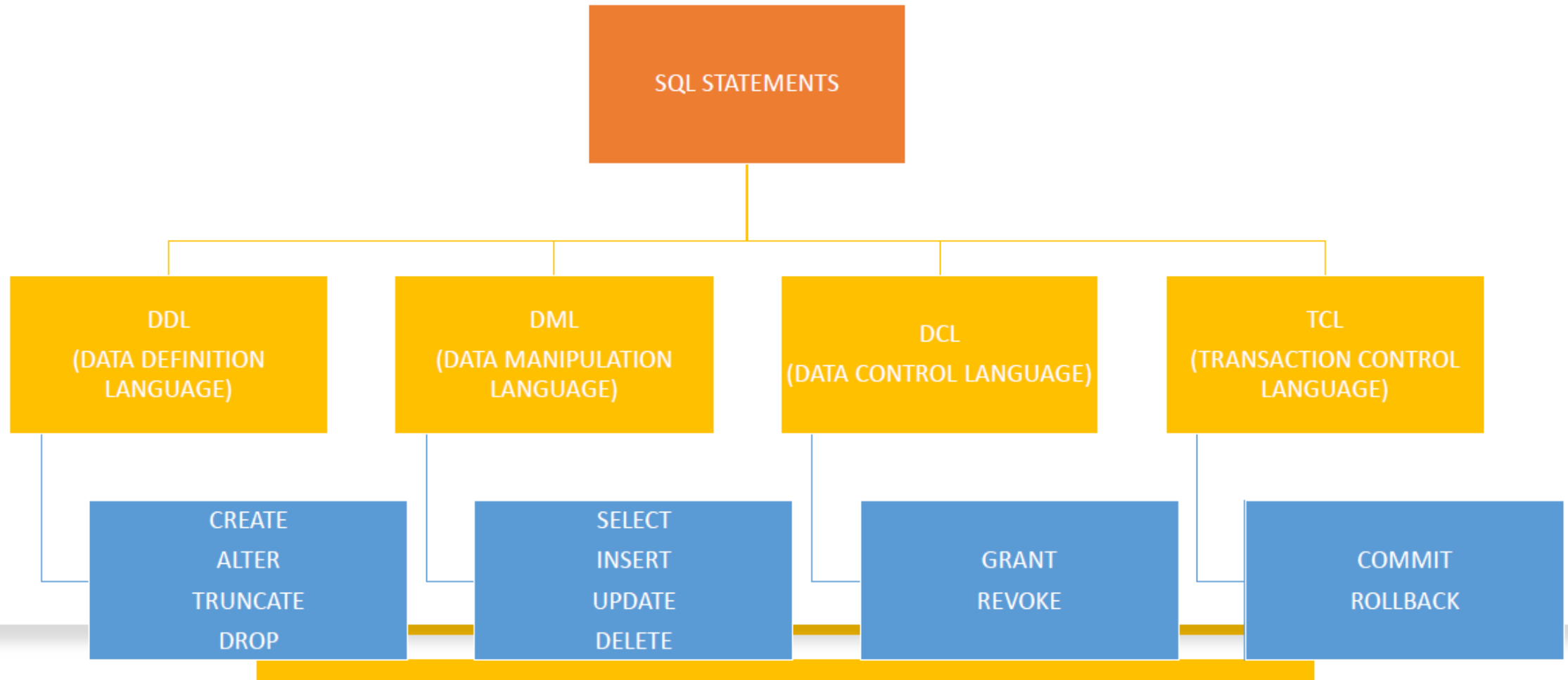
SELECT owner_name,
       SUM(amount) AS total_transacted
FROM financial_transactions GRAPH_TABLE (
  MATCH (p:person WHERE p.name = 'Alice')
    -[:ownerof]-> (:account)
    -[t:transaction]- (:account)
    <-[:ownerof]- (owner:person|company)
  COLUMNS (owner.name AS owner_name, t.amount AS amount)
) AS ft
GROUP BY owner_name;
```


Проблемы стандартов

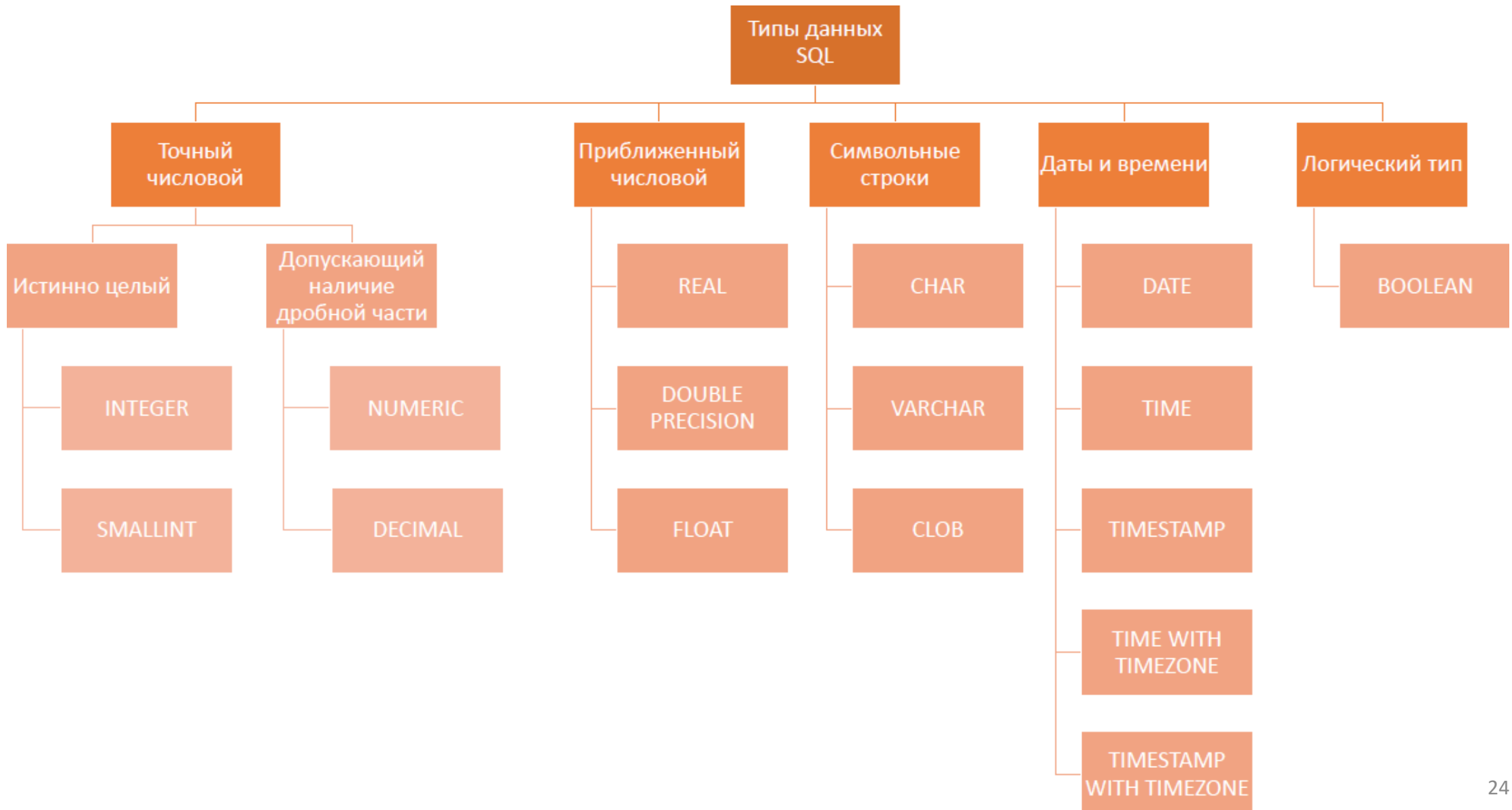
- Core-раздел введен только в 1992 году
- Производители обеспечивают соответствие только core-разделу
- Реализация в частностях все еще остается разной в зависимости от вендора:
 - Разный синтаксис
 - Разная логика
 - Разное действие на уровне обработки запросов
 - etc...



Группы операторов SQL



Основные типы данных SQL



Лексическая структура SQL*

- SQL-программа состоит из последовательности команд.
- Команда представляет собой последовательность компонентов, оканчивающуюся точкой с запятой («;»). Какие именно компоненты допустимы для конкретной команды, зависит от её синтаксиса.
- Компонентом команды может быть:
 - ключевое слово,
 - идентификатор,
 - идентификатор в кавычках,
 - оператор,
 - строка (или константа),
 - специальный символ.
- Компоненты обычно разделяются пробельными символами (пробел, табуляция, перевод строки).
- Синтаксис SQL **не очень строго определяет**, какие компоненты идентифицируют команды, а какие — их операнды или параметры
- Синтаксис конкретных команд следует уточнять в документации PostgreSQL

*См. подробнее: [PostgreSQL : Документация: 16: 4.1. Лексическая структура : Компания Postgres Professional](#)

Пример лексически правильной SQL-программы

`CREATE TABLE` students (id integer);

`CREATE TABLE` "LeCtUrEs" (id integer, course text);

`SELECT` * `FROM` students;

`SELECT` id `FROM` "LeCtUrEs";

`INSERT INTO` "LeCtUrEs" `VALUES` (123, 'Data Bases');

`UPDATE` "LeCtUrEs" `SET` id=124;

`UPDATE` "LeCtUrEs" `SET` course='Information retrieval' `WHERE` id=124;

Ключевые слова

Идентификатор в кавычках – case-sensitive!

Идентификатор

Специальный символ

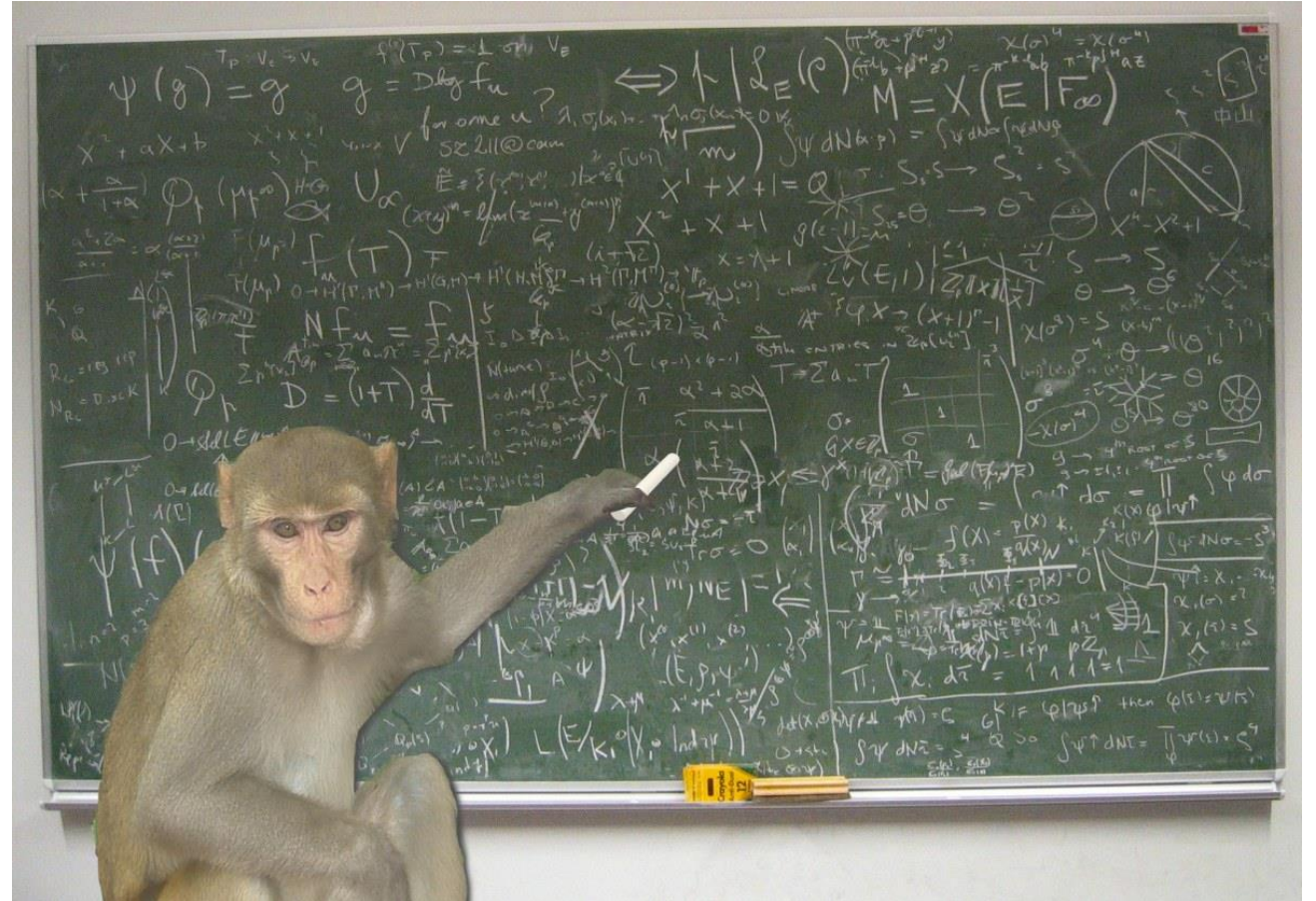
Оператор

Константы

Пример комментариев

```
CREATE TABLE lecturers (  
    id integer, --comment #1  
    name text, --comment #2  
    /*  
    Multiline  
    comments  
    */  
    birth_date date DEFAULT now()  
);
```

Начнем обзорное знакомство с основными группами команд SQL – DDL, DML

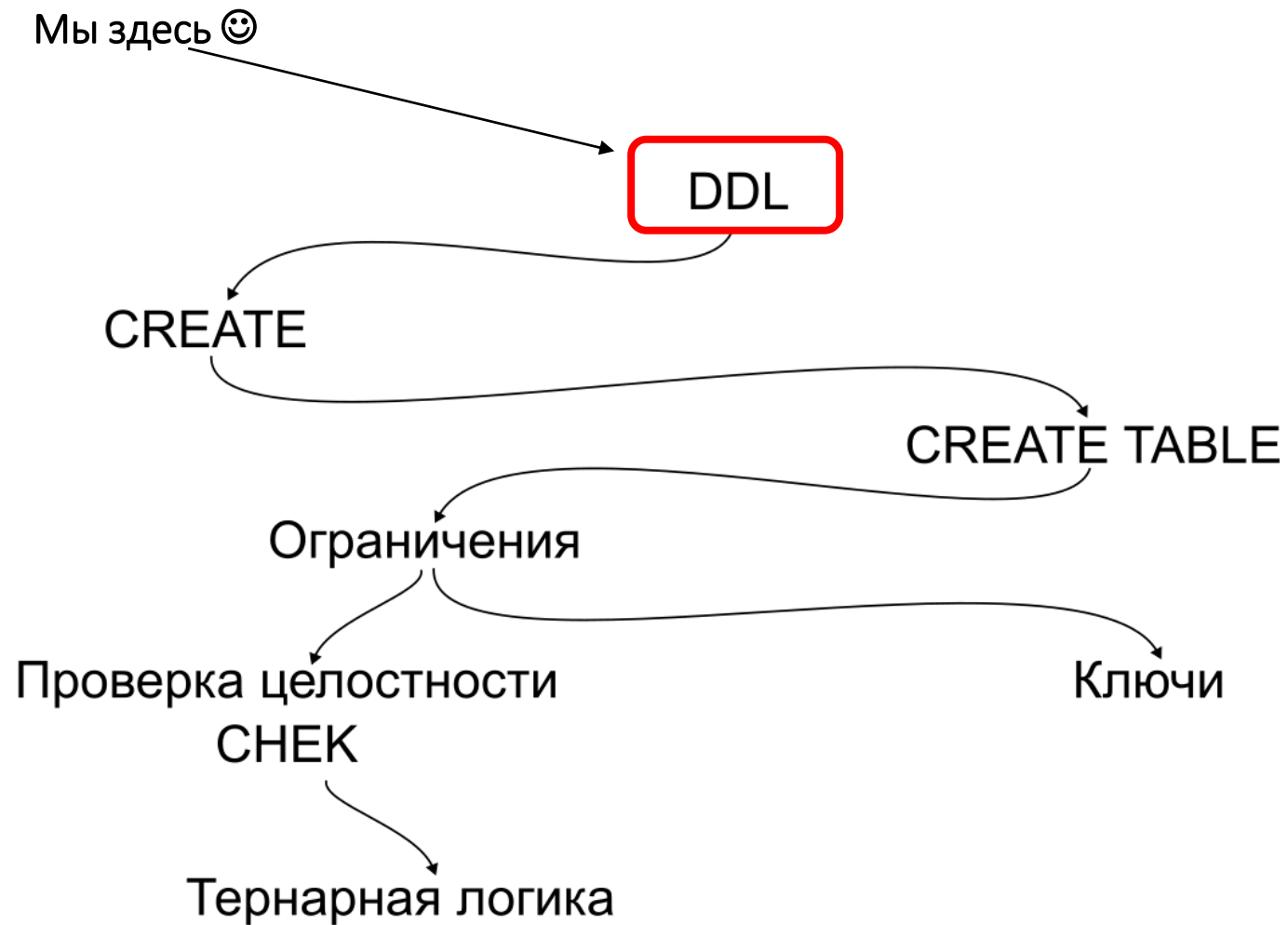


II. Основы DDL

Команды DDL

- ▶ CREATE
- ▶ ALTER
- ▶ TRUNKATE
- ▶ DROP

Траектория нашего разговора



CREATE TABLE – базовый синтаксис создания таблиц

```
CREATE [TEMPORARY] TABLE [ IF NOT EXISTS ] имя_таблицы ([  
    { имя_столбца тип_данных ограничение_таблицы }  
    [, ... ]  
])
```

ограничение_таблицы:

```
[ CONSTRAINT имя_ограничения ]  
    { CHECK ( выражение ) | UNIQUE [ NULLS [ NOT ] DISTINCT ] ( имя_столбца [, ... ] ) |  
    PRIMARY KEY ( имя_столбца [, ... ] ) | EXCLUDE ( элемент_исключения WITH оператор [,  
    ... ] ) [ WHERE ( предикат ) ] | FOREIGN KEY ( имя_столбца [, ... ] ) REFERENCES  
    целевая_таблица [ ( целевой_столбец [, ... ] ) ] [ MATCH FULL | MATCH PARTIAL | MATCH  
    SIMPLE ] [ ON DELETE ссылочное_действие ] [ ON UPDATE ссылочное_действие ] }  
    [ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```


Создаем таблицу students

```
CREATE TABLE students (
```

`student_id`

`int`

`PRIMARY KEY,`

`first_name varchar(50),`

`last_name varchar(50),`

`age int NOT NULL,`

`email varchar(100)`

```
);
```

Ограничение

Имя столбца

Тип данных
(int is alias for integer)

Ограничения в таблицах

- Служат средством поддержания целостности данных
- Гарантируют, что при внесении данных в таблицу не будут нарушены взаимосвязи между данными
- Обеспечивают ссылочную целостность между таблицами (отношениями) – за счет внешних ключей
- Ограничение определяется после типа данных либо отдельным «утверждением» при определении таблицы



Основные ограничения в SQL

- **NOT NULL** - соответствующий столбец не принимает значения NULL
- **UNIQUE** - группа из одного или нескольких столбцов таблицы может содержать только уникальные значения:

```
CREATE TABLE products (  
  product_no integer UNIQUE,  
  name text,  
  price numeric  
);
```

```
CREATE TABLE products (  
  product_no integer,  
  name text,  
  price numeric,  
  UNIQUE (product_no)  
);
```

```
CREATE TABLE example (  
  a integer,  
  b integer,  
  c integer,  
  UNIQUE (a, c)  
);
```

Основные ограничения в SQL: PRIMARY KEY

- **PRIMARY KEY** - ограничение первичного ключа означает, что образующий его столбец или группа столбцов может быть уникальным идентификатором строк в таблице. **Для этого требуется, чтобы значения были одновременно уникальными и отличными от NULL**

```
CREATE TABLE products (  
  product_no integer UNIQUE NOT NULL,  
  name text,  
  price numeric  
);
```



```
CREATE TABLE products (  
  product_no integer PRIMARY KEY,  
  name text,  
  price numeric  
);
```

```
CREATE TABLE example (  
  a integer,  
  b integer,  
  c integer,  
  PRIMARY KEY (a, c)  
);
```

Основные ограничения в SQL: FOREIGN KEY

- **FOREIGN KEY** - задает столбец или группу столбцов таблицы, значения которых должны совпадать со значениями столбцов первичного ключа другой таблицы, указанной в этом ограничении
- В схеме ниже таблицу orders называют **подчинённой таблицей**, а products — **главной**. Соответственно, столбцы называют так же подчинённым и главным (или **ссылающимся и целевым**).

```
CREATE TABLE products (  
  product_no integer PRIMARY KEY,  
  name text,  
  price numeric  
);
```

```
CREATE TABLE orders (  
  order_id integer PRIMARY KEY,  
  product_no integer REFERENCES products (product_no)  
  quantity integer  
);
```



Основные ограничения в SQL: FOREIGN KEY

- **FOREIGN KEY** может ссылаться на группу столбцов целевого отношения:

```
CREATE TABLE t1 (  
  a integer PRIMARY KEY,  
  b integer,  
  c integer,  
  FOREIGN KEY (b, c) REFERENCES other_table (c1, c2)  
);
```

Основные ограничения в SQL: CHECK

- **CHECK** - задает произвольное условие на значения одного или нескольких столбцов в одной строке таблицы.
- В ограничении CHECK задаётся выражение, возвращающее логический результат, по которому определяется, будет ли успешна операция добавления или изменения для конкретных строк:
 - TRUE или UNKNOWN - операция выполняется успешно
 - FALSE - возникает ошибка, операция не меняет ничего в базе данных

Основные ограничения в SQL: CHECK

- Пример ограничения CHECK:

```
CREATE TABLE products (  
    product_no integer,  
    name text,  
    price numeric CHECK (price > 0)  
);
```


Основные ограничения в SQL

- К ограничениям принято также относить команду DEFAULT. С ее помощью можно задать значение по умолчанию
- Значение задаётся выражением без переменных (в частности, перекрёстные ссылки на другие столбцы текущей таблицы в нём не допускаются). Также не допускаются подзапросы. Тип данных выражения, задающего значение по умолчанию, должен соответствовать типу данных столбца.
- Это выражение будет использоваться во всех операциях добавления данных, в которых не задаётся значение данного столбца. Если значение по умолчанию не определено, таким значением будет NULL.

```
CREATE TABLE order (  
    order_id INTEGER PRIMARY KEY,  
    order_number INTEGER NOT NULL,  
    order_date DATE DEFAULT now()::date  
);
```

Синтаксис ограничений

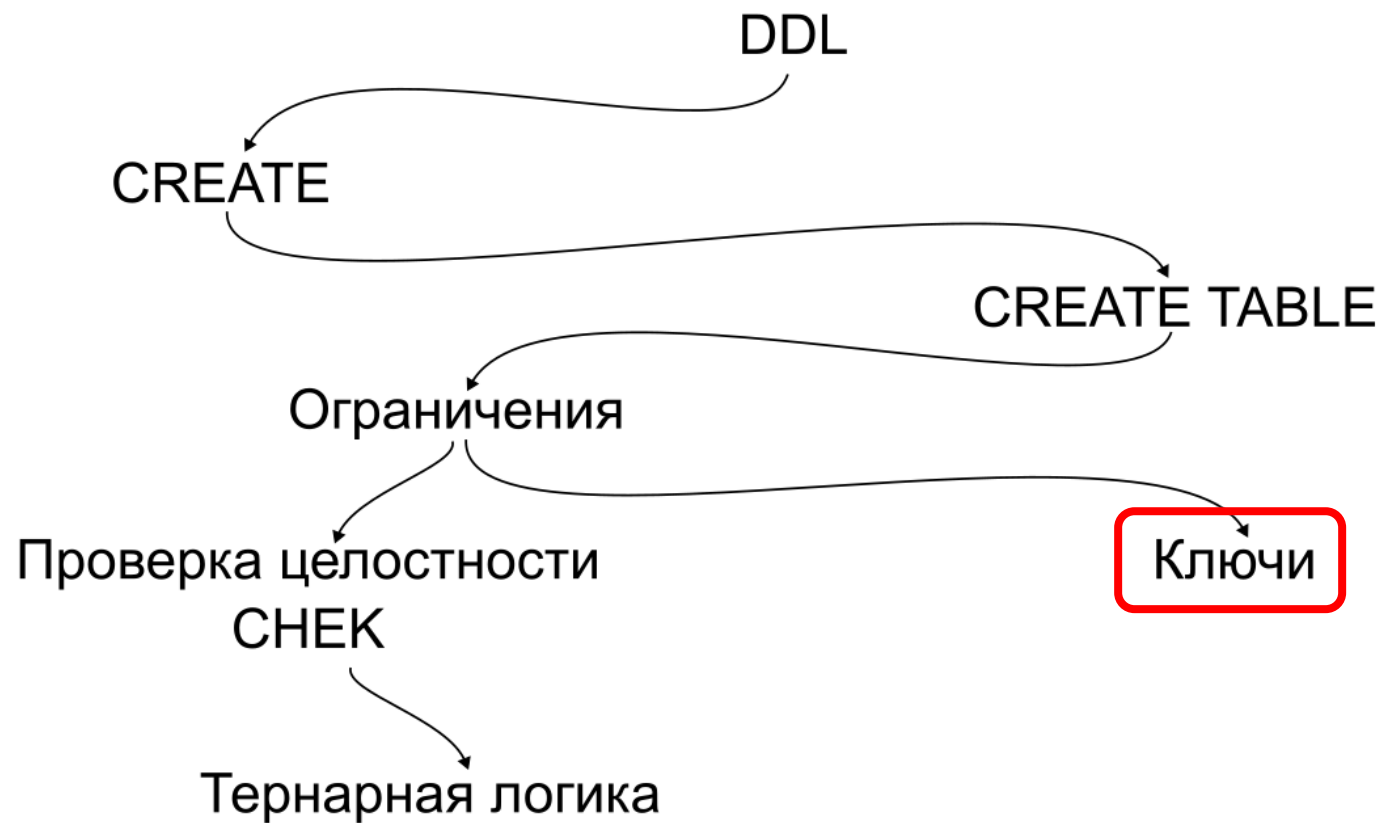
- Допустимо формулировать ограничения столбцов и ограничения таблиц
- Ограничениям можно присваивать имена с помощью команды **CONSTRAINT** (это полезно при выводе ошибок)

```
CREATE TABLE products (  
  product_no integer,  
  name text,  
  price numeric CHECK (price > 0)  
);
```

```
CREATE TABLE products (  
  product_no integer,  
  name text,  
  price numeric,  
  discounted_price numeric  
  CHECK (price > discounted_price)  
);
```

```
CREATE TABLE products (  
  product_no integer,  
  name text,  
  price numeric,  
  discounted_price numeric,  
  CONSTRAINT valid_discount CHECK (price >  
  discounted_price)  
);
```

Траектория нашего разговора



Ключи:
потенциальный,
первичный,
альтернативный,
естественный,
суррогатный,
внешний

- **Потенциальный ключ** - в реляционной модели данных это подмножество атрибутов отношения, удовлетворяющее требованиям:
 - уникальности - нет и не может быть двух кортежей данного отношения, в которых значения этого подмножества атрибутов совпадают (равны)
 - несократимости (минимальности) - в составе потенциального ключа отсутствует меньшее подмножество атрибутов, удовлетворяющее условию уникальности. Иными словами, если из потенциального ключа убрать любой атрибут, он утратит свойство уникальности
- Потенциальный ключ существует всегда, даже если он включает все атрибуты отношения (следует из свойств отношений)
- Потенциальных ключей может быть несколько

Ключи:
потенциальный,
первичный,
альтернативный,
естественный,
суррогатный,
внешний

- **Потенциальный ключ** может быть:
 - Простым – если состоит из одного атрибута
 - Составным – если состоит из двух и более атрибутов отношения

Ключи:
потенциальный,
первичный,
альтернативный,
естественный,
суррогатный,
внешний

- Первичный ключ (PRIMARY KEY) – это такой потенциальный ключ отношения, который выбран в качестве «основного»
 - Любой потенциальный ключ пригоден на роль первичного
- Альтернативный ключ (ключи) – все остальные потенциальные ключи отношения

Ключи:
потенциальный,
первичный,
альтернативный,
естественный,
суррогатный,
внешний

- Естественный ключ – ключ, основанный на существующем атрибуте отношения
- Суррогатный ключ – основан на специальном, техническом атрибуте, не имеющим содержательного значения (id). Несмотря на очевидные достоинства (неизменность, гарантированная уникальность, эффективность и т.п.) у суррогатных ключей есть и свои недостатки (неинформативность, использование вместо нормализации, уязвимость генераторов)

Ключи:
потенциальный,
первичный,
альтернативный,
естественный,
суррогатный,
внешний


- Для отражения функциональных зависимостей между кортежами разных отношений используется дублирование первичного ключа одного отношения (родительского) в другое (дочернее). Атрибуты, представляющие собой копии ключей родительских отношений, называются внешними ключами.
- Внешний ключ в отношении R2 – это непустое подмножество FK множества атрибутов этого отношения, такое, что:
 - 1) Существует отношение R1 с потенциальным ключом PK;
 - 2) Каждое значение внешнего ключа FK в текущем значении отношения R2 обязательно совпадает со значением ключа PK некоторого кортежа в текущем значении отношения R1.
- Отношения R1 и R2 необязательно различны.

Мы еще вернемся к
рассмотрению ключей в
следующих лекциях 😊



Траектория нашего разговора





Тернарная логика – для чего она в SQL?

- Если значение в ячейки таблицы (значение атрибута в кортеже) неизвестно, то оно обычно заполняется специальным маркером NULL
- NULL не является числовым или символьным типом, как его сравнивать с действительными значениями (с реальными данными?)
- Решение – тернарная логика!
- Следует различать маркер NULL, «реально» фигурирующий в ячейках таблицы, и логическое значение UNKNOWN, которое получается в результате логических операций, одним из операндов которых является NULL

Таблицы истинности для тернарной логики

NOT(A)		AND(A, B)					OR(A, B)					XOR(A, B)				
A	$\neg A$	$A \wedge B$		B			$A \vee B$		B			$A \oplus B$		B		
F	T	A	F	F	F	F	F	F	F	F	F	F	F	F	F	F
U	U		U	F	U	U	U	U	U	U	U	U	U	U	U	U
T	F		T	F	U	T	T	T	T	T	T	T	T	U	F	T

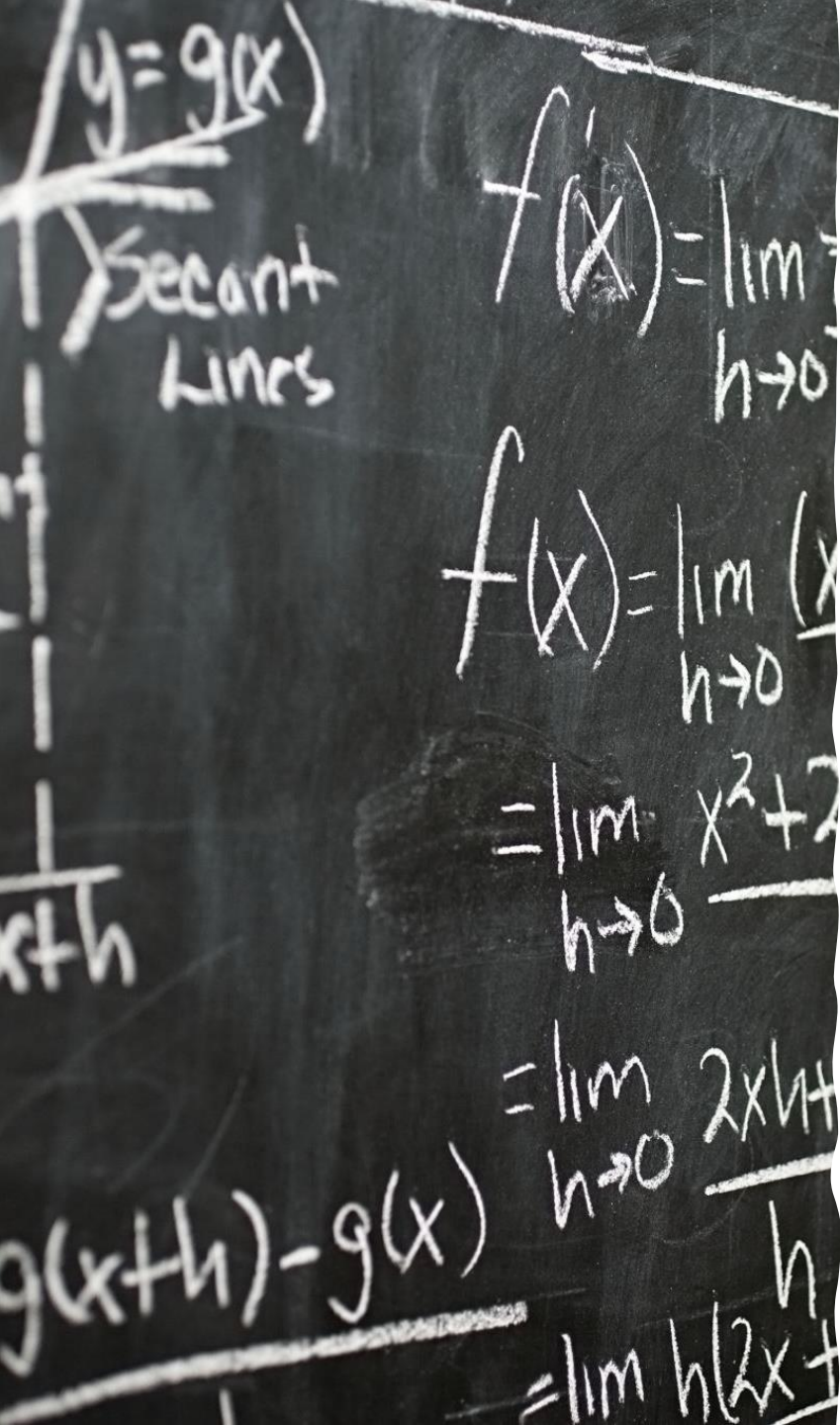
Предикаты сравнения истинности

- SQL предоставляет несколько стандартных предикатов, позволяющих провести логическое сравнение:
- IS [NOT] TRUE
- IS [NOT] FALSE
- IS [NOT] UNKNOWN

<i>p</i>	IS TRUE	IS FALSE	IS UNKNOWN		IS NOT TRUE	IS NOT FALSE	IS NOT UNKNOWN
TRUE	TRUE	FALSE	FALSE		FALSE	TRUE	TRUE
FALSE	FALSE	TRUE	FALSE		TRUE	FALSE	TRUE
UNKNOWN	FALSE	FALSE	TRUE		TRUE	TRUE	FALSE

Основное правило вывода для тернарной логики

Все сравнения с маркером NULL
приводят к значению UNKNOWN



Примеры

- $\text{NULL} = 5 \Rightarrow \text{UNKNOWN}$
- $5 = \text{NULL} \Rightarrow \text{UNKNOWN}$
- $\text{NULL} \leq 5 \Rightarrow \text{UNKNOWN}$
- $\text{col_1} = 5 \Rightarrow \text{UNKNOWN}$

(для тех строк, в которых атрибут col_1 содержит NULL)

- $\text{col_1} = \text{col_2} \Rightarrow \text{UNKNOWN}$

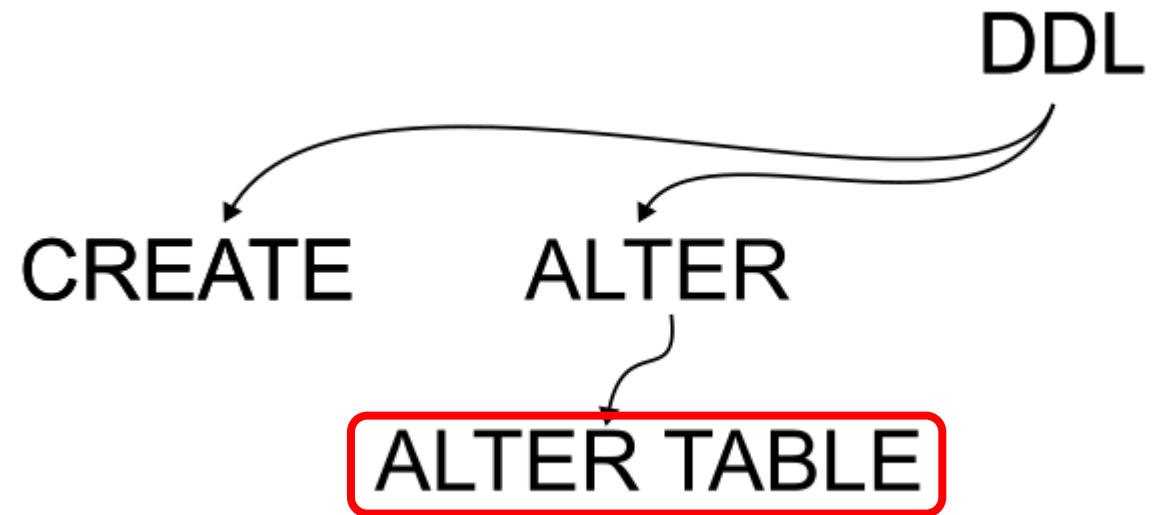
(для тех строк, в которых атрибуты col_1 И / ИЛИ col_2 содержат NULL)

- $\text{NULL} = \text{NULL} \Rightarrow \text{UNKNOWN}$

Что нужно помнить о тернарной логике при запросах

- Ключевые слова
 - IN
 - WHERE
 - HAVING
- строго требуют значение TRUE, поэтому NOT FALSE для них недостаточно, со значением UNKNOWN они работать не будут (как было показано ранее, для проверки по команде CHECK действует более слабое условие).
- Для сравнения значения с NULL нужно использовать специальный предикат:
 - IS [NOT] NULL
- Можно также использовать IS [NOT] DISTINCT FROM с типом NULL:
 - *1 IS [NOT] DISTINCT FROM NULL -> t[f]*
- При наличии сомнений – обращаться к документации!

Траектория
нашего
разговора



ALTER TABLE: базовый синтаксис

ALTER TABLE [IF EXISTS] [ONLY] имя [*]
действие [, ...]

Где *действие* может быть следующим:

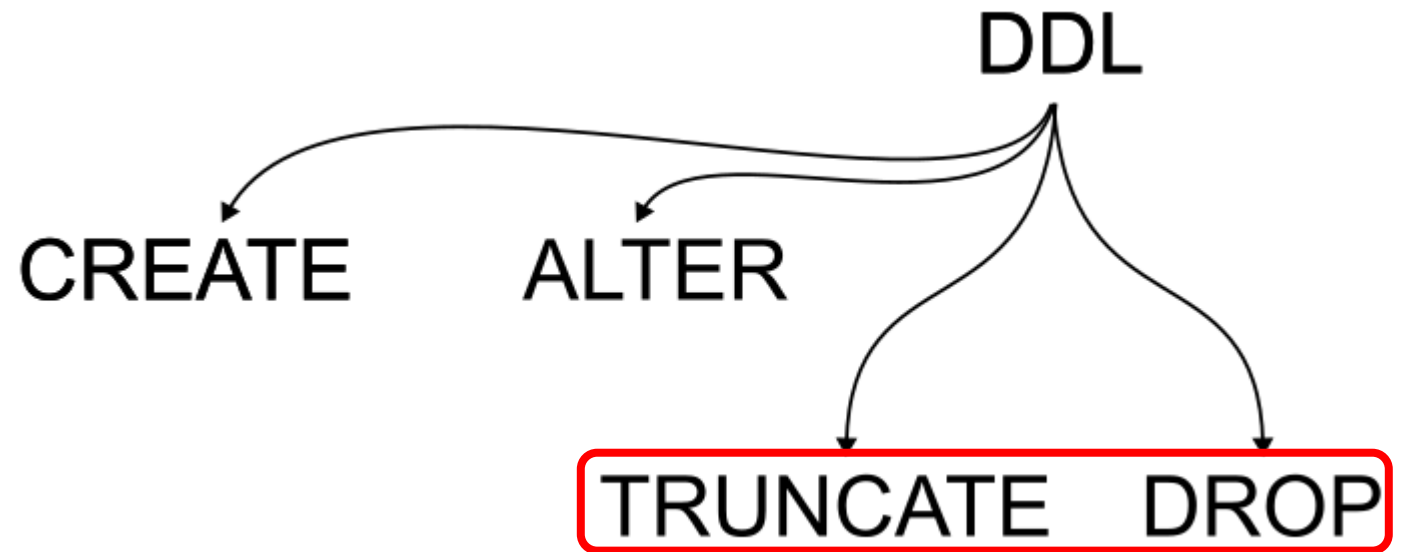
- ADD [COLUMN] [IF NOT EXISTS] имя_столбца тип_данных [COLLATE правило_сортировки] [ограничение_столбца [...]]
- DROP [COLUMN] [IF EXISTS] имя_столбца [RESTRICT | CASCADE]
- ALTER [COLUMN] имя_столбца [SET DATA] TYPE тип_данных [COLLATE правило_сортировки] [USING выражение]
- ALTER [COLUMN] имя_столбца SET DEFAULT выражение
- И др.

ALTER TABLE меняет определение существующей таблицы

ALTER TABLE: пример использования

```
CREATE TABLE PERSON (  
    ID      INTEGER,  
    LAST_NAME VARCHAR(255),  
    FIRST_NAME VARCHAR(255) NOT NULL,  
    AGE     INTEGER,  
    CONSTRAINT PK_Person PRIMARY KEY (ID, LAST_NAME)  
);  
ALTER TABLE PERSON  
ADD CONSTRAINT PK_Person PRIMARY KEY (ID, LAST_NAME);  
ALTER TABLE PERSON  
DROP CONSTRAINT PK_Person;
```

Траектория
нашего
разговора



TRUNCATE: базовый синтаксис

TRUNCATE [TABLE] [ONLY] имя [*] [, ...]

[RESTART IDENTITY | CONTINUE IDENTITY] [CASCADE | RESTRICT]

- Команда TRUNCATE быстро удаляет все строки из набора таблиц. Она действует так же, как безусловная команда DELETE для каждой таблицы, но гораздо быстрее, так как она фактически не сканирует таблицы. Более того, она немедленно высвобождает дисковое пространство, так что выполнять операцию VACUUM после неё не требуется. Наиболее полезна она для больших таблиц.
- ВАЖНО – сама таблица остается в БД

DROP TABLE: базовый синтаксис

`DROP TABLE [IF EXISTS] имя [, ...] [CASCADE | RESTRICT]`

- DROP TABLE удаляет таблицы из базы данных. Удалить таблицу может только её владелец, владелец схемы или суперпользователь. Чтобы опустошить таблицу, не удаляя её саму, вместо этой команды следует использовать DELETE или TRUNCATE.
- DROP TABLE всегда удаляет все индексы, правила, триггеры и ограничения, существующие в целевой таблице. Однако чтобы удалить таблицу, на которую ссылается представление или ограничение внешнего ключа в другой таблице, необходимо дополнительно указать CASCADE. (С указанием CASCADE зависимое представление удаляется полностью, тогда как в случае с ограничением внешнего ключа удаляется именно это ограничение, а не вся таблица, к которой оно относится.)

III. Основы DML

Команды DML

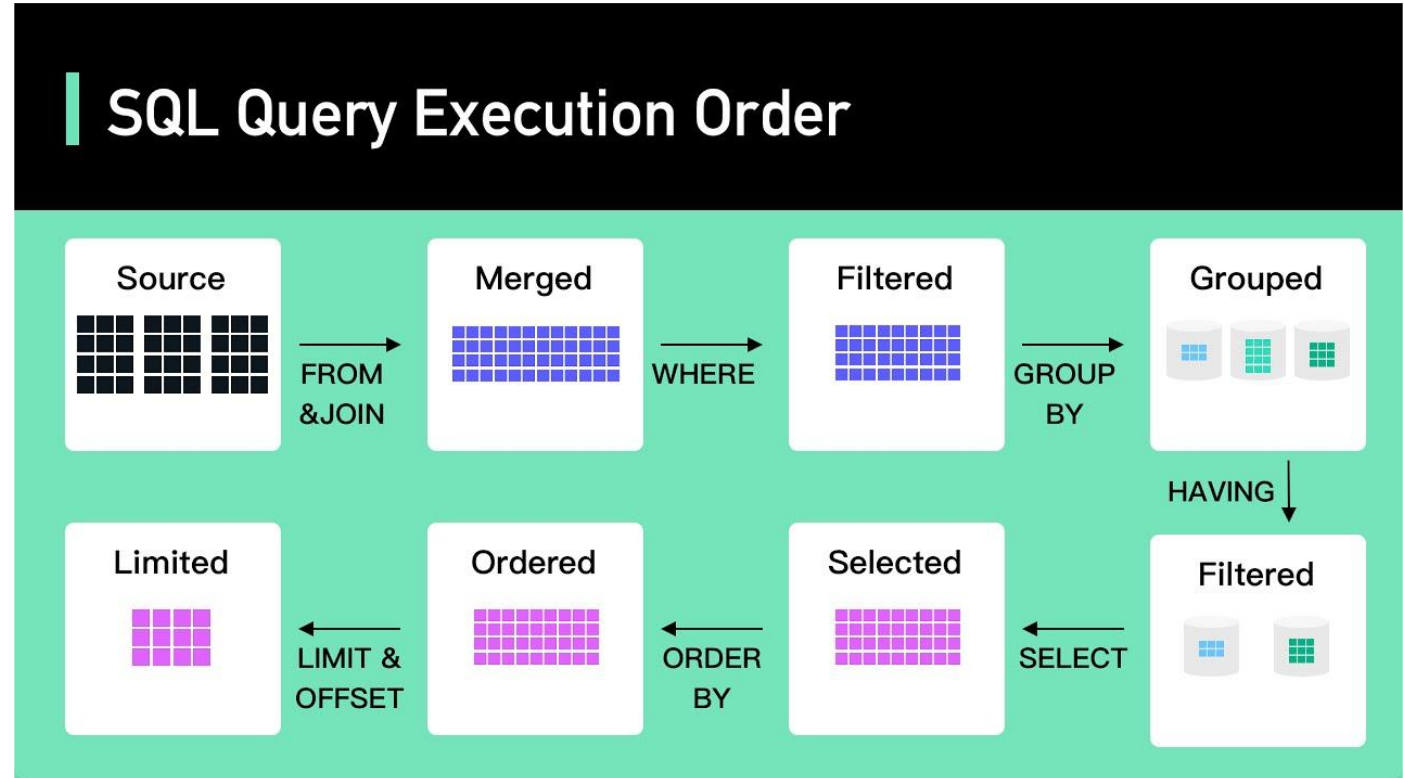
- ▶ **SELECT**
- ▶ **INSERT**
- ▶ **UPDATE**
- ▶ **DELETE**

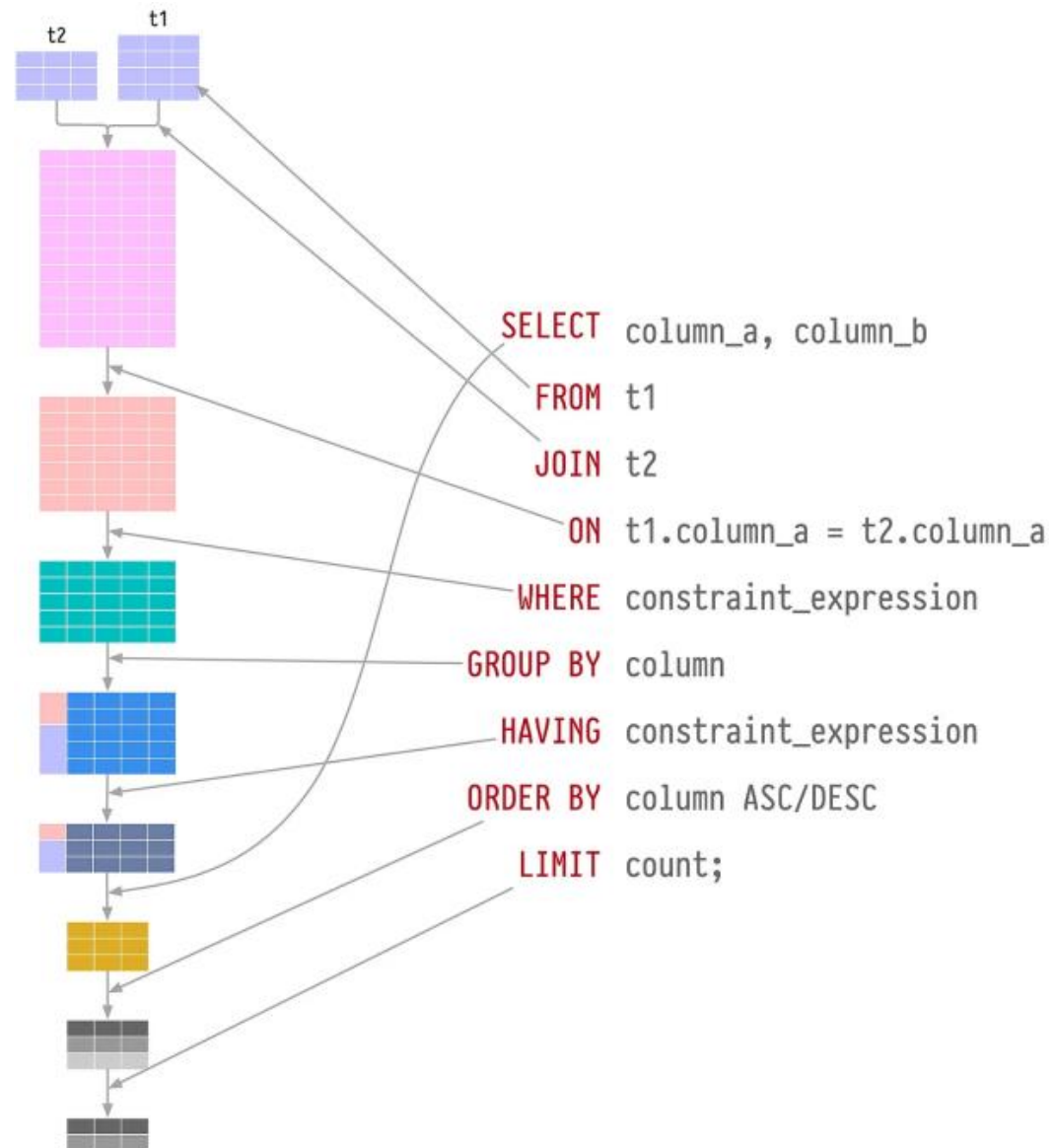
SQL: порядок выполнения запросов

- SQL – декларативный язык, в нем вы описываете то, что вы хотите получить, а не то, как это сделать
- Хотя разработчики языка пытались максимально снизить «порог входа» для начинающих работать с SQL, запросы на SQL не тривиальны
- Процесс или команда получения данных из базы данных называется запросом. В SQL запросы формулируются с помощью команды SELECT
- Разберемся, как работает SELECT-запрос

Этапы выполнения запроса SELECT

1. FROM / JOIN
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT / OFFSET





Пример работы запроса SELECT

- Мы хотим знать название только двух городов, кроме Сан-Бруно, в которых проживает два или более жителей.
- Мы также хотим получить результат, упорядоченный по алфавиту

CITIZEN	
Name	char(20)
City_id	integer

CITY	
City_id	integer
City_name	char(50)



CITIZEN	
Name	City_id
Andre	3
Rachel	2
Jenny	1
Dough	3
Kevin	1
Sarah	2
Trevor	3
Al	1
Yung	1

CITY	
City_id	City_name
1	Palo Alto
2	Sunnyvale
3	San Bruno

Пример работы запроса SELECT

```
SELECT city.city_name AS "City"  
FROM citizen  
JOIN city  
ON citizen.city_id = city.city_id  
WHERE city.city_name != 'San Bruno'  
GROUP BY city.city_name  
HAVING COUNT(*) >= 2  
ORDER BY city.city_name ASC  
LIMIT 2
```

CITIZEN	
Name	City_id
Andre	3
Rachel	2
Jenny	1
Dough	3
Kevin	1
Sarah	2
Trevor	3
Al	1
Yung	1

CITY	
City_id	City_name
1	Palo Alto
2	Sunnyvale
3	San Bruno

Пример работы запроса SELECT

```
SELECT city.city_name AS "City"  
FROM citizen  
JOIN city  
ON citizen.city_id = city.city_id  
WHERE city.city_name != 'San Bruno'  
GROUP BY city.city_name  
HAVING COUNT(*) >= 2  
ORDER BY city.city_name ASC  
LIMIT 2
```

Name	City_id	City_id	City_name
Andre	3	1	Palo Alto
Andre	3	2	Sunnyvale
Andre	3	3	San Bruno
Rachel	2	1	Palo Alto
Rachel	2	2	Sunnyvale
Rachel	2	3	San Bruno
Jenny	1	1	Palo Alto
Jenny	1	2	Sunnyvale
Jenny	1	3	San Bruno
Dough	3	1	Palo Alto
Dough	3	2	Sunnyvale
Dough	3	3	San Bruno
Kevin	1	1	Palo Alto
Kevin	1	2	Sunnyvale
Kevin	1	3	San Bruno
Sarah	2	1	Palo Alto
Sarah	2	2	Sunnyvale
Sarah	2	3	San Bruno
Trevor	3	1	Palo Alto
Trevor	3	2	Sunnyvale
Trevor	3	3	San Bruno
Al	1	1	Palo Alto
Al	1	2	Sunnyvale
Al	1	3	San Bruno
Yung	1	1	Palo Alto
Yung	1	2	Sunnyvale
Yung	1	3	San Bruno

Пример работы запроса SELECT

```
SELECT city.city_name AS "City"  
FROM citizen  
JOIN city  
ON citizen.city_id = city.city_id  
WHERE city.city_name != 'San Bruno'  
GROUP BY city.city_name  
HAVING COUNT(*) >= 2  
ORDER BY city.city_name ASC  
LIMIT 2
```

Name	City_id	City_name
Andre	3	San Bruno
Rachel	2	Sunnyvale
Jenny	1	Palo Alto
Dough	3	San Bruno
Kevin	1	Palo Alto
Sarah	2	Sunnyvale
Trevor	3	San Bruno
Al	1	Palo Alto
Yung	1	Palo Alto

Пример работы запроса SELECT

```
SELECT city.city_name AS "City"  
FROM citizen  
JOIN city  
ON citizen.city_id = city.city_id  
WHERE city.city_name != 'San Bruno'  
GROUP BY city.city_name  
HAVING COUNT(*) >= 2  
ORDER BY city.city_name ASC  
LIMIT 2
```

Name	City_id	City_name
Rachel	2	Sunnyvale
Jenny	1	Palo Alto
Kevin	1	Palo Alto
Sarah	2	Sunnyvale
Al	1	Palo Alto
Yung	1	Palo Alto

Пример работы запроса SELECT

```
SELECT city.city_name AS "City"  
FROM citizen  
JOIN city  
ON citizen.city_id = city.city_id  
WHERE city.city_name != 'San Bruno'  
GROUP BY city.city_name  
HAVING COUNT(*) >= 2  
ORDER BY city.city_name ASC  
LIMIT 2
```

Name	City_id	City_name
Rachel	2	Sunnyvale
Sarah	2	
Jenny	1	Palo Alto
Kevin	1	
Al	1	
Yung	1	

Пример работы запроса SELECT

```
SELECT city.city_name AS "City"  
FROM citizen  
JOIN city  
ON citizen.city_id = city.city_id  
WHERE city.city_name != 'San Bruno'  
GROUP BY city.city_name  
HAVING COUNT(*) >= 2  
ORDER BY city.city_name ASC  
LIMIT 2
```

Name	City_id	City_name
Rachel	2	Sunnyvale
Sarah	2	
Jenny	1	Palo Alto
Kevin	1	
Al	1	
Yung	1	

Пример работы запроса SELECT

```
SELECT city.city_name AS "City"  
FROM citizen  
JOIN city  
ON citizen.city_id = city.city_id  
WHERE city.city_name != 'San Bruno'  
GROUP BY city.city_name  
HAVING COUNT(*) >= 2  
ORDER BY city.city_name ASC  
LIMIT 2
```

City
Sunnyvale
Palo Alto

Пример работы запроса SELECT

```
SELECT city.city_name AS "City"  
FROM citizen  
JOIN city  
ON citizen.city_id = city.city_id  
WHERE city.city_name != 'San Bruno'  
GROUP BY city.city_name  
HAVING COUNT(*) >= 2  
ORDER BY city.city_name ASC  
LIMIT 2
```

City
Palo Alto
Sunnyvale