Санкт – Петербургский Политехнический Университет им. Петра Великого

Институт Прикладной Математики и Механики

Высшая школа прикладной математики и вычислительной физики

Кафедра Телематика

Отчет по курсовой работе

по дисциплине: «Теоретические основы базы данных»

по теме: «Автоматизация процесса постановки спектакля в театре»

Студент группы 3630201/70101 Я. А. Ковальчук

Преподаватель С. Г. Попов

Санкт – Петербург

2020

Содержание

[Содержание 2](#_Toc42156813)

[Введение 3](#_Toc42156814)

[1. Аналитика 4](#_Toc42156815)

[1.1. Описание предметной области 4](#_Toc42156816)

[1.1.1. Подготовительный этап. Подбор персонала 4](#_Toc42156817)

[1.1.2. Смета спектакля 5](#_Toc42156818)

[1.1.3. Актерский состав 5](#_Toc42156819)

[1.1.4. Репетиционный период 6](#_Toc42156820)

[1.2. Формулировка целей и функций системы 7](#_Toc42156821)

[1.3. Выделение сущностей и их атрибутов 7](#_Toc42156822)

[1.4. Обоснование сущностей 9](#_Toc42156823)

[1.5. Схема объектов 10](#_Toc42156824)

[1.6. ER-диаграмма 11](#_Toc42156825)

[1.6.1. Чтение ER-диаграммы 11](#_Toc42156826)

[2. Проектирование 12](#_Toc42156827)

[2.1. Создание схемы базы данных 12](#_Toc42156828)

[2.2. Таблицы атрибутов базы данных 15](#_Toc42156829)

[2.3. Генерация базы данных 18](#_Toc42156830)

[2.4. Заполнение базы данных 18](#_Toc42156831)

[3. Реализация запросов 20](#_Toc42156832)

[3.1. Запрос №1 20](#_Toc42156833)

[3.2. Запрос №2 21](#_Toc42156834)

[3.3. Запрос №3 22](#_Toc42156835)

[3.4. Запрос №4 26](#_Toc42156836)

[3.5. Запрос №5 27](#_Toc42156837)

[3.6. Запрос №6 31](#_Toc42156838)

[3.7. Запрос №7 33](#_Toc42156839)

[3.8. Запрос №8 35](#_Toc42156840)

[Заключение 38](#_Toc42156841)

[Список литературы 39](#_Toc42156842)

[Приложение 1 40](#_Toc42156843)

[Приложение 2 44](#_Toc42156844)

Введение

В данном отчете описан процесс создания базы данных для выбранной предметной области и работы с полученной базой данных. В качестве предметной области выбран театр, а именно его внутренний функционал. База данных должна предоставлять информацию о предстоящих спектаклях, режиссерах, актерах, составах, участвующих в постановках.

Работа включает в себя несколько этапов:

1. Аналитика. Она включает в себя анализ предметной области, выявление основных целей и функций системы, выделение сущностей и их атрибутов.
2. Проектирование базы данных. Включает в себя создание схемы базы данных, создание таблицы атрибутов, генерацию базы данных и заполнение ее записями.
3. Реализация запросов. Включает в себя создание запросов и анализ результатов выполнения.

Для выполнения работы был выбрана свободная реляционная СУБД MySQL.

# Аналитика

## Описание предметной области

Внутренний функционал театра при подготовке спектакля.

Каждый год театр осуществляет постановку различных спектаклей. Каждый спектакль имеет определенный бюджет. Театр возглавляет директор, в функции которого входят контроль за постановкой спектаклей, утверждение репертуара, прием на работу новых служащих, расчет сметы для реализации каждого нового спектакля.

### Подготовительный этап. Подбор персонала

Работа над спектаклем начинается с опубликования приказа о принятии пьесы к постановке. В этом документе указываются: режиссер - постановщик, название пьесы, первоначальный бюджет, дата отбора актерского состава и дата совещания по планированию выпуска постановки.

В основе каждой постановки лежит пьеса. Одна пьеса может иметь разные вариации постановок, в зависимости от идеи режиссера. Директор утверждает пьесу, которую необходимо поставить. Каждая пьеса имеет свой жанр. Директор назначает режиссера, который в дальнейшем ведет всю деятельность по подготовке спектакля, согласовывая ее с директором. Режиссера выбирают либо из уже имеющихся режиссеров театра, либо приглашают режиссера из других театров. На данный момент влияет несколько факторов:

* опыт режиссера;
* постановки, которые уже были им реализованы;
* занятость в других спектаклях;
* требуемая заработная плата.

Далее директор утверждает сотрудников необходимых для начала работы:

* художников по костюмам,
* декораторов,
* хореографов,
* музыкантов,
* звукорежиссёров.

Данная команда выбирается из работников театра в зависимости от их занятости в других спектаклях, опыта. После чего издается приказ, в котором прописаны все сотрудники, задействованные в спектакле.

### Смета спектакля

Директор составляет предварительную смету спектакля на декорации, мебель, бутафорию, костюмы и другие затраты на материальное оформление спектакля. Составляются заявки на материалы необходимые на оформление спектакля. Ведется работа над составлением графика производства декораций, костюмов. Приобретается материал для производства. Выдаются задания по изготовлению материального оформления спектакля по цехам и конкретным исполнителям. Изготавливаются жесткие и мягкие декорации, бутафория, живописные завесы, костюмы, головные уборы, обувь, парики, цветы и приобретается дополнительное оборудование. Заказывается исходящий реквизит.

### Актерский состав

В соответствии с установленной в приказе датой отбора актерского состава ведется кастинг. На роли актеров утверждает режиссер постановки.

Режиссер предоставляет список ролей необходимых в спектакле. Роли подразделяются на две категории: главная роль и второстепенная. Для главных ролей также необходимо утверждение дублеров. На одну главную роль подбирают 1-2 дублеров. Дублеров выбирают из актерского состава театра.

В театре существует основной актерский состав, а также на главные роли могут быть приглашены актеры других театров. Каждый актер может быть уже задействован в других постановках театра, быть на гастролях, а также на больничном или в отпуске. Актеры имеют свои собственные вокальные или внешние данные, которые могут подходить или нет к определенной роли. Все актеры могут иметь звания заслуженных и народных артистов, могут быть лауреатами конкурсов. Также актерами театра могут быть и студенты театральных училищ. Каждый актер должен иметь дату утверждения на роль и дату снятия с роли. С каждым из актеров заключают персональный контракт на определенную сумму.

Первым делом утверждают актеров на главные роли. У каждого актера главной роли должен быть дублер. На главные роли могут быть утверждены только те актеры, которые имеют звания заслуженных и народных артистов или лауреата всероссийских конкурсов. Также важен опыт, внешние данные актеров, вокальные возможности, а также занятость в других спектаклях.

Далее утверждается состав актеров второстепенных ролей. Ими могут быть актеры основного состава, а также студенты театральных училищ.

### Репетиционный период

Когда вся команда для создания спектакля собрана необходимо утверждение даты премьеры и составление расписания репетиций.

Репетиционный период спектакля начинается с составления плана сценического выпуска спектакля. В нем учитываются такие моменты как: репетиции спектакля в классах, репетиции на малой сцене, репетиции на основной сцене, репетиции на основной сцене с частичной подачей готовых элементов декораций и костюмов, репетиции с актерами по актам, по картинам, прогонные и генеральная репетиция.

В театре находится несколько репетиционных и одна основная сцена. Репетиции проходят по установленному графику, в котором прописываются даты, место проведения репетиции и задействованные актеры.

Дата премьеры утверждается директором не позднее, чем за 6 месяцев до нее.

## Формулировка целей и функций системы

Выделены следующие цели:

* Улучшить ведение базы актерского состава театра;
* Улучшить ведение базы спектаклей театра;
* Оптимизировать работу по составлению графика показов спектаклей в театре;
* Исключить ошибки составления контрактов для актеров, утвержденных на роль;
* Улучшить процесс подготовки спектакля к показу: начиная с утверждения постановки до утверждения даты показа.

Выделены следующие функции системы:

* Утверждение новой постановки;
* Составление списка ролей, задействованных в постановке;
* Составление, изменение актерского состава, принимающего участие в конкретном спектакле;
* Составление, изменение графика показов спектакля;
* Составление контрактов для актеров, утвержденных на роль;

## Выделение сущностей и их атрибутов

Выделены следующие сущности и атрибуты:

1. Режиссер – физическое лицо, которое утверждает начало работы над постановкой.

* Фамилия;
* Имя;
* Отчество;
* Контактные данные.

1. Постановка – театральное представление, которое реализует конкретный режиссер.

* Название.

1. Пьеса – произведение, на основе которого может быть реализована постановка.

* Название;
* Автор;
* Год создания.

1. Роль в пьесе – художественный, литературный образ, созданный драматургом в пьесе.

* Название;
* Тип (главная, второстепенная).

1. Роли – действующее лицо в постановке, воплощаемое в сценической игре актером.

* Название;
* Тип (главная, второстепенная).

1. Актер – физическое лицо, которое задействовано в постановке в конкретной роли.

* Фамилия;
* Имя;
* Отчество;
* Возраст;
* Пол.

1. Контракт – документ, подтверждающий назначение актера на конкретную роль.

* Дата назначения на роль;
* Гонорар;
* Подпись актера.

1. Спектакль – постановка, которая будет показана в театре.

* Дата;
* Время;
* Тип (премьера, плановый показ).

1. Состав спектакля – роли, задействованные в конкретном спектакле.

* Номер состава;
* Код спектакля;
* Код актера;

1. Достижения – награды, звания, которые были получены актерами в процессе их творческой деятельности.

* Тип достижения.

## Обоснование сущностей

Необходимость выделенных сущностей вытекает из описания предметной области:

* Сущности «Актеры», «Спектакль» и «Роли» являются ключевыми в данной предметной области, без которых невозможно функционирование системы.
* Сущность «Состав спектакля» необходима для связи ключевых сущностей.
* Сущность «Постановка» и «Пьеса» необходимы для описания спектакля.
* Сущность «Достижения актера» необходима для хранения информации об актерах, которые имеют достижения.
* Сущность «Роль в пьесе» необходима для описания ролей.
* Сущность «Контракт» необходима для документального утверждения актера на роль.
* Сущность «Режиссер» необходима для хранения информации о режиссерах, которые ставят спектакли.

## Схема объектов

На Рис. 1 представлена схема объектов, которая демонстрирует отношение между сущностями.

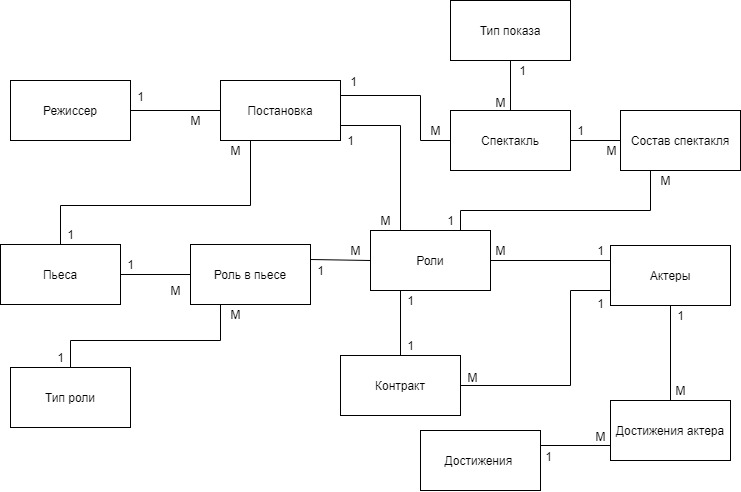


Рисунок 1. Схема объектов

## ER-диаграмма

На Рис. 2 представлена ER-диаграмма, описывающая основные функции системы.

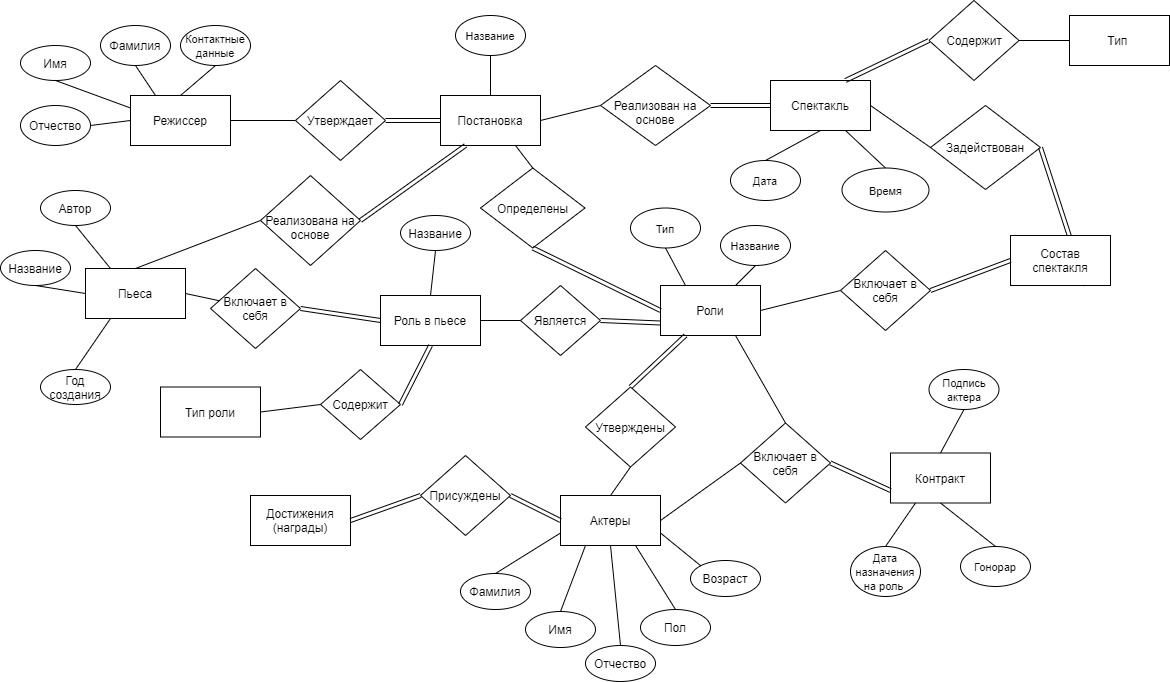


Рисунок 2. ER-диаграмма

### Чтение ER-диаграммы

* Режиссер утверждает постановку
* Постановка реализована на основе пьесы
* Пьеса включает в себя роли в пьесе, а каждая роль содержит тип (главная или второстепенная)
* В постановке определены роли, которые являются ролями в пьесе
* На роли утверждают актеров, которые подписывают контракт, в котором прописаны роли
* Актерам присуждены достижения
* Состав спектакля включает в себя роли
* Спектакль основан на постановке
* В спектакле задействован состав спектакля и спектакль содержит тип показа

# Проектирование

## Создание схемы базы данных

На Рис. 3 и Рис. 4 изображены спроектированные схемы базы данных на русском и английском языках соответственно.

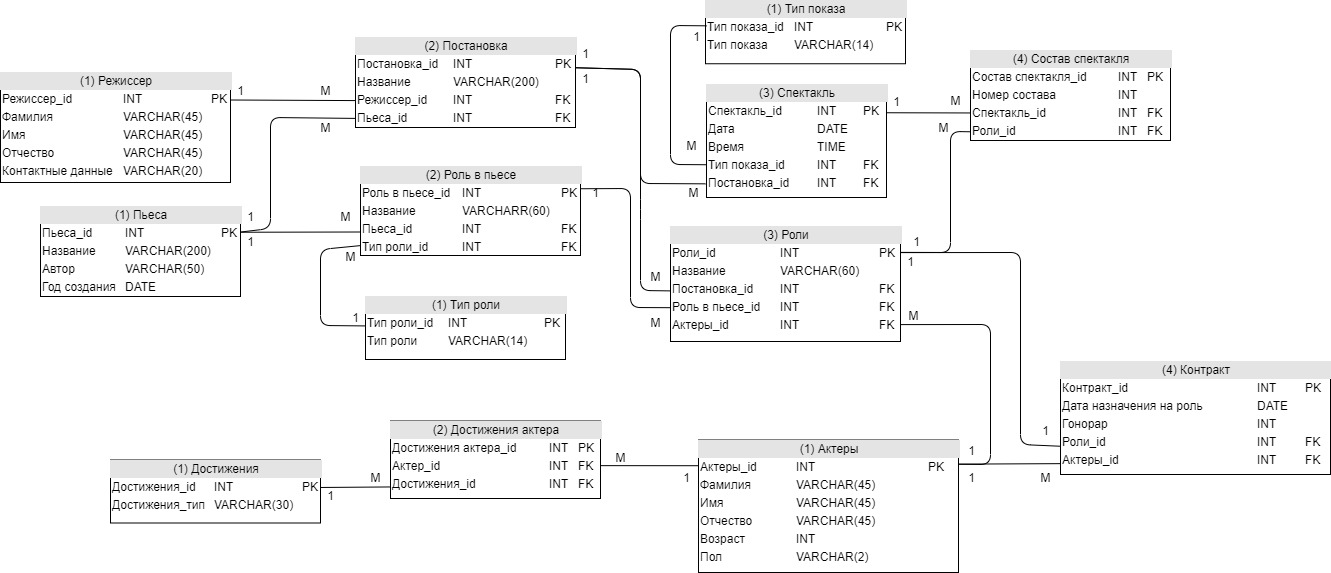


Рисунок 3. Схема базы данных на русском языке

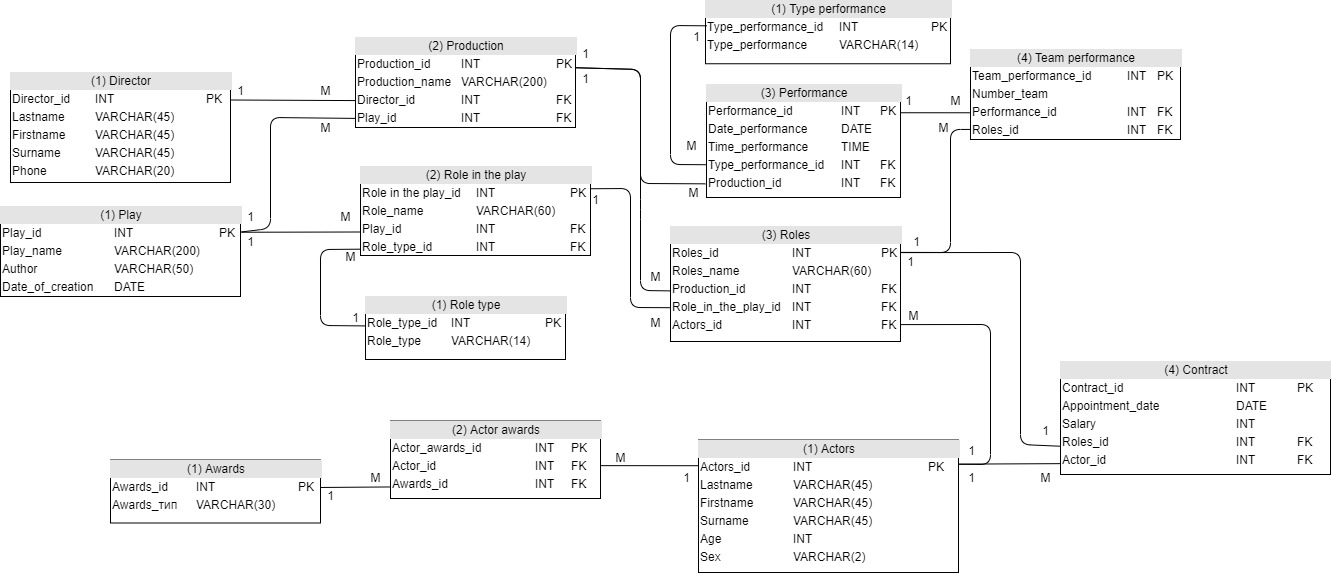


Рисунок 4. Схема базы данных на английском языке

## Таблицы атрибутов базы данных

В Таблице 1 - Таблице 13 представлены все атрибуты базы данных на русском и английском языках, также указаны тип и размер атрибутов и обоснование выбранного типа.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Режиссер (Director) | | | | | |
| Название атрибута (рус.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Режиссер\_id | Director\_id | INT | - | PK | Первичный ключ |
| Фамилия | Lastname | VARCHAR | 45 | - | Длина максимальной фамилии в мире |
| Имя | Firstname | VARCHAR | 45 | - | Размер не меньше максимальной длины фамилии |
| Отчество | Surname | VARCHAR | 45 | - | Размер не меньше максимальной длины фамилии |
| Контактные данные | Phone | VARCHAR | 20 | - | Размер не меньше международного формата (15 цифр) с учетом знаков при записи. |

Таблица 1. Режиссер

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Пьеса (Play) | | | | | |
| Название атрибута (рус.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Пьеса\_id | Director\_id | INT | - | PK | Первичный ключ |
| Название | Play\_name | VARCHAR | 200 | - | Средняя длина названий пьес |
| Автор | Author | VARCHAR | 50 | - | Размер не меньше максимальной длины фамилии с учетом добавления инициалов |
| Год создания | Date\_of\_creation | DATE | - | - | Используется для хранения только даты в формате ‘YYYY-MM-DD’ |

Таблица 2. Пьеса

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Роль в пьесе (Role\_in\_the\_play) | | | | | |
| Название атрибута (рус.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Роль\_в\_пьесе\_id | Role\_in\_the\_play\_id | INT | - | PK | Первичный ключ |
| Название | Role\_name | VARCHAR | 60 | - | Размер больше, чем длина самого длинного имени |
| Пьеса\_id | Play\_id | INT | - | FK | Ссылается на «Пьеса» |
| Тип роли\_id | Role\_type\_id | INT | - | FK | Ссылается на «Тип роли» |

Таблица 3. Роль в пьесе

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Тип роли (Role type) | | | | | |
| Название атрибута (руск.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Тип роли\_id | Role\_type\_id | INT | - | PK | Первичный ключ |
| Тип роли | Role\_type | VARCHAR | 14 | - | Длина самого длинного слова в данном словаре |

Таблица 4. Тип роли

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Постановка (Production) | | | | | |
| Название атрибута (рус.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Постановка\_id | Director\_id | INT | - | PK | Первичный ключ |
| Название | Production\_name | VARCHAR | 200 | - | Средняя длина названий пьес |
| Режиссер\_id | Director\_id | INT | - | FK | Ссылается на «Режиссер» |
| Пьеса\_id | Play\_id | INT | - | FK | Ссылается на «Пьеса» |

Таблица 5. Постановка

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Спектакль (Performance) | | | | | |
| Название атрибута (рус.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Спектакль\_id | Performance\_id | INT | - | PK | Первичный ключ |
| Дата | Date\_performance | DATE | - | - | Используется для хранения только даты в формате ‘YYYY-MM-DD’ |
| Время | Time\_performance | TIME | - |  | Используется для хранения только времени  В формате  ‘HH:MM:SS’ |
| Тип показа\_id | Type\_performance\_id | INT | - | FK | Ссылается на «Тип показа» |
| Постановка\_id | Production\_id | INT | - | FK | Ссылается на «Постановка» |

Таблица 6. Спектакль

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Роли (Roles) | | | | | |
| Название атрибута (руск.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Роли\_id | Roles\_id | INT | - | PK | Первичный ключ |
| Название | Roles\_name | VARCHAR | 60 | - | Размер больше, чем самое длинное имя |
| Постановка\_id | Production\_id | INT | - | FK | Ссылается на «Постановка» |
| Роль в пьесе\_id | Role\_in\_the\_play\_id | INT | - | FK | Ссылается на «Роль в пьесе» |
| Актеры\_id | Actors\_id | INT | - | FK | Ссылается на «Актеры» |

Таблица 7. Роли

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Состав спектакля (Team performance) | | | | | |
| Название атрибута (руск.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Состав спектакля\_id | Team\_performance\_id | INT | - | PK | Первичный ключ |
| Номер состава | Number\_team | INT | - | - | Нумерация состава целочисленное значение |
| Спектакль\_id | Performance\_id | INT | - | FK | Ссылается на «Спектакль» |
| Роли\_id | Roles\_id | INT | - | FK | Ссылается на «Роли» |

Таблица 8. Состав спектакля

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Тип показа (Type performance) | | | | | |
| Название атрибута (руск.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Тип показа\_id | Type\_performance\_id | INT | - | PK | Первичный ключ |
| Тип показа | Type\_performance | VARCHAR | 14 | - | Длина самого длинного слова в данном словаре |

Таблица 9. Тип показа

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Актеры (Actors) | | | | | |
| Название атрибута (рус.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Актеры\_id | Actors\_id | INT | - | PK | Первичный ключ |
| Фамилия | Lastname | VARCHAR | 45 | - | Длина максимальной фамилии в мире |
| Имя | Firstname | VARCHAR | 45 | - | Размер не меньше максимальной длины фамилии |
| Отчество | Surname | VARCHAR | 45 | - | Размер не меньше максимальной длины фамилии |
| Возраст | Age | INT | - | - | Целочисленное значение |
| Пол | Sex | VARCHAR | 3 | - | Длина самого длинного значения |

Таблица 10. Актеры

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Достижения (Awards) | | | | | |
| Название атрибута (руск.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Достижения\_id | Awards\_id | INT | - | PK | Первичный ключ |
| Достижения\_тип | Awards\_type | VARCHAR | 30 | - | Длина самого длинного слова в данном словаре |

Таблица 11. Достижения

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Достижения актера (Actor awards) | | | | | |
| Название атрибута (руск.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Достижения актера\_id | Actor\_awards\_id | INT | - | PK | Первичный ключ |
| Актер\_id | Actor\_id | INT | - | - | Ссылается на «Актеры» |
| Достижения\_id | Awards\_id | INT | - | - | Ссылается на «Достижения» |

Таблица 12. Достижения актера

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Контракт (Contract) | | | | | |
| Название атрибута (руск.) | Название атрибута (англ.) | Тип | Размер | Ключ | Обоснование типа |
| Контракт\_id | Contract\_id | INT | - | PK | Первичный ключ |
| Дата назначения на роль | Appointment\_date\_id | DATE | - | - | Используется для хранения только даты в формате ‘YYYY-MM-DD’ |
| Гонорар | Salary\_id | INT | - | - | Целочисленное значение |
| Роли\_id | Roles\_id | INT |  |  | Ссылается на «Роли» |
| Актеры\_id | Actors\_id | INT |  |  | Ссылается на «Актеры» |

Таблица 13. Контракт

## Генерация базы данных

Для генерации базы данных был написан скрипт в графическом клиенте MySQL Workbench. Скрипт представлен в Приложение 1.

Процесс создания базы данных:

* С помощью команды CREATE DATABASE на сервере создается база данных Theater. С помощью команды USE данная база данных устанавливается в качестве используемой.
* С помощью команды CREATE TABLE IF NOT EXISTS создаются таблицы, описанные в пункте 2.2. Большинство атрибутов таблиц было объявлено как NOT NULL.
* Для первичных ключей с помощью AUTO\_INCREMENT было установлено автоматическое увеличение значения столбца при добавлении новой записи.

## Заполнение базы данных

Для заполнения базы данных был написан скрипт на языке программирования Java в среде разработки IntelliJ IDEA. Данный скрипт представлен в Приложение 2.

Подключение к созданной базе данных происходит с помощью драйвера MySQL Connector/J. После установления соединения драйвер позволяет отправить SQL-команды серверу.

На Рис.5 представлено количество записей в каждой таблицу. Данные в таблицах распределены равномерно. Например, количество постановок у каждого режиссера примерно одинаковое.

Всего записей 136 338.

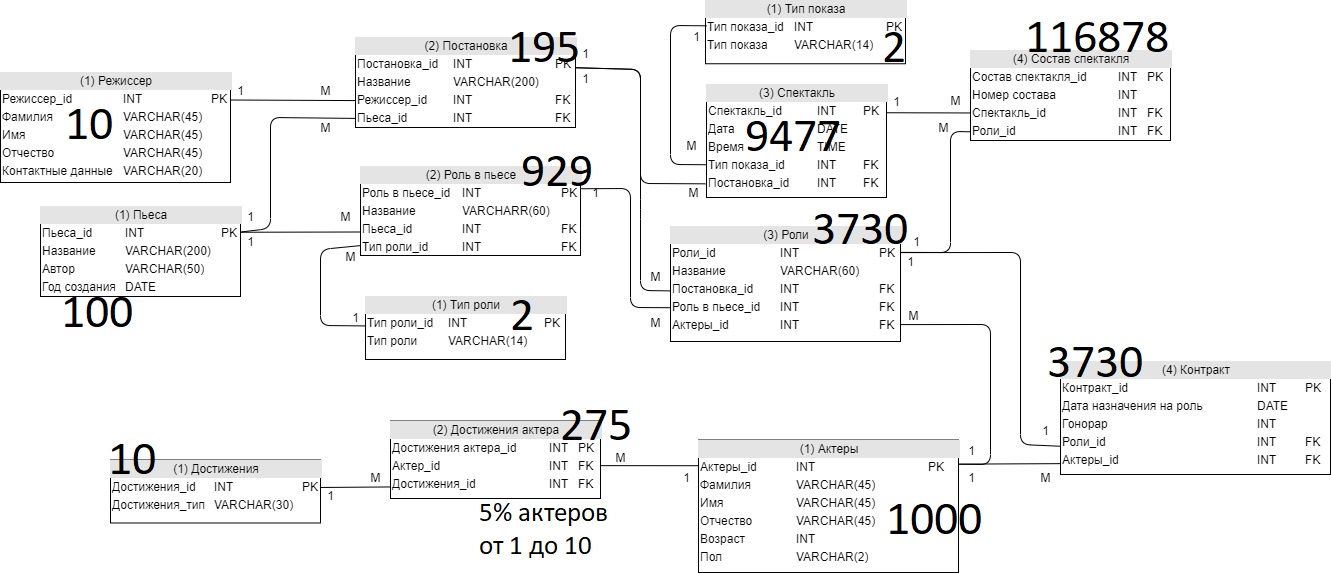


Рисунок 5. Объем данных для заполнения базы данных

# Реализация запросов

### Запрос №1

Формулировка запроса: «Найти режиссера, который поставил спектакль А, где ведущую роль играл актер Б». В данном случае спектакль А имеет id = 25, актер Б - id = 521, ведущая роль role\_type\_id = 1.

Формулировка данного запроса на языке SQL:

SELECT

d.director\_id,

d.lastname,

d.firstname,

d.surname,

d.phone

FROM

Director d

JOIN production pr ON pr.director\_id = d.director\_id

JOIN Performance p ON p.production\_id = pr.production\_id

JOIN Team\_performance t ON t.performance\_id = p.performance\_id

JOIN Roles r ON r.role\_id = t.role\_id

JOIN Role\_in\_the\_play ri ON ri.role\_in\_the\_play\_id = r.role\_in\_the\_play\_id

JOIN Role\_type rt ON rt.role\_type\_id = ri.role\_type\_id

JOIN Actors a ON a.actors\_id = r.actor\_id

WHERE

p.performance\_id = 25 AND

a.actors\_id = 521 AND

rt.role\_type\_id = 1

Результат выполнения данного запроса представлен на Рис. 6.



Рисунок 6. Результат выполнения запроса №1

Время выполнения запроса – 0.078 c.

План выполнения запроса представлен на Рис. 7. План выполнения запроса можно посмотреть с помощью добавления ключевого слова EXPLAIN.

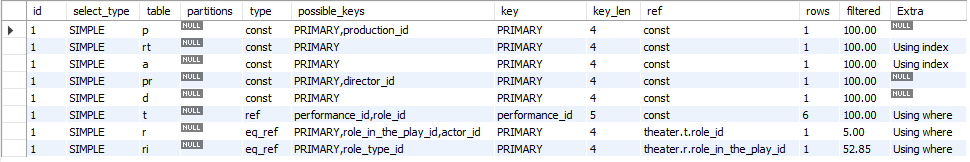


Рисунок 7. План выполнения запроса №1

Тип запроса SIMPLE означает, что данный SELECT запрос не содержит подзапросов или UNION’ов.

Сначала к таблице Performance происходит обращение по первичному ключу. Тип const означает, что таблица содержит не более одной соответствующей строки, которая будет считываться в начале запроса. Далее по индексу происходит обращение к таблицам Role\_type и Actor. Также происходит обращение к таблицам Production и Director. Далее таблица Team\_performance соединяется с таблицей Performance по ключу performance\_id. Таблицы Roles и Role\_in\_the\_play связываются с таблицей Team\_performance и Roles соответственно. Тип eq\_ref означает, что для каждой комбинации строк из предыдущих таблиц будет считываться одна строка из этой таблицы.

### Запрос №2

Формулировка запроса: «Посчитать число актеров, которые участвовали во всех спектаклях A». В данном случае спектакль имеет id = 9.

Формулировка запроса на языке SQL:

SELECT

count(actors\_id)

FROM

Actors a

JOIN Roles r on r.actor\_id = a.actors\_id

JOIN Team\_performance tr on tr.role\_id = r.role\_id

JOIN Performance p on p.performance\_id = tr.performance\_id

WHERE

p.performance\_id = 9

Время выполнения запроса – 0.016 с.

Результат выполнения запроса представлен на Рис. 8



Рисунок 8. Результат выполнения запроса №2

План выполнения запроса представлен на Рис. 9.

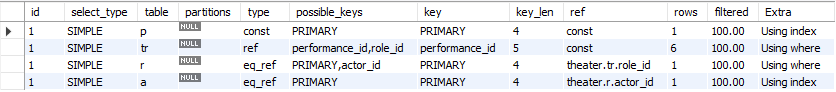


Рисунок 9. План выполнения запроса №2

Тип запроса SIMPLE означает, что данный SELECT запрос не содержит подзапросов или UNION’ов.

Сначала к таблице Performance по индексу, который является первичным ключом происходит обращение. Далее по ключу performance\_id с использование where происходит объединение с таблицей Team\_performance. Затем таблица Roles объединяется с Team\_performance по первичному ключу. Таблица Actor соединяется с таблицей Roles по индексу, который является первичным ключом.

### Запрос №3

#### Запрос №3.1

Формулировка запроса: «Посчитать число спектаклей для каждой постановки». Построить гистограмму.

Формулировка запроса на языке SQL:

SELECT

Count(performance\_id) as PerformanceCount,

performance\_name

FROM

Performance p

Join Production pr on p.production\_id = pr.production\_id

GROUP BY

pr.production\_id

Время выполнения запроса – 0.016 с.

Результат выполнения запроса представлен на Рис. 10. Представлены только первые 20 строк результата.

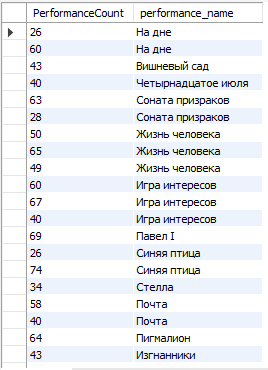


Рисунок 10. Результат выполнения запроса №3.1

План выполнения запроса представлен на Рис. 11.



Рисунок 11. План выполнения запроса №3.1

Тип запроса SIMPLE означает, что данный SELECT запрос не содержит подзапросов или UNION’ов.

Сначала происходит обращение по индексу, который является первичным ключом к таблице Production. Далее таблица Performance соединяется с Production по ключу production\_id.

Для запроса построена гистограмма.

#### Запрос №3.2

Формулировка запроса: «Посчитать число спектаклей для каждой пьесы». Построить гистограмму.

Формулировка запроса на языке SQL:

SELECT

Count(performance\_id) as PerformanceCount,

performance\_name

FROM

Performance p

join production pr on pr.production\_id = p.production\_id

join Play pl on pl.play\_id = pr.play\_id

GROUP BY

pl.play\_id

Время выполнения запроса – 0.016 с.

Результат выполнения запроса представлен на Рис. 12. Представлены только первые 20 строк результата.



Рисунок 12. Результат выполнения запроса №3.2

План выполнения запроса представлен на Рис. 13

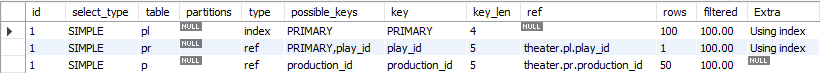


Рисунок 13. План выполнения запроса №3.2

Тип запроса SIMPLE означает, что данный SELECT запрос не содержит подзапросов или UNION’ов.

Сначала происходит обращение по индексу, который является первичным ключом к таблице Play. Далее таблица Production соединяется с Play по ключу play\_id. Далее таблица Performance соединяется с Production по ключу production\_id.

Для данного запроса была построена гистограмма.

### Запрос №4

Формулировка запроса: «Посчитать число актеров с одинаковым числом ролей».

Формулировка запроса на языке SQL:

SELECT

count(actor\_id) as ActorCount,

RolesCount

FROM

(SELECT

Roles.actor\_id,

count(role\_id) AS RolesCount

FROM

Roles

JOIN Actors ON Actors.actors\_id = Roles.actor\_id

GROUP BY

actor\_id) AS tmp

GROUP BY

RolesCount

Время выполнения – 0.016 с.

Результат выполнения представлен на Рис. 14.

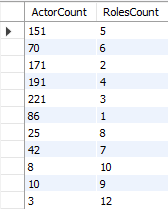


Рисунок 14. Результат выполнения запроса №4

План выполнения запроса представлен на Рис. 15.

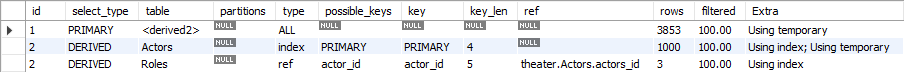


Рисунок 15. План выполнения запроса №4

Запрос с id = 1 является внешним, он ожидает результата от подзапроса с id = 2 и соединяется с временной таблицей из подзапроса. В подзапросе происходит обращение по индексу, который является первичным ключом к таблице Actors. Далее таблица Roles соединяется с таблицей Actors по ключу actors\_id.

### Запрос №5

#### Запрос №5.1

Формулировка запроса: «Найти актеров в максимальном и минимальном числе спектаклей».

Формулировка запроса на языке SQL:

SELECT

count(p.performance\_id) as PerformanceCount,

Actors.lastname,

Actors.firstname,

Actors.surname

FROM

Performance p

JOIN Team\_performance tp on tp.performance\_id = p.performance\_id

JOIN Roles on tp.role\_id = Roles.role\_id

JOIN Actors on Roles.actor\_id = Actors.actors\_id

GROUP BY

Actors.actors\_id

HAVING

PerformanceCount in (select max(PerformanceCount)

FROM (

SELECT count(p.performance\_id) as PerformanceCount, Actors.\*

FROM Performance p

JOIN Team\_performance tp on tp.performance\_id = p.performance\_id

JOIN Roles on tp.role\_id = Roles.role\_id

JOIN Actors on Roles.actor\_id = Actors.actors\_id

GROUP BY Actors.actors\_id) as tmp)

Время выполнения – 0.844 для минимального, 0.656 для максимального

Результат выполнения представлен на Рис. 16, Рис. 17



Рисунок 16. Выполнение запроса с минимальным значением



Рисунок 17. Выполнение запроса с максимальным значением

План выполнения запроса представлен на Рис. 18.

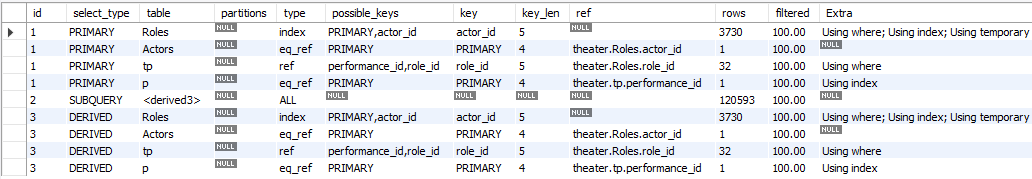


Рисунок 18. План выполнения запроса №5.1

Запрос с id = 1 является внешним. Сначала в нем происходит обращение к таблице Roles по ключу actor\_id. Далее таблица Actors соединяется с Roles по первичному ключу. Затем таблица Team\_performance соединяется с таблицей Roles по ключу role\_id, а таблица Performance соединяется с таблицей Team\_performance по первичному ключу. Подзапрос с id = 2 ожидает результата от подзапроса с id = 3, просматривая его с целью найти максимальное или минимальное значение. В подзапросе с id = 3 сначала происходит обращение по ключу actor\_id к таблице Roles\_id. Далее Actors соединяется с Roles по первичному ключу. Таблица Team\_performance соединяется с таблицей Roles по ключу role\_id. Таблица Performance соединяется с таблицей Team\_performance по первичному ключу.

#### Запрос №5.2

Формулировка запроса: «Найти актеров в максимальном и минимальном числе постановок».

Формулировка запроса на языке SQL:

SELECT

count(Production.production\_id) as ProductionCount,

Actors.lastname,

Actors.firstname,

Actors.surname

FROM

Production

JOIN Roles on Production.production\_id = Roles.production\_id

Join Actors on Roles.actor\_id = Actors.actors\_id

GROUP BY

Actors.actors\_id

HAVING

ProductionCount in (select min(ProductionCount)

FROM (

SELECT

count(Production.production\_id) as ProductionCount, Actors.\*

FROM Actors join Roles on Roles.actor\_id = Actors.actors\_id

JOIN Production on Production.production\_id = Roles.production\_id

GROUP BY

Actors.actors\_id) as tmp)

Время выполнения – 0.046 c. для максимального, 0. 031 с. для минимального.

Результат выполнения для максимального количества представлен на Рис. 19.

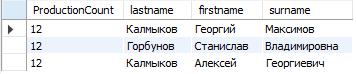


Рисунок 19. Результат выполнения запроса №5.2. max

Результат выполнения для минимального количества представлен на Рис. 20.

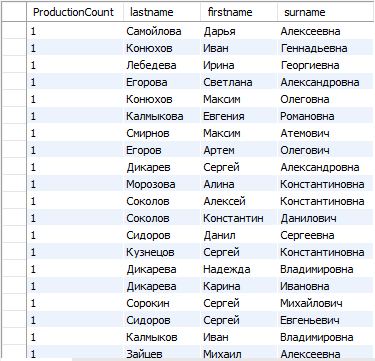


Рисунок 20. Результат выполнения запроса №5.2 min

План выполнения запроса представлен на Рис. 21.

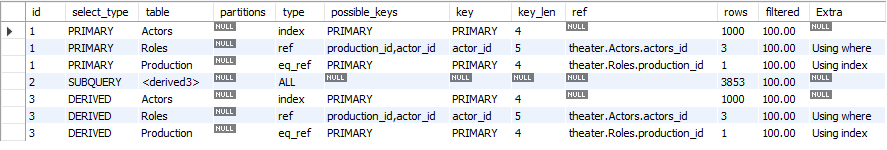


Рисунок 21. План выполнения запроса №5.2

Запрос с id = 1 является внешним и ожидает результат от подзапроса с id = 2. В нем происходит обращение к таблице Actors по первичному ключу. Таблица Roles соединяется с таблицей Actors по ключу actor\_id. Подзапрос с id = 2 ожидает результата от подзапроса с id = 3, просматривая его с целью найти максимальное или минимальное значение. В подзапросе с id = 3 сначала происходит обращение к таблице Actors по первичному ключу, далее таблица Roles соединяется с Actors по ключу actor\_id. Затем таблица Production соединяется с таблицей Roles по ключу production\_id.

### Запрос №6

Формулировка запроса: «Найти всех актеров, которые участвовали в большем числе постановок, чем актер А». В данном случае актер А имеет id = 35, количество постановок данного актера – 6.

Формулировка запроса на языке SQL:

SELECT

count(Production.production\_id) as ProductionCount,

Actors.lastname,

Actors.firstname,

Actors.surname

FROM

Production

JOIN Roles on Production.production\_id = Roles.production\_id

JOIN Actors on Roles.actor\_id = Actors.actors\_id

GROUP BY Actors.actors\_id

HAVING

ProductionCount > (select count(Production.production\_id) as ProductionCount

FROM

Production

JOIN Roles on Production.production\_id = Roles.production\_id

JOIN Actors on Roles.actor\_id = Actors.actors\_id

WHERW

Actors.actors\_id = 35)

Время выполнения – 0.015 с.

Результат выполнения представлен на Рис. 22.



Рисунок 22. Результат выполнения запроса №6

План выполнения запроса представлен на Рис. 23.

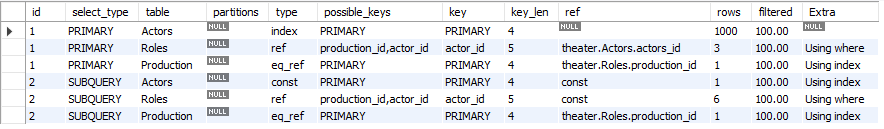


Рисунок 23. План выполнения запроса №6

Запрос с id = 1 является внешним и ожидает результат от подзапроса с id = 2. В нем происходит обращение к таблице Actors по первичному ключу, далее таблица Roles соединяется с Actors по ключу actor\_id. Затем таблица Production соединяется с таблицей Roles по ключу production\_id. В подзапросе с id = 2 выбирается одна запись в таблице Actors, соответствующая actors\_id = 35. Далее таблица Roles соединяется с Actors по ключу actor\_id. Затем таблица Production соединяется с таблицей Roles по ключу production\_id.

### Запрос №7

Формулировка запроса: «Посчитать число задействованных актеров для каждого режиссера в премьере и плановом показе». Построить 3D – гистограмму.

Формулировка запроса на языке SQL:

SELECT

Director.director\_id as director,

Performance\_type.performance\_type as performance\_types,

(SELECT

count(Actors.actors\_id)

FROM Actors

JOIN Roles on Roles.actor\_id = Actors.actors\_id

JOIN Team\_performance on Team\_performance.role\_id = Roles.role\_id

JOIN Performance on Performance.performance\_id = Team\_performance.performance\_id

JOIN Performance\_type on Performance.performance\_type\_id = Performance\_type.performance\_type\_id

JOIN Production on Performance.production\_id = Production.production\_id

JOIN Director on Director.director\_id = Production.director\_id

WHERE

Director.director\_id = director and

Performance\_type.performance\_type = performance\_types) as countActors

FROM

Performance\_type

JOIN Director

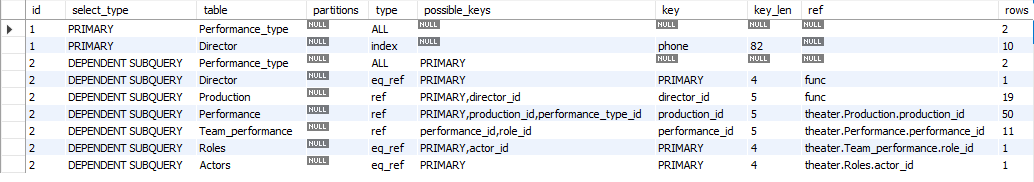
Время выполнения – 0.375 c.

Результат выполнения представлен на Рис. 24.



Рисунок 24. Результат выполнения запроса №7

План выполнения запроса представлен на Рис. 25.



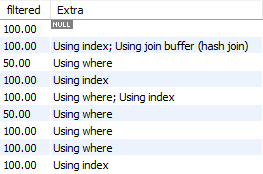


Рисунок 25. План выполнения запроса №7

Запрос с id = 1 является внешним. В нем происходит обращение к таблице Performance\_type. Затем обращение к таблице Director по индексу.

Подзапрос с id = 2 зависит от внешнего запроса. В подзапросе сначала происходит обращение к таблице Performance\_type по первичному ключу. Затем для каждой записи таблицы происходит обращение к Director. С ней соединяется таблица Production по ключу production\_id. Далее таблица Performance соединяется с таблицей Production по ключу production\_id. Таблица Team\_performance соединяется с таблицей Performance по ключу performance\_id. Таблица Roles соединяется с таблицей Team\_performance по ключу role\_id. Таблица Actors соединяется с таблицей Roles по ключу actor\_id.

Для данного запроса была построена 3D-гистограмма.

### Запрос №8

Формулировка запроса: «Посчитать число постановок с одинаковым числом актеров». Построить гистограмму.

Формулировка запроса на языке SQL:

SELECT

count(production\_id) as ProductionCount,

ActorsCount

FROM

(SELECT

Production.production\_id,

count(actors\_id) as ActorsCount

FROM

Production

JOIN Roles on Production.production\_id = Roles.production\_id

JOIN Actors on Actors.actors\_id = Roles.actor\_id

GROUP BY

production\_id) as tmp

GROUP BY

ActorsCount

Время выполнения – 0.016 c.

Результат выполнения представлен на Рис. 26.

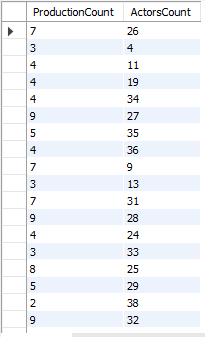


Рисунок 26. Результат выполнения запроса №8

План выполнения запроса представлен на Рис. 27.

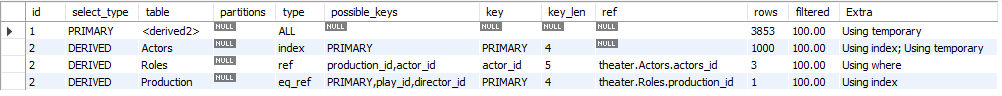


Рисунок 27. План выполнения запроса №8

Запрос с id = 1 является внешним и ожидает результата от подзапроса с id = 2. В подзапросе сначала происходит обращение по индексу, который является первичным ключом к таблице Actors. Затем таблица Roles соединяется с таблицей Actors по ключу actors\_id, далее таблица Production соединяется с таблицей Roles по первичному ключу.

Для данного запроса была построена гистограмма.

# Заключение

В ходе работы была выбрана и проанализирована предметная область театр. В данной предметной области был выбран и автоматизирован процесс внутреннего функционала театра. Также были сформулированы основные цели и функции системы. Были выделены сущности и их атрибуты, а также объяснена их необходимость. Для выбранных сущностей была спроектирована схема объектов и ER-диаграмма. По схеме объектов была спроектирована схема базы данных на двух языках. На языке SQL был написан скрипт создания базы данных. На языке Java написан скрипт заполнения базы данных с общим числом данных более ста тысяч. Было реализовано десять запросов на выборку на языке SQL для спроектированной базы данных. Для четырех запросов были построены гистограммы. Выполнение каждого запроса было объяснено.

# Список литературы

1. MySQL 5.7 Reference Manual. [Электронный ресурс]: MySQL. Режим доступа: <https://dev.mysql.com/doc/refman/5.7/en/> (дата обращения: 01.06.2020).

# Приложение 1

CREATE TABLE IF NOT EXISTS Director(

director\_id int primary key auto\_increment,

lastname varchar(45) not null,

firstname varchar(45) not null,

surname varchar(45),

phone varchar(20) not null unique);

CREATE TABLE IF NOT EXISTS Play(

play\_id int primary key auto\_increment,

play\_name varchar(200) not null,

author varchar(50) not null,

date\_of\_creation DATE not null);

CREATE TABLE IF NOT EXISTS Awards(

awards\_id int primary key auto\_increment,

awards\_type varchar(30) not null);

CREATE TABLE IF NOT EXISTS Role\_type(

role\_type\_id int primary key auto\_increment,

role\_type varchar(14) not null);

CREATE TABLE IF NOT EXISTS Actors(

actors\_id int primary key auto\_increment,

lastname varchar(45) not null,

firstname varchar(45) not null,

surname varchar(45),

age int check(age > 18 and age < 80) not null,

sex varchar(3) not null);

CREATE TABLE IF NOT EXISTS Performance\_type(

performance\_type\_id int primary key auto\_increment,

performance\_type varchar(14) not null);

CREATE TABLE IF NOT EXISTS Production(

production\_id int primary key auto\_increment,

production\_name varchar(200) not null,

director\_id int,

play\_id int,

foreign key (play\_id) references Play(play\_id) on delete restrict on update restrict,

foreign key (director\_id) references Director(director\_id) on delete restrict on update cascade);

CREATE TABLE IF NOT EXISTS Role\_in\_the\_play(

role\_in\_the\_play\_id int primary key auto\_increment,

role\_name varchar(60) not null,

play\_id int,

role\_type\_id int,

foreign key (play\_id) references Play(play\_id) on delete restrict on update restrict,

foreign key (role\_type\_id) references Role\_type(role\_type\_id) on delete restrict on update restrict);

CREATE TABLE IF NOT EXISTS Actor\_awards(

actor\_awards\_id int primary key auto\_increment,

actor\_id int,

awards\_id int,

foreign key (actor\_id) references Actors(actors\_id) on delete restrict on update restrict,

foreign key (awards\_id) references Awards(awards\_id) on delete restrict on update restrict);

CREATE TABLE IF NOT EXISTS Roles(

role\_id int primary key auto\_increment,

role\_name varchar(60) not null,

production\_id int,

role\_in\_the\_play\_id int,

actor\_id int,

foreign key (production\_id) references Production(production\_id) on delete restrict on update restrict,

foreign key (role\_in\_the\_play\_id) references Role\_in\_the\_play(role\_in\_the\_play\_id) on delete restrict on update restrict,

foreign key (actor\_id) references Actors(actors\_id) on delete restrict on update restrict);

CREATE TABLE IF NOT EXISTS Performance(

performance\_id int primary key auto\_increment,

performance\_name varchar(200) not null,

production\_id int,

date\_perfomance date not null,

start\_time time not null,

performance\_type\_id int,

foreign key (production\_id) references Production(production\_id) on delete restrict on update restrict,

foreign key (performance\_type\_id) references Performance\_type(performance\_type\_id)on delete restrict on update restrict);

CREATE TABLE IF NOT EXISTS Team\_performance(

team\_id int primary key auto\_increment,

performance\_id int,

role\_id int,

foreign key (performance\_id) references Performance(performance\_id) on delete restrict on update restrict,

foreign key (role\_id) references Roles(role\_id) on delete restrict on update restrict);

CREATE TABLE IF NOT EXISTS Contract(

contract\_id int primary key auto\_increment,

assignment\_date date not null,

salary int check(salary>5000) not null,

role\_id int,

actor\_id int,

foreign key (role\_id) references Roles(role\_id) on delete restrict on update restrict,

foreign key (actor\_id) references Actors(actors\_id) on delete restrict on update restrict);

# Приложение 2

**public class** Programm {  
 **private static** String[] *firstNameMale* = {**"Андрей"**, **"Владимир"**, **"Александр"**, **"Геннадий"**, **"Петр"**, **"Сергей"**, **"Алексей"**,  
 **"Константин"**, **"Георгий"**, **"Олег"**, **"Михаил"**, **"Евгений"**, **"Роман"**, **"Павел"**, **"Илья"**, **"Иван"**, **"Артем"**, **"Станислав"**,  
 **"Данил"**, **"Максим"**};  
 **private static** String[] *firstNameFemale* = {**"Мария"**, **"Анна"**, **"Ирина"**, **"Дарья"**, **"Елена"**, **"Ольга"**, **"Виктория"**,  
 **"Надежда"**, **"Светлана"**, **"Юлия"**, **"Яна"**,**"Галина"**, **"Татьяна"**, **"Александра"**, **"Алина"**, **"Кристина"**, **"Анастасия"**,  
 **"Евгения"**, **"Карина"**, **"Марина"**};  
  
 **private static** String[] *surNameMale* = {**"Олегович"**, **"Андреевич"**, **"Константинович"**, **"Алексеевич"**, **"Сергеевич"**, **"Георгиевич"**,  
 **"Петрович"**, **"Геннадьевич"**, **"Александрович"**, **"Владимирович"**, **"Олеговна"**,**"Андреевна"**, **"Константиновна"**, **"Алексеевна"**,  
 **"Сергеевна"**, **"Георгиевна"**, **"Петровна"**, **"Геннадьевна"**, **"Александровна"**, **"Владимировна"**, **"Ильич"**, **"Максимов"**, **"Романович"**,  
 **"Павлович"**, **"Атемович"**, **"Евгеньевич"**, **"Данилович"**, **"Станиславович"**, **"Михайлович"**, **"Иванович"**};  
  
 **private static** String[] *surNameFemale* = {**"Олеговна"**,**"Андреевна"**, **"Константиновна"**, **"Алексеевна"**,  
 **"Сергеевна"**, **"Георгиевна"**, **"Петровна"**, **"Геннадьевна"**, **"Александровна"**, **"Владимировна"**, **"Романовна"**, **"Ивановна"**,  
 **"Максимовна"**, **"Артемовна"**, **"Евгеньевна"**, **"Даниловна"**, **"Павловна"**, **"Станиславовна"**, **"Михайловна"**, **"Ильинична"**};  
  
 **private static** String[] *lastNameMale* = {**"Петров"**, **"Иванов"**, **"Сидоров"**, **"Морозов"**, **"Конюхов"**, **"Сорокин"**, **"Дикарев"**,  
 **"Колесников"**, **"Самойлов"**, **"Горбунов"**, **"Калмыков"**, **"Марков"**, **"Соколов"**, **"Смирнов"**, **"Кузнецов"**, **"Лебедев"**, **"Егоров"**,  
 **"Павлов"**, **"Орлов"**, **"Зайцев"**};  
  
 **private static** String[] *lastNameFemale* = {**"Петрова"**,**"Иванова"**, **"Сидорова"**, **"Морозова"**, **"Конюхова"**, **"Сорокина"**,  
 **"Дикарева"**, **"Колесникова"**, **"Самойлова"**, **"Горбунова"**, **"Калмыкова"**, **"Маркова"**, **"Соколова"**, **"Смирнова"**, **"Кузнецова"**,  
 **"Лебедева"**, **"Егорова"**, **"Павлова"**, **"Орлова"**, **"Зайцева"**};  
  
 **private static** String[] *playName* = {**"На дне"**, **"Вишневый сад"**, **"Четырнадцатое июля"**, **"Соната призраков"**, **"Жизнь человека"**,  
 **"Игра интересов"**, **"Павел I"**, **"Синяя птица"**, **"Стелла"**, **"Почта"**, **"Пигмалион"**, **"Изгнанники"**, **"Человек"**, **"Мечтатели"**,  
 **"Корабль"**, **"Квартира"**, **"Бег"**, **"Атилла"**, **"Опера"**, **"Самоубийца"**, **"Потеря"**, **"Собака"**, **"Клоун"**, **"Баня"**, **"Свадьба"**, **"Таня"**,  
 **"Наш город"**, **"Переполох"**, **"Машенька"**, **"Долгий день уходит в ночь"**, **"Мухи"**, **"Дракон"**, **"Золотая антилопа"**, **"Служанки"**, **"Банкрот"**,  
 **"Смерть коммивояжера"**, **"Певица"**, **"Дом на окраине"**, **"Глобус"**, **"Мышеловка"**, **"Вдова Капет"**, **"Обыкновенное чудо"**,  
 **"Визит старой дамы"**, **"Оглянись во гневе"**, **"Чайная"**, **"Семья"**, **"Призраки среди нас"**, **"В поисках радости"**,  
 **"Носорог"**, **"Сторож"**, **"Иркутская история"**, **"Картотека"**, **"Коллекция"**, **"Физики"**, **"Дознание"**, **"Расследование"**, **"Старший сын"**,  
 **"Варшавская история"**, **"Утиная охота"**, **"Самый правдивый"**, **"Осення соната"**, **"Минута покоя"**, **"Свободная пара"**, **"Роза"**,  
 **"Гипотеза"**, **"Змеиная кожа"**, **"Щи"**, **"Там вдали"**, **"Разбитый кувшин"**, **"Фауст"**, **"Женитьба"**, **"Маскарад"**, **"Кокульный домик"**,  
 **"Страна чудес"**, **"Ревизор"**, **"Смотритель"**, **"Стакан воды"**, **"Деньги"**, **"Месяц в деревне"**, **"Маленькая история"**, **"Хулиган"**,  
 **"Гроза"**, **"Буря"**, **"Полет"**, **"Враг"**, **"Хранитель"**, **"Банкротство"**, **"Мелодия жизни"**, **"Враг народа"**, **"Русалочка"**, **"Принц"**,  
 **"Власть тьмы"**, **"Слепые"**, **"Суеверие"**, **"Одинокие"**, **"Глупец и смерть"**, **"Счастливые"**, **"Танцы под дождем"**, **"Чайка"**,  
 **"Кукушка"**, **"Три сестры"**, **"Простая девушка"**, **"Зори"**, **"Король мира"**, **"Тайна семьи"**, **"Новая страница"**, **"Песнь богатыря"**};  
  
 **private static** String[] *author* = {**"М. Горький"**, **"А. Чехов"**, **"Р. Роллан"**, **"А. Стриндберг"**, **"Л. Андреев"**, **"Х. Мартинес"**,  
 **"Д. Мережковский"**, **"М. Метерлинк"**, **"Г. Ксенопулос"**, **"Р. Тагор"**, **"Д. Шоу"**, **"Д.Джойс"**, **"Э. Толлер"**, **"Р. Музиль"**,  
 **"Н. Евреинов"**, **" Н. Белоусов"**, **"К. Кокто"**, **"М. Булгаков"**, **"Е. Замятин"**, **"Н. Короблев"**};  
  
  
 **private static** String[] *sex* = {**"м."**, **"ж."**};  
  
 **private static** String[] *RoleName* = {**"Машенька"**, **"Девочка со двора"**, **"Служанка"**, **"Анастасия"**,  
 **"Соня"**, **"Старушка"**, **"Дочь служанки"**, **"Матушка"**, **"Королева"**, **"Таисия"**, **"Екатерина"**, **"Марья Ивановна"**, **"Катерина"**,  
 **"Клавдия Петровна"**, **"Король"**, **"Андрюша"**, **"Отец"**, **"Петруша"**, **"Петр Иванович"**, **"Петр Иванович"**,  
 **"Мечтатель"**, **"Аркадий Иванович"**,**"Творец"**, **"Писатель"**, **"Художник"**, **"Федор Петрович"**};  
  
  
 **private static** String addAuthor(){  
 Random rand = **new** Random();  
 **int** index = rand.nextInt(*author*.**length**);  
 **return "'"**+*author*[index]+**"'"**;  
 }  
  
 **private static** String addName(String[] mas){  
 Random rand = **new** Random();  
 **int** index = rand.nextInt(mas.**length**);  
 **return "'"**+mas[index]+**"'"**;  
 }  
  
 **private static** String addDate(){  
 **return "'"** + (**int**)(Math.*random*()\*50+1900)+ **"-"** + (**int**)(Math.*random*()\*11+1) + **"-"** + (**int**)(Math.*random*()\*26+1) + **"'"**;  
 }  
  
 **private static** String addDatePerformance(){  
 **return "'"** + (**int**)(Math.*random*()\*10+2018)+ **"-"** + (**int**)(Math.*random*()\*11+1) + **"-"** + (**int**)(Math.*random*()\*26+1) + **"'"**;  
 }  
 **private static** String addDateContract(){  
 **return "'"** + (**int**)(Math.*random*()\*4+2015)+ **"-"** + (**int**)(Math.*random*()\*11+1) + **"-"** + (**int**)(Math.*random*()\*26+1) + **"'"**;  
 }  
  
  
 **private static** String addAge(){  
 **return "'"** + (**int**)(Math.*random*()\*50+18)+**"'"**;  
 }  
  
 **private static int** addRandomId (**int** c) {  
 Random rand = **new** Random( ) ;  
 **return** 1 + rand.nextInt (c) ;  
 }  
  
 **private static** String addTime(){  
 String min[] = {**"00"**, **"30"**};  
 **int** i = (**int**)(Math.*random*()\*2);  
 **return "'"** + (**int**)(Math.*random*()\*10+12)+ **":"** + min[i]+**":00'"**;  
 }  
  
  
 **public static void** main(String[] args){  
 **try**{  
 String url = **"jdbc:mysql://localhost:3306/Theater?serverTimezone=Europe/Moscow&useSSL=false"** +  
 **"&useUnicode=true&characterEncoding=utf8"**;  
 String username = **"root"**;  
 String password =**"223344ya"**;  
 Class.*forName*(**"com.mysql.cj.jdbc.Driver"**).getDeclaredConstructor().newInstance();  
 **try**(Connection connection = DriverManager.*getConnection*(url, username, password)){  
 System.***out***.println(**"Connection successful!"**);  
 Statement statement = connection.createStatement();  
 Statement st1 = connection.createStatement();  
 Statement st2 = connection.createStatement();  
 Statement st3 = connection.createStatement();  
 Statement st4 = connection.createStatement();  
 String mySqlInsert;  
  
 *//актеры +++* **for**(**int** i = 0; i < 1000; i++){  
 **if**(i % 2 == 1)  
 mySqlInsert = **"insert into Actors(lastname, firstname, surname, age, sex)"** +  
 **"values("** + *addName*(*lastNameMale*) +  
 **","**+ *addName*(*firstNameMale*)+**","**+*addName*(*surNameMale*)+**","**+*addAge*()+**", '"**+*sex*[0]+ **"')"**;  
 **else** mySqlInsert = **"insert into Actors(lastname, firstname, surname, age, sex)"** +  
 **"values("** + *addName*(*lastNameFemale*) +  
 **","**+ *addName*(*firstNameFemale*)+**","**+*addName*(*surNameFemale*)+**","**+*addAge*()+**", '"**+*sex*[1]+ **"')"**;  
 statement.executeUpdate(mySqlInsert);  
 }  
 System.***out***.println(**"Actors added!"**);  
  
 *// Пьесы +++* **for**(**int** i = 0; i < 100;i++){  
 mySqlInsert = **"insert into Play(play\_name, author, date\_of\_creation)"** +  
 **"values( '"**+*playName*[i]+**"',"**+*addAuthor*()+ **","**+*addDate*()+**")"**;  
 statement.executeUpdate(mySqlInsert);  
 }  
 System.***out***.println(**"Plays added!"**);  
  
 *//Роль в пьесе* Random rand = **new** Random();  
 **int** indexPlay = 0;  
 **int** countRoleInPlay = 0;  
 **for**(**int** i = 1; i <= 100; i++){  
 indexPlay = rand.nextInt(17)+1;  
 **for**(**int** j = 1; j <= indexPlay; j++){  
 **if**(j % 2 == 1){  
 mySqlInsert = **"insert into Role\_in\_the\_play(role\_name, play\_id, role\_type\_id)"** +  
 **"values("**+*addName*(*RoleName*)+**","**+**"'"**+i+**"',"**+**"'1')"**;  
 } **else**{  
 mySqlInsert = **"insert into Role\_in\_the\_play(role\_name, play\_id, role\_type\_id)"** +  
 **"values("**+*addName*(*RoleName*)+**","**+**"'"**+i+**"',"**+**"'2')"**;  
 }  
 statement.executeUpdate(mySqlInsert);  
 countRoleInPlay++;  
 }  
 }  
 System.***out***.println(**"Role in the play added!"**+countRoleInPlay);  
  
 *//Постановка +++* **int** indexProduction = 0;  
 **int** indexDirector = 0;  
 **int** indexAllProduction = 0;  
 **for**(**int** i = 1; i<=100;i++){  
 indexProduction = rand.nextInt(3)+1;  
 **for**(**int** j = 0; j < indexProduction; j++){  
 indexDirector = rand.nextInt(10)+1;  
 mySqlInsert = **"insert into Production(production\_name, director\_id, play\_id)"** +  
 **"values((select play\_name from Play where play\_id="**+ i+**"), '"**+indexDirector+**"',"**+  
 **"'"**+i+**"')"**;  
 statement.executeUpdate(mySqlInsert);  
 indexAllProduction++;  
 }  
 }  
 System.***out***.println(**"Production added!"**+indexAllProduction);  
  
 *//Достижения актера ++* **int** indexAwards =0;  
 **int** randomActor = 0;  
 **int** countAwards = 0;  
 **for**(**int** i =1;i < 51; i++){  
 randomActor = rand.nextInt(1000)+1;  
 indexAwards =rand.nextInt(10)+1;  
 **for**(**int** j = 1; j<=indexAwards;j++){  
 mySqlInsert = **"insert into Actor\_awards(actor\_id, awards\_id)"** +  
 **"values('"**+randomActor+**"','"**+*addRandomId*(10)+**"')"**;  
 statement.executeUpdate(mySqlInsert);  
 countAwards++;  
 }  
 }  
 System.***out***.println(**"Actor awards added!"**+countAwards);  
  
 *//Роли ++* **int** indexPlay\_id =0;  
 **int** indexCountRole = 0;  
 **int** countRoles = 0;  
 **for**(**int** i = 1; i<= indexAllProduction; i++) {  
 ResultSet resultPlay = st1.executeQuery(**"select \* from Production where production\_id="** + i);  
 **while** (resultPlay.next()) {  
 indexPlay\_id = resultPlay.getInt(4);  
 }  
 ResultSet resultRolesNew = st3.executeQuery(**"select \* from Role\_in\_the\_play where "** +  
 **"play\_id="** + indexPlay\_id);  
 **while** (resultRolesNew.next()) {  
 countRoles = rand.nextInt(3)+1;  
 **for**(**int** j = 0; j<countRoles;j++){  
 mySqlInsert = **"insert into Roles(role\_name, production\_id, role\_in\_the\_play\_id, actor\_id) "** +  
 **"values('"** + resultRolesNew.getString(2) + **"', '"** + i + **"', '"** +  
 resultRolesNew.getInt(1) + **"',"** + *addRandomId*(1000) + **")"**;  
 statement.executeUpdate(mySqlInsert);  
 indexCountRole++;  
 }  
 }  
 }  
 System.***out***.println(**"Roles added!"**+indexCountRole);  
  
 *//Спектакль ++* **int** indexPerformance = 0;  
 **int** countPerformance = 0;  
 **int** randType = 0;  
 **for**(**int** i =1; i<= indexAllProduction;i++){  
 indexPerformance = rand.nextInt(50)+25;  
 **for**(**int** j =0; j<indexPerformance;j++){  
 randType = rand.nextInt(2)+1;  
 mySqlInsert = **"insert into Performance(performance\_name, production\_id,date\_performance, start\_time,"** +  
 **"performance\_type\_id)"** +  
 **"values((select production\_name from Production where production\_id ="**+i+**"), '"**+i+**"', "** +*addDatePerformance*()+**", "**+ *addTime*()+**", '"**+randType+**"')"**;  
 countPerformance++;  
 statement.executeUpdate(mySqlInsert);  
 }  
 }  
 System.***out***.println(**"Performance added!"**+ countPerformance);  
  
 *//Состав спектакля* **int** indexPerformance\_id =0;  
 **int** indexTeam = 1;  
 **int** countTeam = 0;  
 **int** roleCountTeam = 0;  
 **int** randCountTeam =0;  
  
 **for**(**int** i = 1; i<=countPerformance;i++){  
 ArrayList<Integer> vecRole = **new** ArrayList<Integer>();  
 ResultSet resultPerformance = st1.executeQuery(**"select \* from Performance where "** +  
 **"performance\_id="**+i);  
 **while**(resultPerformance.next()){  
 indexPerformance\_id = resultPerformance.getInt(3);  
 }  
 ResultSet resultR = st2.executeQuery(**"select \* from Roles where "** +  
 **"production\_id="**+ indexPerformance\_id);  
 **while**(resultR.next()){  
 vecRole.add(resultR.getInt(1));  
 }  
 Collections.*shuffle*(vecRole);  
 **if**(vecRole.size() > 16){  
 randCountTeam = 16;  
 } **else** {  
 randCountTeam = vecRole.size();  
 }  
 **for**(**int** j = 0; j < randCountTeam; j++){  
 mySqlInsert = **"insert into Team\_performance(number\_team, performance\_id, role\_id)"** +  
 **"values('"**+indexTeam+**"', '"**+i+**"', '"**+vecRole.get(j)+**"')"**;  
 statement.executeUpdate(mySqlInsert);  
 countTeam++;  
 }  
 indexTeam++;  
 }  
  
 System.***out***.println(**"Team performance added!"**+countTeam);  
  
  
 *//Контракт +++* **int** salary = 0;  
 **int** countContract = 0;  
 **for**(**int** i = 1; i<=indexCountRole;i++){  
 salary = rand.nextInt(50000)+30000;  
 mySqlInsert = **"insert into Contract(assignment\_date, salary, role\_id, actor\_id)"** +  
 **"values("**+ *addDateContract*() + **", '"** + salary+**"', '"**+i+**"',"**+**"(select actor\_id from Roles where "** +  
 **"role\_id="**+i+**"))"**;  
 statement.executeUpdate(mySqlInsert);  
 countContract++;  
 }  
 System.***out***.println(**"Contract added!"**+countContract);  
 }  
  
 } **catch** (Exception ex ) {  
 System.***out***.println(**"Connection failed..."**);  
  
 System.***out***.println(ex);  
 }  
 }  
  
};