Лабораторная работа 2

Задачи: освоить операции реляционной алгебры, оконные функции. Сохранение запросов в виде представлений. Фактически после этой работы вы сможете составить запрос любой сложности и создать удобное представление результатов.

**Содержание**

* 1 Алиасы таблиц и выбираемых полей
* 2 Работа с реляционной алгеброй
  + 2.1 Объединения
  + 2.2 Введение в JOIN
    - 2.2.1 Внутреннее объединение INNER JOIN
    - 2.2.2 Объединения слева и справа LEFT JOIN, RIGHT JOIN
    - 2.2.3 Полное объединение FULL JOIN
    - 2.2.4 Декартово произведение
  + 2.3 Операции над множествами
    - 2.3.1 Объединение UNION
    - 2.3.2 Полное объединение UNION ALL
    - 2.3.3 Пересечение INTERSECT
    - 2.3.4 Вычитание EXCEPT
* 3 Оконные функции
* 4 Создание представлений CREATE VIEW
* 5 Использование табличных выражений вместо представлений
* 6 Комплексные выборки
* 7 Защита лабораторной
* 8 Полезные материалы

Алиасы таблиц и выбираемых полей

Для удобства составления запросов можно присваивать таблицам или выборкам алиасы. Это удобно в случае, если в запросе участвует несколько таблиц и вы хотите обратиться к конкретной.

select (select title from title t where t.id=ci.movie\_id) from cast\_info ci;

Также можно присваивать алиасы значениям выбираемых полей. Это, например, удобно для последующей работы с этими полями:

select \* from

(select production\_year, count(1) as cnt from title group by production\_year) t

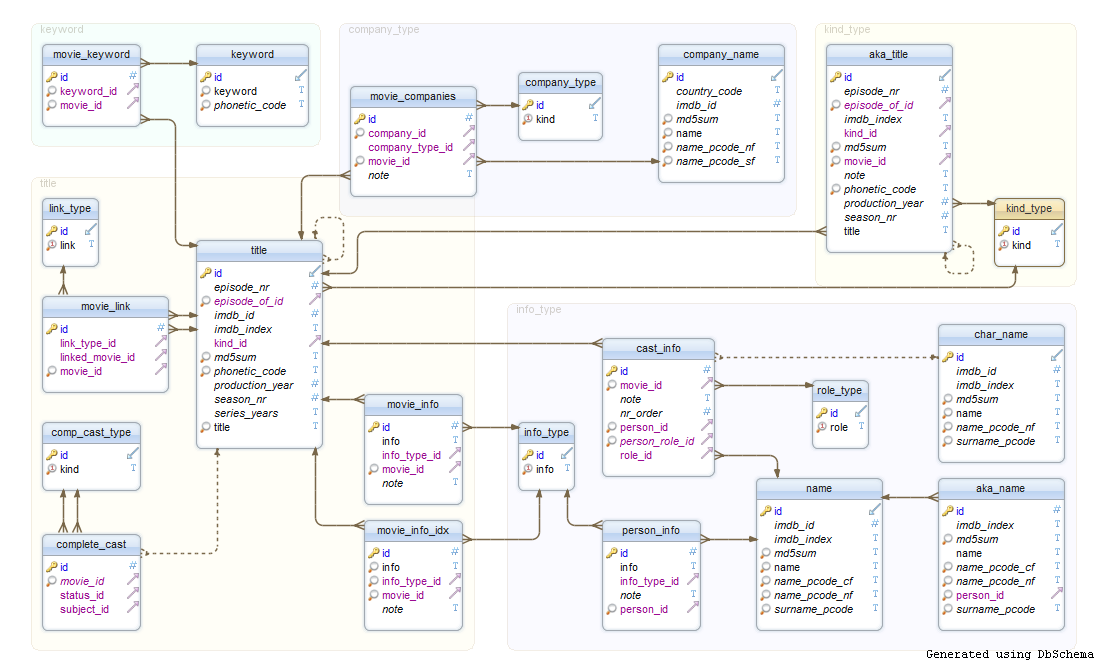
where t.cnt > 50000

Работа с реляционной алгеброй

В предыдущей лабораторной работе вы работали как правило с одной таблицей, выполняя операции селекции (добавляя условия в WHERE) и проекцией (перечисляя, что именно вы хотите выбрать между SELECT и FROM, в том числе, выполняя подзапросы к другим базам). Основные механизмы, которые дают большое преимущество реляционным СУБД относятся к взаимодействию нескольких отношений или результатов выборок. Эта лабораторная работа завершает введение и исчерпывает тему DML.

**Объединения**

Объединять множества в реляционных базах данных можно двумя способами: вертикально и горизонтально. Горизонтальные объединения используются намного чаще, так как реляционные базы нормализованы и нужные вам атрибуты могут быть раскиданы по разным сущностям. Посмотрите, как это выглядит в базе IMDb:



Используя связи между сущностями, вы можете горизонтально объединять таблицы и таким образом расширять результаты вышей выборки. Операции горизонтального соединения таблиц представлены в языках стандарта SQL ключевым словом JOIN. Рассмотрим варианты использования горизонтальных объединений.

**Введение в JOIN**

Присоединение обычно происходит по некоторым атрибутам, которые называются первичными и внешними ключами. Первичный ключ обычно называется id, он содержится во всех таблицах, которые претендуют на участие в верно построенной реляционной базе. Также таблицы могут содержать внешние ключи - это атрибуты, как правило типа unsigned integer, которые содержат те же значения, что и первичные ключи соответствующих сущностей.

Например, теперь, если вы хотите получить отчет, в котором будут следующие поля: имя актера, имя персонажа, фильм, год выпуска фильма, то вам нужно присоединить к таблице cast\_info таблицы, где содержатся названия фильмов, имена актеров и имена персонажей, а именно: char\_name, title и name, которые нужно присоединить по их первичным ключам.

Синтаксис команды join:

SELECT \* FROM t1

JOIN t2 ON t1.id=t2.t1\_id

В условии соединения ON можно указывать множество условий используя скобки и AND/OR.

Альтернативный способ выборки из нескольких таблиц:

SELECT \* FROM t1, t2

WHERE t1.id=t2.t1\_id

В результате запросов вы получите все сопоставленные записи и атрибуты из обеих таблиц.

**Внутреннее объединение INNER JOIN**

Обычный JOIN сопоставляет кортежи соединяемых сущностей по указаному в ON правилу. Если какое-либо из значений по которому соединяется сущность не задано (NULL), то кортеж не попадет в результирующую выборку.

Пример человекопонятной выборки, кто где снимался:

select t.title, n.name, c.name from cast\_info ca

join title t on t.id=ca.movie\_id

join name n on n.id=ca.person\_id

join char\_name c on c.id=ca.person\_role\_id

where t.kind\_id=1

limit 10;

Лимит указан для быстрой демонтсрации запроса.

**Задания:**

* Сделайте следующий отчет: название компании, выпускающей кинофильмы; название фильма; факты о фильме - для какой-либо выбранной вами кинокомпании (выберите более менее активную) за период в 5 последних лет. Диапазон дат укажите не конкретными датами, а относительно сегодняшнего дня, поищите соответствующие функции работы с временем.
* Соберите статистику по количеству фильмов выпускаемых разными странами (ищите страну в movie\_info).

**Объединения слева и справа LEFT JOIN, RIGHT JOIN**

В случае, если вы хотите обязательно выбрать все кортежи из первой таблицы, даже если нет соответствующих картежей во второй, используйте LEFT JOIN. Если соответствующих записей не будет, то атрибуты второй таблицы будут указаны как NULL.

Выбрать ключевые слова с названиями фильмов, но так чтобы все фильмы попали в выборку.

select t.title, k.keyword from title t

left join movie\_keyword mk on mk.movie\_id=t.id

left join keyword k on k.id=mk.keyword\_id

where mk.id is null

limit 100;

Лимит указан для быстрой демонтсрации запроса.

RIGHT JOIN соответственно оставляет все значения в присоединяемой таблице, а для кортежей, для которых не удалось сопоставить кортежи из первой таблицы, прописывает NULL.

**Задания:**

1. Найдите фильмы, в которых нет персонажей.
2. Найдите актеров, которые никогда не снимались в фильмах. (двумя способами: с помощью JOIN и с помощью WHERE NOT EXIST)

**Полное объединение FULL JOIN**

Работает аналогично LEFT и RIGHT JOIN: в случае, если какому-то кортежу из первой или второй таблице не удалось найти пару, то вместо соответствующих значений прописывается NULL.

Пока никаких идей для полезного запроса, full join нужен обычно для того, чтобы обнаруживать нарушения целостности. Попробуйте узнать, есть ли в cast\_info записи с невалидными идентификаторами актеров, фильмов или ролей и фильмы и роли, для которых нет записей в cast\_info. Или проверьте, что в таблице фактов об актерах нет невалидных идентификаторов (ссылки на несуществующих актеров). Если вы корректно удалили актера в первой лабораторной работе или просто ничего не делали, то таких записей не должно быть.

**Декартово произведение**

Как было ранее показано, существует альтернативный способ выборки из нескольких таблиц:

SELECT \* FROM t1, t2 WHERE t1.id=t2.t1\_id

Если не указывать условий, то таблицы перемножатся: то есть в результате вы получите выборку, в которой каждая запись из одной таблицы сопоставляется с каждой записью из другой таблицы.

Посчитайте, сколько всего различных комбинаций актеров и персонажей может быть исходя из базы. В этой выборке следует также учесть, что персонаж может появляться в разных фильмах, но называться одинаково, при этом просто умножив cast\_info на что-то не получится. Выбирайте сразу количество, не выводя результаты выборки.

**Операции над множествами**

Вертикальные объединения обычно используют для манипуляции с готовыми запросами. Для того, чтобы объединить две выборки, нужно, чтобы у них совпадали имена атрибутов.

Синтаксис:

select \* from t1

union

select \* from t2

**Объединение UNION**

Объединяет кортежи выборок, при этом убирает дубли.

**Полное объединение UNION ALL**

Объединяет кортежи выборок, при этом не убирает дубли.

**Пересечение INTERSECT**

Возвращает кортежи, которые есть в обоих отношениях.

**Вычитание EXCEPT**

Возвращает кортежи, которые есть в первом, но нет во втором отношениях.

Оконные функции

Оконные функции - это наиболее мощный инструмент, заложенный в стандарте SQL. Применение оконных функций к месту обычно помогает существенно упростить запросы.

Оконные функции позволяют выполнять подзапросы и собирать статистику для каждого результирующего кортежа по некоторому окну данных. Обычно окном данных выступают кортежи сгруппированные по значению какого-либо атрибута.

Для начала создадим небольшой сэмпл данных через представление (см. ниже), чтобы упростить демонстрационные запросы:

create view movie\_contries as (

select info as country, movie\_id, title, t.production\_year

from movie\_info mi

join title t on mi.movie\_id=t.id and mi.info\_type\_id=8 and t.kind\_id=1

)

Фильмы с указанием страны и года выпуска:

select \* from movie\_contries;

При этом каждый фильм может упоминаться несколько раз, если в производстве участвовало несколько стран.

Данный запрос позволяет узнать, в каком году страны выпустили свой первый фильм:

select distinct(country), min(production\_year) over (partition by country) from movie\_contries;

* partition by country - означает, что выборка будет прохоидть для каждой группы с одинаковым значением country
* min(production\_year) - для каждой из групп, которые определены далее, выбрать минимальное значение
* distinct(country) - так как over не производит группировку сам и вычисляет результат окна для каждой строки исходного отношения, то добавим distinct, чтобы вернуть только уникальные значения для country и соответствующие значения результатов выборки по окну.

Дополнительно:

* <https://habrahabr.ru/post/268983/>
* <http://www.postgresql.org/docs/9.3/static/tutorial-window.html>

Создание представлений CREATE VIEW

Запросы можно сохранять в виде представлений, чтобы затем обращаться к ним как к таблицам.

create view female\_actors as

select \* from name where gender='f'

После этого вы сможете выбирать из представлений как из таблиц:

select count(1) from female\_actors;

Представления поддерживают целостность базы данных, и, если обновятся данные в таблице, из которой строится представление, то обновятся данные и в представлении.

Можно обновлять данные через представления. При этом значения обновятся в соответствующих таблицах. Ограничением в данном случае являются представления, составленные из запросов с использованием функций агрегации. Так как результаты таких представлений не хранят исходные данные, через них невозможно найти кортежи, которые нужно обновить.

Дополнительно о синтаксисе:

* <http://www.postgresql.org/docs/9.3/static/sql-createview.html>

**Задание:** Создайте представление, которое будет показывать проблемы с объектами в базе. Это будет несколько выборок с полями: имя сущности (фильм, актер, компания, персонаж), идентификатор объекта в его таблице, текстовое название объекта, комментарий с описанием проблемы. Эти выборки должны быть объединены с помощью UNION. Выборки следующего содержания:

* Фильмы без актеров (именно фильмы, см. kind\_type)
* Актеры, которые нигде не снимались
* Компании, которые не выпустили ни одного фильма.
* Персонажи, не участвовавшие ни в одном фильме.

После создания представления, намеренно испортите некоторые кортежи в таблицах, чтобы выборки возвращали новые значения. Убедитесь, что ваши изменения попадают в результаты выборки из представления.

Использование табличных выражений вместо представлений

В ситуации, когда у вас недостаточно прав, чтобы создать представление, можно использовать конструкцию WITH ... AS (), с помощью которой можно на время выполнения основного запроса создать именованное временное представление.

Синтаксис использования табличного выражения:

with actress as

(select \* from name where gender='f')

select \* from actress limit 10;

Использование нескольких табличных выражений в одном запросе:

with actress as

(select \* from name where gender='f'),

height\_cm as

(select person\_id, to\_number(info, '999') as height from person\_info where info\_type\_id=22 and info like '%cm%')

select from actress a

join height\_cm h on h.persin\_id;

Здесь также использовано преобразование строки в число с помощью to\_number. Подробнее об этом [тут](http://www.postgresql.org/docs/9.3/static/functions-formatting.html).

Комплексные выборки

Для защиты выполните любые две выборки. Для выполнения этих заданий используйте всё, что знаете.

1. Выберите 5 своих любимых (или случайных) сериалов, поля: название сериала, количество эпизодов, среднее количество персонажей в каждом эпизоде, название эпизода с наибольшим количеством актеров, количество актеров в этом эпизоде.
2. Выберите фильмы, в ключевых словах которых есть 'super' (используйте приведение атрибута к нижнему регистру для сравнения) и выберите для них: название фильма, суммарные кассовые сборы, количество человек, участвовавших над созданием фильма, средний доход с фильма на человека: чистая прибыль (сборы в США минус бюджет) поделенная на количество людей. Подсказка: для выделения числа из строки со значением бюджета подойдет функция с регулярным выражением: regexp\_replace(info, '\D', '', 'g'); и также обратите внимание, что строки с бюджетами содержат разные валюты, отфильтруйте значения только для США.
3. Выберите топ 10 фильмов, над созданием которых потрудилось больше всего людей (таблица cast\_info), поля: название фильма, количество всех, кто участвовал в создании, количество актеров
4. Выберите топ 10 актеров, которые снялись в наибольшем количестве фильмов, поля: имя актера, дата рождения, количество фактов, количество фильмов, в которых он снялся, кинокомпания, с которой актер сотрудника над наибольшим количеством фильмов.
5. Выберите топ 10 кинорежиссеров, которые сняли фильмы с наибольшим количеством задействованных людей, поля: название фильма, среднее количество людей, которые принимают участие в фильме режиссера, дата рождения режиссера, количество фактов о режиссере.

**Важно:** если у вас что-то выполняется очень долго, то выполняйте задания параллельно с третьей лабораторной работой, анализируя план выполнения запроса и создавая индексы.

Защита лабораторной

1. Объяснить, как работают составленные запросы.
2. По просьбе преподавателя модифицировать написанный ранее запрос.

Рекомендуется сохранить все запросы, так как в последующих лабораторных работах они понадобятся.

Полезные материалы

Интерактивные уроки по SQL (уроки строятся на работе с СУБД SQLite, поэтому некоторые функции не будут работать в PostgreSQL, но обычные запросы объясняются хорошо):

1. Learn SQL: <https://www.codecademy.com/learn/learn-sql>
2. Table Transformation: <https://www.codecademy.com/learn/sql-table-transformation>