# **💼 The Advanced SQL + Python Interview Guide (Webinar Special Edition – 2025)**

## **📘 Why this Guide?**

Whether you're a **Data Analyst, Data Engineer, or BI Developer**, interviews today don’t just test syntax. They push you on **real-world problems**, optimization strategies, and your ability to **think like a problem solver.**

This guide:  
 ✅ Gives you **tricky & real-life questions** ✅ Explains the **interviewer’s intent** ✅ Shares **best-practice answers & SQL/Python code snippets** ✅ Equips you for **follow-up questions** so you don’t get stuck

# **🔹 Section 1: 20+ Advanced SQL Conceptual + Scenario-Based Questions**

## **Q1. Find the Second Highest Salary from a Large Table**

**Interviewer’s Intent:** Can you write a scalable, duplicate-safe query?

**Answer Strategy:** Use DENSE\_RANK() to handle duplicate salaries.

sql

SELECT salary

FROM (

SELECT salary, DENSE\_RANK() OVER (ORDER BY salary DESC) AS rank

FROM employees

) AS ranked

WHERE rank = 2;

**Pro Tip:** ✅ Works well even if two people have the same salary  
 ✅ Avoids problems of NULLs and duplicates

## **Q2. RANK vs DENSE\_RANK vs ROW\_NUMBER**

| **Function** | **Behavior** |
| --- | --- |
| ROW\_NUMBER() | Always unique, no gaps |
| RANK() | Gaps in ranking (e.g., 1, 2, 2, 4) |
| DENSE\_RANK() | No gaps (e.g., 1, 2, 2, 3) |

**Use Case:** Leaderboard ranking, paginated results, or product tiers.

## **Q3. Identify Repeat Customers in Last 3 Months**

**SQL Approach:**

sql

SELECT user\_id

FROM orders

WHERE order\_date >= CURRENT\_DATE - INTERVAL '3 months'

GROUP BY user\_id

HAVING COUNT(DISTINCT DATE\_TRUNC('month', order\_date)) > 1;

**Follow-up:**

* How would you classify them in the same query? → Use CASE WHEN
* How to make it dynamic for any period? → Use parameters

## **Q4. CTEs in Real Projects (Bonus: Recursive CTEs)**

**Example:** Employee-Manager Tree

sql

WITH RECURSIVE emp\_hierarchy AS (

SELECT employee\_id, manager\_id, 1 AS level

FROM employees

WHERE manager\_id IS NULL

UNION ALL

SELECT e.employee\_id, e.manager\_id, eh.level + 1

FROM employees e

INNER JOIN emp\_hierarchy eh ON e.manager\_id = eh.employee\_id

)

SELECT \* FROM emp\_hierarchy;

## **Q5. Top 3 Products per Category**

sql

SELECT \*

FROM (

SELECT category, product, RANK() OVER(PARTITION BY category ORDER BY sales DESC) AS rnk

FROM products

) AS ranked

WHERE rnk <= 3;

**Optimization Tip:** Cluster table by category and sales DESC if possible.

## **Q6. Detect Consecutive Login Activity (Churn Prediction)**

sql

SELECT user\_id, login\_date,

LAG(login\_date) OVER (PARTITION BY user\_id ORDER BY login\_date) AS previous\_login,

DATEDIFF(login\_date, LAG(login\_date) OVER (PARTITION BY user\_id ORDER BY login\_date)) AS days\_between

FROM user\_logins;

**Use Case:** Mark users inactive if days\_between > 30.

## **Q7. Optimizing Slow Queries on 10M+ Rows**

Checklist:  
 ✅ Run EXPLAIN PLAN  
 ✅ Ensure **index on WHERE & JOIN columns** ✅ Replace \*\*SELECT \*\*\* with specific columns  
 ✅ Convert **nested subqueries → CTEs or joins** ✅ Use **partition filters** if on partitioned tables

## **Q8. Clean Up Messy Strings**

sql

SELECT TRIM(LOWER(REGEXP\_REPLACE(city\_name, '[^a-zA-Z ]', ''))) AS clean\_city

FROM users;

**Example:** " New-York!! " → "new york"

## **Q9. Sales Funnel in SQL**

| **Stage** | **Example SQL Logic** |
| --- | --- |
| Visited Site | CASE WHEN page = 'landing' THEN 1 END |
| Added to Cart | CASE WHEN action = 'add\_to\_cart' THEN 1 END |
| Purchased | CASE WHEN action = 'purchase' THEN 1 END |
| Conversion Rate | (COUNT(purchased) / COUNT(visited)) \* 100 |

## **Q10. JOIN Types + EXISTS**

| **Type** | **Use Case Example** |
| --- | --- |
| INNER JOIN | Match only users who purchased |
| LEFT JOIN | Show all users, even if no purchase |
| EXISTS | Check if user ever purchased (fast filtering) |

Example:

sql

SELECT user\_id

FROM users u

WHERE EXISTS (

SELECT 1 FROM orders o WHERE o.user\_id = u.user\_id

);

## **Q11. Pivot a Table Without PIVOT Function**

sql

SELECT

user\_id,

SUM(CASE WHEN month = 'Jan' THEN sales ELSE 0 END) AS jan\_sales,

SUM(CASE WHEN month = 'Feb' THEN sales ELSE 0 END) AS feb\_sales

FROM sales\_data

GROUP BY user\_id;

## **Q12. Remove Duplicates Efficiently**

sql

DELETE FROM orders o1

WHERE EXISTS (

SELECT 1 FROM orders o2

WHERE o1.order\_id > o2.order\_id AND o1.data = o2.data

);

## **Q13. Calculate Running Total**

sql

SELECT order\_date, SUM(sales) OVER (ORDER BY order\_date) AS running\_total

FROM orders;

## **Q14. Handle NULLs in Joins**

Use COALESCE to replace NULLs in outer joins:

sql

SELECT COALESCE(o.total, 0) AS total\_sales

FROM products p

LEFT JOIN orders o ON p.product\_id = o.product\_id;

## **Q15. Dynamic Filtering Example**

SQL for dynamic filtering by passing NULL:

sql

WHERE (region = :region OR :region IS NULL)

# **🔹 Section 2: Python Questions (with Interview-Style Answers)**

## **Q1. Difference Between List, Set, Tuple, Dict**

| **Data Type** | **Mutable?** | **Allows Duplicates?** | **Ordered?** |
| --- | --- | --- | --- |
| List | Yes | Yes | Yes |
| Set | Yes | No | No |
| Tuple | No | Yes | Yes |
| Dict | Yes | No (keys) | Yes (3.7+) |

## **Q2. How to Merge Two DataFrames on Multiple Keys**

df\_merged = pd.merge(df1, df2, on=['customer\_id', 'region'], how='inner')

**Follow-up:** What if there are duplicates?  
 👉 Use drop\_duplicates() after merge.

## **Q3. What’s the Difference Between map() and apply()?**

* map(): Works on Series only
* apply(): Works on both Series and DataFrames

Example:

df['new'] = df['column'].map(lambda x: x \* 2)

## **Q4. Explain GroupBy + Aggregation**

df.groupby('region')['sales'].agg(['sum', 'mean'])

## **Q5. Handle Large CSV Files**

✅ Use chunksize in pandas:

for chunk in pd.read\_csv('big.csv', chunksize=100000):

process(chunk)

✅ Or use Dask for parallel processing.

## **Q6. Explain Logging in a Pipeline**

import logging

logging.basicConfig(level=logging.INFO)

logging.info('Pipeline started')

Follow-up: Log errors?

try:

risky\_function()

except Exception as e:

logging.error(f'Error occurred: {e}')

## **Q7. How to Connect to a Database and Run Queries**

import psycopg2

conn = psycopg2.connect(database='db', user='user', password='pass', host='localhost')

cursor = conn.cursor()

cursor.execute("SELECT \* FROM table;")

rows = cursor.fetchall()

# **✅ Final Checklist Table (Interview Quick Revision)**

| **Topic** | **SQL Example** | **Python Example** |
| --- | --- | --- |
| Top-N Query | RANK() OVER PARTITION | groupby().nlargest() |
| Cleaning Data | TRIM(), REGEXP\_REPLACE() | df['col'].str.strip() |
| Pipeline Logging | - | logging.basicConfig() |
| Window Functions | LAG, LEAD, ROW\_NUMBER | df['col'].shift() |
| Joining | INNER, LEFT, EXISTS | pd.merge() |
| Optimization | EXPLAIN PLAN, Indexing | Using chunksize, Dask for large files |

# **🔹 SQL Advanced (More Deep-Dive Questions)**

## **Q16. What’s the Difference Between CROSS JOIN and INNER JOIN?**

| **JOIN Type** | **Behavior** |
| --- | --- |
| INNER JOIN | Combines matching rows from both tables |
| CROSS JOIN | Produces Cartesian product (all combinations) |

**Example CROSS JOIN:**

sql

SELECT a.name, b.product

FROM customers a

CROSS JOIN products b;

**When to use?** ✅ Use for generating permutations (e.g., creating test datasets)

## **Q17. Find Users Who Placed Orders Every Month in 2024**

sql

SELECT customer\_id

FROM orders

WHERE order\_date BETWEEN '2024-01-01' AND '2024-12-31'

GROUP BY customer\_id

HAVING COUNT(DISTINCT DATE\_TRUNC('month', order\_date)) = 12;

**Follow-up:**

* What if data is missing? Add a **calendar table** for robust checks.

## **Q18. How Do You Handle Slowly Changing Dimensions (SCD) in SQL?**

**Strategy:**

* SCD Type 1: Overwrite existing data
* SCD Type 2: Track history with start\_date, end\_date, is\_active

**Example (Type 2):**

sql

UPDATE customer\_dim

SET end\_date = CURRENT\_DATE, is\_active = FALSE

WHERE customer\_id = 101 AND is\_active = TRUE;

INSERT INTO customer\_dim (customer\_id, name, start\_date, is\_active)

VALUES (101, 'Updated Name', CURRENT\_DATE, TRUE);

## **Q19. Explain the Use of Window Frames**

sql

SELECT order\_date,

SUM(sales) OVER (ORDER BY order\_date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) AS 7\_day\_total

FROM orders;

**Why It Matters:** Lets you do rolling averages, moving sums → real-time reporting.

## **Q20. Handling Multi-Valued Columns (Comma-Separated Strings)**

**Split column:** Postgres example:

sql

SELECT unnest(string\_to\_array(tags, ',')) AS tag

FROM posts;

**Interview Trap:** They might test how you **normalize** this into a proper many-to-many relationship.

## **Q21. How to Pivot and Unpivot in SQL?**

**Unpivot Example:**

sql

SELECT user\_id, 'Jan' AS month, jan\_sales AS sales FROM sales

UNION ALL

SELECT user\_id, 'Feb', feb\_sales FROM sales;

## **Q22. Date Dimension Table — Why & How to Build It**

**Why:**

* Avoid bugs in date math
* Ensure gap-filling in reports

**Table Columns:** date, day\_of\_week, week\_number, is\_holiday, etc.

**Tip:** Join it in queries to fill gaps even if no data exists.

## **Q23. Find Gaps in Date Sequences**

sql

WITH date\_diff AS (

SELECT user\_id,

login\_date,

LAG(login\_date) OVER (PARTITION BY user\_id ORDER BY login\_date) AS prev\_login

FROM logins

)

SELECT \*, DATEDIFF(login\_date, prev\_login) AS gap\_days

FROM date\_diff

WHERE gap\_days > 1;

## **Q24. Explain Anti-Joins (Find Non-Matching Records)**

sql

SELECT a.\*

FROM customers a

LEFT JOIN orders b ON a.customer\_id = b.customer\_id

WHERE b.order\_id IS NULL;

✅ Very useful for **finding inactive users, missing data**

## **Q25. Optimize a Query that Uses ORDER BY RANDOM() LIMIT 1**

**Problem:** It scans full table → **very slow for large datasets.**

**Better Solution:**

* Get random **ID** between MIN(id) and MAX(id)
* Or pre-sample 1% rows into a temp table, then pick random.

# **🔹 Python Advanced (More Deep-Dive Questions)**

## **Q8. Explain Difference Between is vs ==**

| **Operator** | **What it Does** |
| --- | --- |
| == | Compares **values** |
| is | Compares **identities (memory address)** |

**Example:**

a = [1, 2]

b = [1, 2]

print(a == b) # True

print(a is b) # False

## **Q9. List Comprehension vs Generator Expression**

**List Comp:**

[x\*2 for x in range(5)]

**Generator Exp:**

(x\*2 for x in range(5))

✅ Generators are more **memory-efficient.**

## **Q10. Explain Mutable Default Arguments Trap**

def append\_item(item, my\_list=[]):

my\_list.append(item)

return my\_list

append\_item(1) # [1]

append\_item(2) # [1, 2] ← Unexpected!

✅ Fix: Use None and create inside function:

def append\_item(item, my\_list=None):

if my\_list is None:

my\_list = []

my\_list.append(item)

return my\_list

## **Q11. Explain args and kwargs**

def func(\*args, \*\*kwargs):

print(args)

print(kwargs)

func(1, 2, 3, a=4, b=5)

✅ Allows flexible input to functions.

## **Q12. What’s the Difference Between deep copy and shallow copy?**

import copy

a = [[1, 2], [3, 4]]

b = copy.copy(a) # Shallow

c = copy.deepcopy(a) # Deep

✅ Shallow → new outer list but inner lists same  
 ✅ Deep → completely new

## **Q13. How to Improve Performance in Pandas (Large Datasets)?**

* Use **categorical data types** for string columns
* Avoid **apply()** if vectorized functions exist
* Use df.query() for complex filters
* Process in **chunks**

## **Q14. Difference Between Thread vs Process in Python**

| **Thread** | **Process** |
| --- | --- |
| Shares memory with main thread | Independent memory |
| Faster for I/O-bound tasks | Better for CPU-bound tasks |

## **Q15. Error Handling Best Practices in Pipelines**

* Always log errors:

try:

risky\_job()

except Exception as e:

logging.error(f"Error: {e}")

* Raise custom exceptions for clarity.

## **Q16. How to Write Unit Tests (Pytest Example)**

def add(x, y):

return x + y

def test\_add():

assert add(2, 3) == 5

Run with:

bash

pytest test\_script.py

## **Q17. Using SQLAlchemy for Safe DB Access**

from sqlalchemy import create\_engine

engine = create\_engine('postgresql://user:pass@localhost/db')

df = pd.read\_sql('SELECT \* FROM table', con=engine)

✅ Prevents SQL injection  
 ✅ Supports connection pooling

## **Q26. How Do You Perform an Efficient Full-Text Search in SQL?**

**Interview Trap:** Using LIKE '%search%' is **slow** for large datasets.

✅ **Better:**

* Postgres: Use to\_tsvector() + to\_tsquery()

sql

SELECT \* FROM articles

WHERE to\_tsvector('english', content) @@ to\_tsquery('machine & learning');

✅ MySQL: Use MATCH() AGAINST()  
 ✅ BigQuery: Use CONTAINS\_SUBSTR() (partial) or **external search index**

## **Q27. How Do You Handle Out-of-Order Streaming Data in SQL (BigQuery/Streaming Pipelines)?**

✅ Use **Watermarks + Late Data Handling**

Example (BigQuery SQL):

sql

SELECT user\_id, COUNT(\*)

FROM STREAM\_TABLE

WHERE \_PARTITIONTIME BETWEEN '2024-01-01' AND '2024-01-07'

GROUP BY user\_id;

➡️ In **Spark SQL:** Use withWatermark() to handle late arrivals.

## **Q28. Delete Duplicate Rows But Keep the Oldest Entry (by created\_at)**

sql

WITH ranked AS (

SELECT \*,

ROW\_NUMBER() OVER (PARTITION BY email ORDER BY created\_at ASC) AS rn

FROM users

)

DELETE FROM users

WHERE user\_id IN (

SELECT user\_id FROM ranked WHERE rn > 1

);

👉 **Tricky Part:** Be ready to explain **why ROW\_NUMBER is better than DISTINCT** for **fine-grained control.**

## **Q29. Explain the Difference Between Materialized Views vs Regular Views**

| **Views** | **Materialized Views** |
| --- | --- |
| Logical only (no storage) | Stores physical data |
| Always fresh | Needs REFRESH |
| Slow for heavy queries | Fast (pre-computed) |

✅ Interview Angle:

* When you want **real-time:** use **normal views**
* When you want **speed & don't mind staleness:** use **materialized**

## **Q30. How to Avoid “GROUP BY” Errors on Non-Aggregated Columns (SQL Mode)**

Example **error:**

sql

SELECT customer\_id, region, COUNT(\*) FROM orders GROUP BY customer\_id;

➡️ **Fix:** Add all non-aggregated columns to GROUP BY:

sql

GROUP BY customer\_id, region

✅ **Trick:** Some databases (MySQL in loose mode) allow it; others (Postgres, strict ANSI SQL) **fail unless corrected.**

## **Q31. Write a Query to Calculate Running AVG with Reset When Value Crosses a Threshold**

Example:

* **Goal:** Restart running total when sales > 1000

**Trick:** Use **SUM() FILTERED by a window frame + CASE**

## **Q32. Find Top 3 Departments with Highest Avg Salary, Excluding Outliers (Above 95th Percentile)**

Postgres example:

sql

WITH salary\_clean AS (

SELECT \*,

PERCENTILE\_CONT(0.95) WITHIN GROUP (ORDER BY salary) OVER () AS p95

FROM employees

)

SELECT department, AVG(salary)

FROM salary\_clean

WHERE salary < p95

GROUP BY department

ORDER BY AVG(salary) DESC

LIMIT 3;

✅ **Deep Dive:** How does PERCENTILE work vs manually computing?

## **Q33. What Happens If You Index a Column With Many NULLs?**

**Answer:**

* Indexing columns with lots of NULLs is **often wasteful** because:  
  + Many DBs **skip NULLs** in B-tree index
  + Filtering WHERE col IS NULL **may not use the index**

✅ **Workaround:** Partial Index (Postgres)

sql

CREATE INDEX idx\_email\_not\_null ON users(email) WHERE email IS NOT NULL;

## **Q34. Find Median Salary Per Department (If MEDIAN() Not Supported)**

**Trick:** Use PERCENTILE\_CONT(0.5) (Postgres)  
 Or manually:

sql

SELECT department,

percentile\_disc(0.5) WITHIN GROUP (ORDER BY salary)

FROM employees

GROUP BY department;

✅ **BigQuery:** Use APPROX\_QUANTILES()

## **Q35. How to Monitor Query Performance in Production?**

✅ Set up:

* **Query logs + scan sizes** (BigQuery)
* **pg\_stat\_statements** (Postgres)
* **Slow query logs** (MySQL)

# **🔹 Advanced + Tricky Python Questions (Continued)**

## **Q18. What’s the Difference Between @staticmethod vs @classmethod?**

| **Decorator** | **First Argument** | **Use Case** |
| --- | --- | --- |
| @staticmethod | None | Utility method, no class needed |
| @classmethod | cls | Factory pattern, alternate constructor |

Example:

class MyClass:

@staticmethod

def util():

return 'no self'

@classmethod

def from\_string(cls, s):

return cls(s)

## **Q19. What’s the Global Interpreter Lock (GIL)?**

**Answer:**

* Python’s **GIL** ensures only 1 thread runs **Python bytecode** at a time.
* ✅ **Good:** Protects memory
* ❌ **Bad:** Blocks true parallelism in **CPU-bound tasks**

✅ **Follow-up:** Use multiprocessing to bypass GIL for CPU-heavy work.

## **Q20. Mutable vs Immutable Data Types**

| **Mutable** | **Immutable** |
| --- | --- |
| List | Tuple |
| Dict | String |
| Set | Frozenset |

**Follow-up:** Why is **string** immutable?  
 ➡️ Ensures **hashability** and avoids unexpected side effects.

## **Q21. How Does Python Handle Memory?**

* Reference Counting + Garbage Collection
* gc module can **track & collect cycles**

## **Q22. Explain Lazy Evaluation in Python**

✅ **Generators & Iterators** implement **lazy eval** → only compute **on demand.**

Example:

gen = (x \* 2 for x in range(10))

## **Q23. How Do You Catch Multiple Exception Types?**

try:

risky\_op()

except (ValueError, KeyError) as e:

logging.error(e)

✅ Cleaner than multiple except blocks.

## **Q24. What’s a Memoryview & Why Use It?**

**Answer:** A memoryview exposes the **internal buffer** of an object (like bytes) **without copying.**

Example:

data = bytearray(b'hello')

view = memoryview(data)

view[0] = 72 # Mutates original data

✅ **Use Case:** High-perf processing (e.g., binary streams)

## **Q25. How Do You Merge Two Dicts in Python 3.9+?**

a = {'x': 1}

b = {'y': 2}

merged = a | b

Earlier versions: {\*\*a, \*\*b}

## **Q26. Difference Between str and repr**

| **Method** | **Purpose** |
| --- | --- |
| **str** | User-friendly output (for print()) |
| **repr** | Debug/dev output, unambiguous |

## **Q27. How Would You Validate a JSON Schema?**

from jsonschema import validate, ValidationError

try:

validate(instance=my\_json, schema=my\_schema)

except ValidationError as e:

print(e)

✅ **Must-know:** API & ETL validations.

## **Q28. How to Cache Expensive Function Calls?**

✅ Use functools.lru\_cache

from functools import lru\_cache

@lru\_cache(maxsize=100)

def expensive\_fn(param):

# long-running work

return result

## **Q29. What’s the Difference Between filter(), map(), reduce()?**

| **Function** | **Purpose** |
| --- | --- |
| map() | Transform each item |
| filter() | Keep items that match condition |
| reduce() | Aggregate to single value |

## **Q30. How Do You Read a Large JSON File Efficiently?**

✅ Instead of loading all:

import json

with open('big.json') as f:

for line in f:

record = json.loads(line)

✅ Or use ijson for **stream parsing.**

## **Q36. What is the Difference Between UNION and UNION ALL?**

| **Operator** | **Behavior** |
| --- | --- |
| UNION | Combines + removes duplicates (slower) |
| UNION ALL | Combines without removing duplicates (faster) |

✅ **Trick Question:** UNION might *appear* to work fine but can **silently drop duplicates** — **always confirm if deduplication is intended.**

## **Q37. How Do You Find the First and Last Purchase Date for Each Customer?**

sql

SELECT customer\_id,

MIN(order\_date) AS first\_purchase,

MAX(order\_date) AS last\_purchase

FROM orders

GROUP BY customer\_id;

✅ **Interview Twist:** Follow-up may ask:  
 *"What if you need the full row, not just the date?"*

👉 Use FIRST\_VALUE() and LAST\_VALUE() window functions.

## **Q38. How Do You Check for Data Skew in a Big Join?**

**Answer Strategy:**

* Run: SELECT join\_key, COUNT(\*) FROM table GROUP BY join\_key ORDER BY COUNT(\*) DESC LIMIT 5;
* Look for **high-count keys** → a **sign of skew.**

✅ **Spark:** Use salting to distribute skewed keys.

## **Q39. Can You Explain Index-Only Scans?**

**Answer:**

* An **Index-Only Scan** means the DB can return **all required columns** just by reading the index — **no table lookup.**

✅ **Example:** SELECT id FROM users WHERE id = 5; (if id is indexed)

## **Q40. How Do You Write an Idempotent SQL Script?**

**Trick Concept:** Idempotent = running it **multiple times has the same effect as running it once.**

Example:

sql

CREATE TABLE IF NOT EXISTS my\_table (...);

INSERT INTO my\_table (...) ON CONFLICT DO NOTHING;

## **Q41. How Can You Debug a Deadlock Issue?**

✅ Steps:

* **Log deadlock events** (Postgres: log\_lock\_waits = on)
* Look at pg\_locks to check blocking
* Add **timeout settings** to avoid indefinite waiting.

## **Q42. Write a Query to Rank Users Based on Consecutive Active Days**

➡️ Use **DENSE\_RANK() + LAG() + DATEDIFF()**

Example:

sql

WITH user\_days AS (

SELECT user\_id, login\_date,

DATEDIFF(login\_date, LAG(login\_date) OVER (PARTITION BY user\_id ORDER BY login\_date)) AS gap

FROM logins

)

SELECT user\_id, COUNT(\*) AS streak\_length

FROM user\_days

WHERE gap = 1

GROUP BY user\_id;

## **Q43. Find Rows Where a Foreign Key Has No Matching Row (Integrity Check)**

sql

SELECT orders.\*

FROM orders

LEFT JOIN customers ON orders.customer\_id = customers.customer\_id

WHERE customers.customer\_id IS NULL;

✅ **Use Case:** Data QA/audit pipelines.

## **Q44. How Do You Optimize a Query That Filters by a Calculated Column?**

Bad:

sql

WHERE YEAR(order\_date) = 2024

👎 Breaks index.

✅ Better: Rewrite:

sql

WHERE order\_date BETWEEN '2024-01-01' AND '2024-12-31'

## **Q45. What’s a Covering Index & When Do You Use It?**

**Answer:** An index that **includes all columns** a query needs. Prevents extra lookup.

Example:

sql

CREATE INDEX idx\_user\_orders ON orders (user\_id, status, order\_date);

# **🔹 Python Deep-Dive Additions (Tricky & High-Level)**

## **Q31. How Do You Profile a Slow Function in Python?**

✅ Use cProfile:

bash

-m cProfile my\_script.py

Or in-code:

import cProfile

cProfile.run('my\_function()')

## **Q32. What’s the Walrus Operator (:=) and Why Useful?**

**Example:**

if (n := len(my\_list)) > 5:

print(f'List has {n} items')

✅ Avoids **recalculating the same expression.**

## **Q33. How Can You Make a Python Object Immutable?**

✅ Use @dataclass(frozen=True)

Example:

from dataclasses import dataclass

@dataclass(frozen=True)

class Point:

x: int

y: int

## **Q34. How Do You Handle API Rate Limits in Python?**

✅ Use time.sleep() + error handling:

import requests

import time

for i in range(100):

response = requests.get(url)

if response.status\_code == 429:

time.sleep(60) # wait & retry

## **Q35. Explain GIL vs Multiprocessing with Example**

✅ **GIL issue:**

import threading

# CPU-bound: won't scale well due to GIL

✅ **Multiprocessing bypass:**

from multiprocessing import Pool

with Pool(5) as p:

p.map(my\_function, my\_list)

## **Q36. How Would You Serialize a Custom Python Object to JSON?**

✅ Implement \_\_dict\_\_ or custom encoder:

import json

class Person:

def \_\_init\_\_(self, name):

self.name = name

p = Person('Alice')

print(json.dumps(p.\_\_dict\_\_))

## **Q37. How to Handle Missing Keys Gracefully in a Dict?**

✅ Use .get() or defaultdict:

value = my\_dict.get('key', 'default')

## **Q38. How to Avoid Circular Imports?**

✅ Solutions:

* Move imports **inside functions**
* Use **absolute imports**
* Restructure code into **smaller modules**

## **Q39. What’s Monkey Patching?**

**Example:**

import mymodule

mymodule.some\_function = lambda: print('patched!')

✅ Useful for **testing** but dangerous in **prod code.**

## **Q40. Explain List vs Generator Performance**

* **List comprehension:** Eager → builds full list in memory.
* **Generator:** Lazy → yields item-by-item, **saves memory.**

Example:

# Generator

(x for x in range(1000000))

✅ **Follow-up:** When dealing with **large files/data → always prefer generators.**