САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ – ПРОЦЕССОВ УПРАВЛЕНИЯ

Сиркин Тимур Владимирович

Курсовая работа

Базы данных и сетевые технологии

Предметная область: Фильмы

Направление 16.Б01-пу

Прикладная математика и информатика

Преподаватель: Филиппов Р.О.

Санкт-Петербург
2017

Содержание

Ссылка на репозиторий GitHub: https://github.com/TimurSirkin/films_DataBase

Описание базы данных:

Главная таблица **Films**

Предназначена для хранения информации о художественных фильмах.

Структура таблицы:

1. ID - Уникальный идентификатор (тип данных: serial)
2. Name - Название фильма (тип данных: varchar)
3. Premiere_Date - Дата премьеры (тип данных: date)
4. Producer_ID - ID Режиссера фильма (тип данных: int)
5. Rental - Сборы (тип данных: real)
6. Country_ID - ID Страны, в который фильм выпущен (тип данных: int)
7. Duration - Длительность (тип данных: time without time zone)
8. Budget - Бюджет (тип данных: real)
9. Age_Limit - Возрастное ограничение (тип данных: int)
10. Director_ID - ID Продюсера (тип данных: int)
11. Studio_ID - ID Киностудии, которая выпустила фильм (тип данных: int)

Связана с таблицами:

- Films_Rating через поле Film_ID таблицы Films_Rating (1:m) - Связь фильма и рейтинга
- Genries через таблицу Films_Genries(m:m) - Жанры фильма
- Persons через таблицу Films_Actors (m:m) - Актеры, играющие в фильме
- Persons через поле Producer_ID (m:1) - Продюсер фильма
- Persons через поле Director_ID (m:1) - Режиссер фильма
- Countries через поле Country_ID (m:1) - Страна, в которой выпущен фильм
- Studios через поле Studio_ID (m:1) - Киностудия, выпустившая фильм

Главная таблица **Persons**

Предназначена для хранения информации об актерах продюсерах и режиссерах.

Структура таблицы:

1. ID - Уникальный идентификатор (тип данных: serial)
2. Name - Имя (тип данных: varchar)
3. Surname - Фамилия (тип данных: varchar)
4. Sex - Пол (тип данных: boolean)
5. Birthdate - Дата рождения (тип данных: date)
6. Country_ID - ID Страна, гражданином которой он является (тип данных: int)

Связана с таблицами:

- Films (см. описание таблицы Films)
- Countries через поле Country_ID (m:1) - Страна, гражданином которой он является

Справочная таблица **Studios**

Предназначена для хранения информации о киностудиях.

Структура таблицы:

1. ID -Уникальный идентификатор (тип данных: serial)
2. Name -Название (тип данных: varchar)
3. Date_of_Create -Дата создания (тип данных: date)
4. Country_ID -ID Страна, в которой она была создана (тип данных: int)

Связана с таблицами:

- Films (см. описание таблицы Films)
- Countries через поле Country_ID (m:1) - Страна, в которой она была создана

Справочная таблица **Countries**

Предназначена для хранения информации о странах.

Структура таблицы:

1. ID -Уникальный идентификатор (тип данных: serial)
2. Name -Название (тип данных: varchar)

Связана с таблицами:

- Films (см. описание таблицы Films)
- Persons (см. описание таблицы Persons)
- Studios (см. описание таблицы Studios)

Справочная таблица **Genres**

Предназначена для хранения информации о жанрах.

Структура таблицы:

1. ID -Уникальный идентификатор (тип данных: serial)
2. Name -Название (тип данных: varchar)

Связана с таблицами:

- Films (см. описание таблицы Films)

Справочная таблица **Rating_Producer**

Предназначена для хранения информации об организациях, чьи оценки фильмов присутствуют в базе данных.

Структура таблицы:

1. ID -Уникальный идентификатор (тип данных: serial)
2. Name -Название (тип данных: varchar)

Связана с таблицами:

- Films_Ratings через поле Rating_Producer_ID таблицы Films_Ratings (m:1) - Связь оценки и организации

Второстепенная таблица **Films_Ratings**

Предназначена для хранения информации о рейтингах.

Структура таблицы:

1. Film_ID -ID фильма (тип данных: serial)
2. Rating_Producer_ID -ID организации (тип данных: int)
3. Rating -Оценка фильма (тип данных: int)

Связана с таблицами:

- Films (см. описание таблицы Films)
- Rating_Producer (см. описание таблицы Rating_Producer)

Связующая таблица **Films_Genries**

Предназначена для связи между таблицами Films и Genries (жанр фильма).

Структура таблицы:

1. Genrie_ID -ID жанра (тип данных: int)
2. Film_ID -ID фильма (тип данных: int)

Связана с таблицами:

- Films (см. описание таблицы Films)
- Genries (см. описание таблицы Films)

Связующая таблица **Films_Actors**

Предназначена для связи между таблицами Films и Persons (актеры, играющие в фильме).

Структура таблицы:

1. Actor_ID -ID актера (тип данных: int)
2. Film_ID -ID фильма (тип данных: int)

Связана с таблицами:

- Films (см. описание таблицы Films)
- Persons (см. описание таблицы Films)

Легкие запросы:

1.

Описание запроса:

Вывожу имя, фамилию и дату рождения актеров, которым меньше 40 лет. Сортирую в алфавитном порядке, начиная с имени.

```
SELECT Name, Surname, Birthdate
FROM Persons
WHERE Birthdate > now() - interval '40 years'
Order BY Name, Surname;
```

Оптимизация запроса:

Было:

```
Sort  (cost=1.56..1.58 rows=8 width=440) (actual time=1.043..1.047 rows=12 loops=1)
  Sort Key: name, surname
  Sort Method: quicksort  Memory: 25kB
  ->  Seq Scan on persons  (cost=0.00..1.44 rows=8 width=440) (actual time=0.466..0.497
rows=12 loops=1)
        Filter: (birthdate > (now() - '40 years'::interval))
        Rows Removed by Filter: 13
 Planning time: 420.046 ms
 Execution time: 1.844 ms
```

```
CREATE INDEX Persons_Birthdate_index ON Persons(Birthdate);
```

Стало:

```
 Sort  (cost=8.40..8.42 rows=8 width=440) (actual time=26.223..26.227 rows=12 loops=1)
  Sort Key: name, surname
  Sort Method: quicksort  Memory: 25kB
  ->  Index Scan using persons_birthdate_index on persons  (cost=0.14..8.28 rows=8
width=440) (actual time=26.082..26.094 rows=12 loops=1)
        Index Cond: (birthdate > (now() - '40 years'::interval))
 Planning time: 0.412 ms
 Execution time: 49.475 ms
```

2.

Описание запроса:

Вывожу имя и фамилию актеров мужского пола из России. Сортирую в алфавитном порядке, начиная с фамилии.

```
SELECT  Surname, Name
FROM Persons
WHERE Sex = 'true' AND Country_ID = 1
Order BY Surname, Name;
```

Оптимизация запроса:

Было:

```
Sort  (cost=1.32..1.33 rows=1 width=436) (actual time=0.053..0.054 rows=5 loops=1)
  Sort Key: surname, name
  Sort Method: quicksort  Memory: 25kB
  ->  Seq Scan on persons  (cost=0.00..1.31 rows=1 width=436) (actual time=0.017..0.022
rows=5 loops=1)
        Filter: (sex AND (country_id = 1))
        Rows Removed by Filter: 20
 Planning time: 0.183 ms
 Execution time: 0.075 ms
```

```
CREATE INDEX Persons_Sex_index ON Persons(Sex);
CREATE INDEX Persons_Country_ID_index ON Persons(Country_ID);
```

Стало:

```
 Sort  (cost=8.16..8.17 rows=1 width=436) (actual time=0.162..0.163 rows=5 loops=1)
  Sort Key: surname, name
  Sort Method: quicksort  Memory: 25kB
  ->  Index Scan using persons_country_id_index on persons  (cost=0.14..8.15 rows=1
width=436) (actual time=0.125..0.129 rows=5 loops=1)
        Index Cond: (country_id = 1)
        Filter: sex
        Rows Removed by Filter: 3
 Planning time: 0.171 ms
 Execution time: 0.187 ms
```

3.

Описание запроса:

Вывожу имя и общую прибыль фильма, при условии, что общие сборы больше бюджета.

```
SELECT   Name, Rental - Budget AS Profit
FROM Films
WHERE Rental > Budget
ORDER BY Profit;
```

Оптимизация запроса:

Было:

```
 Sort  (cost=1.26..1.27 rows=5 width=222) (actual time=12.278..12.281 rows=8 loops=1)
   Sort Key: ((rental - budget))
   Sort Method: quicksort  Memory: 25kB
   ->  Seq Scan on films  (cost=0.00..1.20 rows=5 width=222) (actual time=0.085..0.096
rows=8 loops=1)
         Filter: (rental > budget)
         Rows Removed by Filter: 7
 Planning time: 220.914 ms
 Execution time: 12.339 ms
```

```
CREATE INDEX Films_Rental_index ON Films(Rental);
CREATE INDEX Films_Budget_index ON Films(Budget);
```

Стало:

```
Sort  (cost=10000000001.26..10000000001.27 rows=5 width=222) (actual time=0.038..0.039
rows=8 loops=1)
   Sort Key: ((rental - budget))
   Sort Method: quicksort  Memory: 25kB
   ->  Seq Scan on films  (cost=10000000000.00..10000000001.20 rows=5 width=222)
(actual time=0.020..0.024 rows=8 loops=1)
         Filter: (rental > budget)
         Rows Removed by Filter: 7
 Planning time: 0.142 ms
 Execution time: 0.065 ms
```

4.

Описание запроса:

Вывожу название фильмов и месяц их премьеры, при условии, что он вышел осенью. Сортирую в порядке месяцов.

```
SELECT Name, date_part('month',Premiere_Date) AS Month
FROM Films
WHERE date_part('month',Premiere_Date) > 8 AND date_part('month',Premiere_Date) < 12
Order By date_part('month',Premiere_Date);
```

Оптимизация запроса:

Было:

```
 Sort  (cost=1.39..1.39 rows=1 width=226) (actual time=0.056..0.057 rows=6 loops=1)
   Sort Key: (date_part('month'::text, (premiere_date)::timestamp without time zone))
   Sort Method: quicksort  Memory: 25kB
   ->  Seq Scan on films  (cost=0.00..1.38 rows=1 width=226) (actual time=0.030..0.040
rows=6 loops=1)
         Filter: ((date_part('month'::text, (premiere_date)::timestamp without time zone) >
'8'::double precision) AND (date_part('month'::text, (premiere_date)::timestamp without time
zone) < '12'::double precision))
         Rows Removed by Filter: 9
 Planning time: 0.264 ms
 Execution time: 0.094 ms
```

```
CREATE INDEX Films_Premiere_Date_index ON Films(Premiere_Date);
```

Стало:

```
 Sort  (cost=10000000001.39..10000000001.39 rows=1 width=226) (actual
time=0.059..0.059 rows=6 loops=1)
   Sort Key: (date_part('month'::text, (premiere_date)::timestamp without time zone))
   Sort Method: quicksort  Memory: 25kB
   ->  Seq Scan on films  (cost=10000000000.00..10000000001.38 rows=1 width=226)
(actual time=0.025..0.035 rows=6 loops=1)
         Filter: ((date_part('month'::text, (premiere_date)::timestamp without time zone) >
'8'::double precision) AND (date_part('month'::text, (premiere_date)::timestamp without time
zone) < '12'::double precision))
         Rows Removed by Filter: 9
 Planning time: 0.216 ms
```

Средние запросы:

1.

Описание запроса:

Вывожу имя, фамилию и страну актёров женского пола. Сортирую в алфавитном порядке, начиная с фамилии.

```
SELECT  p.Surname, p.Name, c.Name AS Country
FROM Persons p INNER JOIN Countries c ON c.ID = p.country_ID
WHERE p.Sex = 'false'
Order BY Surname, Name;
```

Оптимизация запроса:

Было:

```
 Sort  (cost=2.75..2.78 rows=12 width=654) (actual time=0.086..0.087 rows=5 loops=1)
   Sort Key: p.surname, p.name
   Sort Method: quicksort  Memory: 25kB
   ->  Hash Join  (cost=1.20..2.54 rows=12 width=654) (actual time=0.050..0.055 rows=5
loops=1)
         Hash Cond: (p.country_id = c.id)
         ->  Seq Scan on persons p  (cost=0.00..1.25 rows=12 width=440) (actual
time=0.023..0.026 rows=5 loops=1)
               Filter: (NOT sex)
               Rows Removed by Filter: 20
         ->  Hash  (cost=1.09..1.09 rows=9 width=222) (actual time=0.015..0.015 rows=9
loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on countries c  (cost=0.00..1.09 rows=9 width=222) (actual
time=0.008..0.010 rows=9 loops=1) Planning time: 0.231 ms
 Execution time: 0.123 ms
```

```
CREATE INDEX Countries_ID_index ON Countries(ID);
CREATE INDEX Persons_Country_ID_index ON Persons(Country_ID);(Создано выше)
CREATE INDEX Persons_Sex_index ON Persons(Sex);(Создано выше)
```

Стало:

```
 Sort  (cost=21.06..21.09 rows=12 width=654) (actual time=0.111..0.114 rows=5 loops=1)
   Sort Key: p.surname, p.name
   Sort Method: quicksort  Memory: 25kB
   ->  Hash Join  (cost=8.63..20.84 rows=12 width=654) (actual time=0.059..0.070 rows=5
loops=1)
         Hash Cond: (c.id = p.country_id)
         ->  Index Scan using countries_id_index on countries c  (cost=0.14..12.27 rows=9
width=222) (actual time=0.015..0.020 rows=9 loops=1)
         ->  Hash  (cost=8.35..8.35 rows=12 width=440) (actual time=0.027..0.027 rows=5
loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Index Scan using persons_sex_index on persons p  (cost=0.14..8.35 rows=12
width=440) (actual time=0.013..0.018 rows=5 loops=1)
                     Index Cond: (sex = false)
                     Filter: (NOT sex)
 Planning time: 0.552 ms
 Execution time: 0.197 ms
```

2.

Описание запроса:

Вывожу название фильмов и имена с фамилиями актеров, которые в них снимались, при условии, что фильм снимала студия Castle Rock Entertainment. Сортирую в порядке названия фильмов.

```
SELECT Temp_Table.Name AS Film, p.Surname, p.Name
FROM (SELECT *
FROM Films_Actors fa INNER JOIN Films f ON fa.Film_ID = f.ID) AS Temp_Table INNER
JOIN Persons p ON Temp_Table.Actor_ID = p.ID
WHERE Temp_Table.Studio_ID = 1
Order BY Temp_Table.Name;
```

Оптимизация запроса:

Было:

```
 Sort  (cost=4.15..4.16 rows=3 width=654) (actual time=0.141..0.142 rows=12 loops=1)
   Sort Key: f.name
   Sort Method: quicksort  Memory: 26kB
   ->  Nested Loop  (cost=1.34..4.12 rows=3 width=654) (actual time=0.075..0.103 rows=12
loops=1)
         ->  Hash Join  (cost=1.20..2.83 rows=3 width=222) (actual time=0.061..0.074 rows=12
loops=1)
               Hash Cond: (fa.film_id = f.id)
               ->  Seq Scan on films_actors fa  (cost=0.00..1.45 rows=45 width=8) (actual
time=0.021..0.024 rows=45 loops=1)
               ->  Hash  (cost=1.19..1.19 rows=1 width=222) (actual time=0.030..0.030 rows=4
loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on films f  (cost=0.00..1.19 rows=1 width=222) (actual
time=0.011..0.015 rows=4 loops=1)
                           Filter: (studio_id = 1)
                           Rows Removed by Filter: 11
         ->  Index Scan using persons_id_index on persons p  (cost=0.14..0.42 rows=1
width=440) (actual time=0.002..0.002 rows=1 loops=12)
               Index Cond: (id = fa.actor_id)
```

```
CREATE INDEX Films_Actors_Film_ID_index ON Films_Actors(Film_ID);
CREATE INDEX Films_ID_index ON Films(ID);
CREATE INDEX Films_Actors_Actor_ID_index ON Films_Actors(Actor_ID);
CREATE INDEX Persons_ID_index ON Persons(ID);
CREATE INDEX Films_Studio_ID_index ON Films(Studio_ID);
```

Стало:

```
 Sort  (cost=17.64..17.65 rows=3 width=654) (actual time=0.398..0.401 rows=12 loops=1)
   Sort Key: f.name
   Sort Method: quicksort  Memory: 26kB
   ->  Nested Loop  (cost=0.41..17.62 rows=3 width=654) (actual time=0.231..0.300 rows=12
loops=1)
         ->  Nested Loop  (cost=0.28..16.32 rows=3 width=222) (actual time=0.219..0.245
rows=12 loops=1)
               ->  Index Scan using films_studio_id_index on films f  (cost=0.14..8.15 rows=1
width=222) (actual time=0.143..0.146 rows=4 loops=1)
                     Index Cond: (studio_id = 1)
               ->  Index Scan using films_actors_film_id_index on films_actors fa
(cost=0.14..8.16 rows=1 width=8) (actual time=0.019..0.021 rows=3 loops=4)
                     Index Cond: (film_id = f.id)
         ->  Index Scan using persons_id_index on persons p  (cost=0.14..0.42 rows=1
width=440) (actual time=0.003..0.003 rows=1 loops=12)
               Index Cond: (id = fa.actor_id)
 Planning time: 137.782 ms
 Execution time: 0.519 ms
```

3.

Описание запроса:

Вывожу название фильмов, их рейтинг и название организации, которая оценивала его, при условии, что больше 8.5 баллов или же его снимала студия Castle Rock Entertainment. Сортирую в обратном порядке рейтинга.

```
SELECT Temp_Table.Name AS Films, rp.Name AS Rating_by, Temp_Table.Rating
FROM (SELECT *
FROM Films f INNER JOIN Films_Ratings fr ON f.ID = fr.Film_ID) AS Temp_Table INNER
JOIN Rating_Producer rp ON rp.ID = Temp_Table.Rating_Producer_ID
WHERE Temp_Table.Rating > 8.5 OR Temp_Table.Studio_ID = 1
ORDER BY -Temp_Table.Rating;
```

Оптимизация запроса:

Было:

```
 Sort  (cost=4.39..4.42 rows=10 width=444) (actual time=0.157..0.159 rows=15 loops=1)
   Sort Key: ((- fr.rating))
   Sort Method: quicksort  Memory: 26kB
   ->  Hash Join  (cost=2.54..4.23 rows=10 width=444) (actual time=0.122..0.141 rows=15
loops=1)
         Hash Cond: (fr.rating_producer_id = rp.id)
         ->  Hash Join  (cost=1.34..2.93 rows=10 width=226) (actual time=0.070..0.084
rows=15 loops=1)
               Hash Cond: (fr.film_id = f.id)
               Join Filter: ((fr.rating > '8.5'::double precision) OR (f.studio_id = 1))
               Rows Removed by Join Filter: 12
               ->  Seq Scan on films_ratings fr  (cost=0.00..1.27 rows=27 width=12) (actual
time=0.011..0.014 rows=27 loops=1)
               ->  Hash  (cost=1.15..1.15 rows=15 width=226) (actual time=0.028..0.028 rows=15
loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     ->  Seq Scan on films f  (cost=0.00..1.15 rows=15 width=226) (actual
time=0.010..0.015 rows=15 loops=1)
         ->  Hash  (cost=1.09..1.09 rows=9 width=222) (actual time=0.042..0.042 rows=9
loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 9kB
               ->  Seq Scan on rating_producer rp  (cost=0.00..1.09 rows=9 width=222) (actual
time=0.031..0.034 rows=9 loops=1)
 Planning time: 0.840 ms
 Execution time: 0.222 ms
```

CREATE INDEX Films_ID_index ON Films(ID);(Создано выше)
CREATE INDEX Films_Ratings_Film_ID_index ON Films_Ratings(Film_ID);
CREATE INDEX Rating_Producer_ID_index ON Rating_Producer(ID);
CREATE INDEX Films_Ratings_Rating_index ON Films_Ratings(Rating);
CREATE INDEX Films_Studio_ID_index ON Films(Studio_ID);(Создано выше)

Стало:

```
Sort  (cost=31.49..31.52 rows=10 width=444) (actual time=21.272..21.274 rows=15 loops=1)
  Sort Key: ((- fr.rating))
  Sort Method: quicksort  Memory: 26kB
  ->  Nested Loop  (cost=0.41..31.33 rows=10 width=444) (actual time=21.176..21.243
rows=15 loops=1)
        ->  Merge Join  (cost=0.27..25.23 rows=10 width=226) (actual time=21.122..21.158
rows=15 loops=1)
              Merge Cond: (f.id = fr.film_id)
              Join Filter: ((fr.rating > '8.5'::double precision) OR (f.studio_id = 1))
              Rows Removed by Join Filter: 12
              ->  Index Scan using films_id_index on films f  (cost=0.14..12.36 rows=15
width=226) (actual time=0.085..0.091 rows=15 loops=1)
              ->  Index Scan using films_ratings_pkey on films_ratings fr  (cost=0.14..12.54
rows=27 width=12) (actual time=21.017..21.025 rows=27 loops=1)
        ->  Index Scan using rating_producer_id_index on rating_producer rp  (cost=0.14..0.60
rows=1 width=222) (actual time=0.004..0.004 rows=1 loops=15)
              Index Cond: (id = fr.rating_producer_id)
 Planning time: 338.317 ms
 Execution time: 21.394 ms
```

Сложные запросы:

1.

Описание запроса:

Вывожу названия фильмов и кол-во актеров, которые в них снимаются, при условии, что им больше 40 лет.

```
SELECT f.Name, count(f.ID) AS Actors_count
FROM Films f
INNER JOIN Films_Actors fa
ON (fa.Film_ID = f.ID)
INNER JOIN Persons p
ON (fa.Actor_ID = p.ID)
WHERE p.Birthdate > now() - interval '40 years'
GROUP BY f.ID;
```

2.

Описание запроса:

Вывожу названия фильмов и кол-во актеров, которые в них снимаются, при условии, что их возраст больше среднего (Предполагается, что все актеры занесенные в БД до сих пор живы).

```
SELECT f.Name, count(f.ID) AS Actors_count
FROM Films f
INNER JOIN Films_Actors fa
ON (fa.Film_ID = f.ID)
INNER JOIN Persons p
ON (fa.Actor_ID = p.ID)
WHERE (EXTRACT(YEAR FROM now()::date) - EXTRACT(YEAR FROM Birthdate))
>(SELECT AVG(EXTRACT(YEAR FROM now()::date) - EXTRACT(YEAR FROM Birthdate))
FROM Persons)
GROUP BY f.ID;
```

3.

Описание запроса:

Вывожу названия студий и кол-во фильмов, которые они сняли, средние сборы за фильм и общие сборы, при условии, что студия создана в США.

```
SELECT s.Name, sum(f.Rental) AS Total_Rental, avg(f.Rental) AS Average_Rental,
count(f.ID) AS Films_count
FROM Studios s
INNER JOIN Films f
ON (f.Studio_ID = s.ID)
WHERE s.Country_ID = 2
GROUP BY (s.Name);
```

4.

Описание запроса:

Вывожу названия студий и кол-во фильмов, которые они сняли, средние сборы за фильм и общие сборы, при условии, студия сняла больше одного фильма.

```
SELECT T.Name, T.Total_Rental, T. Average_Rental, T.Films_count
FROM
(SELECT s.Name, sum(f.Rental) AS Total_Rental, avg(f.Rental) AS Average_Rental,
count(f.ID) AS Films_count
FROM Studios s
INNER JOIN Films f
ON (f.Studio_ID = s.ID)
GROUP BY (s.Name)) AS T
WHERE T.Films_count > 1
ORDER BY T.Average_Rental;
```

.

5.

Описание запроса:

Вывожу названия фильмов и их максимальный рейтинг, при условии, что он выше среднего.

```
SELECT f.Name AS Film, max(fr.Rating) AS Max_Rating
FROM  Films f
INNER JOIN Films_Ratings fr
ON f.ID = fr.Film_ID
INNER JOIN Rating_Producer rp
ON rp.ID = fr.Rating_Producer_ID
WHERE fr.Rating > (SELECT avg(fr.Rating) FROM Films_Ratings fr)
GROUP BY f.Name;
```