Module 2 - Session 2 - Data exploration

Working effectively with data

CivicDataLab

2021/08/11 (updated: 2021-08-12)

Exercise - Web Scraping + Data Exploration

- Link to NALSA dashboard
- Create a CSV file with variables available under the Victim Compensation Schemes table for these states:
  - Delhi
  - Maharashtra
  - Karnataka
  - West Bengal
  - Uttar Pradesh
- Create a chart to compare the yearly compensation numbers between these states
- Create a folder here and upload the dataset (including the chart)

Worksheet Link

Working with databases

Why to use a database ?

- Dealing with large datasets
- Platform agnostic
- Programming language agnostic
- Easy to share and maintain as compared to storing data as multiple data files

A relational database

civic data lab

- Data stored as tables
- Each row in the table is a record with a unique ID called the key (Primary Key).
- The columns of the table hold attributes of the data, and each record usually has a value for each attribute.
- Uses SQL (Structured Query Language) to query (*storing*, *manipulating*, *retrieving*) data

Database terminologies

1. **Schema** - *A database schema is the design of tables, columns, relations, and constraints that make up a logically distinct section of a database*.
2. **Key** - *A key is a database field whose purpose is to uniquely identify a record*. Type off keys:
    - **Candidate Key** - *The set of columns that can each uniquely identify a record and from which the primary key is chosen.*
    - **Primary Key** - *This key uniquely identifies a record in a table. It cannot be null.There can be only one Primary key in a table.*
    - **Foreign Key** - *The key linking a record to a record in another table. A table's foreign key must exist as the primary key of another table.*
3. **SQL** - *Structured Query Language, or SQL, is the most commonly used language to access data from a database*

Understanding keys

Table 1 | Candidate | Primary | Composite

Id	Name	Gender	City	Email	Dep_Id
1	Ajay	M	Delhi	ajay@gmail.com	1
2	Vijay	M	Mumbai	vijay@gmail.com	2
3	Radhika	F	Bhopal	radhika@gmail.com	1
4	Shikha	F	Jaipur	shikha@gmail.com	2
5	Hritik	M	Jaipur	hritik@gmail.com	2

5 rows in set (0.00 sec)

Understanding keys

Table 1 | Candidate | Primary | Composite

Id	Name	Gender	City	Email	Dep_Id
1	Ajay	M	Delhi	ajay@gmail.com	1
2	Vijay	M	Mumbai	vijay@gmail.com	2
3	Radhika	F	Bhopal	radhika@gmail.com	1
4	Shikha	F	Jaipur	shikha@gmail.com	2
5	Hritik	M	Jaipur	hritik@gmail.com	2

5 rows in set (0.00 sec)

Understanding keys

Table 1 | Candidate | **Primary** | Composite

Primary Key | Alternate Key

Id	Name	Gender	City	Email	Dep_Id
1	Ajay	M	Delhi	ajay@gmail.com	1
2	Vijay	M	Mumbai	vijay@gmail.com	2
3	Radhika	F	Bhopal	radhika@gmail.com	1
4	Shikha	F	Jaipur	shikha@gmail.com	2
5	Hritik	M	Jaipur	hritik@gmail.com	2

5 rows in set (0.00 sec)

Understanding keys

civic data lab

Table 1 | Candidate | Primary | Composite

```
mysql> select * from product;
+----------------+------------+-------------+--------------+----------+
| Transaction_Id | Product_Id | Customer_Id | Product      | Quantity |
+----------------+------------+-------------+--------------+----------+
| A1001          | P1005      | C9001       | Smartphone   |        1 |
| A1001          | P2010      | C9001       | Screen guard |        1 |
| A1002          | P2013      | C9003       | Smartwatch   |        1 |
| A1003          | P2010      | C9010       | Screen guard |        2 |
+----------------+------------+-------------+--------------+----------+
4 rows in set (0.00 sec)
```

Foregin Keys

Id in the **Department table** [Primary Key]

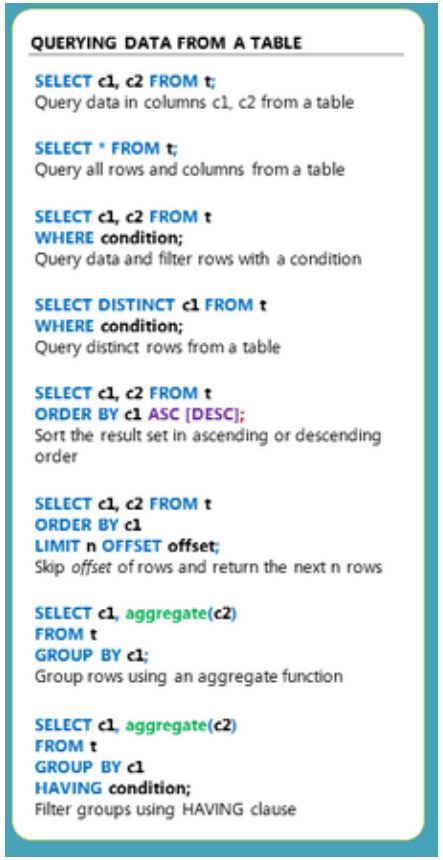
```
mysql> Select * from Department;
+----+-----------+----------+
| Id | Name      | Location |
+----+-----------+----------+
|  1 | Marketing | Kolkata  |
|  2 | Finance   | Mumbai   |
+----+-----------+----------+
2 rows in set (0.14 sec)
```

Dep_Id in the **Employee table** [Foreign Key]

```
+----+---------+--------+--------+-------------------+--------+
| Id | Name    | Gender | City   | Email             | Dep_Id |
+----+---------+--------+--------+-------------------+--------+
| 1  | Ajay    | M      | Delhi  | ajay@gmail.com    |      1 |
| 2  | Vijay   | M      | Mumbai | vijay@gmail.com   |      2 |
| 3  | Radhika | F      | Bhopal | radhika@gmail.com |      1 |
| 4  | Shikha  | F      | Jaipur | shikha@gmail.com  |      2 |
| 5  | Hritik  | M      | Jaipur | hritik@gmail.com  |      2 |
+----+---------+--------+--------+-------------------+--------+
5 rows in set (0.00 sec)
```

```
mysql> select employee.name, employee.city, department.name
    -> from employee inner join department
    -> on employee.dep_id = department.id;
+---------+--------+-----------+
| name    | city   | name      |
+---------+--------+-----------+
| Ajay    | Delhi  | Marketing |
| Radhika | Bhopal | Marketing |
| Vijay   | Mumbai | Finance   |
| Shikha  | Jaipur | Finance   |
| Hritik  | Jaipur | Finance   |
+---------+--------+-----------+
5 rows in set (0.00 sec)
```

SQL Basics



QUERYING DATA FROM A TABLE

SELECT c1, c2 FROM t;
Query data in columns c1, c2 from a table

SELECT * FROM t;
Query all rows and columns from a table

SELECT c1, c2 FROM t
WHERE condition;
Query data and filter rows with a condition

SELECT DISTINCT c1 FROM t
WHERE condition;
Query distinct rows from a table

SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
Sort the result set in ascending or descending order

SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
Skip *offset* of rows and return the next n rows

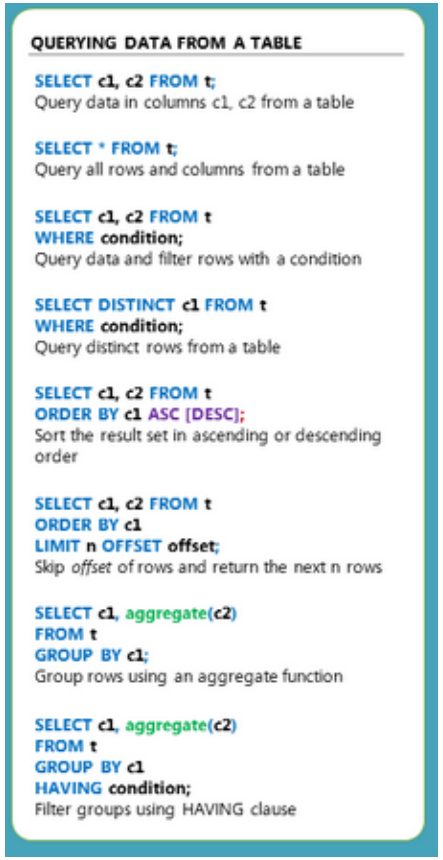
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
Group rows using an aggregate function

SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
Filter groups using HAVING clause

SELECT

SELECT {stuff you want to select} **FROM** {the table that it is in}

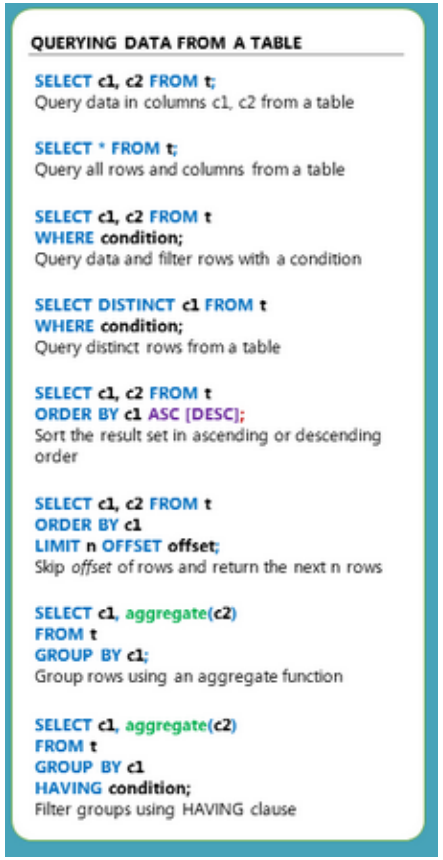
SQL Basics



SELECT ALL

The * is called a “splat” and is a handy, frequently used shortcut to get all columns.

SQL Basics



QUERYING DATA FROM A TABLE

SELECT c1, c2 FROM t;
Query data in columns c1, c2 from a table

SELECT * FROM t;
Query all rows and columns from a table

SELECT c1, c2 FROM t
WHERE condition;
Query data and filter rows with a condition

SELECT DISTINCT c1 FROM t
WHERE condition;
Query distinct rows from a table

SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
Sort the result set in ascending or descending order

SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
Skip *offset* of rows and return the next n rows

SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
Group rows using an aggregate function

SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
Filter groups using HAVING clause

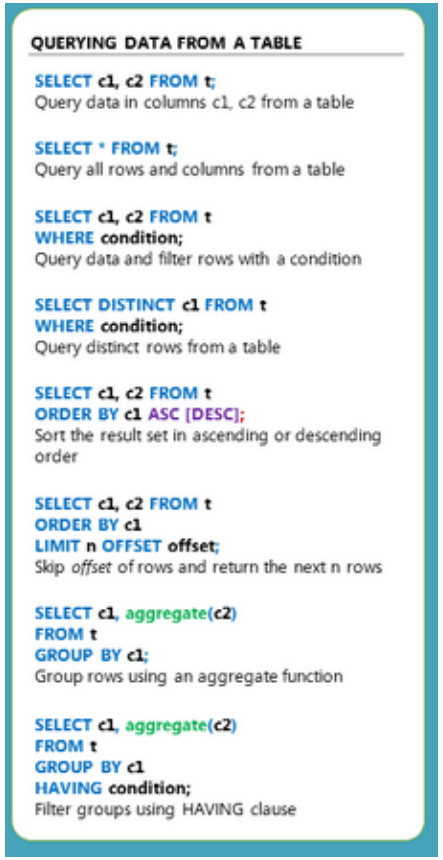
ORDER BY

SELECT {stuff you want to select} FROM {the table that it is in} ORDER BY {column you want to order by}

ASCending (*default*) or **DESC**ending

SELECT * FROM tracks ORDER BY name DESC

SQL Basics



QUERYING DATA FROM A TABLE

SELECT c1, c2 FROM t;
Query data in columns c1, c2 from a table

SELECT * FROM t;
Query all rows and columns from a table

SELECT c1, c2 FROM t
WHERE condition;
Query data and filter rows with a condition

SELECT DISTINCT c1 FROM t
WHERE condition;
Query distinct rows from a table

SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
Sort the result set in ascending or descending order

SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
Skip *offset* of rows and return the next n rows

SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
Group rows using an aggregate function

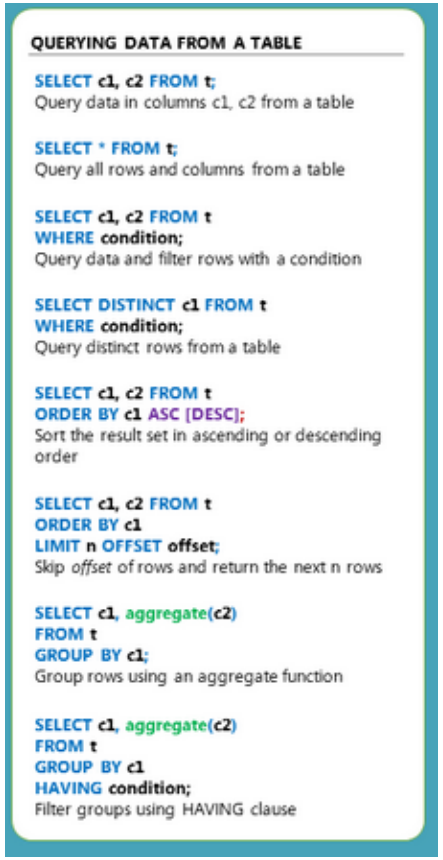
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
Filter groups using HAVING clause

LIMIT

LIMIT the number of rows - SELECT * FROM artists LIMIT {Number to Limit By}

SELECT * FROM artists LIMIT 5

SQL Basics



QUERYING DATA FROM A TABLE

SELECT c1, c2 FROM t;
Query data in columns c1, c2 from a table

SELECT * FROM t;
Query all rows and columns from a table

SELECT c1, c2 FROM t
WHERE condition;
Query data and filter rows with a condition

SELECT DISTINCT c1 FROM t
WHERE condition;
Query distinct rows from a table

SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
Sort the result set in ascending or descending order

SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
Skip *offset* of rows and return the next n rows

SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
Group rows using an aggregate function

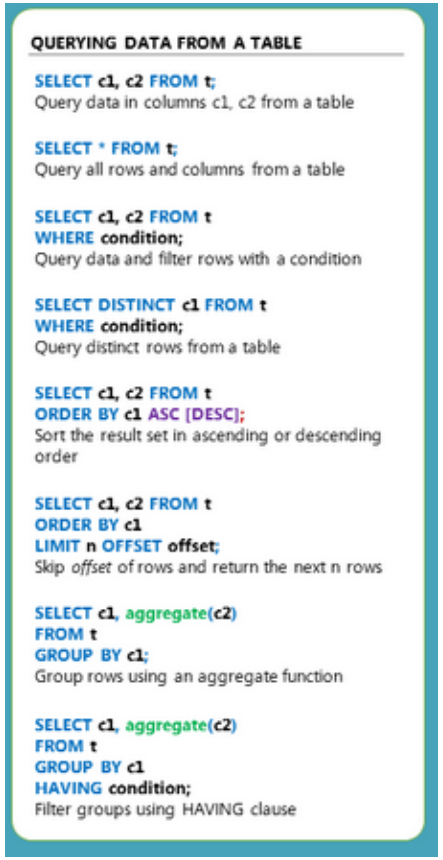
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
Filter groups using HAVING clause

OFFSET

OFFSET - Where to start returning data - SELECT * FROM artists LIMIT 5 OFFSET {Number of rows to skip}

SELECT * FROM artists LIMIT 5 OFFSET 2

SQL Basics



WHERE

The **WHERE** command is followed by the conditions you'd like to filter by.

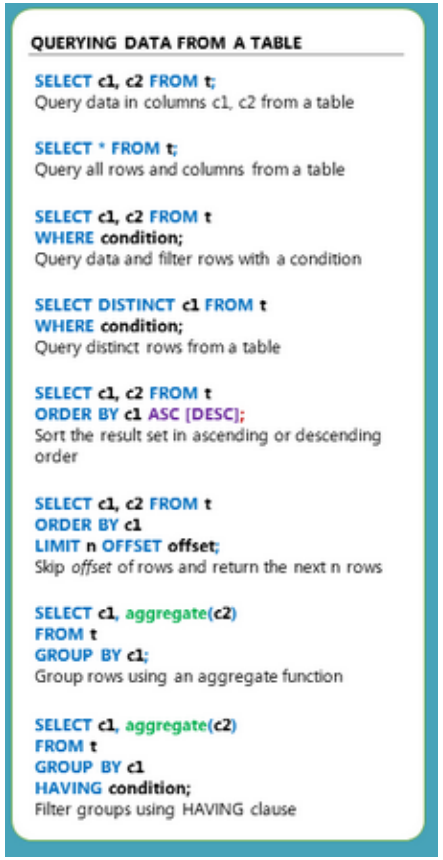
SELECT * FROM artists WHERE {Filter Conditions};

Single clause - SELECT * FROM artists WHERE id = 85

Multiple clauses - SELECT * FROM tracks WHERE album_id = 89 AND composer = 'Green Day'

Combinations - SELECT * FROM tracks WHERE composer = 'Green Day' OR (composer = 'AC/DC' AND milliseconds > 240000)

SQL Basics



QUERYING DATA FROM A TABLE

SELECT c1, c2 FROM t;
Query data in columns c1, c2 from a table

SELECT * FROM t;
Query all rows and columns from a table

SELECT c1, c2 FROM t
WHERE condition;
Query data and filter rows with a condition

SELECT DISTINCT c1 FROM t
WHERE condition;
Query distinct rows from a table

SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
Sort the result set in ascending or descending order

SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
Skip *offset* of rows and return the next n rows

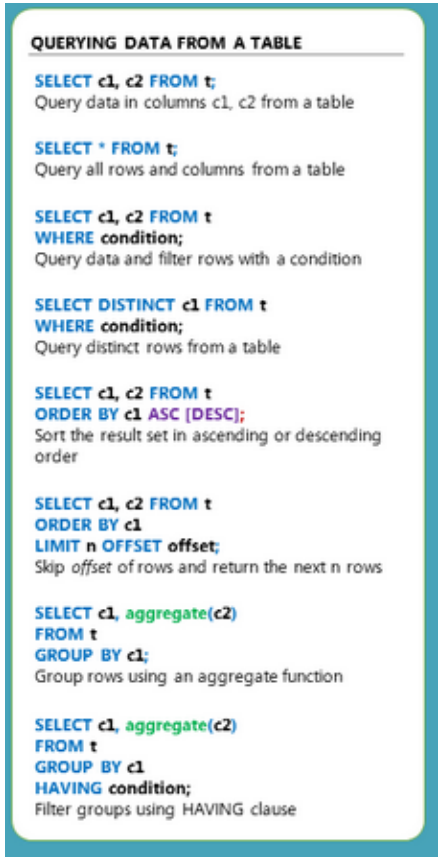
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
Group rows using an aggregate function

SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
Filter groups using HAVING clause

AGGREGATE

FUNCTION	DESCRIPTION
MAX	returns the largest (maximum) number in a sets
MIN	described
COUNT	returns a count of the # of values in a set
COUNT DISTINCT	returns a count of the # of unique (distinct) values in a set
EVERY	returns true if all data inside is true (same as bool_and)
AVG	returns the average (mean) of the set of numbers
SUM	returns the sum of all the values in the set

SQL Basics



QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

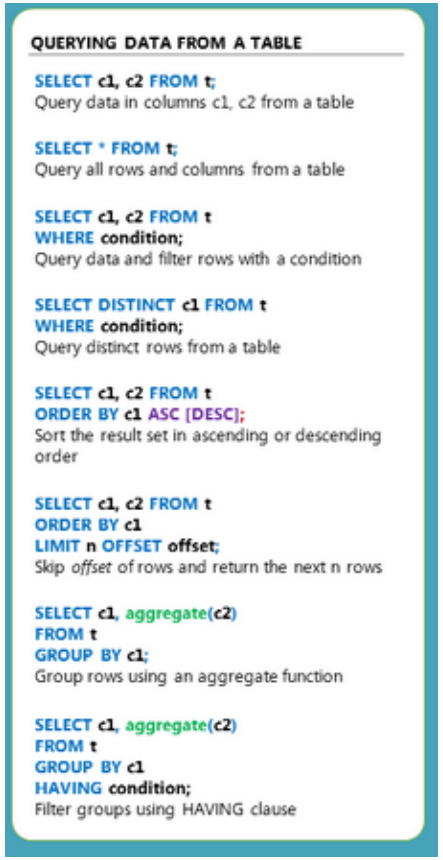
AGGREGATE

MAX/MIN/AVG of one column - SELECT MAX(unit_price), MIN(unit_price), AVG(unit_price) FROM tracks

Total Rows - SELECT COUNT(*) FROM tracks

Unique Values in a column - SELECT COUNT(DISTINCT composer) FROM tracks

SQL Basics



QUERYING DATA FROM A TABLE

```
SELECT c1, c2 FROM t;
```
Query data in columns c1, c2 from a table

```
SELECT * FROM t;
```
Query all rows and columns from a table

```
SELECT c1, c2 FROM t
WHERE condition;
```
Query data and filter rows with a condition

```
SELECT DISTINCT c1 FROM t
WHERE condition;
```
Query distinct rows from a table

```
SELECT c1, c2 FROM t
ORDER BY c1 ASC [DESC];
```
Sort the result set in ascending or descending order

```
SELECT c1, c2 FROM t
ORDER BY c1
LIMIT n OFFSET offset;
```
Skip *offset* of rows and return the next n rows

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1;
```
Group rows using an aggregate function

```
SELECT c1, aggregate(c2)
FROM t
GROUP BY c1
HAVING condition;
```
Filter groups using HAVING clause

GROUP BY

For splitting aggregations into groups

Counting - SELECT genre_id, COUNT(*) FROM tracks GROUP BY genre_id

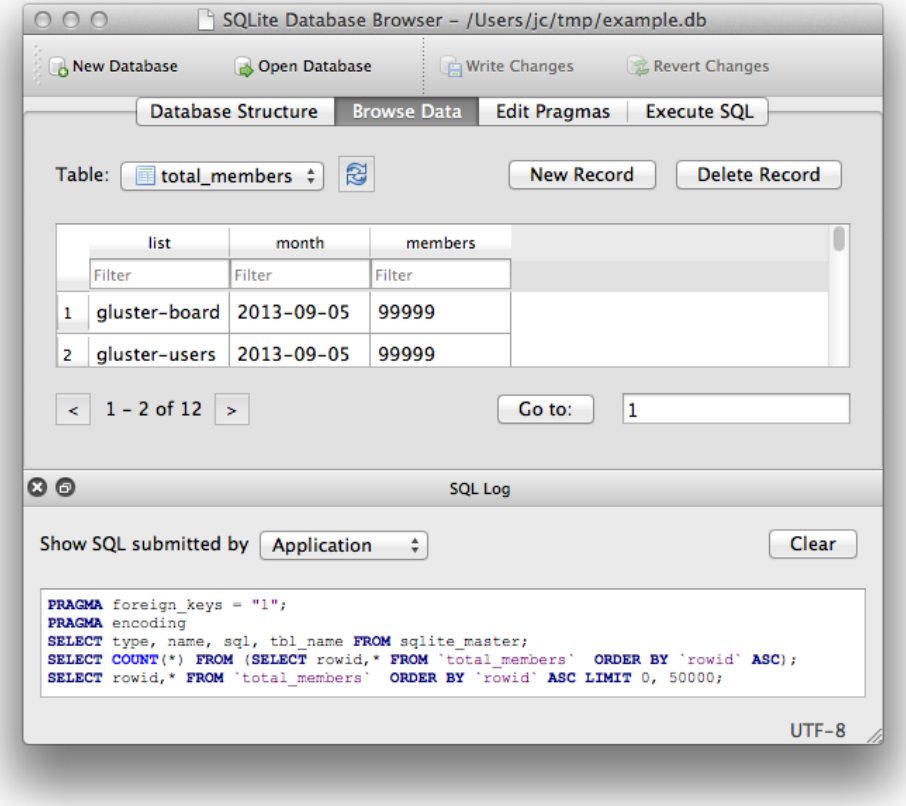
Count but rename column - SELECT composer, COUNT(*) as "count" FROM tracks GROUP BY composer ORDER BY "count" DESC

Aggregate by multiple columns - SELECT media_type_id, genre_id, COUNT(*) FROM tracks GROUP BY media_type_id, genre_id ORDER BY media_type_id, genre_id

Database tools

1. Database - SQLite
2. Database Explorer - SQLite Browser - *For working with database*

> DB Browser for SQLite (DB4S) is a high quality, visual, open source tool to create, design, and edit database files compatible with SQLite.



Database exercises

Exploring mortality data

- Import the CSV file in the database
- Explore variables
- Find total number of rows
- Find unique states
- Find the maximum and minimum number of death across all states and years
- Find total deaths in April 2020 across all states
- Calculate total deaths across years
- Extract the top 5 entries in terms of number of deaths across year and state

Exploring data from eCourts

Dataset - Link - *The database contains 81.2 million cases*

Source: Devdatalab

Objective:

- Understand how the data is structured
- Import the data in a database
- Explore the sample datasets
- Find out the total cases present for each district for the year 2018

Tags

database *large-datasets* *sqlite* *eCourts*

Exercise - Using Databases

- Install SQLite DB Browser
- Create a new database
- Load the judges_clean dataset in the DB
- Find the distribution of male/female judges in **Bengaluru** district court where judge position is *chief metropolitan magistrate*
- Save the file, as CSV, in the drive