Лекция 2. Часть 2

Язык SQL для работы с реляционными базами данных

Structured Query Language – декларативный язык для описания, изменения и извлечения данных, хранимых в реляционных базах данных.

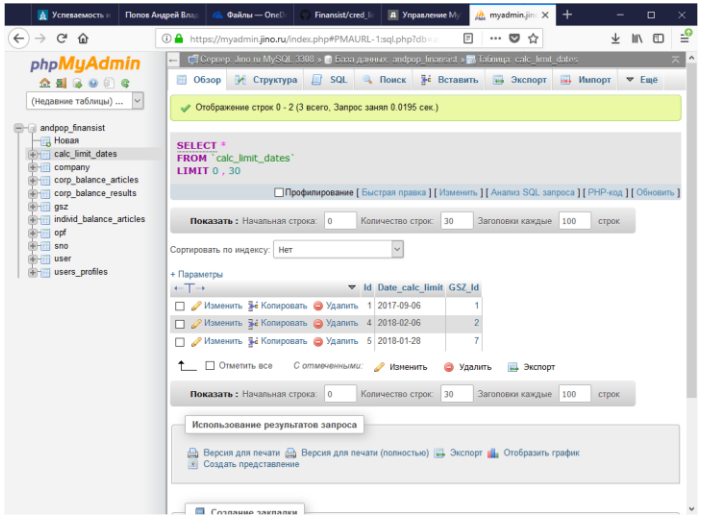
- Стандартизирован организациями ANSI, ISO (SQL-86, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL:2016, SQL:2019)
- Кроссплатформенный
- Простой в изучении и использовании
- Интерактивный и программный режим запросов
- Обеспечивает различные представления данных
- Хорошо подходит для архитектуры «клиент-сервер»

Категории команд

- DDL (Data Definition Language). CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, ALTER INDEX, DROP INDEX.
- DML (Data Manipulation Language). INSERT, UPDATE, DELETE
- DQL (Data Query Language). SELECT
- DCL (Data Control Language). GRANT, REVOKE.
- Команды управления транзакциями. BEGIN, COMMIT, ROLLBACK, SAVEPOINT, SET TRANSACTION.

Интерактивный режим

Web



CLI

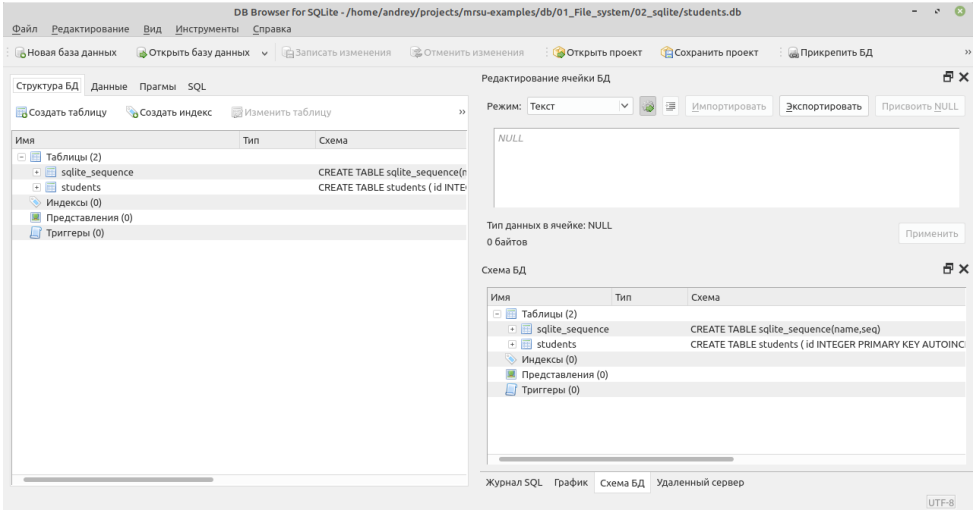
```
andrey@home:~$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 14
Server version: 10.3.22-MariaDB-1:10.3.22+maria~bionic-log mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
MariaDB [(none)]>
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.001 sec)
```

Desktop



Создание базы данных

CREATE [OR REPLACE] {**DATABASE** | SCHEMA} [IF NOT EXISTS] имя_базы
[[DEFAULT] CHARACTER SET [=] charset_name]

```
CREATE DATABASE example CHARACTER SET = 'utf8';
USE example;
```

Создание таблиц

CREATE [OR REPLACE] [TEMPORARY] **TABLE** [IF NOT EXISTS] имя_таблицы
{ имя_столбца атрибуты_столбца | CHECK (expr) }, ...
[CONSTRAINT [constraint_name] CHECK (expression)]
[параметры_таблицы]...

Атрибуты столбца:
[NOT NULL | NULL] [DEFAULT default_value | (expression)] [AUTO_INCREMENT]
[ZEROFILL] [UNIQUE [KEY] | [PRIMARY] KEY] [COMMENT 'string'] ...

```
CREATE TABLE `vendor` (
  `id` int(6) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  `name` varchar(30) NOT NULL,
  `city` varchar(30) NOT NULL,
  `percent` int(4) NOT NULL CHECK (`percent`>0)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Рекомендации по стилю SQL: https://www.sqlstyle.guide/ru/

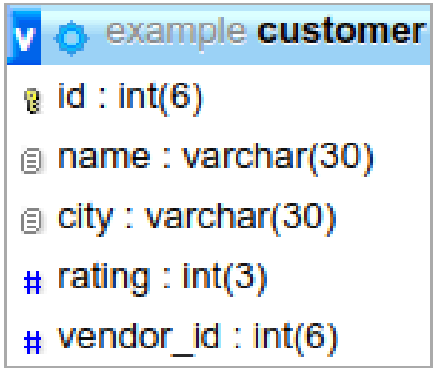
Создание таблиц

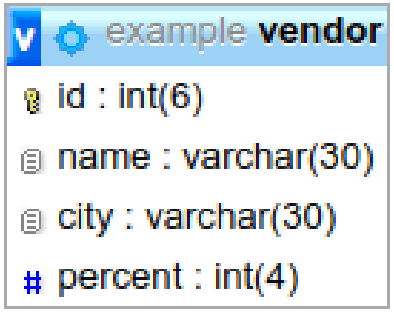
CREATE [TEMPORARY] **TABLE** [IF NOT EXISTS] имя_таблицы
{ имя_столбца атрибуты_столбца | CHECK (expr) }, ...
[CONSTRAINT [constraint_name] CHECK (expression)]
[параметры_таблицы]...

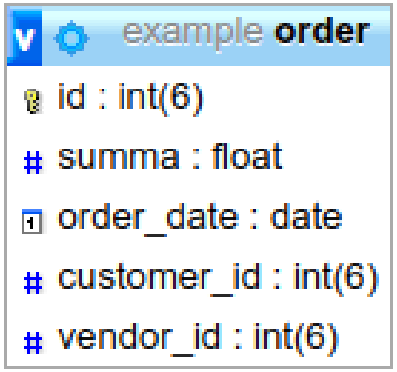
Атрибуты столбца:
[NOT NULL | NULL] [DEFAULT default_value | (expression)] [UNIQUE] | [PRIMARY] KEY] ...

```
CREATE TABLE `vendor` (
  `id` INTEGER PRIMARY KEY,
  `name` VARCHAR(30) NOT NULL,
  `city` VARCHAR(30) NOT NULL,
  `percent` INTEGER NOT NULL CHECK (`percent`>0)
);
```

Создание таблиц







Добавление строк

INSERT INTO имя_таблицы [(имя_столбца, ...)] **VALUES** (значение, ...), ...

INSERT INTO имя_таблицы [(имя_столбца, ...)] **SELECT** (...)

```
INSERT INTO `vendor` (`id`, `name`, `city`, `percent`) VALUES
(1001, 'Иванов', 'Саранск', 12),
(1002, 'Петров', 'Москва', 13),
(1003, 'Андреев', 'Кострома', 10),
(1004, 'Сидоров', 'Саранск', 10);
```

Добавление строк

Vendor (Продавцы)

id	name	city	percent
1001	Иванов	Саранск	12
1002	Петров	Москва	13
1003	Андреев	Кострома	10
1004	Сидоров	Саранск	10

Customer (Покупатели)

id	name	city	rating	vendor_id
2001	Потапов	Саранск	100	1001
2002	Гарин	Владимир	200	1003
2003	Ли	Москва	200	1002
2004	Глухов	Самара	300	1002
2005	Клюев	Саранск	100	1001

Order (Заказы)

id	summa	Date	Pok_Nom	Prod_Nom
3001	18.69	10.12.2016	2005	1001
3002	767.19	10.12.2016	2001	1001
3003	1900.10	10.12.2016	2002	1003
3004	123.45	15.12.2016	2003	1002
3005	100.00	16.12.2016	2004	1002
3006	95.13	15.12.2016	2001	1001

SELECT: выбор из одной таблицы

Выбор полей

```
SELECT id, name, city FROM vendor
SELECT * FROM vendor
```

Условие поиска

```
SELECT id, name, city FROM vendor
WHERE city="Саранск"
```

Логические операции в условии поиска

```
SELECT * FROM customer WHERE rating>=200 AND city="Москва"
```

Значение поля принадлежит множеству

```
SELECT * FROM vendor WHERE city IN ("Москва", "Саранск")
```

Поиск по шаблону

```
SELECT * FROM customer WHERE name LIKE "Ив%"
```

("_" заменяет один символ)

```
SELECT * FROM customer WHERE name RLIKE "ов$";  (MySQL)

SELECT * FROM customer WHERE LEFT(name, 2) = "Ив";    (MySQL)
SELECT * FROM customer WHERE substr(name, 1, 2) = "Ив";    (SQLite)
```

SELECT: выбор из одной таблицы

Построение вычисляемых полей

MySQL
```
SELECT CONCAT(LEFT(name,2),".-", city) AS "Фамилия-Город" FROM vendor
```

SQLite
```
SELECT substr(name,1,2) || ".-" || city) AS "Фамилия-Город" FROM vendor
```

Сортировка выводимых данных

```
SELECT *
FROM vendor
ORDER BY name
```

Функции агрегирования (COUNT, SUM, AVG, MAX, MIN)

Общая сумма заказов

```
SELECT SUM(summa)
FROM `order`
```

Общее число покупателей

```
SELECT COUNT(*)
FROM customer
```

SELECT: выбор из одной таблицы

Промежуточные итоги (группировка)

Для каждого продавца определить максимальную сумму заказа

```
SELECT vendor_id, MAX(summa)
FROM `order`
GROUP BY vendor_id
```

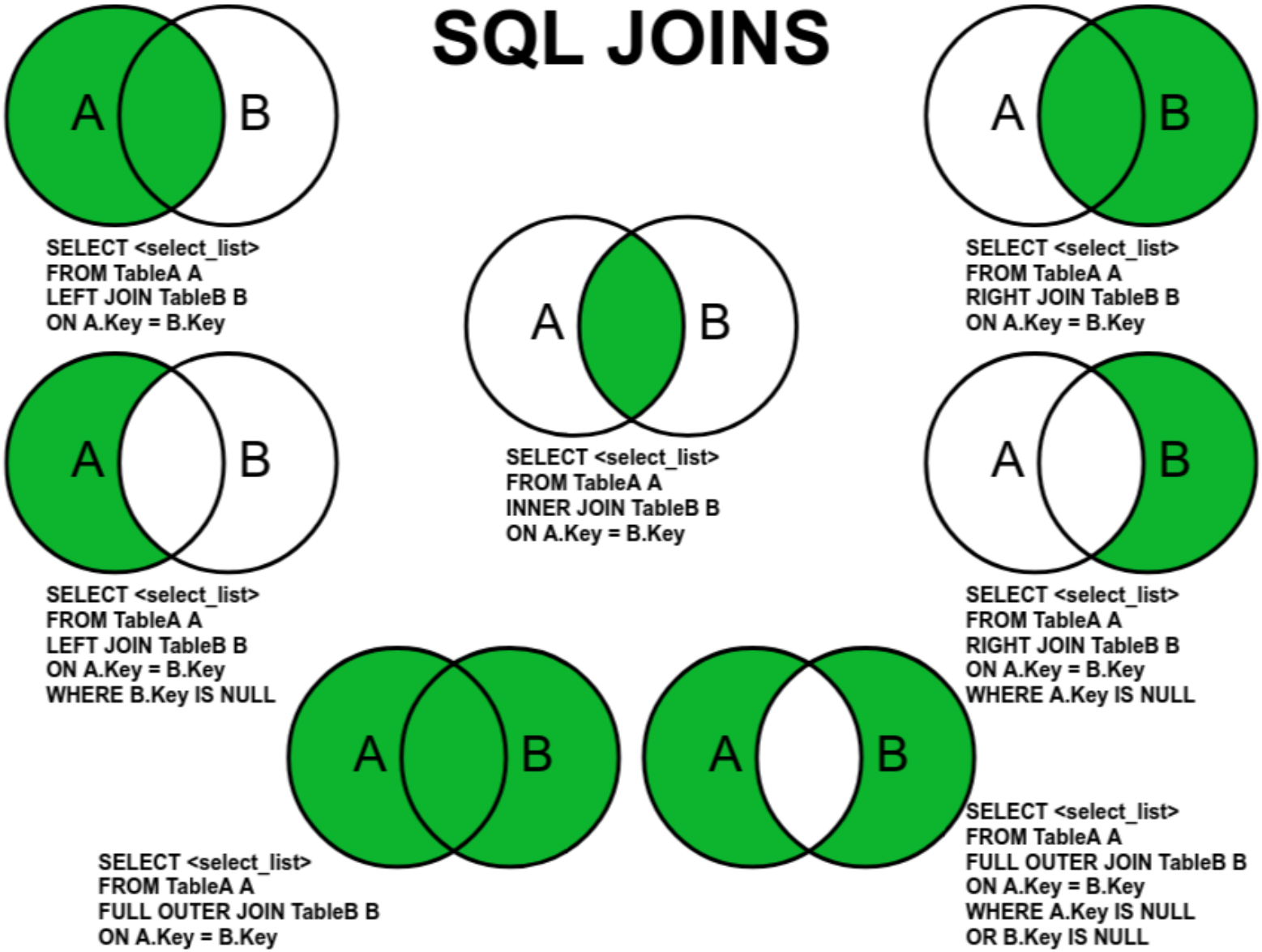
Найти максимальные продажи для каждого продавца на каждый день

```
SELECT vendor_id, order_date, MAX(summa)
FROM `order`
GROUP BY vendor_id, order_date
```

Условия на группы записей - HAVING

```
SELECT vendor_id, order_date, MAX(summa)
FROM `order`
GROUP BY vendor_id, order_date
HAVING MAX(summa)>100
```

SELECT: соединение таблиц



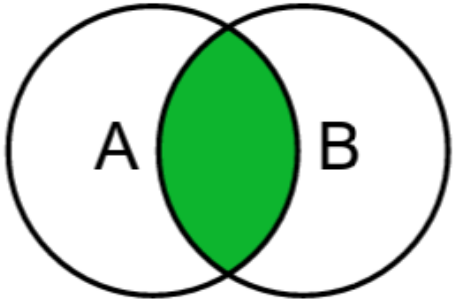
SELECT: соединение нескольких таблиц

Соединение по равенству (внутреннее соединение)

Выбрать всех покупателей и продавцов, живущих в одном городе

```
SELECT customer.name, vendor.name, vendor.city
FROM vendor, customer
WHERE vendor.city=customer.city
```

```
SELECT customer.name, vendor.name, vendor.city
FROM vendor INNER JOIN customer
ON vendor.city=customer.city
```



SELECT: соединение нескольких таблиц

Самообъединение

Найти все пары покупателей, имеющих одинаковый рейтинг

```
SELECT a.name AS Customer1, b.name AS Customer2, a.rating
FROM customer a, customer b
WHERE a.rating=b.rating
```

SELECT: соединение нескольких таблиц

Внешнее соединение

В отличие от внутренних объединений, которые связывают строки двух таблиц, внешние объединения включают в результат также строки, не имеющие пар.

SELECT: соединение нескольких таблиц

Внешнее соединение

В отличие от внутренних объединений, которые связывают строки двух таблиц, внешние объединения включают в результат также строки, не имеющие пар.

Вывести все пары Продавец-Покупатель (в том числе продавцов, не имеющих покупателей)

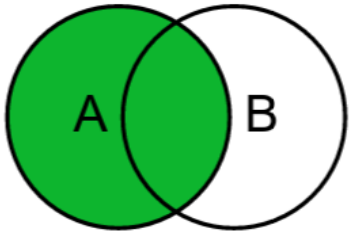
SELECT: соединение нескольких таблиц

Внешнее соединение

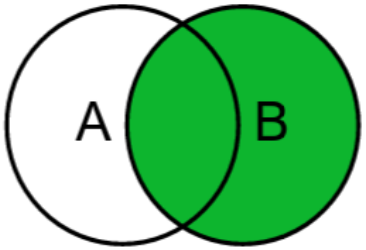
В отличие от внутренних объединений, которые связывают строки двух таблиц, внешние объединения включают в результат также строки, не имеющие пар.

Вывести все пары Продавец-Покупатель (в том числе продавцов, не имеющих покупателей)

```
SELECT vendor.name, customer.name
FROM vendor LEFT OUTER JOIN customer
ON vendor.id = customer.vendor_id
```




```
SELECT vendor.name, customer.name
FROM customer RIGHT OUTER JOIN vendor
ON vendor.id = customer.vendor_id
```



SELECT: вложенные запросы

Обычно внутренний запрос генерирует значения, которые тестируются на предмет истинности предиката.

По фамилии продавца найти все его заказы

```
SELECT * FROM `order`
WHERE vendor_id = (SELECT id FROM vendor WHERE name="Иванов")
```

```
SELECT * FROM `order`
WHERE vendor_id IN (SELECT id FROM vendor WHERE name="Иванов")
```

SELECT: вложенные запросы

Внутри HAVING можно применять подзапросы, которые не дают множества значений.

Найти количество покупателей с рейтингом, превышающим среднее значение для покупателей из Саранска

```
SELECT rating, COUNT(DISTINCT id) FROM customer
GROUP BY rating
HAVING rating > (SELECT AVG(rating)
FROM customer WHERE city="Саранск")
```

SELECT: связанные подзапросы

Найти всех покупателей, сделавших заказы 10.12.2016

```
SELECT * FROM customer a
WHERE “2016-12-10” IN
   (SELECT order_date FROM order b WHERE a.id = b.customer_id)
```

Внутренний запрос должен выполняться отдельно для каждой строки внешнего запроса.

1. Выбирается текущая строка-кандидат из таблицы, указанной во внешнем запросе.
2. Значение этой строки сохраняется в псевдониме из FROM внешнего запроса.
3. Выполняется подзапрос. Каждый раз, когда встречается псевдоним для внешнего запроса, его значение применяется к текущей строке-кандидату (внешней ссылке).
4. Оценивается предикат внешнего запроса на основе подзапроса, выполненного на шаге 3 (будет ли строка-кандидат включена в выходные данные).
5. Процедура повторяется для следующей строки-кандидата.

SELECT: связанные подзапросы

1. **Найдем имена и номера всех продавцов, имеющих более одного покупателя**

```
SELECT id, name
FROM vendor main
WHERE 1 < (SELECT COUNT(*) FROM customer WHERE vendor_id=main.id)
```

2. **Найдем все заказы, сумма в которых превышает среднюю сумму заказа для данного покупателя**

```
SELECT *
FROM order a
WHERE summa > (SELECT AVG(summa) FROM `order` b
    WHERE a.customer_id = b.customer_id)
```

SELECT: ключевое слово EXISTS

Используется только совместно с подзапросами. Результат равен TRUE, если в возвращаемой подзапросом результирующей таблице присутствует хотя бы одна строка, иначе FALSE.

Найти продавцов, имеющих несколько покупателей

```
SELECT DISTINCT vendor_id FROM customer a
WHERE EXISTS (SELECT * FROM customer b WHERE b.vendor_id = a.vendor_id AND
b.id <> a.id)
```

SELECT: ключевые слова ANY, ALL

Используются только совместно с подзапросами. В SQLite нет.

Если подзапросу предшествует ключевое слово **ALL**, условие сравнения считается выполненным, только когда оно выполняется для всех значений в результирующем столбце подзапроса.

Выбрать все заказы, сумма которых превосходит суммы всех заказов, сделанных 10.12.2016

```
SELECT * FROM `order`
WHERE summa > ALL (
  SELECT summa FROM order WHERE order_date="2016-12-10")
```

SELECT: условный оператор CASE

Проверяет условие и возвращает один из возможных вариантов.

Первая форма:	**Вторая форма:**
CASE WHEN условие_1 THEN возвращаемое_значение_1 … WHEN условие_N THEN возвращаемое_значение_N [ELSE возвращаемое_значение] END	CASE проверяемое_значение WHEN сравниваемое_значение_1 THEN возвращаемое_значение_1 … WHEN сравниваемое_значение_N THEN возвращаемое_значение_N [ELSE возвращаемое_значение] END

SELECT: условный оператор CASE

Проверяет условие и возвращает один из возможных вариантов.

```
SELECT id, name AS 'Фамилия',
CASE city
  WHEN 'Саранск' THEN 'Да'
  ELSE 'Нет'
END 'Земляк'
FROM customer;

SELECT id, name AS 'Фамилия',
CASE
  WHEN rating >= 300 THEN 'Молодец'
  WHEN rating >=200 THEN 'Неплохо'
  ELSE 'Старайся больше'
END 'Оценка по рейтингу'
FROM customer;
```

SELECT: оконные функции

Позволяют разбить набор данных на окна - набор строк, в которых происходит вычисление функции. Результаты работы оконных функций добавляются к выборке как еще одно поле.

функция(выражение) **OVER** ([PARTITION BY выражения] [ORDER BY выражения])

Оконные функции:
ROW_NUMBER(), RANK(), FIRST_VALUE(), LAST_VALUE(), ...
Также могут использоваться функции агрегирования SUM(), MIN, MAX, ...

```
SELECT *, ROW_NUMBER() OVER () AS number FROM customer;

SELECT *, ROW_NUMBER() OVER (ORDER BY rating DESC) AS rating_place FROM customer ORDER BY name;

SELECT *, RANK() OVER (ORDER BY rating DESC) AS rank FROM customer ORDER BY name;

SELECT *, MAX(rating) OVER () FROM customer;

SELECT name, city, rating, rating-AVG(rating) OVER () AS delta_rating FROM customer;

SELECT *, COUNT(*) OVER (PARTITION BY city) AS customers_in_city FROM customer;
```

UNION

Объединение строк из нескольких результирующих наборов данных.

- Количество столбцов в каждом запросе одно и то же.
- Столбцы в каждом запросе имеют совместимые типы.
- Имена столбцов в первом запросе задают имена для всего объединения.
- ORDER BY применяется ко всему набору.
- GROUP BY и HAVING применяются к каждому подзапросу.

Вывести всех покупателей и продавцов с указанием их роли и города, где они живут. Сортировать по имени города и фамилии человека.

```
SELECT name, city, 'Продавец' AS 'Роль' FROM vendor
UNION
SELECT name, city, 'Покупатель' FROM customer
ORDER BY city, name;
```

Представления

Объект базы данных, являющийся результатом выполнения запроса к базе данных, определенного с помощью оператора SELECT, в момент обращения к представлению.

- Упрощение запросов.
- Гибкая настройка прав доступа к данным (права даются не на таблицу, а на представление).
- Разделение логику хранения данных и программного обеспечения.

CREATE VIEW имя [столбцы]
AS выборка

```
CREATE VIEW orders_with_names
AS SELECT `order`.order_date, customer.name AS customer_name,
vendor.name AS vendor_name, `order`.summa
FROM `order`, customer, vendor
WHERE `order`.customer_id = customer.id AND `order`.vendor_id = vendor.id;
```

INSERT, UPDATE, DELETE

Вставка строк в таблицу

```
INSERT INTO vendor(name, city, percent) VALUES (“Иванов", "Тверь", 10)
INSERT INTO … SELECT …
```

Изменение строк

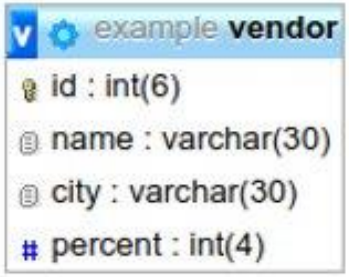
```
UPDATE vendor SET name = “Шахов” WHERE name = “Иванов”
```

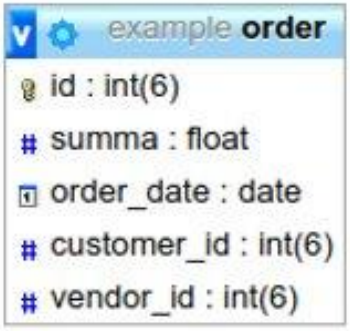
Удаление строк

```
DELETE FROM vendor WHERE name = “Шахов”
```

Задание связей и ограничений

ALTER TABLE имя_таблицы
ADD [CONSTRAINT [имя]] **FOREIGN KEY** [имя_ключа] (имя_столбца,...)
REFERENCES имя_таблицы (имя_столбца,...)
[**ON DELETE** RESTRICT | CASCADE | SET NULL | SET DEFAULT]
[**ON UPDATE** RESTRICT | CASCADE | SET NULL | SET DEFAULT]

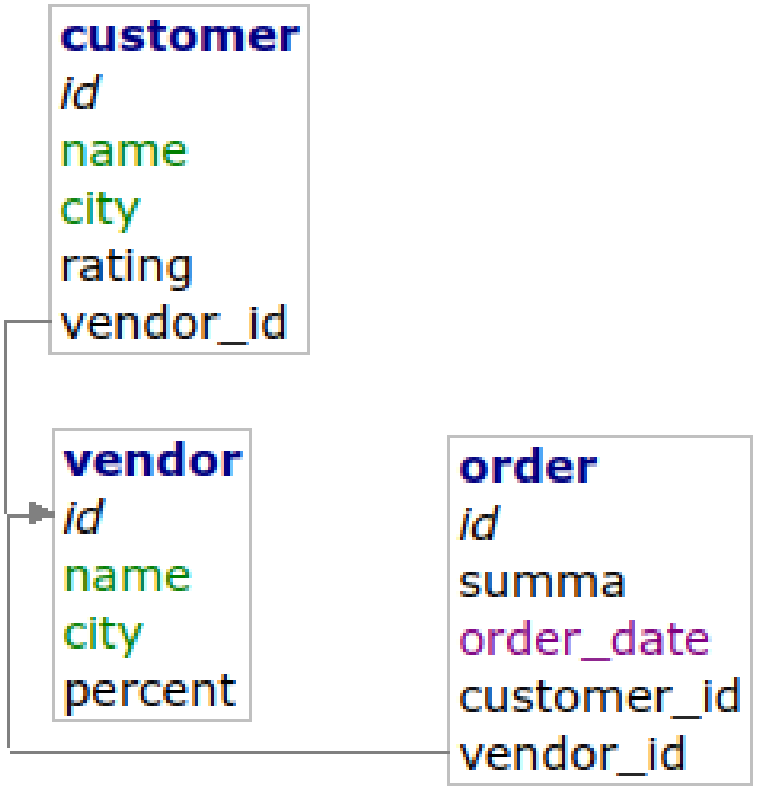




Задание связей и ограничений

```
ALTER TABLE `order`
ADD CONSTRAINT `order_vendor_fk` FOREIGN KEY (`vendor_id`) REFERENCES
`vendor` (`id`) ON DELETE RESTRICT;

ALTER TABLE `customer`
ADD CONSTRAINT `customer_vendor_fk` FOREIGN KEY (`vendor_id`)
REFERENCES `vendor` (`id`) ON DELETE SET NULL;
```



Задание связей и ограничений


```
CREATE TABLE `order` (
  `id` INTEGER PRIMARY KEY,
  `summa` NUMERIC NOT NULL,
  `order_date` DATE NOT NULL,
  `customer_id` INTEGER NOT NULL,
  `vendor_id` INTEGER NOT NULL,
  FOREIGN KEY (`vendor_id`) REFERENCES `vendor` (`id`)
  ON DELETE RESTRICT
);

CREATE TABLE `customer` (
  `id` INTEGER PRIMARY KEY,
  `name` VARCHAR(30) NOT NULL,
  `city` VARCHAR(30) NOT NULL,
  `rating` INTEGER NOT NULL CHECK (`rating`>0),
  `vendor_id` INTEGER NULL,
  FOREIGN KEY (`vendor_id`) REFERENCES `vendor` (`id`)
  ON DELETE SET NULL
);
```

SQL: императивное программирование

Хранимые процедуры, функции и триггеры

- Повторное использование кода
- Сокращение сетевого трафика
- Безопасность
- Уменьшение сложности кода
- Выполнение бизнес-логики

CREATE PROCEDURE имя (параметры) тело_процедуры

CREATE FUNCTION имя (параметры) **RETURNS** тип тело_процедуры

SQL – императивное программирование

Хранимые процедуры и функции

- Переменные (SET, DECLARE)
- Условия (IF … THEN … ELSEIF … ELSE …)
- Множественный выбор (CASE)
- Циклы (WHILE, REPEAT, LOOP)
- Курсоры

SQL – Программный режим