Information Schema

The information schema consists of **a set of views** that contain information about the objects defined in the **current database**. The information schema is **defined in the SQL standard** and can therefore be expected to be **portable and remain stable** — unlike the **system catalogs**, which are specific to **PostgreSQL** and are modeled after implementation concerns. The information schema views **do not**, however, **contain information about PostgreSQL-specific features**; to inquire about those you need to query the system catalogs or other PostgreSQL-specific views.

The information schema itself is a schema named `information_schema`. This schema **automatically exists in all databases**. The owner of this schema is the initial database user in the cluster, and that user naturally has all the privileges on this schema, including the ability to drop it (but the space savings achieved by that are minuscule).

By default, the information schema is **not in the schema search path, so you need to access all objects in it through qualified names**. Since the names of some of the objects in the information schema are generic names that might occur in user applications, you should be careful if you want to put the information schema in the path.

https://www.postgresql.org/docs/13/information-schema.html

Five types

The columns of the information schema views use special data types that are defined in the information schema. These are defined as simple domains over ordinary built-in types. You should not use these types for work outside the information schema, but your applications must be prepared for them if they select from the information schema.

These types are:

- `cardinal_number`

  A nonnegative integer.

- `character_data`

  A character string (without specific maximum length).

- `sql_identifier`

  A character string. This type is used for SQL identifiers, the type `character_data` is used for any other kind of text data.

- `time_stamp`

  A domain over the type `timestamp with time zone`

- `yes_or_no`

  A character string domain that contains either `YES` or `NO`. This is used to represent Boolean (true/false) data in the information schema. (The information schema was invented before the type `boolean` was added to the SQL standard, so this convention is necessary to keep the information schema backward compatible.)

Every column in the information schema has one of these five types.

Experiment 1: Show tables and views

```
select * from information_schema.tables where table_schema =
'information_schema';
```

Tips: The information_schema.tables contains all tables and views defined in the current database. You can use information_schema.views to see only views.

Experiment 2: Show all columns in table

```
select *
from information_schema.columns where table_name = 'movies';
```

Experiment 3: Show all schema

```
select *
from information_schema.schemata;
```

Tips: The current database that the current user has access to

Experiment 4: Show all routines (function+procedure) in current database

```
select *
from information_schema.routines;
```

Experiment 5: Show all triggers in the current database

```
select *
from information_schema.triggers;
```

System Catalogs

The system catalogs are the place where a relational database management system stores schema metadata, such as information about tables and columns, and internal bookkeeping information. PostgreSQL's system catalogs are regular tables. You can drop and recreate the tables, add columns, insert and update values, and severely mess up your system that way. Normally, one should not change the system catalogs by hand, there are normally SQL commands to do that. (For example, `CREATE DATABASE` inserts a row into the `pg_database` catalog — and actually creates the database on disk.) There are some exceptions for particularly

esoteric operations, but many of those have been made available as SQL commands over time, and so the need for direct manipulation of the system catalogs is ever decreasing.

Experiment 6: Show tables and views in pg_catalog

```
select *
from information_schema.tables where table_schema = 'pg_catalog';
```

Tips: You can detect more detail by query on the table of pg_catalog

System Administration Functions

Table 9-70. Database Object Size Functions

Name	Return Type	Description
pg_column_size(any)	int	Number of bytes used to store a particular value (possibly compressed)
pg_database_size(oid)	bigint	Disk space used by the database with the specified OID
pg_database_size(name)	bigint	Disk space used by the database with the specified name
pg_indexes_size(regclass)	bigint	Total disk space used by indexes attached to the specified table
pg_relation_size(relation regclass, fork text)	bigint	Disk space used by the specified fork ('main','fsm','vm', or 'init') of the specified table or index
pg_relation_size(relation regclass)	bigint	Shorthand for pg_relation_size(..., 'main')
pg_size_pretty(bigint)	text	Converts a size in bytes expressed as a 64-bit integer into a human-readable format with size units
pg_size_pretty(numeric)	text	Converts a size in bytes expressed as a numeric value into a human-readable format with size units
pg_table_size(regclass)	bigint	Disk space used by the specified table, excluding indexes (but including TOAST, free space map, and visibility map)
pg_tablespace_size(oid)	bigint	Disk space used by the tablespace with the specified OID
pg_tablespace_size(name)	bigint	Disk space used by the tablespace with the specified name
pg_total_relation_size(regclass)	bigint	Total disk space used by the specified table, including all indexes and TOAST data

https://www.postgresql.org/docs/9.4/functions-admin.html

Experiment 7: Show database_size

```
select oid from pg_database where datname = 'cs307';
select pg_database_size(16394);
select pg_database_size('cs307');
```

Tips: The size of bigint data type in PostgreSQL is 8 bytes and range of bigint data type is -9223372036854775808 to 9223372036854775807

Experiment 8: Show tablespace_size

```
select pg_tablespace_size('pg_default')/1024/1024 as "size in M";
select pg_size_pretty(pg_tablespace_size('pg_default'));
```

Experiment 9: table size vs index size

```
select pg_size_pretty(pg_table_size('movies'));
select pg_size_pretty(pg_indexes_size('movies'));
select pg_size_pretty(pg_total_relation_size('movies'));
```

Experiment 10: column_size

```
select pg_column_size('hello world');
select pg_column_size('');
```