<https://drive.google.com/drive/folders/1moeWYoUtUklJO6NJdWo9OV8zWjRn0rjN>

<https://lukeb.co/sql_jobs_db>

*Mainly "a query" includes CRED (Create, Read, Update, Delete).*

*SQL can run locally or on servers (Physical servers - On-premises, Cloud - Serverless).*

*Databases - Relational (tables, etc) structured with interrelated values (rows and tables), Non-Relational (NoSQL -- not only sql) - (key value pairs, graphs, docs, etc) unstructured data.*

*Examples of Relational Database management services (DMS) - PostgreSQL, MySQL, SQLite, MS SQL Server.*

*Non-relational - MongoDB.*

*Editors can be standalone like Visual Code or inbuilt into a DMS like PostgreSQL.*

*Database -- ERD (Entity Relationship Diagram).*

*Main Fact table including everything I.E. core data -- Dimension (dim) tables including subset info, describe attributes or dimensions of data and they support filtering, grouping, etc.*

*Schema means the pattern of arrangement of tables, columns, etc inside a database.*

**SQL Imp Keywords (all keywords in SQL are upper CAPS):**

* **SELECT & FROM** (Used to import info and check access)

Example,

***SELECT \****

***FROM xyz***

this means select the whole data (\*) inside (from) xyz table.

***SELECT***

***abc\_pkg,***

***glh\_kip***

***FROM xyz***

Means only select abc\_pkg and glh\_kip columns from the xyz dataset

In case of multiple tables with same column names you need to use:

***SELECT***

***xyz.abc\_pkg,***

***xyz.glh\_kip,***

***FROM xyz***

* **LIMIT** (It’s the last statement of the whole SQL code, Limits the output to X rows to reduce retrieval times)

Example,

***SELECT***

***xyz.abc\_pkg,***

***xyz.glh\_kip,***

***FROM xyz***

***LIMIT 5***

This will limit the output to only first 5 rows.

* **DISTINCT** (Gives distinct values)

***SELECT DISTINCT***

***xyz.abc\_pkg***

***FROM xyz***

***LIMIT 5***

This will give unique values in abc\_pkg column, if you add another line to the select block, distinct will give unique “pairs” of key value pairs, example, XYZ – ABC is twice in the column then it’ll give only one in output but it won’t remove XYZ – KIP or GLH – ABC.

* **WHERE** (Adds condition)

***SELECT DISTINCT***

***xyz.abc\_pkg***

***FROM xyz***

***WHERE***

***abc\_pkg > 1234 or abc\_pkg = ‘SOMETHING’***

***LIMIT 5***

Only gives rows which match the condition for the abc\_pkg column. Strictly single quotes ‘ ’ not “ “

* **COMMENTS** (-- for single line comment, /\*text\*/ for multi-line)

***SELECT \* -- this is single line comment***

***/\* This is multi line***

***Comment***

***\*/***

* **ORDER BY** (orders by a particular requirement)

***ORDER BY***

***abc\_pkg ASC/DESC***

Gives the output ordered in ascending or descending (one of the two) order for abc\_pkg column.

**Semi colon (;) is used to denote end of on query block, you get multiple output in this case as per your queries.**

**ORDER TO WRITE COMMANDS**

SELECT column1, column2

FROM table\_name

WHERE condition

GROUP BY column

HAVING condition

ORDER BY column1 ASC/DESC

LIMIT number;

Comparison operators – “<>”, “=”, “<=”, “>=”, “>”, “<”

Logical Operators – “AND”, “OR”, ”BETWEEN”, ”IN”, “()”

“()” – Priority operator, it makes the whole code inside the brackets run before whatever is next to it.

**“<>” – NOT operator, it excludes the value within the output, usually used with WHERE.**

WHERE

abc\_pkg <> ‘XYZ’

this will not allow XYZ to be in the output.

**BETWEEN – gives values between the given values.**

BETWEEN 500 AND 600

Gives values including and above 500 but including and less than 600.

**IN –** (Tells which values only to include in the output)

WHERE

abc\_pkg IN (‘XYZ’, ‘KIP’, ‘LTI’)

Gives values inside abc\_pkg which match the bracket values.

Practice Prob 1:

Get job details (title\_short, location, source, salary) for ‘Data Analyst’ or ‘Business Analyst’ roles.

For DA, job salary above 100K

For BA, job salary above 70K

Only include job in either ‘Boston, MA’ or ‘Anywhere’

SELECT

    job\_title\_short,

    job\_location,

    job\_via,

    salary\_year\_avg

FROM

    job\_postings\_fact

WHERE

    job\_location IN ('Boston, MA', 'Anywhere')

    AND

    (

        (job\_title\_short = 'Data Analyst' AND salary\_year\_avg > 100000)

        OR

        (job\_title\_short = 'Business Analyst' AND salary\_year\_avg > 70000)

    )

**Wildcards (used within the WHERE clause and with the LIKE operator)**

“%”, “\_”

WHERE

    Job\_title LIKE ‘%Analyst%’

This will give output where analyst is included in the title anywhere, % represents zero, one or more characters. If at the end we don’t add % then it’ll give output where Analyst is at the end since we don’t want any characters after it.

WHERE

    Job\_title LIKE ‘%Business\_Analyst%’

Gives out output where Business Analyst is exactly as it is, with a space in between. \_ is used to denote space while allowing other wildcards to function.

**AS (Alias) –** Allows us to replace the given names as per our will

SELECT

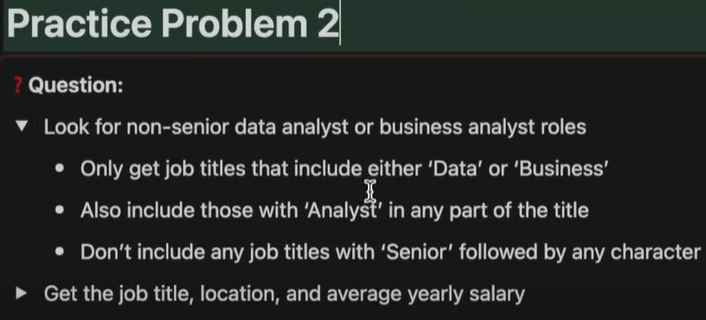
    jpc.job\_title AS jt

    jpc.job\_location AS jl

FROM

    job\_postings\_fact AS jpc

Practice Problem:



SELECT

    job\_title, job\_location, salary\_year\_avg

FROM

    job\_postings\_fact

WHERE

    job\_title NOT LIKE 'Senior%'

AND

    (job\_title LIKE '%Data%' or job\_title LIKE '%Business%')

AND

job\_title LIKE '%Analyst%'

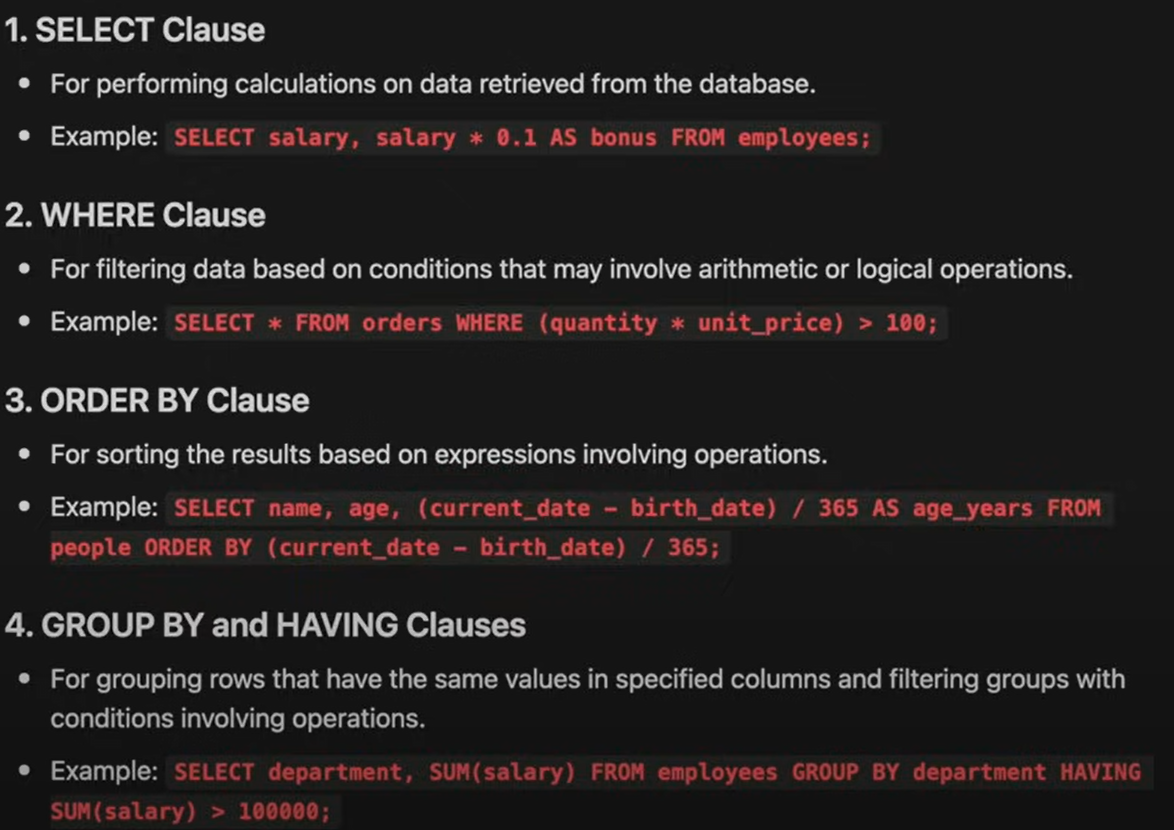
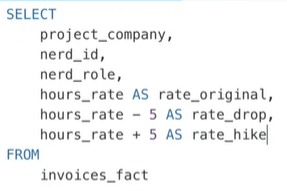
**NULL – Denotes lack of existence of anything. (Differs from 0 and “ “)**

WHERE

Salary IS NULL

Gives values where salary is NULL.

**Arithmetic Operations**



**Aggregation Functions**

***SUM(), COUNT(), AVG(), MAX(), MIN()***

SELECT

    job\_location, salary\_year\_avg, SUM(salary\_year\_avg) AS sum

FROM

    job\_postings\_fact

GROUP BY job\_location HAVING sum > 10000

Gives sum of salary grouped by locations

SELECT

    AVG(salary\_year\_avg) AS salary\_avg, MIN(salary\_year\_avg) AS salary\_min,

    MAX(salary\_year\_avg) AS salary\_max

FROM job\_postings\_fact

WHERE job\_title\_short = ‘Data Analyst’

Gives avg, max and min of yearly salary for data analyst roles.

SELECT

    job\_title\_short AS jobs, COUNT(job\_title\_short),

AVG(salary\_year\_avg) AS salary\_avg, MIN(salary\_year\_avg) AS salary\_min,

    MAX(salary\_year\_avg) AS salary\_max

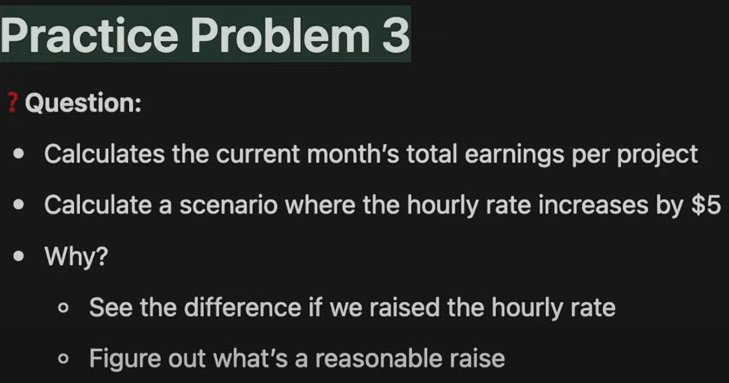
FROM job\_postings\_fact

GROUP BY jobs

Gives avg, max and min grouped by job titles. Count gives number of occurrences of the given input (row value) in the whole column. GROUP BY automatically only gives unique values. COUNT(\*) means count the number of rows in the available columns.

**HAVING has to be used for aggregates and aggregates need HAVING, won’t work with WHERE.**

Practice Problem:



SELECT project\_id,

    SUM(hours\_spent \* hours\_rate) AS original\_cost,

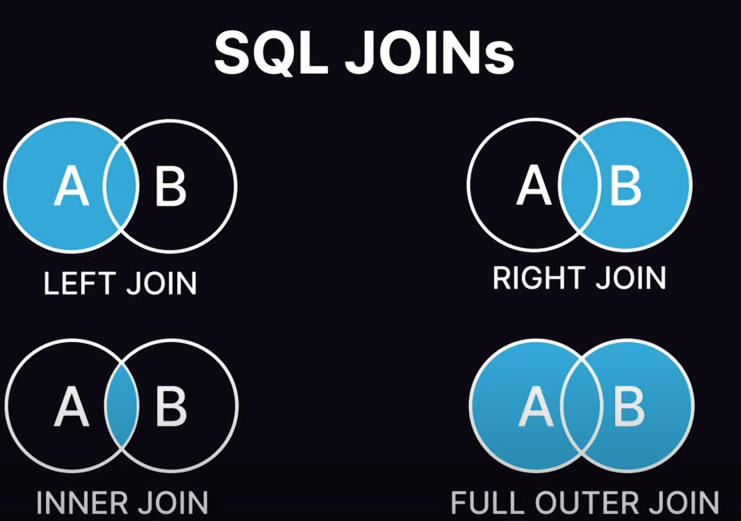
        SUM(hours\_spent \* (hours\_rate + 5)) AS projected\_cost

FROM invoices\_fact

GROUP BY

    project\_id

**JOINS**



1. **LEFT (OUTER) JOIN**

SELECT jobs.job\_id, jobs.job\_title\_short AS Job\_title, comps.name AS company

FROM job\_postings\_fact AS jobs

LEFT JOIN company\_dim AS comps ON jobs.company\_id = comps.company\_id

Gives us list of jobs and the respective company names while connecting the common table (in this case company\_id) while having company\_dim as reference to compare with job\_postings\_fact. It’ll include all the rows data in company\_dim even if it’s not there in fact sheet.

We don’t have company name in the fact sheet, so we connect it to the table which does have it via a common link and JOIN. **Left and right side tables are decided based on the FROM and JOIN command, if we having FROM A JOIN B then it means left side table is A and right side is B. Also try to keep the same sequence in the ON command to avoid confusion.**

1. **RIGHT (OUTER) JOIN**

SELECT jobs.job\_id, jobs.job\_title\_short AS Job\_title, comps.name AS copmany

FROM job\_postings\_fact AS jobs

RIGHT JOIN company\_dim AS comps ON jobs.company\_id = comps.company\_id

This will just take fact sheet table as reference and include all the rows data in that table along with whatever matches we have from company\_dim table, this the above example it doesn’t matter if we use left or right JOIN, the results are the same because we have same row entries in both tables although right JOIN takes longer because we’re including all values from the fact sheet table.

**We specify the table next to JOIN which is not imported through FROM, so if we take FROM A then we must use JOIN B for it work, JOIN A won’t work.**

1. **INNER JOIN**

SELECT jobs.job\_id, jobs\_job\_title

FORM job\_postings\_fact AS jobs

INNER JOIN skills\_job\_dim AS skills\_to\_job ON jobs.job\_id = skills\_to\_job.job\_id

INNER JOIN skills\_dim AS skills ON skills\_to\_job.skill\_id = skills.skill\_id

This first joins fact sheet to skills\_job dim and gives the common values (e.g. job\_id 3,4 don’t have any skill\_id associated with them in the skills\_job dim table so we don’t get 4,5 job\_id values in the output). It then joins the skill\_id from skills\_dim to skills\_job dim wherein skill\_id is linked with actual skill name (e.g. skill\_id 4, 5 don’t have any entry in the skills\_job dim but they are there in skills\_dim, but still they won’t be included in the output since only common stuff is included).

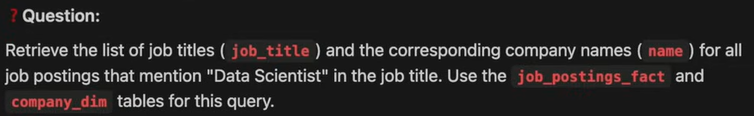
So basically, get job\_id from fact sheet and match with job\_id from skills\_job dim and give only common ones as output, then match skill\_id from skills\_job dim to skill dim and only give common skill\_id as output. Job\_id 3, 4 don’t have any entry in skills\_job dim so they are excluded, then skill\_id 4, 5 don’t have any entry in skills dim so they are excluded as well.

**Columns or sequence of columns is not affected by JOIN, only the rows are, if there is some data in left side which is not on right side then LEFT JOIN will include the data in the rows but RIGHT JOIN won’t, that’s it.**

1. **FULL OUTER JOIN**

Not that important, just includes all the data from both tables.

Practice Problems:



SELECT

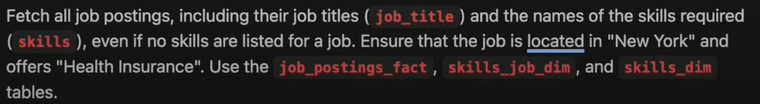
    company\_dim.company\_id,company\_dim.name, jobs.job\_title

FROM job\_postings\_fact AS jobs

LEFT JOIN company\_dim ON jobs.company\_id = company\_dim.company\_id

WHERE

    jobs.job\_title LIKE '%Data Scientist%'



SELECT

    jobs.job\_id, jobs.job\_title, skills\_dim.skills

FROM job\_postings\_fact AS jobs

RIGHT JOIN skills\_job\_dim ON jobs.job\_id = skills\_job\_dim.job\_id

INNER JOIN skills\_dim ON skills\_job\_dim.skill\_id = skills\_dim.skill\_id

WHERE

jobs.job\_location = 'New York'

AND

jobs.job\_health\_insurance = 1

**Order of Execution (for most efficient and logical execution) #NOT THE ORDER OF WRITING#**

FROM/JOIN

WHERE

GROUP BY

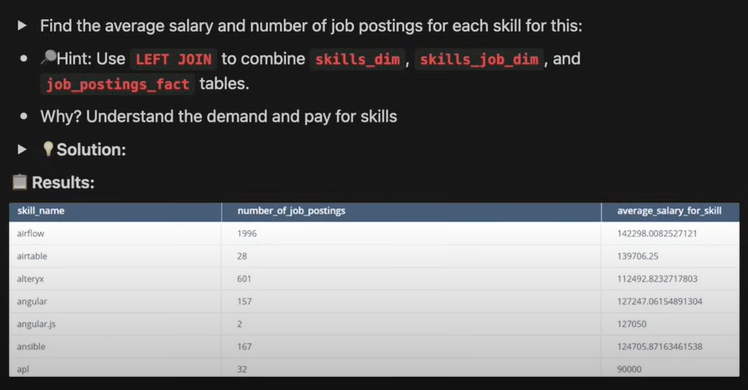
HAVING

SELECT

DISTINCT

ORDERBY

LIMIT/OFFSET



SELECT skills\_dim.skills, COUNT(jobs.job\_title), AVG(jobs.salary\_year\_avg)

FROM job\_postings\_fact AS jobs

LEFT JOIN skills\_job\_dim ON skills\_job\_dim.job\_id = jobs.job\_id

LEFT JOIN skills\_dim ON skills\_dim.skill\_id = skills\_job\_dim.skill\_id

GROUP BY skills\_dim.skills

HAVING skills\_dim.skills IS NOT NULL

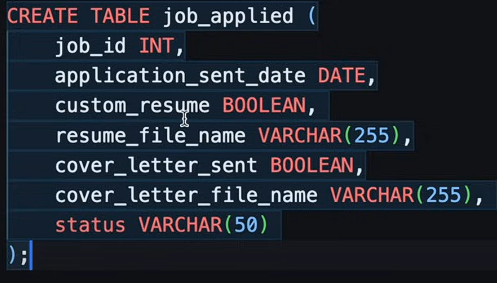
Here if we use INNER join then we don’t need to use HAVING, since INNER join will not take any jobs without any associated skill.

**Common Data Types**

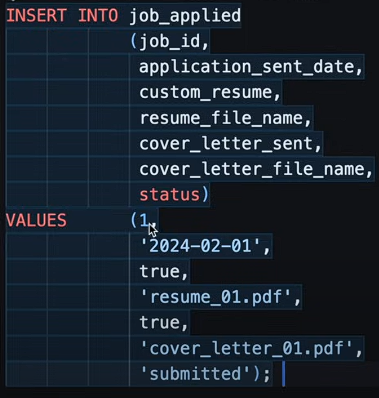
1. **INT – Integers only no decimals.**
2. **NUMERIC(precision, scale) – Allows decimals upto the given value of scale, precision is total number of digits (including decimals). E.g. 123.456 is NUMERIC(6, 3)**
3. **TEXT – Strings with unlimited length**
4. **VARCHAR(n) – String with max limit of ‘n’ characters.**
5. **BOOLEAN – Binary/True or False values**
6. **DATE – YYYYMMDD format**
7. **TIMESTAMP – YYYYMMDD HH:MI:SS**
8. **TIMESTAMP WITH TIME ZONE – YYYYMMDD HH:MI:SS+HH:MI (TIMEZONE offset)**

**Manipulating a Table (Need to use ; semicolon after every such entry)**

1. **CREATE TABLE –** Add columns names followed by their data type.

****

1. **INSERT INTO –** Adds columns to the table (basically you select the columns first then in values bracket you enter the respective values you want to add to those columns. INSERT INTO and VALUES both combined form one entry so semicolon only at the end of values bracket.

****

1. **ALTER TABLE –** Allows changing of table, followed by indented commands:

* **ADD –** Add columns

****

* **RENAME COLUMN**

****

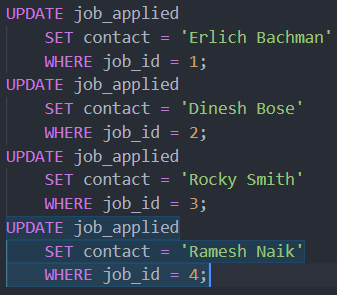
* **ALTER COLUMN –** Change the datatype of a column

****

* **DROP COLUMN –** Delete column

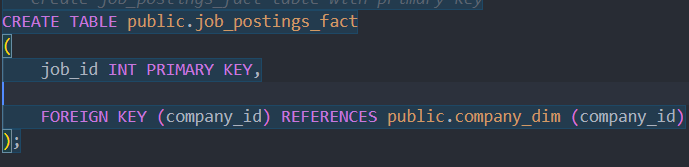
****

1. **Update table\_name SET column\_name = ‘value’ WHERE condition**

****

1. **DROP TABLE –** Delete tables (DROP TABLE job\_applied)

**Primary Key and Foreign Key**



Primary key means the column which is unique or central to the table, for example, for above table the primary key is job\_id because that is what is used as reference for other columns, generally primary key is the first column. The company\_id is a foreign key that is referenced to the company\_dim table where company\_id is primary.

So inside CREATE TABLE bracket, column\_name DATA\_TYPE PRIMARY KEY (if it is the primary key), for foreign key, FOREIGN KEY (column\_name) REFERENCES table\_name (column\_name).

**Casting (::)**

:: (double colons) – **Casting** (used to convert one data type into another)

SELECT timestamp\_column::DATE AS date\_column, ‘123’::INTEGER, ‘true’::BOOLEAN, ‘3.14’::REAL

This will import the entered values into the specified format if possible, e.g. 123 can’t be converted into string/text.

SELECT job\_posted\_date::DATE AS date FROM job\_postings\_fact;

Converts 2023-04-27 17:37:45 into 2023-04-27.

SELECT job\_posted\_date AT TIME ZONE 'UTC' AT TIME ZONE 'IST' FROM job\_postings\_fact

Converts timestamp without timezone into timestamp with adjust time for specified timezone.

SELECT EXTRACT(MONTH FROM job\_posted\_date) AS month FROM job\_postings\_fact

Extracts month/year/day (numerical values in YYYY/MM/DD format) from the date/timestamp.

**Practical Example**

SELECT COUNT(job\_id),

    EXTRACT(MONTH FROM job\_posted\_date) AS month

FROM job\_postings\_fact

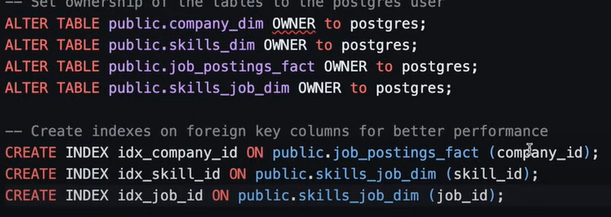
WHERE job\_title\_short = 'Data Analyst'

GROUP BY month

ORDER BY month ASC;

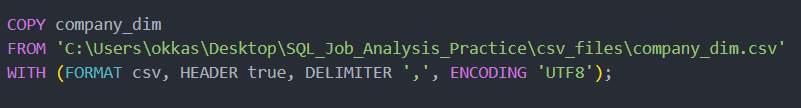
Shows us the trend with total count of data analyst roles throughout the months.

**Miscellaneous Stuff**

****

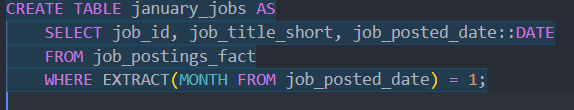
Setting **OWNER** to main username to allow editing access for that user to this database, using ‘public’ before the table name allows public visibility and usage access to all users, not using public specifically will still create the table in public space so it is mostly unnecessary.

**INDEX** command indexes the foreign key columns for better retrieval, not necessary for small databases.



**COPY** table\_name FROM ‘csv file path’ WITH (csv format, first row header is there?, delimiter ‘,’ means csv file has tables separated by ‘,’ so use that while importing, encoding is UTF8).

**CREATE TABLE from referenced data**



This creates tables with the given values where the month column = 1 that is extracted from job\_posted\_date column. When using nested commands, whatever you choose to SELECT will be the data that is used to run the command. For example, here we are selecting job\_id, title and date so that all will be added as columns into the created table, if we only choose certain columns then only those will be added. Don’t forget to add the AS after table name.

**CASE** Expression – A way to apply conditional logic

SELECT

    job\_title\_short,

    job\_location,

    CASE

        WHEN job\_location = 'Anywhere' THEN 'Remote'

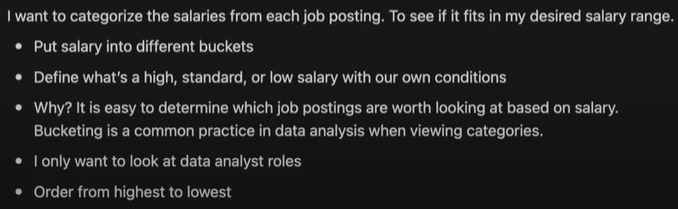
        ELSE 'Onsite'

    END AS location\_category

FROM job\_postings\_fact;

Gives us a new column named location\_category, first look if job\_location matches ‘Anywhere’ if yes then row value is ‘Remote’ else ‘Onsite’, inside that column. You can add any number of WHENs inside CASE, CASE – END is considered one block, basically functions like IF ELSE.

**Practice Problem:**



SELECT

    COUNT(job\_id) AS number\_of\_jobs,

    CASE

        WHEN salary\_year\_avg <= 100000 THEN 'Low'

        WHEN salary\_year\_avg BETWEEN 100000 AND 200000 THEN 'Medium'

        WHEN salary\_year\_avg >= 200000 THEN 'High'

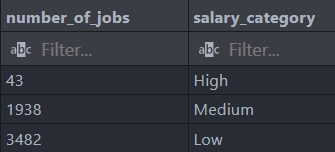
    END AS salary\_category

FROM job\_postings\_fact

WHERE (job\_title\_short = 'Data Analyst' AND salary\_year\_avg IS NOT NULL)

GROUP BY salary\_category

ORDER BY number\_of\_jobs ASC



Gives count of how many jobs under title of ‘Data Analyst’ come under each of the defined salary categories. It helps us understand the distribution of data.

**Subqueries – Query nested inside another query – Simpler usage**

SELEct \* -- Main Query starts

FROM ( *-- Subquery starts here*

    SELECT \*

    FROM job\_postings\_fact

    WHERE EXTRACT(MONTH FROM job\_posted\_date) = 1)

    AS january\_jobs; *-- Subquery ends here – Main Query Ends*

Shows us data of only January month without having to create a separate permanent table inside database. Basically, SELECT everything FROM (SELECT all data FROM job\_postings\_fact WHERE job month is 1) and name it AS january\_jobs. Generally used in SELECT and FROM block.

Used in – SELECT, FROM, WHERE, HAVING.

**Common Table Expressions (CTEs) – Define a temporary result set – Complex use**

**Defined with WITH**

WITH january\_jobs AS ( *-- CTE definition starts here*

    SELECT \*

    FROM job\_postings\_fact

    WHERE EXTRACT(MONTH FROM job\_posted\_date) = 1

    ) *-- CTE definition ends here*

SELECT \*

FROM january\_jobs;

Starts with WITH, enter the temporary table name then define its value by AS (SELECT FROM WHERE block) then use SELECT and FROM temporary table name. This allows flexibility to use temporary table name which can be used as references for further operations, basically like def() function.

Used in – SELECT, INSERT, UPDATE, DELETE.

**Subquery -** Let’s say we want to use company name column from company\_dim table while adding condition that job\_no\_degree\_mention is true from the job\_postings\_fact table.

We can do this in two ways 1. JOIN 2. Subquery

1. JOIN

SELECT DISTINCT name AS company\_name

FROM company\_dim

INNER JOIN job\_postings\_fact

ON company\_dim.company\_id = job\_postings\_fact.company\_id

WHERE job\_postings\_fact.job\_no\_degree\_mention = true

~~(This will not give distinct company names since it is checking based on company id and if job\_postings\_fact has more jobs with same company id so it is taking them multiple times.)~~

Just use DISTINCT while using joins when necessary.

1. Subquery

SELECT name AS company\_name

FROM company\_dim

WHERE company\_id IN (

    SELECT

        company\_id

    FROM job\_postings\_fact

    WHERE job\_no\_degree\_mention = TRUE

)

Both give same results. Here IN operator will automatically remove the duplicates.

**CTE -** Let’s say we want to find the companies with their number of job openings, if no openings then 0.

1. JOIN

SELECT COUNT(job\_postings\_fact.job\_id) AS number\_of\_jobs,

    company\_dim.name AS company\_name

FROM job\_postings\_fact

RIGHT JOIN company\_dim ON job\_postings\_fact.company\_id = company\_dim.company\_id

GROUP BY company\_name

Joins company dim and job postings on company id then takes all data from company dim and matches from job postings fact, counts the job\_id and retrieves company name, groups by company name.

1. CTE

WITH company\_job\_count AS (

    SELECT

        company\_id, COUNT(\*) AS total\_jobs

    FROM job\_postings\_fact

    GROUP BY company\_id

)

SELECT company\_dim.name AS company\_name,

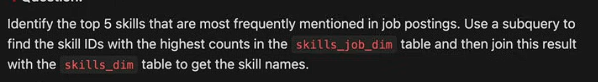
    company\_job\_count.total\_jobs

FROM company\_dim

LEFT JOIN company\_job\_count ON company\_dim.company\_id = company\_job\_count.company\_id

Defines company\_job\_count as, select and group company id and count it from job postings fact. Select company name and total\_jobs from company\_job\_count, join with company dim while keeping all data from company\_dim and matches with job\_count on company\_id.

**Practice Problems:**

****

SELECT skills, COUNT(\*)

FROM (

    SELECT skill\_id

    FROM skills\_job\_dim

    LEFT JOIN job\_postings\_fact ON

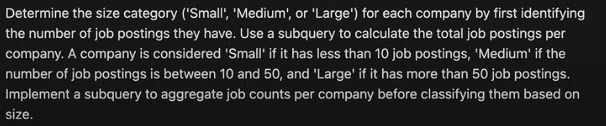
    skills\_job\_dim.job\_id = job\_postings\_fact.job\_id

) AS skill\_id\_list

LEFT JOIN skills\_dim ON skill\_id\_list.skill\_id=skills\_dim.skill\_id

GROUP BY skills

Join skills\_job\_dim and job\_postings\_fact on job\_id then get the skill\_id of whole of skills\_job\_dim and matches from job\_postings\_fact, import it all as skill\_id\_list. Join this with skills\_dim on skill\_id and select skills (names) and their count by grouping based on skills.



SELECT name,

CASE

    WHEN jobs <= 10 THEN 'Small'

    WHEN jobs BETWEEN 10 AND 50 THEN 'Medium'

    WHEN jobs >= 50 THEN 'Large'

END AS company\_category

FROM (

    SELECT name, COUNT(job\_id) AS jobs

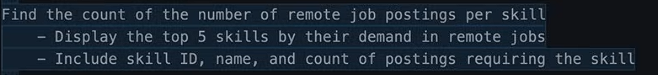
    FROM company\_dim

    INNER JOIN job\_postings\_fact ON

    company\_dim.company\_id = job\_postings\_fact.company\_id

    GROUP BY name

)



WITH jobs\_per\_skill AS (

    SELECT skills AS skill\_name, COUNT(skills\_job\_dim.job\_id) AS total\_jobs

    FROM skills\_job\_dim

    INNER JOIN job\_postings\_fact ON

        skills\_job\_dim.job\_id=job\_postings\_fact.job\_id

    INNER JOIN skills\_dim ON

        skills\_job\_dim.skill\_id=skills\_dim.skill\_id

    WHERE job\_work\_from\_home = TRUE

    GROUP BY skills

)

SELECT skill\_id, skill\_name, total\_jobs

FROM jobs\_per\_skill

INNER JOIN skills\_dim ON jobs\_per\_skill.skill\_name=skills\_dim.skills

ORDER BY total\_jobs DESC

LIMIT 5;

**UNION –** combines results from two or more SELECT statements (No duplicates)

**They need to have the same amount of columns and the data type must match**

SELECT \*

FROM january\_jobs

UNION

SELECT \*

FROM february\_jobs

UNION

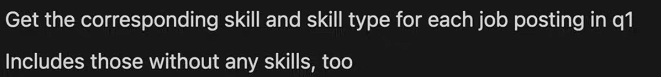
SELECT \*

FROM march\_jobs

Combines all the data from the given tables provided they all have same columns with same data type regardless of what data you’re selecting.

UNION ALL – Does exact same thing as UNION just allows duplicates.

**Practice Problem:**



WITH q1\_jobs AS (

SELECT \*

FROM january\_jobs

UNION

SELECT \*

FROM february\_jobs

UNION

SELECT \*

FROM march\_jobs)

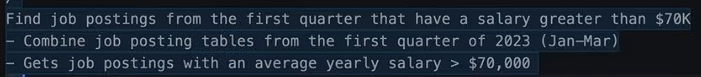
SELECT q1\_jobs.job\_id, job\_title\_short, skills, type

FROM skills\_job\_dim

LEFT JOIN q1\_jobs ON q1\_jobs.job\_id=skills\_job\_dim.job\_id

INNER JOIN skills\_dim ON skills\_job\_dim.skill\_id=skills\_dim.skill\_id

WHERE q1\_jobs.job\_id IS NOT NULL



WITH q1\_jobs AS (

    SELECT \*

    FROM january\_jobs

    UNION

    SELECT \*

    FROM february\_jobs

    UNION

    SELECT \*

    FROM march\_jobs

)

SELECT q1\_jobs.job\_id, q1\_jobs.job\_title\_short, salary\_year\_avg, q1\_jobs.job\_posted\_date::DATE

FROM q1\_jobs

INNER JOIN job\_postings\_fact ON q1\_jobs.job\_id=job\_postings\_fact.job\_id

WHERE salary\_year\_avg > 70000