**Концептуальное проектирование** – создание концептуального представления базы данных, включающее определение типов важнейших сущностей и существующих между ними связей. Описание включает в себя информационные сущности и связи между ними без привязки к конкретной базе данных.

Данная модель создается на основе информации, записанной в спецификациях требований пользователей. Концептуальное проектирование базы данных абсолютно не зависит от таких подробностей ее реализации, как тип выбранной целевой СУБД, набор создаваемых прикладных программ, используемые языки программирования, тип выбранной вычислительной платформы, а также от любых других особенностей физической реализации. При разработке концептуальная модель данных постоянно подвергается тестированию и проверке на соответствие требованиям пользователей. Созданная концептуальная модель данных предприятия является источником информации для этапа логического проектирования базы данных.

Создадим 10 таблиц базы данных. Нам нужны таблицы актёров фильма, самих фильмов, персонажей фильмов, ролей актёров, жанров фильмов, страны производства фильма, студии, рейтингов, режиссёров фильмов, а также связующая таблица жанров и фильмов (многие ко многим). Остальные связи один ко многим.

Актёры

Играют

Роли

Персонажи

Фильмы

Жанры

Режиссёры

Студии

Страны производства

Рейтинги

Есть

Есть

Есть

Жанры и фильмы

Есть

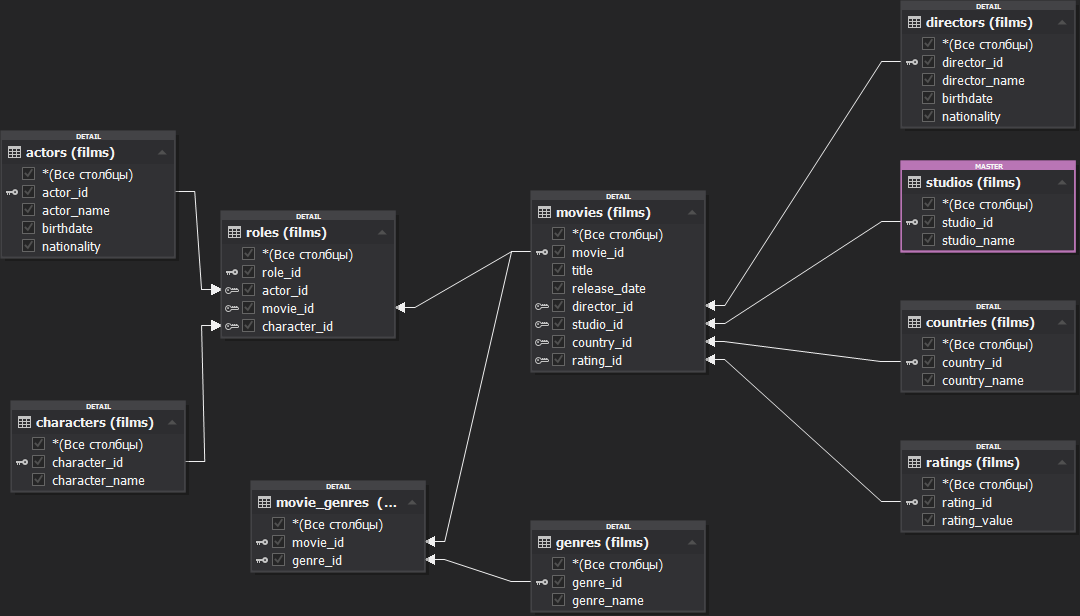
**Логическое проектирование** – создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных.

Преобразование концептуальной модели в логическую модель, как правило, осуществляется по формальным правилам. На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика конкретной СУБД.

Нормализация используется для улучшения модели данных, для того чтобы она удовлетворяла различным ограничениям, позволяющим исключить нежелательное дублирование данных.

Созданная логическая модель данных является источником информации для этапа физического проектирования и обеспечивает разработчика физической базы данных средствами поиска компромиссов, необходимых для достижения поставленных целей, что очень важно для эффективного проектирования. Логическая модель данных играет также важную роль на этапе эксплуатации и сопровождения уже готовой системы. При правильно организованном сопровождении поддерживаемая в актуальном состоянии модель данных позволяет точно и наглядно представить любые вносимые в базу данных изменения, а также оценить их влияние на прикладные программы и использование данных, уже имеющихся в базе.

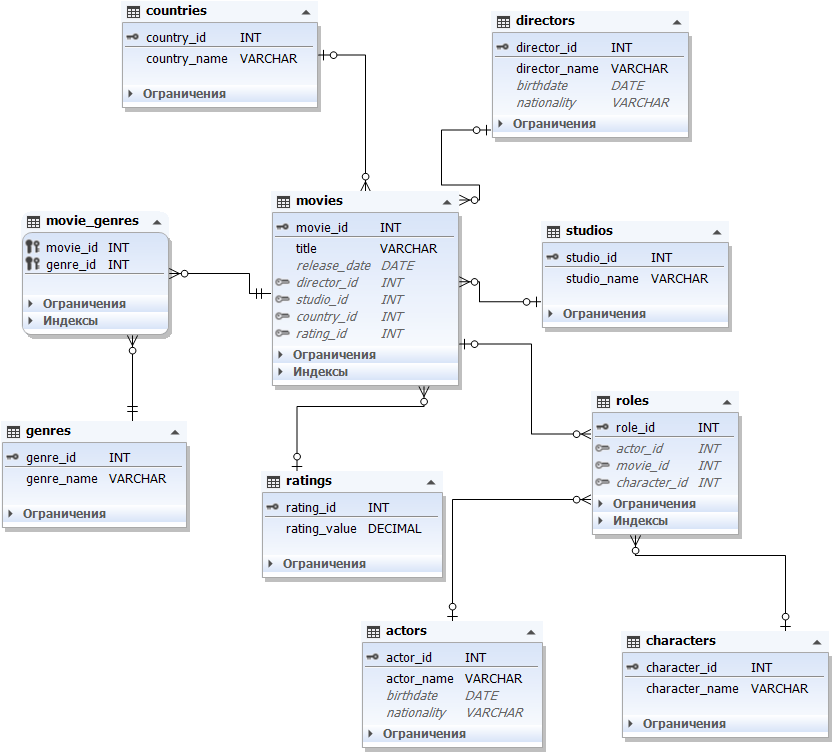
Логическая схема БД представлена ниже. Все таблицы в 3 нормальной форме.



**Физическое проектирование** базы данных – процесс подготовки описания реализации базы данных на вторичных запоминающих устройствах. На этом этапе рассматриваются основные отношения, организация файлов и индексов, предназначенных для обеспечения эффективного доступа к данным, а также все связанные с этим ограничения целостности и средства защиты.

В случае реляционной модели данных под этим подразумевается следующее:

* создание набора реляционных таблиц и ограничений для них на основе информации, представленной в глобальной логической модели данных;
* определение конкретных структур хранения данных и методов доступа к ним, обеспечивающих оптимальную производительность СУБД;
* разработка средств защиты создаваемой системы.



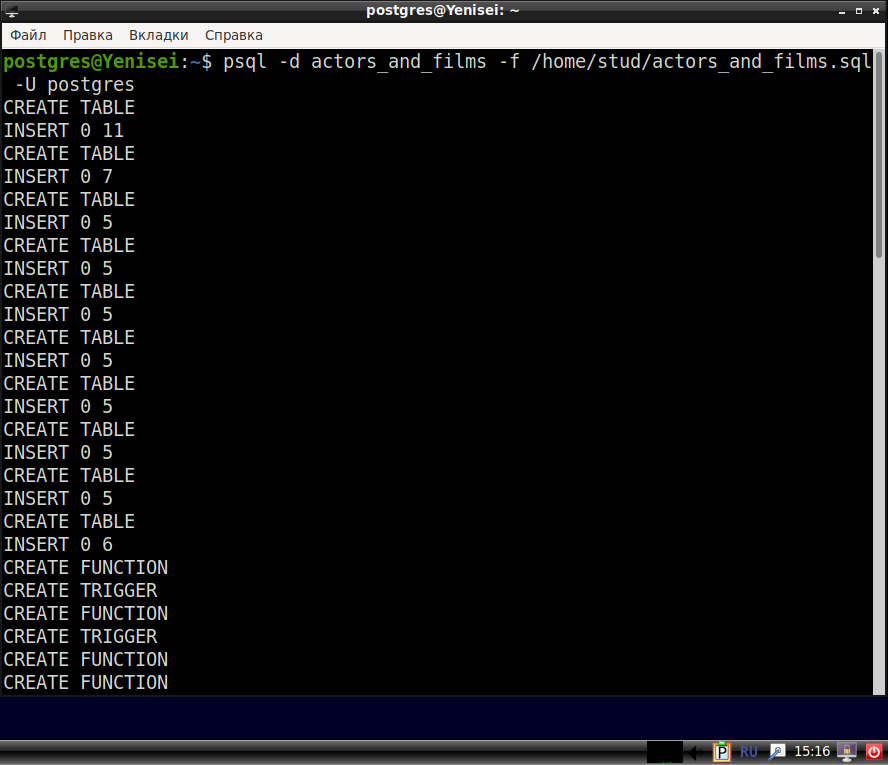
Создадим базу данных actors\_and\_films внутри psql командой:

CREATE DATABASE actors\_and\_films;

В созданном файле *actors\_and\_films.sql*, который будет приложен к данной работе уже созданы таблицы, триггеры и.т.д. Далее мы рассмотрим их работу.

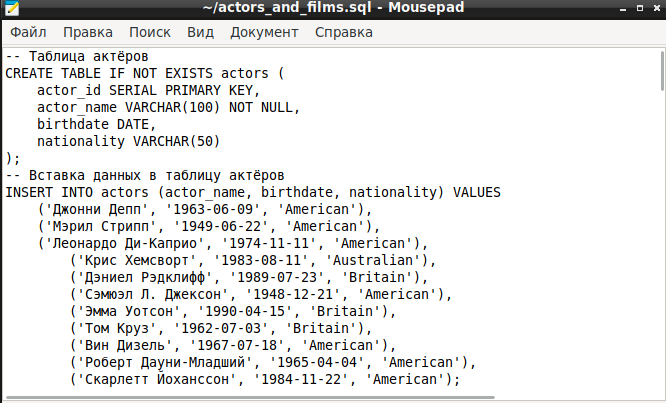
Для редактирования базы данных из файла воспользуемся командой:

psql -d actors\_and\_films -f /home/stud/actors\_and\_films.sql -U postgres



Можем увидеть, что все таблицы, триггеры и функции успешно созданы.

Приведу пример из файла для создания и заполнения значениями таблицы актёров. Остальные таблицы сделаны по такому же принципу. Посмотреть их можно в приложенном SQL – файле.



Рассмотрим, что мы создали. Отобразим содержимое всех таблиц, которые мы создали, используя запросы из файла. У нас создано 10 таблиц, посмотрим их содержимое, используя SELECT. Стоит отметить, что таблицы уже заполнены некоторыми значениями, используя INSERT.

Таблица актёров:

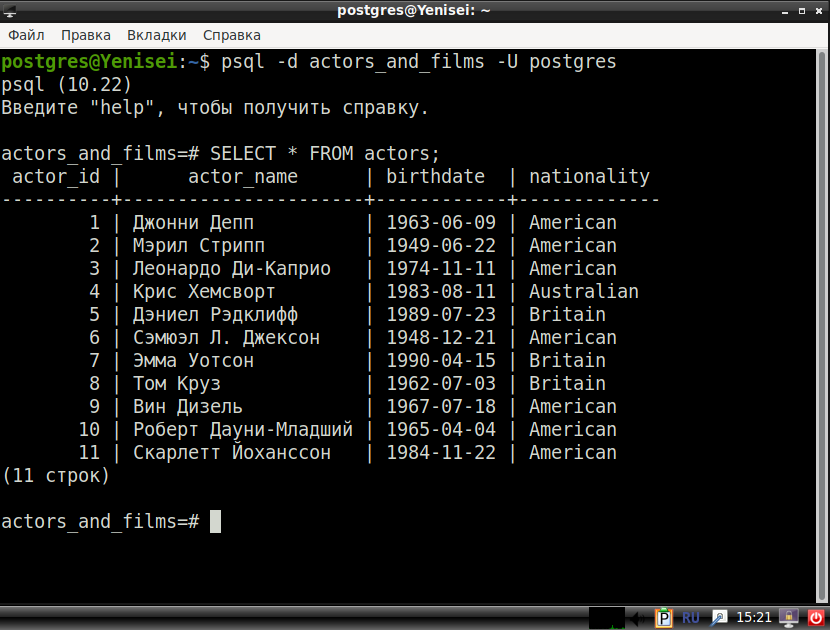


Таблица студий:

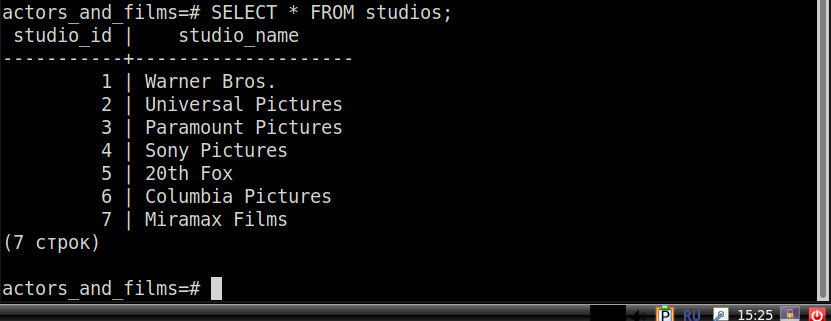


Таблица стран производства фильмов:

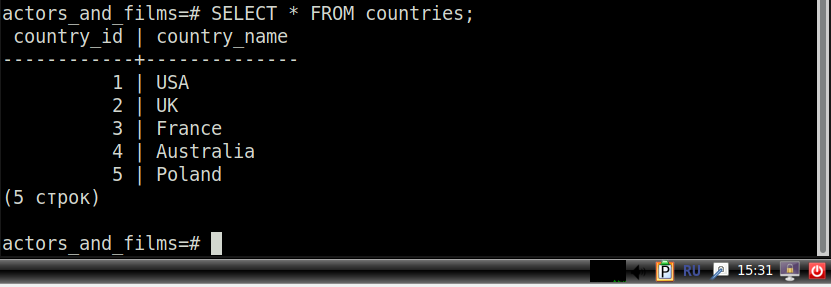


Таблица режиссёров фильмов:

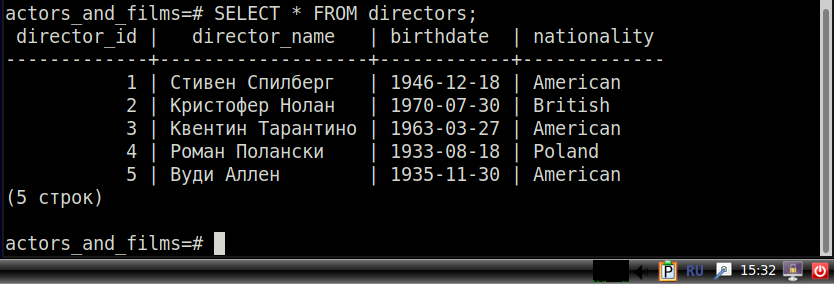


Таблица рейтингов фильмов:

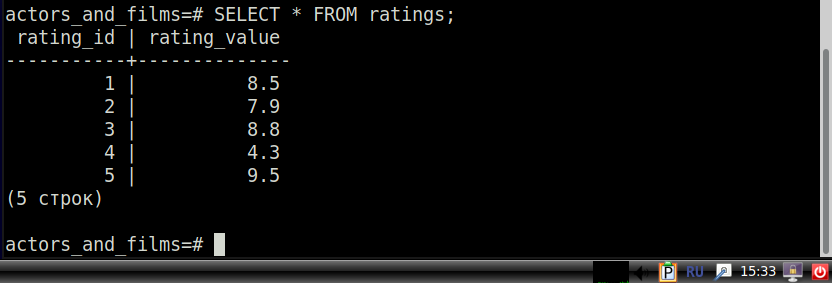


Таблица фильмов:

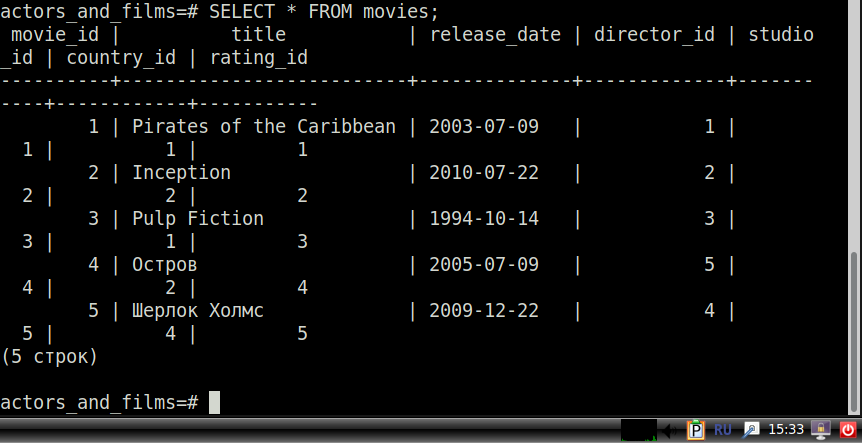


Таблица жанров:

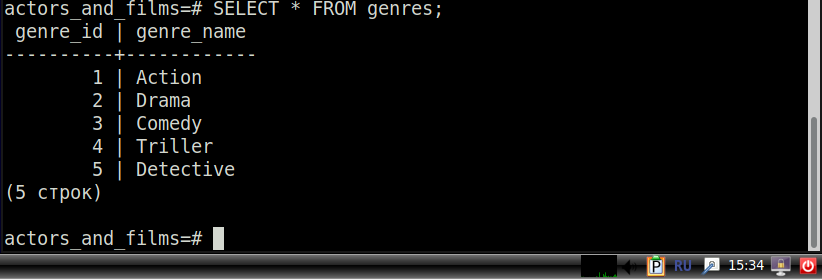


Таблица персонажей фильмов:

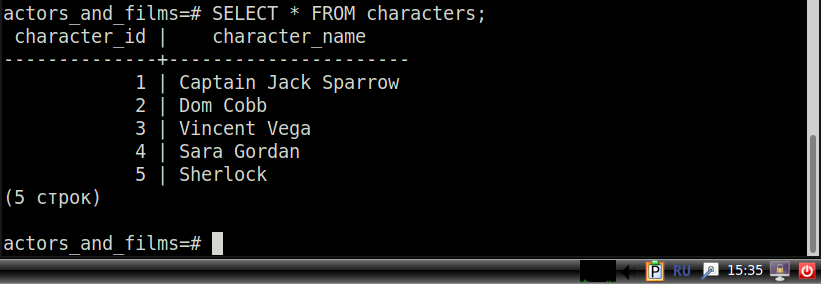


Таблица ролей актёров в фильмах:

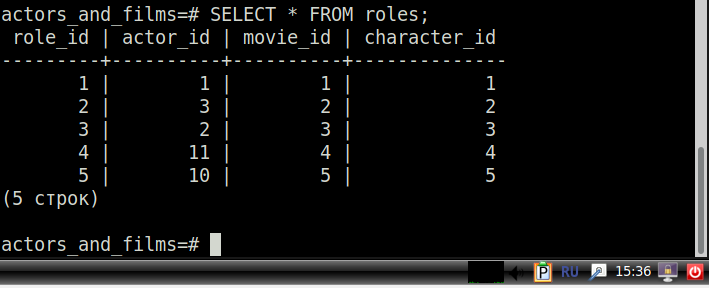
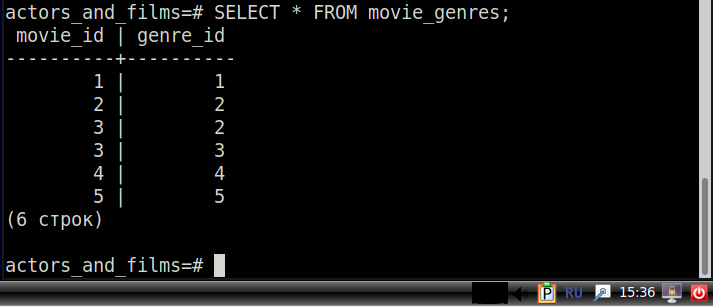
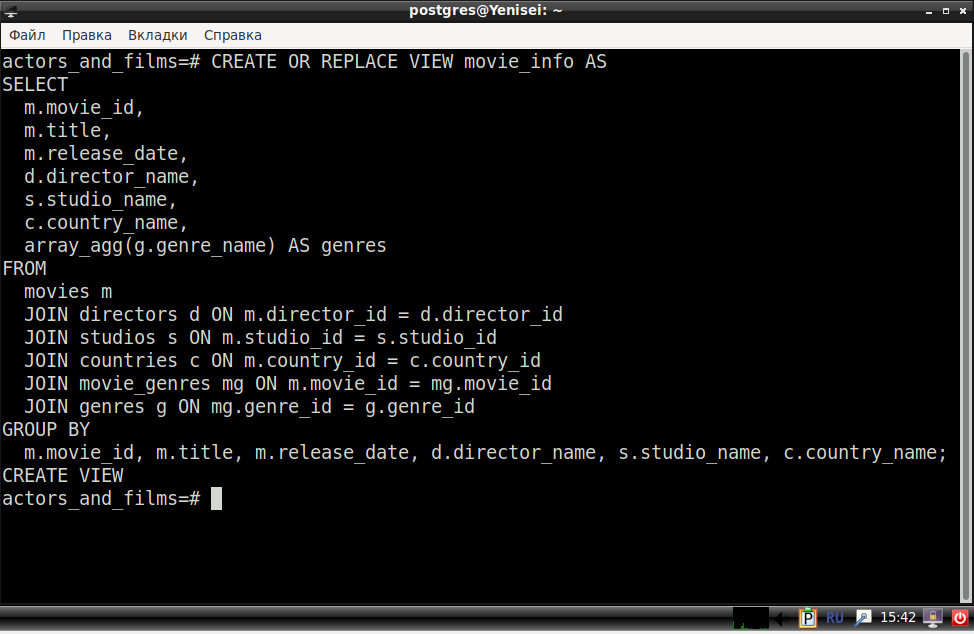


Таблица связи между фильмами и жанрами (многие ко многим).

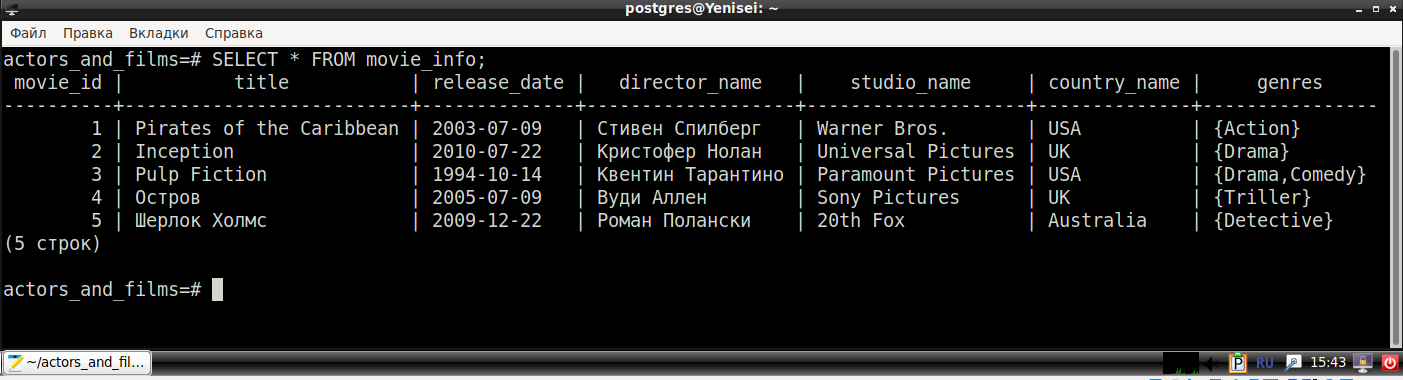


Мы видим, что все таблицы успешно созданы. Однако хочется увидеть более подробную информацию, чем просто id. Для этого существуют представления.

Посмотрев на таблицу с фильмами (movies), мы видим, что там есть id, но не понятно, кто же режиссёр фильма, какая студия делала этот фильм и.т.д. Давайте создадим представление (view) на основе таблицы movies, которое будет включать информацию о режиссёрах, студиях и других аспектах. Для примера давайте включим в представление информацию о фильмах, режиссёрах, студиях, странах и жанрах.

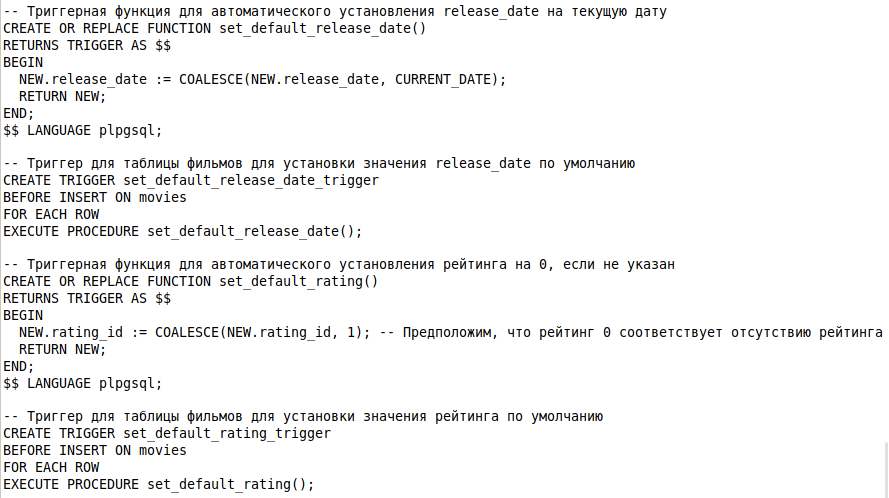


В этом запросе используется array\_agg(g.genre\_name) для объединения жанров в массив. Выведем результат представления и посмотрим, что получилось.

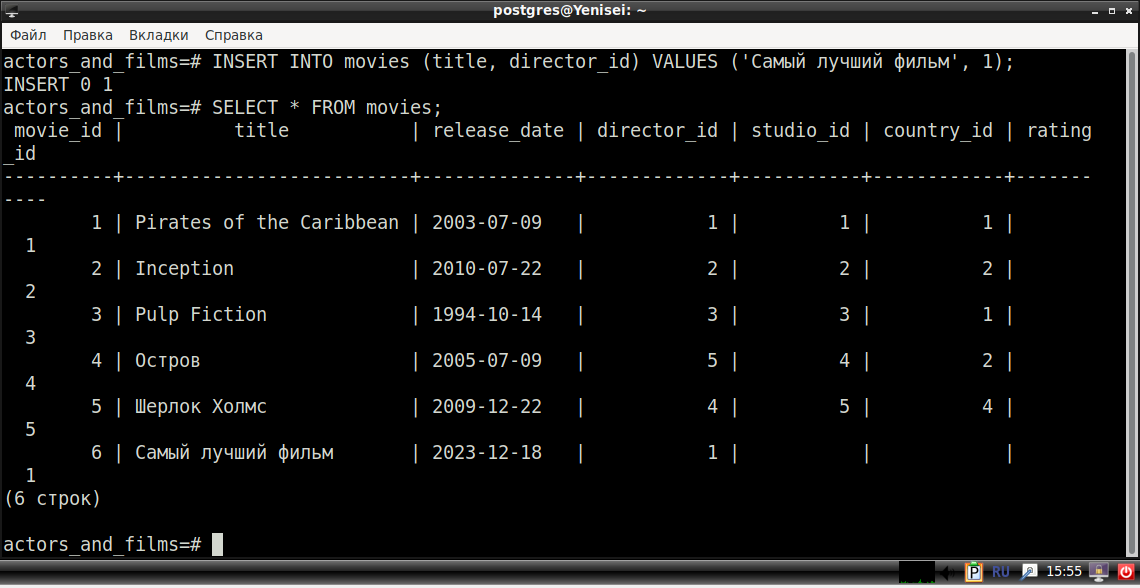


Теперь у нас есть более подробная информация о фильмах.

Мы также создали триггеры и триггерные функции в нашем sql-файле. Ниже представлены эти запросы.

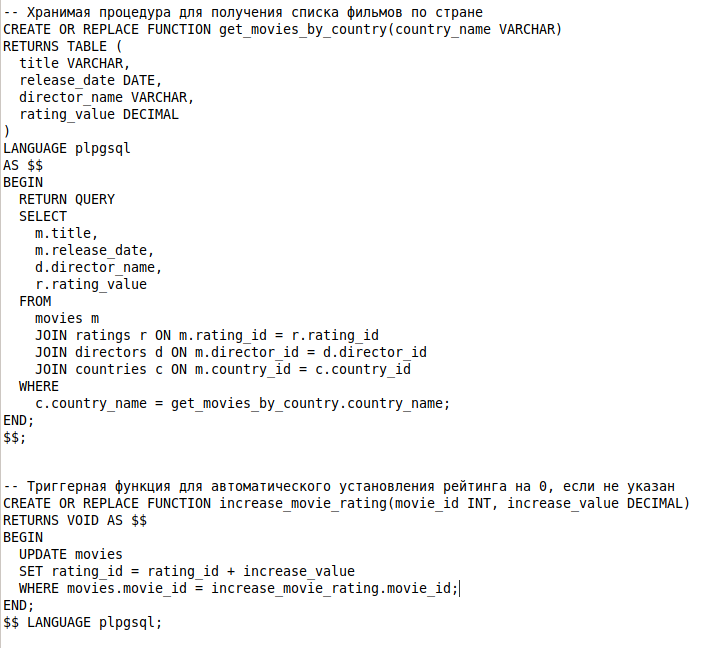


У нас есть триггерная функция, которая устанавливает дату для фильма на текущую, если даже мы её явно не задали. Мы можем проверить это. Для этого добавим новый фильм, введя только название и режиссёра. Я не указывал NOT NULL, поэтому ошибок при добавлении не будет. Дату вводить не будем, можем увидеть, что после выполнения запроса подставилась текущая дата для «самого лучшего фильма».

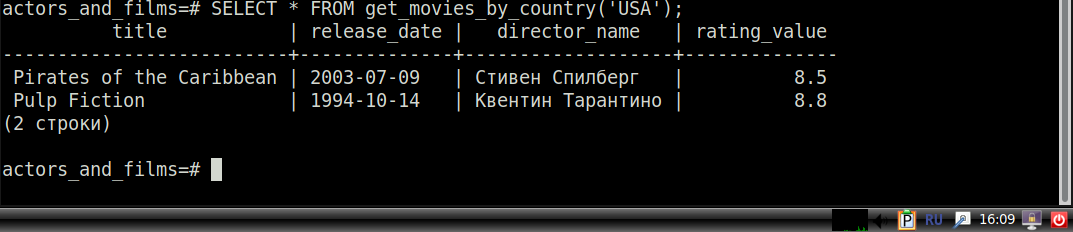


Также этим примером мы проверили работу второго триггер, который ставит рейтинг по умолчанию. Мы рейтинг не указывали при добавлении «самого лучшего фильма», а он добавился с id = 1. Значит триггер работает верно.

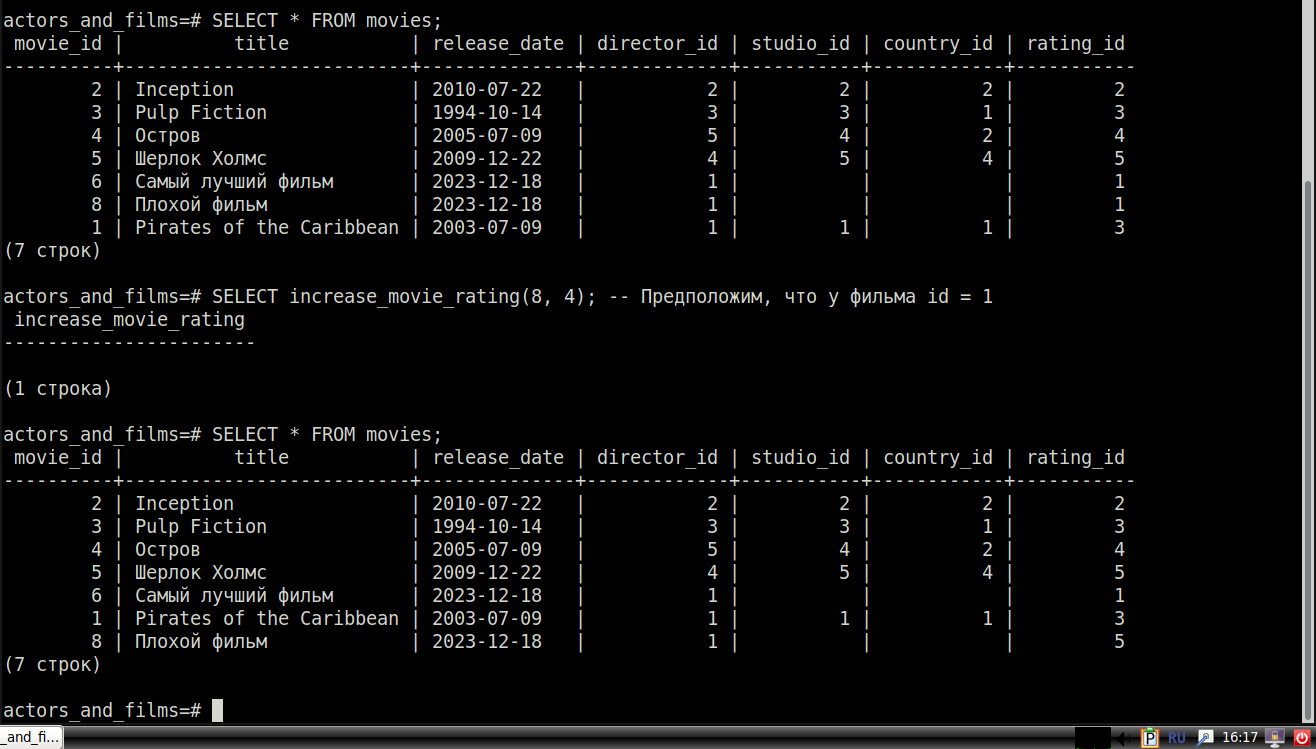
У нас созданы хранимые процедуры. Одна из них выводит список фильмов по стране производства, а другая увеличивает рейтинг фильма. Код представлен ниже.



Проверим работоспособность. Выведем все американские фильмы.

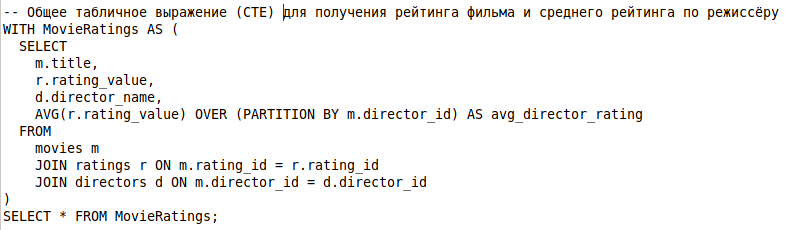


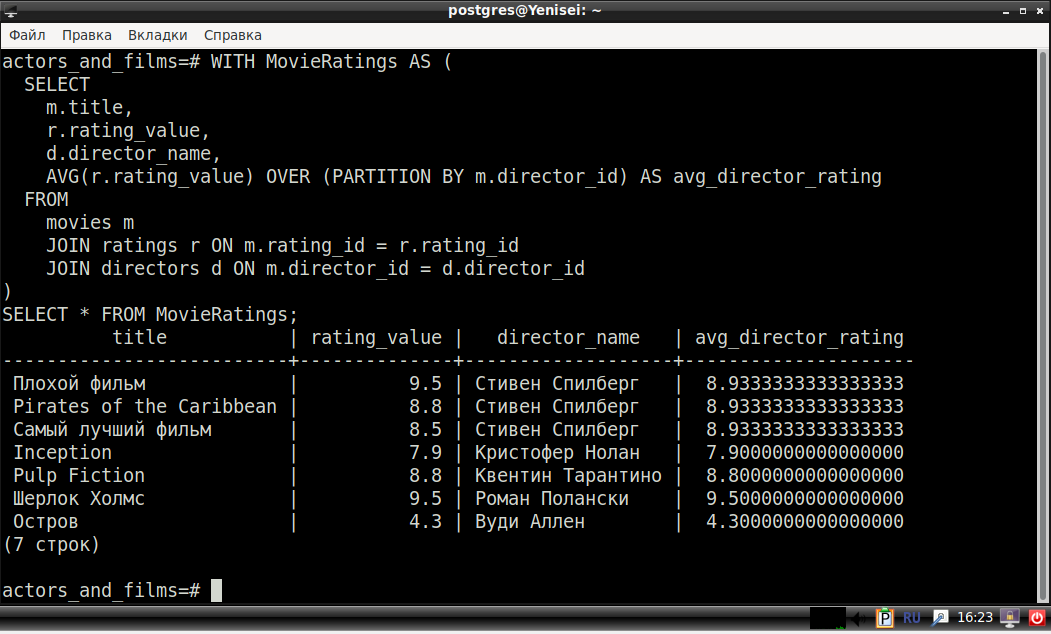
А теперь выполним процедуру для увеличения id рейтинга.



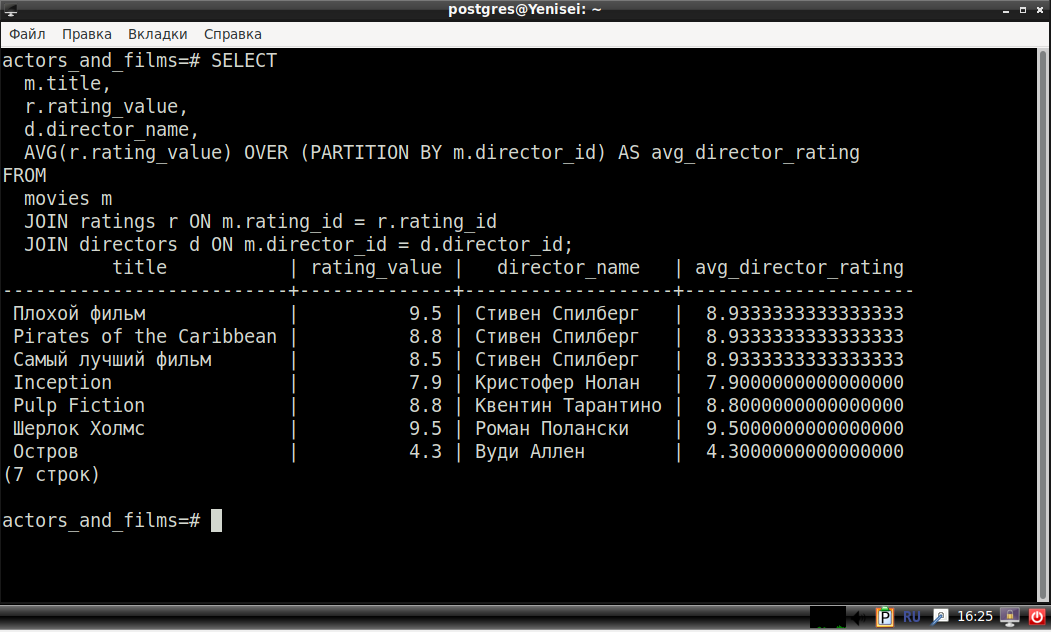
Можем заметить, что наш «самый плохой фильм» c rating\_id = 1 изменился на rating\_id = 5, потому что мы прибавили к нему 4.

Также у нас есть табличное выражение CTE для получения рейтинга фильма и среднего рейтинга по режиссёру.

Проверим его на работоспособность.



Можно сделать тоже самое, используя оконную функцию.



Также создадим оконную функцию накопленной суммы рейтингов по жанрам.

