**Курсовая работа**

«Описание модели хранения данных на примере

популярного веб-сайта: Кинопоиск»

Created by:

Савельев Иван

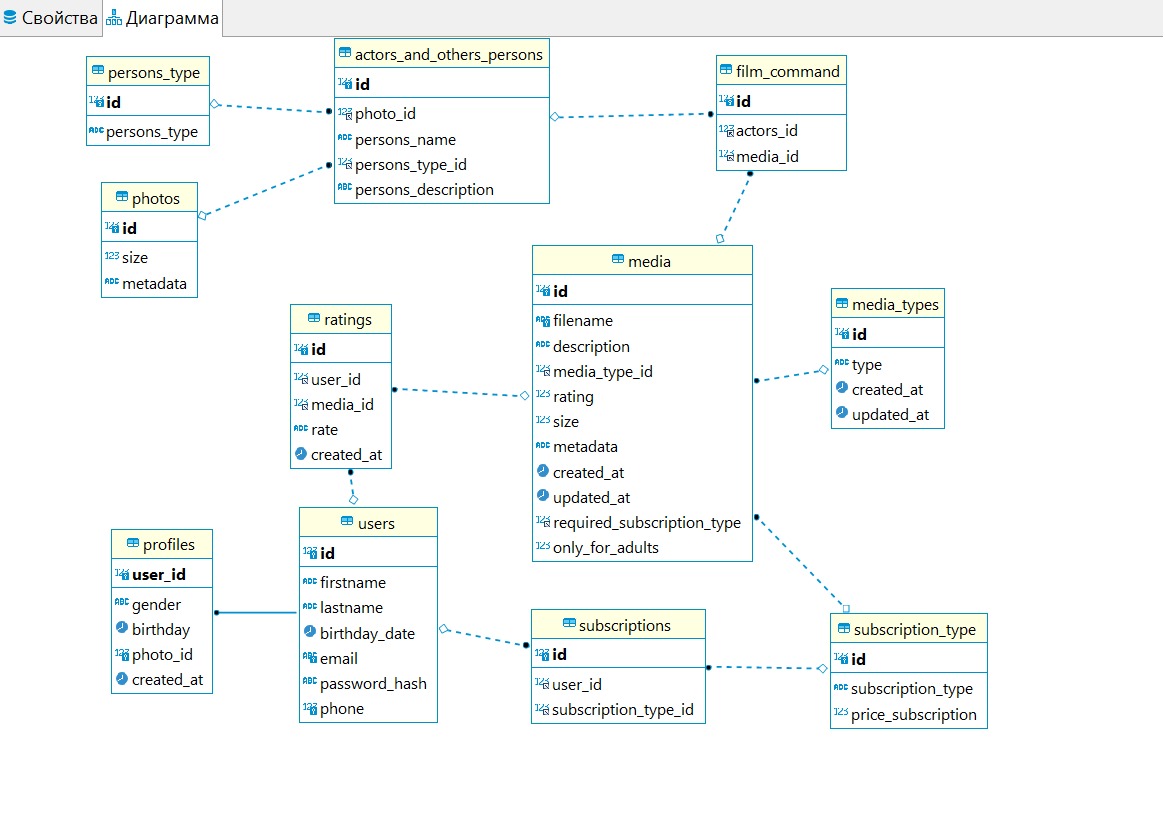
**Общее текстовое описание БД и решаемых ею задач.**

Рассмотрим спроектированную базу данных с использованием СУБД “MySQL” на основе популярного видео-сервиса “Кинопоиск”.

В БД хранится информация о Медиа-контенте (фильмы, сериалы и т.д.) составе производственных команд, так же в ней содержится информация о пользователях, имеющихся у них подписках, а также предусмотрена возможность выставления оценок для Контента пользователями.

С помощью БД присутствует возможность получения информации о составе команд работавших над Конентом. И предусмотрена возможность проверки возраста пользователя для получения доступа к контенту имеющему возрастные ограничения.

**ER-Diagram:**

****

Используя данную базу данных у нас есть возможность решать разного рода задачи.

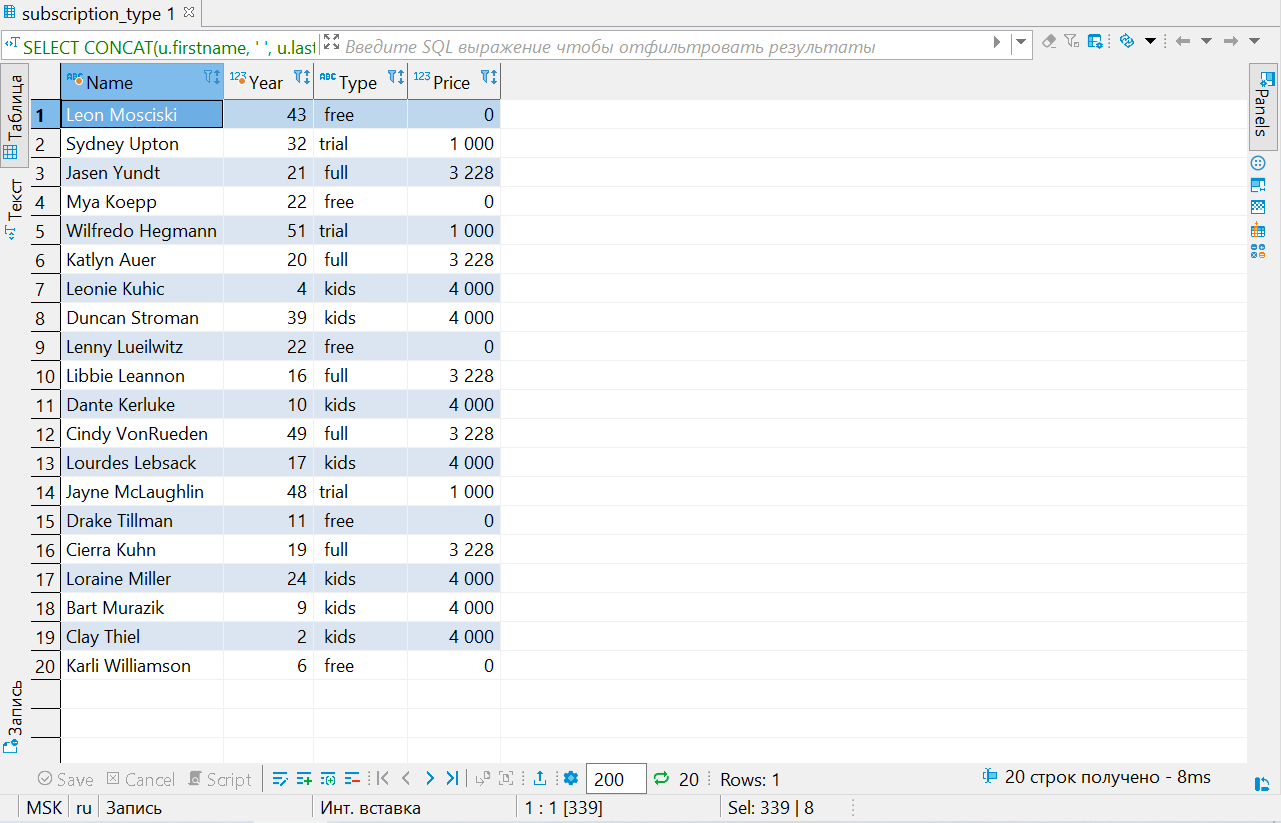
Например:

Можно узнать, какая подписка у пользователя, какова её стоимость, а так же можно вывести возраст пользователей, использую JOIN.

SQL запрос будет выглядеть следующим образом

|  |
| --- |
| **SELECT** **CONCAT**(u.firstname, ' ', u.lastname) **as** Name,  **YEAR**(**CURRENT\_DATE**) - **YEAR**(p.birthday) **AS** `Year`,  tp.subscription\_type **AS** `Type`,  tp.price\_subscription **AS** Price  **FROM** users **AS** u  **JOIN** profiles **AS** p **on** u.id = p.user\_id  **JOIN** subscriptions **AS** pd **on** pd.user\_id = u.id  **JOIN** subscription\_type **AS** tp **on** tp.id = pd.subscription\_type\_id; |

Результаты данного запроса опубликованы ниже:



Для более наглядного вывода результатов запроса за частую используют сортировку в конце кода запроса следующего вида:

|  |
| --- |
| **ORDER** **BY** Price **DESC**; |

Данная сортировка показывает у кого из пользователей самая дорогая подписка, а у кого пробная версия.

Используя агрегацию данных в базе мы можем получить информацию, какие именно фильмы относятся к жанру “ television series”,

**SELECT** **COUNT**(media.filename) Count\_films,

**GROUP\_CONCAT**(media.filename **SEPARATOR** ', ')

Films, mt.`type`

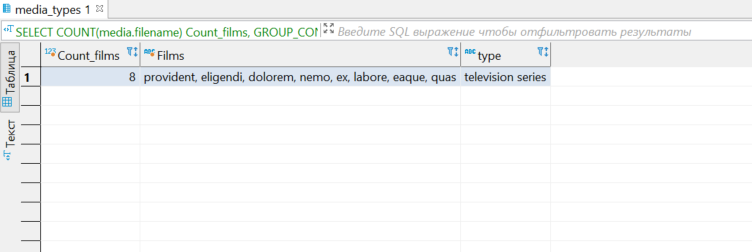
**FROM** media

**JOIN** media\_types mt **ON** mt.id = media.media\_type\_id

**WHERE** `type` = 'television series'

**GROUP** **BY** `type`;

В результате выполнения данного запроса получаем следующий результат:



**Представления**

Основными структурными единицами в реляционных базах данных являются таблицы. Однако язык запросов SQL предоставляет еще один способ организации данных — представления.

Представление — это запрос на выборку (**SELECT**), которому присваивается уникальное имя и который можно сохранять или удалять из базы данных как обычную хранимую процедуру.

Поместим наш первый запрос в представление:

**CREATE** **OR** **REPLACE** **VIEW** name\_info **AS**

**SELECT** **CONCAT**(u.firstname, ' ', u.lastname) **as** Name,

**YEAR**(**CURRENT\_DATE**) - **YEAR**(p.birthday) **AS** `Year`,

tp.subscription\_type **AS** `Type`,

tp.price\_subscription **AS** Price

**FROM** users **AS** u

**JOIN** profiles **AS** p **on** u.id = p.user\_id

**JOIN** subscriptions **AS** pd **on** pd.user\_id = u.id

**JOIN** subscription\_type **AS** tp **on** tp.id = pd.subscription\_type\_id

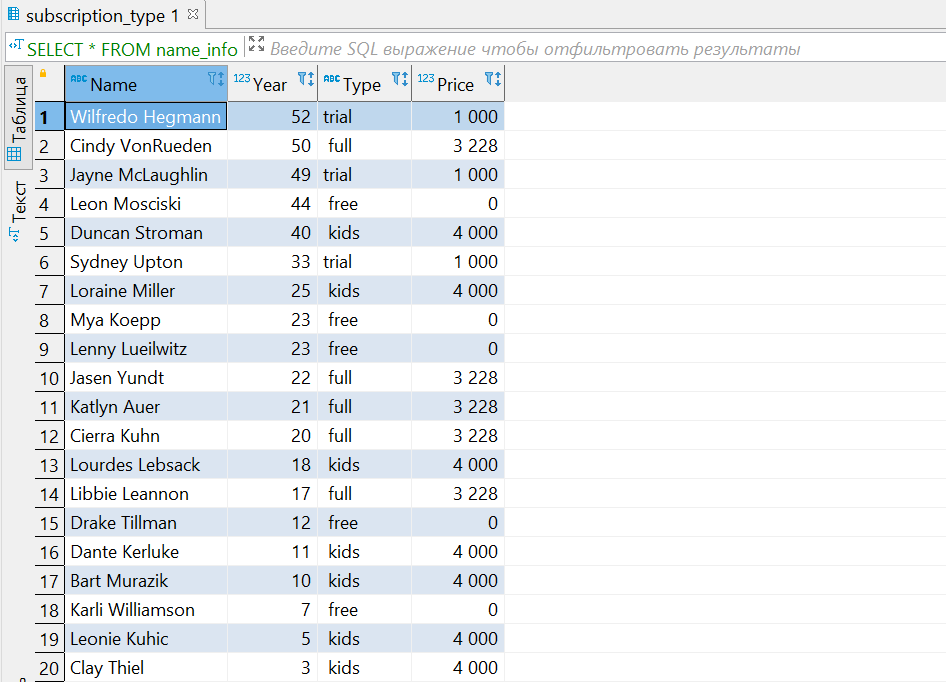
**ORDER** **BY** `Year` **DESC**;

После выполнения этого скрипта мы создаем представление в БД и имеем возможность обращаться к **name\_info** как к обычной таблице.

Выполнив общий запрос вида:

**SELECT** \* **FROM** name\_info;

Мы получаем следующий результат:



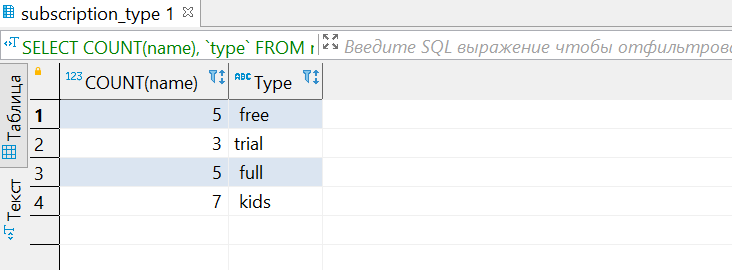
Так же с помощью этого представления мы можем узнать количество пользователей имеющих одну подписку используя простой запрос вида:

**SELECT** **COUNT**(name), `type`

**FROM** name\_info

**GROUP** **BY** `type`;

На выводе получая необходимые нам параметры:



Так же в качестве примера мы можем объединить несколько таблиц в одну используя запрос вида:

**DROP** **VIEW** **IF** **EXISTS** all\_values;

**CREATE** **OR** **REPLACE** **VIEW** all\_values **AS**

**SELECT** u.\*,

p.\*,

pd.subscription\_type\_id,

tp.subscription\_type,

tp.price\_subscription,

mr.rate,

md.required\_subscription\_type,

cf.actors\_id

**FROM** users **AS** u

**JOIN** profiles **AS** p **on** u.id = p.user\_id

**JOIN** subscriptions **AS** pd **on** pd.user\_id = u.id

**JOIN** subscription\_type **AS** tp **on** tp.id = pd.subscription\_type\_id

**JOIN** ratings **AS** mr **on** mr.user\_id = u.id

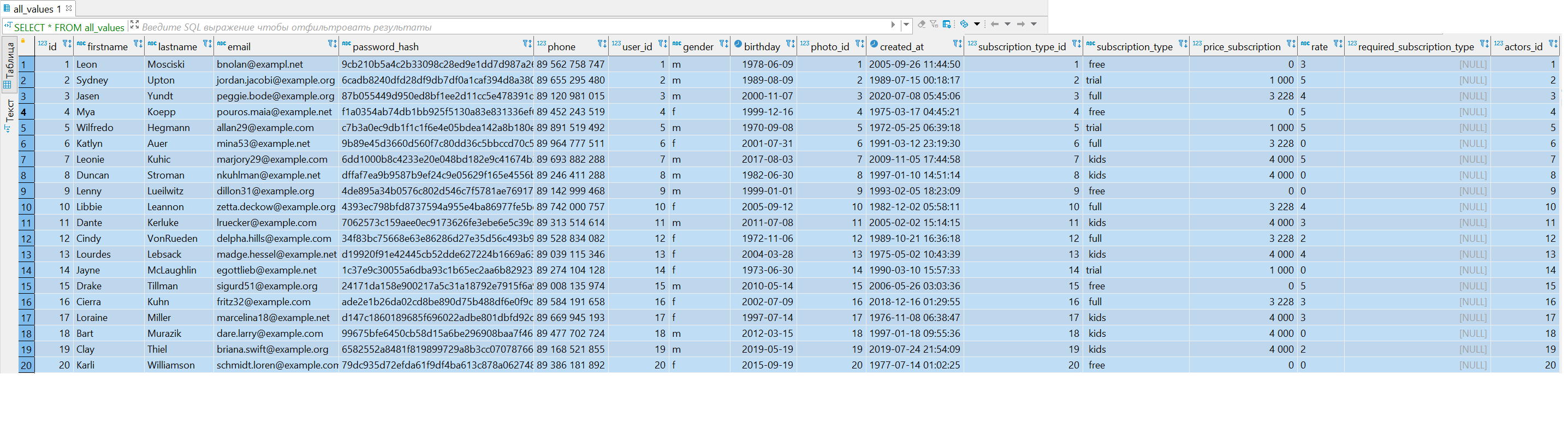
**JOIN** media **AS** md **on** md.id = mr.media\_id

**LEFT** **JOIN** film\_command **AS** cf **on** cf.media\_id = md.id;

Где используя запрос вида:

**SELECT** \* **FROM** all\_values;

Мы получаем всю информацию в следующем виде:



**Процедуры**

Хранимые процедуры – это наиболее часто используемая программные структуры позволяющие сохранять последовательность SQL-операторов и вызывать их по имени функции или процедур, которые хранятся в базе данных, а не в отдельном файле.

Процедура решает наиболее часто возникающие задачи. Такие как, например изменение столбца only\_for\_adults (только для взрослых) где у нас стоит тип BIT (по умолчанию 0) на 1.

Создадим процедуру:

**DROP** **PROCEDURE** **IF** **EXISTS** kinopoisk.change\_mode;

**DELIMITER //**

**CREATE** **PROCEDURE** kinopoisk.change\_mode()

**BEGIN**

**UPDATE** media **SET** only\_for\_adults = 1 **WHERE** media\_type\_id **IN** (2,4);

**END**//

**DELIMITER ;**

Теперь при вызове данной процедуры

**CALL** change\_mode();

будет меняться значение в столбце only\_for\_adults с 0 на 1.

**Триггеры**

**Триггер** — это хранимая процедура, которая не вызывается непосредственно, а исполняется при наступлении определенного события ( вставка, удаление, обновление строки ).

Для оптимизации работы с базой данных и чтобы избавиться от необходимости постоянного вызова процедуры, можно воспользоваться триггером, изменяющим значение only\_for\_adults автоматически после добавления строки в таблицу media. Воспользуемся скриптом создания триггера:

**DROP** **TRIGGER** **IF** **EXISTS** changes;

**DELIMITER //**

**CREATE** **TRIGGER** changes **BEFORE** **INSERT** **ON** media

**FOR** **EACH** **ROW**

**BEGIN**

**IF**(**NEW**.media\_type\_id **IN** (2, 4)) **THEN**

**SET** **NEW**.only\_for\_adults = 1;

**END** **IF**;

**END** //

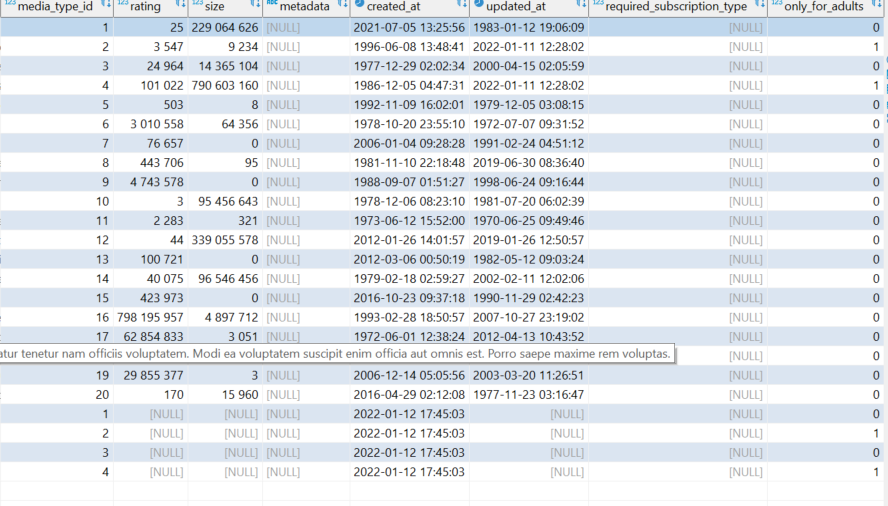
**DELIMITER ;**

После добавления триггера, значение строки в столбце only\_for\_adults, где media\_type\_id равен 2 или 4 будут автоматически изменяться на 1

Для проверки работы триггера используем пакетную вставку

**INSERT** **INTO** media (media\_type\_id) **VALUES** (1), (2), (3), (4);

В результате чего получаем следующее наполнение таблицы media:



**Используемые источники**

1. <https://dev.mysql.com/doc/refman/5.7/en/union.html>
2. <https://dev.mysql.com/doc/refman/5.7/en/subqueries.html>
3. <https://dev.mysql.com/doc/refman/5.7/en/join.html>
4. Линн Бейли. Head First. Изучаем SQL. — СПб.: Питер, 2012. — 592 с.
5. Грофф, Джеймс Р., Вайнберг, Пол Н., Оппель, Эндрю Дж. SQL: полное руководство, 3-е изд. : Пер. с англ. — М.: ООО "И.Д. Вильямс", 2015. — 960 с.
6. Дейт К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL. — Пер. с англ. — СПб.: Символ-Плюс, 2010. — 480 с.
7. Кузнецов М.В., Симдянов И.В. MySQL на примерах. — СПб.: БХВ-Петербург, 2007. — 592с.
8. Кузнецов М.В., Симдянов И.В. MySQL 5. — СПб.: БХВ-Петербург, 2006. — 1024с.
9. Дейт К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.
10. Карвин Б. Программирование баз данных SQL. Типичные ошибки и их устранение. — Рид Групп, 2011. — 336 с.