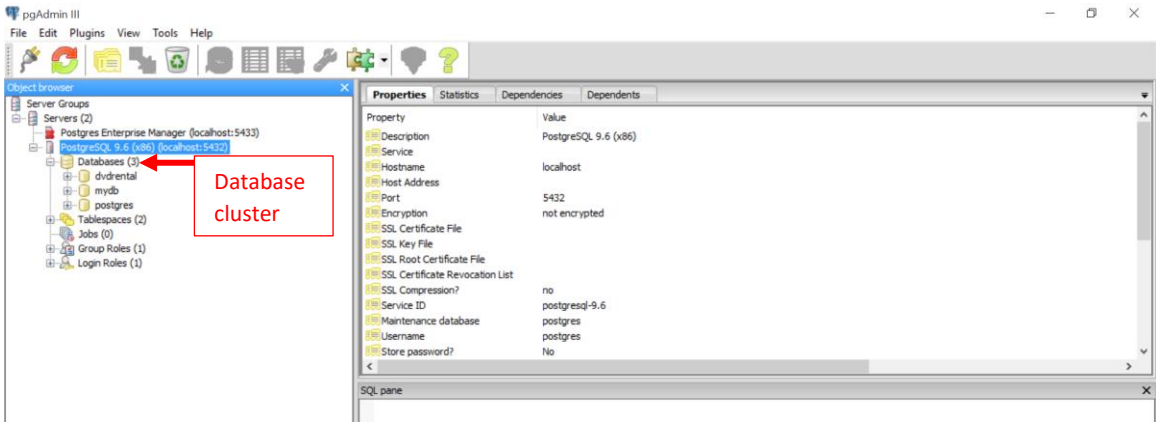
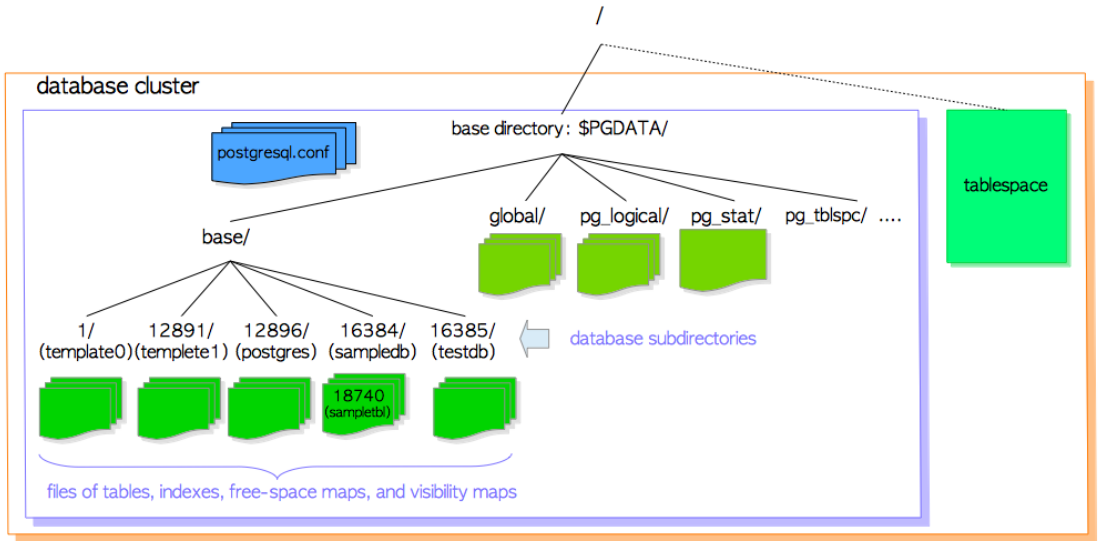
Tutorial 4

1. Database physical storage in PostgreSQL (An overview is provided in this link: www.postgresql.org/docs/current/static/storage.html)

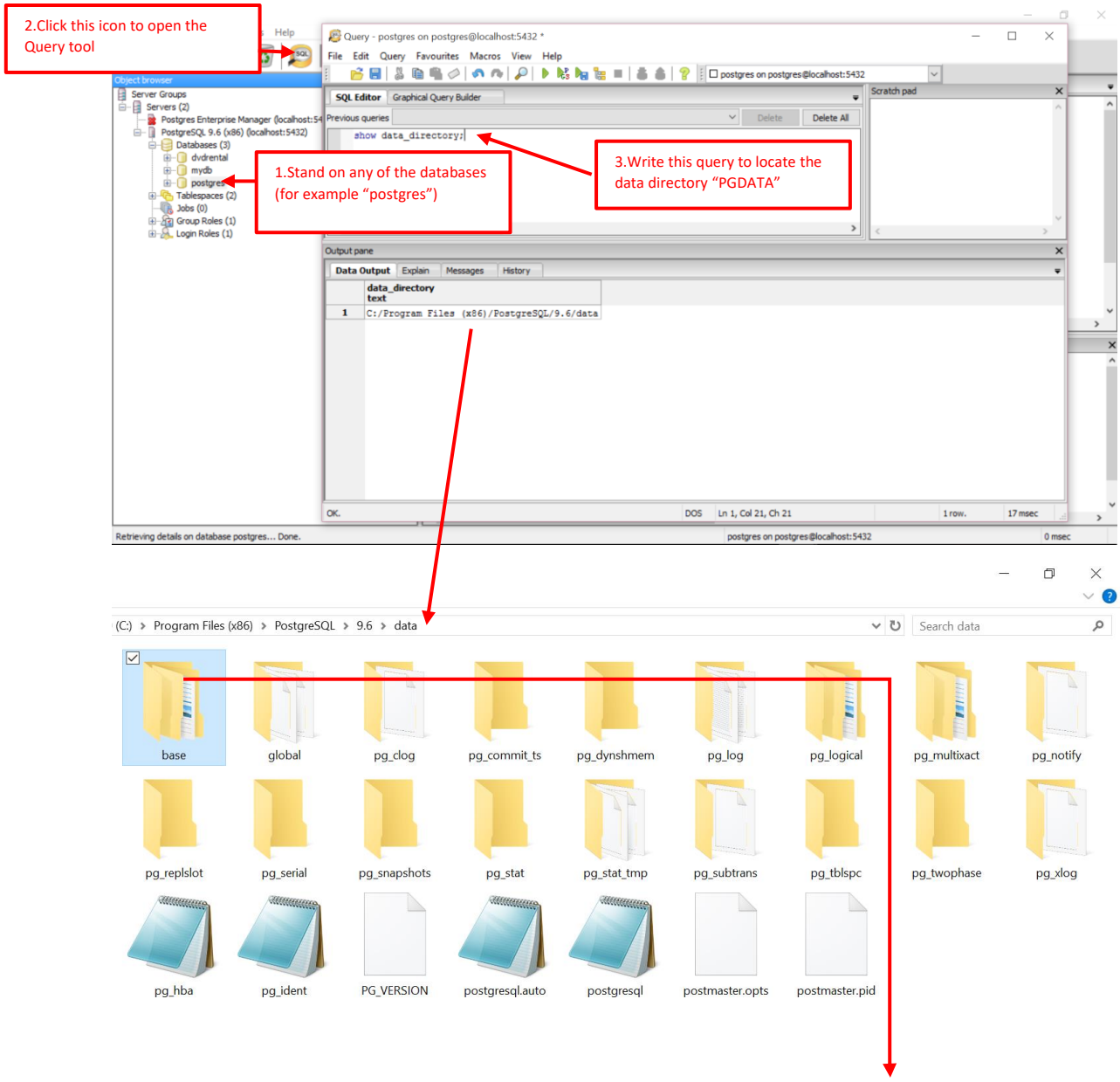
In this section, Key features will be highlighted to apply the concepts introduced in the lecture. First it is important to get an overview of "what is a database cluster?" It is a collection of databases that is managed by a single instance of a running database server. Before you can do anything, you must initialize a database storage area on disk. After initialization, a database cluster will contain a database named "postgres", which is meant as a default database for use by utilities, users and third party applications.



In file system terms, the configuration and data files used by a database cluster are stored together within the cluster's data directory, commonly called PGDATA. The PGDATA directory contains several subdirectories and control files. The cluster configuration files postgresql.conf, pg_hba.conf, and pg_ident.conf are traditionally stored in PGDATA, although it is possible to place them elsewhere.

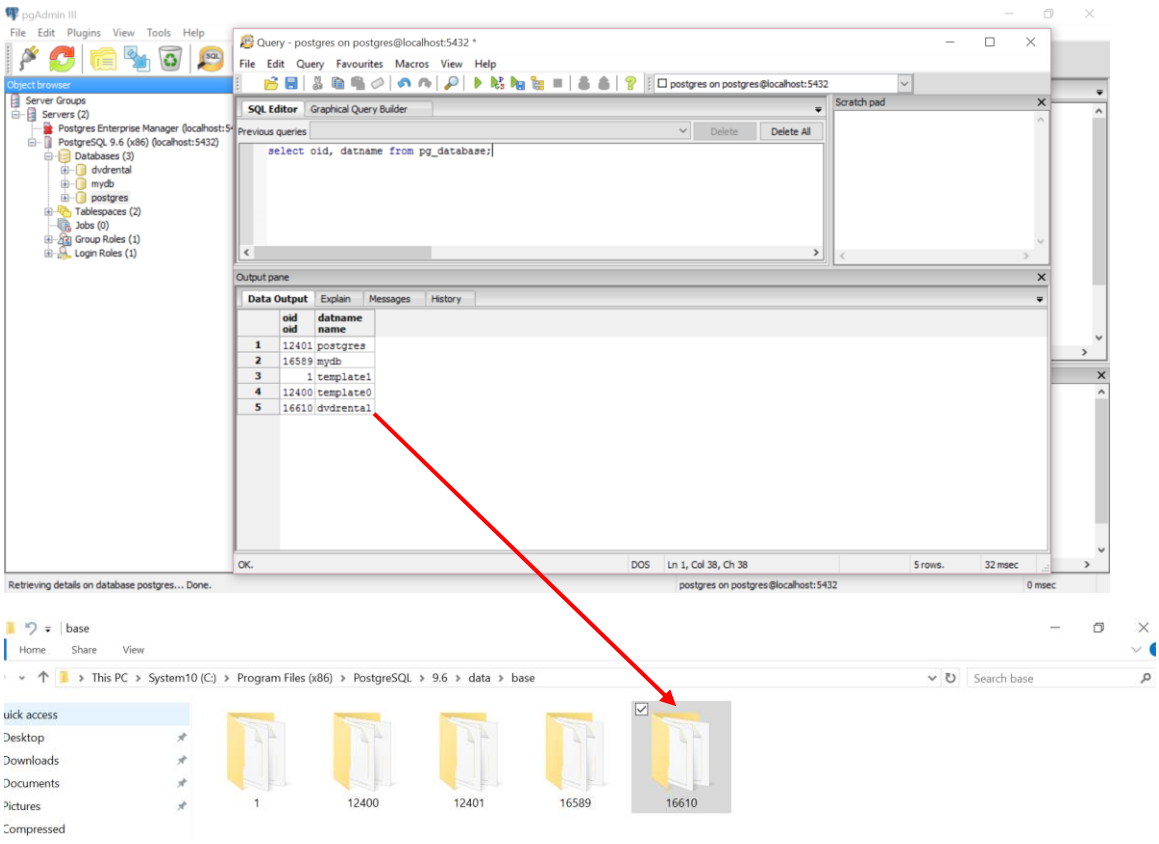


To see where the data directory is, use this query: “show data_directory;”

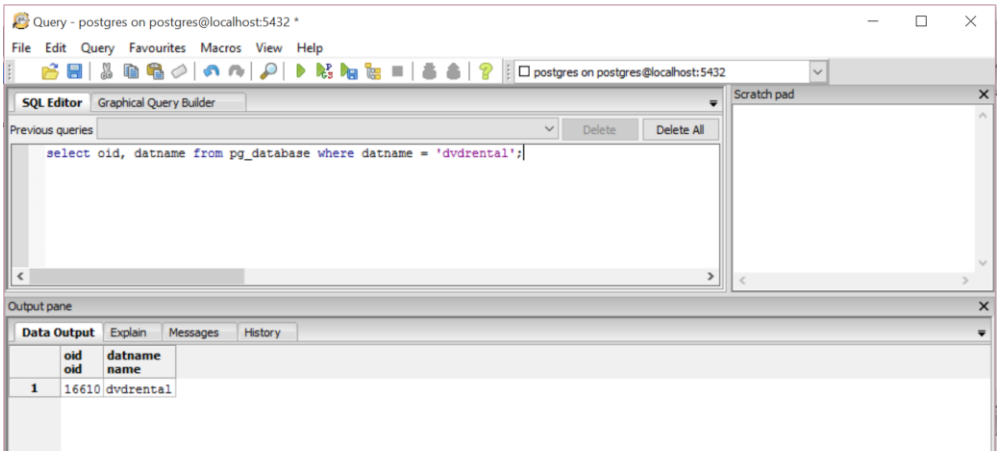


For each database in the cluster there is a subdirectory within PGDATA/base, named after the database's OID in pg_database. This subdirectory is the default location for the database's files; in particular, its system catalogs are stored there.

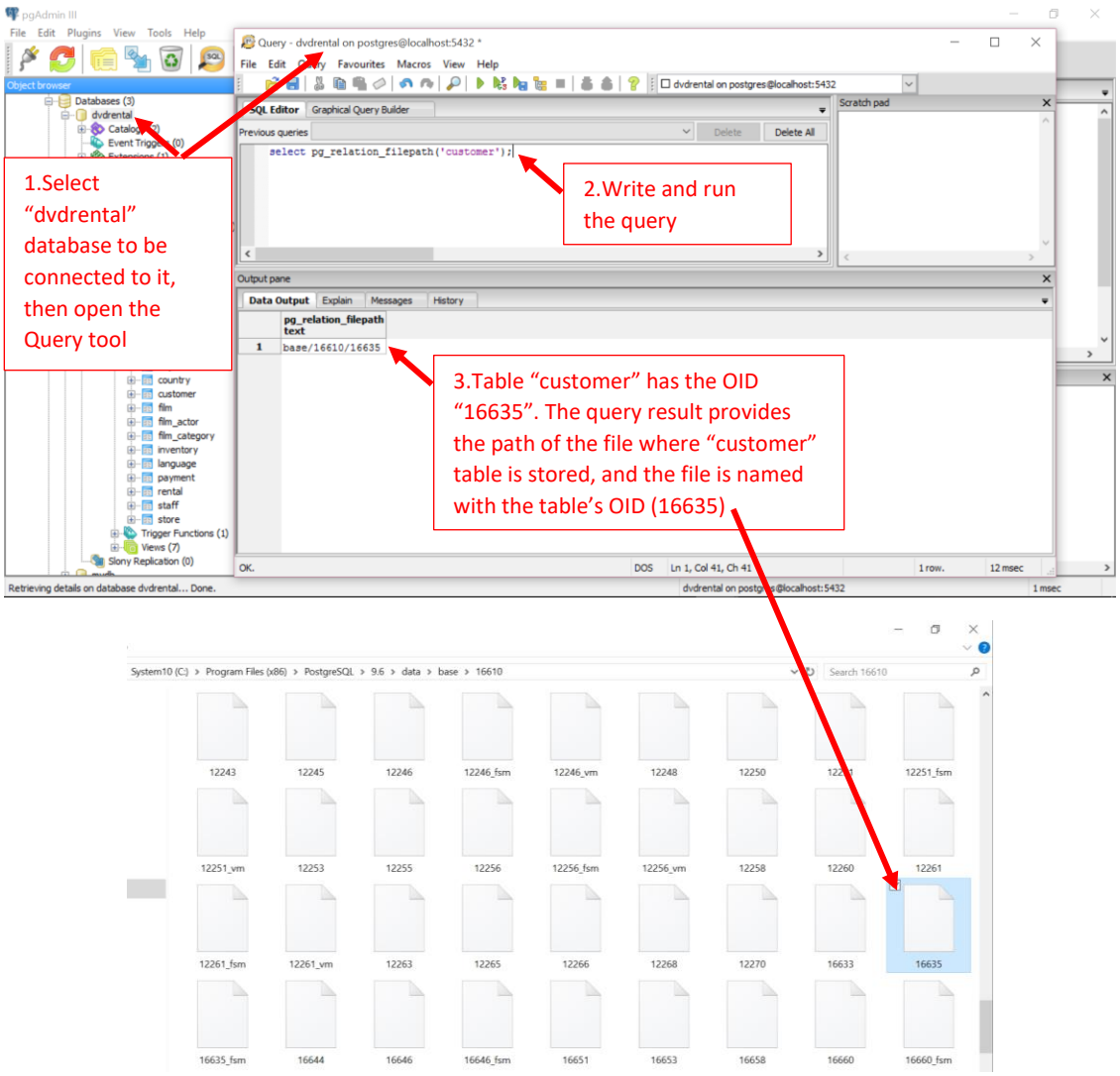
To list the OID for each database in your cluster, use this query: “select oid, datname from pg_database;” The OIDs listed in the query result represent the actual names for the databases in the physical file system. So, the database “dvdrental” is represented by a folder named “16610” and not “dvdrental”. So the path of “dvdrental” database can be: “C:\Program Files (x86)\PostgreSQL\9.6\data\base\16610”, where all files concerning this database are stored.



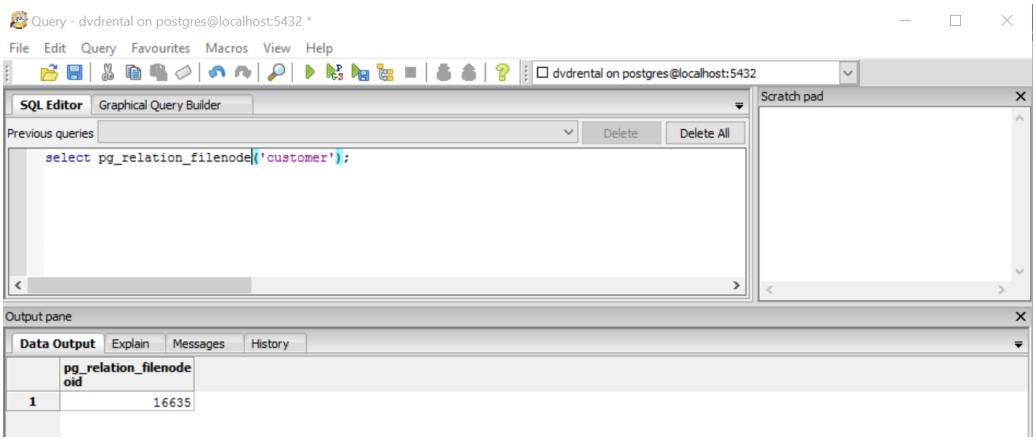
If you would like to find the OID of a specific database, for example, “dvdrental” database, use this query: “select oid, datname from pg_database where datname = 'dvdrental';”



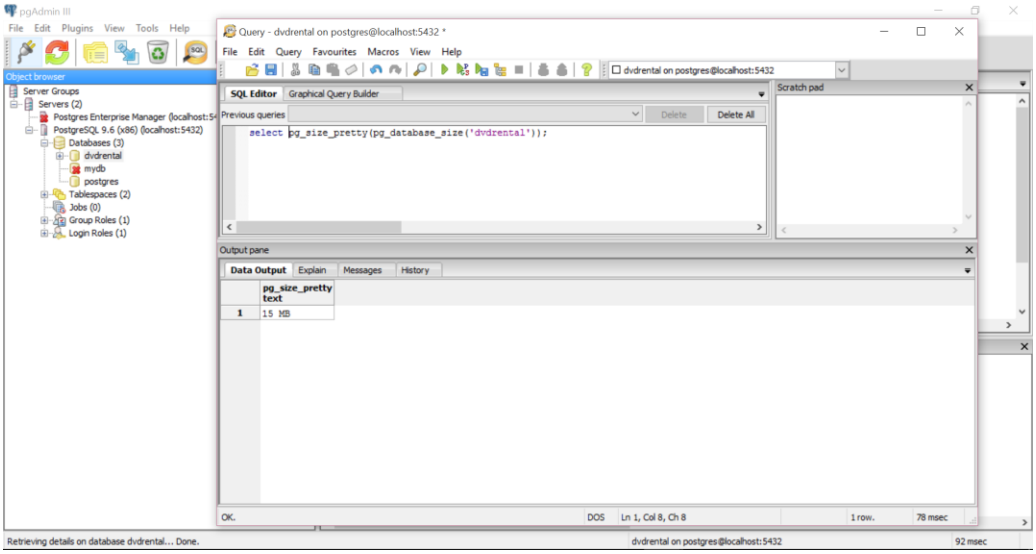
The same goes for the tables created in each database, created tables and objects are stored in the database’s folder “16610” by their OID and not their textual name. PostgreSQL stores each table in the database in a separate file (a heap file), this file is named after the table’s or index’s filenode number. This filenode number often matches the table’s OID. To find the filenode path of a particular table, for example, table “customer” in “dvdrental” database, us this query: “select pg_relation_filepath('customer');”



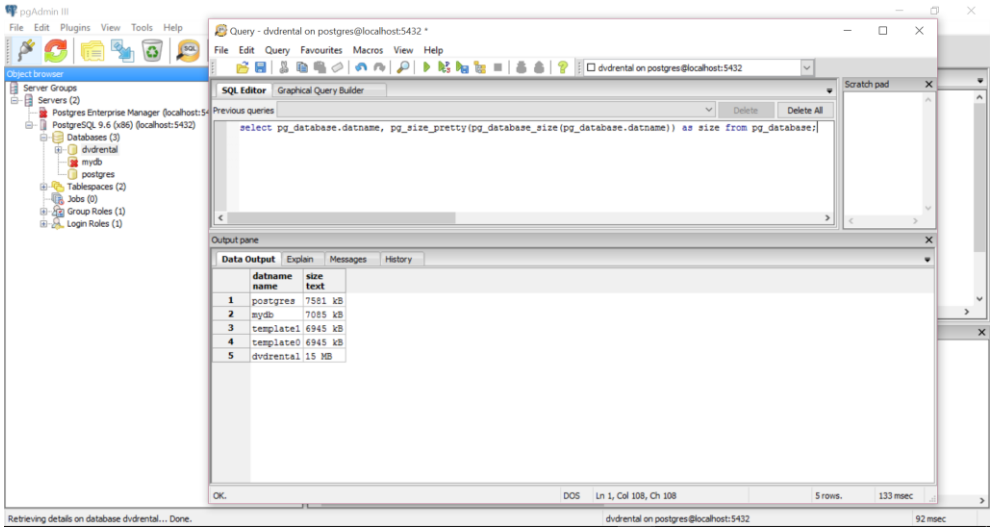
To find only the filenode for table "customer", use this query: "select pg_relation_filenode('customer');"



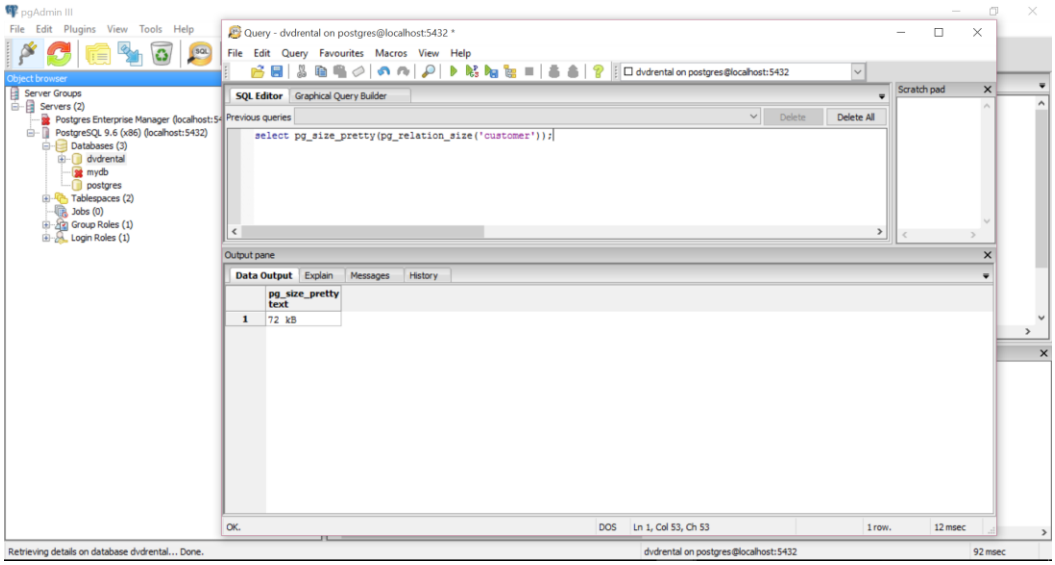
Before going down to the level of rows in a table, it is worthy to try some queries that show more information about the database and its tables. To know the size of the database “dvdrental”, try this query: “select pg_size_pretty(pg_database_size('dvdrental'));”



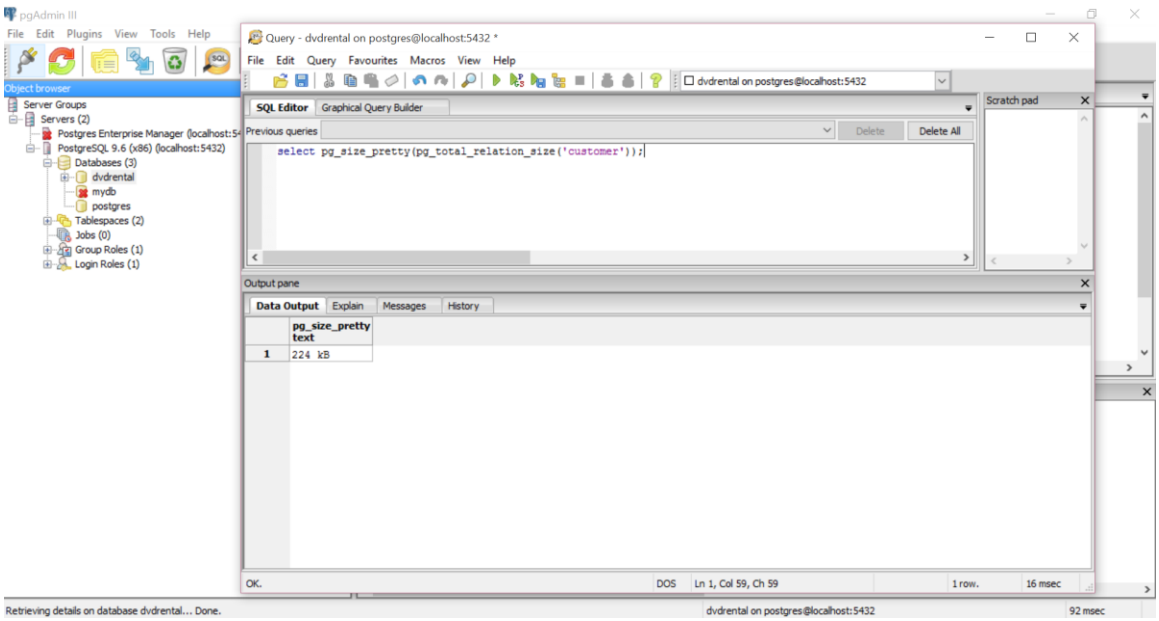
To get information on the size of each database in the database server, try this query: “select pg_database.datname, pg_size_pretty(pg_database_size(pg_database.datname)) as size from pg_database;”



Moving to tables, to get the total size of a table, for example “customer” table, try this query: “select pg_size_pretty(pg_relation_size('customer'));”



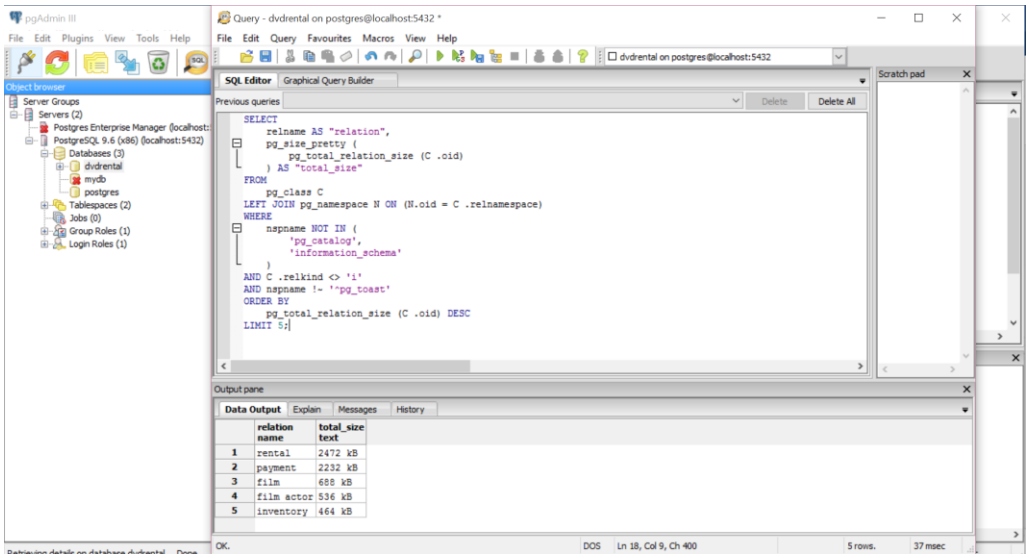
To get the total size of “customer” table including indexes and other objects, try this query: “select pg_size_pretty(pg_total_relation_size('customer'));”



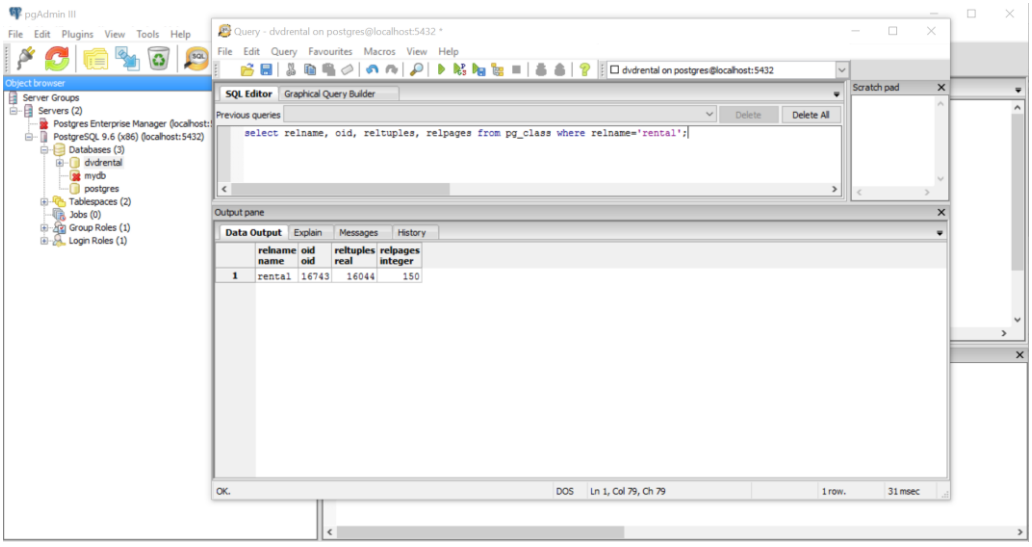
To get a list of the top 5 biggest tables, try this query:

```
“SELECT

   relname AS "relation", pg_size_pretty (pg_total_relation_size (C .oid)) AS "total_size" FROM pg_class C

LEFT JOIN pg_namespace N ON (N.oid = C .relnamespace)
```

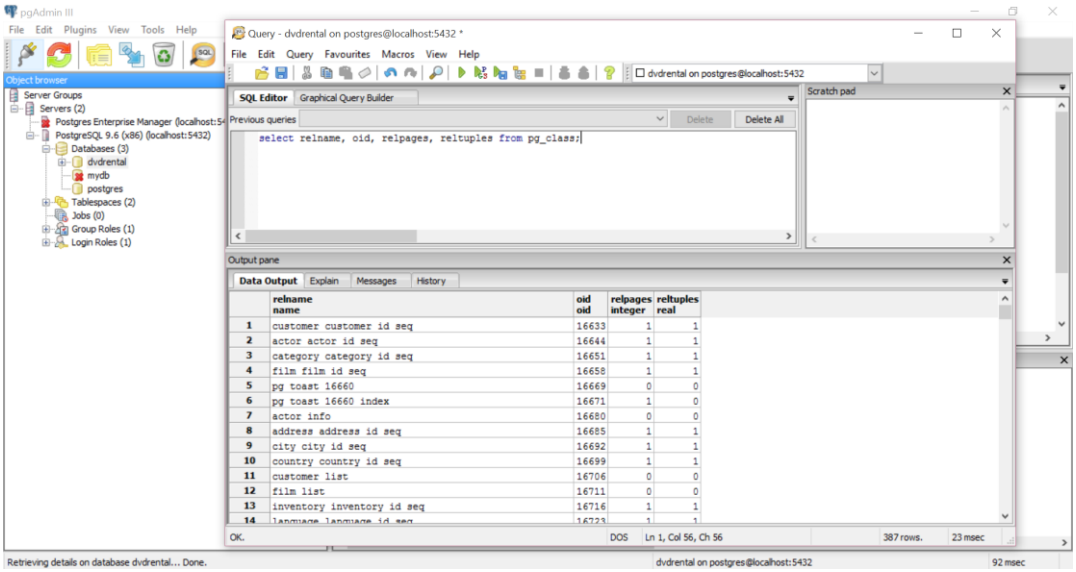
```
WHERE nspname NOT IN ('pg_catalog', 'information_schema') AND C .relkind <> 'i' AND
nspname !~ '^pg_toast'

ORDER BY pg_total_relation_size (C .oid) DESC

LIMIT 5;"
```



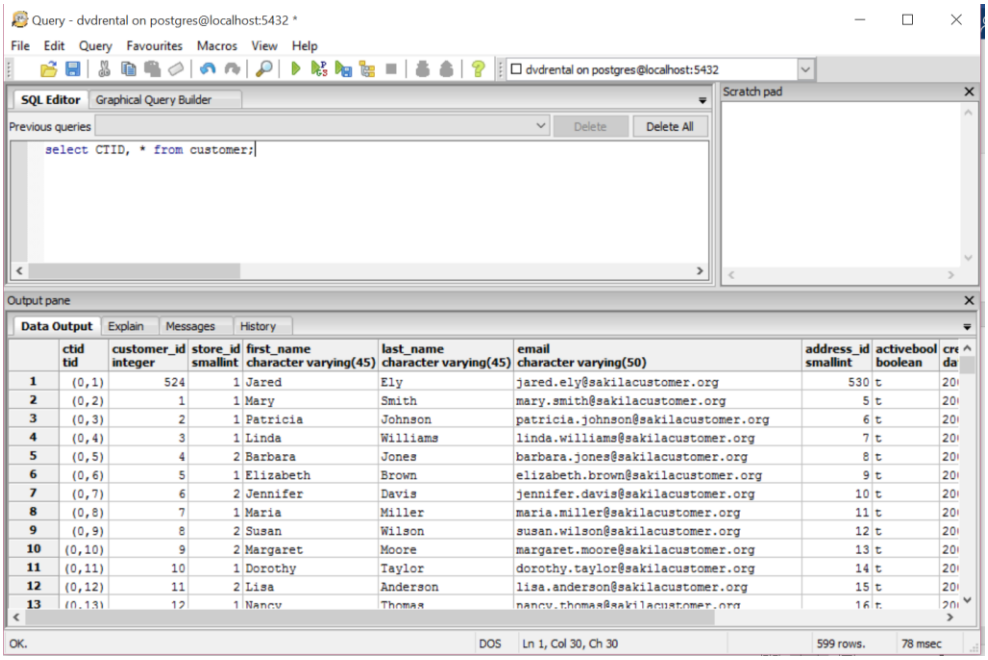
To get information on the number of pages and rows contained in table “rental” (the biggest table in “dvdrental” database, try the following query: “select relname, oid, reltuples, relpages from pg_class where relname='rental';”



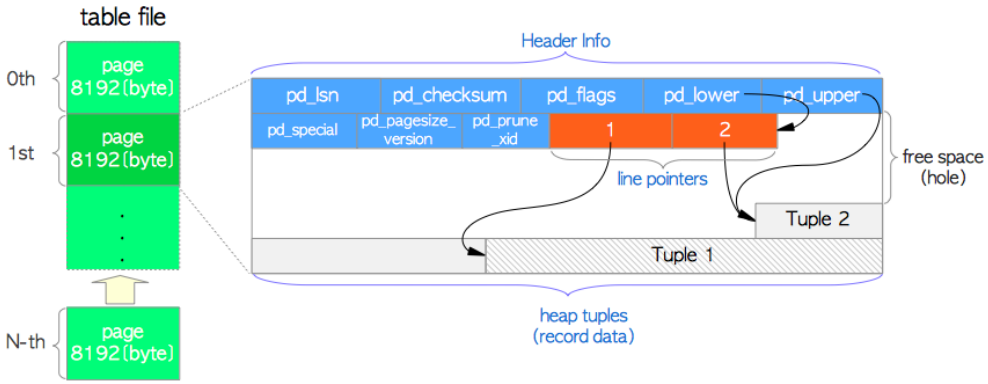
To get information on the storage structure of “dvdrental” database, try this query: “select relname, oid, relpages, reltuples from pg_class;”



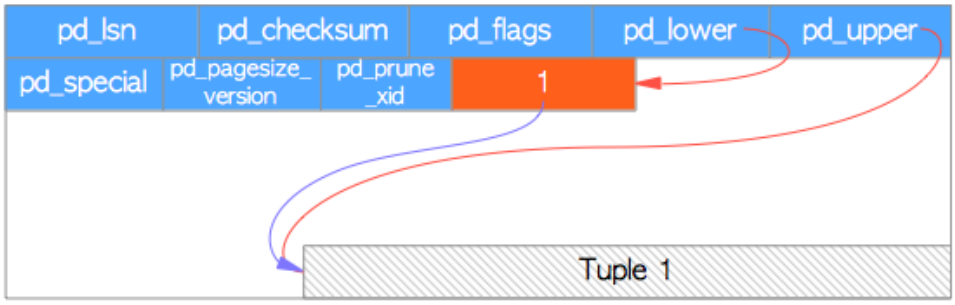
Every table is stored in an array of pages of a fixed size (8 KB), where a particular row is stored in any page. A pointer called CTID is created by PostgreSQL consisting of a page number and the index of the row's identifier. To access the CTID of a row in table "customer", use this query: "select CTID, * from customer;"



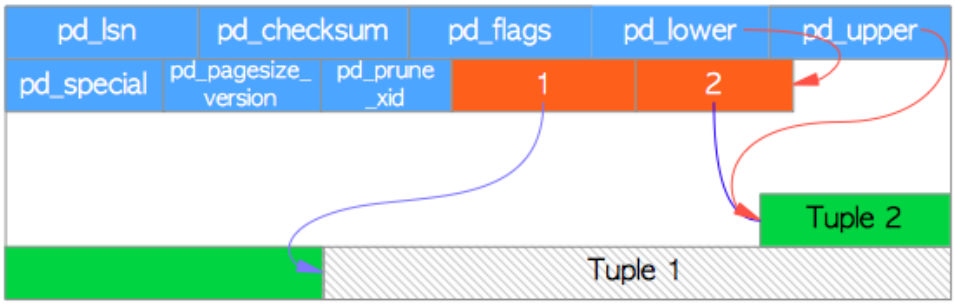
If you scroll down the output pane, you will find that table "customer" is physically stored in an array of 9 pages (from 0 to 8), each page of a fixed size 8 KB. The typical way of storing the table's records or (tuples) in pages inside the heap file is shown in the figure below. The second tuple is stored above the first tuple in a single particular page. The tuple can physically span multiple rows within the page it is stored in. what is not allowed in PostgreSQL is that the tuple spans multiple pages, and this tuple is called by in PostgreSQL as "The Oversized-Attribute Storage Technique (TOAST)".



(a) Before insertion of Tuple 2



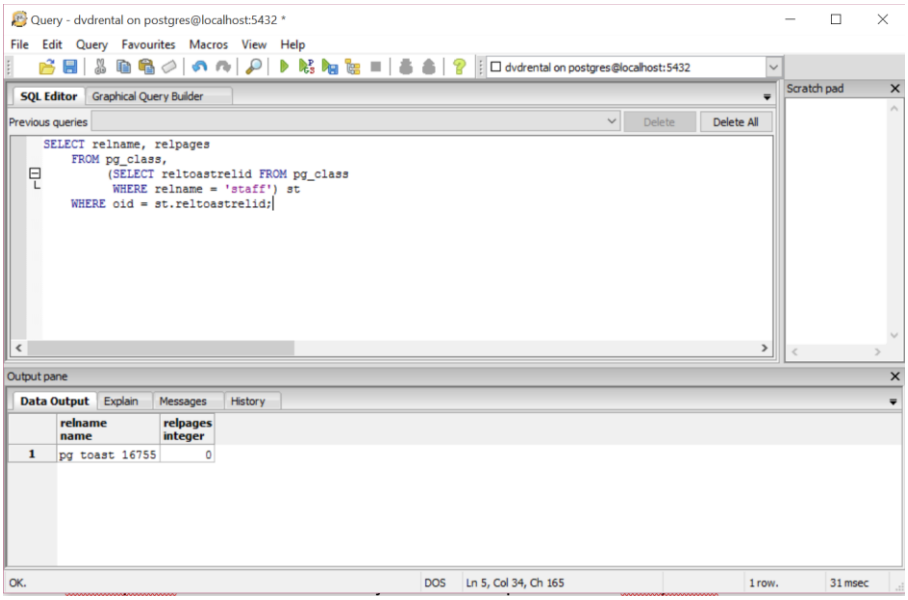
(b) After insertion of Tuple 2



When a tuple is attempted to be stored and exceeds the page size of 8 KB, TOAST breaks the data of that large tuple into smaller pieces (each piece is of size 2 KB) and stores them into a TOAST table. The TOAST table is named as pg_toast_[OID of the original table] and it may or may not be used. For example, the OID of “staff” table is “16755” and the name of its TOAST table is “pg_toast_16755”. Try the following query:

```
SELECT relname, relpages
   FROM pg_class,
      (SELECT reltoastrelid FROM pg_class
```

```
    WHERE relname = 'staff') st
WHERE oid = st.reltoastrelid;
```



Free space map (FSM): www.postgresql.org/docs/current/static/storage-fsm.html

Visibility map (VM): www.postgresql.org/docs/current/static/storage-vm.html

---

Tablespaces: www.postgresql.org/docs/9.5/static/manage-ag-tablespaces.html

Tablespaces in PostgreSQL allow database administrators to define locations in the file system where the files representing database objects can be stored. Once created, a tablespace can be referred to by name when creating database objects. Creation of the tablespace itself must be done as a database superuser, but after that you can allow ordinary database users to use it.

By using tablespaces, an administrator can control the disk layout of a PostgreSQL installation:

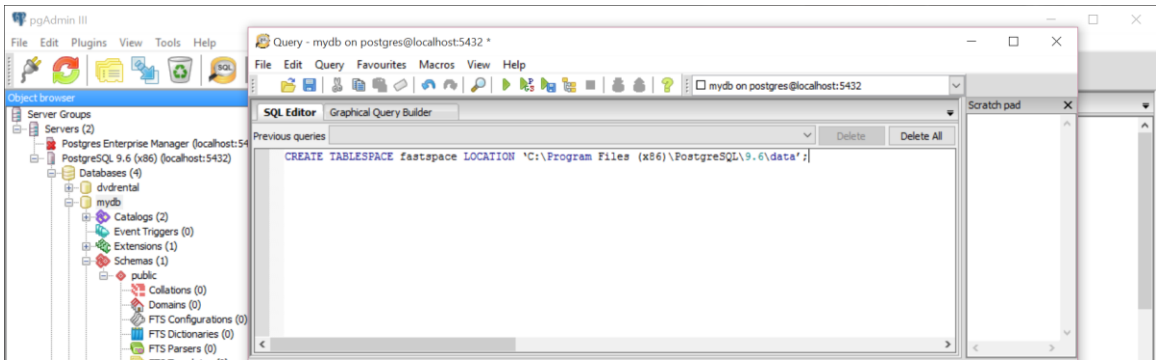
- First, if the partition or volume on which the cluster was initialized runs out of space and cannot be extended, a tablespace can be created on a different partition and used until the system can be reconfigured.

- Second, tablespaces allow an administrator to use knowledge of the usage pattern of database objects to optimize performance. For example, an index which is very heavily used can be placed on a very fast, highly available disk, such as an expensive solid state

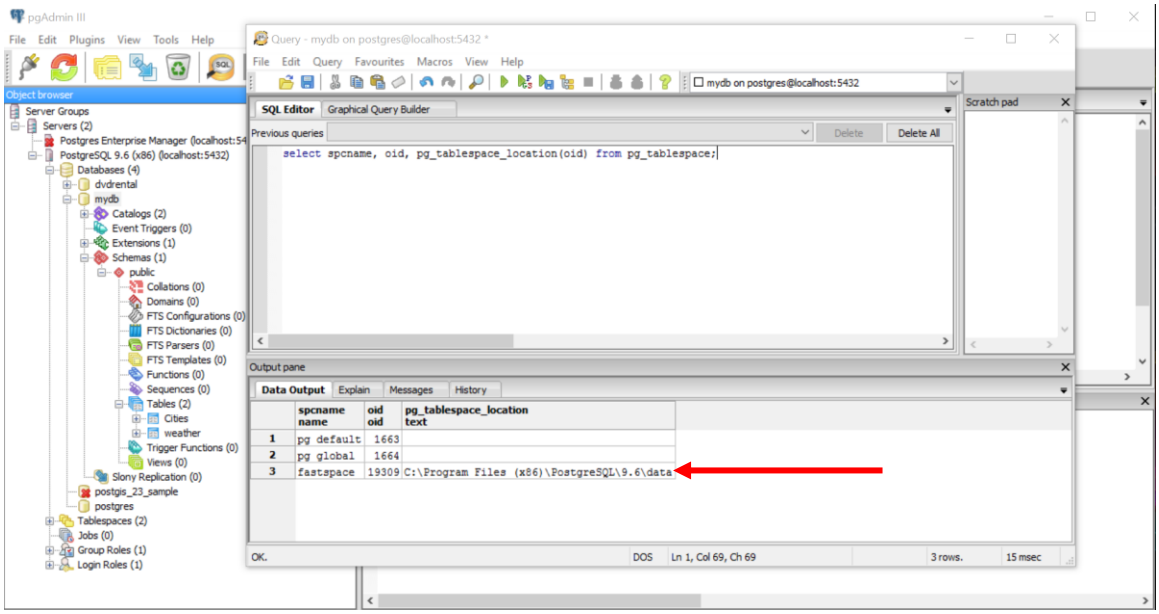
device. At the same time a table storing archived data which is rarely used or not performance critical could be stored on a less expensive, slower disk system.

The location must be an existing, empty directory that is owned by the PostgreSQL operating system user. All objects subsequently created within the tablespace will be stored in files underneath this directory. The location must not be on removable or transient storage, as the cluster might fail to function if the tablespace is missing or lost.

Select “mydb” database that you have created in Tutorial 1 and create a table space using this query: “CREATE TABLESPACE fastspace LOCATION ‘C:\Program Files (x86)\PostgreSQL\9.6\data’;”



Then try this query to make sure that the table space you have created is existing: “select spcname, oid, pg_tablespace_location(oid) from pg_tablespace;”



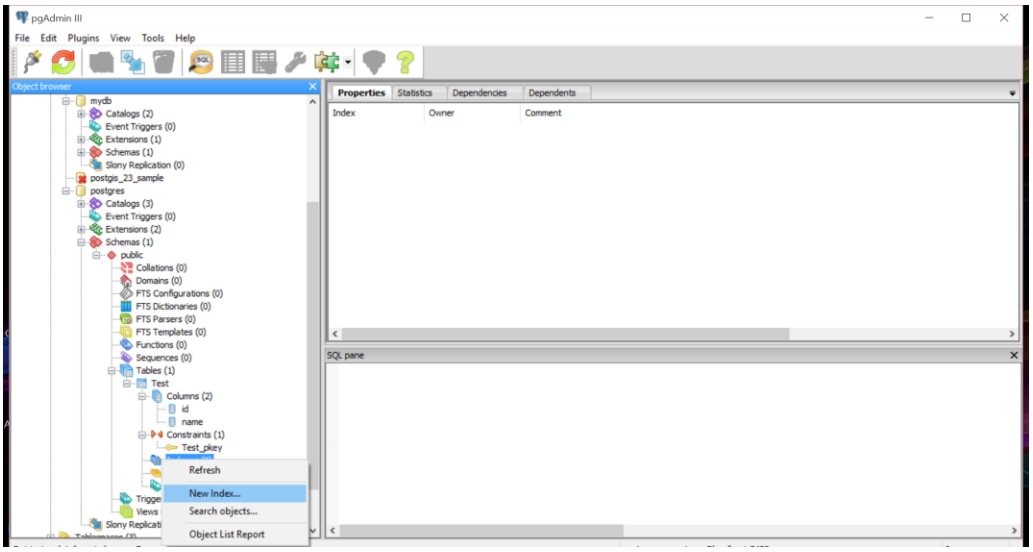
Note: The tablespace must be empty of all database objects before it can be dropped: www.postgresql.org/docs/9.5/static/sql-droptablespace.html

---

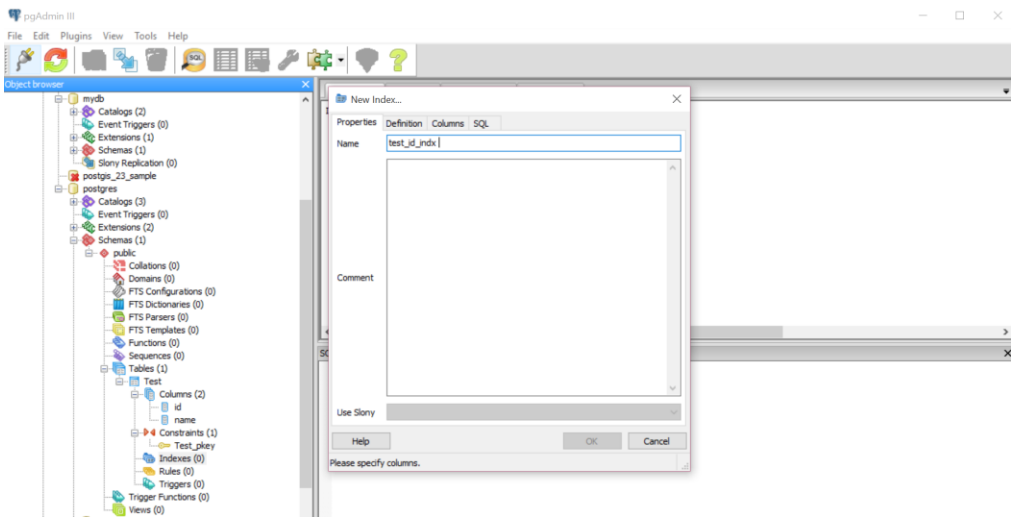
2. Indexes in PostgreSQL

The default index in PostgreSQL is btree index. To create an index, select “postgres” database, in which you have created table “Test” in Tutorial 1. Try to follow the steps below:

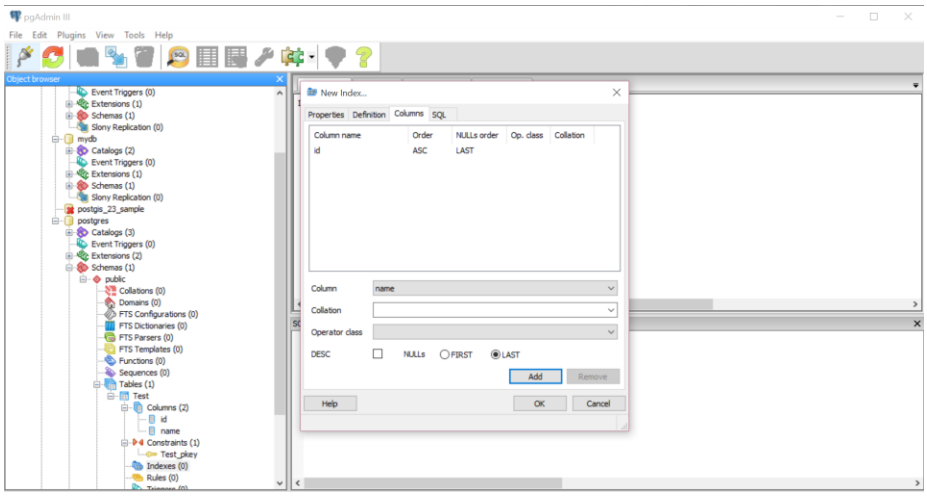
Select the “indexes” node underneath the “Test” table node in the “object browser”. Then, right-click on “indexes” node and select “New Index”.



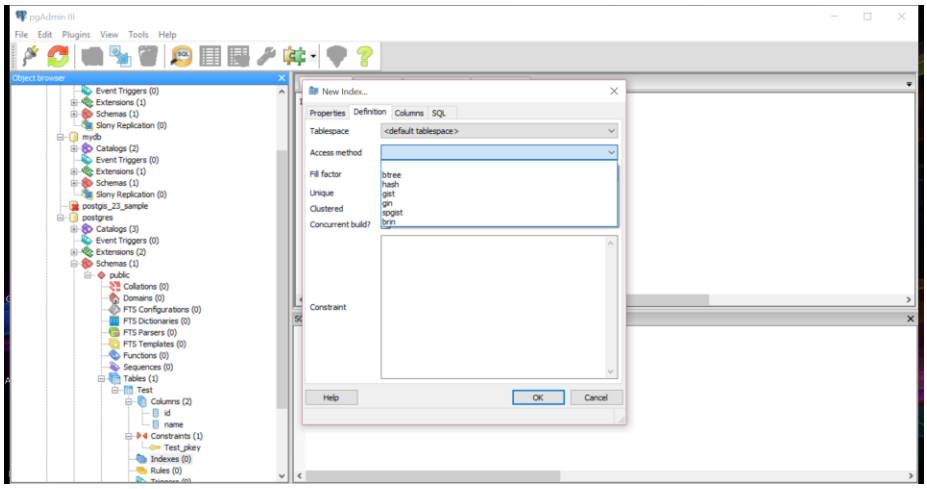
Name the index as “test_id_indx”

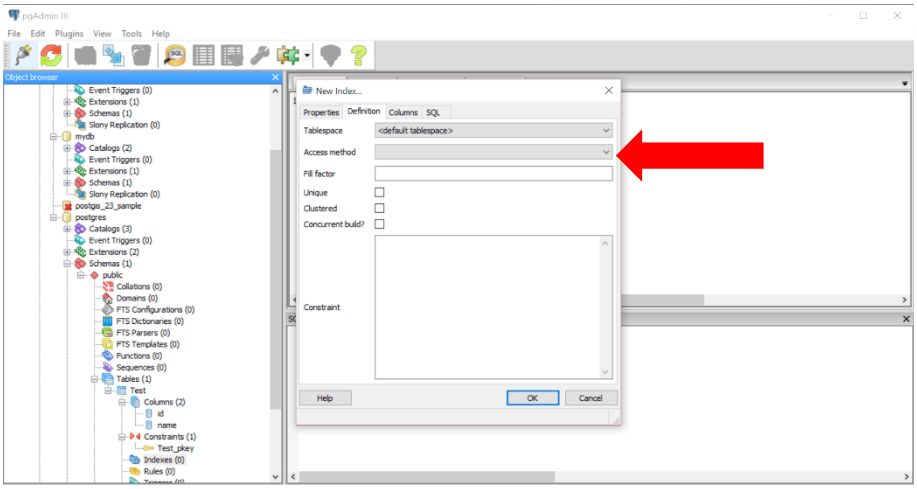


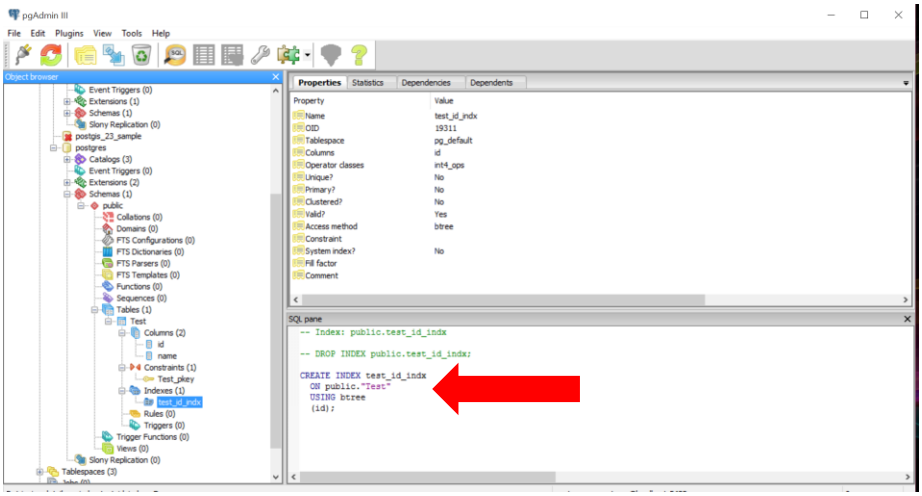
Set the index on “id” column



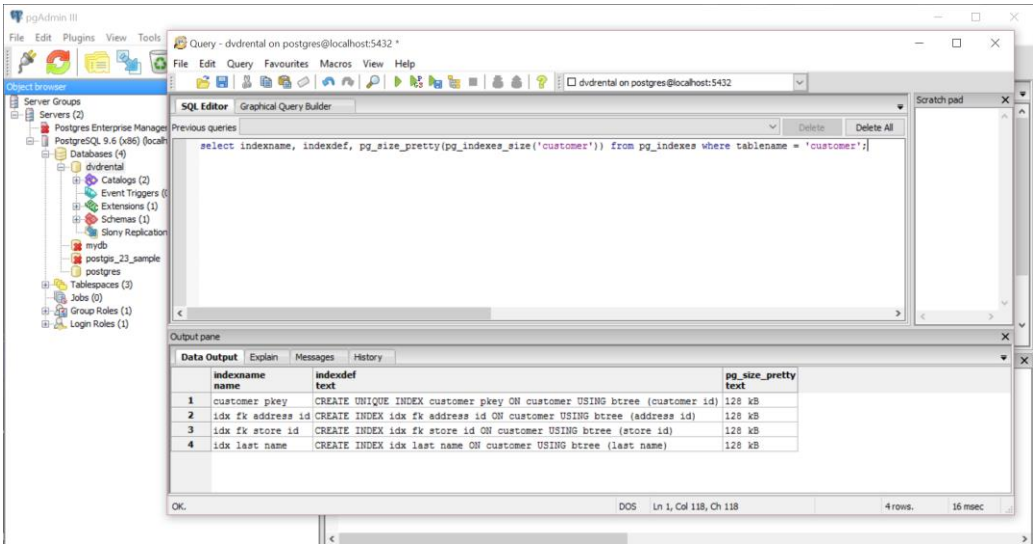
You can choose the access method, which represents the type of the index whether a btree or hash…. However, if you leave that field empty, the index will be automatically created using btree.







To get information on indexes created on a particular table, select “dvdrental” database and try this query: “select indexname, indexdef, pg_size_pretty(pg_indexes_size('customer')) from pg_indexes where tablename = 'customer';”


pgAdmin III
File Edit Plugins View Tools
Object browser
Server Groups
Servers (2)
Postgres Enterprise Manager
PostgreSQL 9.6 (x86) (local
Databases (4)
dvdrental
Catalogs (2)
Event Triggers
Extensions (1)
Schemas (1)
Slony Replication
mydb
postgis_23_sample
postgres
Tablespaces (3)
Jobs (0)
Group Roles (1)
Login Roles (1)
Query - dvdrental on postgres@localhost:5432 *
File Edit Query Favourites Macros View Help
dvdrental on postgres@localhost:5432
SQL Editor
Graphical Query Builder
Previous queries
Delete
Delete All
select indexname, indexdef, pg_size_pretty(pg_indexes_size('customer')) from pg_indexes where tablename = 'customer';
Scratch pad
Output pane
Data Output
Explain
Messages
History
indexname name
indexdef text
pg_size_pretty text
1 customer_pkey CREATE UNIQUE INDEX customer_pkey ON customer USING btree (customer_id) 128 kB
2 idx_fk_address_id CREATE INDEX idx_fk_address_id ON customer USING btree (address_id) 128 kB
3 idx_fk_store_id CREATE INDEX idx_fk_store_id ON customer USING btree (store_id) 128 kB
4 idx_last_name CREATE INDEX idx_last_name ON customer USING btree (last_name) 128 kB
OK.
DOS
Ln 1, Col 118, Ch 118
4 rows.
16 msec