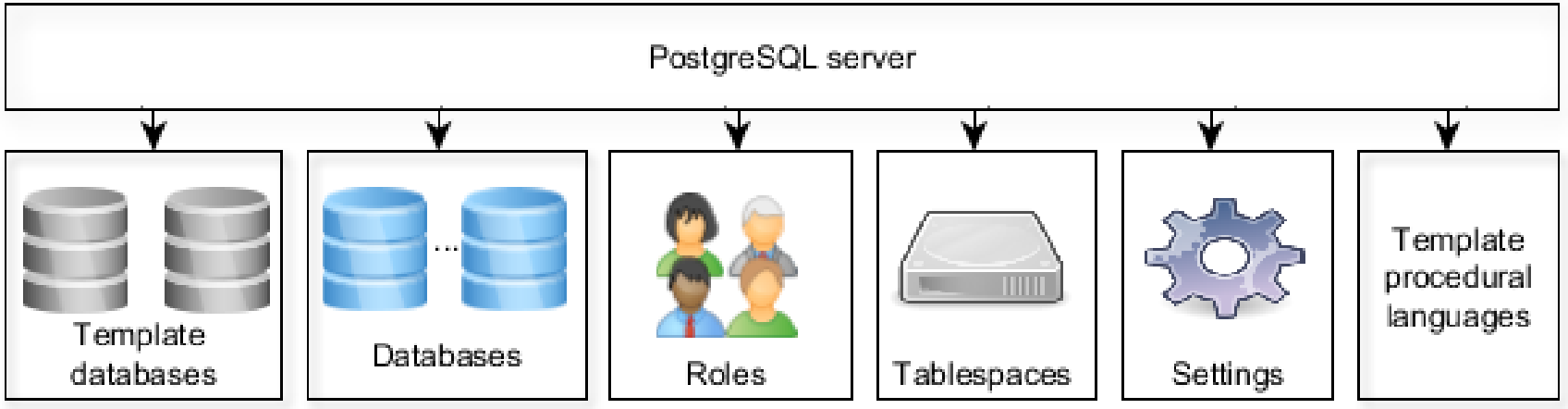
PostgreSQL Object Hierarchy

- Understanding the organization of the PostgreSQL database's logical objects helps with the understanding of object relations and interactions.
- PostgreSQL databases, roles, tablespaces, settings, and template languages have the same level of hierarchy.



Template Databases

- By default, when a database is created, it's cloned from a template database called `template1`.
- The template database contains a set of tables, views, and functions that are used to model the relation between the user-defined database objects.
- These tables, views, and functions are part of the system catalog schema, called `pg_catalog`.

- The PostgreSQL server has two template databases

- `template1`:
  - The default database to be cloned.
  - It can be modified to allow global modification to all the newly-created databases.
  - For example, if someone intends to use a certain extension in all the databases, they can install this extension in the `template1` database.
  - Installing an extension in `template1` won't be cascaded to the already existing databases, but it will affect the databases that will be created after this installation.

- `template0`: A safeguard or version database that has several purposes:
  - If `template1` is corrupted by a user, this can be used to fix `template1`.
  - It's handy in restoring a database dump. When a developer dumps a database, all the extensions are also dumped. If the extension is already installed in `template1`, this will lead to a collision, because the newly-created database already contains the extensions. Unlike `template1`, `template0` doesn't contain encoding-specific or locale-specific data.

User Databases

- You can have as many databases as you want in a database cluster.
- A client that connects to the PostgreSQL server can access only the data in a single database, which is specified in the connection string.
- That means data isn't shared between the databases unless the PostgreSQL foreign data wrapper or DB link extensions are used.

- Every database in the database cluster has an owner and a set of associated permissions to control the actions allowed for a particular role.
- The privileges on PostgreSQL objects, which include databases, views, tables, and sequences, are represented in the `psql` client as follows:
  - <user>=<privileges>/granted by
- If the user part of the privileges isn't present, this means that the privileges are applied to PostgreSQL's special `PUBLIC` role.

- The psql client tool's `\l` meta-command is used to list all the databases in the database cluster with the associated attributes:

```
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)
```

- The database-access privileges are the following:
  - `CREATE (-C)`: The create access privilege allows the specified role to create new schemas in the database.
  - `CONNECT (-c)`: When a role tries to connect to a database, the connect permissions are checked.
  - `TEMPORARY (-T)`: The temporary access privilege allows the specified role to create temporary tables. Temporary tables are very similar to tables, but they aren't persistent, and they're destroyed after the user session is terminated.

- In the preceding example, the `postgres` database has no explicit privileges assigned.
- Also notice that the `PUBLIC` role is allowed to connect to the `template1` database by default.

- Encoding allows you to store text in a variety of character sets, including one-byte character sets, such as SQL_ASCII, or multiple-byte characters sets, such as UTF-8.
- PostgreSQL supports a rich set of character encodings.
- For the full list of character encodings, visit http://www.postgresql.org/docs/current/static/multibyte.html

- In addition to these attributes, PostgreSQL has several other attributes for various purposes, including the following:
  - **Maintenance**: The `datfrozenxid` attribute is used by autovacuum operations.
  - **Storage management**: The `dattablespace` attribute is used to determine to which tablespace the database belongs.
  - **Concurrency**: The `datconnlimit` attribute is used to determine the number of concurrent connections (-1 means no limits).
  - **Protection**: The `datallowconn` attribute disables the connection to a database. This is used mainly to protect `template0` from being altered.

```
postgres=# select datname, datfrozenxid, dattablespace, datconnlimit, datallowconn from pg_database;
  datname  | datfrozenxid | dattablespace | datconnlimit | datallowconn
-----------+--------------+---------------+--------------+--------------
 postgres  |          561 |          1663 |           -1 | t
 template1 |          561 |          1663 |           -1 | t
 template0 |          561 |          1663 |           -1 | f
(3 rows)
```

- The `psql` client tool's `\c` meta-command establishes a new connection to a database and closes the current one.
- It also accepts a connection string, such as a username and password.

```
postgres=# \c template0
FATAL:  database "template0" is not currently accepting connections
Previous connection kept
postgres=#
```

- `pg_catalog` tables are regular tables, thus you can use the `SELECT`, `UPDATE`, and `DELETE` operations to manipulate them.
- Doing so is not recommended, and needs the utmost attention.
- Manipulating catalog tables manually can lead to silent errors, database crashes, data integrity corruption, and unexpected behavior.

Roles

- Roles belong to the PostgreSQL server cluster and not to a certain database.
- A role can either be a database user or a database group.
- The role concept subsumes the concepts of users and groups in the old PostgreSQL versions.
- For compatibility reasons, with PostgreSQL version 8.1 and later, the `CREATE USER` and `CREATE GROUP` SQL commands are still supported.

- The roles have several attributes, which are as follows:
  - `SUPERUSER`: A superuser role can bypass all permission checks except the `LOGIN` attribute.
  - `LOGIN`: A role with the `LOGIN` attribute can be used by a client to connect to a database.
  - `CREATEDB`: A role with the create database attribute can create databases.
  - `CREATEROLE`: A role with this feature enabled can create, delete, and alter other roles.
  - `REPLICATION`: A role with this attribute can be used to stream replication.

- `PASSWORD`: The `PASSWORD` role can be used with the `md5` and `scram-sha-256` authentication method. The password expiration can be controlled by specifying the validity period. Note that this password differs from the OS password. In newer versions of PostgreSQL server—mainly 10 and 11—it's recommended to use `scram-sha-256`, instead of `md5`, because it's more secure.
- `CONNECTION LIMIT`: This specifies the number of concurrent connections that the user can initiate. Connection creation consumes hardware resources; thus, it's recommended to use connection pooling tools such as **Pgpool-II**, **Yandex Odyssey**, **PgBouncer**, or some APIs, such as **Apache DBCP** or **c3p0**.
- `INHERIT`: If specified, the role will inherit the privileges assigned to the roles that it's a member of. If not specified, `INHERIT` is the default.
- `BYPASSRLS`: If specified, this role can bypass **row-level security (RLS)**.

- During the installation of PostgreSQL, the `postgres` superuser role is created.
- `CREATE USER` is equivalent to `CREATE ROLE` with the `LOGIN` option, and `CREATE GROUP` is equivalent to `CREATE ROLE` with the `NOLOGIN` option.

- A role can be a member of another role to simplify accessing and managing database permissions; for example, you can create a role with no login, also known as a group, and grant it permission to access the database objects.
- If a new role needs to access the same database objects with the same permissions as the group, the new role could be assigned a membership to this group.
- This is achieved by the `GRANT` and `REVOKE` SQL commands.

Tablespaces

- A **tablespace** is a defined storage location for a database or database objects.
- Tablespaces are used by administrators to achieve the following:
  - **Maintenance**: If the hard disk partition runs out of space where the database cluster is created and can't be extended, a tablespace on another partition can be created to solve this problem by moving the data to another location.
  - **Optimization**: Heavily-accessed data could be stored in fast media, such as a **solid-state drive (SSD)**. At the same time, tables that are not performance-critical could be stored on a slow disk.

- The SQL statement to create a tablespace is `CREATE TABLESPACE`.

Template Procedural Languages

- Template procedural languages are used to register a new language in a convenient way.
- There are two ways to create a programming language:
  - The first is by specifying only the name of the programming language. In this method, PostgreSQL consults the programming language template and determines the parameters.
  - The second way is to specify the name as well as the parameters.
- The SQL command to create a language is `CREATE LANGUAGE`.

- In PostgreSQL versions 9.1 and later, `CREATE EXTENSION` can be used to install a programming language.

- The template procedural languages are maintained in the `pg_pltemplate` catalog table.
- This table might be decommissioned in favor of keeping the procedural language information in their installation scripts.

Settings

- The PostgreSQL settings control different aspects of the PostgreSQL server, including replication, write-ahead logs, resource consumption, query planning, logging, authentication, statistic collection, garbage collection, client connections, lock management, error handling, and debug options.

- The following SQL command shows the number of PostgreSQL settings.
- Note that this number might differ slightly between different installations as well as customized settings:

```
postgres=# SELECT count(*) FROM pg_settings;
 count
-------
   289
(1 row)
```

- The parameters can be as follows:
  - **Boolean**: `0`, `1`, `true`, `false`, `on`, `off`, or any case-insensitive form of the previous values. The `ENABLE_SEQSCAN` setting falls into this category.
  - **Integer**: An integer might specify a memory or time value; there is an implicit unit for each setting, such as seconds or minutes. In order to avoid confusion, PostgreSQL allows units to be specified. For example, you could specify 128 MB as a `shared_buffers` setting value.
  - **Enum**: These are predefined values, such as `ERROR` and `WARNING`.
  - **Floating point**: `cpu_operator_cost` has a floating point domain. `cpu_operator_cost` is used to optimize the PostgreSQL execution plans.
  - **String**: A string might be used to specify the file location on a hard disk, such as the location of the authentication file.

- The setting context determines how to change a setting's value and when the change can take effect.
- The setting contexts are as follows:
  - `internal`: The setting cannot be changed directly. You might need to recompile the server source code or initialize the database cluster to change this. For example, the length of PostgreSQL identifiers is 63 characters.
  - `postmaster`: Changing a setting value requires restarting the server. Values for these settings are typically stored in the PostgreSQL `postgresql.conf` file.
  - `sighup`: No server restart is required. The setting change can be made by amending the `postgresql.conf` file, followed by sending a SIGHUP signal to the PostgreSQL server process.
  - `backend`: No server restart is required. They can also be set for a particular session.
  - `superuser`: Only a superuser can change this setting. This setting can be set in `postgresql.conf` or via the `SET` command.
  - `user`: This is similar to superuser, and is typically used to change the session-local values.

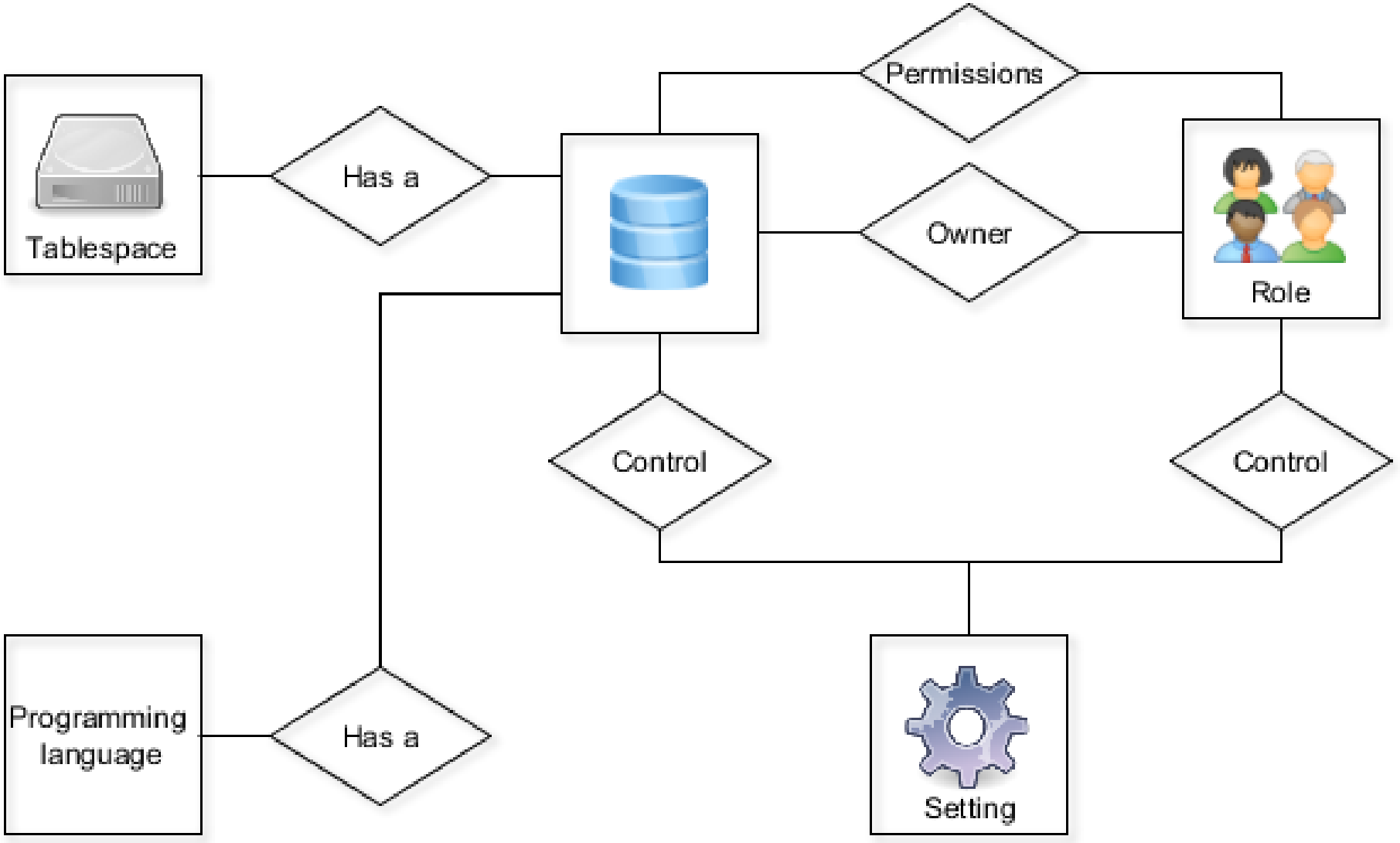
- PostgreSQL provides the `SET` and `SHOW` commands to change and inspect the value of a setting parameter, respectively.
- These commands are used to change the setting parameters in the superuser and user context.
- Typically, changing the value of a setting parameter in the `postgresql.conf` file makes the effect global.

- PostgreSQL's `ALTER SYSTEM` command and `postgresql.auto.conf` provide a convenient way to change the configuration of the whole database cluster without editing the `postgresql.conf` file manually.
- It uses `postgresql.auto.conf` to overwrite the configuration parameters in `postgresql.conf`,
  so `postgresql.auto.conf` has a higher priority than `postgresql.conf`.

- PostgreSQL also provides the `pg_reload_conf()` function, which is equivalent to sending the SIGHUP signal to the PostgreSQL process.
- In general, it's preferable to use `pg_reload_conf()` or reload the configuration settings via the `init` script because it's safer than the SIGHUP kill signal due to human error.

PostgreSQL High-Level Object Interaction

- To sum up, a PostgreSQL server can contain many databases, programming languages, roles, and tablespaces.
- Each database has an owner and a default tablespace; a role can be granted permission to access or can own several databases.
- The settings can be used to control the behavior of the PostgreSQL server on several levels, such as the database and the session.
- Finally, a database can use several programming languages


Permissions
Tablespace
Has a
Owner
Role
Control
Control
Programming language
Has a
Setting