Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

«Тверской государственный университет»

Математический факультет

Направление «Компьютерная безопасность»

Кафедра компьютерной безопасности и математических методов управления

Индивидуальный проект

по дисциплине “Основы построения защищенных баз данных”

Построение защищенной базы данных и применение модели разграничения доступа к базе данных

Кинопоиск

Работу выполнили:

Студентка V курса, группы М-55

Мамонова Мария Александровна

Студент V курса, группы М-54

Коханчик Антон Дмитриевич

Проверила:

Доцент Кафедры КБ и ММУ Цирулева Валентина Михайловна

Тверь, 2017 г.

Содержание

[E-R диаграмма 3](#_Toc501289070)

[Запросы к базе данных 4](#_Toc501289071)

[Представления 4](#_Toc501289072)

[Процедуры и функции 15](#_Toc501289073)

[Триггеры 22](#_Toc501289074)

[Модели доступа 31](#_Toc501289075)

[Ролевая модель доступа 31](#_Toc501289076)

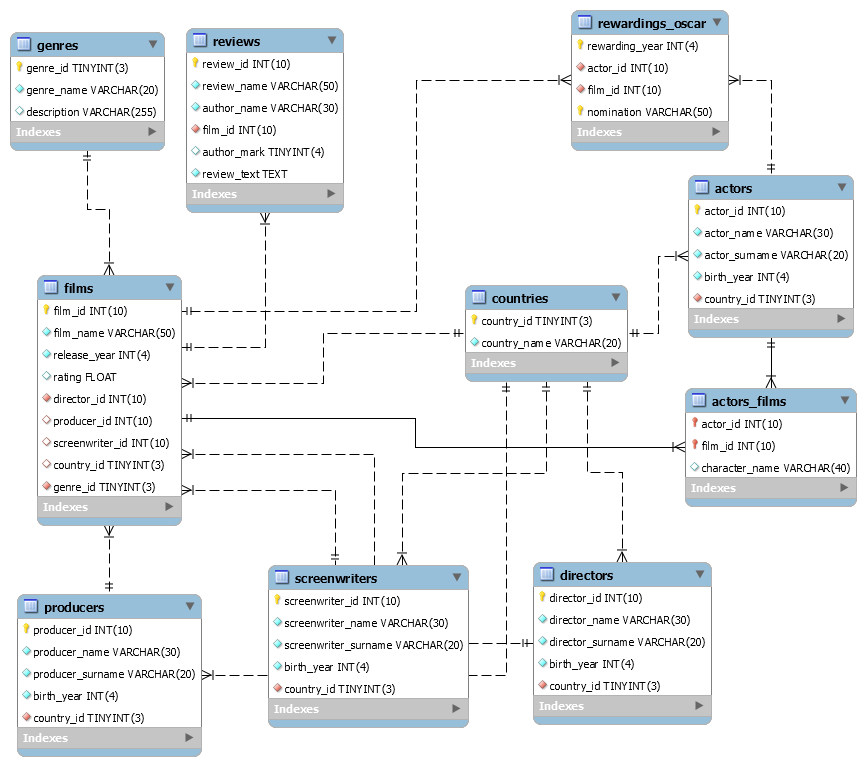
[1. Словесное описание и определение ролей и наборов их привилегий к объектам базы данных 31](#_Toc501289077)

[2. Схема ролей и их привилегий 32](#_Toc501289078)

[3. Иерархия ролей 32](#_Toc501289079)

[4. Имитация ролевой модели с помощью дискреционной модели разграничения доступа в MySQL 33](#_Toc501289080)

E-R диаграмма



Запросы к базе данных

# Представления

*1. Отобрать все фильмы, выпущенные позже 2007 года. Представление с условием проверки.*

create view newFilms as select \* from films where release\_year > 2007 order by release\_year with local check option;  
select \* from newFilms;

Результат:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| film\_id | film\_name | release\_year | rating | director\_id | producer\_id | screenwriter\_id | country\_id | genre\_id |
| 111543 | Темный рыцарь | 2008 | 8.502 | 41477 | 41477 | 41477 | 1 | 2 |
| 325439 | Чтец | 2008 | 7.935 | 10346 | 65747 | 36297 | 1 | 8 |
| 463647 | Железная леди | 2011 | 7.013 | 1190481 | 15755 | 1190793 | 3 | 22 |
| 470179 | Красная Шапочка | 2011 | 6.126 | 30189 | 37859 | 1047677 | 1 | 1 |
| 477764 | Дети шпионов 4D | 2011 | 3.695 | 30966 | 30966 | 30966 | 1 | 2 |
| 437410 | Темный рыцарь: Возрождение легенды | 2012 | 8.144 | 41477 | 41477 | 41477 | 1 | 2 |
| 461981 | Одинокий рейнджер | 2013 | 6.791 | 30870 | 10207 | 226493 | 1 | 13 |
| 522941 | Выживший | 2015 | 7.814 | 563585 | 6784 | 734897 | 1 | 4 |

*2. Отобрать все фильмы, фигурирующие в номинациях премии Оскар*

create view OscarFilms as select film\_name from films where film\_id in (select film\_id from rewardings\_Oscar);  
select \* from OscarFilm;

Результат:

|  |
| --- |
| film\_name |
| Крамер против Крамера |
| Блеск |
| Темный рыцарь |
| Чтец |
| Темный рыцарь: Возрождение легенды |
| Железная леди |
| Выживший |

*3. Отобрать все фильмы, снятые в Англии*

create view englishFilms as select film\_name from films where country\_id = (select country\_id from countries where country\_name = "Великобритания") with local check option;  
select \* from englishFilms;

Результат:

|  |
| --- |
| film\_name |
| Гарри Поттер и Тайная комната |
| Гарри Поттер и философский камень |
| Десятое королевство (сериал) |
| Железная леди |

*4. Отобрать количество фильмов, снятых на территории США в период 1980-2017 год*

create view americanFilms as select count(film\_id) from films where (release\_year between 1995 and 2000) and country\_id in (select country\_id from countries where country\_name = "США");  
select \* from americanFilms;

Результат:

|  |
| --- |
| count(film\_id) |
| 2 |

*5. Отобрать всех сценаристов, продюсеров и режиссеров "оскороносных" фильмов*

create view OscarPersons as select concat(director\_name, " ", director\_surname, ", режиссер") as разработчик from directors where director\_id in (select director\_id from films where film\_id in (select film\_id from rewardings\_Oscar))   
union select concat(producer\_name, " ", producer\_surname, ", продюсер") from producers where producer\_id in (select producer\_id from films where film\_id in (select film\_id from rewardings\_Oscar))   
union select concat(screenwriter\_name, " ", screenwriter\_surname, ", сценарист") from screenwriters where screenwriter\_id in (select screenwriter\_id from films where film\_id in (select film\_id from rewardings\_Oscar));  
select \* from OscarPersons;

Результат:

|  |
| --- |
| разработчик |
| Роберт Бентон, режиссер |
| Скотт Хикс, режиссер |
| Кристофер Нолан, режиссер |
| Стивен Долдри, режиссер |
| Филлида Ллойд, режиссер |
| Алехандро Иньярриту, режиссер |
| Стэнли Р. Джаффе, продюсер |
| Джейн Скотт, продюсер |
| Кристофер Нолан, продюсер |
| Донна Джильотти, продюсер |
| Дэмиэн Джонс, продюсер |
| Стив Голин, продюсер |
| Роберт Бентон, сценарист |
| Ян Сарди, сценарист |
| Кристофер Нолан, сценарист |
| Дэвид Хэа, сценарист |
| Эби Морган, сценарист |
| Марк Смит, сценарист |

*6. Вывести Топ-7 фильмов*

create view TopFilms as select film\_name, rating from films order by rating desc limit 7;  
select \* from TopFilms;

Результат:

|  |  |
| --- | --- |
| film\_name | rating |
| Властелин колец: Возвращение Короля | 8.611 |
| Темный рыцарь | 8.502 |
| Матрица | 8.492 |
| Десятое королевство (сериал) | 8.457 |
| Титаник | 8.371 |
| Пираты Карибского моря:Проклятие Черной жемчужины | 8.342 |
| Гарри Поттер и философский камень | 8.193 |

*7. Вывести авторов статей, которые всегда ставят оценки ниже рейтинга фильма*

create view badUsers as select distinct author\_name from reviews as t1 where not exists(select author\_mark from reviews as t3 where author\_mark > (select rating from films as t2 where t3.film\_id = t2.film\_id) and t1.author\_name = t3.author\_name);  
select \* from badUsers;

Результат:

|  |
| --- |
| author\_name |
| 1mira1 |
| Cas\_000 |

*8. Вывести всех актеров, которые являются ещё и режиссерами или продюсерами (возможно в других фильмах)*

create view universalPersons as select concat(actor\_name, " ", actor\_surname, ", режиссёр") as актёр from actors as t1 where if(exists(select \* from directors as t2 where concat(actor\_name, " ", actor\_surname) = concat(director\_name, " ", director\_surname)), true,false)   
union select concat(actor\_name, " ", actor\_surname, ", продюсер") as актёр from actors as t1 where if(exists(select \* from producers as t2 where concat(actor\_name, " ", actor\_surname) = concat(producer\_name, " ", producer\_surname)), true,false);  
select \* from universalPersons;

Результат:

|  |
| --- |
| актёр |
| Питер Джексон, режиссёр |
| Питер Джексон, продюсер |
| Леонардо ДиКаприо, продюсер |

*9. Вывести для каждого актера из списка его фильмов тот, у которого наибольший рейтинг*

create view bestFilm as select concat(actor\_name, " ", actor\_surname) as актёр, film\_name, max(rating), character\_name from actors inner join actors\_films as t1 on(actors.actor\_id = t1.actor\_id) inner join films on(t1.film\_id = films.film\_id) group by t1.actor\_id;   
select \* from bestFilm;

Результат:

|  |  |  |  |
| --- | --- | --- | --- |
| актёр | film\_name | max(rating) | character\_name |
| Хит Леджер | Темный рыцарь | 8.502 | Joker |
| Хьюго Уивинг | Матрица | 8.492 | Agent Smith |
| Дэниэл Лапэйн | Десятое королевство (сериал) | 8.457 | Prince Wendell |
| Эдриан Броуди | Кинг Конг | 7.573 | Jack Driscol |
| Карен Аллен | Индиана Джонс: В поисках утраченного ковчега | 8.007 | Marion Ravenwood |
| Винсент Риган | 300 спартанцев | 7.629 | Captain |
| Харрисон Форд | Индиана Джонс: В поисках утраченного ковчега | 8.007 | Indy |
| Кэрри-Энн Мосс | Матрица | 8.492 | Trinity |
| Джонни Депп | Пираты Карибского моря:Проклятие Черной жемчужины | 8.342 | Jack Sparrow |
| Гари Олдман | Красная Шапочка | 6.126 | Solomon |
| Аарон Экхарт | Темный рыцарь | 8.502 | Harvey Dent |
| Ноа Тейлор | Блеск | 7.87 | David Helfgott - Adolescent |
| Киану Ривз | Матрица | 8.492 | Neo |
| Армин Мюллер-Шталь | Блеск | 7.87 | Peter |
| Дэнни Трехо | Дети шпионов 4D | 3.695 | Uncle Machete |
| Лоренс Фишбёрн | Матрица | 8.492 | Morpheus |
| Вигго Мортенсен | Властелин колец: Возвращение Короля | 8.611 | Aragorn |
| Дональд Сазерленд | Кукловоды | 6.438 | Andrew Nivens |
| Джерард Батлер | 300 спартанцев | 7.629 | King Leonidas |
| Уильям Фихтнер | Одинокий рейнджер | 6.791 | Butch Cavendish |
| Роберт Инглунд | Кошмар на улице Вязов | 7.703 | Fred Krueger |
| Элайджа Вуд | Властелин колец: Возвращение Короля | 8.611 | Frodo |
| Кристиан Бэйл | Темный рыцарь | 8.502 | Bruce Wayne |
| Кимберли Уильямс-Пэйсли | Десятое королевство (сериал) | 8.457 | Virginia Lewis |
| Кейт Уинслет | Титаник | 8.371 | Rose DeWitt Bukater |
| Джессика Альба | Дети шпионов 4D | 3.695 | Marissa Wilson |
| Дастин Хоффман | Крамер против Крамера | 8.011 | Ted Kramer |
| Джейн Александр | Крамер против Крамера | 8.011 | Margaret Phelps |
| Рэйф Файнс | Чтец | 7.935 | Michael Berg |
| Мэрил Стрип | Крамер против Крамера | 8.011 | Joanna Kramer |
| Джек Блэк | Кинг Конг | 7.573 | Carl Denham |
| Кира Найтли | Пираты Карибского моря: Сундук мертвеца | 8.098 | Elizabeth Swann |
| Джеффри Раш | Пираты Карибского моря:Проклятие Черной жемчужины | 8.342 | Barbossa |
| Джули Уорнер | Кукловоды | 6.438 | Mary Sefton |
| Наоми Уоттс | Кинг Конг | 7.573 | Ann Darrow |
| Шон Эстин | Властелин колец: Возвращение Короля | 8.611 | Sam |
| Эрик Тал | Кукловоды | 6.438 | Sam Nivens |
| Орландо Блум | Властелин колец: Возвращение Короля | 8.611 | Legolas |
| Питер Джексон | Кинг Конг | 7.573 | Gunner |
| Леонардо ДиКаприо | Титаник | 8.371 | Jack Dawson |
| Энн Хэтэуэй | Темный рыцарь: Возрождение легенды | 8.144 | Selina |
| Джим Бродбент | Железная леди | 7.013 | Denis Thatcher |
| Том Харди | Темный рыцарь: Возрождение легенды | 8.144 | Bane |
| Дэниэл Рэдклифф | Гарри Поттер и Тайная комната | 8.193 | Harry Potter |
| Эмма Уотсон | Гарри Поттер и Тайная комната | 8.193 | Hermione Granger |
| Руперт Гринт | Гарри Поттер и Тайная комната | 8.193 | Ron Weasley |
| Билли Зейн | Титаник | 8.371 | Cal Hockley |
| Дайэнн Уист | Десятое королевство (сериал) | 8.457 | Evil Queen |
| Хэзер Лэнгенкэмп | Кошмар на улице Вязов | 7.703 | Nancy Thompson |
| Рональд Лейси | Индиана Джонс: В поисках утраченного ковчега | 8.007 | Major Arnold Toht |
| Сьюзэн Браун | Железная леди | 7.013 | June |
| Давид Кросс | Чтец | 7.935 | Young Michael Berg |
| Аманда Сайфред | Красная Шапочка | 6.126 | Valerie |
| Майкл Фассбендер | 300 спартанцев | 7.629 | Stelios |
| Донал Глисон | Выживший | 7.814 | Captain Andrew Henry |
| Шайло Фернандес | Красная Шапочка | 6.126 | Peter |
| Арми Хаммер | Одинокий рейнджер | 6.791 | John Reid (Lone Ranger) |
| Роуэн Бланчард | Дети шпионов 4D | 3.695 | Rebecca Wilson |

*10. Посчитать средний рейтинг фильмов США, снятых после 2000 года*

create view ratingAmericanFilms as select avg(rating) as средний\_рейтинг from films where release\_year > 2000 and exists(select \* from countries where country\_name = "США" and countries.country\_id = films.country\_id);  
select \* from ratingAmericanFilms;

Результат:

|  |
| --- |
| средний\_рейтинг |
| 7.426090869036588 |

*11. Вывести всех актеров в порядке убывания успешности (имеется в виду количество полученных наград)*

create view successfulness as select concat(actor\_name, " ", actor\_surname) as актёр, (select count(nomination) from rewardings\_oscar where actor\_id = t1.actor\_id) as наград from actors t1 order by наград desc;  
select \* from successfulness;

Результат:

|  |  |
| --- | --- |
| актёр | наград |
| Мэрил Стрип | 2 |
| Энн Хэтэуэй | 1 |
| Леонардо ДиКаприо | 1 |
| Кейт Уинслет | 1 |
| Хит Леджер | 1 |
| Джеффри Раш | 1 |
| Роберт Инглунд | 0 |
| Аманда Сайфред | 0 |
| Дональд Сазерленд | 0 |
| Рональд Лейси | 0 |
| Дэнни Трехо | 0 |
| Билли Зейн | 0 |
| Ноа Тейлор | 0 |
| Дэниэл Рэдклифф | 0 |
| Джонни Депп | 0 |
| Винсент Риган | 0 |
| Орландо Блум | 0 |
| Дэниэл Лапэйн | 0 |
| Наоми Уоттс | 0 |
| Кира Найтли | 0 |
| Рэйф Файнс | 0 |
| Джессика Альба | 0 |
| Роуэн Бланчард | 0 |
| Кристиан Бэйл | 0 |
| Донал Глисон | 0 |
| Уильям Фихтнер | 0 |
| Давид Кросс | 0 |
| Вигго Мортенсен | 0 |
| Хэзер Лэнгенкэмп | 0 |
| Армин Мюллер-Шталь | 0 |
| Руперт Гринт | 0 |
| Аарон Экхарт | 0 |
| Том Харди | 0 |
| Кэрри-Энн Мосс | 0 |
| Карен Аллен | 0 |
| Эрик Тал | 0 |
| Хьюго Уивинг | 0 |
| Джули Уорнер | 0 |
| Джек Блэк | 0 |
| Джейн Александр | 0 |
| Арми Хаммер | 0 |
| Элайджа Вуд | 0 |
| Майкл Фассбендер | 0 |
| Джерард Батлер | 0 |
| Сьюзэн Браун | 0 |
| Лоренс Фишбёрн | 0 |
| Дайэнн Уист | 0 |
| Киану Ривз | 0 |
| Эмма Уотсон | 0 |
| Гари Олдман | 0 |
| Джим Бродбент | 0 |
| Харрисон Форд | 0 |
| Питер Джексон | 0 |
| Эдриан Броуди | 0 |
| Шон Эстин | 0 |
| Дастин Хоффман | 0 |
| Кимберли Уильямс-Пэйсли | 0 |
| Шайло Фернандес | 0 |

*12. Вывести все награды, полученные лучшим актером*

create view bestActor as select nomination, rewarding\_year, (select film\_name from films as t2 where t1.film\_id = t2.film\_id) from rewardings\_oscar as t1 where actor\_id in (select actor\_id from actors where concat(actor\_name, " ", actor\_surname) = (select актёр from successfulness limit 1));  
select \* from bestActor;

Результат:

|  |  |  |
| --- | --- | --- |
| nomination | rewarding\_year | Name\_exp\_3 |
| Лучшая женская роль второго плана | 1980 | Крамер против Крамера |
| Лучшая женская роль | 2012 | Железная леди |

*13. Вывести всех актеров с количеством фильмов, в которых они снимались*

create view filmsActor as select concat(actor\_name, " ", actor\_surname) as актёр, count(film\_id) as количество\_фильмов from actors natural join actors\_films group by actor\_id;  
select \* from filmsActor;

Результат:

|  |  |
| --- | --- |
| актёр | количество\_фильмов |
| Хит Леджер | 1 |
| Хьюго Уивинг | 1 |
| Дэниэл Лапэйн | 1 |
| Эдриан Броуди | 1 |
| Карен Аллен | 1 |
| Винсент Риган | 1 |
| Харрисон Форд | 1 |
| Кэрри-Энн Мосс | 1 |
| Джонни Депп | 4 |
| Гари Олдман | 1 |
| Аарон Экхарт | 1 |
| Ноа Тейлор | 1 |
| Киану Ривз | 1 |
| Армин Мюллер-Шталь | 1 |
| Дэнни Трехо | 1 |
| Лоренс Фишбёрн | 1 |
| Вигго Мортенсен | 1 |
| Дональд Сазерленд | 1 |
| Джерард Батлер | 1 |
| Уильям Фихтнер | 1 |
| Роберт Инглунд | 1 |
| Элайджа Вуд | 1 |
| Кристиан Бэйл | 2 |
| Кимберли Уильямс-Пэйсли | 1 |
| Кейт Уинслет | 2 |
| Джессика Альба | 1 |
| Дастин Хоффман | 1 |
| Джейн Александр | 1 |
| Рэйф Файнс | 1 |
| Мэрил Стрип | 2 |
| Джек Блэк | 1 |
| Кира Найтли | 1 |
| Джеффри Раш | 2 |
| Джули Уорнер | 1 |
| Наоми Уоттс | 1 |
| Шон Эстин | 1 |
| Эрик Тал | 1 |
| Орландо Блум | 3 |
| Питер Джексон | 1 |
| Леонардо ДиКаприо | 2 |
| Энн Хэтэуэй | 1 |
| Джим Бродбент | 1 |
| Том Харди | 2 |
| Дэниэл Рэдклифф | 2 |
| Эмма Уотсон | 2 |
| Руперт Гринт | 2 |
| Билли Зейн | 1 |
| Дайэнн Уист | 1 |
| Хэзер Лэнгенкэмп | 1 |
| Рональд Лейси | 1 |
| Сьюзэн Браун | 1 |
| Давид Кросс | 1 |
| Аманда Сайфред | 1 |
| Майкл Фассбендер | 1 |
| Донал Глисон | 1 |
| Шайло Фернандес | 1 |
| Арми Хаммер | 1 |
| Роуэн Бланчард | 1 |

*14. Вывести режиссеров в порядке убывания количества наград, полученных актерами за съемки в их фильмах*

create view badDirectors as select concat(director\_name, " ", director\_surname) as режиссер, (select count(nomination) from rewardings\_oscar as t1 where exists(select \* from films as t2 where t1.film\_id = t2.film\_id and t3.director\_id = t2.director\_id)) as количество\_наград from directors as t3 order by количество\_наград desc;  
select \* from badDirectors;

Результат:

|  |  |
| --- | --- |
| режиссер | количество\_наград |
| Кристофер Нолан | 2 |
| Алехандро Иньярриту | 1 |
| Роберт Бентон | 1 |
| Стивен Долдри | 1 |
| Скотт Хикс | 1 |
| Филлида Ллойд | 1 |
| Дэвид Карсон | 0 |
| Кэтрин Хардвик | 0 |
| Крис Коламбус | 0 |
| Зак Снайдер | 0 |
| Роберт Родригес | 0 |
| Лана Вачовски | 0 |
| Уэс Крэйвен | 0 |
| Стюарт Орм | 0 |
| Питер Джексон | 0 |
| Гор Вербински | 0 |
| Джеймс Кэмерон | 0 |
| Стивен Спилберг | 0 |

# Процедуры и функции

*1. Отобрать всех актеров, играющих в заданном фильме N*

create procedure actorFromFilm (film varchar(50))   
select actor\_name, actor\_surname from actors natural join actors\_films where film\_id = (select film\_id from films where film\_name = film);  
call actorFromFilm("Индиана Джонс: В поисках утраченного ковчега");

Результат:

|  |  |
| --- | --- |
| actor\_name | actor\_surname |
| Карен | Аллен |
| Харрисон | Форд |
| Рональд | Лейси |

*2. Отобрать все фильмы заданного актера*

create procedure actorFilms (actor varchar(55))   
select film\_name from actors natural join actors\_films natural join films where (actor\_name = substring\_index(actor, " ", 1) or actor\_name = substring\_index(actor, " ", -1)) and   
(actor\_surname = substring\_index(actor, " ", -1) or actor\_surname = substring\_index(actor, " ", 1));  
call actorFilms("Джонни Депп");  
call actorFilms("Депп Джонни");

Результат:

|  |
| --- |
| film\_name |
| Пираты Карибского моря:Проклятие Черной жемчужины |
| Кошмар на улице Вязов |
| Пираты Карибского моря: Сундук мертвеца |
| Одинокий рейнджер |

*3. Отобрать количество фильмов, выпущенные с Т1 по Т2 годы*

delimiter //  
create function numberFilmsInTimeInterval (t1 int(4), t2 int(4))  
returns int  
begin

declare num int;  
select count(\*) into num from films where release\_year between t1 and t2;  
 return num;

end//  
delimiter ;  
select numberFilmsInTimeInterval(1990, 2000);

Результат:

|  |
| --- |
| numberFilmsInTimeInterval(1990, 2000) |
| 5 |

*4. Отобрать все фильмы, фигурирующие в номинациях премии Оскар в заданном году*

create procedure OscarReward (y int(4))  
select film\_name from films where film\_id in (select film\_id from rewardings\_Oscar where rewarding\_year = y);  
call OscarReward(2009);

Результат:

|  |
| --- |
| film\_name |
| Темный рыцарь |
| Чтец |

*5. Отобрать все рецензии о заданном фильме*

create procedure ReviewsAboutFilm (film varchar(50))  
select \* from reviews where film\_id = (select film\_id from films where film\_name = film);  
call ReviewsAboutFilm("Властелин колец: Возвращение Короля");

Результат:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| review\_id | review\_name | author\_name | film\_id | author\_mark | review\_text |
| 1 | Грандиозная битва за Средиземье | Jesse Pinkman | 3498 | 10 | «Возвращение короля» — эпичное завершение трилогии Властелин колец. Питер Джексон снял нечто прекрасное, красивое, и захватывающее дух кино. Актерский состав поразил, настолько актеры вжились в роль, что веришь всему, что происходит на экране. Сразу хочу отметить игру Энди Серкиса, просто отлично передал образ Голлума, его внутренний мир, поведение, его жалкое существование под влиянием кольца. Элайджа Вуд в этой части открыл персонажа по полной, здесь уже и переживания видны, и хорошая игра, Фродо стал более сильным, храбрым и смелым хоббитом. Вигго Мортенсен выложился на все сто процентов, мне в его игре понравилось все, он уже необычный странник из севера,а уже правитель который поведет свое войско на битву со злом. Все остальные актеры просто великолепны. Битва на Пеленнорских полях показана очень эпично, захватывающе, особенно когда армия Рохана вступила в бой с харадримцами. Красивейшие пейзажи, горы, просторы средиземья все выглядит идеально. «Возвращение Короля» — шедевр киноиндустрии, получивший множество Оскаров. Питер Джексон просто гениален! Браво! Ждем с нетерпением «Хоббита». |
| 2 |  | chopin | 3498 | 9 | В этих строках хочу выразить свое мнение. Фильм на мой взгляд получился лучше книги. Кто-нибудь когда читал книгу не задавал себе вопрос, почему описание идет так занудисто? Честно признаюсь, несколько страниц я перелистнул, бегло просмотрев текст. Фильм же начинается, и сразу же начинают происходить события. Гендальф уехал, Гендальф приехал. Мы не знаем, что прошло много лет. А в книге? Все немного скучнее. Смотря фильм не спрашиваешь себя сколько проходит времени. Все три часа (а в общем больше девяти) находишься в напряжении. Конечно есть ляпы. Когда Фродо убегает от призраков на плоту, непонятно почему лошадь, причем очень здоровая лошадь, не смогла одолеть расстояние которое маленький хоббит перепрыгнул шутя? Из братства кольца мои симпатии на стороне Леголаса и Арагорна. |

*6. Отобрать все фильмы заданного жанра от заданного режиссера*

create procedure statisticOfGenreForDirector (genre varchar(20), director varchar(55))  
select film\_name from films where genre\_id = (select genre\_id from genres where genre\_name = genre) and director\_id = (select director\_id from directors where (director\_name = substring\_index(director, " ", 1) and director\_surname = substring\_index(director, " ", -1)) or (director\_name = substring\_index(director, " ", -1) and director\_surname = substring\_index(director, " ", 1)));  
call statisticOfGenreForDirector("Фентези", "Гор Вербински");

Результат:

|  |
| --- |
| film\_name |
| Пираты Карибского моря:Проклятие Черной жемчужины |
| Пираты Карибского моря: Сундук мертвеца |

*7. Отобрать всех актеров, получивших Оскар в заданном году*

create procedure Oscar (y int(4))   
select actor\_name, actor\_surname from actors where actor\_id in (select actor\_id from rewardings\_Oscar where rewarding\_year = y);  
call Oscar(2009);

Результат:

|  |  |
| --- | --- |
| actor\_name | actor\_surname |
| Хит | Леджер |
| Кейт | Уинслет |

*8. Вывести страну, на территории которой снято больше всего фильмов*

delimiter //  
create function mostPopularCountry()  
returns varchar(20)  
begin

declare country varchar(20);  
select country\_name into country from countries as t1 group by country\_id limit 1;  
return country;

end //  
delimiter ;  
select mostPopularCountry();

Результат:

|  |
| --- |
| mostPopularCountry() |
| США |

*9. Вывести количество наград, полученные заданным актером*

delimiter //  
create function rewardingsOfActor (actor varchar(55))  
returns int  
begin

declare num int;  
select count(\*) as количество into num from rewardings\_Oscar as t1 natural join films where exists(select \* from actors\_films where actors\_films.film\_id = t1.film\_id and exists(select \* from actors where concat(actor\_name, " ", actor\_surname) = actor and actors.actor\_id = actors\_films.actor\_id));  
return num;

end//  
delimiter ;  
select rewardingsOfActor("Мэрил Стрип");

Результат:

|  |
| --- |
| rewardingsOfActor("Мэрил Стрип") |
| 2 |

*10. Вывести самый популярный жанр*

delimiter //  
create function mostPopularGenre()  
returns varchar(20)  
begin

declare genre varchar(20);

select genre\_name as жанр into genre from genres natural join films group by genre\_id order by count(genre\_id) desc limit 1;  
return genre;

end //  
delimiter ;  
select mostPopularGenre();

Результат:

|  |
| --- |
| mostPopularGenre() |
| Фентези |

*11. Выбрать все фильмы, в которых играли N1 и N2*

create procedure duet(actor1 varchar(55), actor2 varchar(55))  
select film\_name from films as t1 where exists(select \* from actors\_films as t2 where exists(select \* from actors as t3 where concat(actor\_name, " ", actor\_surname) = actor1 and t3.actor\_id = t2.actor\_id) and t2.film\_id = t1.film\_id and film\_id in (select film\_id from actors\_films as t4 where exists(select \* from actors as t5 where concat(actor\_name, " ", actor\_surname) = actor2 and t5.actor\_id = t4.actor\_id) group by film\_id));  
call duet("Джонни Депп", "Орландо Блум");

Результат:

|  |
| --- |
| film\_name |
| Пираты Карибского моря:Проклятие Черной жемчужины |
| Пираты Карибского моря: Сундук мертвеца |

*12. Вывести для заданного актера из списка его фильмов тот, у которого наибольший рейтинг*

create procedure bestFilmOfActor (actor varchar(55), out film varchar(50))  
select film\_name into film from actors inner join actors\_films as t1 on(actors.actor\_id = t1.actor\_id) inner join films on(t1.film\_id = films.film\_id) where concat(actor\_name, " ", actor\_surname) = actor group by t1.actor\_id;  
call bestFilmOfActor("Хит Леджер", @film);  
select @film;

Результат:

|  |
| --- |
| @film |
| Темный рыцарь |

*13. Вывести количество "провальных" фильмов заданного актера (фильмы, рейтинг которых ниже среднего)*

create procedure countBadFilmsOfActor (actor varchar(55), out count\_films int)  
select count(film\_name) into count\_films from films where film\_id in (select film\_id from actors\_films where actor\_id = (select actor\_id from actors where actor\_name = substring\_index(actor, " ", 1) and actor\_surname = substring\_index(actor, " ", -1))) group by rating having rating < (select avg(rating) from films);  
call countBadFilmsOfActor("Джонни Депп", @count\_films);  
select @count\_films;

Результат:

|  |
| --- |
| @count\_films |
| 1 |

*14. Вывести количество самых востребованных актеров (количество фильмов, в которых они снимались больше среднего)*

create procedure countOfMostPopularActors (out num int)  
select count(\*) into num from (select \* from actors natural join actors\_films as t1 group by actor\_id having count(film\_id) > (select avg(n) from (select count(film\_id) as n from actors\_films group by actor\_id) as t)) as t2;  
call countOfMostPopularActors(@num);  
select @num;

Результат:

|  |
| --- |
| @num |
| 11 |

# Триггеры

*1. Создать триггер, который при вставке актера в таблицу Актёры проверяет наличие указанной страны, и если та не указана, ставит страну 9 (Неизвестно)*

delimiter //  
create trigger insert\_check before insert on actors  
for each row

begin  
if new.country\_id is null then  
set new.country\_id = (select country\_id from countries where country\_name = 'Неизвестно');  
end if;  
end //

delimiter ;  
insert into actors value (1, "Джонни", "Пробный", 2017, null);

Результат:  


*2. Создать триггер для вставки в таблицу Страны, который бы переводил первую букву названия страны в верхний регистр*

delimiter //  
create trigger upper before insert on countries  
for each row

begin  
set new.country\_name = CONCAT(upper(left(new.country\_name,1)), substr(new.country\_name,2));  
end //

delimiter ;  
insert into countries value (14, "исландия");

Результат:  


*3. Создать триггер, который при удалении страны из таблицы Страны заменяет все вхождения этой страны в других таблицах на страну 9 (Неизвестно)*

delimiter //  
create trigger befor\_delete\_country before delete on countries  
for each row

begin  
update actors set country\_id = 9 where country\_id = old.country\_id;  
update films set country\_id = 9 where country\_id = old.country\_id;  
update directors set country\_id = 9 where country\_id = old.country\_id;  
update producers set country\_id = 9 where country\_id = old.country\_id;  
update screenwriters set country\_id = 9 where country\_id = old.country\_id;  
 end //

delimiter ;  
delete from countries where country\_name = "Новая Зеландия";

Результат:

До  


После  

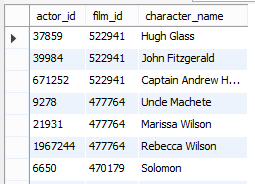

*4. Создать триггер для каскадного удаления фильма*

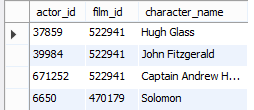
delimiter //  
create trigger before\_delete\_film before delete on films  
for each row

begin  
delete from reviews where film\_id = old.film\_id;  
delete from actors\_films where film\_id = old.film\_id;  
end //

delimiter ;  
delete from films where film\_name = "Дети шпионов 4D";

Результат:

До  


После  


*5. Создать триггер для вставки в таблицу Награждения, который проверяет, входит ли номинация в допустимое множество*

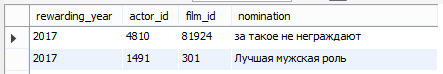
create table nominations (nomination varchar(50))CHARACTER SET = UTF8;

insert into nominations values  
("Лучшая женская роль"),  
("Лучшая женская роль второго плана"),  
("Лучшая мужская роль"),  
("Лучшая мужская роль второго плана");

delimiter //  
create trigger before\_insert\_reward before insert on rewardings\_oscar  
for each row

begin  
if new.nomination not in (select nomination from nominations) then  
set new.nomination = "за такое не неграждают";  
end if;  
end //

delimiter ;  
insert into rewardings\_oscar value  
(2017, 1491, 301, "Лучшая мужская роль"),  
(2017, 4810, 81924, "Лучшая кошачья мята");

Результат:  


*6. Создать триггер, который заполняет таблицу Аудит при изменении таблицы Фильмы*

create table AuditFilms (  
 date\_operation datetime,  
 name\_operation varchar(20),  
 attribute varchar (20),  
 old\_data text,  
 new\_data text  
)CHARACTER SET = UTF8;

delimiter //  
create trigger after\_update\_films after update on films  
for each row

begin  
if new.film\_id != old.film\_id then  
 insert into auditfilms value (now(), "update", "film\_id", old.film\_id, new.film\_id);  
end if;  
if new.film\_name != old.film\_name then  
 insert into auditfilms value (now(), "update", "film\_name", old.film\_name, new.film\_name);  
end if;  
if new.release\_year != old.release\_year then  
 insert into auditfilms value (now(), "update", "release\_year", old.release\_year, new.release\_year);  
end if;  
if new.rating != old.rating then  
 insert into auditfilms value (now(), "update", "rating", old.rating, new.rating);  
end if;  
if new.director\_id != old.director\_id then  
 insert into auditfilms value (now(), "update", "director\_id", old.director\_id, new.director\_id);  
end if;  
if new.producer\_id != old.producer\_id then  
 insert into auditfilms value (now(), "update", "producer\_id", old.producer\_id, new.producer\_id);  
end if;  
if new.screenwriter\_id != old.screenwriter\_id then  
 insert into auditfilms value (now(), "update", "screenwriter\_id", old.screenwriter\_id, new.screenwriter\_id);  
end if;  
if new.country\_id != old.country\_id then  
 insert into auditfilms value (now(), "update", "country\_id", old.country\_id, new.country\_id);  
end if;  
if new.genre\_id != old.genre\_id then  
 insert into auditfilms value (now(), "update", "genre\_id", old.genre\_id, new.genre\_id);  
end if;  
end//

delimiter ;  
update films set film\_name = "Буль-буль. ААААА! ПОМОГИТЕ!" where film\_id = 2213;

Результат:  
До  


После  


*7. Создать триггер, который проверяет рецензии на наличие нецензурных слов (для примера используется слово "редиска") и заменяет их (его) на "нехороший человек"*

delimiter //  
create trigger before\_insert\_reviews before insert on reviews  
for each row

begin  
if new.review\_text like "%редиска%" then  
set new.review\_text = replace(new.review\_text, "редиска", "нехороший человек");  
end if;  
end//

delimiter ;  
insert into reviews value (9, "la-la", "anonimus", 2213, 9, "Ок, только режисер - редиска, убил Джека. Он точно редиска");

Результат:  


*8. Создать триггер, который проверяет, есть ли у вставляемой рецензии название. Если нет, дает название "без названия"*

delimiter //  
create trigger before\_insert\_reviews before insert on reviews  
for each row

begin  
if new.review\_name = " " or new.review\_name is null then  
set new.review\_name = "без названия";  
end if;  
end//

delimiter ;  
insert into reviews (review\_name, author\_name, film\_id, author\_mark, review\_text) value (null, "anonimus", 2213, 9, " Пробный текст.... ");

Результат:  


*9. Создать триггер, который переводит фамилии и имена всех новых актеров в верхний регистр*

delimiter //  
create trigger before\_insert\_actors before insert on actors  
for each row

begin  
set new.actor\_name = CONCAT(upper(left(new.actor\_name,1)), substr(new.actor\_name,2));  
set new.actor\_surname = CONCAT(upper(left(new.actor\_surname,1)), substr(new.actor\_surname,2));  
end//

delimiter ;  
insert into actors value (2, "иван", "ван", 2017, 9);

Результат:  


*10. Создать таблицу Люди, содержащую поле "деятельность". Теперь данные о всех актерах, режиссерах, продюсерах, сценаристах вставляются в эту таблицу. Создать триггер, который будет при вставке в таблицу Люди переносить данные о новом человеке в нужную таблицу.*

create table People(  
 p\_id int(10),  
 p\_name varchar(30),  
 p\_surname varchar(20),  
 p\_year int(4),  
 p\_country tinyint(3),  
 p\_special varchar(15) )CHARACTER SET = UTF8;

delimiter //  
create trigger after\_insert\_people after insert on people  
for each row

begin  
if new.p\_special = "актер" then  
insert into actors value (new.p\_id, new.p\_name, new.p\_surname, new.p\_year, new.p\_country);  
end if;  
if new.p\_special = "режиссер" then  
insert into directors value (new.p\_id, new.p\_name, new.p\_surname, new.p\_year, new.p\_country);  
end if;  
if new.p\_special = "сценарист" then  
insert into screenwriters value (new.p\_id, new.p\_name, new.p\_surname, new.p\_year, new.p\_country);  
end if;  
if new.p\_special = "продюсер" then  
insert into producers value (new.p\_id, new.p\_name, new.p\_surname, new.p\_year, new.p\_country);  
end if;  
end //

delimiter ;  
insert into people values  
(3, "иван", "реж", 2017, 9, "режиссер"),  
(3, "Петр", "Прод", 2017, 9, "продюсер");

Результат:  
  


*11. Создать триггер, который при удалении из таблицы Актеры заносит данные об удаляемом в таблицу Архив*

create table archive(  
 date\_operation datetime,  
 name\_operation varchar(20),  
 d\_id int(10),  
 d\_name varchar(30),  
 d\_surname varchar(20),  
 d\_year int(4),  
 d\_country tinyint(3))CHARACTER SET = UTF8;

delimiter //  
create trigger before\_delete\_actors before delete on actors  
for each row

begin  
insert into archive value(now(), "delete", old.actor\_id, old.actor\_name, old.actor\_surname, old.birth\_year, old.country\_id);  
end//

delimiter ;  
delete from actors where actor\_id = 1;

Результат:  


*12-13. Создать триггер для логирования операция по таблице Актеры-Фильмы*

create table actosr\_films\_log(  
 date\_operation datetime,  
 name\_operation varchar(20),  
 a\_id int(10),  
 f\_id int(10),  
 a\_character varchar(40))CHARACTER SET = UTF8;

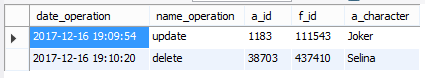
delimiter //  
create trigger after\_update\_actors\_films after update on actors\_films  
for each row

begin  
insert into actosr\_films\_log value(now(), "update", old.actor\_id, old.film\_id, old.character\_name);  
end//  
delimiter ;

delimiter //  
create trigger before\_delete\_actors\_films before delete on actors\_films  
for each row

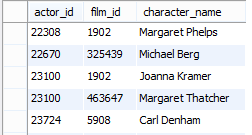
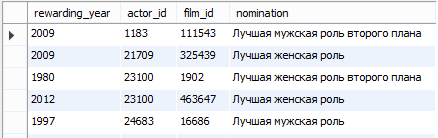
begin  
insert into actosr\_films\_log value(now(), "delete", old.actor\_id, old.film\_id, old.character\_name);  
end//  
delimiter ;

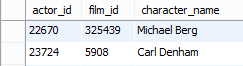
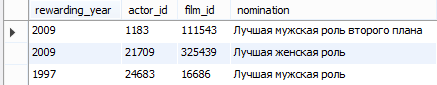
update actors\_films set character\_name = "Джокер" where actor\_id = 1183 and film\_id = 111543;  
delete from actors\_films where actor\_id = 38703 and film\_id = 437410;

Результат:  


*14. Создать триггер для каскадного удаления актера*

delimiter //  
create trigger before\_delete\_actors before delete on actors  
for each row  
begin  
delete from actors\_films where actor\_id = old.actor\_id;  
delete from rewardings\_oscar where actor\_id = old.actor\_id;  
end //  
delimiter ;  
delete from actors where actor\_id = 23100;

Результат:  
До  
  


После  
  


Модели доступа

# Ролевая модель доступа

## 1. Словесное описание и определение ролей и наборов их привилегий к объектам базы данных

Введем следующие роли: Администратор (admin), Модератор (moderator), Автор (author), Редактор статей (editor), Пользователь (customer).

Определим набор их привилегий по отношению к базе данных Кинопоиск.

Администратор имеет все возможные права: может создавать новые записи по всем таблицам, просматривать все таблицы, изменять их, удалять записи из них и сами таблицы, менять привилегии других ролей.

Автор может: создавать новые записи по таблице reviews, просматривать эту таблицу.

Редактор статей может: просматривать, изменять, добавлять и удалять записи из таблицы reviews.

Модератор может: создавать новые записи по таблицам actors, actors\_films, films, rewarding\_Oscar, directors, screenwriters, producers, изменять эти таблицы, просматривать их.

Пользователь может только просматривать все таблицы.

## 2. Схема ролей и их привилегий

## 3. Иерархия ролей

Иерархия ролей данной модели следующая:

## 4. Имитация ролевой модели с помощью дискреционной модели разграничения доступа в MySQL

*Создание «ролей»:*

create user admin identified by 'admin';  
create user moderator identified by 'moderator';  
create user editor identified by 'editor';  
create user author identified by 'author';  
create user customer identified by 'customer';

*Просмотр таблицы пользователей. Проверяем, что все «рол»и создались.*

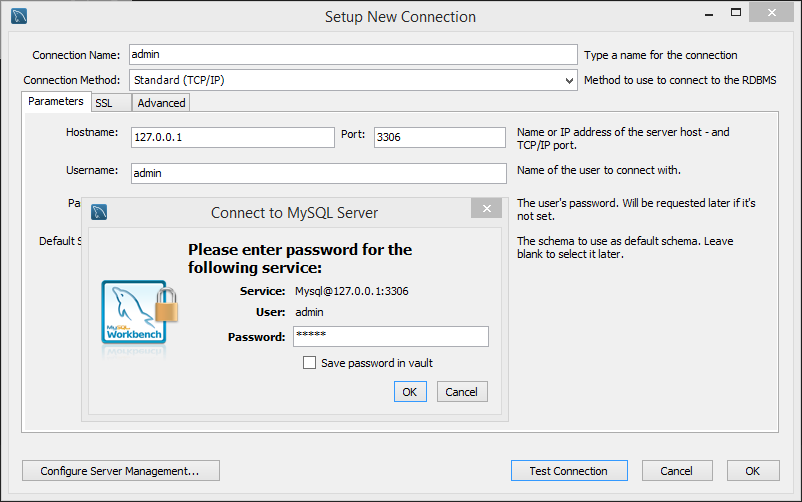
select user from mysql.user;

|  |
| --- |
| user |
| admin |
| author |
| customer |
| editor |
| moderator |
| mysql.sys |
| root |

*Наделение «роли» admin заявленными привилегиями*

grant all on films.\* to admin;  
grant grant option on films.\* to admin;

*Авторизация под «ролью» admin.*



*От имени admin даем заявленные привилегии «роли» moderator*

use films;

grant select, update, insert on films.actors to moderator;  
grant select, update, insert on films.actors\_films to moderator;  
grant select, update, insert on films.films to moderator;  
grant select, update, insert on films.rewardings\_Oscar to moderator;  
grant select, update, insert on films.directors to moderator;  
grant select, update, insert on films.screenwriters to moderator;  
grant select, update, insert on films.producers to moderator;

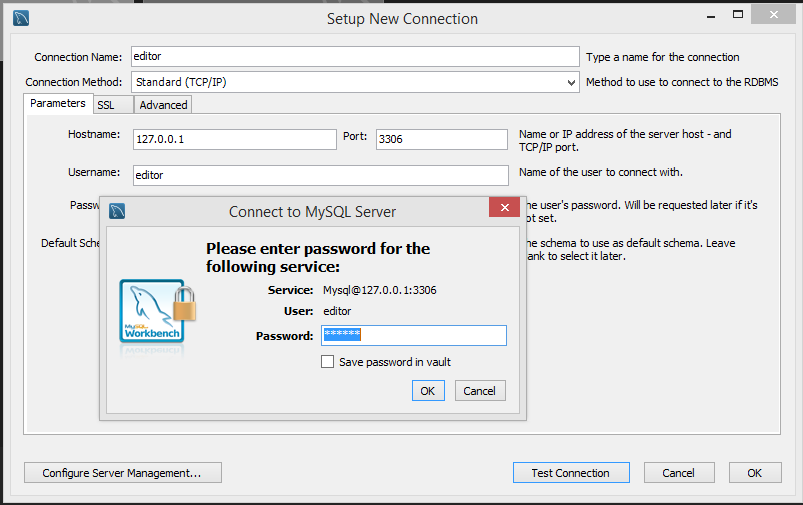
*От имени admin даем заявленные привилегии «роли» editor*

grant select, update, insert, grant option on films.reviews to editor;

*От имени admin даем заявленные привилегии «роли» customer*

grant select on films.\* to customer;

*Авторизация под «ролью» editor*



*От имени editor даем заявленные привилегии «роли» author*

use films;

grant select, insert on films.reviews to author;

*Просмотр всех «ролей» с их привилегиями*

select Db, User, Grantor, Table\_name, Table\_priv from mysql.tables\_priv where Db != 'sys';

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Db | User | Grantor | Table\_name | Table\_priv |
| films | moderator | admin@localhost | directors | Select,Insert,Update |
| films | moderator | admin@localhost | actors | Select,Insert,Update |
| films | moderator | admin@localhost | rewardings\_oscar | Select,Insert,Update |
| films | moderator | admin@localhost | films | Select,Insert,Update |
| films | moderator | admin@localhost | actors\_films | Select,Insert,Update |
| films | moderator | admin@localhost | screenwriters | Select,Insert,Update |
| films | moderator | admin@localhost | producers | Select,Insert,Update |
| films | editor | admin@localhost | reviews | Select,Insert,Update,Grant |
| films | author | editor@localhost | reviews | Select,Insert |

# Мандатная модель разграничения доступа

## 1. Словесное описание

Рассмотрим применение данной модели на примере таблицы films.

Введем следующие роли с уровнями секретности:

|  |  |
| --- | --- |
| Роль | Уровень секретности (доступа) |
| Администратор (admin) | 2 |
| Пользователь (customer) | 1 |

Таким образом, Администратор имеет доступ к информации уровня 2 и 1, а Пользователь – к информации уровня 1.

Допустим (в рамках примера), что Пользователь не должен получать доступ к фильмам, снятым не в США. Тогда таблица films после выполнения кода

alter table films add access tinyint(2);  
update films set access = 1 where country\_id = 1;  
update films set access = 2 where country\_id != 1;

будет выглядеть следующим образом:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| film\_id | film\_name | release\_year | rating | director\_id | producer\_id | screenwriter\_id | country\_id | genre\_id | access |
| 301 | Матрица | 1999 | 8.492 | 23330 | 1790 | 23329 | 1 | 2 | 1 |
| 339 | Индиана Джонс: В поисках утраченного ковчега | 1981 | 8.007 | 22260 | 20397 | 26537 | 1 | 10 | 1 |
| 688 | Гарри Поттер и Тайная комната | 2002 | 8.03 | 24817 | 23449 | 10093 | 3 | 5 | 2 |
| 689 | Гарри Поттер и философский камень | 2001 | 8.193 | 24817 | 23449 | 10093 | 3 | 5 | 2 |
| 1902 | Крамер против Крамера | 1979 | 8.011 | 31799 | 14197 | 31799 | 1 | 8 | 1 |
| 2213 | Титаник | 1997 | 8.371 | 27977 | 27977 | 27977 | 1 | 8 | 1 |
| 3498 | Властелин колец: Возвращение Короля | 2003 | 8.611 | 32383 | 32383 | 32385 | 1 | 5 | 1 |
| 4374 | Пираты Карибского моря:Проклятие Черной жемчужины | 2003 | 8.342 | 30870 | 10207 | 30871 | 1 | 5 | 1 |
| 5198 | Кошмар на улице Вязов | 1984 | 7.703 | 8442 | 3023 | 8442 | 1 | 1 | 1 |
| 5908 | Кинг Конг | 2005 | 7.573 | 32383 | 539928 | 32385 | 2 | 8 | 2 |
| 6226 | Кукловоды | 1994 | 6.438 | 104876 | 8288 | 30871 | 1 | 1 | 1 |
| 16686 | Блеск | 1996 | 7.87 | 29361 | 90143 | 90096 | 4 | 8 | 2 |
| 63991 | Пираты Карибского моря: Сундук мертвеца | 2006 | 8.098 | 30870 | 10207 | 30871 | 1 | 5 | 1 |
| 77217 | Десятое королевство (сериал) | 1999 | 8.457 | 64418 | 78017 | 55143 | 3 | 5 | 2 |
| 81924 | 300 спартанцев | 2007 | 7.629 | 462603 | 8523 | 462603 | 1 | 5 | 1 |
| 111543 | Темный рыцарь | 2008 | 8.502 | 41477 | 41477 | 41477 | 1 | 2 | 1 |
| 325439 | Чтец | 2008 | 7.935 | 10346 | 65747 | 36297 | 1 | 8 | 1 |
| 437410 | Темный рыцарь: Возрождение легенды | 2012 | 8.144 | 41477 | 41477 | 41477 | 1 | 2 | 1 |
| 461981 | Одинокий рейнджер | 2013 | 6.791 | 30870 | 10207 | 226493 | 1 | 13 | 1 |
| 463647 | Железная леди | 2011 | 7.013 | 1190481 | 15755 | 1190793 | 3 | 22 | 2 |
| 470179 | Красная Шапочка | 2011 | 6.126 | 30189 | 37859 | 1047677 | 1 | 1 | 1 |
| 477764 | Дети шпионов 4D | 2011 | 3.695 | 30966 | 30966 | 30966 | 1 | 2 | 1 |
| 522941 | Выживший | 2015 | 7.814 | 563585 | 6784 | 734897 | 1 | 4 | 1 |

Следовательно, дальнейший доступ к данным таблицы может осуществляться следующим образом:

create user admin, customer;  
grant select on films to customer;  
grant all on films to admin;

create view for\_admin as  
 select \* from films where access <= 2 with local check option;  
create view for\_user as  
 select \* from films where access <= 1 with local check option;

delimiter //  
create procedure see\_films (user\_name varchar(50))  
begin  
 if user\_name = 'admin' then  
 select \* from for\_admin;  
 end if;  
 if user\_name = 'customer' then  
 select \* from for\_user;  
 end if;  
end //  
delimiter ;

call see\_films('customer');

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| film\_id | film\_name | release\_year | rating | director\_id | producer\_id | screenwriter\_id | country\_id | genre\_id | access |
| 301 | Матрица | 1999 | 8.492 | 23330 | 1790 | 23329 | 1 | 2 | 1 |
| 339 | Индиана Джонс: В поисках утраченного ковчега | 1981 | 8.007 | 22260 | 20397 | 26537 | 1 | 10 | 1 |
| 1902 | Крамер против Крамера | 1979 | 8.011 | 31799 | 14197 | 31799 | 1 | 8 | 1 |
| 2213 | Титаник | 1997 | 8.371 | 27977 | 27977 | 27977 | 1 | 8 | 1 |
| 3498 | Властелин колец: Возвращение Короля | 2003 | 8.611 | 32383 | 32383 | 32385 | 1 | 5 | 1 |
| 4374 | Пираты Карибского моря:Проклятие Черной жемчужины | 2003 | 8.342 | 30870 | 10207 | 30871 | 1 | 5 | 1 |
| 5198 | Кошмар на улице Вязов | 1984 | 7.703 | 8442 | 3023 | 8442 | 1 | 1 | 1 |
| 6226 | Кукловоды | 1994 | 6.438 | 104876 | 8288 | 30871 | 1 | 1 | 1 |
| 63991 | Пираты Карибского моря: Сундук мертвеца | 2006 | 8.098 | 30870 | 10207 | 30871 | 1 | 5 | 1 |
| 81924 | 300 спартанцев | 2007 | 7.629 | 462603 | 8523 | 462603 | 1 | 5 | 1 |
| 111543 | Темный рыцарь | 2008 | 8.502 | 41477 | 41477 | 41477 | 1 | 2 | 1 |
| 325439 | Чтец | 2008 | 7.935 | 10346 | 65747 | 36297 | 1 | 8 | 1 |
| 437410 | Темный рыцарь: Возрождение легенды | 2012 | 8.144 | 41477 | 41477 | 41477 | 1 | 2 | 1 |
| 461981 | Одинокий рейнджер | 2013 | 6.791 | 30870 | 10207 | 226493 | 1 | 13 | 1 |
| 470179 | Красная Шапочка | 2011 | 6.126 | 30189 | 37859 | 1047677 | 1 | 1 | 1 |
| 477764 | Дети шпионов 4D | 2011 | 3.695 | 30966 | 30966 | 30966 | 1 | 2 | 1 |
| 522941 | Выживший | 2015 | 7.814 | 563585 | 6784 | 734897 | 1 | 4 | 1 |

call see\_films('admin');

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| film\_id | film\_name | release\_year | rating | director\_id | producer\_id | screenwriter\_id | country\_id | genre\_id | access |
| 301 | Матрица | 1999 | 8.492 | 23330 | 1790 | 23329 | 1 | 2 | 1 |
| 339 | Индиана Джонс: В поисках утраченного ковчега | 1981 | 8.007 | 22260 | 20397 | 26537 | 1 | 10 | 1 |
| 688 | Гарри Поттер и Тайная комната | 2002 | 8.03 | 24817 | 23449 | 10093 | 3 | 5 | 2 |
| 689 | Гарри Поттер и философский камень | 2001 | 8.193 | 24817 | 23449 | 10093 | 3 | 5 | 2 |
| 1902 | Крамер против Крамера | 1979 | 8.011 | 31799 | 14197 | 31799 | 1 | 8 | 1 |
| 2213 | Титаник | 1997 | 8.371 | 27977 | 27977 | 27977 | 1 | 8 | 1 |
| 3498 | Властелин колец: Возвращение Короля | 2003 | 8.611 | 32383 | 32383 | 32385 | 1 | 5 | 1 |
| 4374 | Пираты Карибского моря:Проклятие Черной жемчужины | 2003 | 8.342 | 30870 | 10207 | 30871 | 1 | 5 | 1 |
| 5198 | Кошмар на улице Вязов | 1984 | 7.703 | 8442 | 3023 | 8442 | 1 | 1 | 1 |
| 5908 | Кинг Конг | 2005 | 7.573 | 32383 | 539928 | 32385 | 2 | 8 | 2 |
| 6226 | Кукловоды | 1994 | 6.438 | 104876 | 8288 | 30871 | 1 | 1 | 1 |
| 16686 | Блеск | 1996 | 7.87 | 29361 | 90143 | 90096 | 4 | 8 | 2 |
| 63991 | Пираты Карибского моря: Сундук мертвеца | 2006 | 8.098 | 30870 | 10207 | 30871 | 1 | 5 | 1 |
| 77217 | Десятое королевство (сериал) | 1999 | 8.457 | 64418 | 78017 | 55143 | 3 | 5 | 2 |
| 81924 | 300 спартанцев | 2007 | 7.629 | 462603 | 8523 | 462603 | 1 | 5 | 1 |
| 111543 | Темный рыцарь | 2008 | 8.502 | 41477 | 41477 | 41477 | 1 | 2 | 1 |
| 325439 | Чтец | 2008 | 7.935 | 10346 | 65747 | 36297 | 1 | 8 | 1 |
| 437410 | Темный рыцарь: Возрождение легенды | 2012 | 8.144 | 41477 | 41477 | 41477 | 1 | 2 | 1 |
| 461981 | Одинокий рейнджер | 2013 | 6.791 | 30870 | 10207 | 226493 | 1 | 13 | 1 |
| 463647 | Железная леди | 2011 | 7.013 | 1190481 | 15755 | 1190793 | 3 | 22 | 2 |
| 470179 | Красная Шапочка | 2011 | 6.126 | 30189 | 37859 | 1047677 | 1 | 1 | 1 |
| 477764 | Дети шпионов 4D | 2011 | 3.695 | 30966 | 30966 | 30966 | 1 | 2 | 1 |
| 522941 | Выживший | 2015 | 7.814 | 563585 | 6784 | 734897 | 1 | 4 | 1 |