ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

MySQL. MySQL Workbench

Справочные материалы

САМАРА 2022

СОДЕРЖАНИЕ

1. MYSQL WORKBENCH

1.1 Стартовая страница

MySQL – свободная реляционная система управления базами данных (БД). Полная документация на английском языке доступна на сайте https://dev.mysql.com/doc/

В лабораторных работах для работы с MySQL будет использоваться MySQL Workbench – инструмент для проектирования и эксплуатации баз данных MySQL – версии Community Edition.

Стартовая страница MySQL Workbench изображена на рисунке 1.1. На рисунке вкладка 1 – страница подключений к БД (connections), вкладка 2 – страница моделей (models), панель 3 – настроенное подключение к БД.

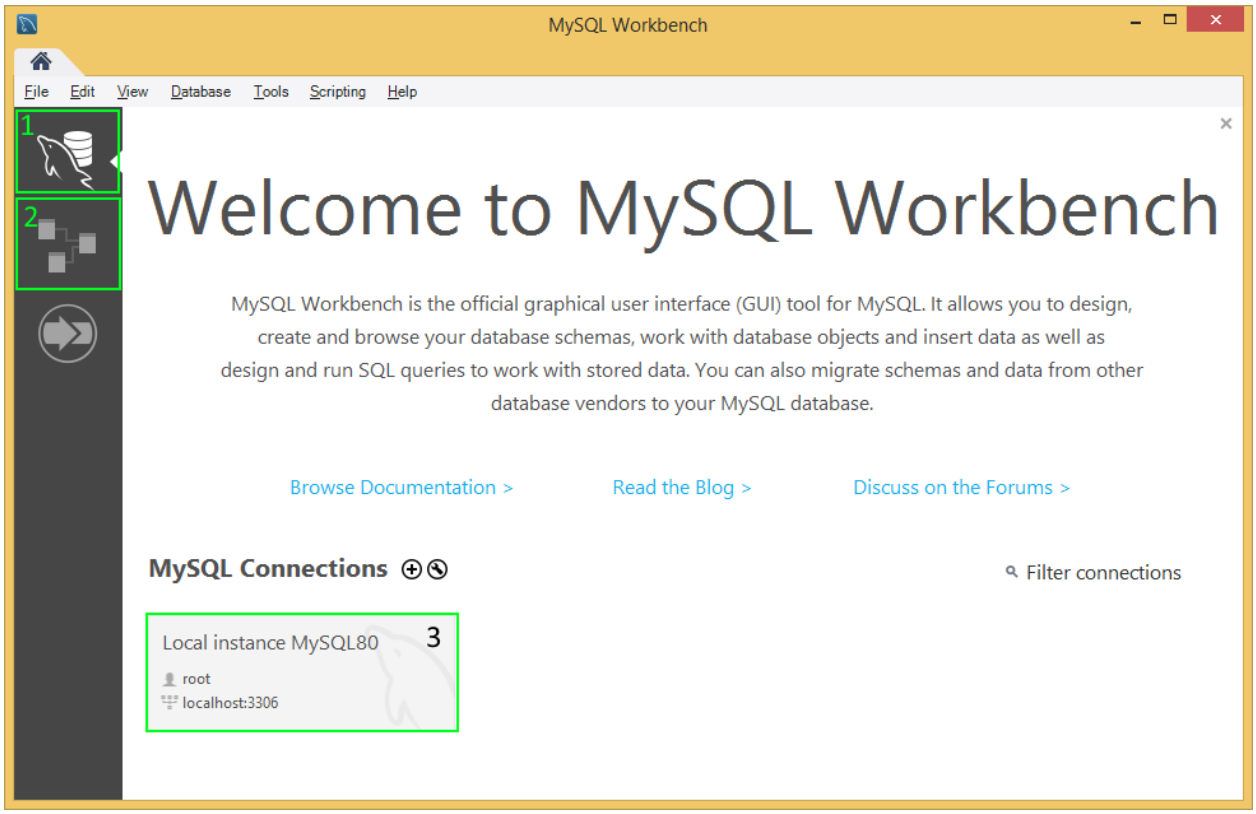


Рисунок 1.1 – Стартовая страница MySQL Workbench.

1.2 Редактор SQL-запросов

После выбора подключения (панель 3 на рисунке 1.1) и указания логина / пароля для доступа к БД (по умолчанию, ***student / P@ssw0rd***) откроется редактор SQL-запросов (SQL editor) (рисунок 1.2).

Вкладка «Администрирование» («Administration») предоставляет возможности по администрированию и анализу производительности сервера, вкладка «Схемы» («Schemas») отображает схемы (базы данных в MySQL), ассоциированные с выбранным подключением.

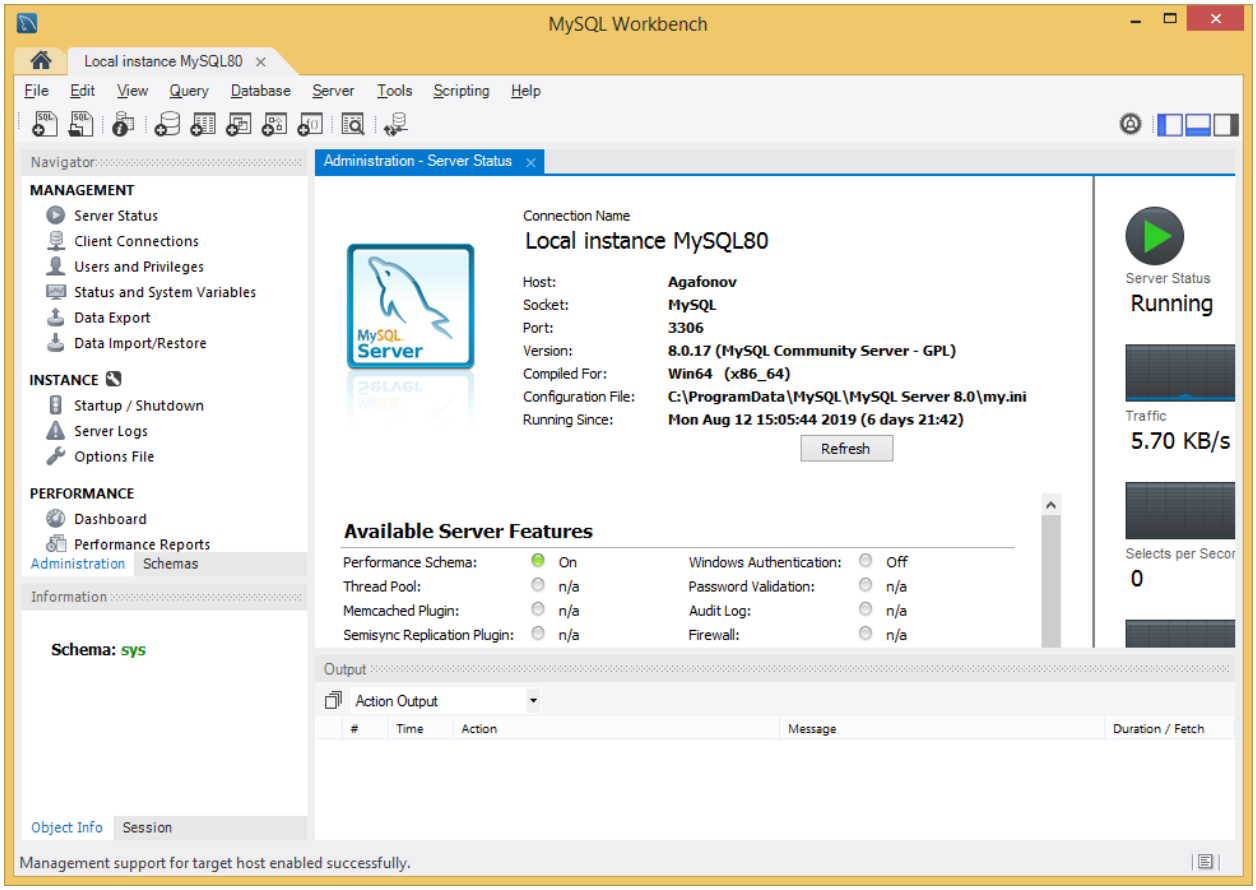


Рисунок 1.2 – Редактор запросов.

Рассмотрим подробнее основные элементы редактора SQL-запросов (рисунок 1.3):

1. Окно подключений к серверу.
2. Вкладка SQL-запросов: основное окно, предназначенное для выполнения SQL-запросов пользователя.
3. Меню взаимодействия с БД: создать / открыть новую вкладку запросов, создать схему и объекты базы данных (таблицы, представления и т.д.).
4. Вкладка «Схемы», предоставляющая доступ к схемам и объектам схем.
5. Помощь: краткое описание синтаксиса основных SQL операций.
6. Окно вывода основной информации о выполненных запросах.

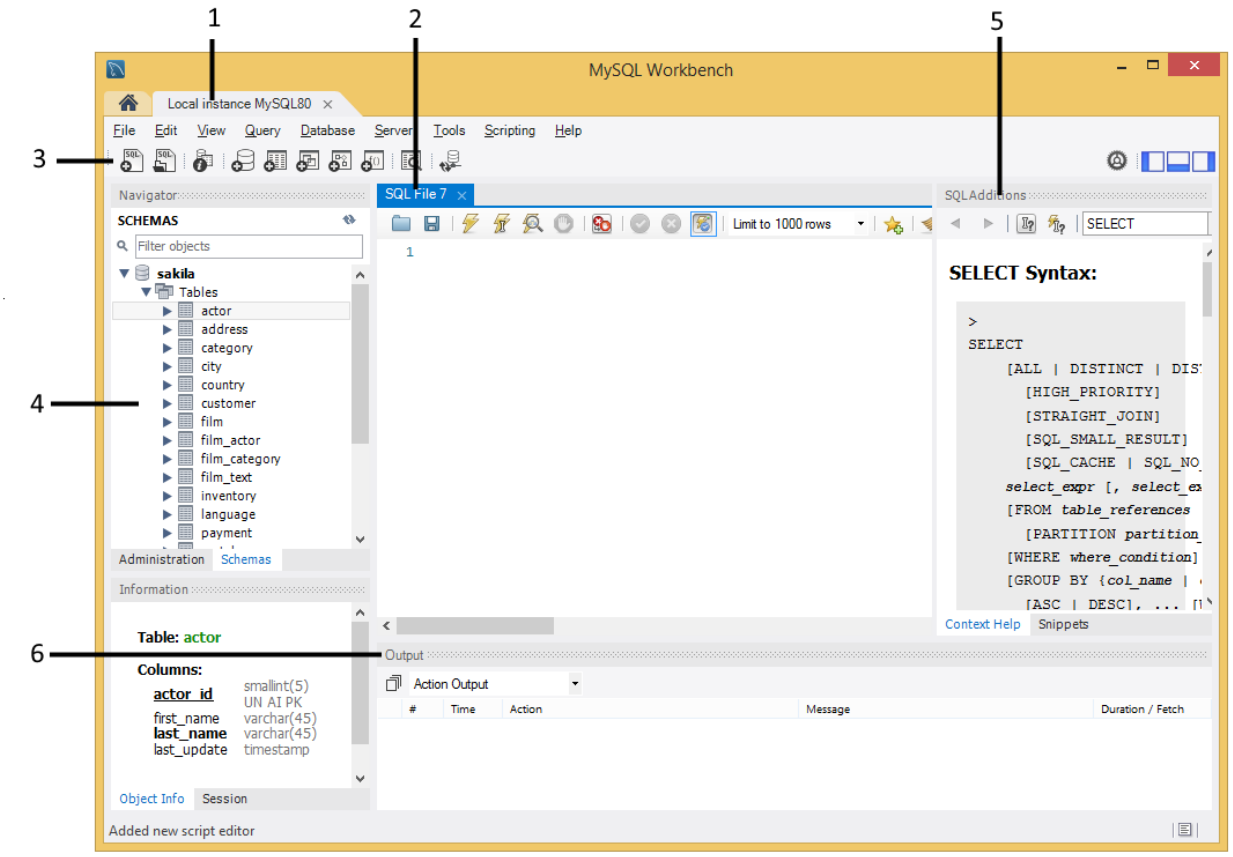


Рисунок 1.3 – Элементы редактора SQL-запросов.

1.3 Выполнение SQL-запроса

Для выполнения запроса в окне «Схемы» необходимо выбрать требуемую БД (двойным кликом по названию), во вкладке SQL-запросов написать исполняемый текст запроса, нажать кнопку «Выполнить запрос» (рисунок 1.4).

На вкладке «Представление результатов» можно изменить способ отображения результатов запроса, статистику выполнения запроса, а также план выполнения запроса.

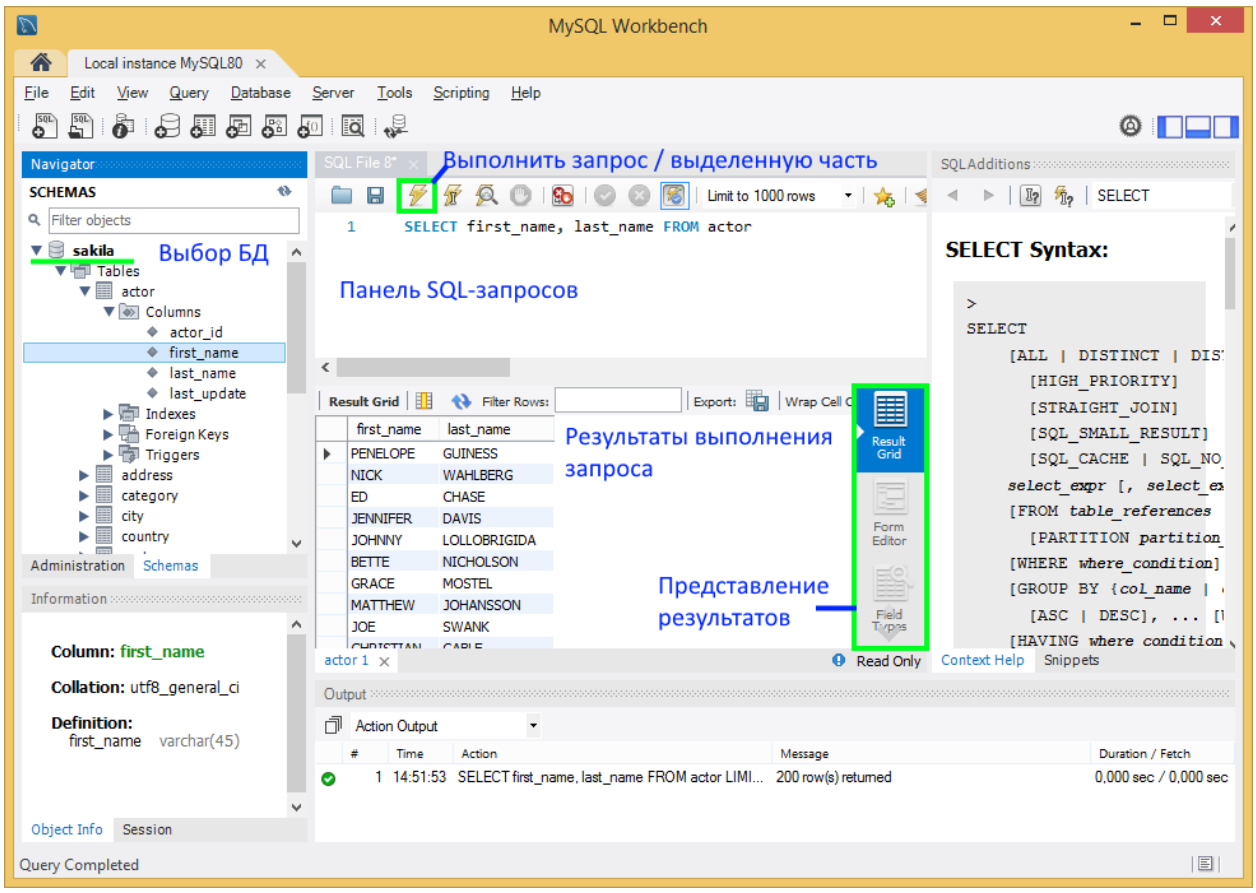


Рисунок 1.4 – Выполнение запроса.

1.4 Создание модели БД

Страница модели БД открывается со стартовой страницы MySQL Workbench (рисунок 1.1) Пример модели БД sakila показан на рисунке 1.5.

Элементы БД (таблицы, представления и т.д.) могут быть созданы как с использованием панели «Physical schemas», так и на EER-диаграмме.

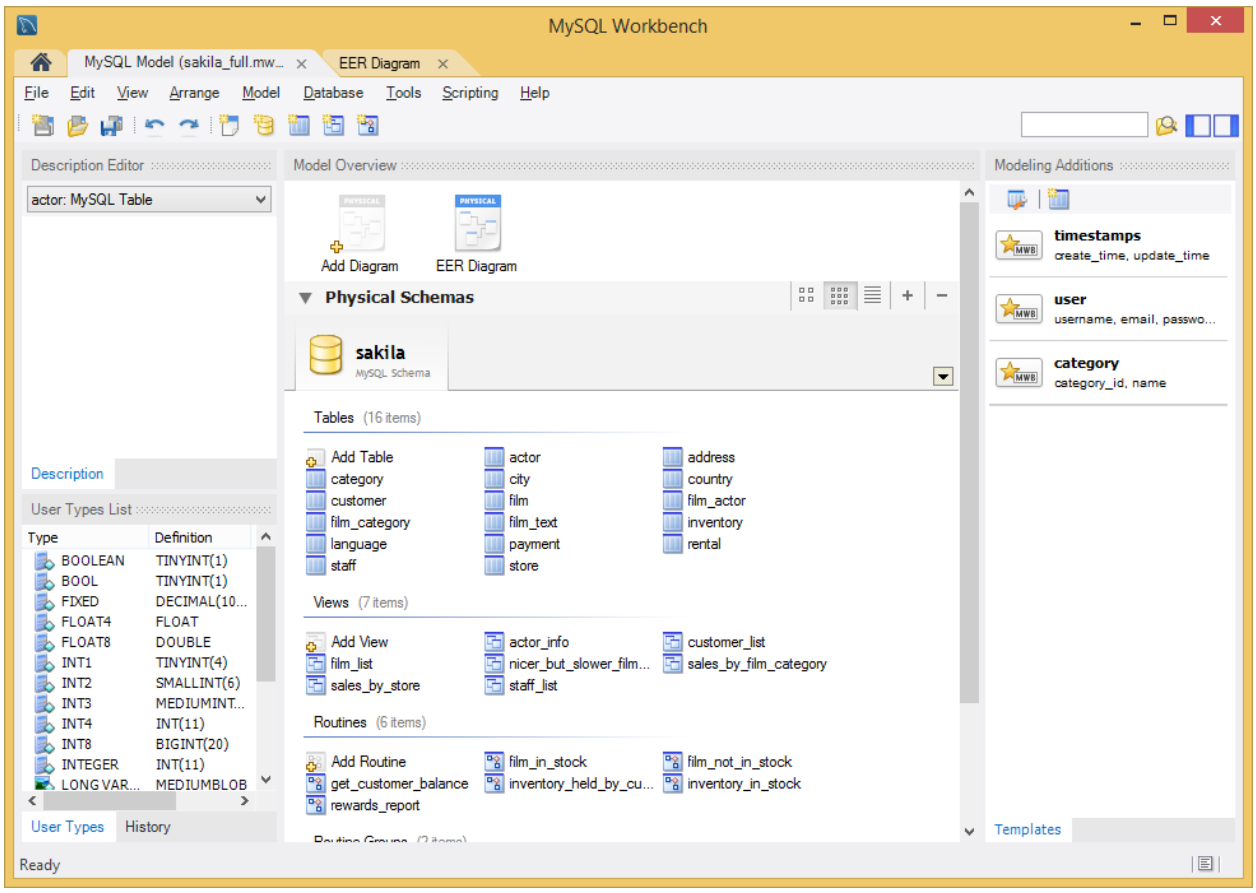


Рисунок 1.5 – Модель БД.

2. КРАТКОЕ ОПИСАНИЕ ОПЕРАТОРОВ SQL

2.1 Оператор SELECT

Упрощенный синтаксис оператора SELECT:

```
SELECT
    [ALL | DISTINCT]
    select_expr [, select_expr ...]
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
```

SELECT применяется для извлечения строк, выбранных из одной или нескольких таблиц. Выражение *select_expr* задает столбцы, в которых необходимо проводить выборку. Если требуется получить только уникальные строки, то можно использовать ключевое слово DISTINCT. Помимо DISTINCT может применяться также ключевое слово ALL (все строки), которое принимается по умолчанию. Кроме того, оператор SELECT можно использовать для извлечения строк, вычисленных без ссылки на какую-либо таблицу. Например:

```
mysql> SELECT 1 + 1;
    -> 2
```

Выражение FROM *table_references* задает таблицы, из которых надлежит извлекать строки. Если указано имя более чем одной таблицы, следует выполнить объединение (JOIN).

Выражение WHERE, если оно задано, указывает условие или условия, которым должны удовлетворять строки. *where_condition* - это выражение, которое оценивается как истинное для каждой выбранной строки. Оператор SELECT выбирает все строки, если нет предложения WHERE.

Выражение GROUP BY используется для определения групп выходных строк, к которым могут применяться агрегатные функции (COUNT, MIN, MAX, AVG и SUM). Если это предложение отсутствует, и используются агрегатные функции, то все столбцы с именами, упомянутыми в SELECT, должны быть включены в

агрегатные функции, и эти функции будут применяться ко всему набору строк, которые удовлетворяют предикату запроса. Если при наличии предложения GROUP BY, в предложении SELECT отсутствуют агрегатные функции, то запрос просто вернет по одной строке из каждой группы. Эту возможность, наряду с ключевым словом DISTINCT, можно использовать для исключения дубликатов строк в результирующем наборе. Для получения значения функции агрегации, примененной к множеству полученных групп, в новой строке используется выражение WITH ROLLUP.

Выражение HAVING применяется после группировки для фильтрации групп по значениям агрегатных функций.

Выражение ORDER BY выполняет сортировку по любому количеству полей, указанных в предложении SELECT. При этом в списке полей могут указываться как имена полей, так и их порядковые позиции в списке предложения SELECT. Сортировку можно проводить по возрастанию (параметр ASC принимается по умолчанию) или по убыванию (параметр DESC).

Выражение LIMIT может использоваться для ограничения количества строк, возвращенных командой SELECT. LIMIT принимает один или два числовых аргумента. Эти аргументы должны быть целочисленными константами. Если заданы два аргумента, то первый указывает на смещение относительно первой возвращаемой строки, а второй задает максимальное количество возвращаемых строк.

2.2 Оператор JOIN

Синтаксис:

```
table_reference, table_reference
table_reference [CROSS] JOIN table_reference
table_reference INNER JOIN table_reference join_condition
table_reference STRAIGHT_JOIN table_reference
table_reference LEFT [OUTER] JOIN table_reference join_condition
table_reference LEFT [OUTER] JOIN table_reference
table_reference NATURAL [LEFT [OUTER]] JOIN table_reference
table_reference RIGHT [OUTER] JOIN table_reference join_condition
table_reference RIGHT [OUTER] JOIN table_reference
table_reference NATURAL [RIGHT [OUTER]] JOIN table_reference
```

где `table_reference` определено, как:

```
table_name [[AS] alias]
```

и `join_condition` определено, как:

```
ON conditional_expr |
USING (column_list)
```

Оператор JOIN позволяет извлекать данные из нескольких таблиц без создания временных таблиц и за один запрос.

Условный оператор ON представляет собой условие в любой форме из числа тех, которые можно использовать в выражении WHERE.

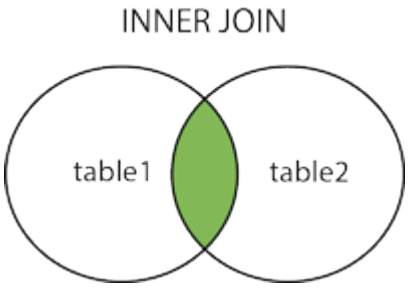
USING (column_list) служит для указания списка столбцов, которые должны существовать в обеих таблицах. Такое выражение USING, как:

```
A LEFT JOIN B USING (C1, C2, C3, ...)
```

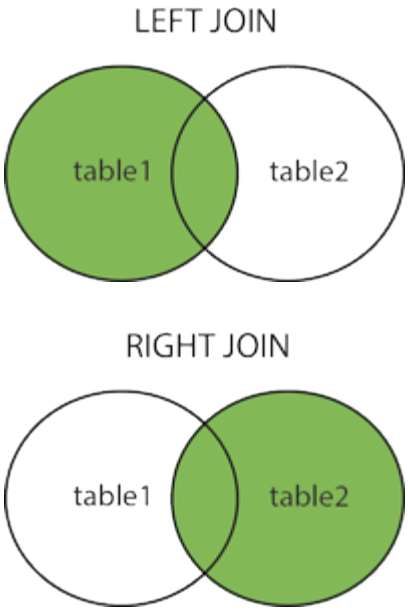
семантически идентично выражению ON, например:

```
A LEFT JOIN B ON A.C1=B.C1 AND A.C2=B.C2 AND A.C3=B.C3,...
```

При внутреннем объединении (INNER JOIN) выбираются только совпадающие данные из объединяемых таблиц.



Чтобы получить данные, которые подходят по условию частично, необходимо использовать внешнее объединение - OUTER JOIN. Такое объединение вернет данные из обеих таблиц (совпадающие по условию объединения) ПЛЮС дополнит выборку оставшимися данными из внешней таблицы, которые по условию не подходят, заполнив недостающие данные значением NULL. Существует два типа внешнего объединения OUTER JOIN - LEFT OUTER JOIN и RIGHT OUTER JOIN.



Выражение NATURAL [LEFT] JOIN для двух таблиц определяется так, чтобы оно являлось семантическим эквивалентом INNER JOIN или LEFT JOIN с выражением USING, в котором указаны все столбцы, имеющиеся в обеих таблицах.

INNER JOIN и , (запятая) являются семантическими эквивалентами. Оба осуществляют полное объединение используемых таблиц. Способ связывания таблиц обычно задается в условии WHERE.

STRAIGHT_JOIN идентично JOIN, за исключением того, что левая таблица всегда читается раньше правой. Это выражение может использоваться для тех (немногих) случаев, когда оптимизатор объединения располагает таблицы в неправильном порядке.

2.3 Оператор UNION

Синтаксис:

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

UNION используется для объединения результатов работы нескольких команд SELECT в один набор результатов. Каждое предложение SELECT в операторе

UNION должно иметь одинаковое количество полей в наборах результатов с одинаковыми типами данных.

Пример:

```
SELECT film.title,film.length from film WHERE film.title LIKE 'cat%'
UNION
SELECT film.title,film.length from film WHERE film.length < 47
```

Результат:

title	length
CAT CONEHEADS	112
CATCH AMISTAD	183
ALIEN CENTER	46
IRON MOON	46
KWAI HOMEWARD	46
LABYRINTH LEAGUE	46
RIDGEMONT SUBMARINE	46

2.4 Оператор INSERT

Упрощенный синтаксис оператора INSERT:

```
INSERT [IGNORE]
    [INTO] tbl_name
    [(col_name [, col_name] ...)]
    {VALUES | VALUE} (value_list) [, (value_list)] ...
    [ON DUPLICATE KEY UPDATE assignment_list]
ИЛИ
INSERT [IGNORE]
    [INTO] tbl_name
    SET assignment_list
    [ON DUPLICATE KEY UPDATE assignment_list]
ИЛИ
INSERT [IGNORE]
    [INTO] tbl_name
    [(col_name [, col_name] ...)]
    SELECT ...
    [ON DUPLICATE KEY UPDATE assignment_list]
```

Оператор INSERT вставляет новые строки в существующую таблицу. Форма данной команды INSERT... {VALUES | VALUE} вставляет строки в соответствии с точно указанными в команде значениями. При использовании формы данной команды INSERT... SET каждому полю, присутствующему в таблице, присваивается значение в виде имя поля = 'значение'. Форма INSERT... SELECT вставляет строки, выбранные из другой таблицы или таблиц.

Если некоторые поля таблицы имеют ключи PRIMARY или UNIQUE, и производится вставка новой строки, в которой эти поля имеют дублирующее значение, то действие команды аварийно завершается и выдается ошибка №1062 ("Duplicate entry 'val' for key N"). Если в команде INSERT указано ключевое слово IGNORE, то вставка записей не прерывается, а строки с дублирующими значениями просто не вставляются.

Если некоторые поля таблицы имеют ключи PRIMARY или UNIQUE, и производится вставка новой строки, в которой эти поля имеют дублирующее значение, то при использовании выражения ON DUPLICATE KEY UPDATE будет выполнена операция UPDATE для такой строки.

2.5 Оператор UPDATE

Упрощенный синтаксис оператора UPDATE:

```
UPDATE [IGNORE] table_reference
    SET assignment_list
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

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. Если задано выражение ORDER BY, то строки будут обновляться в указанном в нем порядке.

Если указывается ключевое слово IGNORE, то команда обновления не будет прервана, даже если при обновлении возникнет ошибка дублирования ключей. Строки, из-за которых возникают конфликтные ситуации, обновлены не будут.

Выражение LIMIT ограничивает число обновляемых строк.

2.6 Оператор DELETE

Упрощенный синтаксис оператора DELETE:

```
DELETE [IGNORE] FROM tbl_name
```

```
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

ИЛИ

```
DELETE [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
    FROM table_references
    [WHERE where_condition]
```

Оператор DELETE удаляет из таблицы *tbl_name* строки, удовлетворяющие заданным в *where_condition* условиям, и возвращает число удаленных записей. Если оператор DELETE запускается без определения WHERE, то удаляются все строки. Если применяется выражение ORDER BY, то строки будут удалены в указанном порядке. Выражение LIMIT ограничивает число удаляемых строк.

Можно указать несколько таблиц в операторе DELETE для удаления строк из одной или нескольких таблиц в зависимости от условия в предложении WHERE. В данном случае нельзя использовать выражения ORDER BY или LIMIT.

2.7 Оператор WITH

Пример:

```
WITH
  cte1 AS (SELECT a, b FROM table1),
  cte2 AS (SELECT c, d FROM table2)
SELECT b, d FROM cte1 JOIN cte2
WHERE cte1.a = cte2.c;
```

В данном примере cte1 и cte2 являются общими табличными выражениями (common table expression). Общее табличное выражение (CTE) - это именованный временный набор результатов, который существует в области действия одного оператора и на который можно ссылаться позже в этом операторе, возможно, несколько раз.

Для определения CTE используется выражение WITH содержащее один или несколько разделенных запятыми выражений. Общие табличные выражения позволяют существенно уменьшить объем кода, если многократно приходится обращаться к одним и тем же производным таблицам.

3. СПРАВОЧНИК ФУНКЦИЙ MYSQL

3.1 Основные функции для работы со строками

- **CONCAT**: объединяет строки. В качестве параметра принимает от 2-х и более строк, которые надо соединить:

```
SELECT CONCAT('Tom', ' ', 'Smith');
#Результат:  'Tom Smith'
```

  При этом в функцию можно передавать не только непосредственно строки, но и числа, даты - они будут преобразовываться в строки и также объединяться.

- **CONCAT_WS**: также объединяет строки, но в качестве первого параметра принимает разделитель, который будет соединять строки:

```
SELECT CONCAT_WS(' ', 'Tom', 'Smith', 'Age:', 34);
#Результат: 'Tom Smith Age: 34'
```

- **LENGTH**: возвращает количество символов в строке. В качестве параметра в функцию передается строка, для которой надо найти длину:

```
SELECT LENGTH('Tom Smith');
#Результат: 9
```

- **LTRIM**: удаляет начальные пробелы из строки. В качестве параметра принимает строку:

```
SELECT LTRIM('  Apple');
#Результат: 'Apple'
```

- **RTRIM**: удаляет конечные пробелы из строки. В качестве параметра принимает строку:

```
SELECT RTRIM(' Apple    ');
#Результат: ' Apple'
```

- **TRIM**: удаляет начальные и конечные пробелы из строки. В качестве параметра принимает строку:

```
SELECT TRIM('  Tom Smith   ');
#Результат: 'Tom Smith'
```

  С помощью дополнительного оператора можно задать где именно удалить пробелы: BOTH (в начале и в конце), TRAILING (только в конце), LEADING (только в начале):

```
SELECT TRIM(LEADING  FROM '  Tom Smith   ');
#Результат: 'Tom Smith   '
```

- **LOCATE(find, search [, start])**: возвращает позицию первого вхождения подстроки find в строку search. Дополнительный параметр start позволяет установить позицию в строке search, с которой начинается поиск подстроки find. Если подстрока search не найдена, то возвращается 0:

```

SELECT LOCATE('om', 'Tom Smith');
#Результат: 2

```

- **LEFT**: вырезает с начала строки определенное количество символов. Первый параметр функции - строка, а второй - количество символов, которые надо вырезать сначала строки:

```
SELECT LEFT('Apple', 3);
#Результат:  'App'
```

- **RIGHT**: вырезает с конца строки определенное количество символов. Первый параметр функции - строка, а второй - количество символов, которые надо вырезать сначала строки:

```
SELECT RIGHT('Apple', 3);
#Результат:  'ple'
```

- **SUBSTRING(str, start [, length])**: вырезает из строки str подстроку, начиная с позиции start. Третий необязательный параметр передает количество вырезаемых символов:

```
SELECT SUBSTRING('Tom Smith 1990', 5);
#Результат:  'Smith 1990'
SELECT SUBSTRING('Tom Smith 1990', 5,5);
#Результат:  'Smith'
```

- **SUBSTRING_INDEX(str, delimiter, count)**: вырезает из строки str подстроку. Параметр delimiter определяет разделитель внутри строки. А параметр count определяет, до какого вхождения разделителя надо вырезать подстроку. Если count положительный, то подстрока вырезается с начала, если count отрицательный, то с конца строки str:

```
SELECT SUBSTRING_INDEX('Tom Smith 1990', ' ', 1);
#Результат:  'Tom'
SELECT SUBSTRING_INDEX('Tom Smith 1990', ' ', -2);
#Результат:  'Smith 1990'
```

- **REPLACE(search, find, replace)**: заменяет в строке find подстроку search на подстроку replace. Первый параметр функции - строка, второй - подстрока, которую надо заменить, а третий - подстрока, на которую надо заменить:

```
1    SELECT REPLACE('Tom Smith 1990', 'Smith 1990', '1990');
     #Результат:  'Tom 1990'
```

- **INSERT(str, start, length, insert)**: вставляет в строку str, заменяя length символов с позиции start подстрокой insert. Первый параметр функции - строка, второй - позиция, с которой надо заменить, третий - сколько символов с позиции start надо заменить вставляемой подстрокой, четвертый параметр - вставляемая подстрока:

```
1    SELECT INSERT('Tom 1990' , 5, 1, 'Smith ');
     #Результат:  'Tom Smith 990'
```

- **REVERSE**: переворачивает строку наоборот:

```
1    SELECT REVERSE('123456789');
     #Результат:  '987654321'
```

- **LOWER**: переводит строку в нижний регистр:

```
1    SELECT LOWER('Apple');
     #Результат:  'apple'
```

- **UPPER**: переводит строку в верхний регистр:

```
1    SELECT UPPER('Apple');
     #Результат:  'APPLE'
```

- **SPACE(num)**: возвращает строку, которая содержит num пробелов:

```
1    SELECT SPACE(6);
     #Результат:  '      '
```

- **REPEATE(str, count)**: возвращает строку, которая содержит определенное количество повторов подстроки str. Количество повторов задается через параметр count.

```
1    SELECT REPEAT('ab', 5);
     #Результат:  'abababababab'
```

- **LPAD(str, length, pad)**: добавляет слева от строки str некоторое количество символов, которые определены в параметре pad. Количество добавляемых символов вычисляется по формуле length - LENGTH(str). Если параметр length меньше длины строки str, то эта строка усекается до length символов.

```
SELECT LPAD('Tom Smith', 13, '*');
#Результат:  '****Tom Smith'
```

- **RPAD(str, length, pad)**: добавляет справа от строки str некоторое количество символов, которые определены в параметре pad. Количество добавляемых символов вычисляется по формуле length - LENGTH(str). Если параметр length меньше длины строки str, то эта строка усекается до length символов.

```
SELECT RPAD('Tom Smith', 13, '*');   -- Tom Smith****
#Результат:  'Tom Smith****'
```

3.2 Основные функции для работы с числами

- **ROUND**: округляет число. В качестве первого параметра передается число. Второй параметр указывает на длину. Если длина представляет положительное число, то оно указывает, до какой цифры после запятой идет округление. Если длина представляет отрицательное число, то оно указывает, до какой цифры с конца числа до запятой идет округление:

```
SELECT ROUND(1342.345, 2);
#Результат: 1342.35
 (SELECT ROUND(1342.345, -2));
 #Результат: 1300
```

- **TRUNCATE(number, decimal_places)**: возвращает число, усеченное до определенного количества десятичных знаков, определяемого параметром decimal_places.

```
SELECT TRUNCATE(1342.345, 2);
#Результат: 1342.34
```

- **ABS**: возвращает абсолютное значение числа.

```
SELECT ABS(-123);
#Результат: 123
```

- **CEILING**: возвращает наименьшее целое число, которое больше или равно текущему значению.

```
SELECT CEILING(-123.45);
#Результат: -123
SELECT CEILING(123.45);
#Результат: -124
```

- **FLOOR**: возвращает наибольшее целое число, которое меньше или равно текущему значению.

```
SELECT FLOOR(-123.45);
```

```
2       #Результат: -124
        SELECT FLOOR(123.45);
        #Результат: 123
```

- **POWER**: возводит число в определенную степень.

```
        SELECT POWER(5, 2);
        #Результат: 25
1
2       SELECT POWER(5, 3);
        #Результат: 125
```

- **SQRT**: возвращает квадратный корень числа.

```
1       SELECT SQRT(225);
        #Результат: 15
```

- **SIGN**: возвращает -1, если число меньше 0, и возвращает 1, если число больше 0. Если число равно 0, то возвращает 0.

```
        SELECT SIGN(-5);
1       #Результат: -1
2       SELECT SIGN(7);
        #Результат: 1
```

- **RAND**: генерирует случайное число с плавающей точкой в диапазоне от 0 до 1.

```
1       SELECT RAND();
2       #Результат: 0.8201949808934657
```

3.3 Основные функции для работы с датами и временем

Получение даты и времени

- Функции **NOW()**, **SYSDATE()**, **CURRENT_TIMESTAMP()** возвращают текущую локальную дату и время на основе системных часов в виде объекта datetime. Все три функции возвращают одинаковый результат

```
        SELECT NOW();
        #Результат: 2018-05-25 21:34:55
1       SELECT SYSDATE();
2       #Результат: 2018-05-25 21:34:55
3       SELECT CURRENT_TIMESTAMP();
        #Результат: 2018-05-25 21:34:55
```

- Функции **CURDATE** и **CURRENT_DATE** возвращают текущую локальную дату в виде объекта date:

```
SELECT CURRENT_DATE();
#Результат: 2018-05-25
SELECT CURDATE();
#Результат: 2018-05-25
```

- Функции **CURTIME** и **CURRENT_TIME** возвращают текущее время в виде объекта time:

```
SELECT CURRENT_TIME();
#Результат: 20:47:45
SELECT CURTIME();
#Результат: 20:47:45
```

- **UTC_DATE** возвращает текущую локальную дату относительно GMT

```
SELECT UTC_DATE();
#Результат: 2018-05-25
```

- **UTC_TIME** возвращает текущее локальное время относительно GMT

```
SELECT UTC_TIME();
#Результат: 17:47:45
```

Парсинг даты и времени

- **DAYOFMONTH(date)** возвращает день месяца в виде числового значения
- **DAYOFWEEK(date)** возвращает день недели в виде числового значения
- **DAYOFYEAR(date)** возвращает номер дня в году
- **MONTH(date)** возвращает месяц даты
- **YEAR(date)** возвращает год из даты
- **QUARTER(date)** возвращает номер квартала года
- **WEEK(date [, first])** возвращает номер недели года. Необязательный параметр позволяет задать стартовый день недели. Если этот параметр равен 1, то первым днем считается понедельник, иначе воскресенье
- **LAST_DAY(date)** возвращает последний день месяца в виде даты
- **DAYNAME(date)** возвращает название дня недели
- **MONTHNAME(date)** возвращает название текущего месяца
- **HOUR(time)** возвращает час времени
- **MINUTE(time)** возвращает минуту времени
- **SECOND(time)** возвращает секунду времени

Примеры функций:

Функция	Результат
DAYOFMONTH('2018-05-25')	25
DAYOFWEEK('2018-05-25')	6
DAYOFYEAR('2018-05-25')	145
MONTH('2018-05-25')	5
YEAR('2018-05-25')	2018
QUARTER('2018-05-25')	2
WEEK('2018-05-25', 1)	21
LAST_DAY('2018-05-25')	2018-05-31
DAYNAME('2018-05-25')	Friday
MONTHNAME('2018-05-25')	May
HOUR('21:25:54')	21
MINUTE('21:25:54')	25
SECOND('21:25:54')	54

Функция EXTRACT

Функция EXTRACT извлекает из даты и времени какой-то определенный компонент. Ее формальный синтаксис:

```
1    EXTRACT(unit FROM datetime)
```

Значение datetime представляет исходную дату и (или) время, а значение unit указывает, какой компонент даты или времени будет извлекаться. Параметр unit может представлять одно из следующих значений:

- SECOND (секунды)
- MINUTE (минуты)
- HOUR (час)
- DAY (день)
- MONTH (месяц)
- YEAR (год)
- MINUTE_SECOND (минуты и секунды)
- HOUR_MINUTE (часы и минуты)
- DAY_HOUR (день и часы)
- YEAR_MONTH (год и месяц)
- HOUR_SECOND (часы, минуты и секунды)
- DAY_MINUTE (день, часы и минуты)
- DAY_SECOND (день, чаы, минуты и секунды)

Примеры вызова функции:

Вызов	Результат
EXTRACT(SECOND FROM '2018-05-25 21:25:54')	54
EXTRACT(MINUTE FROM '2018-05-25 21:25:54')	25
EXTRACT(HOUR FROM '2018-05-25 21:25:54')	21
EXTRACT(DAY FROM '2018-05-25 21:25:54')	25
EXTRACT(MONTH FROM '2018-05-25 21:25:54')	5
EXTRACT(YEAR FROM '2018-05-25 21:25:54')	2018
EXTRACT(MINUTE_SECOND FROM '2018-05-25 21:25:54')	2554
EXTRACT(DAY_HOUR FROM '2018-05-25 21:25:54')	2521
EXTRACT(YEAR_MONTH FROM '2018-05-25 21:25:54')	201805
EXTRACT(HOUR_SECOND FROM '2018-05-25 21:25:54')	212554
EXTRACT(DAY_MINUTE FROM '2018-05-25 21:25:54')	252125
EXTRACT(DAY_SECOND FROM '2018-05-25 21:25:54')	25212554

Функции для манипуляции с датами

Ряд функций позволяют производить операции сложения и вычитания с датами и временем:

- **DATE_ADD(date, INTERVAL expression unit)** возвращает объект DATE или DATETIME, который является результатом сложения даты date с определенным временным интервалом. Интервал задается с помощью выражения INTERVAL expression unit, где INTERVAL предоставляет ключевое слово, expression - количество добавляемых к дате единиц, а unit - тип единиц (часы, дни и т.д.) Параметр unit может иметь те же значения, что и в функции EXTRACT, то есть DAY, HOUR и т.д.
- **DATE_SUB(date, INTERVAL expression unit)** возвращает объект DATE или DATETIME, который является результатом вычитания из даты date определенного временного интервала
- **DATEDIFF(date1, date2)** возвращает разницу в днях между датами date1 и date2
- **TO_DAYS(date)** возвращает количество дней с 0-го года
- **TIME_TO_SEC(time)** возвращает количество секунд, прошедших с момента полуночи

Примеры применения:

Вызов	Результат
DATE_ADD('2018-05-25', INTERVAL 1 DAY)	2018-05-26
DATE_ADD('2018-05-25', INTERVAL 3 MONTH)	2018-08-25
DATE_ADD('2018-05-25 21:31:27', INTERVAL 4 HOUR)	2018-05-26 01:31:27
DATE_SUB('2018-05-25', INTERVAL 4 DAY)	2018-05-21
DATEDIFF('2018-05-25', '2018-05-27')	-2

DATEDIFF('2018-05-25', '2018-05-21')	4
DATEDIFF('2018-05-25', '2018-03-21')	65
TO_DAYS('2018-05-25')	737204
TIME_TO_SEC('10:00')	36000

Форматирование дат и времени

- **DATE_FORMAT(date, format)** возвращает объект DATE или DATETIME, отформатированный с помощью шаблона format
- **TIME_FORMAT(date, format)** возвращает объект TIME или DATETIME, отформатированный с помощью шаблона format

Обе функции в качестве второго параметра принимают строку форматирования или шаблон, который показывает, как оформатировать значение. Этот шаблон может принимать следующие значения:

- %m: месяц в числовом формате 01..12
- %c: месяц в числовом формате 1..12
- %M: название месяца (January...December)
- %b: аббревиатура месяца (Jan...Dec)
- %d: день месяца в числовом формате 00..31
- %e: день месяца в числовом формате 0..31
- %D: номер дня месяца с суффиксом (1st, 2nd, 3rd...)
- %y: год в виде двух чисел
- %Y: год в виде четырех чисел
- %W: название дня недели (Sunday...Saturday)
- %a: аббревиатура дня недели (Sun...Sat)
- %H: час в формате 00..23
- %k: час в формате 0..23
- %h: час в формате 01..12
- %l: час в формате 1..12
- %i: минуты в формате 00..59
- %r: время в 12-ти часовом формате (hh:mm:ss AM или PM)
- %T: время в 24-ти часовом формате (hh:mm:ss)
- %S: секунды в формате 00..59
- %p: AM или PM

Примеры применения:

Вызов	Результат
DATE_FORMAT('2018-05-25', '%d/%m/%y')	25/05/18
DATE_FORMAT('2018-05-25 21:25:54', '%d %M %Y')	25 May 2018
DATE_FORMAT('2018-05-25 21:25:54', '%r')	09:25:54 PM
TIME_FORMAT('2018-05-25 21:25:54', '%H:%i:%S')	21:25:24
TIME_FORMAT('21:25:54', '%k:%i')	21:25

4. СХЕМА БД SAKILA

База данных sakila представляет собой базу данных сети магазинов проката DVD фильмов. На рисунке 3.1 представлена схема БД sakila.

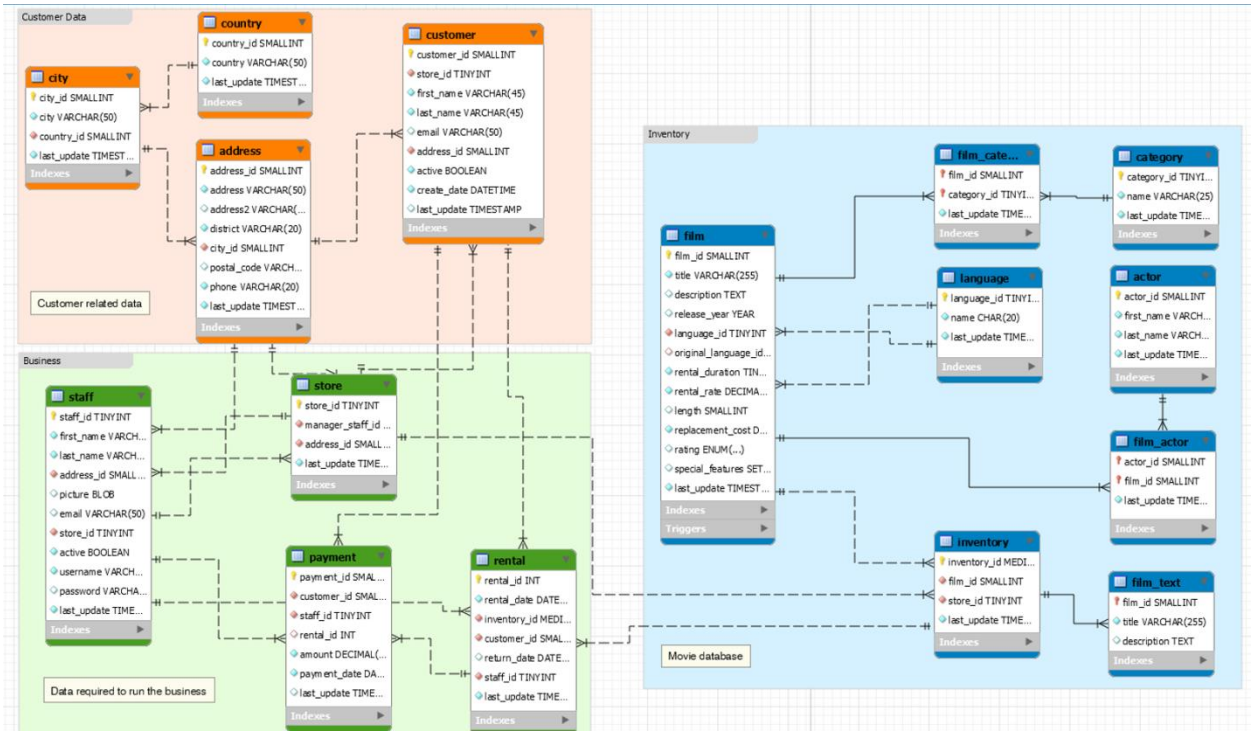


Рисунок 3.1 – Схема БД sakila.

БД sakila включает в себя 16 таблиц.

Информация о клиентах магазинов хранится в таблицах customer, *address*, *city*, *country*.

Данные о магазинах, в которых были совершены покупки, продавцах, платежах и арендных сроках содержатся в таблицах *store*, *staff*, *payment*, *rental*.

Остальные таблицы содержат информации о базе DVD фильмов сети магазинов: описание и характеристики фильмов (*film*), жанры фильмов (*film_category*, *category*), данные об актерах (*actor*, *film_actor*), данные о наличии фильмов в магазинах (*inventory*, *film_text*).

Подробное описание БД sakila представлено на сайте https://dev.mysql.com/doc/sakila/en/.