Лабораторная работа 1

Задачи: освоить самые необходимые навыки настройки СУБД, получения данных и манипуляции с ними, а также составления простых отчетов.

**Содержание**

* 1 Введение
* 2 Часть 1: Установка PostgreSQL
* 3 Часть 2: Основы администрирования PostgreSQL
* 4 Часть 3: импорт и экспорт базы данных IMDB
  + 4.1 Структура базы IMDB
* 5 Часть 4: Простые операции CRUD
  + 5.1 Добавление данных INSERT
  + 5.2 Выборка данных SELECT
  + 5.3 Обновление данных UPDATE
  + 5.4 Удаление данных DELETE
* 6 Часть 5: Агрегация данных
  + 6.1 Преобразование атрибутов по некоторым правилам
  + 6.2 Группировка агрегированных данных GROUP BY
* 7 Защита лабораторной работы
* 8 Полезные материалы

**Введение**

Так как у некоторых групп лабораторные работы начинаются раньше первой лекции, то предлагается краткий список терминов, используемых в лабораторной работе.

PostgreSQL - популярная реляционная система управления базами данных. Эта СУБД используется многими крупными компаниями, являясь единственной хорошо развитой свободной альтернативой наряду с MySQL. Но по сравнению с MySQL, PostgreSQL предоставляет больше возможностей для работы с большими объемами данных (не "big data", но до терабайта).

В качестве базы данных в лабораторных работах будет использоваться база фильмов IMDB (сам сайт также использует эту СУБД). Дамп базы достаточно большой, поэтому, если у вас есть возможность, скачайте и импортируйте его заранее.

Рекомендуется использовать Ubuntu 18.04+ и PostgreSQL 8.1+. Также нужно примерно 10 Гб места на диске.

Для выполнения запросов подойдет и терминал, но можно использовать IDE (например, DataGrip или любую другую от JetBrains с аналогичным плагином).

База данных, которая используется в лабораторных работах: <https://disk.yandex.ru/d/kBJmjsAAa74oVA>

**Часть 1: Установка PostgreSQL**

Первая задача состоит в том, чтобы установить СУБД и проверить ее работоспособность.

Выполните в терминале:

sudo apt-get update && sudo apt-get install postgresql postgresql-contrib

Сервер PostgreSQL создает отдельно пользователя в системе для доступа к базе. Чтобы переключиться на этого пользователя, выполните:

sudo -i -u postgres

Теперь вы можете войти в интерактивный режим работы с СУБД:

psql

Приглашение в интерактивном режиме выглядит так:

postgres=#

Чтобы посмотреть, какие базы уже есть в системе, наберите:

\l

Примерный результат:

postgres=# \l

List of databases

Name | Owner | Encoding | Collate | Ctype | Access privileges

-----------+----------+----------+-------------+-------------+-----------------------

postgres | postgres | UTF8 | en\_US.UTF-8 | en\_US.UTF-8 |

template0 | postgres | UTF8 | en\_US.UTF-8 | en\_US.UTF-8 | =c/postgres +

| | | | | postgres=CTc/postgres

template1 | postgres | UTF8 | en\_US.UTF-8 | en\_US.UTF-8 | =c/postgres +

| | | | | postgres=CTc/postgres

(3 rows)

Альтернативно, можно выполнить запрос:

SELECT datname FROM pg\_database;

Чтобы работать с конкретной базой, ее нужно выбрать. Выполните \c database\_name:

postgres=# \c imdb

You are now connected to database "imdb" as user "postgres".

Чтобы узнать, какие таблицы есть базе, выполните:

\d

Альтернативный запрос:

SELECT table\_name FROM information\_schema.tables WHERE table\_schema = 'public';

Чтобы узнать, какие есть колонки в таблице:

\d+ table

или

\d table

Альтернативный запрос:...

SELECT column\_name FROM information\_schema.columns WHERE table\_name ='table';

**Часть 2: Основы администрирования PostgreSQL**

Следующая задача состоит в том, чтобы настроить два важный параметра:

* логирование запросов - чтобы подтвердить, что вы честно выполняли лабу
* ручное подтверждение вносимых изменений - чтобы в случае некорректных запросов вы могли откатить свои изменения простым способом.

Про второй механизм подробнее. Когда вы вносите изменения в данные, они не сразу вступают в силу. СУБД создает diff аналогичный тому, который можно видеть в git. После этого вы начинаете работать с измененной версией, но в других сессиях данные по-прежнему старые. Если вы что-то сделали неправильно, вы можете откатить изменения в своей сессии с помощью команды rollback. Если же все изменения корректны, подтвердите их, выполнив commit. Закоммиченные изменения откатить намного сложнее, поэтому как правило в СУБД отключают опцию autocommit, которая подтверждает изменения автоматически.

Когда вы завершаете сессию, выполняется rollback. Если вы убиваете процесс, то он может еще некоторое время "держать" данные, не давая их изменить.

Приступим к конфигурированию.

PostgreSQL представлен в системе в виде сервиса, управлять которым можно как и обычно через команду service. Как правило, для внесения каких-либо изменений нужно перезапустить сервис.

Конфигурационный файл:

sudo vim /etc/postgresql/9.\*/main/postgresql.conf

Допишите или раскомментируйте:

log\_line\_prefix = '%t %c %u ' # time sessionid user

log\_statement = 'all'

Управлять некоторыми параметрами можно прямо из сессии с СУБД. Например включение подробного логирования:

SELECT set\_config('log\_statement', 'all', true);

Если вы используете Ubuntu, то по умолчанию лог-файлы могут быть найдены по пути /var/log/postgresql/\*. Если же у вас другой дистрибутив, то конфигурация может отличаться. (Например, для ArchLinux нужно в конфигурационном файле /var/lib/postgres/data/postgresql.conf указать режим логированния с помощью log\_destination='stderr' и включить запись в файл logging\_collector=on. На самом деле, лучше найти эти строчки в конфигурационном файле и почитать, что они значат. Внутри конфига всё хорошо объяснено).

Чтобы отключить автокоммит, от пользователя postgres допишите в файл или создайте новый, если его нет ~/.psqlrc:

\set AUTOCOMMIT off

Также можно инициировать процедуру, которая внесет изменения глобально только в случае выполнения commit:

BEGIN;

-- манипуляции с данными

COMMIT;

**Часть 3: импорт и экспорт базы данных IMDB**

Две наиболее важные операции. Выполняйте в сессии пользователя postgres.

Экспортировать базу данных:

pg\_dump dbname | gzip > filename.gz

Импортировать базу:

gunzip -c filename.gz | psql dbname

Попробуйте импортировать базу IMDB:

<https://disk.yandex.ru/d/kBJmjsAAa74oVA>

**Важно:** прежде чем импортировать дамп, нужно создать базу данных:

В psql выполните:

create database imdb;

**Важно: прежде чем импортировать дамп, нужно включить автокоммит.**

Это займет некоторое время (20 минут - норм).

Также можно отдельно импортировать схему и данные частями:

[https://disk.yandex.ru/d/qItnYO7-Jx-3Rw](https://1drv.ms/u/s!ArcpUIBd3-wx-flxJbZj4jmdkRv2RQ?e=WmgbZX)

Используйте, например: ls imdb3\*.gz | xargs gunzip | psql dbname

Также можно импортировать только конкретные таблицы, указав их через ключ --table.

Можно импортировать только схему: --schema-only или только данные: --data-only

**Структура базы IMDB**

У каждой таблицы есть идентификатор, указанный как первичный ключ (id). По нему выбирать быстрее всего.

Основные таблицы и их описание:

* title - названия фильмов (поле title) и год выпуска (поле production\_year); если это сериал, то также здесь можно найти номер эпизода
* movie\_info - характеристики и факты о фильме: movie\_id - идентификатор из таблицы title (далее для краткой записи: title.id), info\_type\_id - идентификатор из таблицы info\_type (info\_type.id), info - текстовое поле со значением характеристики.
* name - актеры (имя и пол)
* person\_info - характеристики и факты об актерах также с названиями характеристик из (info\_type.id)
* char\_name - роли (имена персонажей)
* cast\_info - таблица со связью ролей (person\_role\_id), актеров (person\_id) и фильмов (movie\_id)

**Часть 4: Простые операции CRUD**

К простым операциям манипуляции данными (Create, Read, Update, Delete) относятся:

* Добавление: INSERT
* Выборка: SELECT
* Обновление: UPDATE
* Удаление: DELETE

**Добавление данных INSERT**

Чтобы добавить новую запись в таблицу, нужно вычислить ее идентификатор. Для этого в PostgreSQL используются последовательности - числа, которые меняются по заданным правилам (обычно просто инкрементируются на единицу).

Чтобы посмотреть список всех последовательностей выполните:

SELECT c.relname FROM pg\_class c WHERE c.relkind = 'S';

Именование последовательностей обычно выбирают предсказуемым, чтобы легко было понять, к какой таблице они относятся.

Синтаксис INSERT выглядит так: сначала в скобках перечисляются атрибуты, которые будут вставлены, а затем после VALUES в скобках указываются значения. Можно также не перечислять атрибуты, тогда в VALUES нужно по порядку указать значения для всех.

Попробуйте добавить себя в список актеров:

insert into name (id, name, gender) values(nextval('name\_id\_seq'), 'Ivan Savin', 'm');

Здесь nextval('name\_id\_seq') генерирует следующее значение для последовательности name\_id\_seq.

**Важно:** В предлагаемом дампе базы последовательности обнулены и не могут сгенерировать уникальный идентификатор сразу. Чтобы это исправить, укажите текущее значение последовательности максимальным идентификатором в таблице, к которой она относится. Пример:

select max(id) from name;

select setval('name\_id\_seq', 5555233);

Если вы отключили автокоммит, то, так как вы вносите изменения в данные, завершите операцию, выполнив:

commit;

Если вы не уверены в своих изменениях, выполните:

rollback;

За одну операцию INSERT можно вставлять несколько строк данных. Для этого после VALUES нужно перечислить кортежи данных через запятую:

insert into name (id, name, gender)

values(nextval('name\_id\_seq'), 'Dmitry Burmistrov', 'm'),

(nextval('name\_id\_seq'), 'Victor Yakovlev', 'm');

**Выборка данных SELECT**

Для чтения данных из базы используется ключевое слово SELECT, после которого указывается список атрибутов, которые нужно получить в выборке. Если указать вместо списка атрибутов "\*", то выберутся все. Самый простой запрос выборки из базы данных выглядит следующим образом:

select \* from info\_type;

Не пробуйте выбрать все данные из больших таблиц (title, name) - это займет много времени. Если вы хотите выбрать несколько кортежей данных для примера, то ограничьте результаты с помощью LIMIT:

select \* from title limit 10;

Условия выборки указываются после ключевого слова WHERE. Условия можно комбинировать с помощью скобок и слов OR и AND. Примеры условий:

* WHERE title='Databases' - простое условие равенства
* WHERE title like '%base%' - поиск по подстроке, "%" - любое количество любых символов
* WHERE created\_date > now() - сравнение даты с текущим моментом; см. также <http://www.postgresql.org/docs/8.3/static/functions-datetime.html>
* WHERE title not in ('Databases', 'Networks') - значение не входит в список
* WHERE not exists (SELECT \* FROM ...) - выполняется, если подзапрос вернул хотя бы одну запись
* WHERE artist\_id in (SELECT id FROM artist...) - подзапрос определяет множество значений.

Пример запроса с условиями:

select \* from title where title like '%Matrix' and production\_year=1999;

Также в блоке с перечислением атрибутов можно указывать подзапросы. Подзапрос будет выполняться в последнюю очередь для каждого кортежа, удовлетворяющего остальным условиям. Также, чтобы использовать условия из основного запроса в этом подзапросе, лучше указывать название атрибута вместе с таблицей, в которой он принадлежит:

select (select info from info\_type where info\_type.id=person\_info.info\_type\_id), person\_info.info from person\_info where person\_id=1732058;

Если нужно вывести только уникальные кортежи, используйте distinct:

select distinct production\_year from title;

Попробуйте найти ваши любимые фильмы, указывая часть названия и комбинируя условия, указывая год выхода. Попробуйте найти ваших любимых актеров и факты о них.

**Обновление данных UPDATE**

Чтобы обновить данные, нужно указать, какие параметры вы хотите обновить и условия выборки обновляемых данных:

update cast\_info set person\_id=1732058 where movie\_id=3514559;

Если вы не укажите условия, то обновятся значения во всей таблице, обычно это не нужно.

В запросе на обновление также можно использовать подзапросы. Единственное ограничение: нельзя в подзапросе использовать обновляемую таблицу, так как СУБД в этом случае можно ввести в бесконечный цикл обновления. Пример более понятного запроса на обновление:

***Предостережение****: не выполняйте следующий запрос, пока не разберетесь, что именно он делает.*

update cast\_info set person\_id=(select id from name where name='Savin Ivan')

where movie\_id=(select id from title where title like 'The Matrix' and production\_year=1999);

Если вы не закоммитите изменения, то обновляемые записи останутся залоченными, то есть их невозможно будет обновить в других сессиях. Попробуйте не выполняя коммит, открыть новую сессию и попытаться обновить те же данные. В результате новая сессия зависнет, ожидая завершения транзакции в первой сессии. Закоммитьте изменения в первой сессии.

Попробуйте назначить себя и своих друзей на подходящие роли в ваших любимых фильмах. Составьте запрос, который будет демонстрировать, кто где и какую роль играет (используя подзапросы).

**Удаление данных DELETE**

Синтаксис удаления данных аналогичен синтаксису выборки за исключением того, что вместо "SELECT \* FROM" достаточно написать "DELETE FROM". Будьте внимательны, удаляя данные и проверяйте условия перед отправкой коммита.

Удалите актеров, которые вам не нравятся (или актера, выбранного случайно). Это сделать не так просто, как может показаться, потому что их идентификаторы используются в других таблицах, а нарушать целостность данных нельзя. Для корректного удаления, нужно найти все связи актера с фильмами и удалить сначала их, после этого удалить информацию о них из таблицы фактов, после этого станет доступно удаление.

**Часть 5: Агрегация данных**

Часто с помощью СУБД генерируют различные полезные отчеты. В любой популярной СУБД есть агрегирующие функции, с помощью которых, можно собрать статистику о данных. Самая простая статистика: количество записей, удовлетворяющих заданным условиям. Пример выбора количества фильмов в базе:

select count(1) from title where kind\_id=(select id from kind\_type where kind='movie');

Для числовых атрибутов также помимо COUNT можно использовать SUM, AVG и другие востребованные функции.

Попробуйте вывести, в скольких фильмах снимались ваши любимые актеры.

**Преобразование атрибутов по некоторым правилам**

Если нужно преобразовать какой-либо атрибут по правилам (условиям), то можно использоваться конструкцию CASE ... WHEN ... THEN ... END. В этом случае данная конструкция будет обозначать новый атрибут у записи. Рассмотрим её детальнее.

case

when ***condition1*** then ***result1***

when ***condition2*** then ***result2***

else ***result3***

end

Если условие ***condition1*** верно (т.е. **true**), то атрибуту будет присвоено значение ***result1***. Если ***condition1*** неверно, то case перейдёт к следующему when. Если ни один из ***condition*** не выполняется, то атрибуту будет присвоено значение, указанное в else (если в этом случае отсутствует else, то атрибут будет равен null).

Представим таблицу T:

|  |
| --- |
| **a** |
| 1 |
| 2 |
| 3 |
| 10 |

Добавим теперь к каждой записи новый атрибут, который будет обозначать значение атрибута a словами:

select

a,

case

when a = 1 then 'one'

when a = 2 then 'two'

when a = 3 then 'three'

else 'other'

end

from t;

Результат:

|  |  |
| --- | --- |
| **a** | **case** |
| 1 | one |
| 2 | two |
| 3 | three |
| 10 | other |

Как видно из результата, новый атрибут получил название case. Если мы хотим переименовать его в желаемый вариант, то это можно сделать с помощью as:

select

a,

case

when a = 1 then 'one'

when a = 2 then 'two'

when a = 3 then 'three'

else 'other'

end as new\_attribute

from t;

|  |  |
| --- | --- |
| **a** | **new\_attribute** |
| 1 | one |
| 2 | two |
| 3 | three |
| 10 | other |

Также, чтобы каждый раз не писать when a = , можно сразу указать, по какому атрибуту мы бежим:

select

a,

case a

when 1 then 'one'

when 2 then 'two'

when 3 then 'three'

else 'other'

end as new\_attribute

from t;

Таким образом, используя case..when.. можно помечать нужные нам записи для дальнейшей обработки данных. Например, добавить к фильму атрибут, обозначающий, относится фильм к 20 или 21 веку.

select

\*,

case

when production\_year >= 1900 and production\_year < 2000 then 'XX'

when production\_year >= 2000 and production\_year < 2100 then 'XXI'

end as century

from title

Сокращённый результат:

|  |  |
| --- | --- |
| id | 2395294 |
| title | (1979-03-11) |
| production\_year | 1979 |
| century | XX |

**Группировка агрегированных данных GROUP BY**

Статистику можно также сгруппировать по некоторому атрибуту, который возвращается запросом. Например, чтобы вывести количества фильмов, выпущенных в каждый год, выполните запрос:

select production\_year, count(1) from title

where kind\_id=(select id from kind\_type where kind='movie')

group by production\_year

order by production\_year;

В запросе также результаты отсортированы по году выпуска с помощью ORDER BY.

Попробуйте собрать следующие статистики: количество актеров и актрис; среднее количество фильмов в год, выпущенных в XX веке, и, выпущенных в XXI веке (с учетом текущего года).

Попробуйте также узнать среднее количество ролей в фильмах в различные годы. Этот запрос может выполняться долго, поэтому рекомендуется сначала отлаживать его на небольшом количестве данных, используя LIMIT или какие-либо условия.

**Защита лабораторной работы**

1. Показать логи выполнения запросов.
2. Прокомментировать для агрегаций, что и как происходит.
3. Написать запрос по просьбе преподавателя.

**Полезные материалы**

Интерактивный урок по основам SQL запросов (видео лекции + задания): <https://www.codeschool.com/courses/try-sql>