Relational Databases & SQL

Introduction

Relational databases management systems (RDBMS) are used for storing tabular (relational) data in such a way that it is easy to store and query (retrieve or "select"). A well-optimized schema and query makes it possible to query data from tables that contain millions or even billions of records. Some of the major relational databases used today include Microsoft SQL Server, Oracle, PostgreSQL, MySQL, MariaDB, and SQLite. Some of the major benefits of using a relational database include:

- Relatively simple performance optimizations
- Well-defined behavior for high-concurrency usage
- Data integrity constraints
- Transactions and ACID compliance

Relational databases use some variation of SQL to manipulate database schemas (data definitions) and query data. Simple queries are often portable and usable across database vendors and versions, but not always. It's important to reference the particular database's documentation for exact usage. It's also worth noting that a query that performs quickly on one database may perform poorly on another. This is due to differences in the internal workings of a database and its configuration.

For all database work in this training, PostgreSQL will be used.

NoSQL

While many modern RDBMS offer replication and some form of sharding, which provide data redundancy and performance scaling, historically this has been hard to do and not always reliable. NoSQL refers to the name of various technologies and a movement away from traditional RDBMS to systems that do not use SQL or traditional storage mechanisms which scale very easily. Google BigTable, Redis, CouchDB, Cassandra, Memcache, and HBase are some of those systems. Aside from not using tabular data for storage, these systems typically do not support transactions or ACID compliance and sometimes embrace what's called "eventual consistency": this means that data replicated across servers is not always the same. However, depending on usage these systems can be easier to scale for both data storage, high-performance usage, and data redundancy.

NoSQL technologies will not be covered, but there are many popular and well-supported systems that are worth learning.

PostgreSQL Introduction

Execute the following to install PostgreSQL and its PHP extension: `sudo apt-get install postgresql php5-pgsql`

PostgreSQL is a mature and very well-supported RDBMS with many extensions and features that make it a joy to work with. It's very fast and capable of very complex queries that may not perform well on other systems. MySQL and variants may be slightly more popular than PostgreSQL, but PostgreSQL offers many more features with less inconsistency and surprises in the behavior which make it a good piece of software to learn about SQL and basic database administration.

PG documentation can be found here: http://www.postgresql.org/ but the site's search feature isn't always useful. Try using Google and including "postgresql" among the search terms to find specific documentation.

When installed, PG typically creates a "postgres" system user that is a superuser/admin. This user can be used for creating new databases, database users, server configuration, etc. The installation process should also create a database named "postgres." This user and database should be used for storing and accessing data because it is a superuser. Create a separate user and database:

- Change to the postgres user: `sudo -i -u postgres`
- Create a new PG user: `createuser trainee`
- Create a new DB owned by the trainee user: `createdb -O trainee trainee`
- Verify the above by using the CLI client to connect to the database. Press Ctrl-D to exit. `psql -U trainee trainee`
- Set a password for the trainee user:
    - `psql -U postgres`
    - `ALTER USER trainee PASSWORD 'somethingSecure';`
    - Press Ctrl-D to exit
- By default, Postgres is only configured to allow local connections when a user's system name is the same. This should be modified to allow network connections when using a password. Add "`host all all password`" to `/etc/postgresql/9.*/main/pg_hba.conf`. It should look something like this:

```
86 # TYPE  DATABASE        USER            ADDRESS                 METHOD
87
88 # "local" is for Unix domain socket connections only
89 local   all             postgres                                peer
90 local   all             all                                     peer
91
92 # IPv4 local connections:
93 host    all             all             127.0.0.1/32            md5
94 # IPv6 local connections:
95 host    all             all             ::1/128                 md5
96 host    all             all                                     password
97 #host    all     all               password
98 # Allow replication connections from localhost, by a user with the
99 # replication privilege.
100 #local   replication     postgres                                peer
```

- Log out of the postgres user by using the `exit` command or press Ctrl-D
- Reload the PostgreSQL configuration: `sudo /etc/init.d/postgresql reload`
- Verify that you are able to connect using the new user and password: `psql -U trainee -h 127.0.0.1`

psql is the CLI client that can be used for both schema changes and data changes. It has many features that make it very fast and easy to explore or manipulate a database, but is a poor tool for writing any kind of large query. Issue "\?" to get a list of commands and learn how to list tables, schemas, and show details about a table. pgAdmin is a helpful GUI tool that offers many of the features of `psql` as well as UI tools that reduce the need for SQL knowledge, but it is sometimes buggy and slow. Use `sudo apt-get install pgadmin3` to install pgAdmin on an Ubuntu system or visit one of the following links for your OS:

- Windows: http://www.pgadmin.org/download/windows.php
- OSX: http://www.pgadmin.org/download/macosx.php

Core Concepts

Learning SQL is the first step to working with relational databases. W3Schools has a good introduction: http://www.w3schools.com/sql/, but the key is to practice. Few programmers take the time to build a solid understanding of SQL, but the investment pays off quickly when working with non-trivial data. It is not uncommon to work with long queries that cover dozens of tables, so SQL knowledge is important.

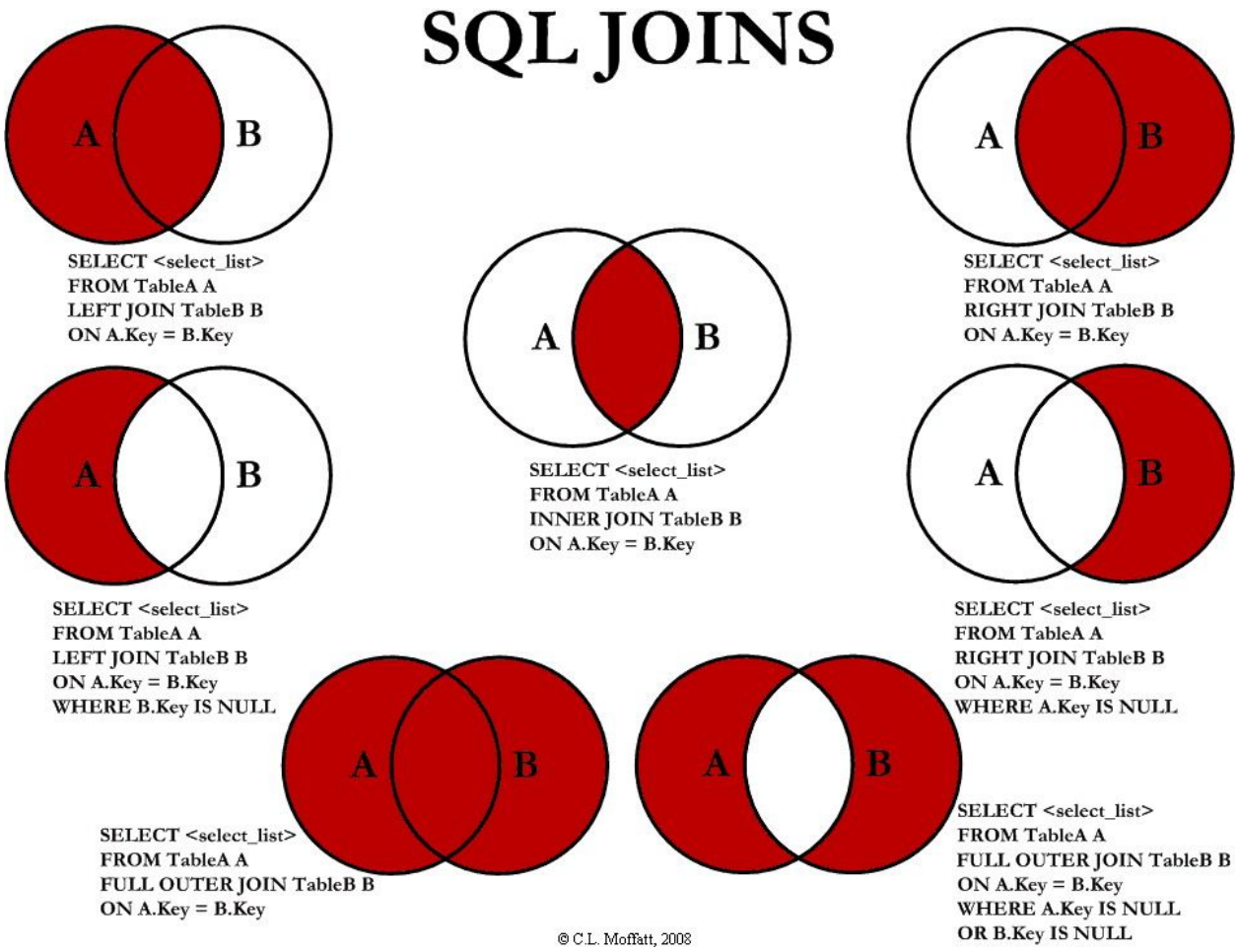
Spend time learning how to use each of these topics and how to use them in PostgreSQL:

- Data types
- Primary keys
- Transactions and ACID compliance
- Indexes
- CHECK constraints
- Foreign keys and data cascading
- Triggers and Rules

A major reason for using a relational database is the ability to enforce data integrity. When designing a database schema, it is important to decide and enforce exactly what data should be permitted. This will prevent bad data from being entered which can later cause bugs in client software. Here are some examples:

- If a field is required, make it NOT NULL
- If you're storing a birth date, use a date type
- A first name is probably not longer than 50 characters
- Salary amounts should be greater than $0. Integer or decimal? How many decimals?
- If one field's value determines the possible values for another field, use a CHECK constraint

SQL Table Joins



References: [Visual Representation of SQL Joins]

Normalization

Database normalization refers to data structuring to reduce duplication and redundancy and prevent various kinds of "anomalies." There are 6 levels of normalization. This video provides a basic introduction: https://www.youtube.com/watch?v=y_MDbbqQIUU. Please spend time researching this topic.

Not all database schemas require extreme levels of normalization. Every database schema should be designed according to expected usage. Normalization is extremely important for transactional databases (OLTP) to simplify data creation. However, for non-transactional databases (like data warehouses), it can be very helpful to have duplicate data if it simplifies queries and improves performance. Make decisions based on business requirements and expected use-cases.