SQL for DA/DS

Basics and Advanced Queries

Typical SQL Query

Run | Limit 100 | Forma

```
1  SELECT *
2  FROM tutorial.orders
3  WHERE id BETWEEN 10 AND 20
4  ORDER BY quantity ASC;
```

✓ 11 rows | 440B returned in 627ms | Copy | Add chart

	id	account_id	occurred_at	quantity	amount_us
1	11	1001	2016-07-30 03:26:30	137	773.6
2	10	1001	2016-06-30 12:32:05	148	878.5
3	13	1001	2016-09-26 23:28:25	158	951.1
4	12	1001	2016-08-28 07:13:39	196	1182.6
5	15	1001	2016-11-25 23:21:32	210	1283.1
6	16	1001	2016-12-24 05:53:13	269	1719.2
7	14	1001	2016-10-26 20:31:30	294	1993.5
8	20	1021	2015-12-11 16:53:18	504	2580.6
9	18	1021	2015-10-12 02:21:56	539	2747.1
10	17	1011	2016-12-21 10:59:34	541	2734.5
11	19	1021	2015-11-11 07:37:01	558	2936.9

How different roles use SQL

- Data Analyst
  - Retrieving data
  - Analysis + Dashboard
- Data Scientist
  - Retrieving data
  - Analysis + Modeling
- Data Engineer
  - Efficient Data pipelines
  - Database migration, etc.

SQL Interview Setup

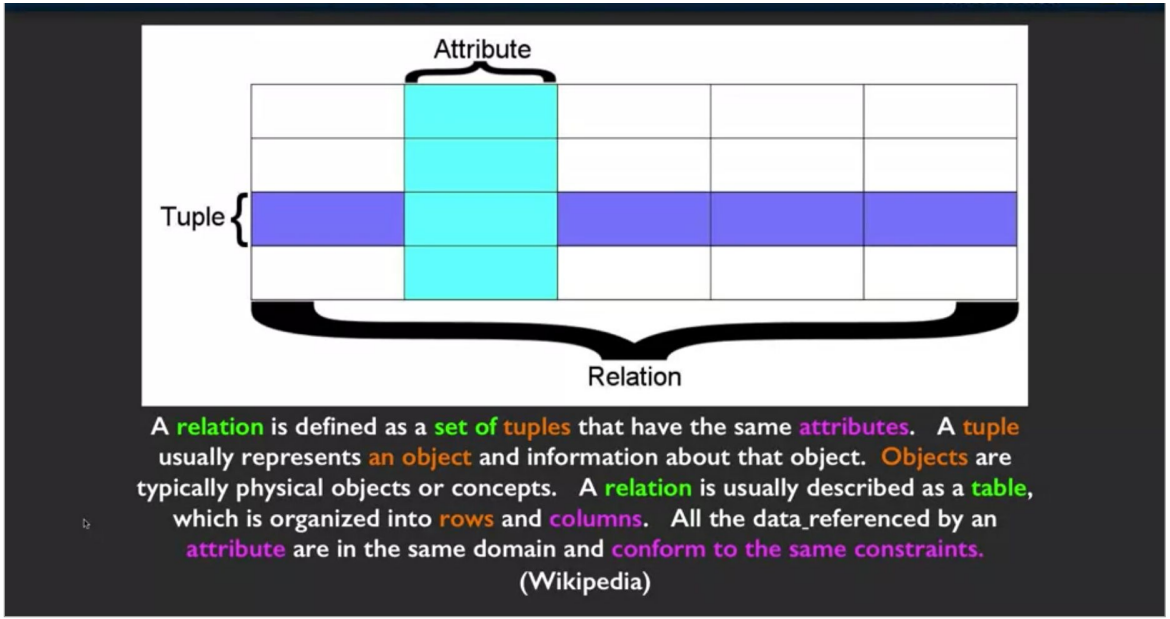
- A typical interview format:
  - Pure SQL data challenge
    - Timed ~3 hours
  - 2-3 days
- Online shared platform like coderpad
- Platform of your choice even Jupyter Notebook/doc/text
- Multiple Choice Questions on HackerRank/other similar platforms
- Behavioral quiz/questions by the team

What’s the interviewer looking for?

- Fundamental understanding on concepts
- Knowledge of basic data structures and commands
- Different types of JOINs
- Window Functions
- Efficiency in Queries
- Differences between different commands such as WHERE vs HAVING
- Critical thinking
- Communication skills!

What’s SQL?

- Structured Query Language
- A relational database system
  - Rows/Columns
  - Tables
  - Connections



Relational Databases (RDBs)

- Stores rows and columns in tables
- Useful in efficiently retrieving data from those tables
- Multiple tables are joined together
- Powerful when data is retrieved from multiple tables

Why SQL?

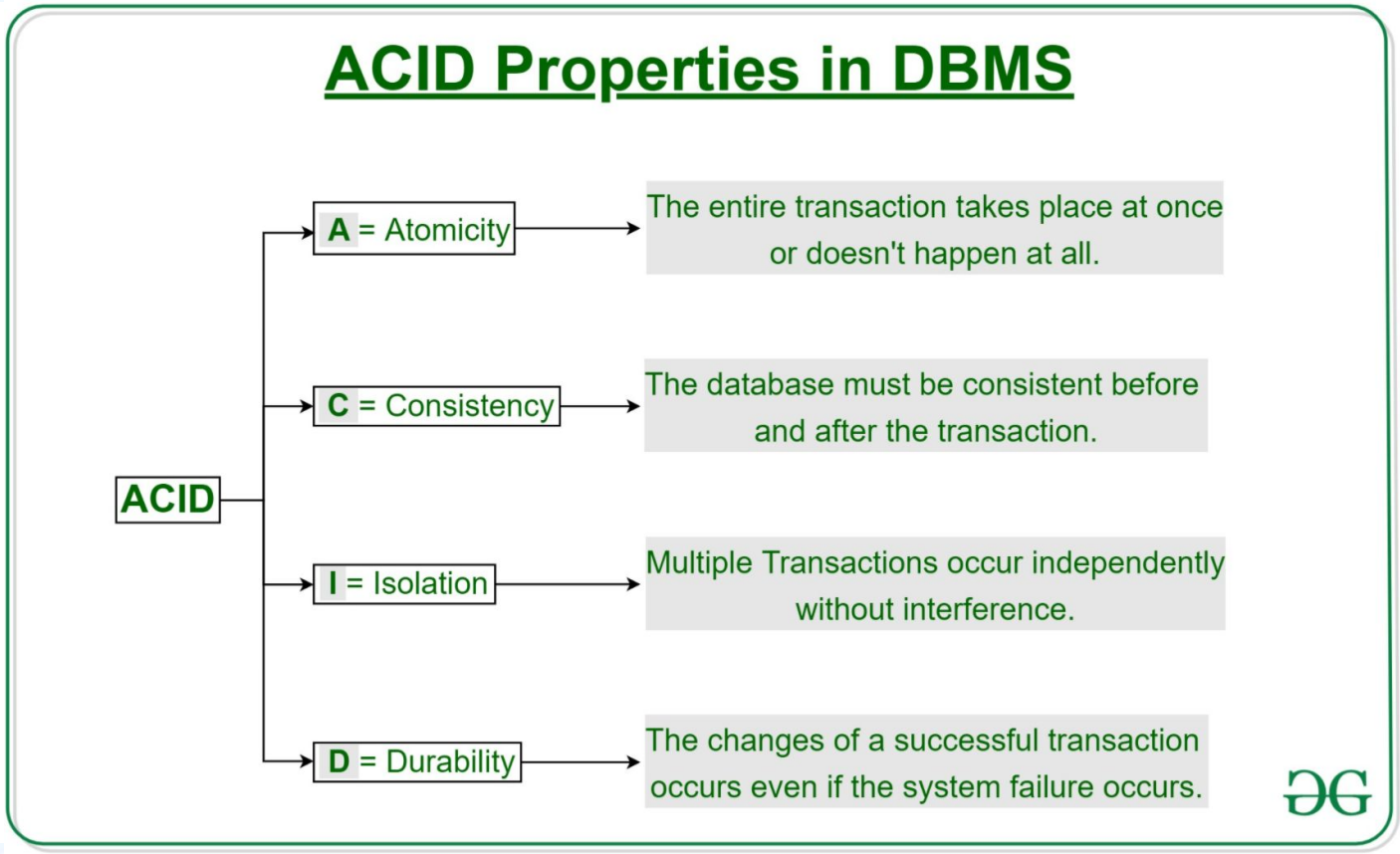
- Why not Excel?
- SQL is much faster (minutes vs hours)
- SQL can handle complex tables and relationships

- RULE of THUMB
  - Small amount of data → Excel
  - Math/specific functions → Excel
  - Large amount of data → SQL
  - Manipulating large data → SQL

The Beauty of SQL

- Simple to write and understand
- High speed:
  - quickly and efficiently retrieve a large amount of records
- Well defined standards: ANSI (American National Standards Institute)
- Portability:
  - used on laptops, PCs, mobile phones
- Multiple data view
  - users can make different views of the database structure

ACID Properties of SQL*



Common Database Systems

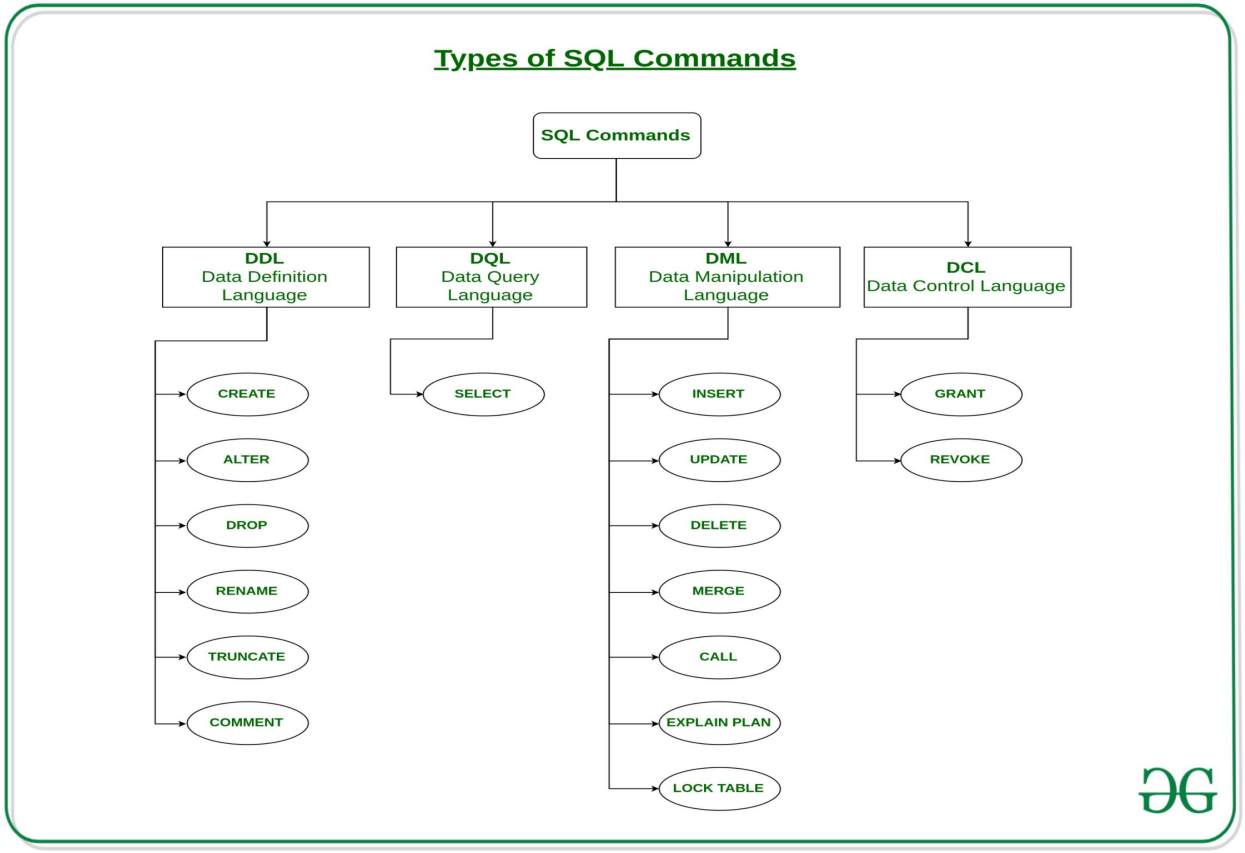
Three major Database Management Systems in wide use:

- Oracle - Large, commercial, enterprise-scale
- MySQL - Simpler but very fast and scalable - commercial open source
- sqlServer - Very nice - from Microsoft
- Other smaller projects, free and open source and gaining popularity:
  - HSQL
  - SQLite (DB Browser)
  - Postgres PgAdmin/Postgres/VSCode/terminal

Three Main Subsets of SQL

- Data definition language (**DDL**): to define the data structure it consists of the commands like CREATE, ALTER, DROP, etc.
- Data manipulation language (**DML**): to manipulate already existing data in the database.
- Data control language (**DCL**): to control access to data in the database and includes commands such as GRANT, REVOKE.

Subsets of SQL



Basic Types of SQL Commands: CRUD

Structured Query Language is the language we use to issue commands to the database:

1. Create a table
2. Retrieve some data
3. Update/Insert data
4. Delete data

1. Creating a table

```
CREATE TABLE data_courses(
 course_id SERIAL PRIMARY KEY,
 course_name VARCHAR (50) UNIQUE NOT NULL,
 course_instructor VARCHAR (100) NOT NULL,
 topic VARCHAR (20) NOT NULL);
```

General Structure of the Table

```
CREATE TABLE table_name (

 column_name TYPE column_constraint,

 table_constraint table_constraint);
```

Inserting Records in the Table

```
INSERT INTO table(column1, column2, ...)

VALUES(value1, value2, ...);
```

Inserting Values into the Table

```
INSERT INTO data_courses(course_name, course_instructor, topic)
VALUES('Python', 'Olga Boldaviera', 'Python');

INSERT INTO data_courses(course_name, course_instructor, topic)
VALUES('PostgreSQL', 'Manjula Mishra', 'SQL');

INSERT INTO data_courses(course_name, course_instructor, topic)
VALUES('Machine Learning, 'Tina Kovacova', 'Data Science');
```

Common Column Constraints in SQL*

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Uniquely identifies a row/record in another table
- CHECK - Ensures that all values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column when no value is specified
- INDEX - Used to create and retrieve data from the database very quickly

2. Retrieving Data

Syntax:

SELECT column_1, Column_2, ...

FROM Table_name;

*SELECT * FROM data_courses;*

3. Updating a Record in the Table

UPDATE table

SET column1 = value1,

column2 = value2 ,...

WHERE Condition;

UPDATE data_courses SET course_name = 'Advanced SQL'

WHERE course_instructor = 'Manjula Mishra';

4. Deleting a Record from the Table

DELETE FROM table

WHERE condition;

DELETE from data_courses

WHERE course_name = 'Python';

Basic Functions in SQL

Basic Commands

- SELECT
- WHERE
- LIMIT
- ORDER BY

Logical Operators:

- LIKE
- IN
- BETWEEN IS NULL
- AND
- OR
- NOT (e.g.NOT BETWEEN)

Aggregate functions in SQL

- COUNT() – counts the total number of records.
- SUM() – sums the values in records.
- AVG() – averages values in records.
- MAX() – finds the highest value in all of the records.
- MIN() – finds the lowest value in all of the records.

Intermediate Functions

- DISTINCT
- GROUP BY
- CASE
- JOINS
  - INNER JOIN
  - OUTER JOIN
  - LEFT JOIN
  - SELF JOIN

Advanced Functions

- SQL data type
- SQL date format
- Data wrangling with SQL
- Subqueries
- SQL window functions

The Order Queries are Executed

The query steps don’t happen in the order they’re written:

The way it’s written

- SELECT.....
- FROM & JOIN....
- WHERE .....
- GROUP BY...
- HAVING....
- ORDER BY....
- LIMIT.....

How you should think about it

FROM & JOIN...

WHERE

GROUP BY

HAVING

SELECT

ORDER BY

LIMIT

Schema in SQL and why it’s important

- Database design is an art of its own!
- Collection of logical structure of data
- Blueprint of how the database is constructed
  - In case of RDBs, divided into database tables

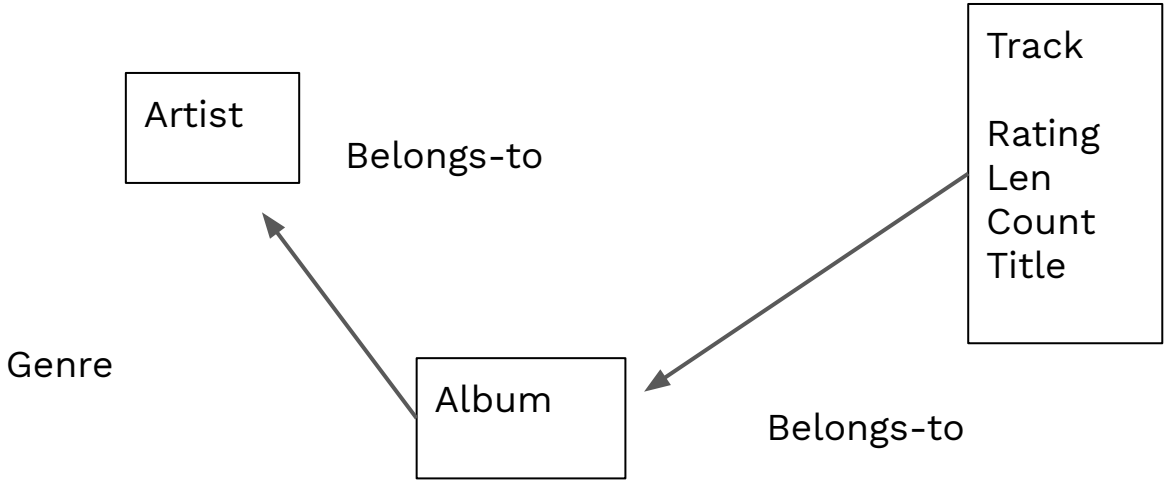
- The goal is to avoid the bad mistake and design clean and easily understood databases
- Database design starts with a picture
- It is to avoid duplication
  - Storage perspective
  - Fast indexing

Example

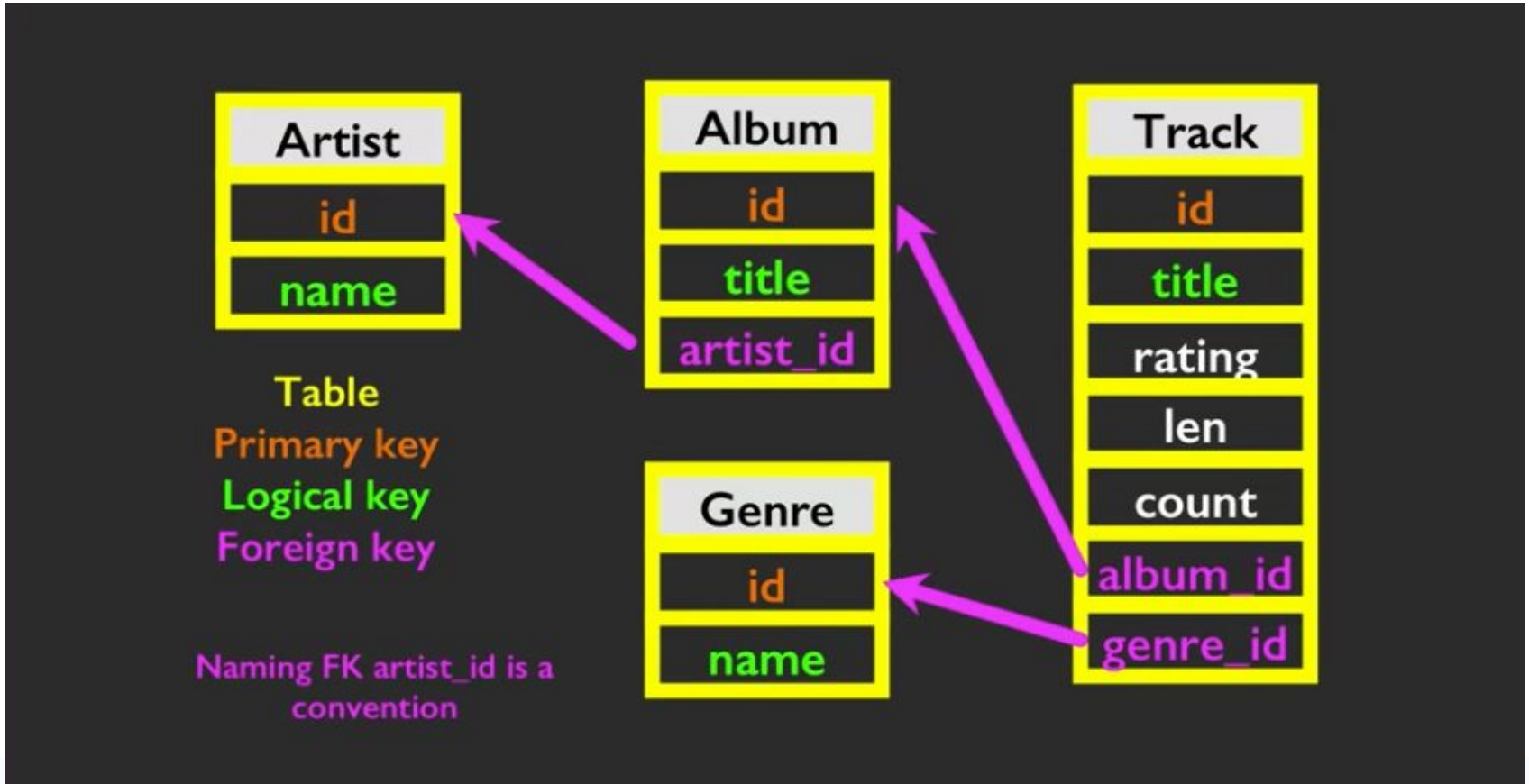
Track	Len	Artist	Album	Genre	Rating	Count
Hells Bells	5:13	AC/DC	Who Made Who	Rock	★★★★★	61
Shake Your Foundations	3:54	AC/DC	Who Made Who	Rock	★★★★★	70
Chase the Ace	3:01	AC/DC	Who Made Who	Rock		56
For Those About To Rock (We ...	5:54	AC/DC	Who Made Who	Rock	★★★★★	61
Dúlamán	3:43	Altan	Natural Wonders M...	New Age		31
Rode Across the Desert	4:10	America	Greatest Hits	Easy Listen...	★★★★★	23
Now You Are Gone	3:08	America	Greatest Hits	Easy Listen...	★★★★★	18
Tin Man	3:30	America	Greatest Hits	Easy Listen...	★★★★★	23
Sister Golden Hair	3:22	America	Greatest Hits	Easy Listen...	★★★★★	24
Track 01	4:22	Billy Price	Danger Zone	Blues/R&B	★★★★★	26
Track 02	2:45	Billy Price	Danger Zone	Blues/R&B	★★★★★	18
Track 03	3:26	Billy Price	Danger Zone	Blues/R&B	★★★★★	22
Track 04	4:17	Billy Price	Danger Zone	Blues/R&B	★★★★★	18
Track 05	3:50	Billy Price	Danger Zone	Blues/R&B	★★★★★	21
War Pigs/Luke's Wall	7:58	Black Sabbath	Paranoid	Metal	★★★★★	25
Paranoid	2:53	Black Sabbath	Paranoid	Metal	★★★★★	22
Planet Caravan	4:35	Black Sabbath	Paranoid	Metal	★★★★★	25
Iron Man	5:59	Black Sabbath	Paranoid	Metal	★★★★★	26
Electric Funeral	4:53	Black Sabbath	Paranoid	Metal	★★★★★	22
Hand of Doom	7:10	Black Sabbath	Paranoid	Metal	★★★★★	23
Rat Salad	2:30	Black Sabbath	Paranoid	Metal	★★★★★	31
Jack the Stripper/Fairies Wear ...	6:14	Black Sabbath	Paranoid	Metal	★★★★★	24
Bomb Squad (TECH)	3:28	Brent	Brent's Album			1
clay techno	4:36	Brent	Brent's Album			2
Heavy	3:08	Brent	Brent's Album			1
Hi metal man	4:20	Brent	Brent's Album			1
Mistro	2:58	Brent	Brent's Album			1

Building a database/Scheme

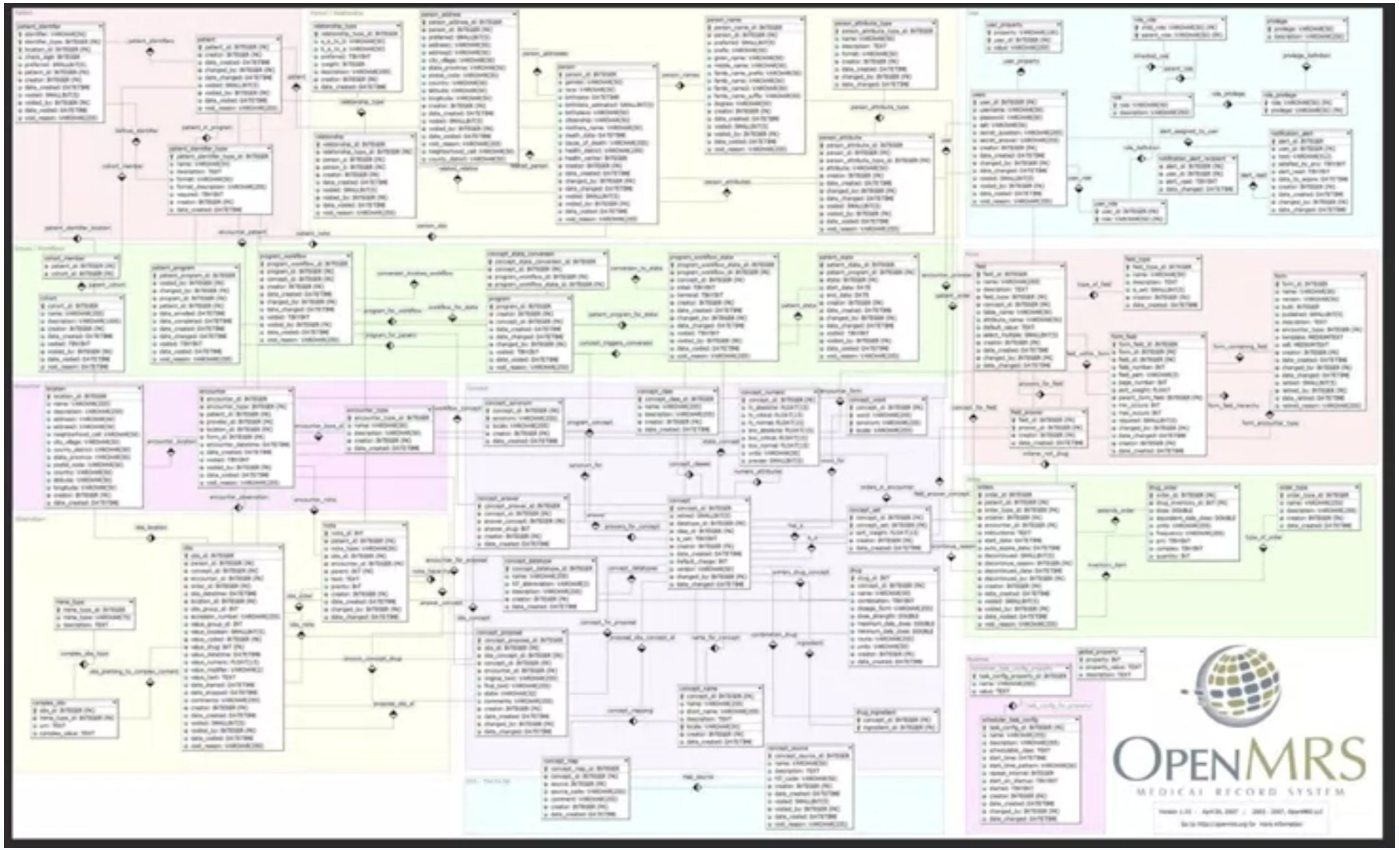
Track
Album
Artist
Genre
Rating
Len
Count



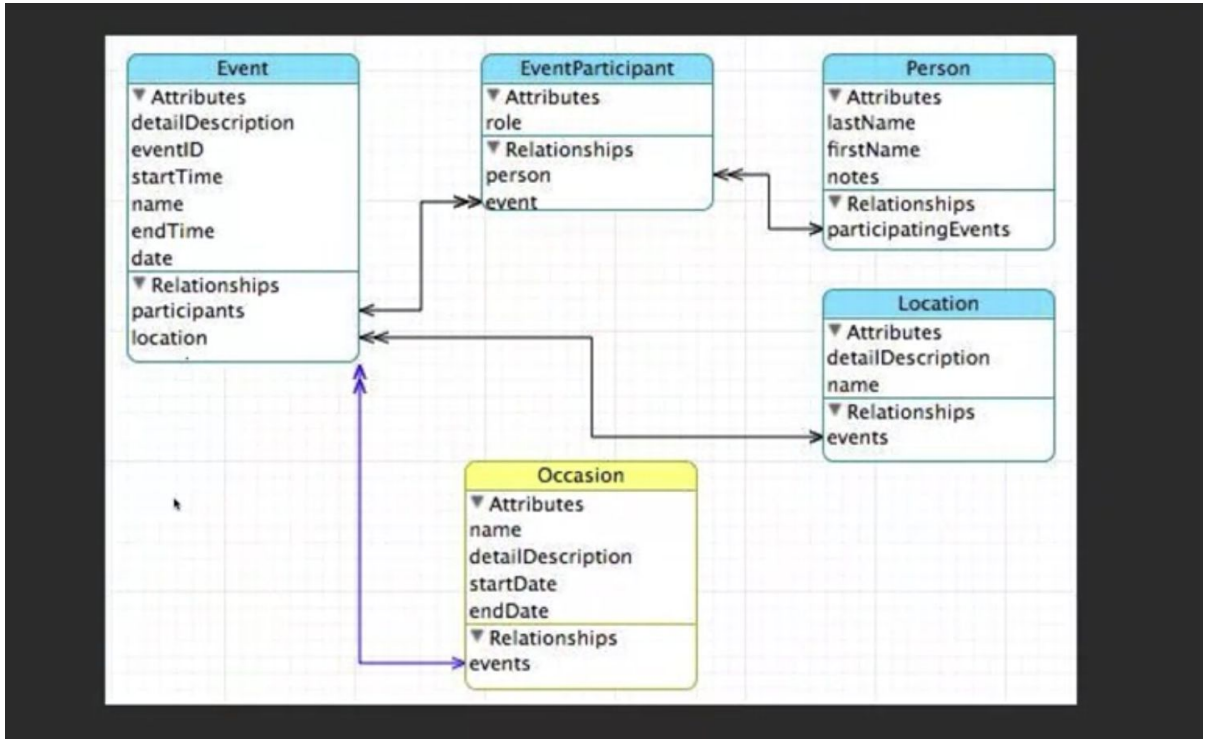
The schema


Artist
id
name
Album
id
title
artist_id
Track
id
title
rating
len
count
album_id
genre_id
Genre
id
name
Table
Primary key
Logical key
Foreign key
Naming FK artist_id is a convention

Real world schema!



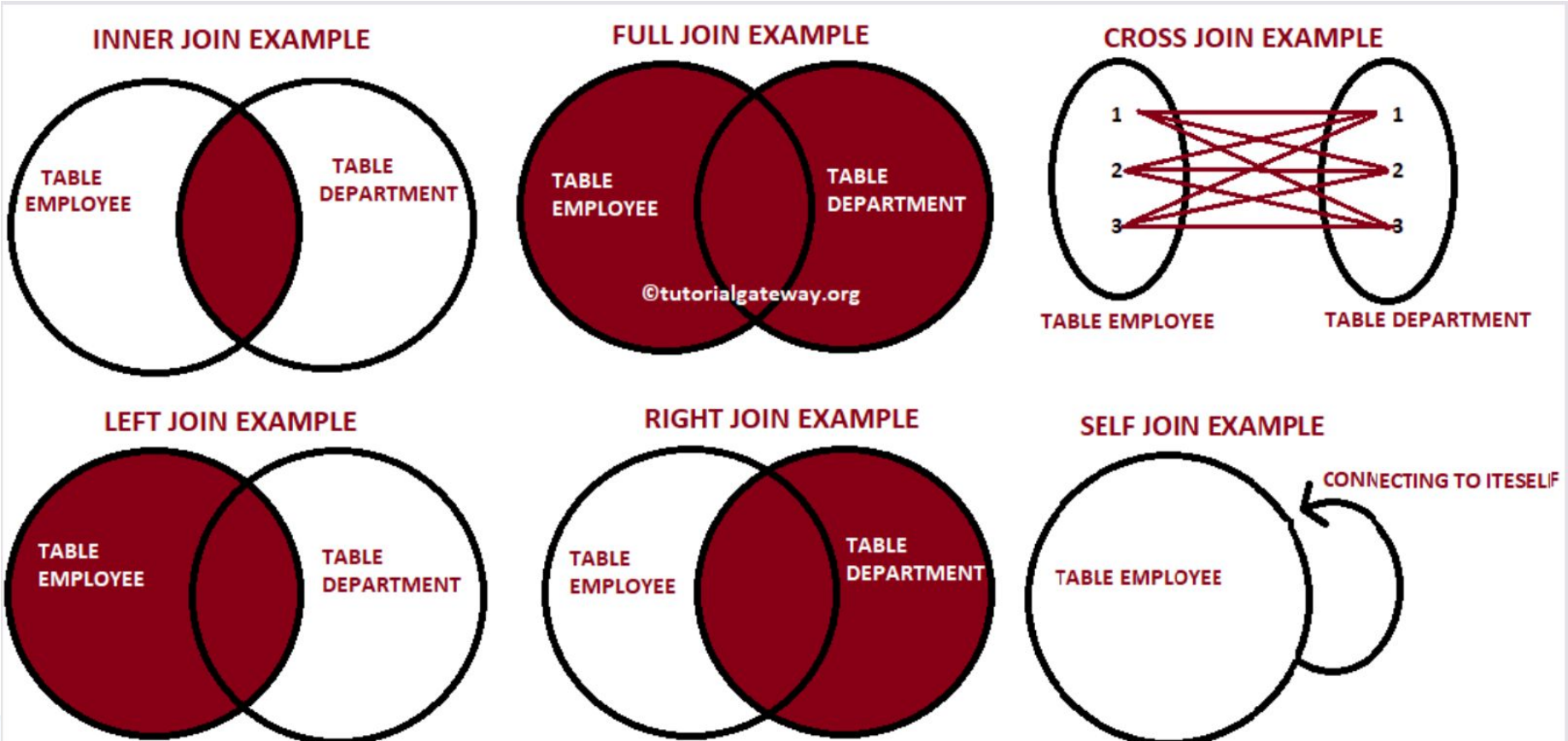
Entity Relationship Diagram



Relationship types*:

- One to One
- One to Many and Many to One
- Many to Many
- Self Referencing Relationships

JOINS types


INNER JOIN EXAMPLE
TABLE EMPLOYEE
TABLE DEPARTMENT
FULL JOIN EXAMPLE
TABLE EMPLOYEE
TABLE DEPARTMENT
©tutorialgateway.org
CROSS JOIN EXAMPLE
1
2
3
1
2
3
TABLE EMPLOYEE
TABLE DEPARTMENT
LEFT JOIN EXAMPLE
TABLE EMPLOYEE
TABLE DEPARTMENT
RIGHT JOIN EXAMPLE
TABLE EMPLOYEE
TABLE DEPARTMENT
SELF JOIN EXAMPLE
CONNECTING TO ITESELF
TABLE EMPLOYEE

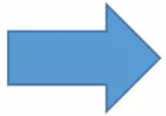
CROSS JOIN and SELF JOIN

CROSS JOIN

Tables

Color
Red
Blue

Size
Small
Medium
Large
Extra Large



Query Result

Color	Size
Red	Small
Blue	Small
Red	Medium
Blue	Medium
Red	Large
Blue	Large
Red	Extra Large
Blue	Extra Large

SELF JOIN

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City

FROM Customers A, Customers B

WHERE A.CustomerID <> B.CustomerID

AND A.City = B.City

ORDER BY A.City;
```

Syntax for JOINing two tables

```
SELECT teams.conference AS conference,
       AVG(players.weight) AS average_weight
  FROM benn.college_football_players players
  JOIN benn.college_football_teams teams
    ON teams.school_name = players.school_name
 GROUP BY teams.conference
 ORDER BY AVG(players.weight) DESC
```

Joining Multiple Tables

```
SELECT v.name, c.name,  p.lastname
FROM vehicle v
INNER JOIN color c ON  v.color_id = c.id
INNER JOIN person p ON v.person_id = p.id ;
```

Slightly different syntax

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

GROUP BY

```sql
SELECT year,
       month,
       COUNT(*) AS count
  FROM tutorial.aapl_historical_stock_price
 GROUP BY year, month
 ORDER BY month, year
```

GROUP BY works with aggregate functions

100 rows | 2KB returned in 422ms

Copy

Add chart

	year	month	count
1	2000	1	20
2	2001	1	21
3	2002	1	21
4	2003	1	21
5	2004	1	20
6	2005	1	20
7	2006	1	20
8	2007	1	20
9	2008	1	21
10	2009	1	21
11	2010	1	21
12	2011	1	20

HAVING clause

Run | Limit 100 | Forma | NEW | Manage edito

```
1  SELECT year,
2         month,
3         MAX(high) AS month_high
4    FROM tutorial.aapl_historical_stock_price
5   GROUP BY year, month
6  HAVING MAX(high) > 400
7   ORDER BY year, month
```

30 rows | 720B returned in 406ms | Copy | Add chart

	year	month	month_high
1	2011	7	404.5
2	2011	9	422.86
3	2011	10	426.7
4	2011	11	408
5	2011	12	409.09
6	2012	1	458.24
7	2012	2	547.61
8	2012	3	621.45
9	2012	4	644
10	2012	5	596.76
11	2012	6	590
12	2012	7	619.87

What's the difference between WHERE and HAVING?

Let’s Guess the answers of these problem

What is the difference between NULL value, zero and blank space?

Answers

A NULL value is not the same as zero or a blank space. A NULL value is a value which is 'unavailable, unassigned, unknown or not applicable.' On the other hand, zero is a number, and a blank space is treated as a character.

The NULL value can be treated as unknown and missing value as well, but zero and blank spaces are different from the NULL value.

Let’s Guess the answers of these problem

What is the difference between BETWEEN and IN condition operators?

The BETWEEN operator is used to display rows based on a range of values. The values can be numbers, text, and dates as well. BETWEEN operator gives us the count of all the values occurs between a particular range.

The IN condition operator is used to check for values contained in a specific set of values. IN operator is used when we have more than one value to choose.

Practice Problem*

Write an SQL query to get the third maximum salary of an employee from a table named employee_table (Hint: it involves writing a subquery).

```
SELECT TOP 1 salary
FROM (
SELECT TOP 3 salary
FROM employee_table
ORDER BY salary DESC ) AS emp
ORDER BY salary ASC;
```

Practice Problem*

```
sql> SELECT * FROM runners;
+----+-------------+
| id | name        |
+----+-------------+
|  1 | John Doe    |
|  2 | Jane Doe    |
|  3 | Alice Jones |
|  4 | Bobby Louis |
|  5 | Lisa Romero |
+----+-------------+

sql> SELECT * FROM races;
+----+----------------+-----------+
| id | event          | winner_id |
+----+----------------+-----------+
|  1 | 100 meter dash |  2        |
|  2 | 500 meter dash |  3        |
|  3 | cross-country  |  2        |
|  4 | triathalon     |  NULL     |
+----+----------------+-----------+
```

```
SELECT * FROM runners WHERE id NOT IN (SELECT winner_id FROM races)
```

What's the result of these queries? Different or same?

```
SELECT * FROM runners WHERE id NOT IN (SELECT winner_id FROM races WHERE winner_id IS NOT null)
```

Explanation (assuming ANSI SQL):

If the set being evaluated by the SQL NOT IN condition contains *any* values that are null, then the outer query here will return an empty set, even if there are many runner ids that match winner_ids in the races table

CASE (WHEN..THEN) with Multiple Conditions

Example Table

```
 gid    |    datepose     |    pvc
--------+-----------------+-----------
1       |  1961           | 01
2       |  1949           |
3       |  1990           | 02
1       |  1981           |
1       |                 | 03
1       |                 |
```

```
SELECT
 *,
 CASE
  WHEN (pvc IS NULL OR pvc = '') AND (datepose < 1980) THEN '01'
  WHEN (pvc IS NULL OR pvc = '') AND (datepose >= 1980) THEN '02'
  WHEN (pvc IS NULL OR pvc = '') AND (datepose IS NULL OR datepose = 0) THEN '0
  ELSE '00'
 END AS modifiedpvc
FROM my_table;


 gid | datepose | pvc | modifiedpvc
-----+----------+-----+-------------
   1 |     1961 | 01  | 00
   2 |     1949 |     | 01
   3 |     1990 | 02  | 00
   1 |     1981 |     | 02
   1 |          | 03  | 00
   1 |          |     | 03
(6 rows)
```

Window Functions: SUM, COUNT, and AVG

Run | Limit 100 | Format SQL | View History...

```
1  SELECT start_terminal,
2         duration_seconds,
3         SUM(duration_seconds) OVER
4           (PARTITION BY start_terminal) AS running_total,
5         COUNT(duration_seconds) OVER
6           (PARTITION BY start_terminal) AS running_count,
7         AVG(duration_seconds) OVER
8           (PARTITION BY start_terminal) AS running_avg
9    FROM tutorial.dc_bikeshare_q1_2012
10   WHERE start_time < '2012-01-08'
```

✓ 100 rows | 4KB returned in 605ms

	start_terminal	duration_seconds	running_total	running_count	running_avg
1	31000	277	12207	16	762.9375
2	31000	1422	12207	16	762.9375
3	31000	398	12207	16	762.9375
4	31000	414	12207	16	762.9375
5	31000	3340	12207	16	762.9375
6	31000	291	12207	16	762.9375
7	31000	2661	12207	16	762.9375
8	31000	387	12207	16	762.9375
9	31000	520	12207	16	762.9375
10	31000	393	12207	16	762.9375
11	31000	447	12207	16	762.9375

Window Functions: ROW_NUMBER() vs RANK()

Run | Limit 100 | Forma | NEW | Manage

```
1  SELECT start_terminal,
2         start_time,
3         duration_seconds,
4         ROW_NUMBER() OVER (ORDER BY start_time)
5                           AS row_number
6    FROM tutorial.dc_bikeshare_q1_2012
7   WHERE start_time < '2012-01-08'
```

100 rows | 3KB returned in 471ms | Copy | Add chart

	start_terminal	start_time	duration_seconds	row_number
1	31245	2012-01-01 00:04:00	475	1
2	31400	2012-01-01 00:10:00	1162	2
3	31400	2012-01-01 00:10:00	1145	3
4	31101	2012-01-01 00:15:00	485	4
5	31102	2012-01-01 00:15:00	471	5
6	31017	2012-01-01 00:17:00	358	6
7	31236	2012-01-01 00:18:00	1754	7
8	31101	2012-01-01 00:22:00	259	8
9	31014	2012-01-01 00:24:00	516	9
10	31101	2012-01-01 00:25:00	913	10
11	31303	2012-01-01 00:29:00	1097	11
12	31222	2012-01-01 00:30:00	490	12
13	31230	2012-01-01 00:32:00	1045	13
14	31107	2012-01-01 00:32:00	1035	14
15	31107	2012-01-01 00:33:00	1060	15

Page 1 of 1 | Showing 1-100 of 100 res

Run | Limit 100 | Forma | NEW | Manage

```
1  SELECT start_terminal,
2         duration_seconds,
3         RANK() OVER (PARTITION BY start_terminal
4                      ORDER BY start_time)
5                    AS rank
6    FROM tutorial.dc_bikeshare_q1_2012
7   WHERE start_time < '2012-01-08'
```

	start_terminal	start_time	duration_seconds	rank
1	31000	2012-01-01 15:32:00	74	1
2	31000	2012-01-02 12:40:00	291	2
3	31000	2012-01-02 19:15:00	520	3
4	31000	2012-01-03 07:22:00	424	4
5	31000	2012-01-03 07:22:00	447	4
6	31000	2012-01-03 12:32:00	1422	6
7	31000	2012-01-04 17:36:00	348	7
8	31000	2012-01-05 15:13:00	277	8
9	31000	2012-01-05 17:25:00	3340	9
10	31000	2012-01-06 07:28:00	414	10
11	31000	2012-01-06 07:28:00	398	10
12	31000	2012-01-06 11:36:00	399	12
13	31000	2012-01-06 11:36:00	412	12
14	31000	2012-01-06 17:29:00	2661	14
15	31000	2012-01-06 22:10:00	393	15

Page 1 of 1 | Showing 1-100 of 100 results

How to find duplicate records?

How to find a duplicate record?

1. duplicate records with one field
2. duplicate records with more than one field

1. duplicate records with one field

```
SELECT name, COUNT(email)
FROM users
GROUP BY email
HAVING COUNT(email) > 1
```

2. duplicate records with more than one field

```
SELECT name, email, COUNT(*)
FROM users
GROUP BY name, email
HAVING COUNT(*) > 1
```

Making Queries Efficient*

- Define business requirements first (what exactly is needed)
- SELECT fields instead of using SELECT *
- Avoid SELECT DISTINCT (large amount of processing power is required)
- Create joins with INNER JOIN (not WHERE)

Making Queries Efficient*

- Use WHERE instead of HAVING to define filters (fiter it before you use HAVING)
- Use wildcards at the end of a phrase only (LIKE ‘Char%’ instead of ‘%Char%’)
- Use LIMIT to sample query results.
- Run queries off-peak hours on large sets: talk to your DBA.

Resources

- Advanced SQL for Interview preparation: https://quip.com/2gwZArKuWk7W
- Window Functions: https://www.postgresql.org/docs/9.1/tutorial-window.html
- Multiple tables joins https://academy.vertabelo.com/blog/illustrated-guide-multiple-join/
- SQL query efficiency for production: https://www.sisense.com/blog/8-ways-fine-tune-sql-queries-production-databases/
- Good tutorial https://www.javatpoint.com/sql-tutorial
- Practice platform https://pgexercises.com/
- Practice platform: https://mode.com/sql-tutorial/sql-sub-queries/
- Interactive tutorial: https://sqlbolt.com/topic/subqueries
- SQL details https://en.wikipedia.org/wiki/SQL
- SQLite in Python https://realpython.com/python-sql-libraries/
- Query planning https://sqlite.org/queryplanner.html
- SQL in Python https://stackoverflow.com/questions/44981986/sqlalchemy-er-diagram-in-python-3