Databases

For projects with databases it is important to provide a db/migrations folder including all the steps used to setup the database require for the program to work. Instructions for the database manager should also be included in the installation part of the readme.

PostgreSQL

* createdb - on cmd to create base database in username
* psql - to enter sql shell
* \q - to quit shell
* \l - to list all databases
* \c <database-name> - to connect to databse

Once connecting to database, to insert data:

* INSERT INTO <table-name>(<column1>, <column2>, …)

VALUES

(<value1>, <value2>,..),

(<value1>, <value2>,..);

Delete data:

* DELETE FROM <table> WHERE <column> = <val>;

Update data:

* UPDATE <table> SET <column> = <val> WHERE <column> = <val>;

Querys

A query is when a specific set of data is requested from a database and the database uses the query to select, format and return it matching the query.

Selecting Data: SELECT <columns> FROM "<table>" WHERE …;

* WHERE <column> IN (‘<value>’); - Select data which matches value in column
* WHERE <column> BETWEEN <value> AND <value>; - data between values
* WHERE <column> > <value>; - mathematical operators can be used in conditions
* WHERE <column> LIKE(‘value%’); - Looks for values inc value at beginning
* WHERE <column> NOT LIKE(‘%a%’); - Looks for values which don’t include ‘a’

Conditions can be linked with AND and OR. XOR allows for exclusive or, so value must be one or other but not both. XOR will separate conditions so multiple conditions can be used eitherside without brackets.

To select NULL values the phrase IS NULL or IS NOT NULL can be used accordingly.

Notes:

* NOT can be put before LIKE and IN.
* Quotes can be escaped using standard notation \’

Editing results:

* columns can be modied by mathematical operators to change their output on selection, for example dividing one column by another:
  + SELECT name, gdp/population FROM world WHERE population > 200000000;
* Results can be ordered using the ORDER BY <column>, <column> command
  + SELECT winner FROM nobel WHERE winner LIKE(‘Sir%’) ORDER BY yr DESC, winner ASC; This will order year descending first then each year by winner name ascending.
* Since IN(<column>) returns 0 or 1 as a result for each row, it can be used to order items. Putting as an ORDER BY ASC will order by results not matching the IN clause first then results which match.
* GROUP BY with SUM and COUNT applies the operations to GROUP BY columns sharing values. e.g. SELECT continent, SUM(gdp) FROM world GROUP BY continent will only give one value for each type of continent, summing population or GDP (whatever is selected to sum by). If GROUP BY is used all columns must be aggregated by SUM or COUNT else an error will occur.
* HAVING can be used with GROUP by to filter the GROUPED rows, since WHERE will only filter before the GROUP BY clause. e.g. SELECT continent, SUM(population) FROM world GROUP BY continent HAVING SUM(population)>500000000

SQL Operations:

* = 🡸 equal too
* <> 🡸 not equal too
* <, <=, >, >= 🡸 standard operators
* BETWEEN <val> and <val2> 🡸 inclusive between operator
* General operations such as \* & /
* ROUND(‘<column>’, decimal place) 🡸 Decimal place can be negative
* LENGTH(‘<column>’) 🡸 returns length, can be used to compare or output
* LEFT(‘<column>’, num) 🡸 returns number of letters from left of column
* COUNT(<column>) 🡸 returns number of rows matched
* DISTINCT(<column) 🡸 returns only the unique values selected
* SUM(<column>) 🡸 sums all values in rows matched
* MAX(<column>) 🡸returns max value of rows matched
* AVG(<column>) 🡸returns average value of rows matched
* ABS(<column>) 🡸 only positive value

Other functions: <https://sqlzoo.net/wiki/FUNCTIONS>

Using SELECT in SELECT - Selects can be used inside of selects to create more complex querys from a large database. This allows a table to be queryed then more specific results from the returned table to also be queried refining the results. It also allows tables to be compared, since diffferent tables can be queried in different select methods. If the subquery result contains more than one result, it is imporant to use the IN selector to compare results otherwise the main query will fail, since it is comparing one row to more than one sub-result. e.g: SELECT name, continent FROM world WHERE continent IN (SELECT continent FROM world WHERE name='Brazil' OR name='Mexico')

The JOIN function allows a many to one relationship to automatically related. It takes two table names and joins data dependent on the specified id columns given. e.g. SELECT \* FROM game JOIN goal ON (id=matchid); This matches values from the game uniquie id to the goals listed with each game unique id (in the goal ‘matchid’ column). Three way joins can be created using an additional JOIN with a table so long as its unique key exisits as a forgin key in the previously joined table. The extra data from the third join will be added as a new column in the result. LEFT JOIN can be used to include an empty row if the second table doesn’t include the id of the primary joining table. e.g. SELECT games.yr, city.country FROM games LEFT JOIN city ON (games.city = city.name); will leave blank rows if there is a games year but no city assigned.

Self joining can be done to a table which has columns with repeating data. To do this labels must be introduced to distinguish the instances of the same table to be joined. Labels are created in the FROM clause where the label is places after the table name. Then each column name uses a .label after it to be identified. e.g.

SELECT \* FROM route R1, route R2 WHERE R1.num=R2.num AND R1.company=R2.company;

SELECT \* FROM route R1, route R2 WHERE R1.stop=R2.stop; - will show where stops of different bus routes are the same, so it will contain some duplicate values but also show when two different routes stop in the same place, this could then be compared with time and see if there are any clashes.

CASE allows different values to be returned under different conditions, if there are no conditions matched and a ELSE clause isn’t given, null will be returned. e.g.

CASE WHEN condition1 THEN value1

WHEN condition2 THEN value2

ELSE def\_value

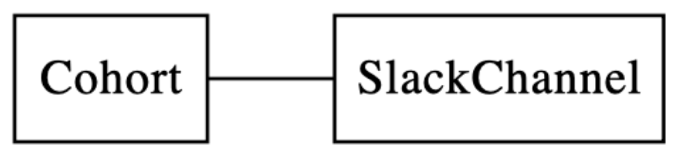
END;

<https://www.postgresqltutorial.com/>

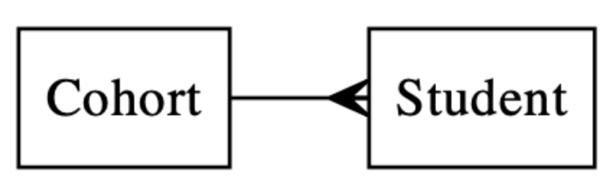
Entity Relationship Diagrams

To keep databases consistent it is important to diagram the relationships each entity will have with eachother. There is one item per row of a table each of which has a unique identifier key. When refering to items from one table to another, the item should be identified using the unique key, since any other value is liable to change.

One to one relationships - One cohort has one slack channel. One to one relationships mean one record in a table is associated with only one record in another table. Can be tabulated simply using the unique key of one in another table. Insert the forgein key of one into another.



One to many relationships - Cohort has many students. Can be tabulated by adding the unique key of the ‘one’ into the table for the ‘many’, this will be called the forgein key.



Many to many - Students are members of many slack channels and slack channels have many students. An issue here is if a student adds a slack channel a database would have to do two operations to add both the student to the slack channel and the channel to the student. To avoid this join tables are used where each unique key is related to eachother:



Connecting to PSQL Databse using Ruby

Using the ‘pg’ gem it is possible to conenct the a psql database directly from ruby code. Create a connection using the gem:

‘conn = PG::Connection.open(:dbname => ‘test’, :user => ENV['USER'], <:login, :password, etc…>)’

conn.copy\_data

How Databases Work

Each database has a Databse Management System (DBMS), like psql, which interacts and manages all stored data. There are four ways to interact with data:

CRUD - Create, Read, Update, Delete

There are two types of database:

* Relational - Tables made of columns and rows, with strict datatypes. Use the Structured Query Language (SQL). Patterns of data arrangment are called schema, and everything must be defined before any data can be added.
* Non-relational - NoSQL database which covers databases which aren’t strictly relational. Good for getting a database running quickly as no schema is required.

Most popular DBMS:

1. Oracle
2. MySQL
3. Microsoft SQL
4. PostgreSQL
5. mongoDB (NoSQL)