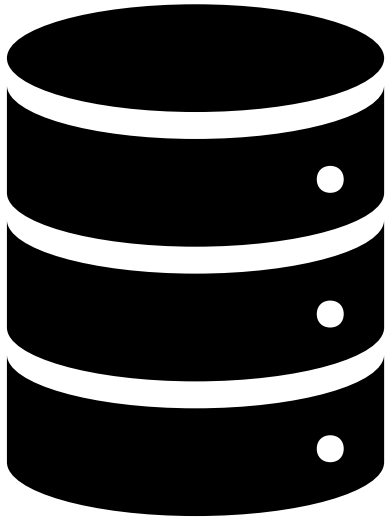
Базы данных



Лекция 2. Структурированный язык запросов SQL

Меркурьева Надежда
✉ merkurievanad@gmail.com
@merkurievanad

ФИВТ МФТИ, 2019

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

Реляционная БД

- В основе – реляционная модель данных
  - Таблица ≈ Отношение
  - Заголовок отношения ≈ Список наименований колонок таблицы
  - Кортеж ≈ Строка таблица
  - Тело отношения ≈ Все строки таблицы
- Средство манипуляции – реляционные системы управления базами данных
- Способ манипуляции – специальный язык запросов

Structed 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

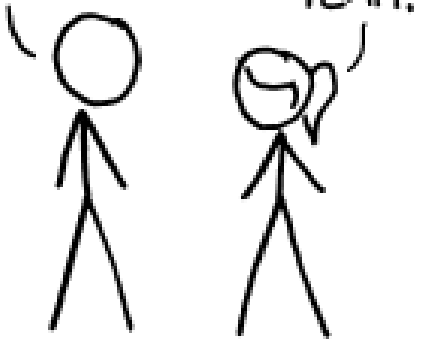
- Предпосылки:
  - Разное ПО от разных производителей
  - Собственная реализация языка запросов
- Хотели получить:
  - Переносимость ПО
- Получили:
  - Частичная переносимость

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.

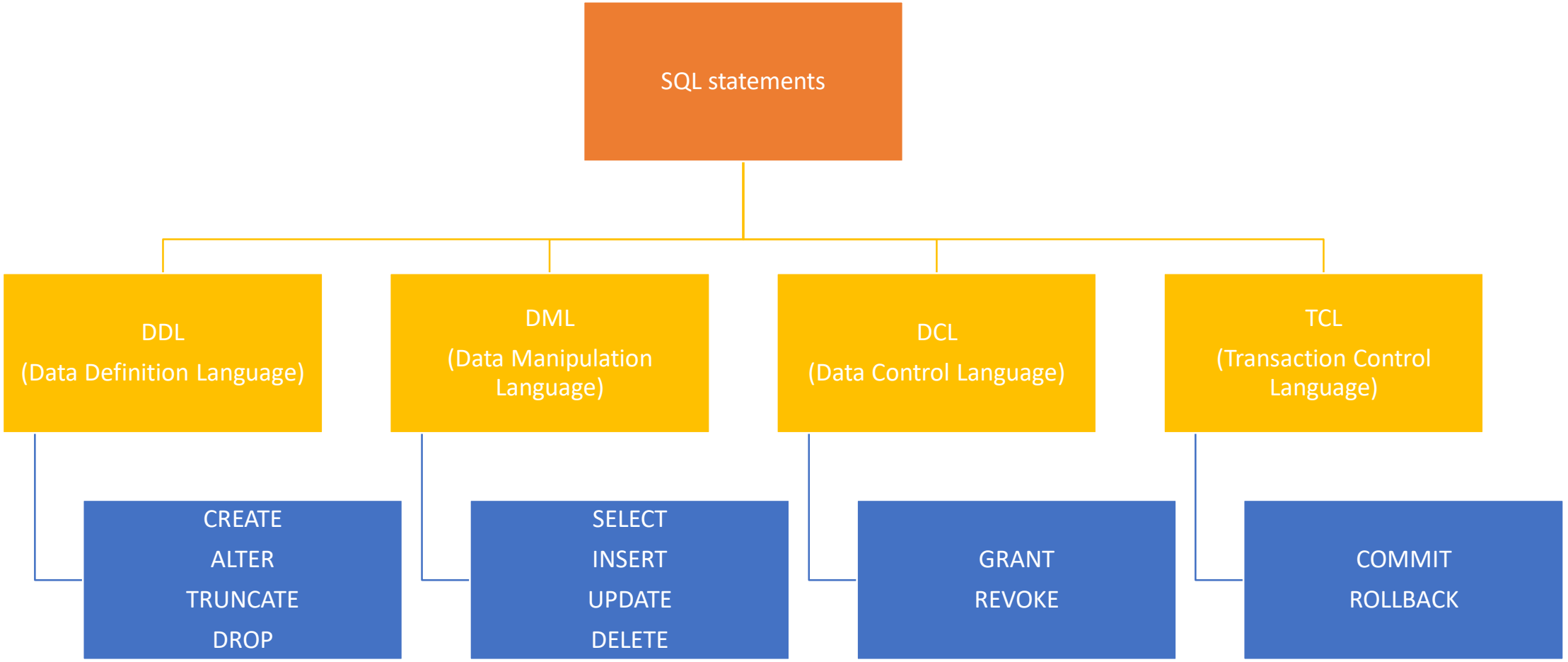
Стандарты языка SQL

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, нескалярные 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

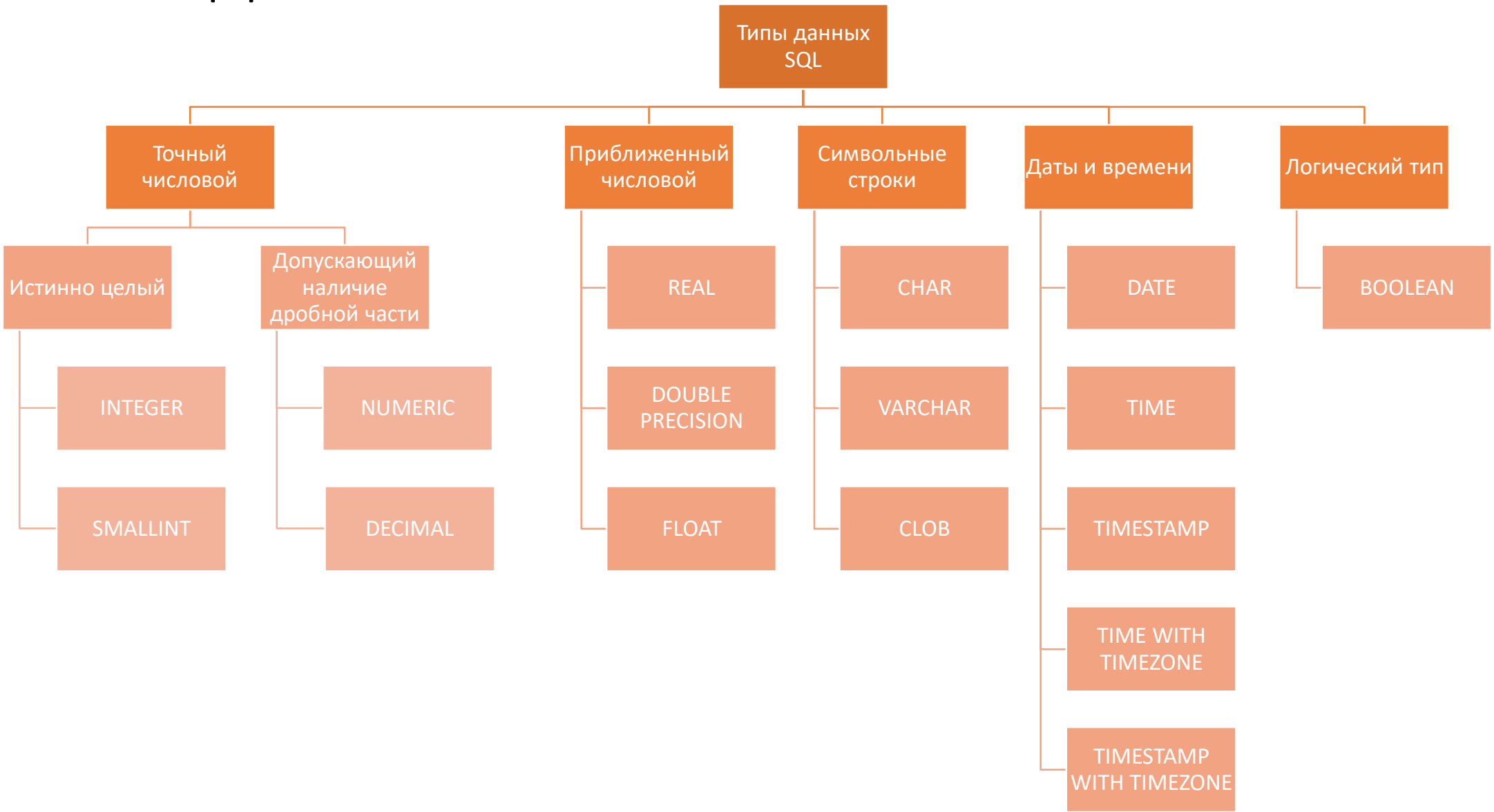
Проблемы стандартов

- Core-раздел стандарта (введен в 1992)
- Производители обеспечивают соответствие только Core
- Различия в реализации
- Различия в исполнении
- 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_DATE DATE,
   ALTER_EGO  VARCHAR(50)
);
```

DDL: CREATE

NAME	AGE	BIRTH_DATE	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;
```

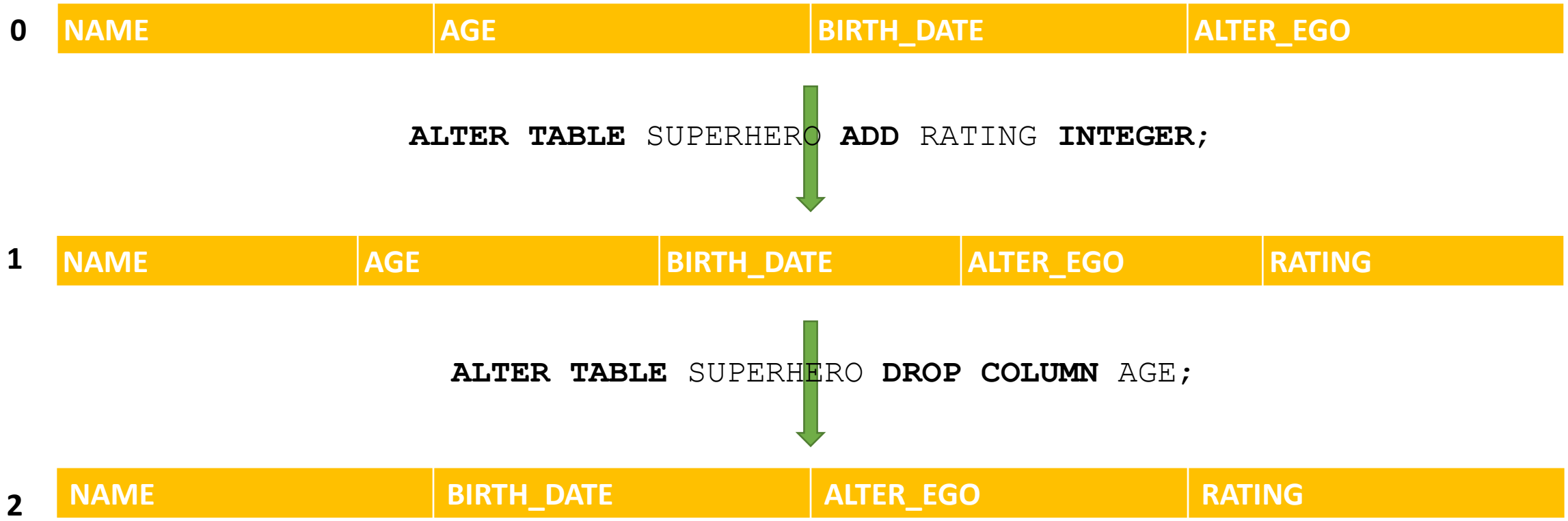
Реальных кейсов применения больше!

DDL: ALTER

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

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

DDL: ALTER


0
NAME
AGE
BIRTH_DATE
ALTER_EGO
ALTER TABLE SUPERHERO ADD RATING INTEGER;
1
NAME
AGE
BIRTH_DATE
ALTER_EGO
RATING
ALTER TABLE SUPERHERO DROP COLUMN AGE;
2
NAME
BIRTH_DATE
ALTER_EGO
RATING

DDL: TRUNCATE

TRUNCATE: физическое удаление данных из объекта единым куском. Данные нельзя удалять частично или по условию.

```
TRUNCATE TABLE table_name;
```

```
TRUNCATE TABLE SUPERHERO;
```

DDL: TRUNCATE

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

```
TRUNCATE TABLE SUPERHERO;
```

NAME	BIRTH_DATE	ALTER_EGO	RATING

DDL: DROP

DROP: удаление объекта из базы

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

```
DROP TABLE SUPERHERO;
```

DDL: DROP

NAME	AGE	BIRTH_DATE	ALTER_EGO

Операторы манипуляции данными (DML)

SELECT	• Выбирает данные, удовлетворяющие заданным условиям
INSERT	• Добавляет новые данные
UPDATE	• Изменяет (обновляет) существующие данные
DELETE	• Удаляет существующие данные

DML: INSERT

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

DML: INSERT

```
INSERT INTO SUPERHERO(
     NAME,
     BIRTH_DATE,
     ALTER_EGO,
     RATING)
VALUES (
     'Natasha Romanoff',
     '01-AUG-1999',
     'Black Widow',
     59 );
```

DML: INSERT

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-1999	Black Widow	59

DML: UPDATE

```
UPDATE table_name
   SET update_assignment_comma_list
 WHERE conditional_experssion;
```

DML: UPDATE

```
UPDATE SUPERHERO
   SET BIRTH_DATE = '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];
```

Отличия от **TRUNCATE**:

- Удаление записей происходит построчно
- Можно удалить не все данные, а только те, которые удовлетворяют условию
- Можно «откатить» удаление данных

DML: DELETE

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

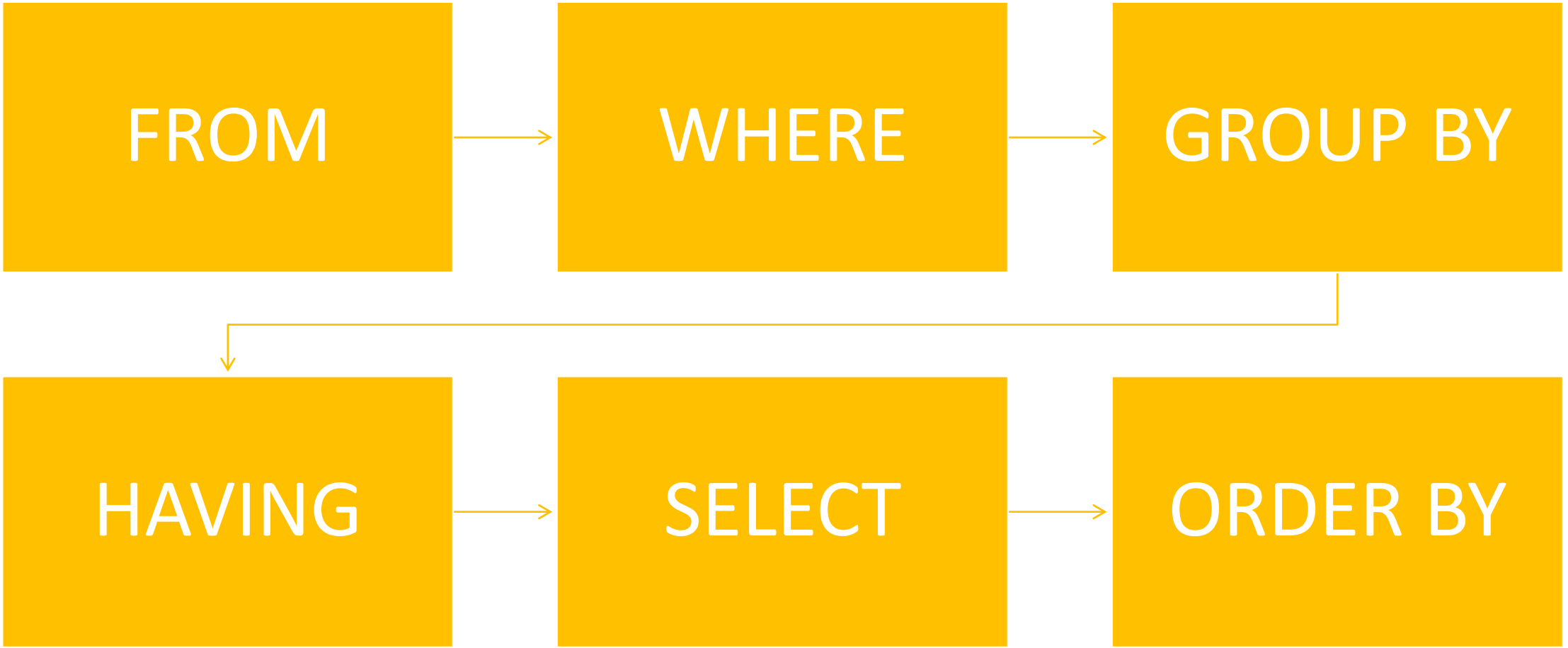
DML: UPDATE

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: 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_DATE	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-BEF-1976	Thor	74
Clint Barton	17-DEC-1969	Hawkeye	55
Wanda Marya Maximoff	22-OCT-1974	Scarlet Witch	81
Pietro Django Maximoff	22-OCT-1974	Quicksilver	82
Charles Francis Xavier	30-JUN-1933	Professor X	100
Jean Elaine Grey-Summers	12-SEP-1961	Phoenix	93
Wade Winston 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 Francis Xavier	Professor X
Jean Elaine Grey-Summers	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 < value3;

- **WHERE** X < value_1 **AND** X > value_2 **OR** X = value_3

- Приоритет операций: NOT, AND, OR

WHERE: специальные функции

Функции ниже используются для типов данных, для которых определен результат сравнения

- **WHERE** X **BETWEEN** *value_1* **AND** *value_2*:
  - `X >= value_1 AND X <= value_2`

- **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_DATE	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-BEF-1976	Thor	74
Clint Barton	17-DEC-1969	Hawkeye	55
Wanda Marya Maximoff	22-OCT-1974	Scarlet Witch	81
Pietro Django Maximoff	22-OCT-1974	Quicksilver	82
Charles Francis Xavier	30-JUN-1933	Professor X	100
Jean Elaine Grey-Summers	12-SEP-1961	Phoenix	93
Wade Winston 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_DATE	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-BEF-1976	Thor	74
Clint Barton	17-DEC-1969	Hawkeye	55
Wanda Marya Maximoff	22-OCT-1974	Scarlet Witch	81
Pietro Django Maximoff	22-OCT-1974	Quicksilver	82
Charles Francis Xavier	30-JUN-1933	Professor X	100
Jean Elaine Grey-Summers	12-SEP-1961	Phoenix	93
Wade Winston 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_DATE, count(ALTER_EGO)
  FROM SUPERHERO
 WHERE BIRTH_DATE = '22-OCT-1974'
 GROUP BY BIRTH_DATE;
```

SELECT: GROUP BY

NAME	BIRTH_DATE	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-BEF-1976	Thor	74
Clint Barton	17-DEC-1969	Hawkeye	55
Wanda Marya Maximoff	22-OCT-1974	Scarlet Witch	81
Pietro Django Maximoff	22-OCT-1974	Quicksilver	82
Charles Francis Xavier	30-JUN-1933	Professor X	100
Jean Elaine Grey-Summers	12-SEP-1961	Phoenix	93
Wade Winston Wilson	13-APR-1980	Deadpool	89
James Howlett	01-JAN-1887	Wolverine	99

SELECT: GROUP BY

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

BIRTH_DATE	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

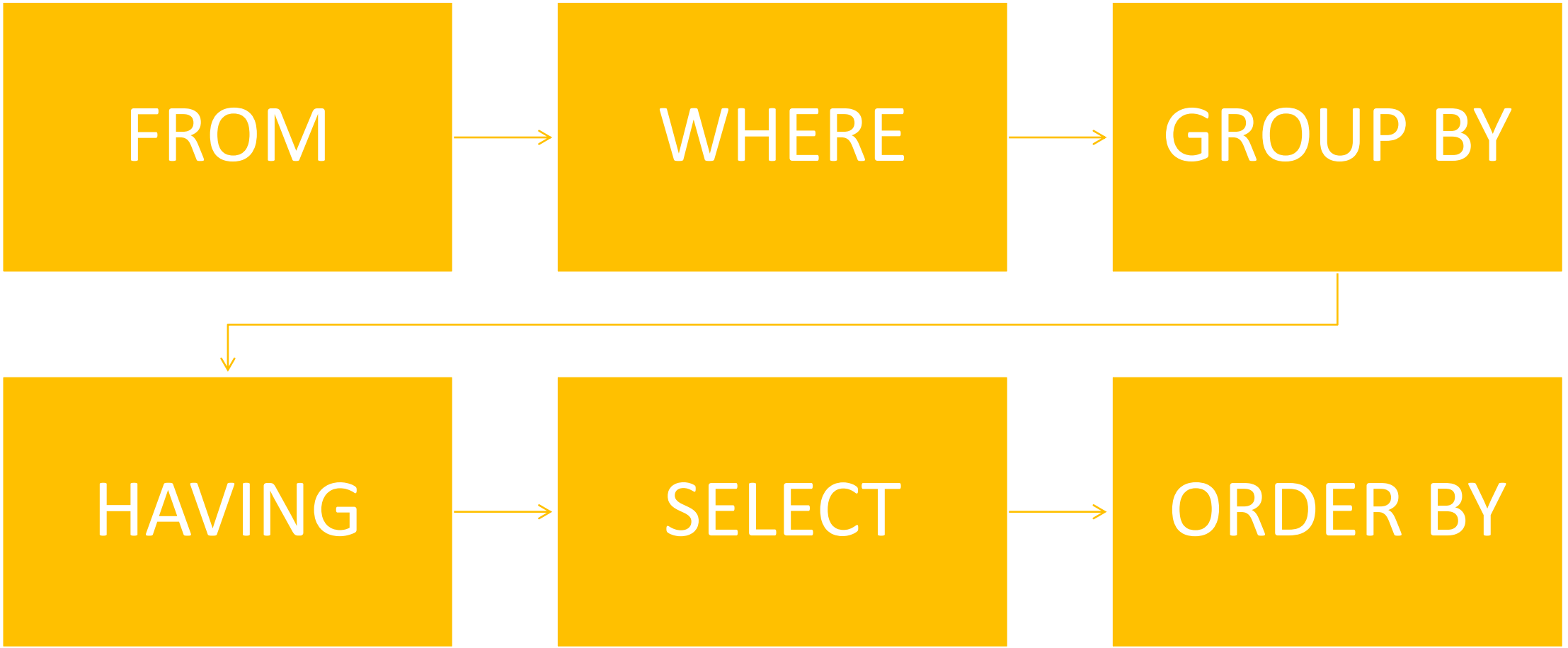
ID	ITEM	MAGAZINE	PRICE	DELIVERY
5	Mobile phone	1	150	15
9	Scanner	2	200	7
12	Headphones	4	200	0
7	Mobile phone	3	250	10
8	Printer	1	300	15
10	Camera	2	500	5
11	Camera	3	550	10
4	Laptop	1	999	15
1	Computer	1	1000	15
6	Laptop	2	1001	5
2	Computer	2	1007	5
3	Computer	3	1100	10

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

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;
```

Соединение запросов

- **CROSS JOIN** (полное декартово произведение таблиц)
- **INNER JOIN** (исключает несовпадающие строки)
- **OUTER JOIN** (содержит несовпадающие строки):
  - LEFT [OUTER] JOIN
  - RIGHT [OUTER] JOIN
  - FULL [OUTER] JOIN

CROSS JOIN

```
SELECT *
  FROM Table_1
 CROSS JOIN Table_2;

SELECT column_list_1,
       column_list_2
  FROM Table_1,
       Table_2;
```

На выходе: полное декартово произведение 2 таблиц

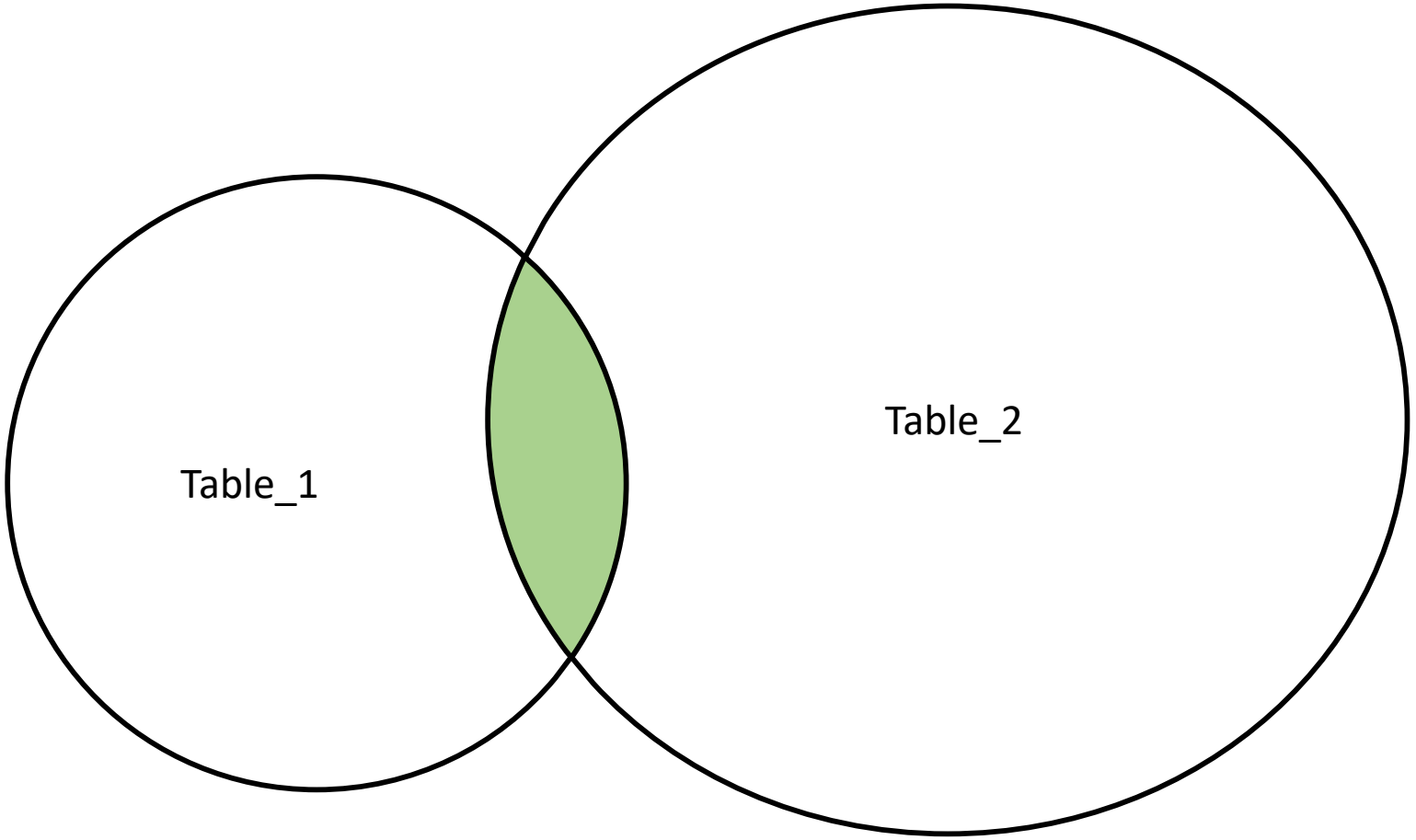
INNER JOIN

«Сцепление» строк 2 таблиц по заданном условию

```
SELECT column_name(s)
  FROM Table1
 INNER JOIN Table2
    ON Table1.column_name = Table2.column_name;
```

В результирующей таблице только те строки, которые совпали по заданному условию соединения

INNER JOIN



INNER JOIN

PROD_ID	PROD_NAME
123	Computer
124	Laptop
125	Scanner
126	Printer
127	Camera
128	Mobile phone

MANUFACTURER_NAME	PROD_ID
Lenovo	123
Lenovo	124
HP	123
HP	124
HP	125
HP	126
Samsung	128
Lenovo	125
Samsung	123
Samsung	129
LG	129

INNER JOIN

- Имеем 2 таблицы:
  - `PRODUCT (ID продукта, Наименование продукта)`
  - `PROD_MAN ( Наименование производителя, ID продукта)`
- Хотим получить связку «продукт-производитель»:
  1. Соединяем таблицы `PRODUCT` и `PROD_MAN`, получаем искомую таблицу
  2. Оставляем интересующие нас колонки

INNER JOIN

```
SELECT PROD_NAME,
       MANUFACTURER_NAME
  FROM PRODUCT P
 INNER JOIN PROD_MAN PM
    ON P.PROD_ID = PM.PROD_ID;
```

INNER JOIN

PROD_ID	PROD_NAME
123	Computer
124	Laptop
125	Scanner
126	Printer
127	Camera
128	Mobile phone

MANUFACTURER_NAME	PROD_ID
Lenovo	123
Lenovo	124
HP	123
HP	124
HP	125
HP	126
Samsung	128
Lenovo	125
Samsung	123
Samsung	129
LG	129

INNER JOIN: результат выполнения

PROD_NAME	MANUFACTURER_NAME
Computer	Lenovo
Computer	HP
Computer	Samsung
Laptop	Lenovo
Laptop	HP
Scanner	HP
Scanner	Lenovo
Printer	HP
Mobile phone	Samsung

LEFT JOIN

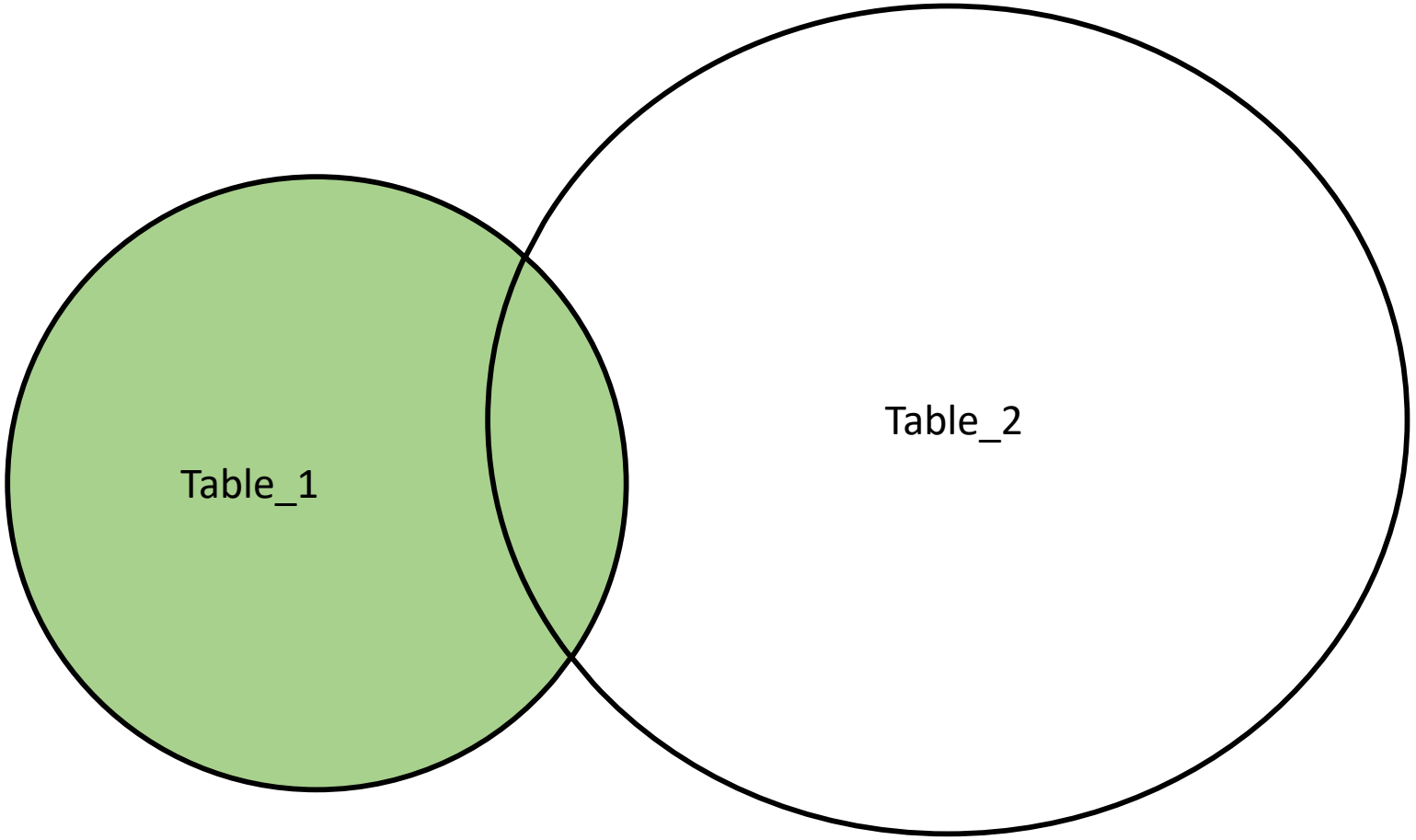
«Сцепление» строк 2 таблиц по заданном условию

```
SELECT column_name(s)
  FROM Table1
  LEFT JOIN Table2
    ON Table1.column_name = Table2.column_name;
```

В результирующей таблице присутствуют **все** строки «левой» таблицы. Те строки, которые не получается соединить с «правой» таблицей, все равно попадают в результирующую таблицу

Значение полей, соответствующих «правой» таблице, тогда заполняются специальным значением NULL

LEFT JOIN



LEFT JOIN

- Задача:
  - Выяснить производителей *всех* продуктов, имеющихся в магазине

- Решение:
  - Использовать LEFT JOIN вместо INNER JOIN в предыдущей задаче

LEFT JOIN

```
SELECT PROD_NAME,
       MANUFACTURER_NAME
  FROM PRODUCT P
  LEFT JOIN PROD_MAN PM
    ON P.PROD_ID = PM.PROD_ID;
```

LEFT JOIN

PROD_ID	PROD_NAME
123	Computer
124	Laptop
125	Scanner
126	Printer
127	Camera
128	Mobile phone

MANUFACTURER_NAME	PROD_ID
Lenovo	123
Lenovo	124
HP	123
HP	124
HP	125
HP	126
Samsung	128
Lenovo	125
Samsung	123
Samsung	129
LG	129

LEFT JOIN: результат выполнения

PROD_NAME	MANUFACTURER_NAME
Computer	Lenovo
Computer	HP
Computer	Samsung
Laptop	Lenovo
Laptop	HP
Scanner	HP
Scanner	Lenovo
Printer	HP
Camera	NULL
Mobile phone	Samsung

RIGHT JOIN

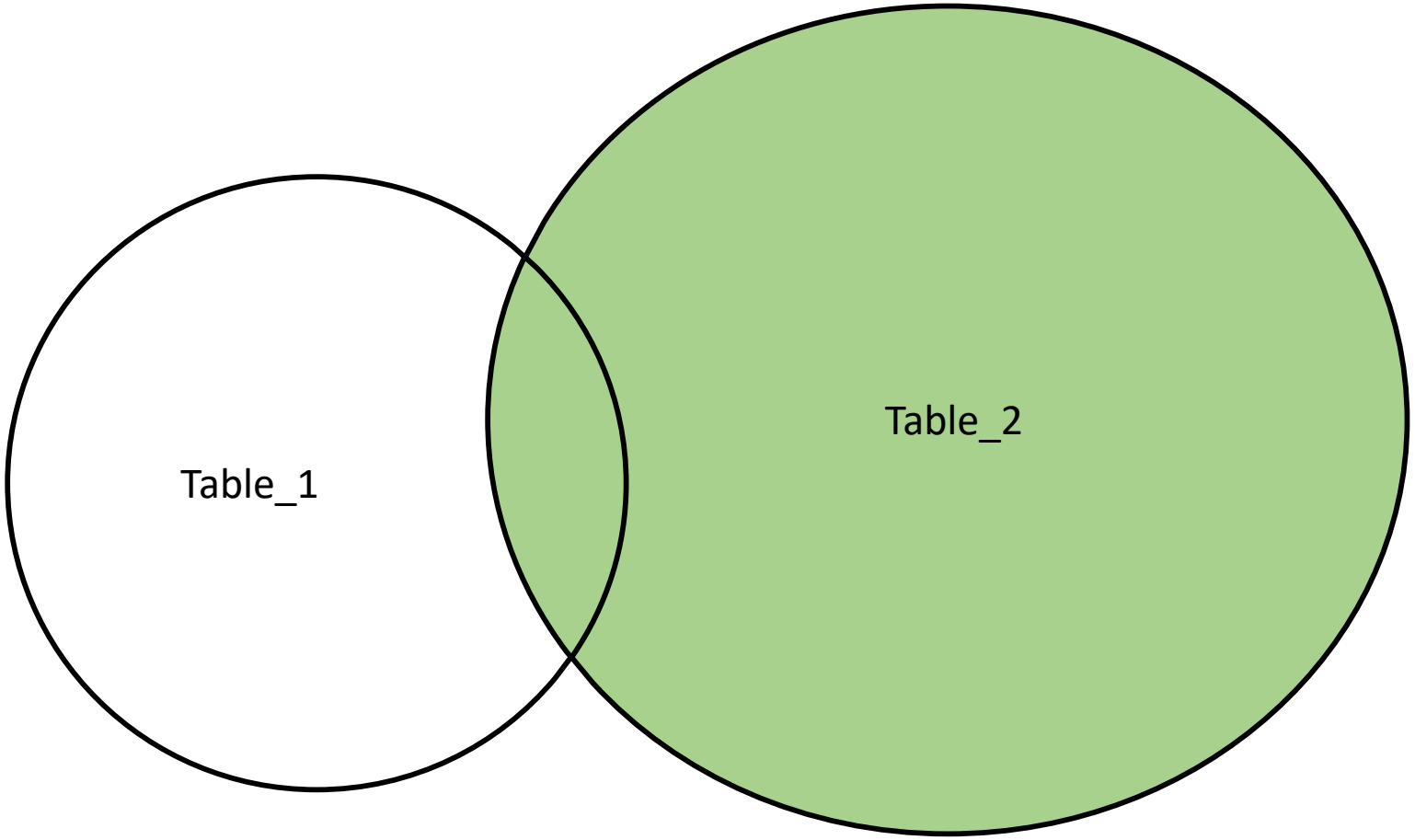
«Сцепление» строк 2 таблиц по заданном условию

```
SELECT column_name(s)
  FROM Table1
 RIGHT JOIN Table2
    ON Table1.column_name = Table2.column_name;
```

В результирующей таблице присутствуют **все** строки «правой» таблицы. Те строки, которые не получается соединить с «левой» таблицей, все равно попадают в результирующую таблицу.

Значение полей, соответствующих «левой» таблице, тогда заполняются значениями NULL

RIGHT JOIN



RIGHT JOIN

- Задача:
  - Выяснить продукты всех производителей, имеющихся на складе

- Решение:
  - Использовать RIGHT JOIN вместо INNER JOIN в предыдущей задаче

RIGHT JOIN

```
SELECT PROD_NAME,
       MANUFACTURER_NAME
  FROM PRODUCT P
 RIGHT JOIN PROD_MAN PM
    ON P.PROD_ID = PM.PROD_ID;
```

RIGHT JOIN

PROD_ID	PROD_NAME
123	Computer
124	Laptop
125	Scanner
126	Printer
127	Camera
128	Mobile phone

MANUFACTURER_NAME	PROD_ID
Lenovo	123
Lenovo	124
HP	123
HP	124
HP	125
HP	126
Samsung	128
Lenovo	125
Samsung	123
Samsung	129
LG	129

RIGHT JOIN: результат выполнения

PROD_NAME	MANUFACTURER_NAME
Computer	Lenovo
Computer	HP
Computer	Samsung
Laptop	Lenovo
Laptop	HP
Scanner	HP
Scanner	Lenovo
Printer	HP
Mobile phone	Samsung
NULL	Samsung
NULL	LG

FULL JOIN

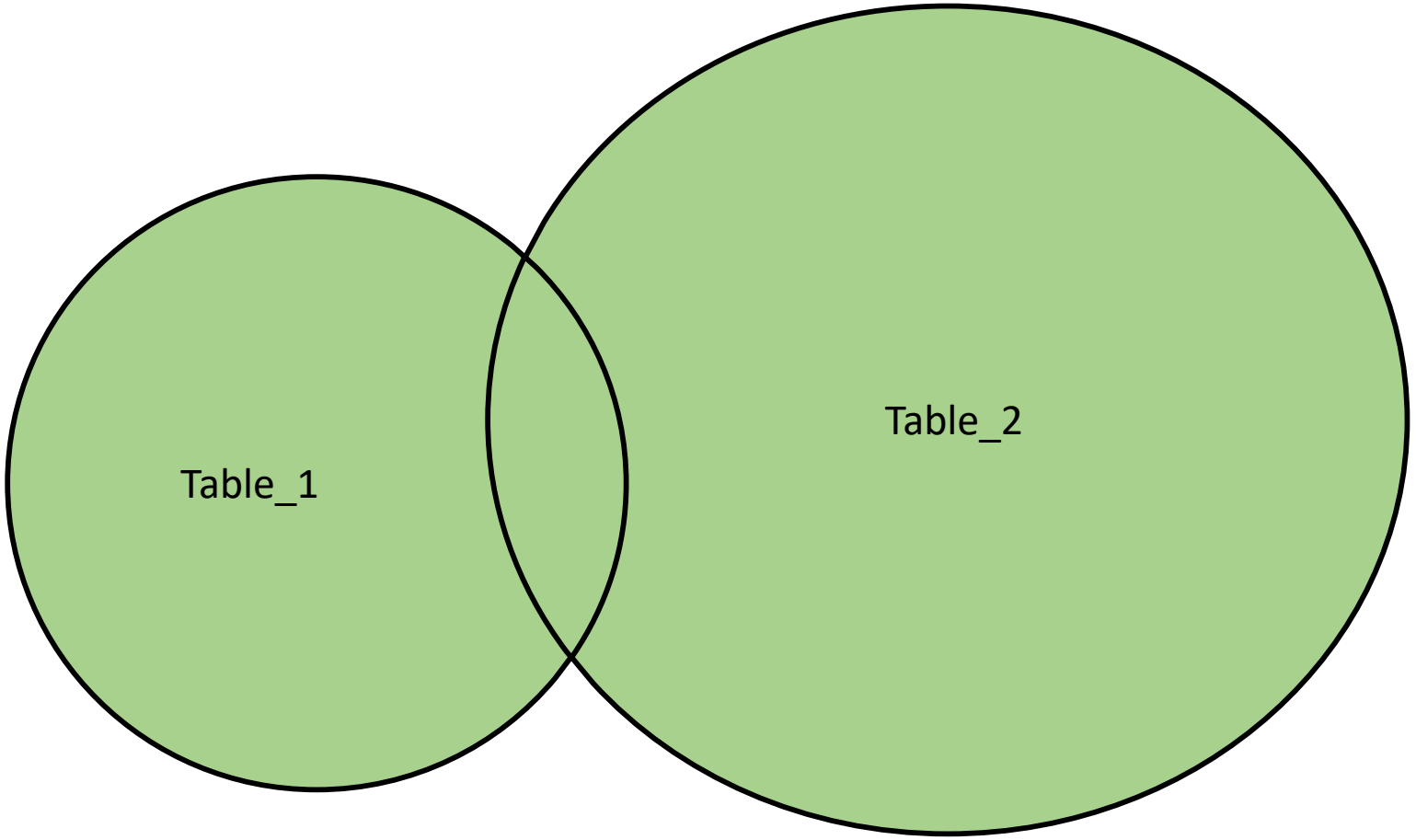
«Сцепление» строк 2 таблиц по заданном условию

```
SELECT column_name(s)
  FROM Table1
  FULL JOIN Table2
    ON Table1.column_name = Table2.column_name;
```

В результирующей таблице присутствуют **все** строки «левой» и «правой» таблиц.

Иными словами, является комбинацией левого и правого соединения.

FULL JOIN



FULL JOIN

```
SELECT PROD_NAME,
       MANUFACTURER_NAME
  FROM PRODUCT P
  FULL JOIN PROD_MAN PM
    ON P.PROD_ID = PM.PROD_ID;
```

FULL JOIN

PROD_ID	PROD_NAME
123	Computer
124	Laptop
125	Scanner
126	Printer
127	Camera
128	Mobile phone

MANUFACTURER_NAME	PROD_ID
Lenovo	123
Lenovo	124
HP	123
HP	124
HP	125
HP	126
Samsung	128
Lenovo	125
Samsung	123
Samsung	129
LG	129

FULL JOIN: результат выполнения

PROD_NAME	MANUFACTURER_NAME
Computer	Lenovo
Computer	HP
Computer	Samsung
Laptop	Lenovo
Laptop	HP
Scanner	HP
Scanner	Lenovo
Printer	HP
Camera	NULL
Mobile phone	Samsung
NULL	Samsung
NULL	LG

SELF-JOIN

- Не является отдельным видом соединения
- Бывает полезен для некоторого типа задач

- Задача:
  - Дана таблица RELATIONS (PERSON_ID, NAME, FATHER_ID)
  - Необходимо сформировать таблицу с именами отцов и детей

SELF-JOIN

PERSON_ID	NAME	FATHER_ID
1	Иванов Иван Иванович	10
2	Иванов Николай Иванович	1
3	Петров Дмитрий Сергеевич	12
4	Петров Анатолий Дмитриевич	3
5	Петров Петр Дмитриевич	3
6	Степанов Сергей Иванович	13
7	Степанов Матвей Степанович	6
8	Яковлев Андрей Степанович	6

SELF-JOIN

```
SELECT T2.NAME AS FATHER_NM,
       T1.NAME AS SON_NM
  FROM RELATIONS T1 -- дети
 INNER JOIN RELATIONS T2 -- отцы
    ON T1.FATHER_ID = T2.PERSON_ID
```

SELF-JOIN

PERSON_ID	NAME	FATHER_ID
1	Иванов Иван Иванович	10
2	Иванов Николай Иванович	1
3	Петров Дмитрий Сергеевич	12
4	Петров Анатолий Дмитриевич	3
5	Петров Петр Дмитриевич	3
6	Степанов Сергей Иванович	13
7	Степанов Матвей Степанович	6
8	Яковлев Андрей Степанович	6

SELF-JOIN: результат выполнения

FATHER_NM	SON_NM
Иванов Иван Иванович	Иванов Николай Иванович
Петров Дмитрий Сергеевич	Петров Анатолий Дмитриевич
Петров Дмитрий Сергеевич	Петров Петр Дмитриевич
Степанов Сергей Иванович	Степанов Матвей Степанович
Степанов Сергей Иванович	Яковлев Андрей Степанович