Пятая лекция
java for web
SQL

Что такое База Данных ?

База данных (БД) - упорядоченный набор логически взаимосвязанных данных, используемых совместно, и которые хранятся в одном месте.

Если коротко, то простейшая БД это обычная таблица со строками и столбцами в которой хранится разного рода информация (примером может служить таблица в Excel).

Данные в одних таблицах, как правило, связаны с данными других таблиц, откуда и произошло название "реляционные".

С БД нераздельно связано такое понятие как Системы управления базами данных (СУБД), которые предоставляют функционал для работы с БД.

Язык SQL как раз и является частью СУБД, которая осуществляет управление информацией в БД. Мы будем считать БД набором обычных таблиц, которые хранятся в отдельных файлах.

Что такое SQL?

SQL - простой язык программирования, который имеет немного команд и которой может научиться любой желающий.

Расшифровывается как Structured Query Language - язык структурированных запросов, который был разработан для работы с БД, а именно, чтобы получать /добавлять /изменять данные, иметь возможность обрабатывать большие массивы информации и быстро получать структурированную и сгруппированную информацию. Существует множество версий языка SQL, но для соответствия стандартам ANSI (American National Standards Institute) они должны поддерживать основные ключевые слова (такие как SELECT - выбрать, UPDATE - обновить, DELETE - уничтожить, INSERT - вставить, WHERE - где и другие).

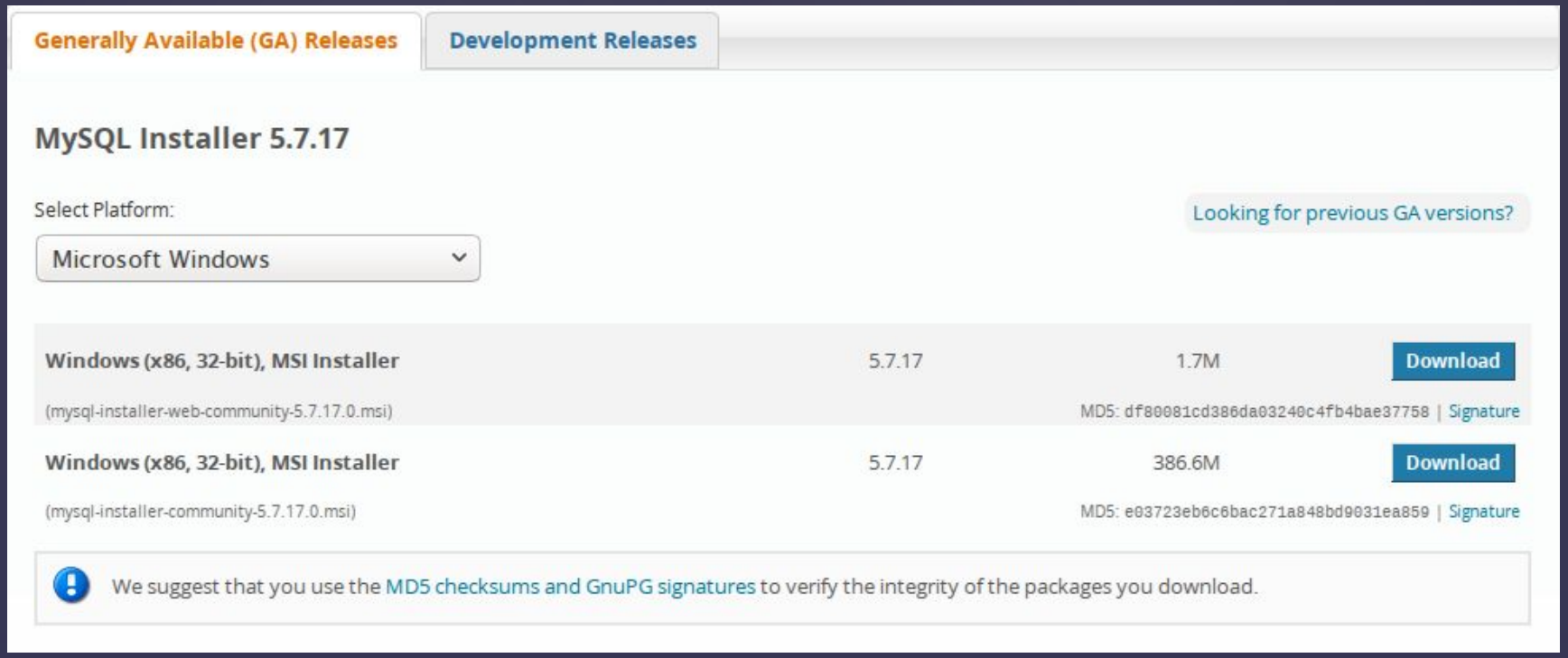
Также существует и много СУБД, но основными из них являются: MySQL, Microsoft Access, Microsoft SQL Server, Oracle SQL, IBM DB2 SQL, PostgreSQL та Sybase Adaptive Server SQL.

Чтобы работать с SQL кодом, нам понадобится одна из вышеперечисленных СУБД. Для обучения мы будем использовать СУБД MySQL.

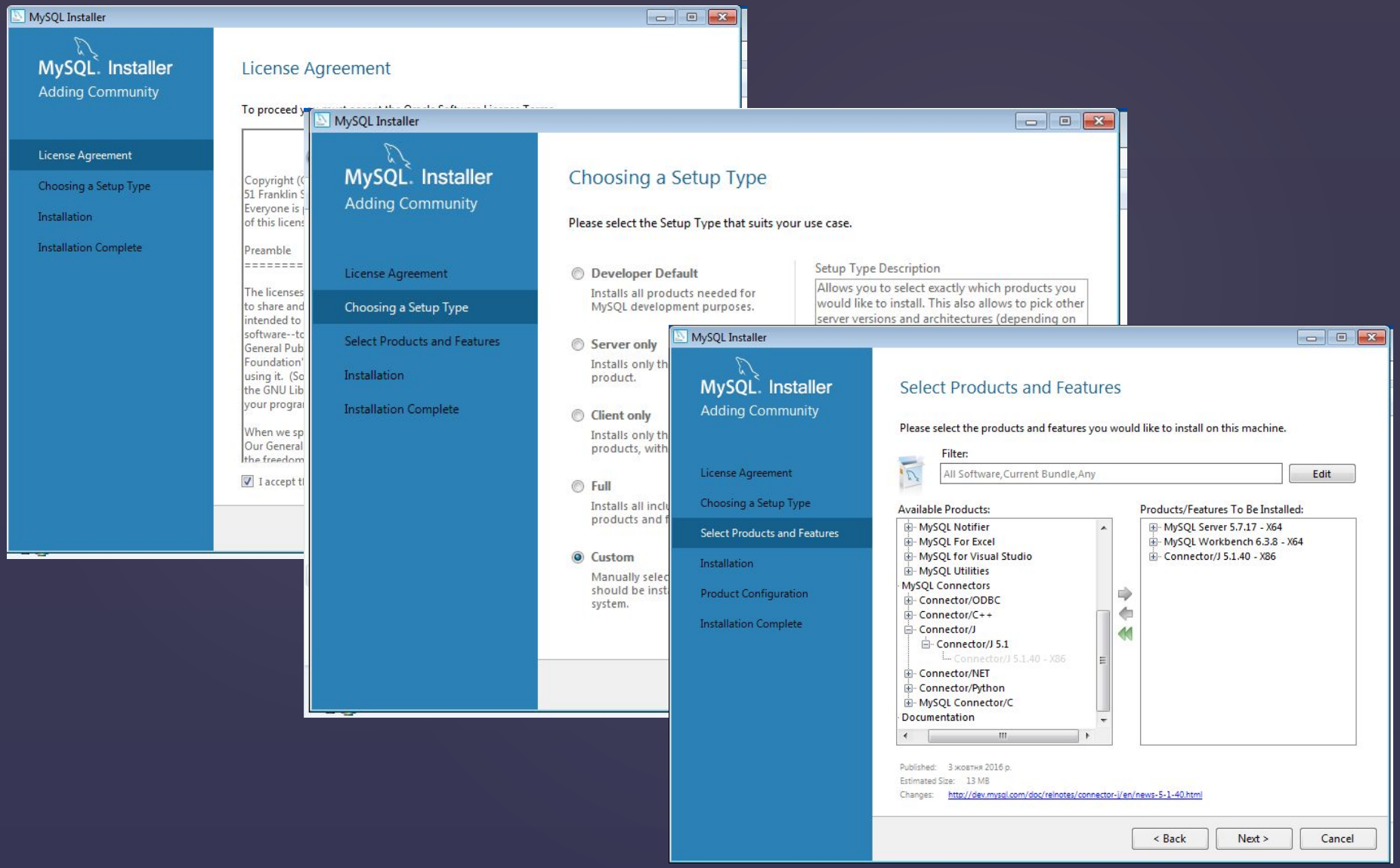
Заметка: Многие СУБД имеют свои команды, в дополнение к существующим стандартам SQL.

Установка MySQL?

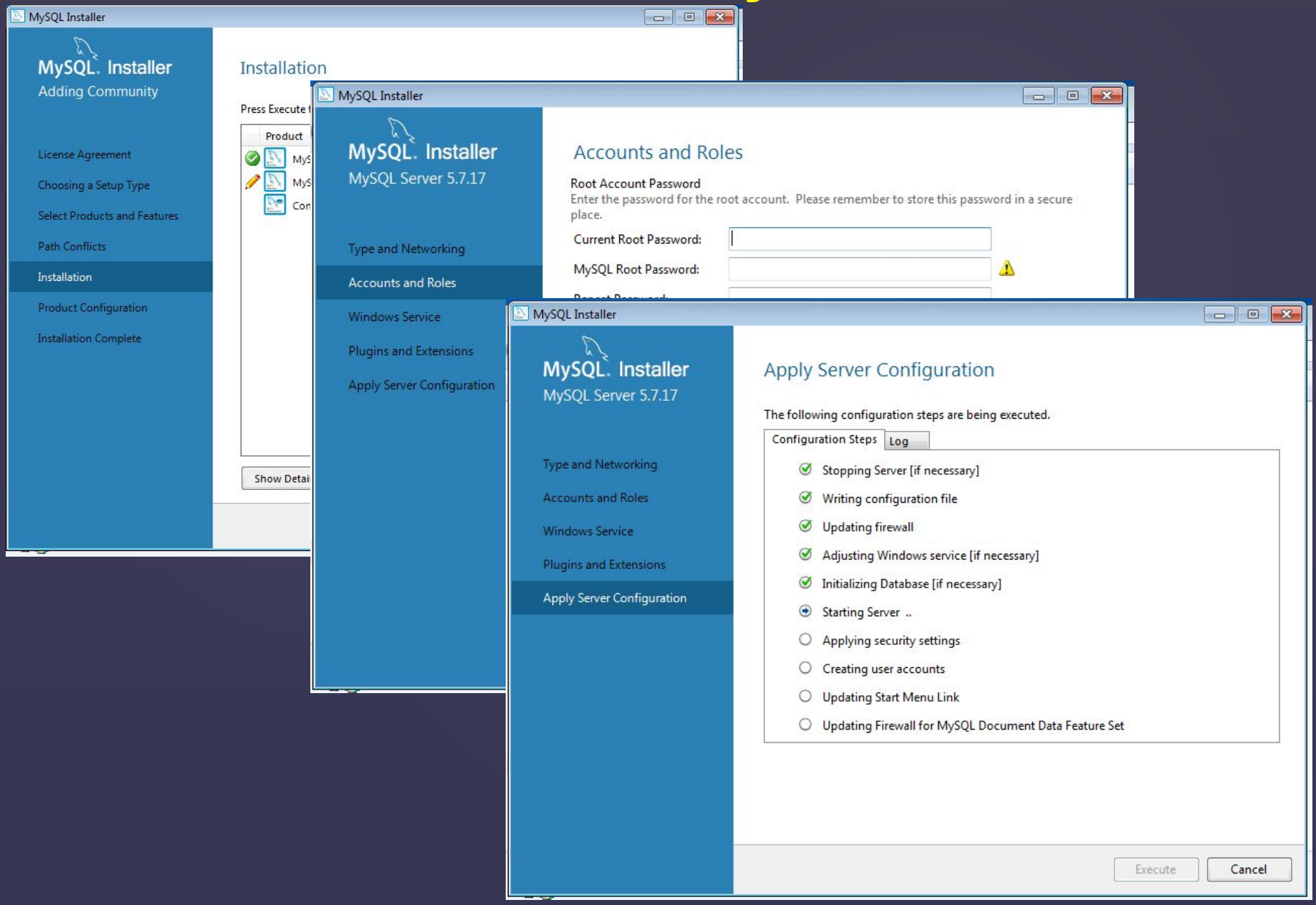
http://dev.mysql.com/downloads/installer/
переходим на сайт и скачиваем
MySQL Installer



Установка MySQL?



Установка MySQL?



MySQL Workbench

MySQL Workbench — инструмент для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию БД.

Workbench дает возможность визуально проектировать вашу базу данных, т.е. составлять схему БД.

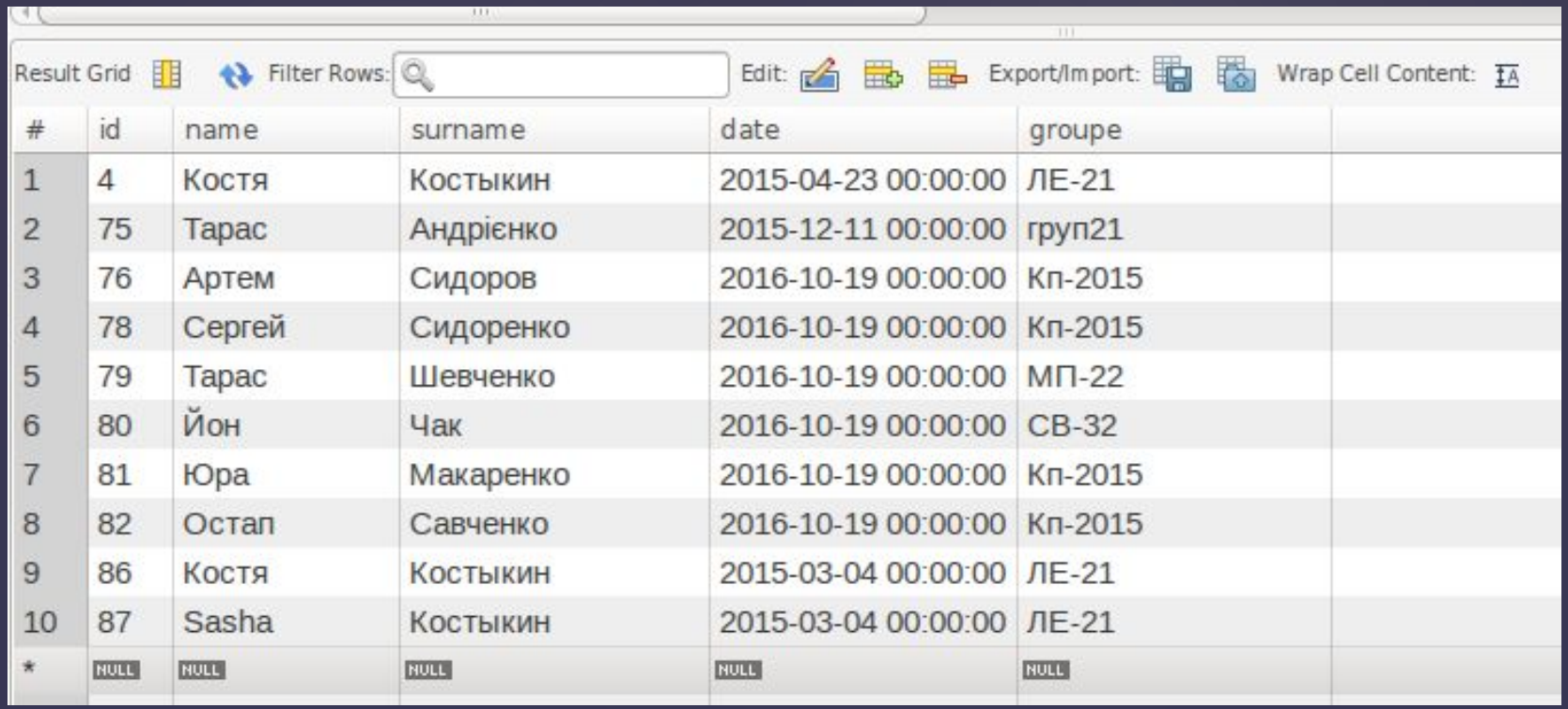
В таком варианте вы сразу видите, каким образом связаны между собой таблицы, можете группировать таблицы по каким-либо параметрами и отражать это на схеме. При этом визуальное проектирование удобно не только для того, чтобы кому-то рассказывать о проектируемой БД, но и для личного использования.

Программа имеет встроенный редактор SQL-кода, с помощью которого можно быстро внести любые правки в SQL-запросы. При этом возможно строить запросы любой сложности, получать различные выборки из таблиц, связывать их, создавать новые таблицы и редактировать существующие, работать с ключами, полями, связями. Одним словом — полноценный SQL-редактор.

MySQL Workbench позволяет осуществлять синхронизацию локальной схемы БД с реальной базой на вашем локальном или рабочем сервере. Благодаря этому после проектирования не требуется вручную создавать таблицы в базе на вашем сервере, достаточно сделать несколько простых действий в программе, после чего на рабочем сервере будет создана полноценная база данных со всеми указанными связями и параметрами.

Таблицы данных SQL

БД чаще всего содержат одну или несколько таблиц. Каждая ячейка идентифицируется по названию (например, "Friends" (Друзья) или "Orders" (Заказы)). Таблицы содержат записи с данными. Ниже представлена таблица, названная "**students**" :



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

#	id	name	surname	date	groupe
1	4	Костя	Костыкин	2015-04-23 00:00:00	ЛЕ-21
2	75	Тарас	Андрієнко	2015-12-11 00:00:00	груп21
3	76	Артем	Сидоров	2016-10-19 00:00:00	Кп-2015
4	78	Сергей	Сидоренко	2016-10-19 00:00:00	Кп-2015
5	79	Тарас	Шевченко	2016-10-19 00:00:00	МП-22
6	80	Йон	Чак	2016-10-19 00:00:00	СВ-32
7	81	Юра	Макаренко	2016-10-19 00:00:00	Кп-2015
8	82	Остап	Савченко	2016-10-19 00:00:00	Кп-2015
9	86	Костя	Костыкин	2015-03-04 00:00:00	ЛЕ-21
10	87	Sasha	Костыкин	2015-03-04 00:00:00	ЛЕ-21
*	NULL	NULL	NULL	NULL	NULL

Запросы SQL

Каждое предложение SQL — это либо *запрос* данных из базы, либо обращение к базе данных, которое приводит к изменению данных в базе. В соответствии с тем, какие изменения происходят в базе данных, различают следующие типы запросов:

- запросы на создание или изменение в базе данных новых или существующих объектов (при этом в запросе описывается тип и структура создаваемого или изменяемого объекта);
- запросы на получение данных;
- запросы на добавление новых данных (записей);
- запросы на удаление данных;
- обращения к СУБД.

Основным объектом хранения реляционной базы данных является таблица, поэтому все SQL-запросы — это операции над таблицами. В соответствии с этим, запросы делятся на:

- запросы, оперирующие самими таблицами (создание и изменение таблиц);
- запросы, оперирующие с отдельными записями (или строками таблиц) или наборами записей.

Создание таблиц (CREATE TABLE)

Существует два способа создания таблиц:

1) большинство СУБД обладают визуальным интерфейсом для интерактивного создания таблиц и управление ими;

2) таблицами можно манипулировать, используя операторы SQL. Стоит отметить, что, когда вы используете интерактивный инструментарий СУБД, на самом деле вся работа выполняется операторами SQL, т.е. интерфейс сам создает эти команды незаметно для пользователя.

Для создания таблиц программным способом используют оператор CREATE TABLE. Для этого нужно указать следующие данные:

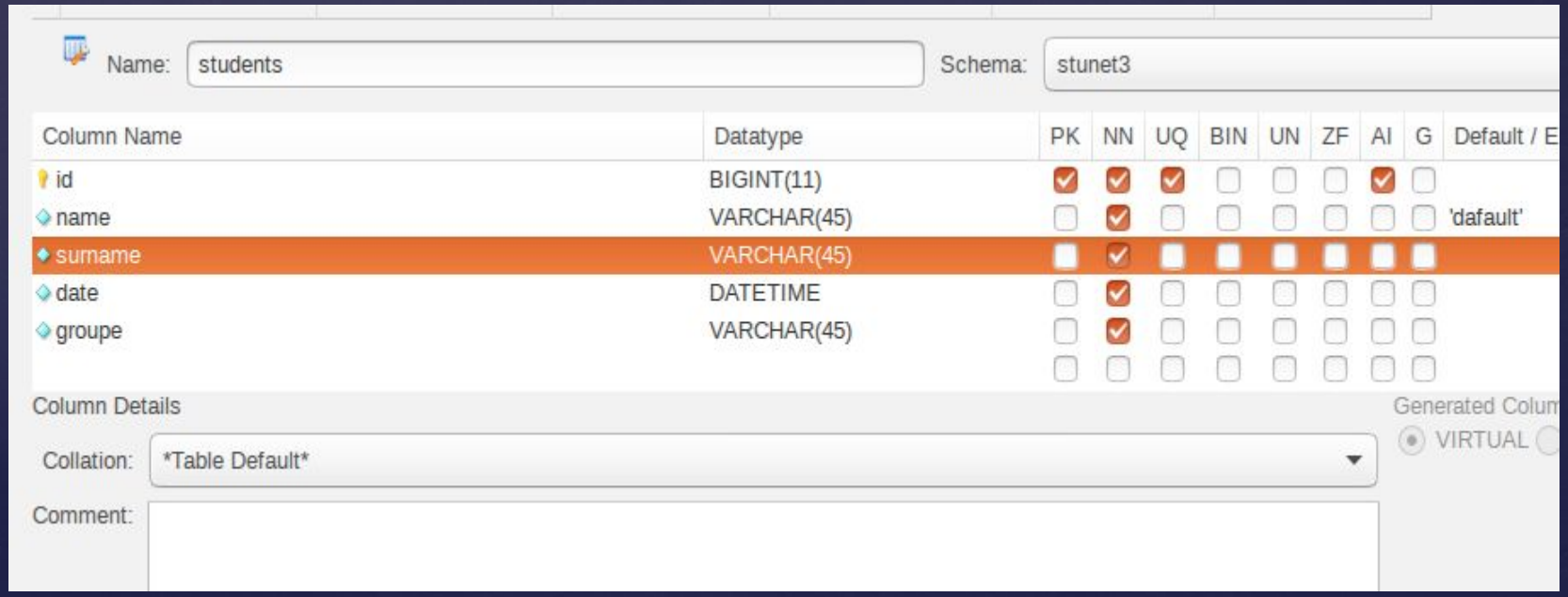
- имя таблицы, которое указывается после ключевого слова CREATE TABLE
- имена и определения столбцов таблицы, отделены запятыми
- в некоторых СУБД также требуется, чтобы было указано местоположение таблицы.

Давайте создадим новую таблицу и назовем ее students:

```
CREATE TABLE `students` (
  `id` bigint(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL DEFAULT 'dafault',
  `surname` varchar(45) NOT NULL,
  `date` datetime NOT NULL,
  `groupe` varchar(45) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=90 DEFAULT CHARSET=utf8;
```

Атрибуты полей

PK - primary key (данное поле является первичным ключом);
NN - not null (данное поле не может быть пустым);
UQ - unique index (данное поле должно содержать уникальные значения, т.е. неповторяющиеся);
BIN - binary (поле типа TEXT с учётом регистра символов при поиске значений хранящихся в нём);
UN - unsigned (целые беззнаковые значения - неотрицательные);
ZF - zero field (заполнение нулями). Этот атрибут добавляет к значению в строке нули слева, если длина значения меньше длины поля.
AI - auto incriment (счётчик++);



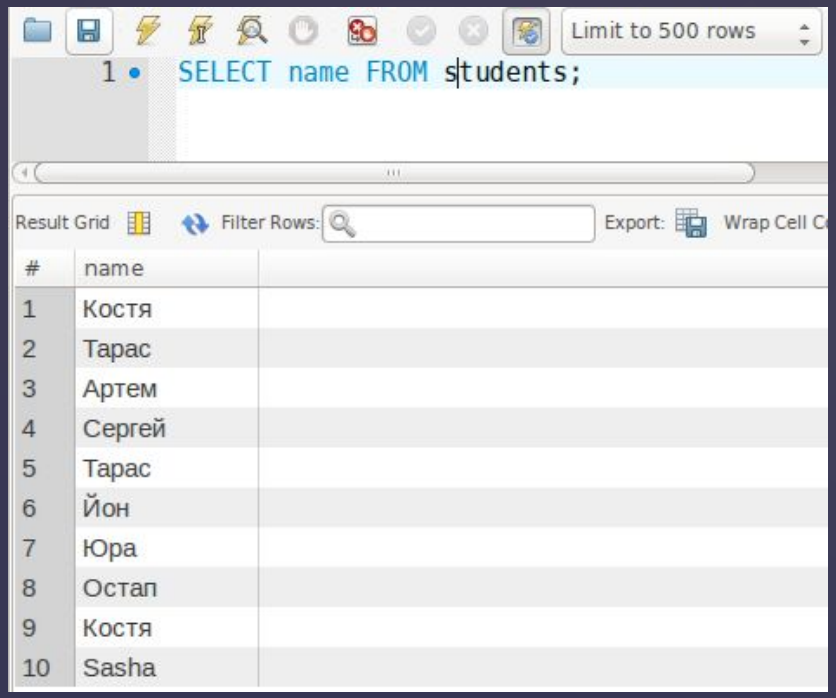
Выборка данных (SELECT)

Самым первым и главным оператором в SQL является SELECT. С его помощью мы можем отбирать необходимые нам поля данных в таблице.

1. Выборка отдельных полей.

SELECT name FROM students;

Видим, что наш SQL запрос отобрал колонку name из таблицы students.



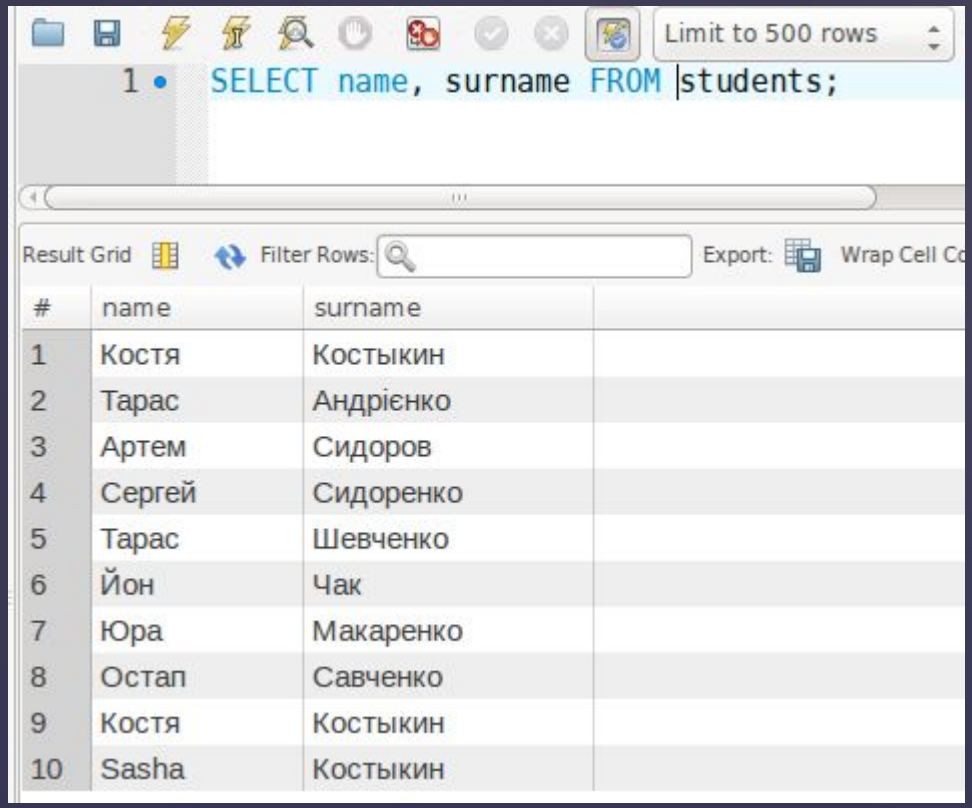
Выборка данных (SELECT)

2. Выборка нескольких полей.

Допустим, нам необходимо выбрать имя и фамилию. Для этого просто перечисляем необходимые поля через запятую:

SELECT name, surname FROM students;

Видим, что наш SQL запрос отобрал колонку name и surname из таблицы students.



#	name	surname
1	Костя	Костыкин
2	Тарас	Андрієнко
3	Артем	Сидоров
4	Сергей	Сидоренко
5	Тарас	Шевченко
6	Йон	Чак
7	Юра	Макаренко
8	Остап	Савченко
9	Костя	Костыкин
10	Sasha	Костыкин

3.Выборка всех столбцов.

Если же нам необходимо получить всю таблицу со всеми полями, тогда просто ставим знак звездочка (*):

SELECT * FROM students;

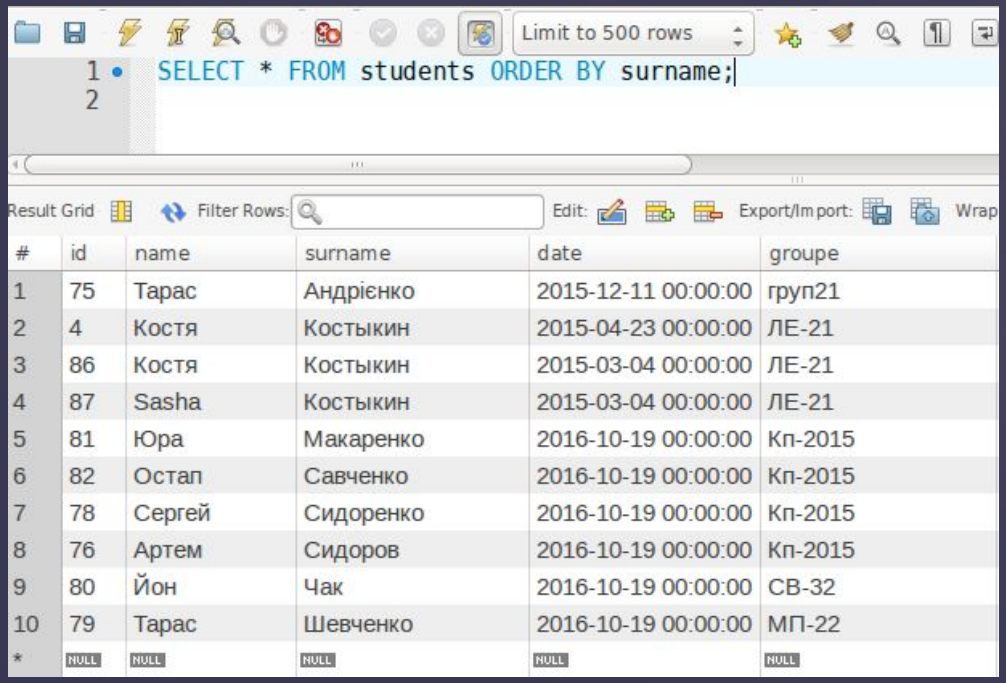
Все операторы в SQL нечувствительны к регистру, поэтому вы можете писать как большими буквами, так и маленькими (как правило, их принято писать большими буквами, чтобы различать от названий полей и таблиц). Названия же таблиц и полей является наоборот чувствительными к регистру и должны писаться точно как в БД.

Сортировка (ORDER BY)

В будущем нам может понадобиться сортировать нашу выборку - в алфавитном порядке для текста или по возрастанию/убыванию - для цифровых значений. Для таких целей в SQL есть специальный оператор **ORDER BY** .

Давайте всю нашу таблицу посортируем фамилии, а именно по столбцу surame.

SELECT * FROM students ORDER BY surname



#	id	name	surname	date	groupe
1	75	Тарас	Андрієнко	2015-12-11 00:00:00	груп21
2	4	Костя	Костыкин	2015-04-23 00:00:00	ЛЕ-21
3	86	Костя	Костыкин	2015-03-04 00:00:00	ЛЕ-21
4	87	Sasha	Костыкин	2015-03-04 00:00:00	ЛЕ-21
5	81	Юра	Макаренко	2016-10-19 00:00:00	Кп-2015
6	82	Остап	Савченко	2016-10-19 00:00:00	Кп-2015
7	78	Сергей	Сидоренко	2016-10-19 00:00:00	Кп-2015
8	76	Артем	Сидоров	2016-10-19 00:00:00	Кп-2015
9	80	Йон	Чак	2016-10-19 00:00:00	СВ-32
10	79	Тарас	Шевченко	2016-10-19 00:00:00	МП-22
*	NULL	NULL	NULL	NULL	NULL

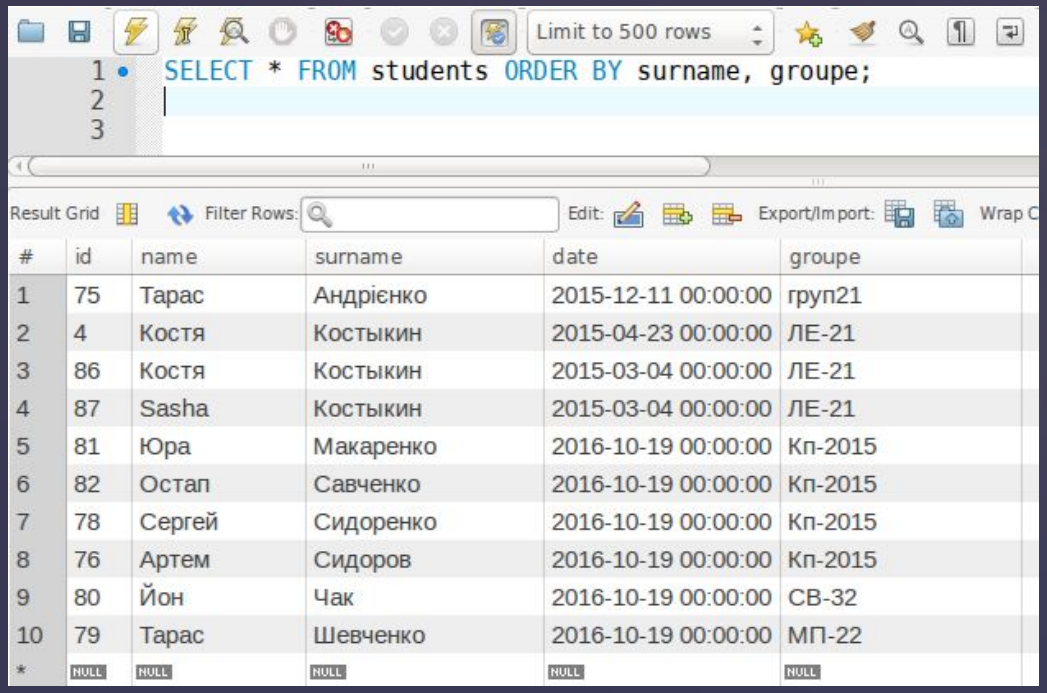
Видим, что запрос посортировал записи по возрастанию в поле surname.
Обязательно нужно соблюдать последовательность расположения операторов, т.е. оператор ORDER BY должен идти в самом конце запроса. В противном случае будет получено сообщение об ошибке.

Также особенностью оператора ORDER BY является то, что он может сортировать данные по полю, которого мы не выбирали в запросе, то есть достаточно, чтобы оно вообще было в БД.

Сортировка (ORDER BY)

Сортировка по нескольким полям.

Теперь посортируем наш пример дополнительно за еще одним полем. Пусть это будет поле groupe. **SELECT * FROM students ORDER BY surname, groupe;**



#	id	name	surname	date	groupe
1	75	Тарас	Андрієнко	2015-12-11 00:00:00	груп21
2	4	Костя	Костыкин	2015-04-23 00:00:00	ЛЕ-21
3	86	Костя	Костыкин	2015-03-04 00:00:00	ЛЕ-21
4	87	Sasha	Костыкин	2015-03-04 00:00:00	ЛЕ-21
5	81	Юра	Макаренко	2016-10-19 00:00:00	Кп-2015
6	82	Остап	Савченко	2016-10-19 00:00:00	Кп-2015
7	78	Сергей	Сидоренко	2016-10-19 00:00:00	Кп-2015
8	76	Артем	Сидоров	2016-10-19 00:00:00	Кп-2015
9	80	Йон	Чак	2016-10-19 00:00:00	СВ-32
10	79	Тарас	Шевченко	2016-10-19 00:00:00	МП-22
*	NULL	NULL	NULL	NULL	NULL

Очередность сортировки будет зависеть от порядка расположения полей в запросе. То есть, в нашем случае сначала данные будут рассортированы по колонке surname , а затем по groupe.

Несмотря на то, что по умолчанию оператор ORDER BY сортирует по возрастанию, мы можем также прописать сортировки значений по убыванию. Для этого в конце запроса проставляем оператор DESC (что является сокращением от слова DESCENDING).

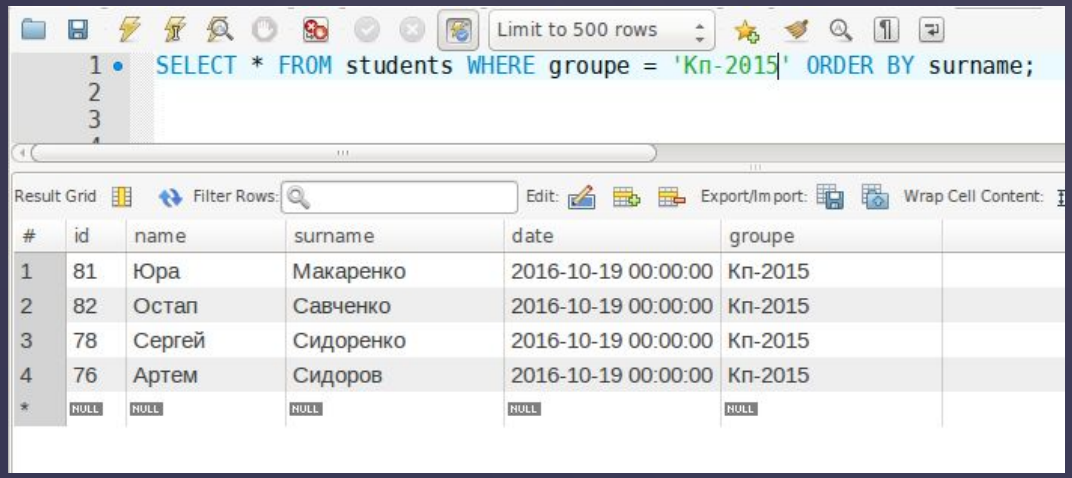
SELECT * FROM students ORDER BY surname DESC;

Фильтрация данных (WHERE)

В большинстве случаев необходимо получать не все записи, а только те, которые соответствуют определенным критериям. Поэтому для осуществления фильтрации выборки в SQL есть специальный оператор **WHERE.**

Давайте из нашей таблицы, например, отберем записи, относящиеся только к определенной группе. Для этого мы укажем дополнительный параметр отбора, который будет фильтровать значение по колонке **groupe**.

SELECT * FROM students WHERE groupe = 'Кп-2015' ORDER BY surname;



#	id	name	surname	date	groupe
1	81	Юра	Макаренко	2016-10-19 00:00:00	Кп-2015
2	82	Остап	Савченко	2016-10-19 00:00:00	Кп-2015
3	78	Сергей	Сидоренко	2016-10-19 00:00:00	Кп-2015
4	76	Артем	Сидоров	2016-10-19 00:00:00	Кп-2015
*	NULL	NULL	NULL	NULL	NULL

Как видим, условие отбора взято в одинарные кавычки, что является обязательным при фильтровании текстовых значений. При фильтровании числовых значений кавычки не нужны.

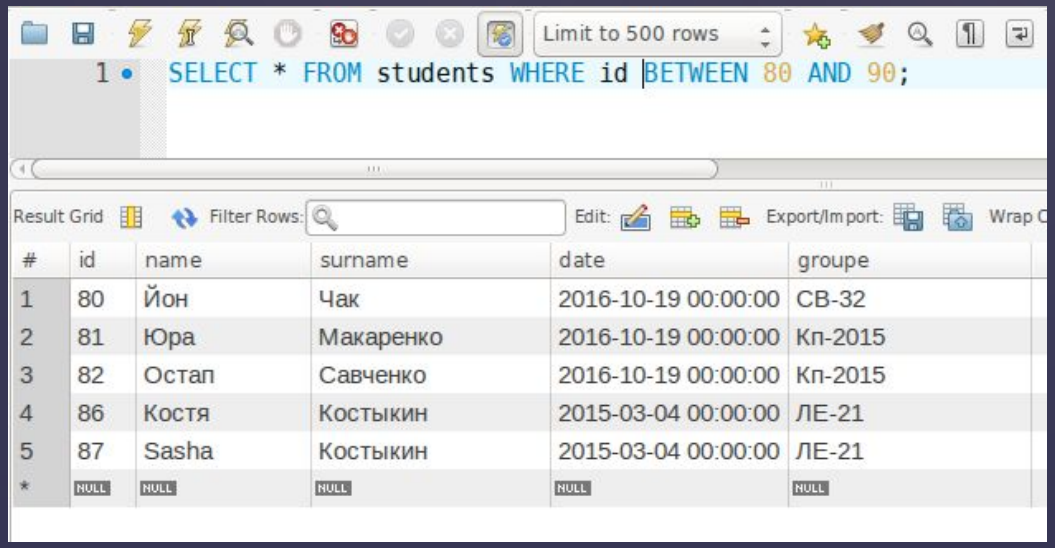
Фильтрация данных (WHERE)

В таблице ниже указан перечень условных операторов, поддерживаемых SQL:

Знак операции	Значение
=	Равно
<>	Не равно
<	Меньше
<=	Меньше или равно
>	Больше
>=	Больше или равно
BETWEEN	Между двумя значениями
IS NULL	Отсутствует запись

Фильтрация по диапазону значений (BETWEEN).

Для отбора данных, которые лежат в определенном диапазоне, используется оператор BETWEEN. В следующем запросе будут отобраны все значения, лежащие в пределах от 80 до 90 включительно, в поле **id**.

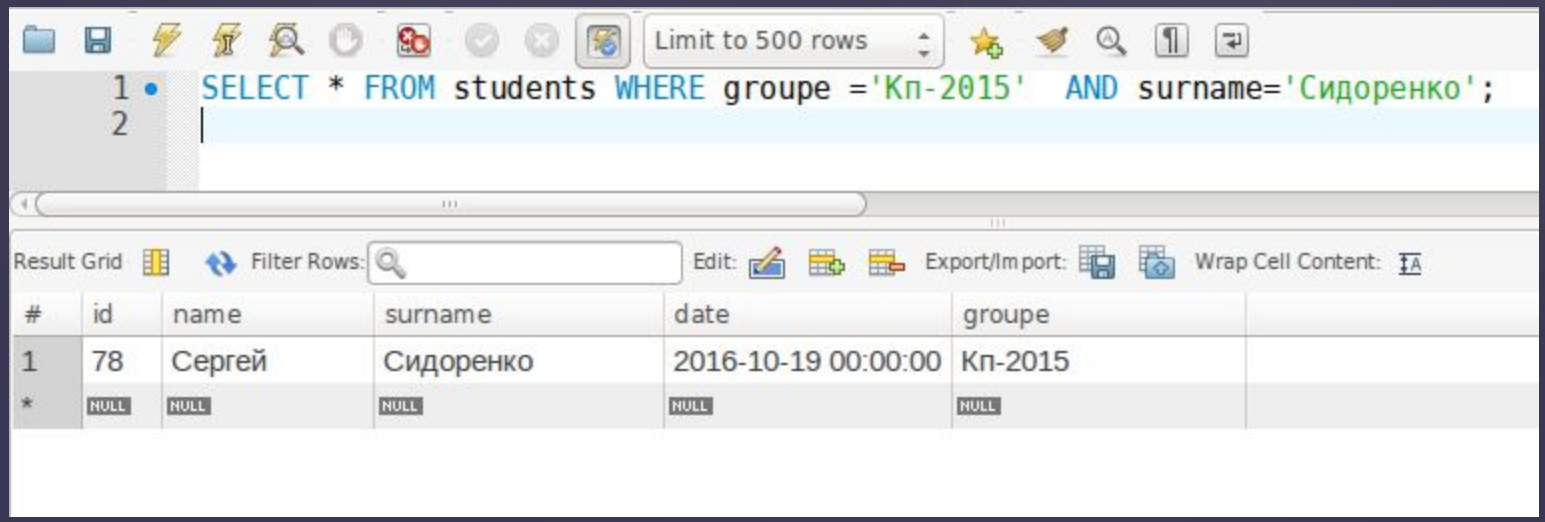


#	id	name	surname	date	groupe
1	80	Йон	Чак	2016-10-19 00:00:00	СВ-32
2	81	Юра	Макаренко	2016-10-19 00:00:00	Кп-2015
3	82	Остап	Савченко	2016-10-19 00:00:00	Кп-2015
4	86	Костя	Костыкин	2015-03-04 00:00:00	ЛЕ-21
5	87	Sasha	Костыкин	2015-03-04 00:00:00	ЛЕ-21
*	NULL	NULL	NULL	NULL	NULL

Расширенное фильтрации (AND, OR, IN, NOT).

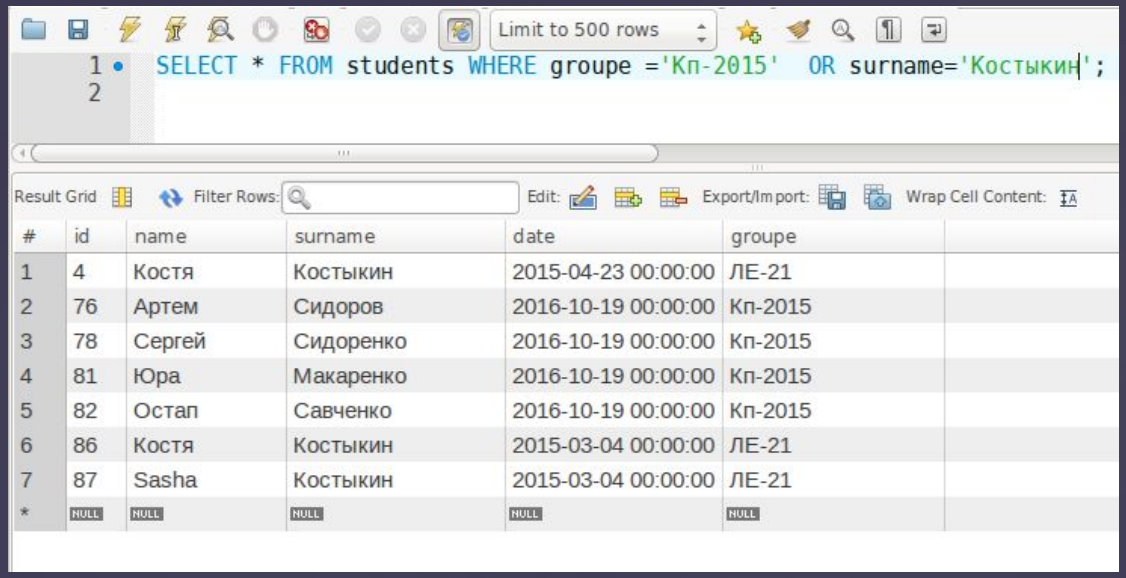
Язык SQL не ограничивается фильтрацией по одному условию, для собственных целей вы можете использовать достаточно сложные конструкции для выборки данных одновременно по многим критериям. Для этого в SQL есть дополнительные операторы, которые расширяют возможности оператора WHERE. Такими операторами являются: **AND, OR, IN, NOT**. Приведем несколько примеров работы данных операторов.

SELECT * FROM students WHERE groupe = 'Кп-2015' AND surname='Сидоренко';



Расширенное фильтрации (AND, OR, IN, NOT).

SELECT * FROM students WHERE groupe = 'Кп-2015' OR surname='Костыкин';



#	id	name	surname	date	groupe
1	4	Костя	Костыкин	2015-04-23 00:00:00	ЛЕ-21
2	76	Артем	Сидоров	2016-10-19 00:00:00	Кп-2015
3	78	Сергей	Сидоренко	2016-10-19 00:00:00	Кп-2015
4	81	Юра	Макаренко	2016-10-19 00:00:00	Кп-2015
5	82	Остап	Савченко	2016-10-19 00:00:00	Кп-2015
6	86	Костя	Костыкин	2015-03-04 00:00:00	ЛЕ-21
7	87	Sasha	Костыкин	2015-03-04 00:00:00	ЛЕ-21
*	NULL	NULL	NULL	NULL	NULL

Оператор IN выполняет ту же функцию, что и OR, однако имеет ряд преимуществ:

- При работе с длинными списками, предложение с IN легче читать;
- Используется меньшее количество операторов, что ускоряет обработку запроса;
- Самое важное преимущество IN в том, что в его конструкции можно использовать дополнительную конструкцию SELECT, что открывает большие возможности для создания сложных подзапросов.

SELECT * FROM students WHERE surname IN ('Савченко','Костыкин');

Ключевое слово NOT позволяет убрать ненужные значения из выборки. Также его особенностью является то, что оно проставляется перед названием столбца, участвующего в фильтровании, а не после.

SELECT * FROM students WHERE NOT surname IN ('Савченко','Костыкин');

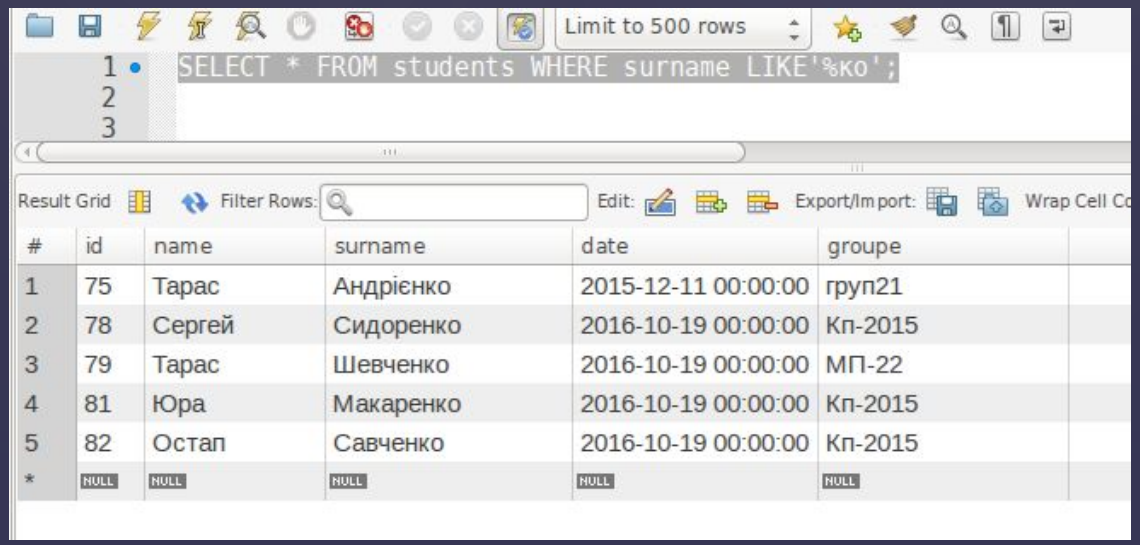
Символы подстановки и регулярные выражения (LIKE)

Часто, для фильтрации данных, нам нужно будет осуществить выборку не по точному совпадении условия, а по приближенному значению. То есть когда, например, мы ищем товар, название которого соответствует определенному шаблону или содержит определенные символы или слова. Для таких целей в SQL существует оператор LIKE, который ищет приближенные значения. Для конструирования такого шаблона используются **метасимволы** (специальные символы для поиска части значения).

символ подчеркивания (_) замещает любой одиночный символ. Например, 'b_t' будет соответствовать словам 'bat' или 'bit', но не будет соответствовать 'brat'.

знак процента (%) замещает последовательность любого числа символов (включая символы нуля). Например '%p%t' будет соответствовать словам 'put', 'posit', или 'opt', но не 'spite'.

SELECT * FROM students WHERE surname LIKE '%ко';



#	id	name	surname	date	groupe
1	75	Тарас	Андрієнко	2015-12-11 00:00:00	груп21
2	78	Сергей	Сидоренко	2016-10-19 00:00:00	Кп-2015
3	79	Тарас	Шевченко	2016-10-19 00:00:00	МП-22
4	81	Юра	Макаренко	2016-10-19 00:00:00	Кп-2015
5	82	Остап	Савченко	2016-10-19 00:00:00	Кп-2015
*	NULL	NULL	NULL	NULL	NULL

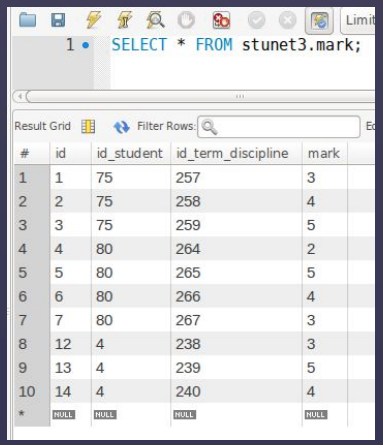
Статистические функции SQL

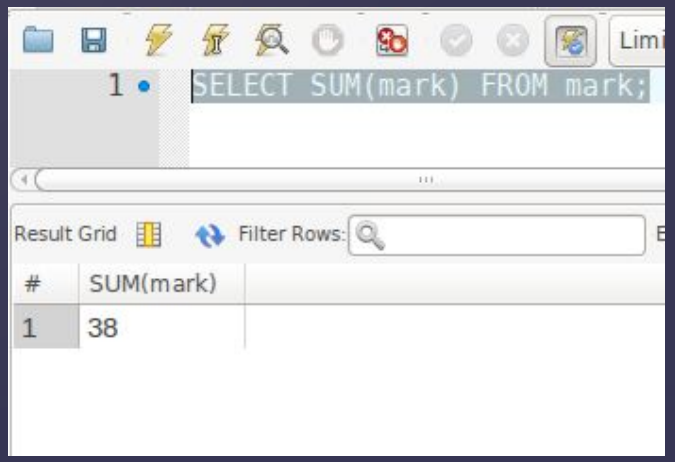
Статистические функции помогают нам получить готовые данные без их выборки. SQL-запросы с этими функциями часто используются для анализа и создания различных отчетов. Примером таких выборок может быть: определение количества строк в таблице, получение суммы значений по определенному полю, поиск наибольшего /наименьшего или среднего значения в указанном столбце таблицы. Также отметим, что статистические функции поддерживаются всеми СУБД без особых изменений в написании.

Знак операции	Значение
COUNT()	Возвращает число строк в таблице или столбце
SUM()	Возвращает сумму значений в столбце
MIN()	Возвращает наименьшее значение в столбце
MAX()	Возвращает наибольшее значение в столбце
AVG()	Возвращает среднее значение в столбце

SELECT COUNT(*) FROM students; - возвращает количество всех строк в таблице

SELECT SUM(mark) FROM mark; - возвращает суму всех чисел из колонки mark таблицы mark;





Объединение таблиц (JOIN)

Наиболее мощной особенностью языка SQL есть возможность сочетать различные таблицы в оперативной памяти СУБД при выполнении запросов. Объединение очень часто используются для анализа данных. Как правило, данные находятся в разных таблицах, что позволяет их более эффективно хранить (поскольку информация НЕ дублируется), упрощает обработку данных и позволяет масштабировать базу данных (возможно добавлять новые таблицы с дополнительной информацией).

С помощью инструкции JOIN можно объединить колонки из нескольких таблиц в одну. При этом целостность таблиц не нарушается. Существует три типа JOIN-выражений:

INNER JOIN применяется для получения только тех строк, для которых существует соответствие записей в главной и присоединяемой таблице. Иначе говоря, условие condition должно выполняться всегда.

OUTER JOIN

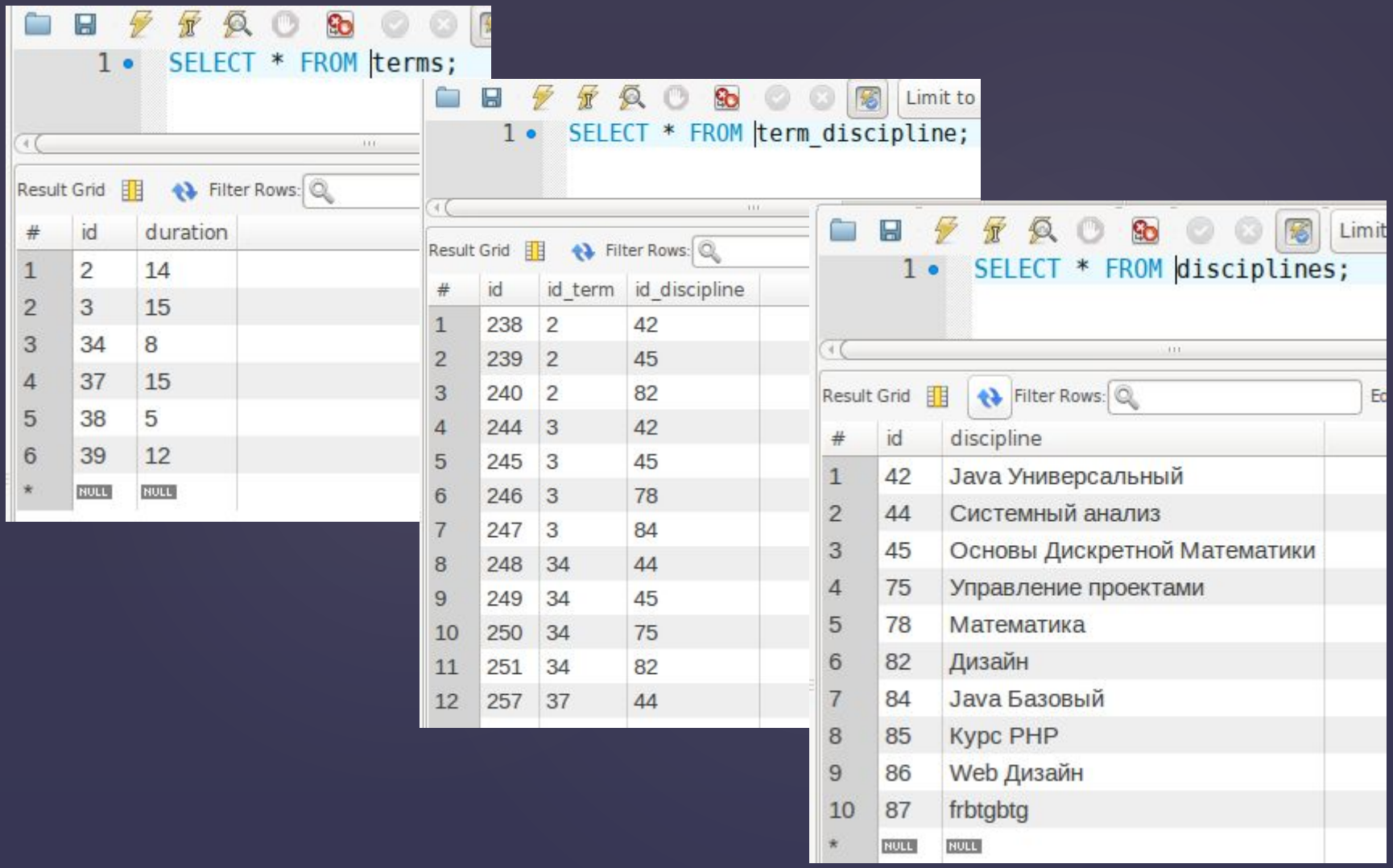
В свою очередь, OUTER JOIN может быть LEFT и RIGHT (слово OUTER часто опускается).

В случае с **LEFT JOIN** из главной таблицы будут выбраны все записи, даже если в присоединяемой таблице нет совпадений, то есть условие condition не будет учитывать присоединяемую

CROSS JOIN применяется если необходимо получить все возможные сочетания из обеих таблиц. Этот вид объединения сдедует использовать с большой осторожностью, поскольку он снижает производительность и часто (что кстати видно из примера содержит избыточную информацию.

condition — условие по которому таблицы объеденяются.

Пример объединение таблиц



```
SELECT * FROM terms;
```

Result Grid | Filter Rows:

#	id	duration
1	2	14
2	3	15
3	34	8
4	37	15
5	38	5
6	39	12
*	NULL	NULL

```
SELECT * FROM term_discipline;
```

Result Grid | Filter Rows:

#	id	id_term	id_discipline
1	238	2	42
2	239	2	45
3	240	2	82
4	244	3	42
5	245	3	45
6	246	3	78
7	247	3	84
8	248	34	44
9	249	34	45
10	250	34	75
11	251	34	82
12	257	37	44

```
SELECT * FROM disciplines;
```

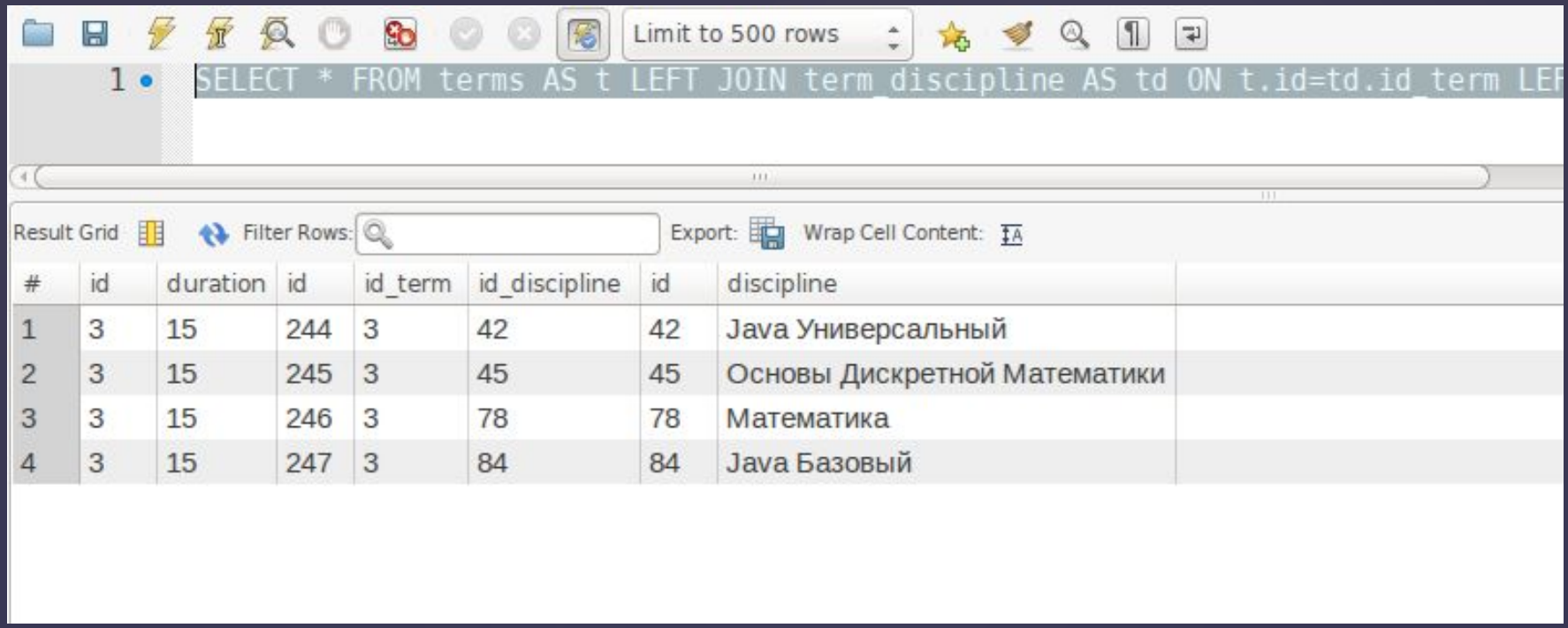
Result Grid | Filter Rows:

#	id	discipline
1	42	Java Универсальный
2	44	Системный анализ
3	45	Основы Дискретной Математики
4	75	Управление проектами
5	78	Математика
6	82	Дизайн
7	84	Java Базовый
8	85	Курс PHP
9	86	Web Дизайн
10	87	frbtgbtg
*	NULL	NULL

```
SELECT * FROM terms AS t LEFT JOIN term_discipline AS td ON t.id=td.id_term LEFT JOIN disciplines AS d ON td.id_discipline=d.id WHERE id_term=3;
```

Результат объединение таблиц

SELECT * FROM terms AS t LEFT JOIN term_discipline AS td ON t.id=td.id_term LEFT JOIN disciplines AS d ON td.id_discipline=d.id WHERE id_term=3;



#	id	duration	id	id_term	id_discipline	id	discipline
1	3	15	244	3	42	42	Java Универсальный
2	3	15	245	3	45	45	Основы Дискретной Математики
3	3	15	246	3	78	78	Математика
4	3	15	247	3	84	84	Java Базовый

Добавление данных (INSERT INTO)

INSERT - добавляет данные в таблицу.

Как видно из названия, оператор INSERT используется для вставки (добавления) строк в таблицу базы данных. Добавление можно осуществить несколькими способами:

- добавить одну полную строку
- добавить часть строки
- добавить результаты запроса.

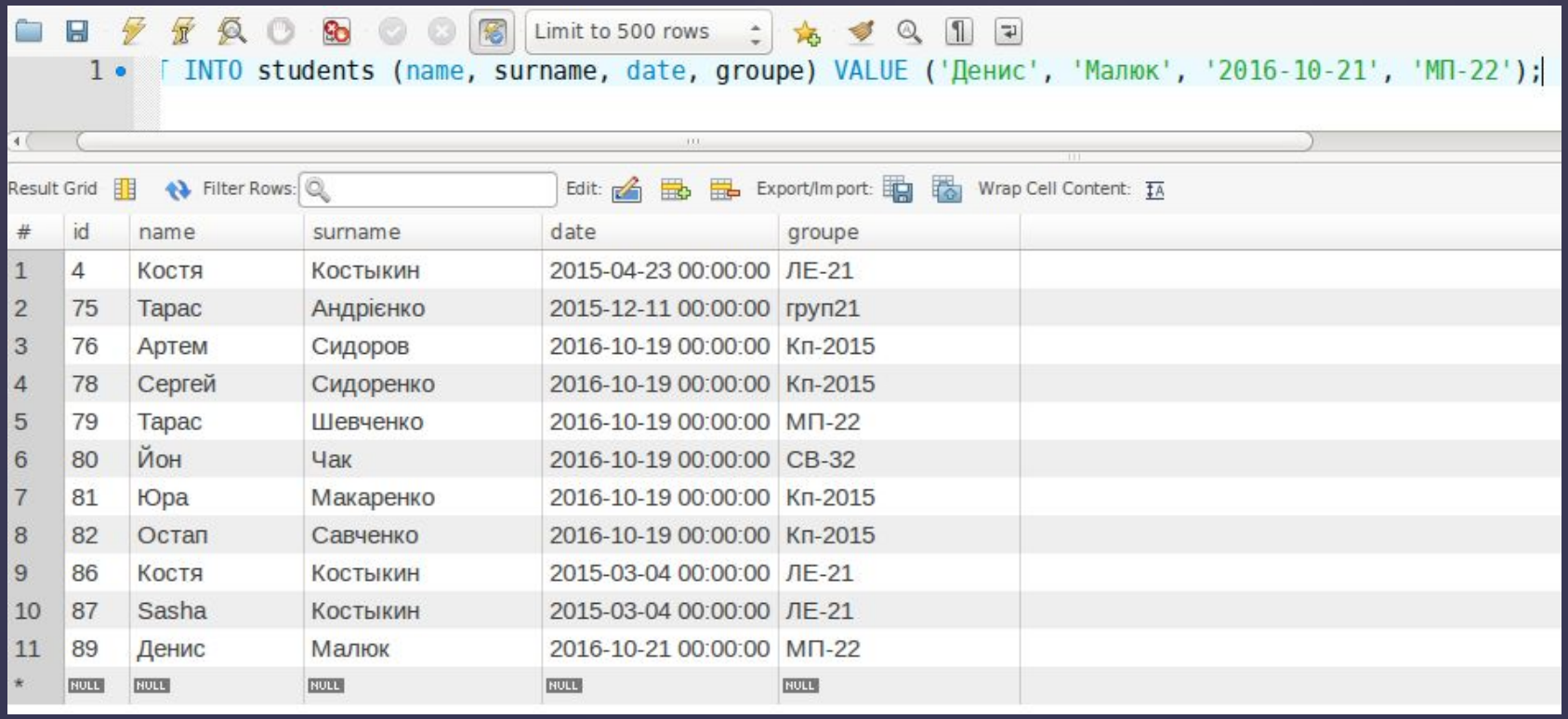
Итак, чтобы добавить новую строку в таблицу, нам необходимо указать название таблицы, перечислить названия колонок и указать значение для каждой колонки с помощью конструкции INSERT INTO название_таблицы (поле1, поле2 ...) VALUES (значение1, значение2 ...). Рассмотрим на примере.

INSERT INTO students (name, surname, date, groupe) VALUE ('Денис', 'Малюк', '2016-10-21', 'МП-22');

Используя данный синтаксис, мы можем пропустить некоторые столбцы. Это значит, что вы вводите значение для одних столбцов но не предлагаете их для других.
Это означает, что, если не указано никакое значение, будет использовано значение по умолчанию. Если вы пропускаете столбец таблицы, которая не допускает появления в своих строках значений NULL и не имеет значения, определенного для использования по умолчанию, СУБД выдаст сообщение об ошибке, и это строка не будет добавлена.

Результат добавление данных

INSERT INTO students (name, surname, date, groupe) VALUE ('Денис', 'Малюк', '2016-10-21', 'МП-22');



#	id	name	surname	date	groupe
1	4	Костя	Костыкин	2015-04-23 00:00:00	ЛЕ-21
2	75	Тарас	Андрієнко	2015-12-11 00:00:00	груп21
3	76	Артем	Сидоров	2016-10-19 00:00:00	Кп-2015
4	78	Сергей	Сидоренко	2016-10-19 00:00:00	Кп-2015
5	79	Тарас	Шевченко	2016-10-19 00:00:00	МП-22
6	80	Йон	Чак	2016-10-19 00:00:00	СВ-32
7	81	Юра	Макаренко	2016-10-19 00:00:00	Кп-2015
8	82	Остап	Савченко	2016-10-19 00:00:00	Кп-2015
9	86	Костя	Костыкин	2015-03-04 00:00:00	ЛЕ-21
10	87	Sasha	Костыкин	2015-03-04 00:00:00	ЛЕ-21
11	89	Денис	Малюк	2016-10-21 00:00:00	МП-22
*	NULL	NULL	NULL	NULL	NULL

Обновление и удаление данных из таблицы

Оператор **UPDATE** обновляет столбцы в соответствии с их новыми значениями в строках существующей таблицы.

В выражении **SET** указывается, какие именно столбцы следует модифицировать и какие величины должны быть в них установлены. В выражении **WHERE**, если оно присутствует, задается, какие строки подлежат обновлению. В остальных случаях обновляются все строки.

```
UPDATE students SET name='Mark' WHERE id='90';
```

Оператор **DELETE** - удаляет данные из таблицы.

Как видно из названия, оператор DELETE используется для удаления строк из таблицы базы данных удовлетворяющие заданным в WHERE условиям, и возвращает число удаленных записей.

```
DELETE FROM students WHERE id='89';
```

Если оператор DELETE запускается без определения WHERE, то удаляются все строки.

Внешние ключи (foreign keys)

Внешние ключи — это как раз те связующие цепочки, которые связывают таблицы между собой. Они позволяют вам разместить «студентов» в одной таблице, «дисциплины» в другой, а оценки по этим дисциплинам, в третьей, таким образом в базе минимизируется избыточность данных.

Просто добавив объявления внешних ключей, мы добились встроенной защиты целостности данных. Если мы попытаемся выполнить запрос INSERT или UPDATE со значением внешнего ключа для таблицы invoice, база данных автоматически проверит существует ли данное значение в связанной таблице. Если указанных значений в связанных таблицах не существует — база данных не выполнит запрос INSERT/UPDATE, сохранив таким образом целостность данных.

Теперь не придется проверять вручную родительскую таблицу на существование конкретных значений, прежде чем вставить данные в таблицу-потомок. Также можете спокойно удалять записи. Хотите избежать ошибок новым способом? Меньше кодирования — лучший способ для ленивых программистов.

MySQL позволяет нам контролировать таблицы-потомки во время обновления или удаления данных в родительской таблице с помощью подвыражений: ON UPDATE и ON DELETE. MySQL поддерживает 5 действий, которые можно использовать в выражениях ON UPDATE и/или ON DELETE.

CASCADE: если связанная запись родительской таблицы обновлена или удалена, и мы хотим чтобы соответствующие записи в таблицах-потомках также были обновлены или удалены. Что происходит с записью в родительской таблице, тоже самое произойдет с записью в дочерних таблицах. Однако не забывайте, что здесь можно легко попасться в ловушку бесконечного цикла.

RESTRICT:если связанные записи родительской таблицы обновляются или удаляются со значениями которые уже/еще содержатся в соответствующих записях дочерней таблицы, то база данных не позволит изменять записи в родительской таблице. Обе команды NO ACTION и RESTRICT эквивалентны отсутствию

подвыражений ON UPDATE or ON DELETE для внешних ключей.