

# УЖЕ ЗНАЕМ:

## Базы данных:

- Данные хранятся по заранее определённым правилам (схема данных)
- Работа с данными по заранее определённым правилам

## Реляционная модель данных:

- Логическая модель данных, не зависящая от физических структур
- В основе – математика и логика
- Реляционная алгебра

# УЖЕ ЗНАЕМ:

- Реляционная алгебра:
  - Ключевым является понятие отношения:
    - Нет 2 одинаковых кортежей
    - Порядок кортежей не определен
    - Порядок атрибутов в заголовке не определен
  - Арность отношения – количество атрибутов
  - Заголовок отношения – список атрибутов
  - Домен атрибута – множество допустимых значений
  - Тело отношения – множество кортежей, входящих в его состав

# УЖЕ ЗНАЕМ:

- Операции реляционной алгебры:
  - Теоретико-множественные:
    - Объединение
    - Пересечение
    - Разность
  - Специальные реляционные:
    - Проекция
    - Ограничение
    - Соединение
    - Деление

YEAH... SURE




WE KNOW

# РЕЛЯЦИОННЫЕ БАЗЫ

- В основе – реляционная модель данных
  - Таблица  $\approx$  Отношение
  - Заголовок отношения  $\approx$  Список наименований колонок таблицы
  - Кортеж  $\approx$  Строка таблицы
  - Тело отношения  $\approx$  Все строки таблицы
- Средство манипуляции – реляционные системы управления базами данных (СУБД)
- Способ манипуляции – специальный язык запросов

# STRUCTURED QUERY LANGUAGE (SQL)

- Предметно-ориентированный язык (Domain-specific language)
- Используется для работы с реляционными БД
- Управление большим количеством информации одним запросом
- Не нужно указывать, **как** получаем запись



# ИСТОРИЯ РАЗВИТИЯ SQL

- Дональд Чэмбэрлин и Рэй Бойс, IBM:
  - Square: (*Specifying Queries As Relational Expressions*)
  - SEQUEL (*Structured English QUery Language*), 1973-1974
  - Пэт Селинджер – cost-based optimizer
  - Рэймонд Лори – компилятор запросов
- Позднее SEQUEL -> SQL
- Калифорнийский университет Беркли:
  - QUEL – не выдержал конкуренции с SQL

# СТАНДАРТИЗАЦИЯ ЯЗЫКА SQL

## Предпосылки:

- Разное ПО от разных производителей
- Собственная реализация языка запросов

## Хотели получить:

- Переносимость ПО

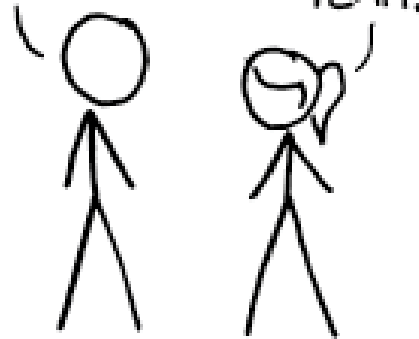
## Получили:

- Частичная переносимость

## HOW STANDARDS PROLIFERATE: (SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION:  
THERE ARE  
14 COMPETING  
STANDARDS.

14?! RIDICULOUS!  
WE NEED TO DEVELOP  
ONE UNIVERSAL STANDARD  
THAT COVERS EVERYONE'S  
USE CASES.



SOON:

SITUATION:  
THERE ARE  
15 COMPETING  
STANDARDS.



# СТАНДАРТЫ ЯЗЫКА SQL

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, нескаларные 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

# ПРОБЛЕМЫ СТАНДАРТОВ

Core-раздел стандарта (введен в 1992)

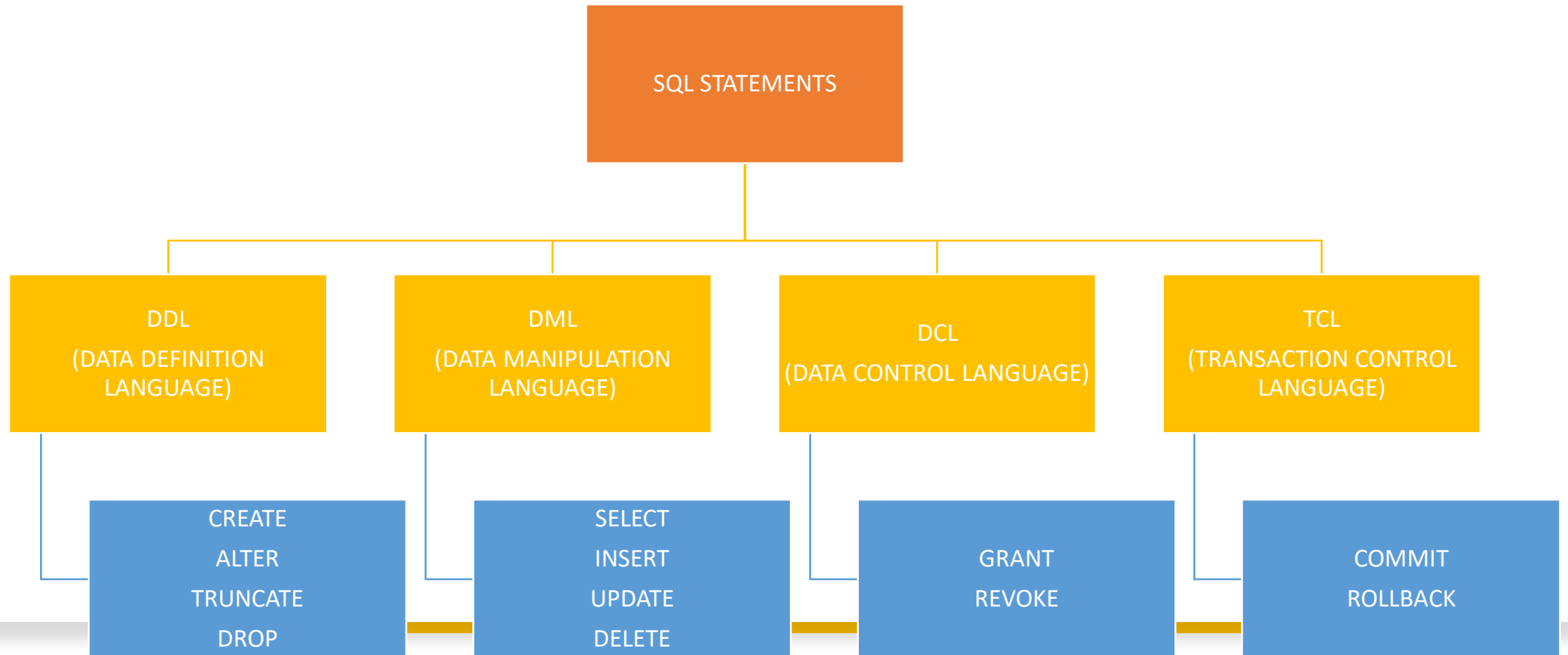
Производители обеспечивают соответствие только Core

Различия в реализации

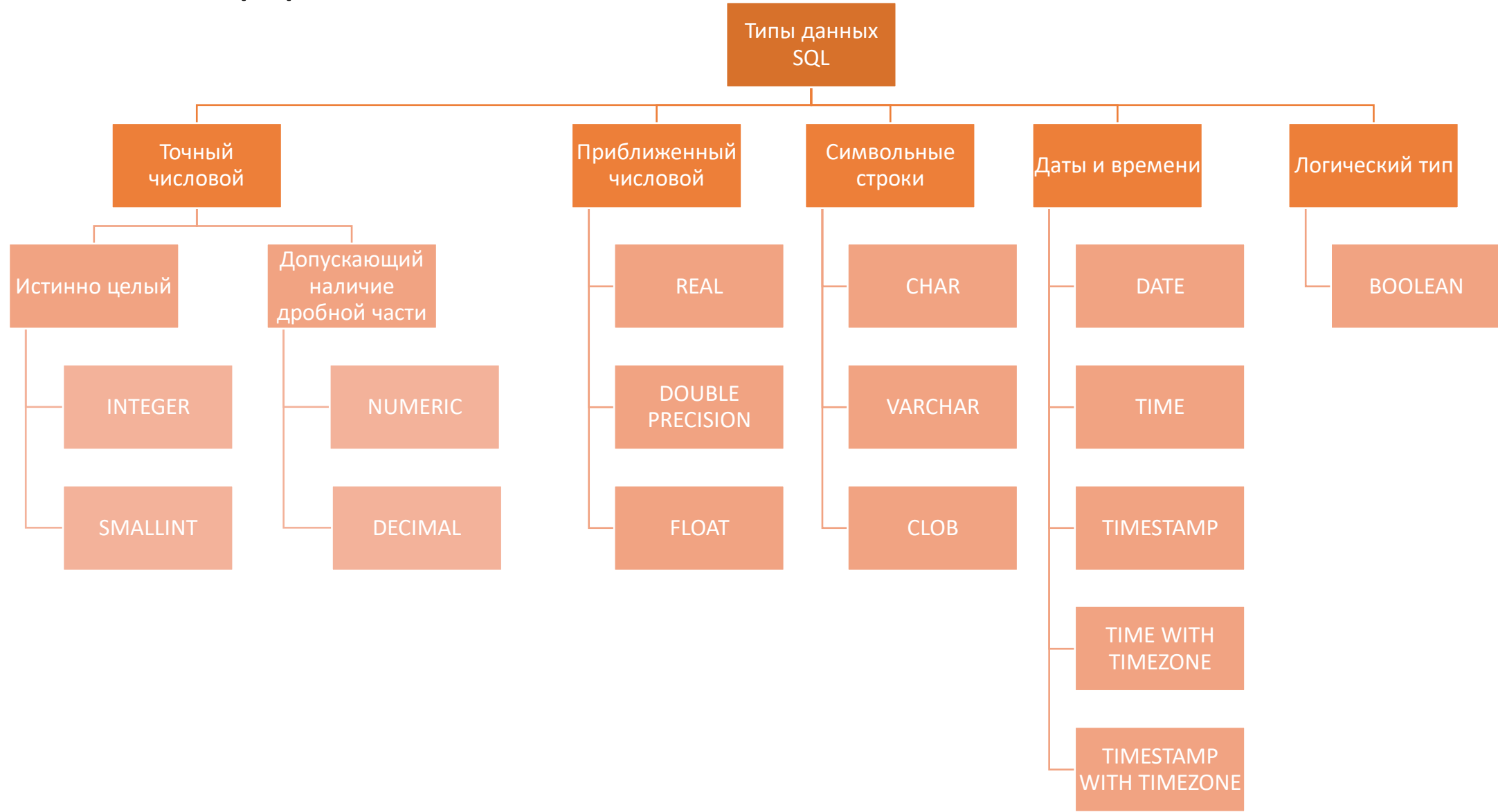
Различия в синтаксисе

Различия в логике

# ОПЕРАТОРЫ SQL



# ТИПЫ ДАННЫХ SQL



# ОПЕРАТОРЫ ОПРЕДЕЛЕНИЯ ДАННЫХ (DDL)

CREATE

- Операция создания объектов БД

ALTER

- Оператор модификации объектов БД

DROP

- Оператор удаления объектов БД

TRUNCATE

- Оператор удаления содержимого объекта БД



# DDL: CREATE

**CREATE:** создание объекта в базе

```
CREATE TABLE table_name(  
    column_name_1 datatype_1,  
    ...  
    column_name_N datatype_N  
);
```

---

# DDL: CREATE

```
CREATE TABLE SUPERHERO (  
    NAME          VARCHAR(100) ,  
    AGE           INTEGER,  
    BIRTH_DT      DATE,  
    ALTER_EGO     VARCHAR(50)  
);
```

# DDL: CREATE



NAME	AGE	BIRTH_DT	ALTER_EGO
------	-----	----------	-----------



# DDL: ALTER

**ALTER:** внесение изменений в объекты базы

```
ALTER TABLE table_name ADD column_name datatype;
```

```
ALTER TABLE table_name DROP column_name;
```

```
ALTER TABLE table_name RENAME column_name TO new_column_name;
```

```
ALTER TABLE table_name ALTER column_name TYPE datatype;
```

*Реальных кейсов применения больше!*

---

# DDL: ALTER

1. **ALTER TABLE** SUPERHERO **ADD** RATING **INTEGER**;
2. **ALTER TABLE** SUPERHERO **DROP COLUMN** AGE;

# DDL: ALTER

0	NAME	AGE	BIRTH_DT	ALTER_EGO
---	------	-----	----------	-----------

**ALTER TABLE** SUPERHERO **ADD** RATING **INTEGER**;

1	NAME	AGE	BIRTH_DT	ALTER_EGO	RATING
---	------	-----	----------	-----------	--------

**ALTER TABLE** SUPERHERO **DROP COLUMN** AGE;

2	NAME	BIRTH_DT	ALTER_EGO	RATING
---	------	----------	-----------	--------

# DDL: TRUNCATE

## **TRUNCATE :**

- Физическое удаление данных из объекта единым куском
- Данные нельзя удалять частично или по условию

**TRUNCATE TABLE** table\_name;

# DDL: TRUNCATE

A thick yellow horizontal bar spans the width of the slide, with a vertical yellow bar extending downwards from its right end.

```
TRUNCATE TABLE SUPERHERO;
```

# DDL: TRUNCATE

NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	100
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90

**TRUNCATE TABLE** SUPERHERO;

NAME	BIRTH_DATE	ALTER_EGO	RATING
------	------------	-----------	--------

# DDL: DROP

**DROP:** удаление объекта из базы

```
DROP TABLE [IF EXISTS] table_name;
```

# DDL: DROP

A thick yellow horizontal bar spans the width of the slide, with a vertical yellow bar extending downwards from its right end.

```
DROP TABLE SUPERHERO;
```



# DDL: DROP

NAME	AGE	BIRTH_DT	ALTER_EGO
------	-----	----------	-----------



# ОПЕРАТОРЫ МАНИПУЛЯЦИИ ДАННЫМИ (DML)

SELECT

- Выборка данных, удовлетворяющих заданным условиям

INSERT

- Добавление новых данных

UPDATE

- Изменение существующих данных

DELETE

- Удаление существующих данных

# DML: INSERT



## **INSERT**

```
INTO table_name [(comma_separated_column_names)]  
VALUES (comma_separated_values);
```

# DML: INSERT

**INSERT**

**INTO** SUPERHERO (NAME, BIRTH\_DT, ALTER\_EGO, RATING)

**VALUES** ( 'Natasha Romanoff', '01-AUG-1999', 'Black Widow', 59 );

# DML: INSERT

NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	100
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1999	Black Widow	59

# DML: UPDATE



```
UPDATE table_name  
    SET update_assignment_comma_list  
[WHERE conditional_expression];
```

# DML: UPDATE

```
UPDATE SUPERHERO  
    SET BIRTH_DT = '01-AUG-1940'  
    WHERE NAME = 'Natasha Romanoff';
```

# DML: UPDATE

NAME	BIRTH_DATE	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	100
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1940	Black Widow	59

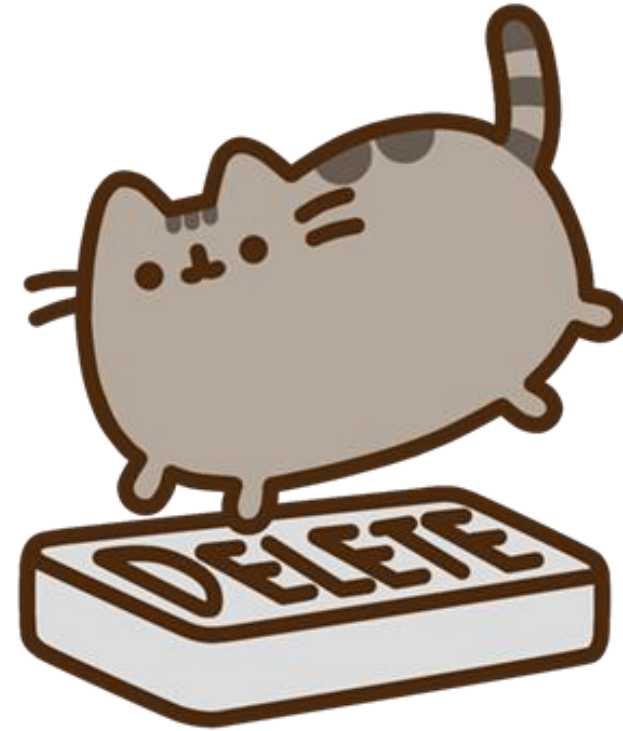


# DML: DELETE

**DELETE**

**FROM** table\_name

[**WHERE** conditional\_expression];



# DML: DELETE vs DDL:TRUNCATE

DELETE	TRUNCATE
«Удаление» построчное	Удаление всего блока разом
Можно задать условия для удаления	Условия для удаления задать нельзя
Возможность отката изменений	Возможности отката изменений нет
Физически строки не удаляются, только отмечаются «невидимыми» с определенного момента Для удаления нужен VACUUM	Удаление данных и высвобождение дискового пространства происходит сразу

# DML: DELETE

```
DELETE  
  FROM SUPERHERO  
  WHERE NAME = 'Bruce Banner';
```

# DML: DELETE

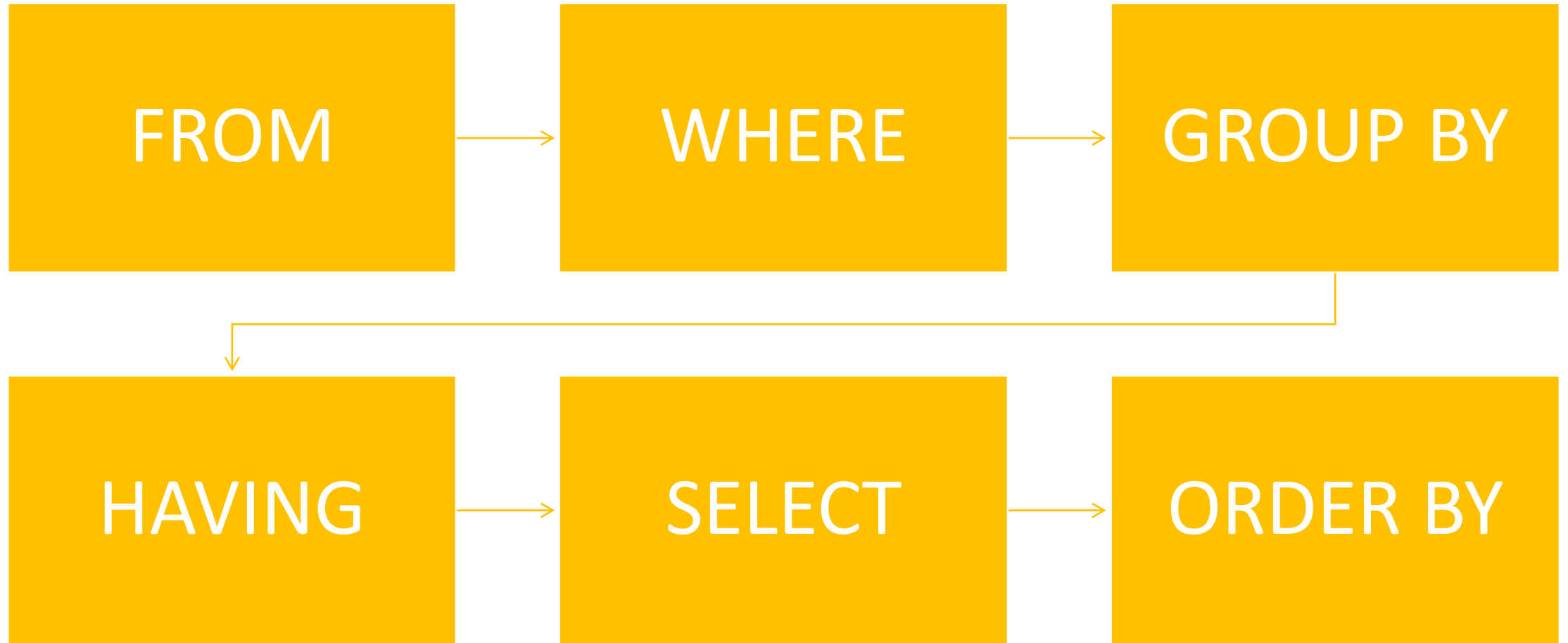
NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	100
<del>Bruce Banner</del>	<del>28-FEB-1969</del>	<del>Hulk</del>	<del>80</del>
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1940	Black Widow	59

# DML: SELECT



```
SELECT [DISTINCT] select_item_comma_list
FROM table_reference_comma_list
[WHERE conditional_expression]
[GROUP BY column_name_comma_list]
[HAVING conditional_expression]
[ORDER BY order_item_comma_list];
```

# ПОРЯДОК ВЫПОЛНЕНИЯ ЗАПРОСА



# SELECT: FROM

```
SELECT ALTER_EGO  
      FROM SUPERHERO;
```

Декартово произведение:

```
SELECT NAME, ALTER_EGO, COMICS_N  
      FROM SUPERHERO, COMICS;
```

# DML: SELECT

NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	98
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1940	Black Widow	59
Thor	13-FEB-1976	Thor	74
Clint Barton	17-DEC-1969	Hawkeye	55
Wanda Maximoff	22-OCT-1974	Scarlet Witch	81
Pietro Maximoff	22-OCT-1974	Quicksilver	82
Charles Xavier	30-JUN-1933	Professor X	100
Jean Grey	12-SEP-1961	Phoenix	93
Wade Wilson	13-APR-1980	Deadpool	89
James Howlett	01-JAN-1887	Wolverine	99



# SELECT: WHERE

```
SELECT NAME, ALTER_EGO  
FROM SUPERHERO  
WHERE RATING > 90;
```

NAME	ALTER_EGO
Tony Stark	Iron Man
Charles Xavier	Professor X
Jean Grey	Phoenix
James Howlett	Wolverine

```
SELECT NAME, ALTER_EGO  
FROM SUPERHERO  
WHERE RATING < 50;
```

NAME	ALTER_EGO
------	-----------

# WHERE: AND, OR, NOT

- **WHERE** X = value\_1 **AND** X <> value\_2;
  - **WHERE** X = value\_1 **OR** X <> value\_2;
  - **WHERE** X = value\_1 **AND NOT** X < value\_3;
  - **WHERE** X < value\_1 **AND** X > value\_2 **OR** X = value\_3
  - Приоритет операций: NOT, AND, OR
-

# WHERE: СПЕЦИАЛЬНЫЕ ФУНКЦИИ

Функции используются для типов данных, для которых определен результат сравнения

- **WHERE** X **BETWEEN** *value\_1* **AND** *value\_2*:
    - (X >= *value\_1* **AND** X <= *value\_2*) **OR** (X >= *value\_2* **AND** X <= *value\_1*)
  - **WHERE** X **IN** (*value\_1*, *value\_2*, ..., *value\_N*):
    - X = *value\_1* **OR** X = *value\_2* **OR** ... **OR** X = *value\_N*
-

# WHERE: СПЕЦИАЛЬНЫЕ ФУНКЦИИ

- **LIKE:**

- Находит строки определенных форматов
- % – несколько символов
- \_ – ровно 1 символ

```
SELECT column_1, column_2, ...  
    FROM table_name  
    WHERE column_N LIKE pattern;
```

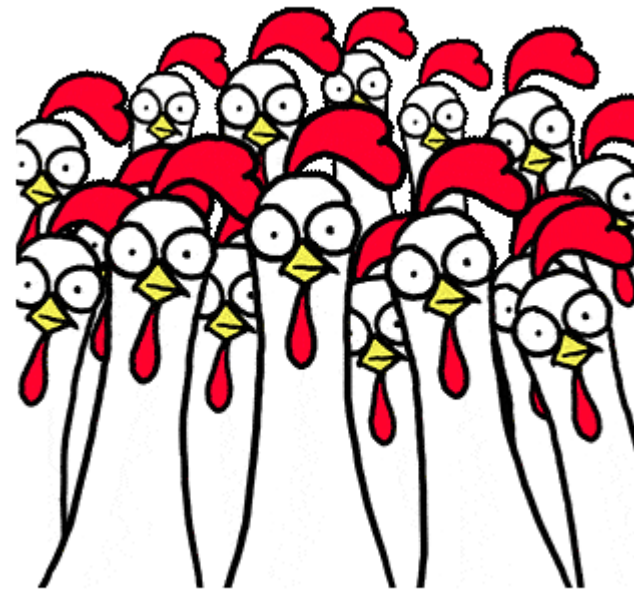
---

# SELECT: GROUP BY

Результат выполнения: таблица с нужной группировкой

```
SELECT DATE_BIRTH  
        , RATING  
        , NAME  
FROM SUPERHERO  
WHERE RATING < 90  
        AND RATING > 70  
GROUP BY DATE_BIRTH;
```

Пример неработающего кода



# DML: GROUP BY

NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	98
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1940	Black Widow	59
Thor	13-FEB-1976	Thor	74
Clint Barton	17-DEC-1969	Hawkeye	55
Wanda Maximoff	22-OCT-1974	Scarlet Witch	81
Pietro Maximoff	22-OCT-1974	Quicksilver	82
Charles Xavier	30-JUN-1933	Professor X	100
Jean Grey	12-SEP-1961	Phoenix	93
Wade Wilson	13-APR-1980	Deadpool	89
James Howlett	01-JAN-1887	Wolverine	99

# АГРЕГИРУЮЩИЕ ФУНКЦИИ

COUNT

- Определяет количество строк в результирующей таблице

MAX

- Определяет наибольшее из всех выбранных значений данного поля

MIN

- Определяет наименьшее из всех выбранных значений данного поля

SUM

- Определяет сумму всех выбранных значений данного поля

AVG

- Определяет среднее для всех выбранных значений данного поля

# АГРЕГИРУЮЩИЕ ФУНКЦИИ

```
SELECT  count (ALTER_EGO)  
         FROM  SUPERHERO;
```



# АГРЕГИРУЮЩИЕ ФУНКЦИИ

NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	98
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1940	Black Widow	59
Thor	13-FEB-1976	Thor	74
Clint Barton	17-DEC-1969	Hawkeye	55
Wanda Maximoff	22-OCT-1974	Scarlet Witch	81
Pietro Maximoff	22-OCT-1974	Quicksilver	82
Charles Xavier	30-JUN-1933	Professor X	100
Jean Grey	12-SEP-1961	Phoenix	93
Wade Wilson	13-APR-1980	Deadpool	89
James Howlett	01-JAN-1887	Wolverine	99

# АГРЕГИРУЮЩИЕ ФУНКЦИИ

```
SELECT  count (ALTER_EGO)  
FROM  SUPERHERO;
```

COUNT (ALTER_EGO)
12

# SELECT: GROUP BY

```
SELECT BIRTH_DT, count (ALTER_EGO)
FROM SUPERHERO
WHERE BIRTH_DT = '22-OCT-1974'
GROUP BY BIRTH_DT;
```


# SELECT: GROUP BY

NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	98
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1940	Black Widow	59
Thor	13-FEB-1976	Thor	74
Clint Barton	17-DEC-1969	Hawkeye	55
Wanda Maximoff	22-OCT-1974	Scarlet Witch	81
Pietro Maximoff	22-OCT-1974	Quicksilver	82
Charles Xavier	30-JUN-1933	Professor X	100
Jean Grey	12-SEP-1961	Phoenix	93
Wade Wilson	13-APR-1980	Deadpool	89
James Howlett	01-JAN-1887	Wolverine	99

# SELECT: GROUP BY

```
SELECT BIRTH_DT,  
        count(ALTER_EGO)  
FROM SUPERHERO  
WHERE BIRTH_DT = '22-OCT-1974'  
GROUP BY BIRTH_DT;
```

BIRTH_DT	COUNT (ALTER_EGO)
22-OCT-1974	2



# SELECT: HAVING

- Используется в связке с GROUP BY для наложения ограничений на выборку уже **после** группировки
- Ограничение с использованием WHERE накладывать можно только **до** группировки

```
GROUP BY column_name(s)  
HAVING expression_clause
```

---

# ТИПИЧНЫЕ ЗАПРОСЫ

```
SELECT ITEM,  
        avg(PRICE)  
FROM CATALOG  
GROUP BY ITEM;
```

ITEM	AVG (PRICE)
Computer	1035.67
Laptop	1000
Mobile phone	200
Printer	300
Scanner	200
Camera	525
Headphones	200

# ТИПИЧНЫЕ ЗАПРОСЫ

```
SELECT ITEM, avg(PRICE)
FROM CATALOG
GROUP BY ITEM
HAVING avg(PRICE) <= 500;
```

ITEM	AVG (PRICE)
Mobile phone	200
Printer	300
Scanner	200
Headphones	200



# ТИПИЧНЫЕ ЗАПРОСЫ

```
SELECT ITEM, min(PRICE)
FROM CATALOG
GROUP BY ITEM
HAVING min(PRICE) <= 500;
```

ITEM	MIN (PRICE)
Mobile phone	150
Printer	300
Scanner	200
Headphones	200
Camera	500



# SELECT: ORDER BY

```
SELECT ID, ITEM, MAGAZINE, PRICE, DELIVERY  
  FROM CATALOG  
  ORDER BY PRICE ASC;
```

ASC – по возрастанию

DESC – по убыванию

---

# SELECT: ORDER BY

ID	ITEM	MAGAZINE	PRICE	DELIVERY
5	Mobile phone	1	150	15
9	Scanner	2	200	7
12	Headphones	4	200	0
7	Mobile phone	3	250	10
8	Printer	1	300	15
10	Camera	2	500	5
11	Camera	3	550	10
4	Laptop	1	999	15
1	Computer	1	1000	15
6	Laptop	2	1001	5
2	Computer	2	1007	5
3	Computer	3	1100	10

# SELECT: ORDER BY

```
SELECT MAGAZINE,  
        count(ITEM)  
FROM CATALOG  
WHERE DELIVERY < 15  
GROUP BY MAGAZINE  
HAVING count(ITEM) > 1  
ORDER BY count(ITEM);
```



# SELECT: ORDER BY

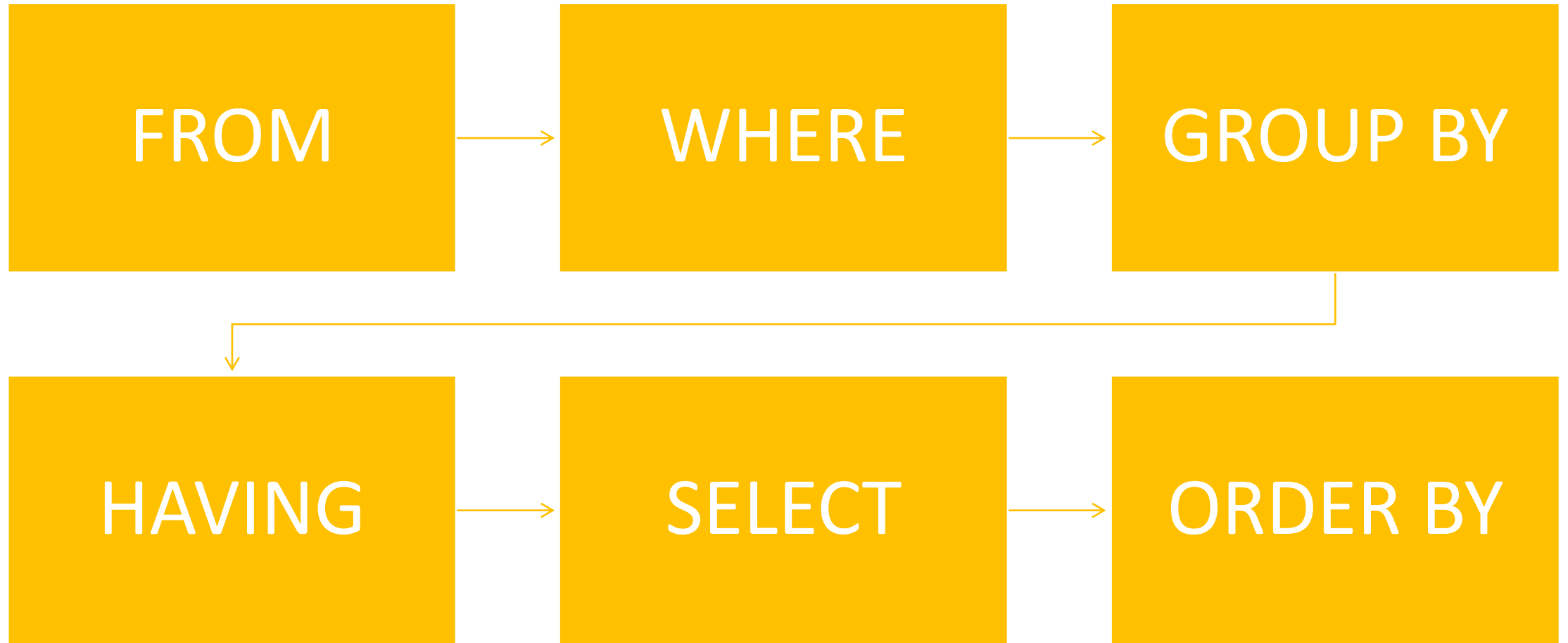
ID	ITEM	MAGAZINE	PRICE	DELIVERY
5	Mobile phone	1	150	15
9	Scanner	2	200	7
12	Headphones	4	200	0
7	Mobile phone	3	250	10
8	Printer	1	300	15
10	Camera	2	500	5
11	Camera	3	550	10
4	Laptop	1	999	15
1	Computer	1	1000	15
6	Laptop	2	1001	5
2	Computer	2	1007	5
3	Computer	3	1100	10

# SELECT: ORDER BY

```
SELECT MAGAZINE,  
        count(ITEM)  
FROM CATALOG  
WHERE DELIVERY < 15  
GROUP BY MAGAZINE  
HAVING count(ITEM) > 1  
ORDER BY count(ITEM);
```

MAGAZINE	COUNT (ITEM)
3	3
2	4

# ПОРЯДОК ВЫПОЛНЕНИЯ ЗАПРОСА



# SELECT: DISTINCT

Ключевое слово DISTINCT:

```
SELECT DISTINCT  
    ITEM  
FROM CATALOG;
```

## ITEM

Computer

Laptop

Mobile phone

Printer

Scanner

Camera

Headphones



# SELECT: \*

```
SELECT *  
  FROM CATALOG  
 WHERE ITEM = 'Scanner';
```

ID	ITEM	MAGAZINE	PRICE	DELIVERY
9	Scanner	2	200	7

# SELECT: ALIAS

```
SELECT column_name AS alias_column_name  
    FROM table_name alias_table_name;
```

# SELECT: ALIAS

```
SELECT MAGAZINE,  
        count(ITEM) AS cnt_items  
FROM CATALOG  
WHERE DELIVERY < 15  
GROUP BY MAGAZINE  
HAVING count(ITEM) > 1  
ORDER BY cnt_items;
```