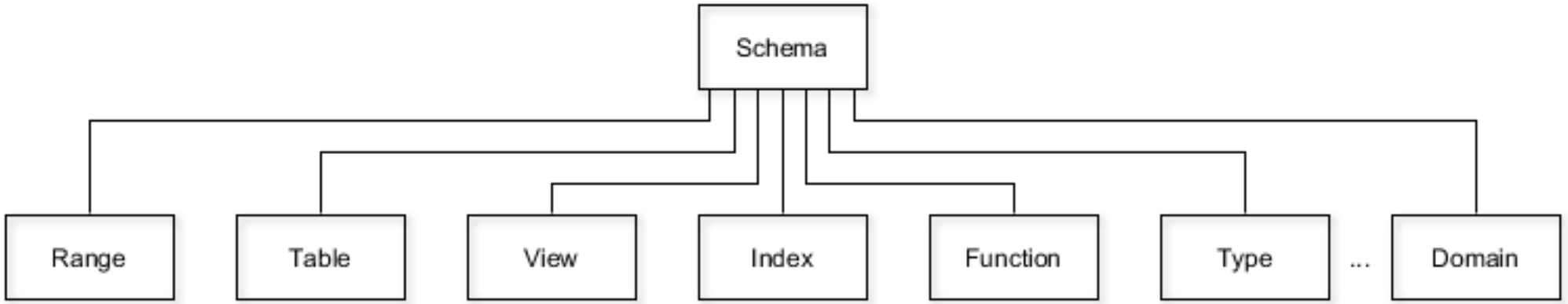
PostgreSQL Database Components

- A PostgreSQL database could be considered a container for database schemas; the database must contain at least one schema.
- A database schema is used to organize the database objects in a manner similar to namespaces in high-level programming languages.

Schemas

- Object names can be reused in different schemas without conflict.
- The schema contains all the database-named objects, including tables, views, functions, aggregates, indexes, sequences, triggers, data types, domains, and ranges.



- By default, there is a schema called *public* in the template databases.
- That means all the newly-created databases also contain this schema.
- All users, by default, can access this schema implicitly.
- Again, this is inherited from the template databases.
- Allowing this access pattern simulates a situation where the server is not schema-aware.
- This is useful in small companies where there's no need to have complex security.
- Also, this enables a smooth transition from non-schema-aware databases.

- In a multi-user and multi-database environment setup, remember to revoke the ability for all users to create objects in the public schema.
- This is done with the `REVOKE CREATE ON SCHEMA public FROM PUBLIC;` command in the newly-created database, or in the `template1` database.

- When a user wants to access a certain object, they can specify the schema name and the object name separated by a period (.).

```
postgres=# SELECT * FROM pg_catalog.pg_database;
```

```
  datname  | datdba | encoding | datcollate  |  datctype   | datistemplate | datallowconn | datconnlimit | datlastsyso
id | datfrozenxid | datminmxid | dattablespace |                datacl
-----------+--------+----------+-------------+-------------+---------------+--------------+--------------+------------
---+--------------+------------+---------------+------------------------------------
 postgres  |     10 |        6 | en_US.UTF-8 | en_US.UTF-8 | f             | t            |           -1 |         131
27 |          561 |          1 |          1663 |
 template1 |     10 |        6 | en_US.UTF-8 | en_US.UTF-8 | t             | t            |           -1 |         131
27 |          561 |          1 |          1663 | {=c/postgres,postgres=CTc/postgres}
 template0 |     10 |        6 | en_US.UTF-8 | en_US.UTF-8 | t             | f            |           -1 |         131
27 |          561 |          1 |          1663 | {=c/postgres,postgres=CTc/postgres}
(3 rows)

(END)
```

```
postgres=# TABLE pg_catalog.pg_database;
```

- Qualified database-object names are sometimes tedious to write, so most developers prefer to use the unqualified object name, which is composed of the object name without the schema.
- PostgreSQL provides a `search_path` setting that's similar to the using directive in the C++ language.
- `search_path` is composed of schemas that are used by the server to search for the object.
- The default `search_path` is `$user`, `public`.
- If there's a schema with the same name as the user, it will be used first to search for objects or to create new objects.
- If the object isn't found in the schemas specified in the `search_path`, an error will be thrown.

```
postgres=# SHOW search_path;
   search_path
-----------------
 "$user", public
(1 row)
```

- Generally speaking, you shouldn't rely on implicit conventions and information such as `SELECT *`, `NATURAL JOIN`, or `JOIN USING`.
- In this context, it's recommended to use fully-qualified names.

Schema Usages

- Schemas are used for the following reasons:
  - **Control authorization**: In a multi-user database environment, you can use schemas to group objects based on roles.
  - **Organize database objects**: You can organize the database objects in groups based on business logic. For example, historical and auditing data could be logically grouped and organized in a specific schema.
  - **Maintain third-party SQL code**: The extensions available in the contribution package can be used with several applications. Maintaining these extensions in separate schemas enables the developer to reuse these extensions and to update them easily.

- `CREATE SCHEMA` enters a new schema into the current database.
- For more information about the syntax of the `CREATE SCHEMA` command, you can use the `psql \h` meta-command.

```
postgres=# \h create schema
Command:     CREATE SCHEMA
Description: define a new schema
Syntax:
CREATE SCHEMA schema_name [ AUTHORIZATION role_specification ] [ schema_element [ ... ] ]
CREATE SCHEMA AUTHORIZATION role_specification [ schema_element [ ... ] ]
CREATE SCHEMA IF NOT EXISTS schema_name [ AUTHORIZATION role_specification ]
CREATE SCHEMA IF NOT EXISTS AUTHORIZATION role_specification

where role_specification can be:

    user_name
  | CURRENT_USER
  | SESSION_USER
```

Tables

- The PostgreSQL tables are used internally to model views and sequences.
- In PostgreSQL, tables can be of different types:
  - **Ordinary table**: The table lifespan starts with table creation and ends with table dropping.
  - **Temporary table**: The table lifespan is the user session. This is often used with procedural languages to model business logic.
  - **Unlogged table**: Unlogged tables are faster than ordinary tables, because data isn't written into the WAL files. Unlogged tables aren't crash-safe. Also, since streaming replication is based on shipping the WAL files, unlogged tables can't be replicated to the standby node.
  - **Child table**: A child table is an ordinary table that inherits one or more tables. The inheritance is often used with constraint exclusion to physically partition the data on the hard disk and to improve performance by retrieving a subset of data that has a certain value.

- The `CREATE TABLE` syntax is quite long; it's full syntax can be found at http://www.postgresql.org/docs/current/static/sql-createtable.html.
- The `CREATE TABLE SQL` command normally requires the following input:
  - Table name of the created table.
  - The table type.
  - The table's storage parameters. These parameters are used to control the table's storage allocation and several other administrative tasks.
  - The table columns, including the data type, default values, and constraint.
  - The cloned table name and the options to clone the table.

- To create a table, you can write a query as simple as the following:

```
postgres=# CREATE TEMP TABLE temp  AS SELECT 1 AS one;
SELECT 1
postgres=# select * from temp;
 one
-----
   1
(1 row)
```