Data analysis roadmap => (learn it use it) => iterate

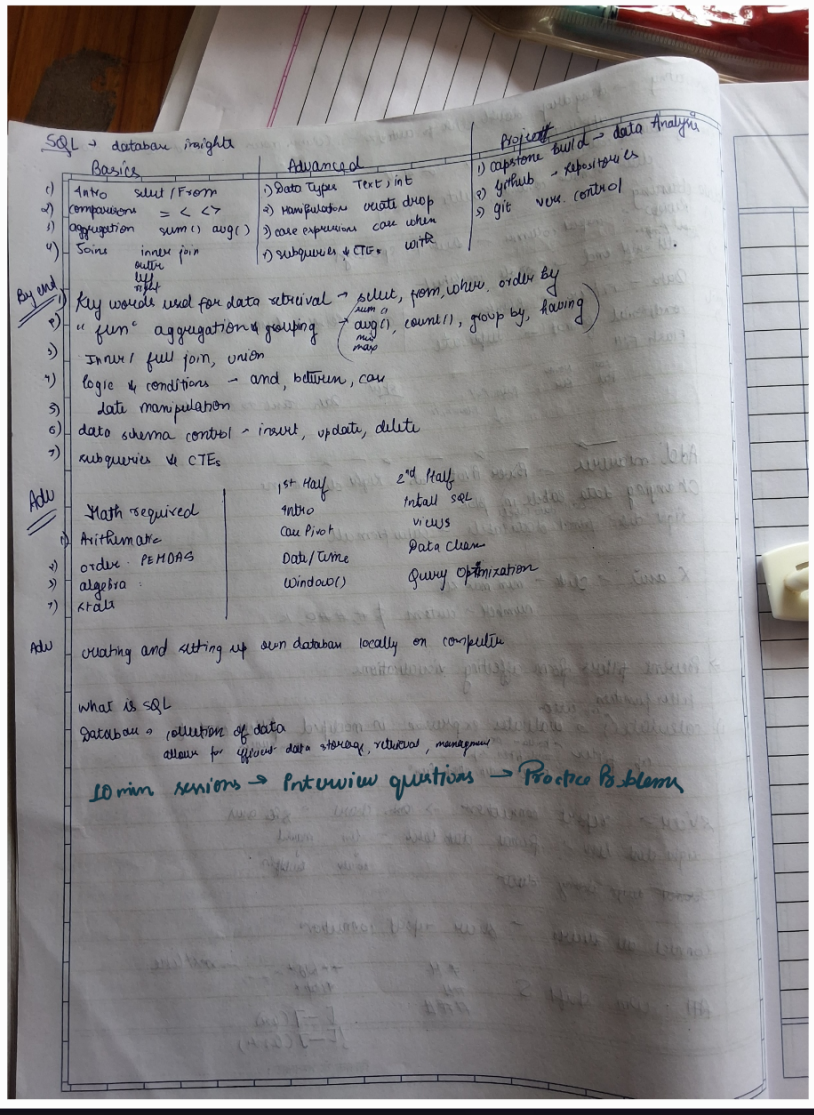
1) Technical skills
2) Domain knowledge
3) Analytical skills => Problem solving, critical thinking, Research, Math skills → Algebra, Prob. stats
4) soft skills

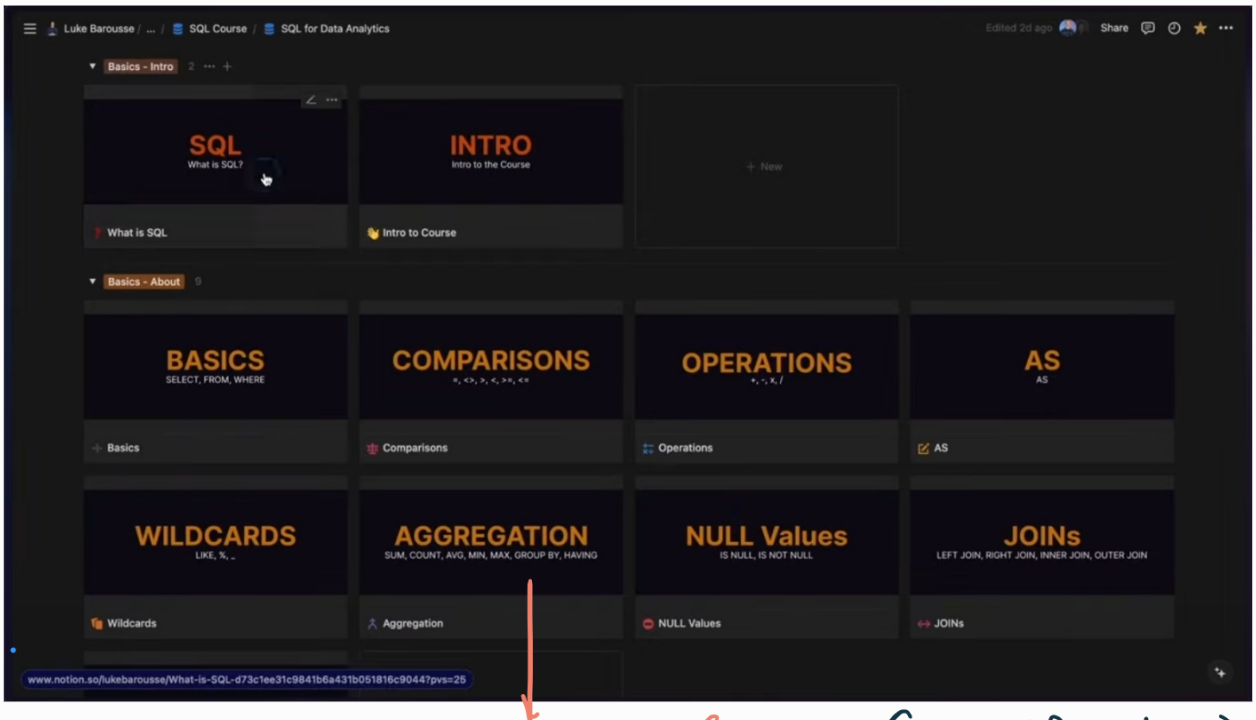
Use skills where

1) course work projects
2) Projects on courses
3) Portfolio projects for resume

How to start

1) Start with technical, incorporate other





select, group by, having (not with where)
order by

Data science journey → is demand in data science

Excel → SQL → python → Power BI

Spreadsheet | Database | programming | Dashboarding

SQL ↻ 6:30 ⇒ 11 hr.

What is SQL → 1) where I will be writing and executing SQL

→ 2) database

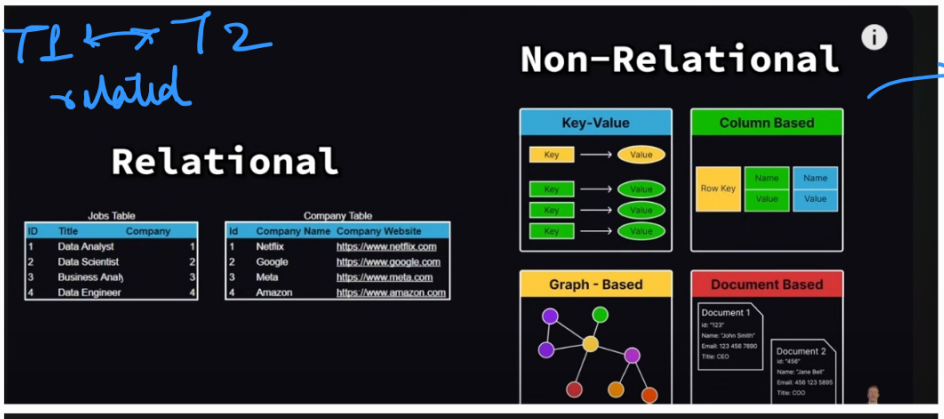
Excel file → million row limit

database → millions of excel files

Databases → relational

not only SQL

T1 ↔ T2 related



→ unstructured NoSQL

linkedIn connections

google docs

What is a query?

A query is a request for data from a database, allowing you to perform various operations:

- **Operations:** Includes creating, reading, updating, and deleting data (CRUD)

Query => request for data from database (CRUD) operations

create | select | update | delete

Databases stored locally on computer or server

What is a database?

A database is a collection of data

- **Structure:** Allows for efficient data storage, retrieval, and management
- **Types of Databases:** Includes relational databases (organized in tables) and non-relational databases (for unstructured data)

company server

on-Prem

(Serverless) cloud servers

aws

google cloud

azure

1) PostgreSQL 2) SQLite → (3/1)

(1,2)

Where to write query

editor → from database provider & cloud provider

- → PostgreSQL, MySQL
- → google cloud, azure
- → code editor → VS code

Manage all SQL files
run queries

SQLite Viz ⇒ open source, browser

① How and where SQL queries are run

DBMS → manage data, execute SQL code

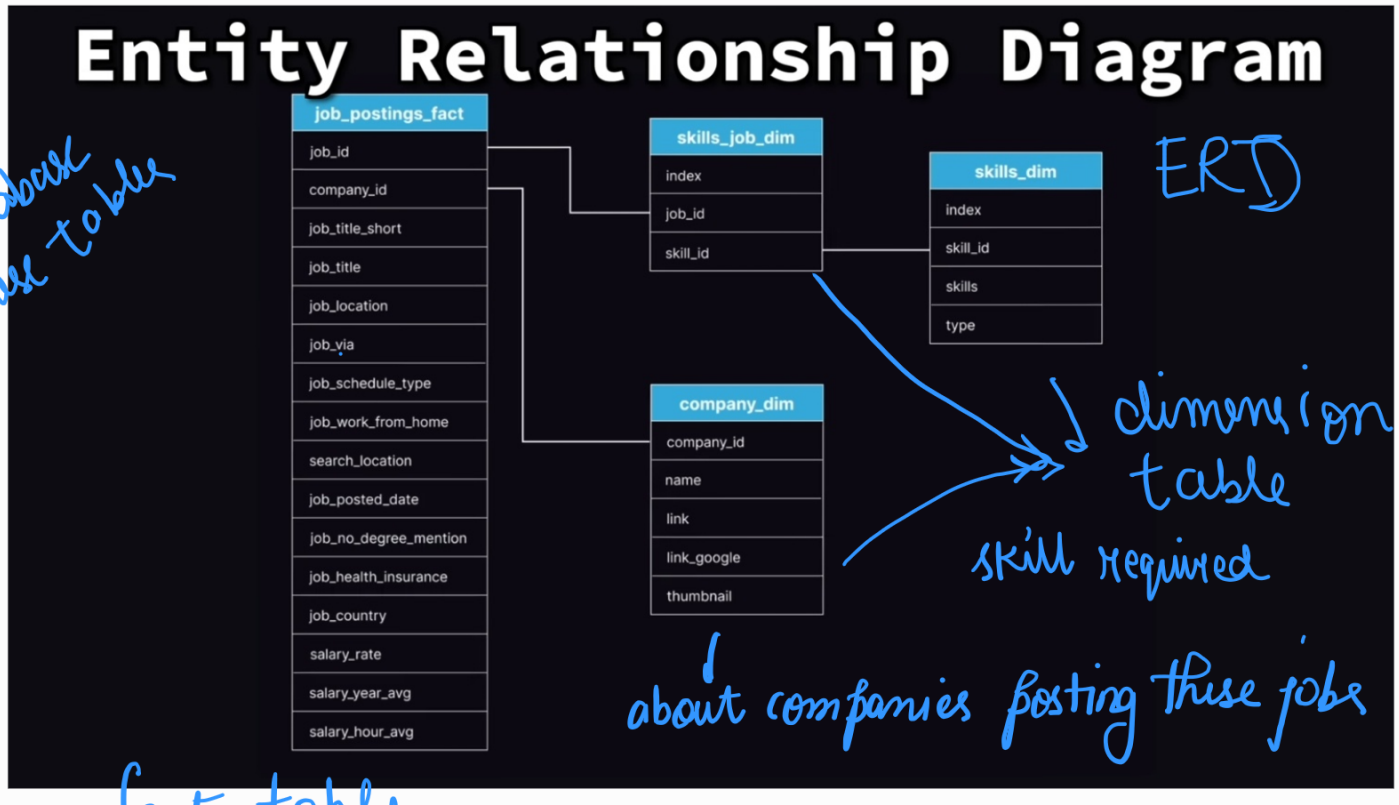
[code/GUI] → [MySQL server] → [Query Engine] → [Storage Engine] → [Database Files]

select *

② what and how databases are used to store data

Intro to course

1) What datasets we will be using
2) running them
3) Question => Job seeker
   Goal => identifying highest paying Jobs
   Optimal skills to have



Fact table => contains core data for business analysis
Measure and record business events (eg job postings)
actual events
high volume of record
usually foreign key to associate to dim table

Dim table :- describes attributes or dimensions of the data
eg (skills, companies)
⇒ Support filtering, grouping and labelling of facts in report
gen:
⇒ Fewer rows of data but more descriptive

- SELECT / FROM
- LIMIT
- DISTINCT
- WHERE
- -- COMMENTS
- /**/ MULTI-LINE COMMENT
- ORDER BY

↗ indentation doesn't matter

↗ querying all can be intensive for servers

① [Select * from Table] :- To see if I have access to it
⇒ run ▶ or ctrl enter
⇒ not case sensitive, Table name depends ↗ SQLite Not Post / Yes
Col name

② [select C1, C2 from Table]
T.C1, T.C2

③ [limit 5] ⇒ at very last

④

DISTINCT

```sql
SELECT DISTINCT
    job_title_short
```

Notes:

- `DISTINCT` - get unique rows
- This usually isn't used for the entire query
  - Why? It takes up a lot of processing power
  - Used within certain SQL functions (more on that later)

; → end of SQL statement

last one shows in SQLite

others in different windows

Select

select ;

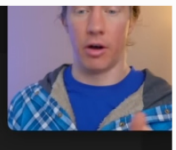
(5) Where right after From

(6) order by default asc

Null is smallest if no values

(7) order

Order to Write Commands



```sql
SELECT column1, column2, ...
FROM table_name
WHERE condition
GROUP BY column
HAVING condition
ORDER BY column1 [ASC|DESC] ...
LIMIT number;
```

- *Note: This is separate from how the order of a query is executed, but instead the order in it must be written*

where NOT C= —
where C<> —

Comparision

where having
BETWEEN AND OR NOT IN

1) NOT <>
2) where C between X_1 and X_2 (included) <=, >=
3) where C in (X_1, X_2, X_3)
   (C = X_1 or C = X_2 . . .)
4) () goes first

Wild Cards

Wildcards

```
3  WHERE
4      job_title LIKE '%Data%'
```

Notes:

- Wildcards - used to substitute one or more characters in a string
- Wildcards are used with the `LIKE` operator
- All of this (i.e., `LIKE`, `%`, and `_`) is used in the `WHERE` clause

→ like Not like

% => 0, 1, more characters %A%D

_ => 1 one, single character

As (can skip) select C_1 as X, C_2 as X ....) from Table (as) y

You C_1
You C_2

Operations

Select C_1 + 5 from T

Example Locations Used:

1. SELECT Clause

- For performing calculations on data retrieved from the database.
- Example: `SELECT salary, salary * 0.1 AS bonus FROM employees;`

2. WHERE Clause

- For filtering data based on conditions that may involve arithmetic or logical operations.
- Example: `SELECT * FROM orders WHERE (quantity * unit_price) > 100;`

3. ORDER BY Clause

- For sorting the results based on expressions involving operations.
- Example: `SELECT name, age, (current_date - birth_date) / 365 AS age_years FROM people ORDER BY (current_date - birth_date) / 365;`

4. GROUP BY and HAVING Clauses

- For grouping rows that have the same values in specified columns and filtering groups with conditions involving operations.
- Example: `SELECT department, SUM(salary) FROM employees GROUP BY department HAVING SUM(salary) > 100000;`

→ works under all

% remainder
How much left after dividing

1) checking even odd 2 % = 0 / = 1) divisibility check
2) ( ) % n → (0 → n−1) cycle through values
   Eg days (current + days ahead) % 7
   (Sat Sun Mon Tue Wed Th Fri)
3) Repeat every 'N' step → for (i in c(1, 11))
   Print (round i)
   if (i %% N == 0)
   Print ('Break')
4) Clock (10am + x hour) %% 12

Modulus keeps value within cycle → (start + step) % max value
ensures value between (0 → max−1)

Operators → across row
Aggregation → down the column

Aggregation

Aggregation Functions: These functions compute a single result from input values.

- `SUM()` : Adds together all values in a specific column.
- `COUNT()` : Counts the number of rows that match a specified criterion.
- `AVG()` : Calculates the average value of a numeric column.
- `MAX()` : Finds the maximum value in a set of values.
- `MIN()` : Finds the minimum value in a set of values.

select
group by
having => (
can't use with where
order by

1) count(*) → Total number of rows in table
2) count(distinct column name)

GROUP BY

```
1  GROUP BY
2      job_location
```

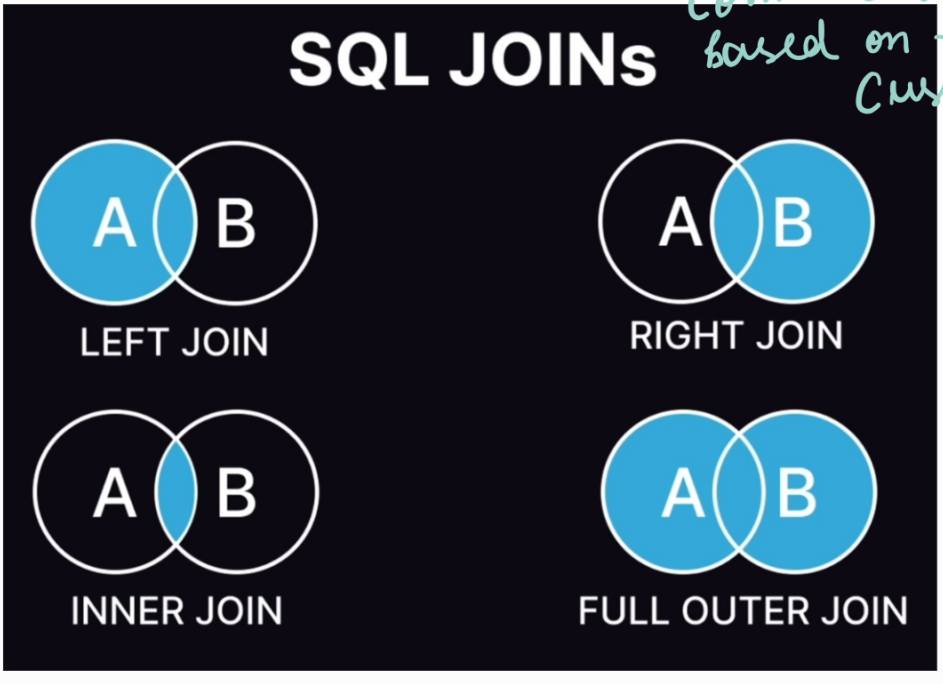
📝 **Notes:**

- `GROUP BY` - groups rows that have the same values into summary rows
- E.g. "total number of sales by each item"
- ⚠️Note: If you have an aggregate function, you need to include every column that's not aggregated (if not it gives an error)

Null ≠ 0 " " → use within where having

SQL JOINs

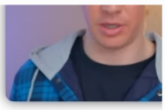
combine rows from 2 or more tables based on related col between them (usually foreign key)



left join

```sql
SELECT
    job_postings.job_id,
    job_postings.job_title_short,
    job_postings.company_id,
    companies.company_id
FROM job_postings_fact AS job_postings
LEFT JOIN company_dim AS companies
    ON job_postings.company_id = companies.company_id
```

```sql
SELECT
  job_postings.job_id,
  job_postings.job_title,
  skills_to_job.skill_id
FROM
  job_postings_fact AS job_postings
INNER JOIN skills_job_dim AS skills_to_job ON job_postings.job_id = skills_to_job.job_id
INNER JOIN skills_dim AS skills ON skills_to_job.skill_id = skills.skill_id
```



job_postings_fact (Table A)

job_id	company_id	job_title
1	1	Data Analyst, Senior
2	2	Data Scientist, Product Analytics
3	3	Senior Data Analyst
4	4	Manager, Business Analyst
5	5	Azure Data Engineer

skills_job_dim (Table B)

job_id	skill_id
1	2
1	3
2	1
2	3
5	1

skills_dim (Table C)

skill_id	skills
1	python
2	r
3	sql
4	scala
5	java

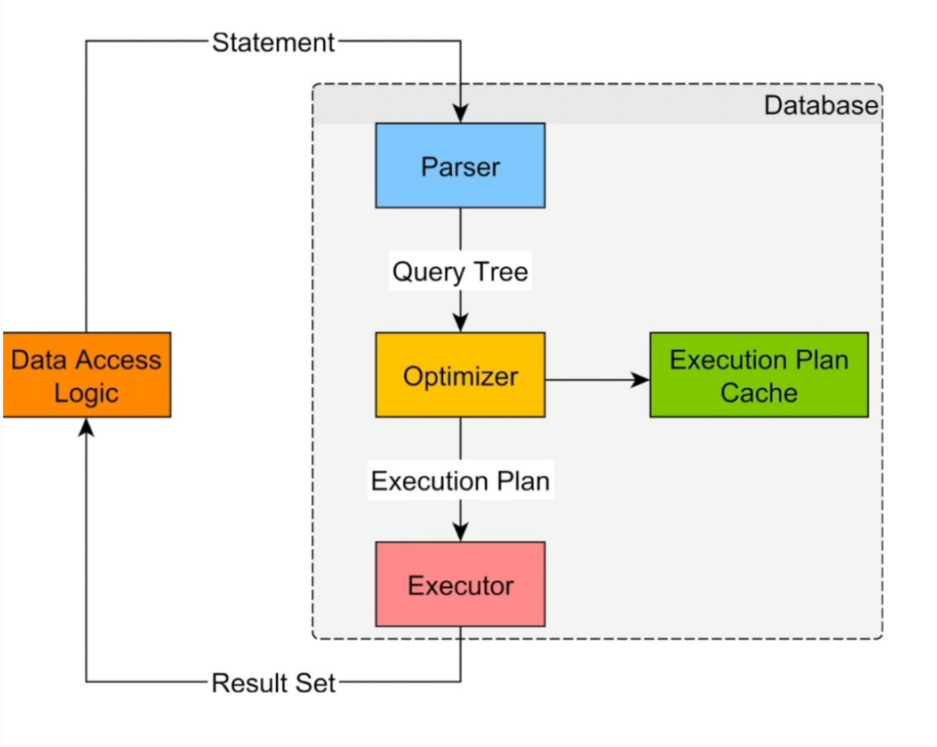
Results Table

job_id	job_title	skills
1	Data Analyst, Senior	r
1	Data Analyst, Senior	sql
2	Data Scientist, Product Analytics	python
2	Data Scientist, Product Analytics	sql
5	Azure Data Engineer	python

full outer join => not used much

Full Order of Execution

1. **FROM / JOIN**
   - Specifies the tables to retrieve data from and how to join them.
2. **WHERE**
   - Filters rows based on conditions.
3. **GROUP BY**
   - Group rows share a property so aggregate functions can be applied.
4. **HAVING**
   - Filters groups based on aggregate conditions (used after `GROUP BY`).
5. **SELECT**
   - Select specific columns to display in the final result.
6. **DISTINCT**
   - Removes duplicate rows from the result set (applied after `SELECT`).
7. **ORDER BY**
   - Sorts the result set based on specified columns/values.
8. **LIMIT / OFFSET**
   - Limits the number of rows returned, often used for pagination.



Why This Order?

- **This sequence ensures SQL queries are processed efficiently and logically.**
- Data sources and joins are defined first; filtering happens early to reduce the amount of data processed, grouping and aggregation follow, then specific fields are selected, and finally, the output is ordered and limited.
- This flow reflects the logical processing order necessary to construct the final dataset and optimization principles databases use to execute queries efficiently.

- Group by, all col in select either in group by or aggregate
- Aggregate functions ignore Nulls except count *

Advance topics => PostgreSQL

Create table
Insert into
alter table
update
Drop table

create - alter - drop

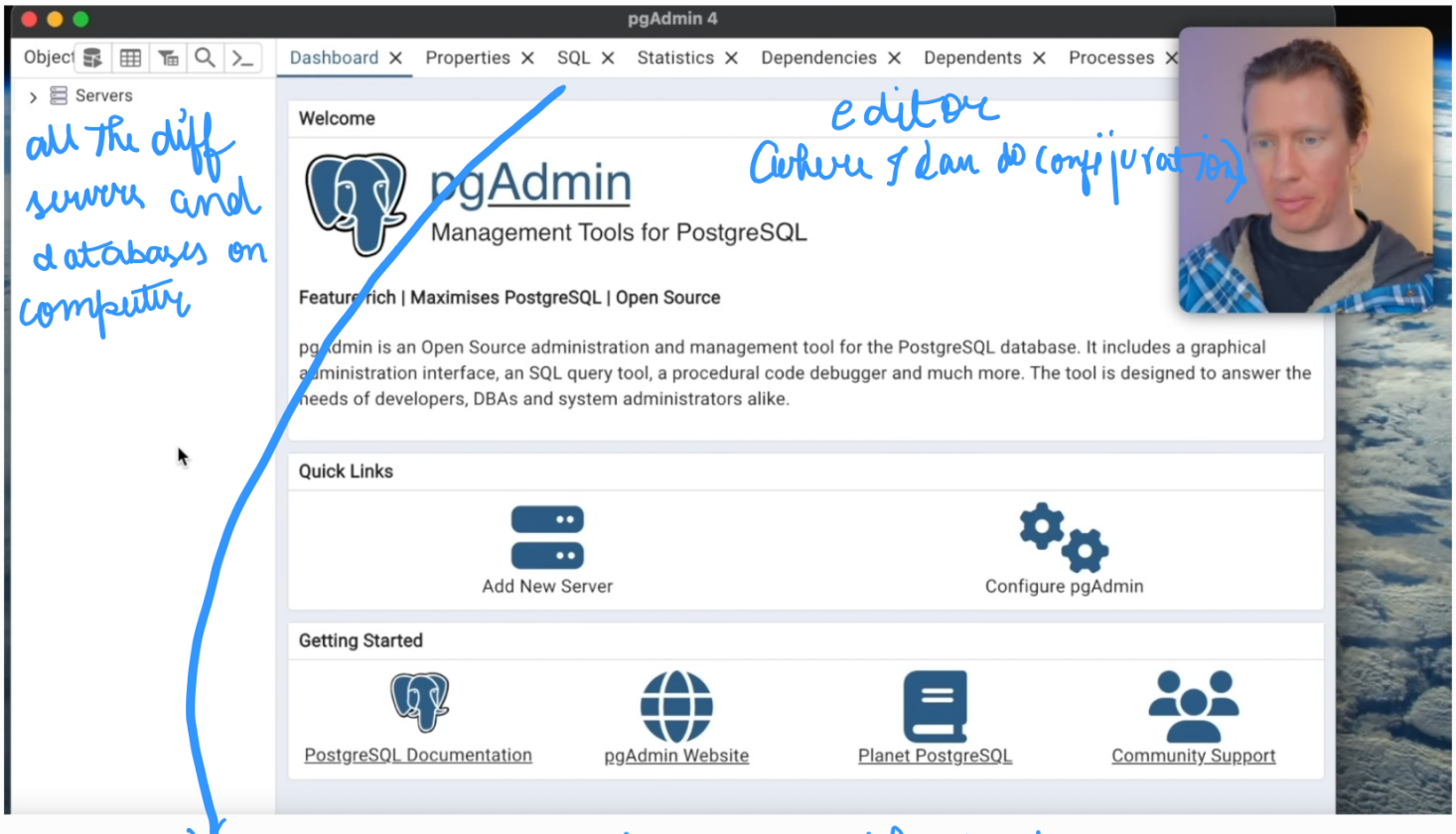
1) Datatypes
2) Manipulate
3) Database load
4) Dates
5) Case expression
6) Subqueries & CTEs
7) union

SQLite vize => good for practicing and learning SQL

VS code => Track different SQL files we are using
Connect to host of different SQL databases

PostgreSQL => most admired and desired

Concepts learned, specifically SQL syntax, can be applied to different databases



all the diff servers and databases on computer

editor
Server & can do configuration

=> can run SQL queries and interact with database
=> but if other like MySQL, SQL server, we will not use pgAdmin
=> have to launch pgAdmin to get database running

VS code

1. What's a code editor (or IDE)?
   a. Code Editor - A digital notepad for writing and organizing programming code. Easy editing, checking, and running code snippets
   b. Integrated Development Environment (IDE) - A supercharged code editor with tools for writing, testing, and fixing code all in one place. Like a code editor with a toolbelt

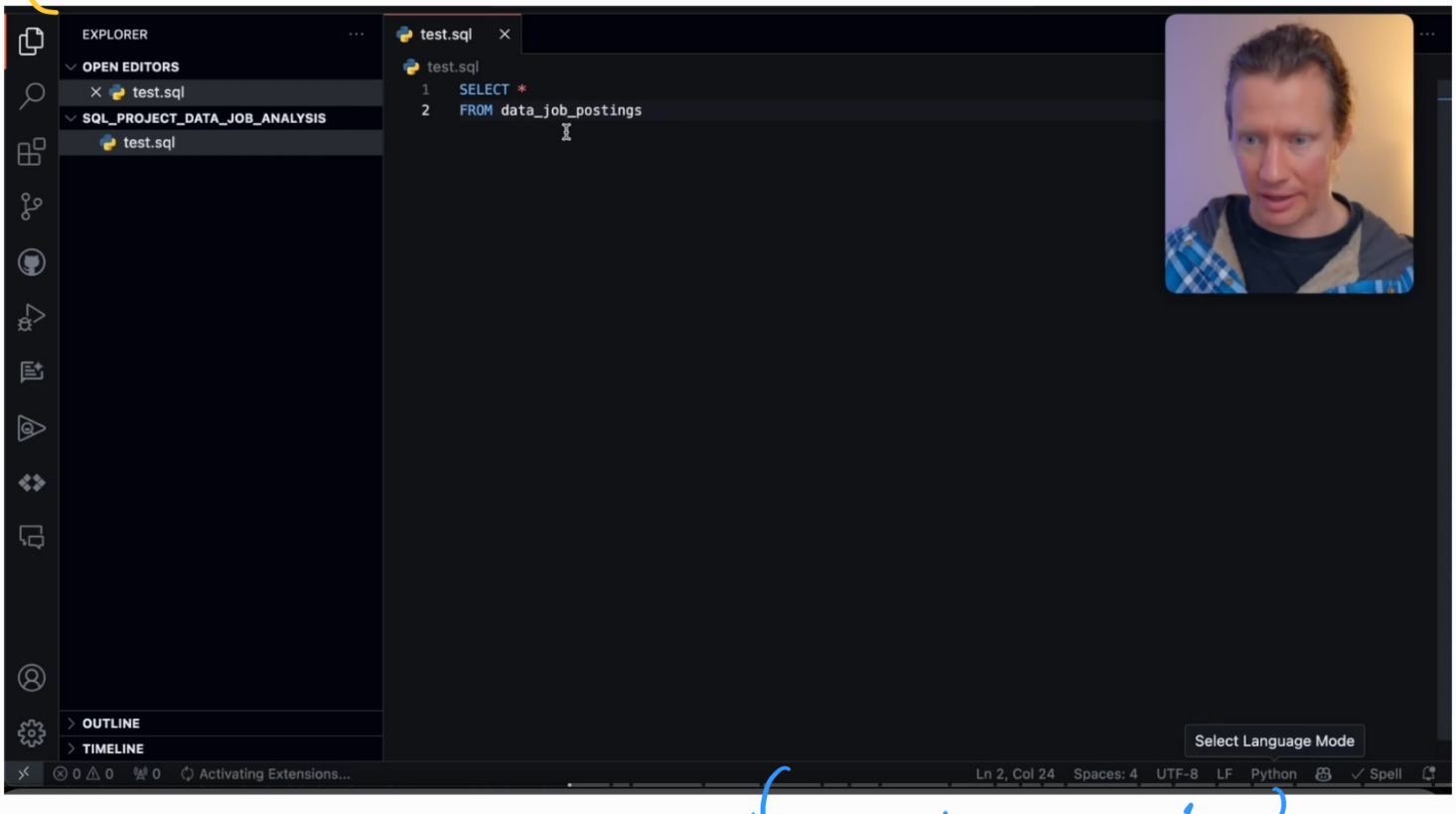
VS Code is a code editor packed with features like an IDE: debugging, syntax highlighting, and extensions for various programming tasks

code editors specific for SQL => DataGrip (paid)

DBeaver (free) cross platform database tool

DBeaver Community is a free cross-platform database tool for developers, database administrators, analysts, and everyone working with data. It supports all popular SQL databases like MySQL, MariaDB, PostgreSQL, SQLite, Apache Family, and more.

more powerful for running SQL queries, can do lot more functionality (.sql files)



right click -> editor language / select the language / SQL

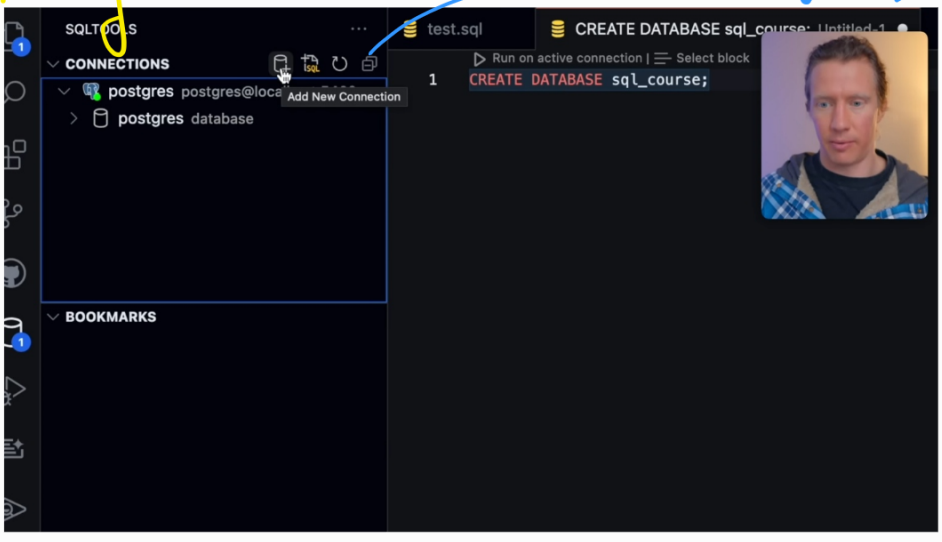
=> how to VS code connect to database

Creating database

Create database sql_course; → Run on active connection
right click → Run selected query
CTRL CTRL

pgadmin → right click PostgreSQL → Refresh

Making connection → postgreSQL → fill



→ Test connection
Save connection

Data Types

INT VARCHAR BOOLEAN TIMESTAMP NUMERIC

job_id	job_title	job_work_from_home	job_posted_date	salary_year_avg
0	Machine Learning Engineer	0	2023-06-17 00:00:00	101029
1	Data Center Engineer	0	2023-04-27 00:00:00	82500
2	Business Data Analyst	0	2023-11-09 00:00:00	54988
3	Data Scientist	0	2023-07-22 00:00:00	157500
4	Data Scientist Job at Belgium	1	2023-12-22 00:00:00	NULL
5	Data Entry Specialist (Part-Tim...	1	2023-12-27 00:00:00	NULL

→ can have decimal place than 'numeric'

About Types

- Data types specify the expected format of a value.
- Why?
  - These are needed to be specified for **data integrity** to ensure only the correct data is stored in a column.
  - They also help databases process data **more quickly** and store it with **less space**.

=> first line of defense for clean data

🌟Common Examples

- `INT` => whole no (-2,147,483,648 2,147,483,647) for regular int (4 bytes)
- `NUMERIC(precision, scale)` (total no of digits , after .)
- `TEXT` => variable-length strings with unlimited length)
- `VARCHAR(n)`
- `BOOLEAN` => T, F, null
- `DATE` (YYYY-MM-DD)
- `TIMESTAMP` " HH:MM:SS
- `TIMESTAMP WITH TIME ZONE`

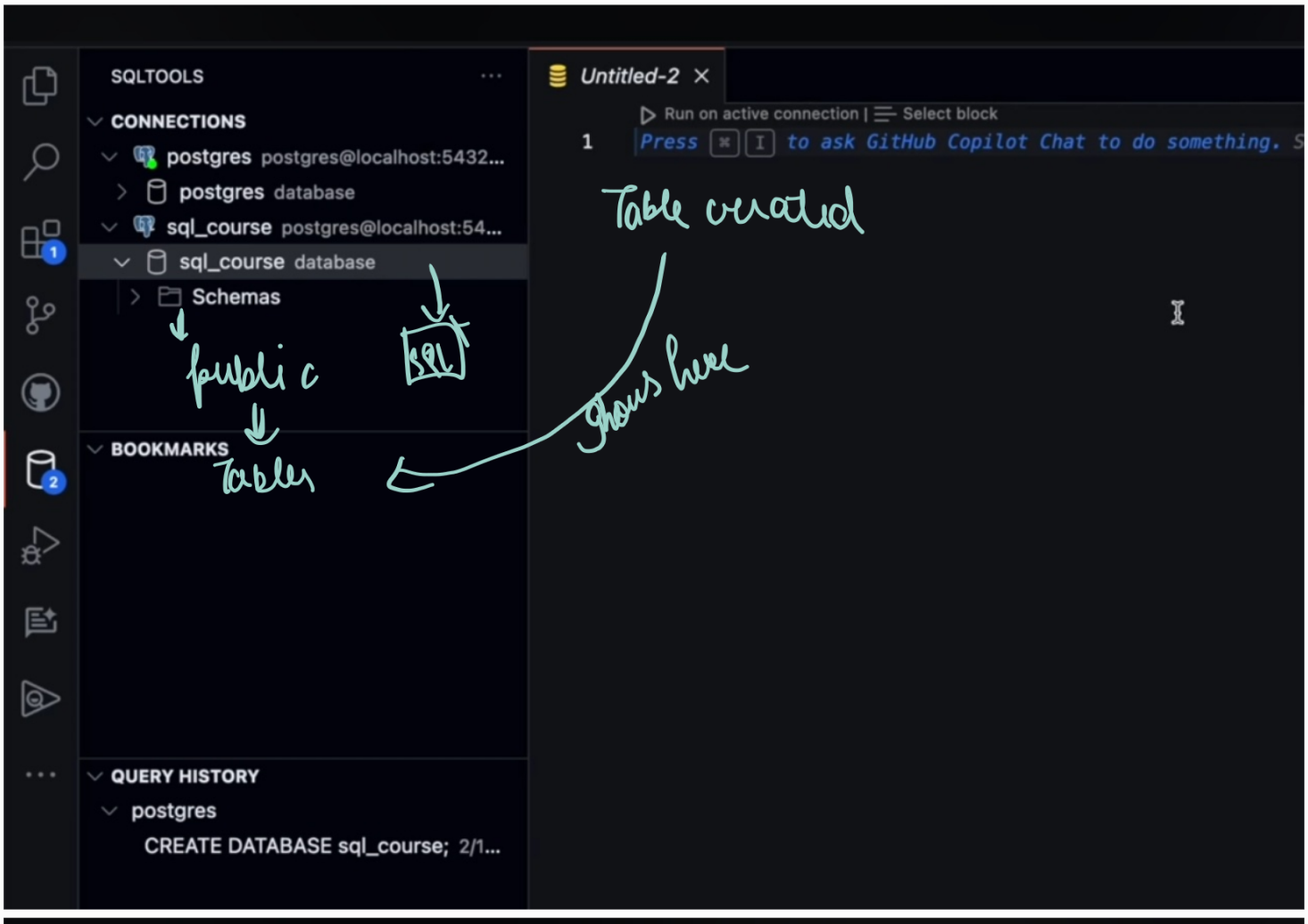
1) create table T (id int, name varchar(20)) ;

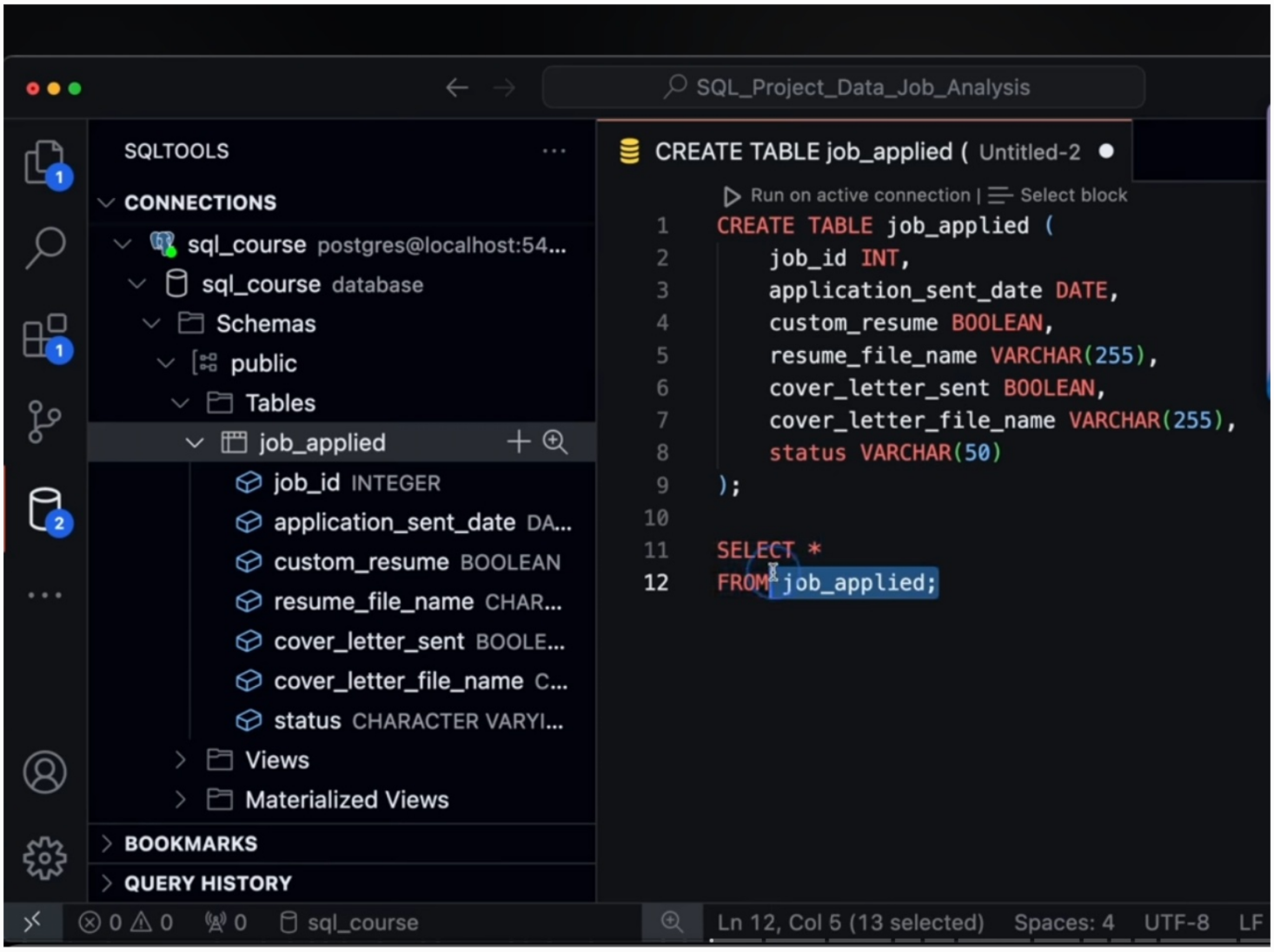
Manipulate Tables (Be careful, can't always undo)

The following statements are used to manipulate tables.

- `CREATE TABLE` : create tables
- `INSERT INTO` : add columns (data) to your tables
- `ALTER TABLE` : alter tables
  - `ADD` : add columns
  - `RENAME COLUMN` : rename columns
  - `ALTER COLUMN` : change the datatype of a column
  - `DROP COLUMN` : delete a column
- `DROP TABLE` : delete tables => be careful as permanent

1) pgAdmin => make sure database running
   VS code => SQLTools => click on database => check at bottom name


SQLTOOLS
CONNECTIONS
postgres postgres@localhost:5432...
postgres database
sql_course postgres@localhost:54...
sql_course database
Schemas
BOOKMARKS
QUERY HISTORY
postgres
CREATE DATABASE sql_course; 2/1...
Untitled-2
Run on active connection | Select block
1
Press ⌘ I to ask GitHub Copilot Chat to do something.
Table created
shows here
public
SQL
Tables


SQL_Project_Data_Job_Analysis
SQLTOOLS
CONNECTIONS
sql_course postgres@localhost:54...
sql_course database
Schemas
public
Tables
job_applied
job_id INTEGER
application_sent_date DA...
custom_resume BOOLEAN
resume_file_name CHAR...
cover_letter_sent BOOLE...
cover_letter_file_name C...
status CHARACTER VARYI...
Views
Materialized Views
BOOKMARKS
QUERY HISTORY
CREATE TABLE job_applied (Untitled-2
Run on active connection | Select block
1 CREATE TABLE job_applied (
2 job_id INT,
3 application_sent_date DATE,
4 custom_resume BOOLEAN,
5 resume_file_name VARCHAR(255),
6 cover_letter_sent BOOLEAN,
7 cover_letter_file_name VARCHAR(255),
8 status VARCHAR(50)
9);
10
11 SELECT *
12 FROM job_applied;
0 0 0 sql_course
Ln 12, Col 5 (13 selected) Spaces: 4 UTF-8 LF

② Insert into T (col1, col2 ...)
values (v1, v2 ....),
(
)
(
);

③ Alter table

ALTER TABLE

Notes:

- `ALTER TABLE` - used to select the table that you will add, delete, or modify columns in
- Similar to using `FROM` to specify a table for querying

```
ALTER TABLE table_name
-- ADD column_name datatype;
-- RENAME COLUMN column_name TO new_name;
-- ALTER COLUMN column_name TYPE datatype;
-- DROP COLUMN column_name
```

owner to postgres

UPDATE

④

Notes:

- `UPDATE` - used to modify existing data in a table.
- `SET` - specifies the column to be updated and the new value for that column.
- `WHERE` - filters which rows to update based on a condition.

```
UPDATE table_name
SET column_name = 'new_value'
WHERE condition;
```

update T
set name = 'Billy'
where job_id = 1;

update T
set name = 'Sara'
where job_id = 2;

ALTER COLUMN

📝 **Notes:**

- `ALTER COLUMN` - used to modify the properties of an existing column in a table
- **Change Data Type:** Modify the column's data type, subject to compatibility between the old and new types.

```
ALTER TABLE table_name
ALTER COLUMN column_name TYPE new_data_type;
```

  - Limitations: Certain data types can't be changed
- **Set/Change Default Value:** Assign a default value to the column, which will be used for new rows if no value is specified.

```
ALTER TABLE table_name
ALTER COLUMN column_name SET DEFAULT default_value;
```

- **Drop Default Value:** Remove the default value from the column if one exists.

```
ALTER TABLE table_name
ALTER COLUMN column_name DROP DEFAULT;
```

Loading database

🐘 How to create our database

1. Download CSV & SQL files
2. Create tables in the database — in VS code via SQL file provided
3. Load data into tables — empty tables

① Download file, open SQL data folder drag and drop files

② i) create database sql-course;

ii)

Make sure connection is established, check down sql_course written)

1) click on postgres database, => click + SQL file
   run command create database sql_course ;
2) go to pgAdmin 4 => sql_course click
3) vs code -> make new connection

① id int primary key

② Primary key (job_id, skill_id)

③ Foreign key (job_id) references public. Tablename (job_i

1) create database SQL_course

2) drop database if exists SQL_course

3) use SQL - course

Primary key ; Foreign key

=> unique identifier , can't be null

=> each table can have only 1 primary key (can be a single column or combination of columns)

(student can appear in multiple subjects combo needs to be unique)

primary key (student_id, subject)

foreign key (C_1, C_2) references tab (C_1, C_2)

Rules (column types must match → same data type, size, order)

can't use partial keys ⇒ all columns must be included in the match

↳ means using only part of composite key

=> id int auto_increment primary key

(say student table, marks table with PK, FK

can't insert marks for student who doesn't exist in 'students'

can't delete student if marks still refer to them)

Delete from T where id = 1;

	PK	FK (Referential integrity)
What it does	1) uniquely identifies each row	1) connect rows between tables
Where used	2) main table (owner)	2) dependent table (child)
Uniqueness	3) unique not null	3) can repeat (if many to one)
		Won't let you insert invalid reference

Eg => student (if id don't exist can't delete) Marks (Tables) (can't enter marks if data here exists)

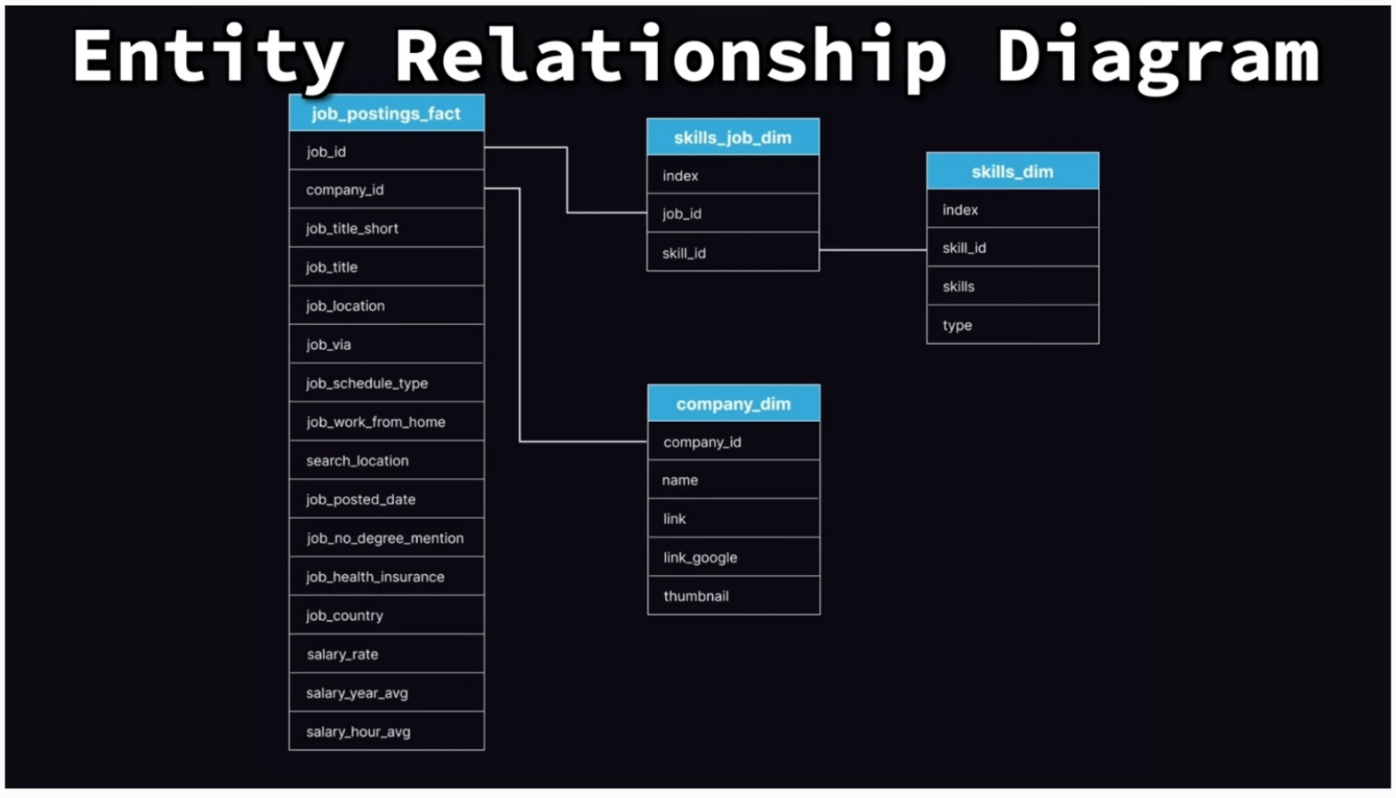
=> self referencing table

```
Create table emp (
    emp_id int primary key,
    name var(50),
    manager_id int,
    foreign key (manager_id) reference employee (emp_id)
```

An employee can be someone else's manager → recursive relationship;

setting ownership

```
alter table T owner to postgres ;
```



Public keyword (not directly used in core SQL like select, join etc)

① Public as user role / Access group (PostgreSQL, Oracle) mostly used in shared environments in enterprise settings

⇒ grant select on students to Public

② Public as schema (in PostgreSQL)

⇒ in PostgreSQL, there is default schema called public → by default puts all your tables in public schema

Schema is like a folder or namespace inside a database - it organises table

select * from students = select * from public.students

⇒

MySQL doesn't use public

Creating indexes on foreign key columns for better performance

1) index is like a search shortcut for database,
2) works like index in book, go straight to page instead of flipping everyone
3) makes lookups, joins, where, filters, F/K checks faster

Why index foreign keys ← often used in

Join queries
where filters
on delete / on update integrity check] speeds up

MySQL automatically creates, others manually

syntax: create index idx_name on FK(C1)

---

delete from students ⇒ all data deleted, table structure remains

" where id=

drop table students ⇒ data & structure of table d

Date functions → Advance

- Date
- Timestamp
- Timestamp with Time zone

1. select current_date; (PostgreSQL, MySQL)
   → select getdate(); (SQL server)

2. Extracting parts of date
   → select extract (year / month / day from '2020-06-21');
   DOY → day of year
   DOW → day of week
   (PostgreSQL, MySQL)
   (SQL server)
   → select datepart(");

3. select NOW();

4. Add days, months or years to a date
   → select date_add ('2025-06-21', interval 7 DAY); - MySQL
   → select '2025-06-21' :: date + interval '7 days'; - Postgres

5. Subtract
   select date_sub ('2025-06-21', interval 30 day); MySQL

6. Datediff()
   select datediff ('D1', 'D2'); - MySQL
   " (DAY, ' "); - SQL Server

7. select to_char (NOW(), 'YYYY-MM-DD'); - Postgres
   select format (getdate(), "); → SQL server

Handling Dates

Date Functions in SQL: Used to perform operations on date and time values

- `::DATE` : Converts to a date format by removing the time portion
- `AT TIME ZONE` : Converts a timestamp to a specified time zone
- `EXTRACT` : Gets specific date parts (e.g., year, month, day)

::DATE

```
SELECT
    timestamp_column::DATE AS date_column
FROM
    table_name;
```

Notes:

- `::` - used for casting, which means converting a value from one data type to another
  - You can use it to convert a host of different data types

```
SELECT '2023-02-19'::DATE, '123'::INTEGER, 'true'::BOOLEAN, '3.14'::REAL;
```

- `::DATE` - convert this value into a date format
  - In this case, it's converting a timestamp into a date format
  - **Date Format:** `YYYY-MM-DD`
    - Example: `2024-02-06`
  - **Timestamp Format:** `YYYY-MM-DD HH:MM:SS`
    - Example: `2024-02-06 15:04:05`

AT TIME ZONE

Notes:

- `AT TIME ZONE` - converts timestamps between different time zones
- It can be used on timestamps with or without time zone information
- Recall:
  - **TIMESTAMP**
    - A specific date and time without timezone: `2024-02-06 15:04:05`
    - Format: `YYYY-MM-DD HH:MI:SS`
  - **TIMESTAMP WITH TIME ZONE**
    - A specific date and time with time zone information: `2024-02-06 15:04:05+00:00`
    - Similar to `TIMESTAMP`, but includes time zone information

- **Timestamps with Time Zone:**
  - Stored as UTC, displayed per query's or system's time zone
  - `AT TIME ZONE` converts UTC to the specified time zone correctly

```
SELECT
    column_name AT TIME ZONE 'EST'
FROM
    table_name;
```

- **Timestamps without Time Zone (our situation):**
  - Treated as local time in PostgreSQL
  - Using `AT TIME ZONE` assumes the machine's time zone for conversion; specify it, or the default is UTC

```
SELECT
    column_name AT TIME ZONE 'UTC' AT TIME ZONE 'EST'
FROM
    table_name
```

specifies the timezone the value is at

converts to this

UT - universal time
UTC - Universal time, coordinated
Z
ZULU] same as UTC

Extract keyword

Date and time parts:
1) Year
2) Month 1-12
3) day 1-31
4) Hour 0-23
5) minute 0-59
6) second 0-59
7) milli " only supported by some DBs
8) microsecond mySQL, postgres

Week Quarter:
1) week of the year (1-53)
2) Quarter " (1-4)
3) DOY " (0-366)
4) DOW week (0-6) in postgres Sunday = 0
5) ISODOW " (1-7 1 - Monday
6) ISOWEEK
7) ISOYEAR

Epoch and timezone:
Timezone_hour
timezone_minute
timezone

Creating table from query results

```
create table T as
    select * from T2
    where Extract(Month ...) = 1;
```

CASE Expression

Notes:

A `CASE` expression in SQL is a way to apply conditional logic within your SQL queries.

```
-- Example SQL query using a CASE statement
SELECT
    CASE
        WHEN column_name = 'Value1' THEN 'Description for Value1'
        WHEN column_name = 'Value2' THEN 'Description for Value2'
        ELSE 'Other'
    END AS column_description
FROM
    table_name;
```

- `CASE` - begins the expression
- `WHEN` - specifies the condition(s) to look at
- `THEN` - what to do when the condition is `TRUE`
- `ELSE` (optional) - provides output if none of the `WHEN` conditions are met
- `END` - concludes the `CASE` expression

Subqueries and CTEs

Subqueries and **Common Table Expressions** (CTEs): Used for organizing and simplifying complex queries.

- Helps break down the query into smaller, more manageable parts
- When to use one over the other?
  - **Subqueries** are for simpler queries
  - **CTEs** are for more complex queries

creates 'temporary result set' inside SQL query and perform analysis on them

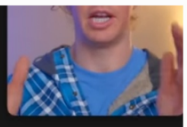
Subqueries -> queries nested inside larger query
can be used in select, from, where, having

select * from (select * from ...) as ____
subquery starts ends

1) Scalar subquery returns single value
2) row 1 row
3) Table table

correlated (depends on outer query)
n - d

🧩Subqueries



📝 **Notes:**

- Subquery - query within another query
- It can be used in several places in the main query
  - Such as the `SELECT`, `FROM`, `WHERE`, or `HAVING` CLAUSES
- It's executed first, and the results are passed to the outer query
  - It is used when you want to perform a calculation before the main query can complete its calculation

CTEs =>

🌐CTEs - Common Table Expression

📝 **Notes:**

- CTE - A temporary result set that you can reference within a `SELECT` `INSERT` `UPDATE` or `DELETE` statement.
- Exists only during the execution of a query
- It's a defined query that can be referenced in the main query or other CTEs
- `WITH` - used to define CTE at the beginning of a query.

```
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;
```

below it

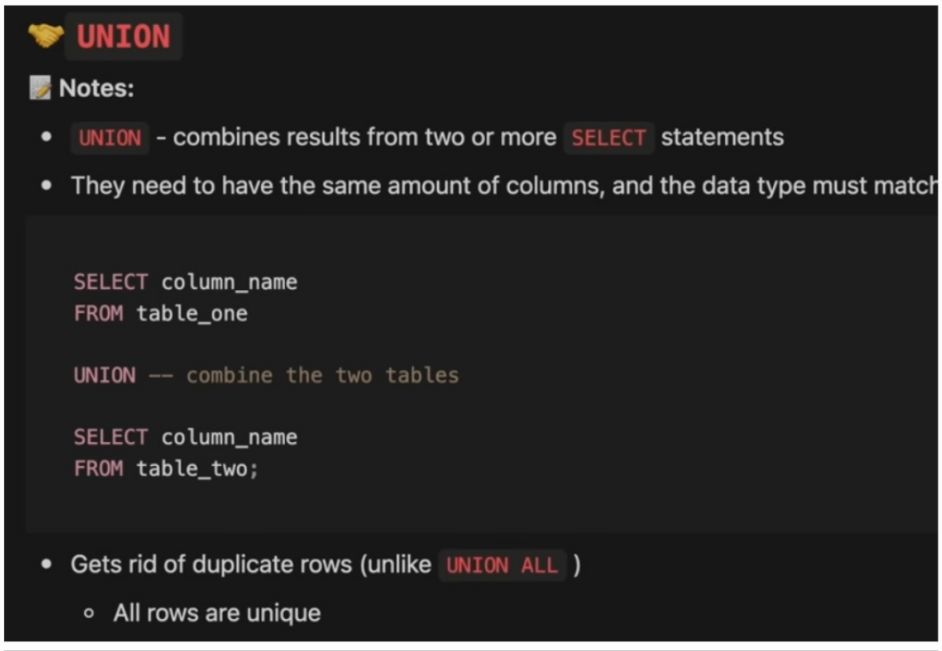
`UNION` Operators

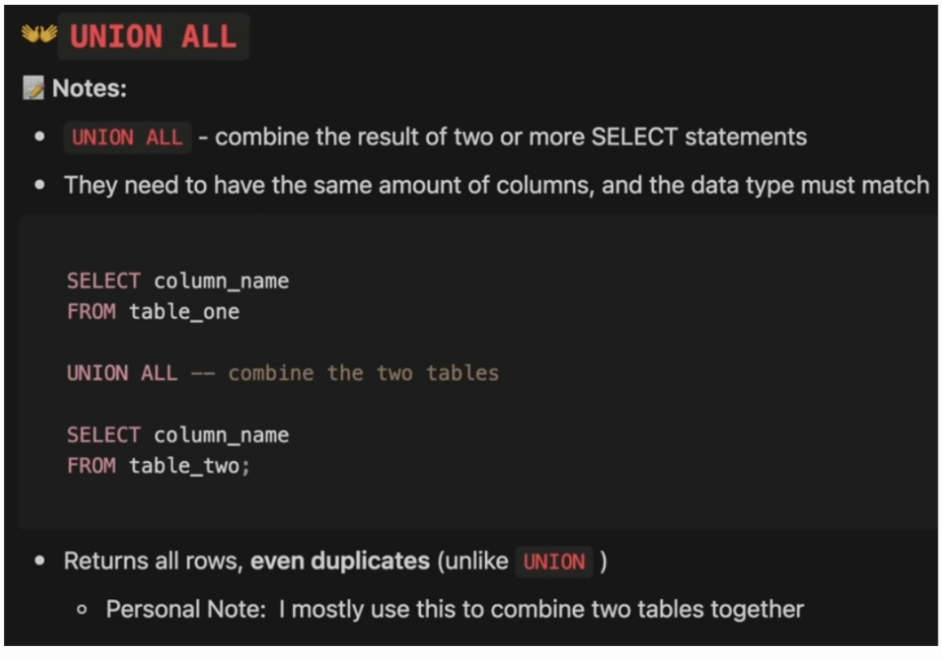
Combine result sets of two or more `SELECT` statements into a single result set.

- `UNION` : Remove duplicate rows
- `UNION ALL` : Includes all duplicate rows

⚠️Note: Each `SELECT` statement within the `UNION` must have the same number of columns in the result sets with similar data types.

Join when combine table that maybe relate on a single value





? Questions to Answer

1. What are the top-paying jobs for my role?

2. What are the skills required for these top-paying roles?

3. What are the most in-demand skills for my role?

4. What are the top skills based on salary for my role?

5. What are the most optimal skills to learn?

• Optimal: High Demand AND High Paying

PROJECT

```
Developer's Computer          ---------------------->          GitHub Server
       (Local)                                                    (Remote)

  * Working Directory       <---------------------->          Project Copy
  * Staging Area
  * Local Repository
```

Project
(as we work on it and collect document, it goes to)
↓
staging area
(when we have something we wanna save or commit)
↓ saves to
local repository

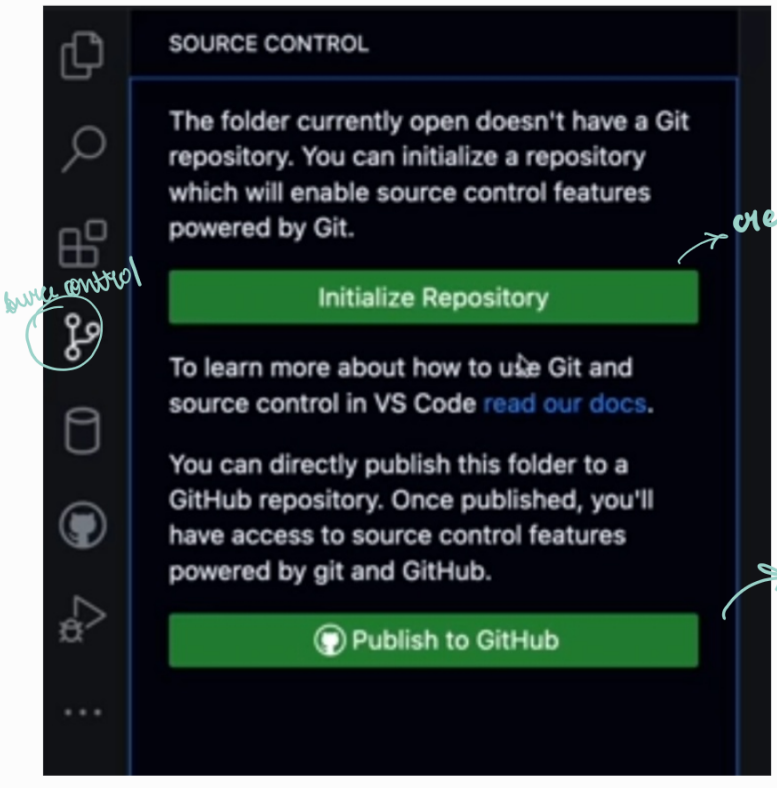
for sharing / collaborating
host it to github
host a copy

git → most popular version control system

Create Local & Remote Repository

Numerous options:

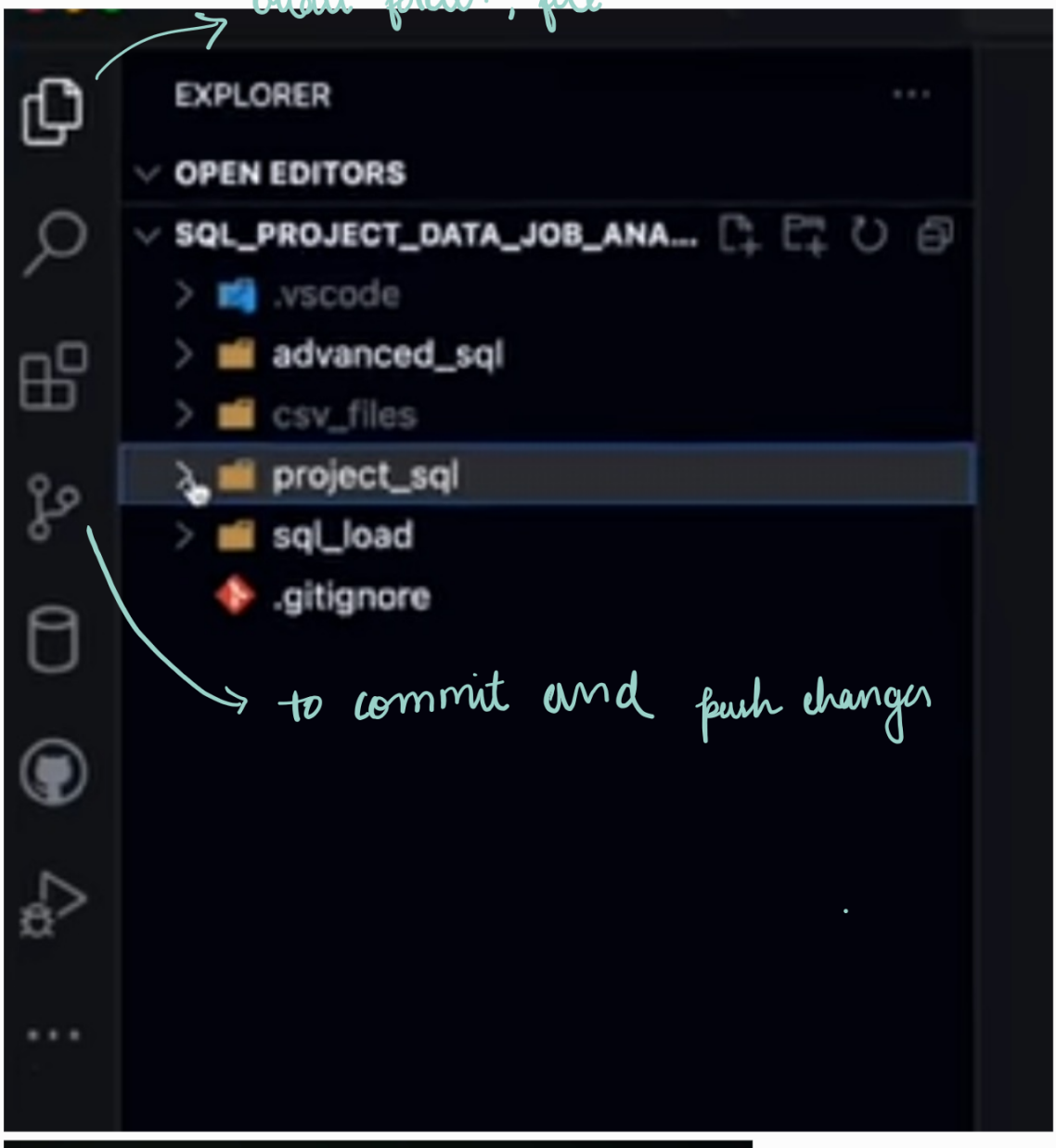
1. Create a new repository through **VS Code (recommended)**
2. Create a new repository on **GitHub Desktop**
3. Create a new repository on **GitHub**, then clone it onto your device.
4. Use the Command Line or Terminal (not covered)

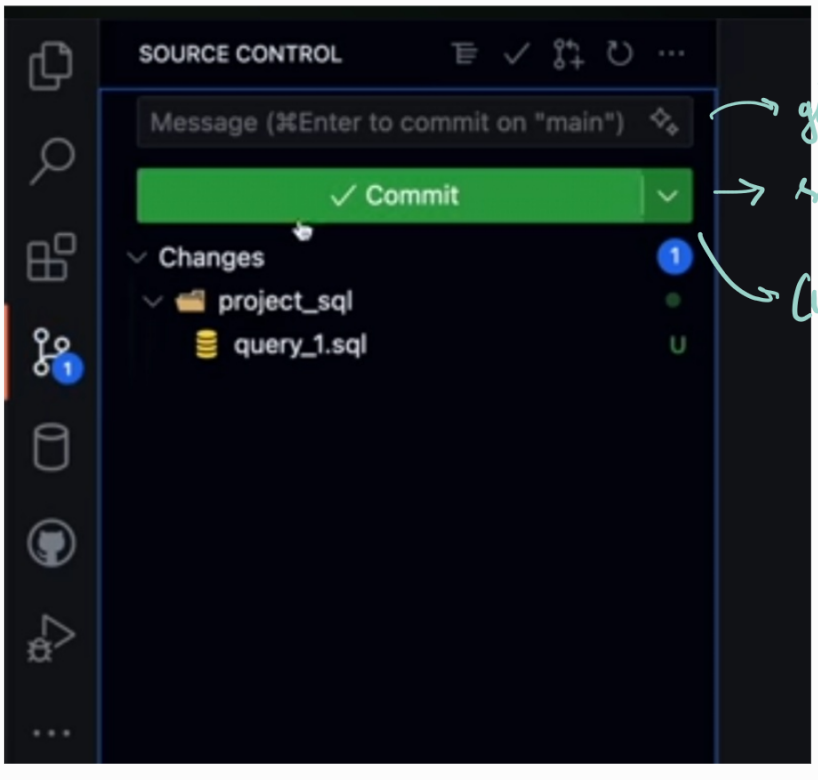


creates local repository (.git folder)
doesn't push to github

initializes the repo locally
also pushes all content here to github

Example of pushing changes from local repo / remote repo


create folder, file
EXPLORER
OPEN EDITORS
SQL_PROJECT_DATA_JOB_ANA...
.vscode
advanced_sql
csv_files
project_sql
sql_load
.gitignore
to commit and push changes


SOURCE CONTROL
Message (⌘Enter to commit on "main")
Commit
Changes
project_sql
query_1.sql
U
give it a name
sync changes → (to remove repository)
(update on local repository)

Pulling

github → <> code

Add a readme → commit changes

VScode → source control → ... → Pull, Push → pull

Comparision operators

=, != <>, >, >=, <, <=,

Between

in + NOT (in the beginning)

is

is null

like

Joins => used to combine columns from two or more tables based on a related column, usually a foreign key
not necessary defining FK

Union => combine rows

Top demand skill for my job
=> skill how many jobs it appeared

```
found (col, 0)
order by ___ desc,
         ___ desc
```

```
Select
    skills,
    skill_id
    count(*)
from
where
group by
```

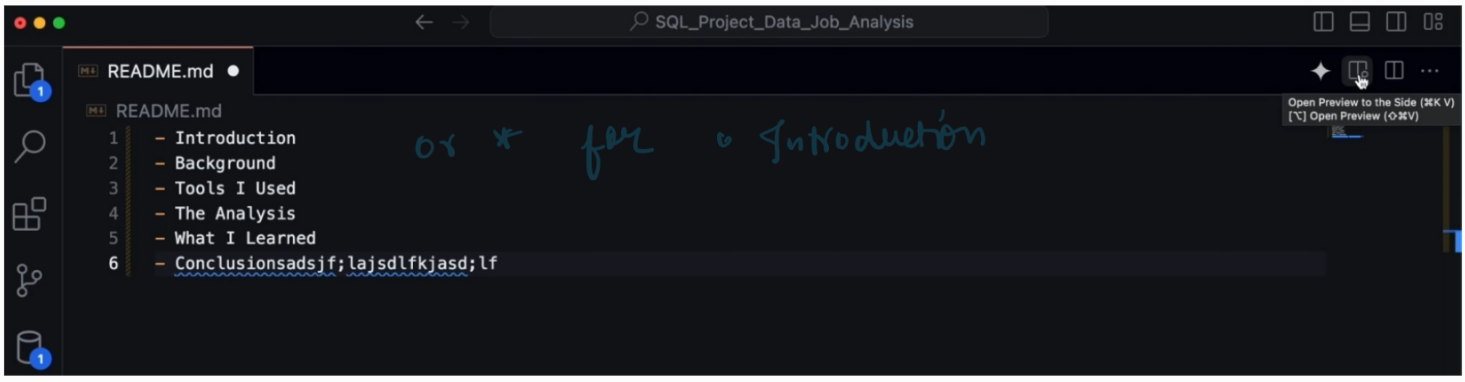
No error as 1 to 1 related

skills skill_id

functionally dependent

Can have 2 or more columns in group by

Github



Formats

1) Headers # H1 ## H2 ### H3
2) Bullets - , *
3) links [name](/folder name/)
4) Image ![Text](local path) (have to upload the photos in vs code
5) **Bold**
6) *Italics*
7) ```sql
   ```

pinning the projects

customize your pins → select → can drag now wherever you want to place
   ```