Python Interview Questions

1) What is the difference between global and local scope?

- A variable created inside a function belongs to the local scope of that function, and can only be used inside that function.
- A variable created in the main body of the Python code is a global variable and belongs to the global scope. Global variables are available from within any scope, global and local.

2) What is an iterator in Python?

- An iterator is an object that contains a countable number of values.
- An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.
- Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods __iter__() and __next__().

3) What is the __init__() function in Python?

- All classes in Python have a function called __init__(), which is always executed when the class is being initiated.
- We can use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created.

4) When should you use lambda functions in Python?

- Use lambda functions when an anonymous function is required for a short period of time.

5) What is the difference between lists, tuples and sets?

Lists, tuples, and sets are all used to store multiple items in a single variable, but they have different properties:

- A list is ordered and changeable. It allows duplicate values.
- A tuple is ordered but unchangeable (immutable). It also allows duplicates.
- A set is unordered, unindexed, and contains only unique items. It is changeable, but you cannot modify individual elements by index.

6) How can you check if all the characters in a string are alphanumeric?

- You can use the `isalnum()` method, which returns True if all the characters are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9).

7) How can you convert a string to an integer?

- You can use the `int()` function, like this:

```
num = "5"
convert = int(num)
```

8) What is indentation in Python, and why is it important?

- Indentation refers to the spaces at the beginning of a code line. Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Python uses indentation to indicate a block of code.
- Python will give you an error if you skip the indentation.

9) What is the correct syntax to output the type of a variable or object in Python?

```
print(type(x))
```

10) Which collection does not allow duplicate members?

- SET

11) What is Inheritance in Python?

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class.

12) What is the output of the following code?

```
x = 41

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

- Above ten,
  and also above 20!

13) Can you list Python's primary built-in data types, in categories?

- Text Type: `str`
- Numeric Types: `int`, `float`, `complex`
- Sequence Types: `list`, `tuple`, `range`
- Mapping Type: `dict`
- Set Types: `set`, `frozenset`
- Boolean Type: `bool`
- Binary Types: `bytes`, `bytearray`, `memoryview`

14) What are Membership Operators?

- Membership operators are used to test if a sequence is present in an object. The `in` and `not in` operators are examples of these:

```
x = ["apple", "banana"]
print("banana" in x) # returns True

x = ["apple", "banana"]
print("pineapple" not in x) # returns True
```

15) Which statement can be used to avoid errors if an if statement has no content?

- The `pass` statement

16) What are Arbitrary Arguments?

- Arbitrary Arguments are often shortened to `*args` in Python documentations.
- If you do not know how many arguments that will be passed into your function, add a `*` before the parameter name in the function definition. This way the function will receive a tuple of arguments, and can access the items accordingly.

17) How can you create and use a Module in Python??

- To create a module just save the code you want in a file with the file extension `.py`:

```
def greeting(name):
  print("Hello, " + name)
```

- Now we can use the module we just created, by using the `import` statement:

```
import mymodule

mymodule.greeting("Jonathan")
```

18) Can you copy a List in Python by simply writing: `list2 = list1`?

- No, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`.
- To make a copy of a list, you can use `copy()` or the `list()` method.

19) How can you return a range of characters of a string?

- You can return a range of characters by using the "slice syntax".
- Specify the start index and the end index, separated by a colon, to return a part of the string, for example:

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[2:5])
```

20) What is a class in Python, and how do you use it?

- A Class is like an object constructor, or a "blueprint" for creating objects.
- You can create a class with the class keyword:

```
class MyClass:
x = 5
```

Now we can use the class named MyClass to create objects:

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

21) When should you use comprehensive function in Python?

Comprehensions in Python (including list, set, and dictionary comprehensions) are a concise and efficient way to create new sequences (lists, sets, dictionaries) from existing iterables. You should use them in the following scenarios:

Transforming Data:, Filtering Data:, Combining Transformation and Filtering: , Creating Dictionaries or Sets:, Improving Readability and Conciseness:

squared_numbers = [x**2 for x in numbers]

even_numbers = [x for x in numbers if x % 2 == 0]

long_words_uppercase = [word.upper() for word in words if len(word) > 5]

my_dict = {k: v for k, v in zip(keys, values)}

22) When should you use generators function in Python?

Python generators are primarily used in scenarios requiring efficient, memory-conscious iteration, particularly when dealing with large datasets or infinite sequences.

Flask Interview Questions

- **What is Flask and why is it called a "microframework"?**
  - Explain its lightweight nature and lack of built-in features compared to full-stack frameworks like Django.
- Explain the purpose of `Flask(__name__)` when creating a Flask application.
  - Discuss how `__name__` helps Flask locate resources like templates and static files.
- **What is routing in Flask, and how do you define routes?**
  - Explain the concept of mapping URLs to view functions using the `@app.route()` decorator.
- **How do you handle HTTP methods (GET, POST, PUT, DELETE) in Flask?**
  - Demonstrate how to specify methods in the `@app.route()` decorator.
- What is the Flask `request` object and how is it used?
  - Explain its role in accessing incoming request data like form data, query parameters, and headers.
- **What are Flask templates and how do you use them with Jinja2?**
  - Discuss the separation of concerns and dynamic content generation.
- Explain the `url_for()` function and its benefits.
  - Discuss dynamic URL generation and avoiding hardcoded URLs.
- **How do you handle errors in Flask (e.g., 404 Not Found)?**
  - Explain the use of `errorhandler()` decorators.

Intermediate Flask Concepts:

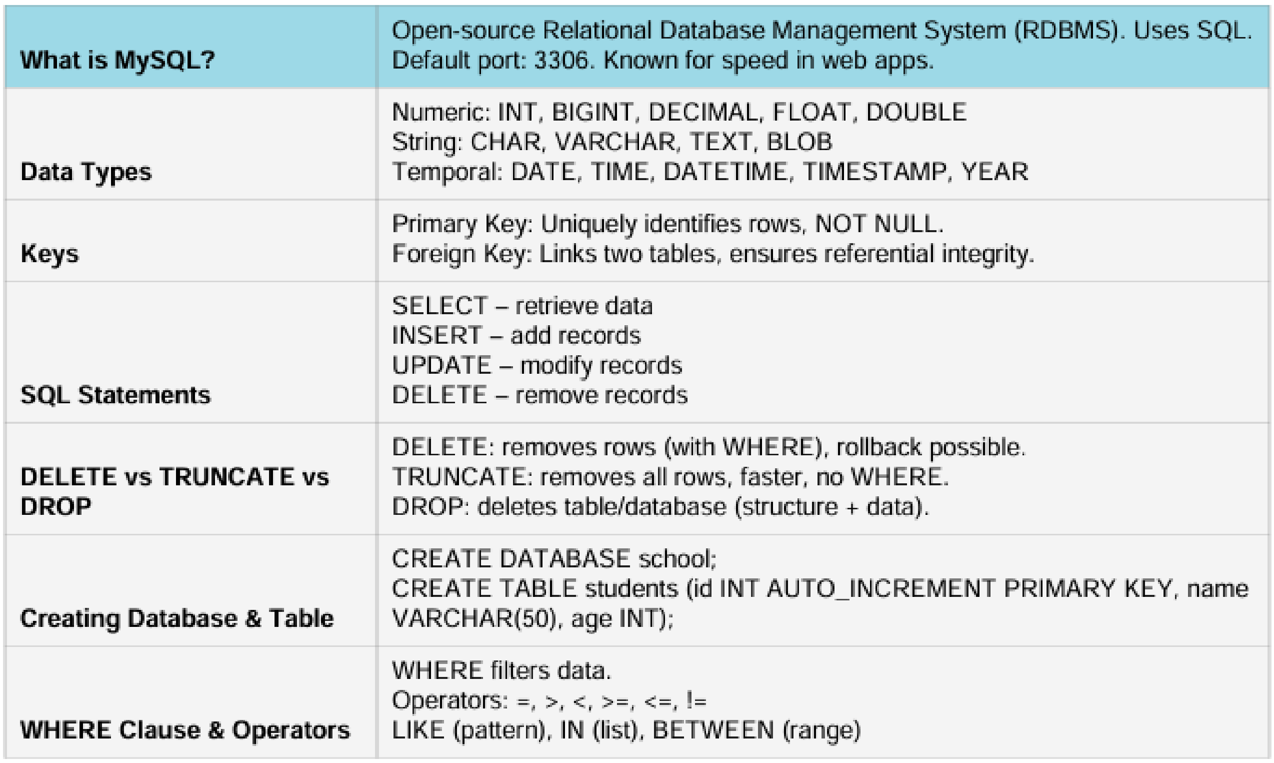
- **How do you manage sessions in Flask?**
  - Discuss session management using the `session` object and its configuration.
- **Explain the concept of blueprints in Flask.**
  - Discuss modularity and organizing larger Flask applications.
- **How do you connect to a database in Flask?**
  - Discuss using ORMs like SQLAlchemy or direct database connectors.
- **What are Flask extensions, and can you give examples?**
  - Discuss extending Flask's functionality with libraries like Flask-SQLAlchemy, Flask-Login, Flask-WTF.
- **How do you handle file uploads in Flask?**
  - Explain using the `request.files` object.
- **Discuss the use of context processors in Flask.**
  - Explain how to inject variables into all templates.

Advanced Flask Concepts and Best Practices:

- **How do you structure a large Flask application for maintainability and scalability?**
  - Discuss project structure, blueprints, and potentially using a factory pattern.
- **Explain how to handle asynchronous tasks in Flask.**
  - Discuss using libraries like Celery or RQ for background tasks.

- **How do you secure a Flask application (e.g., against XSS, CSRF)?**
    - Discuss using Flask-WTF for CSRF protection, input validation, and secure cookie settings.
- **Describe your approach to testing Flask applications.**
    - Discuss unit testing, integration testing, and using Flask's testing client.
- **How do you deploy a Flask application in production?**
    - Discuss using WSGI servers (Gunicorn, uWSGI), web servers (Nginx, Apache), and containerization (Docker).
- **What are the trade-offs between Flask and other Python web frameworks like Django or FastAPI?**
    - Discuss use cases, features, and performance considerations.
- **Explain how Flask supports middleware for cross-cutting concerns.**
    - Discuss request and response hooks for implementing functionalities like logging, authentication, or caching.

MySQL Interview Basics - Quick Cheat Sheet



What is MySQL?	Open-source Relational Database Management System (RDBMS). Uses SQL. Default port: 3306. Known for speed in web apps.
Data Types	Numeric: INT, BIGINT, DECIMAL, FLOAT, DOUBLE String: CHAR, VARCHAR, TEXT, BLOB Temporal: DATE, TIME, DATETIME, TIMESTAMP, YEAR
Keys	Primary Key: Uniquely identifies rows, NOT NULL. Foreign Key: Links two tables, ensures referential integrity.

SQL Statements	SELECT – retrieve data
	INSERT – add records
	UPDATE – modify records
	DELETE – remove records

DELETE vs TRUNCATE vs DROP	DELETE: removes rows (with WHERE), rollback possible. TRUNCATE: removes all rows, faster, no WHERE. DROP: deletes table/database (structure + data).

Creating Database & Table	CREATE DATABASE school;
	CREATE TABLE students (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(50), age INT);

WHERE Clause & Operators	WHERE filters data. Operators: =, >, <, >=, <=, != LIKE (pattern), IN (list), BETWEEN (range)

1. What is MySQL, and how does it differ from other relational database management systems?

- **MySQL**: An open-source Relational Database Management System (RDBMS) that uses SQL (Structured Query Language) to manage and manipulate data.
- **Differences**:
  - **Free and Open Source**: MySQL is widely available (though it also has enterprise editions).
  - **Cross-platform**: Runs on Linux, Windows, macOS.
  - **Performance**: Known for speed in read-heavy applications (like web apps).
  - **Community Support**: Large developer base and active support.
  - **Comparison**: Oracle DB is more enterprise-focused, PostgreSQL is more feature-rich (ACID compliance, advanced functions), while MySQL is simpler and lightweight.

2. Explain the different data types available in MySQL

MySQL data types fall into 3 major categories:

- **Numeric Types**:
  - `INT`, `SMALLINT`, `BIGINT` → whole numbers.
  - `DECIMAL(p,s)`, `NUMERIC` → exact fixed-point numbers.
  - `FLOAT`, `DOUBLE` → approximate floating-point numbers.
- **String (Character) Types**:
  - `CHAR(n)` → fixed length string.
  - `VARCHAR(n)` → variable length string.
  - `TEXT` (tinytext, text, mediumtext, longtext) → long text storage.
  - `BLOB` → binary large object (images, files).
- **Temporal (Date & Time) Types**:
  - `DATE` → YYYY-MM-DD.
  - `TIME` → HH:MM:SS.
  - `DATETIME` → YYYY-MM-DD HH:MM:SS.
  - `TIMESTAMP` → auto-stores time of insert/update.
  - `YEAR` → 4-digit year.

3. What is a primary key and a foreign key in MySQL? How are they used?

- **Primary Key**:
  - Uniquely identifies each row in a table.
  - Must be unique and NOT NULL.
  - Example: `id` in a `users` table.
- **Foreign Key**:
  - Creates a link between two tables.
  - Refers to the primary key in another table.
  - Enforces referential integrity (ensures consistency).

4. Explain the purpose of SELECT, INSERT, UPDATE, and DELETE statements

- **SELECT**: Retrieve data.
- `SELECT name, age FROM students WHERE age > 18;`
- **INSERT**: Add new records.
- `INSERT INTO students (name, age) VALUES ('John', 20);`
- **UPDATE**: Modify existing records.
- `UPDATE students SET age = 21 WHERE name = 'John';`
- **DELETE**: Remove records.
- `DELETE FROM students WHERE age < 18;`

5. What is the difference between DELETE, TRUNCATE, and DROP?

- **DELETE**: Removes rows from a table (can use `WHERE` clause). Keeps structure. Rollback possible.
- **TRUNCATE**: Removes **all rows** from a table quickly. Cannot use `WHERE`. Structure remains. Rollback may not be possible (DDL).
- **DROP**: Deletes the **entire table/database** (structure + data). Cannot rollback.

6. How do you create a database and a table in MySQL?

- **Create Database**:
- `CREATE DATABASE school;`
- **Create Table**:

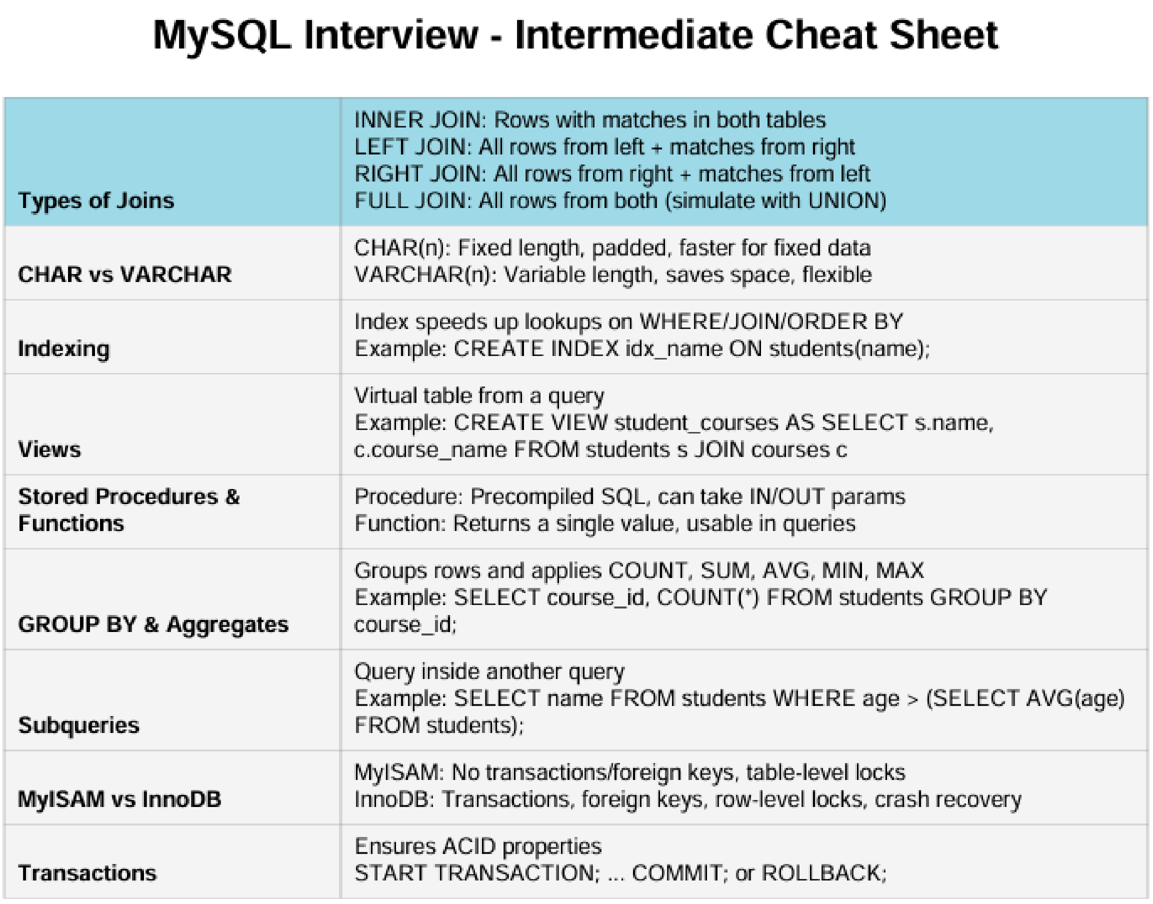
```
CREATE TABLE students (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    age INT,
    grade VARCHAR(10)
);
```

7. What is the default port for MySQL server?

- **3306**

8. Explain the use of WHERE clause and common operators

- **WHERE clause**: Filters records that meet a condition.
- `SELECT * FROM students WHERE age > 18;`
- **Common Operators**:
  - `=` → equal
  - `>` , `<`, `>=`, `<=` → comparison
  - `<>` or `!=` → not equal
  - `LIKE` → pattern matching (`'J%'` = starts with J)
  - `IN` → check within a list (`IN (18, 20, 22)`)
  - `BETWEEN` → range (`BETWEEN 18 AND 25`)



MySQL Interview - Intermediate Cheat Sheet

Types of Joins	INNER JOIN: Rows with matches in both tables LEFT JOIN: All rows from left + matches from right RIGHT JOIN: All rows from right + matches from left FULL JOIN: All rows from both (simulate with UNION)
CHAR vs VARCHAR	CHAR(n): Fixed length, padded, faster for fixed data VARCHAR(n): Variable length, saves space, flexible
Indexing	Index speeds up lookups on WHERE/JOIN/ORDER BY Example: CREATE INDEX idx_name ON students(name);
Views	Virtual table from a query Example: CREATE VIEW student_courses AS SELECT s.name, c.course_name FROM students s JOIN courses c
Stored Procedures & Functions	Procedure: Precompiled SQL, can take IN/OUT params Function: Returns a single value, usable in queries
GROUP BY & Aggregates	Groups rows and applies COUNT, SUM, AVG, MIN, MAX Example: SELECT course_id, COUNT(*) FROM students GROUP BY course_id;
Subqueries	Query inside another query Example: SELECT name FROM students WHERE age > (SELECT AVG(age) FROM students);
MyISAM vs InnoDB	MyISAM: No transactions/foreign keys, table-level locks InnoDB: Transactions, foreign keys, row-level locks, crash recovery
Transactions	Ensures ACID properties START TRANSACTION; ... COMMIT; or ROLLBACK;

1. Types of Joins in MySQL

- **INNER JOIN** → Returns rows with matching values in both tables.
- `SELECT s.name, c.course_name`
- `FROM students s`
- `INNER JOIN courses c ON s.course_id = c.id;`
- **LEFT JOIN** → Returns all rows from the left table + matching rows from the right.
- **RIGHT JOIN** → Returns all rows from the right table + matching rows from the left.
- **FULL JOIN (simulated with UNION)** → Returns rows when there is a match in either table.

2. Difference Between CHAR and VARCHAR

- **CHAR(n)**: Fixed length. Always uses *n* bytes, padded with spaces if shorter. Faster for fixed-size data (e.g., country codes, PINs).
- **VARCHAR(n)**: Variable length. Uses only required storage + 1 or 2 bytes overhead. Better for variable text (e.g., names, emails).

3. Indexing in MySQL

- **Index**: Data structure (B-Tree, Hash, etc.) that improves speed of retrieval operations.
- Without index → full table scan.
- With index → faster lookups on columns used in `WHERE`, `JOIN`, `ORDER BY`.
- Example:
- `CREATE INDEX idx_name ON students(name);`

4. Views in MySQL

- A **view** is a virtual table based on a query.
- Used to simplify queries, provide security, and hide complexity.
- Example:
- `CREATE VIEW student_courses AS`
- `SELECT s.name, c.course_name`
- `FROM students s`
- `JOIN courses c ON s.course_id = c.id;`
-
- `SELECT * FROM student_courses;`

5. Stored Procedures and Functions

- **Stored Procedure**: Precompiled SQL code stored in the DB. Can accept IN/OUT parameters.

```
CREATE PROCEDURE GetStudents()
BEGIN
    SELECT * FROM students;
END;
```

Call with: `CALL GetStudents();`

- **Function**: Returns a single value. Used in queries.

```
CREATE FUNCTION GetStudentCount() RETURNS INT
RETURN (SELECT COUNT(*) FROM students);
```

6. GROUP BY and Aggregate Functions

- **GROUP BY** groups rows with the same values into summary rows.
- Often used with aggregate functions:
  - `COUNT()` → number of rows
  - `SUM()` → total
  - `AVG()` → average
  - `MIN()/MAX()` → smallest/largest
- Example:

```
SELECT course_id, COUNT(*) AS total_students
FROM students
GROUP BY course_id;
```

7. Subquery in MySQL

- A query inside another query.
- Example:

```
SELECT name FROM students
WHERE age > (SELECT AVG(age) FROM students);
```

→ Finds students older than the average age.

8. MyISAM vs InnoDB Storage Engines

Feature	MyISAM	InnoDB (default)
Transactions	❌ Not supported	✅ Supported
Foreign Keys	❌ No	✅ Yes
Locking	Table-level	Row-level
Speed	Faster for reads	Better for mixed workloads
Reliability	Less crash-safe	Crash recovery

9. Transactions in MySQL

- A **transaction** = group of SQL statements executed as a single unit.
- Ensures **ACID properties** (Atomicity, Consistency, Isolation, Durability).
- Commands:

```
START TRANSACTION;
UPDATE accounts SET balance = balance - 500 WHERE id = 1;
UPDATE accounts SET balance = balance + 500 WHERE id = 2;
COMMIT;  -- saves changes
ROLLBACK; -- undo changes if error
```

1. Logical Architecture of MySQL

- **Client Layer**: Handles connections, authentication, security.
- **SQL Layer (Parser & Optimizer)**: Parses queries, checks syntax, optimizes execution plans.
- **Storage Engine Layer**: Manages how data is stored/retrieved (e.g., InnoDB, MyISAM).
- **File System & OS**: Actual storage of data files, logs, indexes.

2. Normalization vs Denormalization

- **Normalization**: Process of organizing data to reduce redundancy and improve integrity.
  - Example: Splitting customer details into separate `customers` and `orders` tables.
  - Used when consistency and minimal redundancy are critical.
- **Denormalization**: Adding redundancy to speed up reads (e.g., combining tables for fewer joins).
  - Used in analytics/reporting systems for performance.

3. Optimizing Slow Queries

- **Use `EXPLAIN`**: Analyzes how MySQL executes a query. Shows indexes used, join types, scan methods.
- `EXPLAIN SELECT name FROM students WHERE age > 20;`
- **Best Practices**:
  - Add indexes to frequently queried columns.
  - Use `LIMIT` when possible.
  - Avoid `SELECT *`.
  - Optimize joins (ensure indexed keys).
  - Analyze with `SHOW PROFILE`, `slow_query_log`.

4. Sharding and Scaling

- **Vertical Scaling**: Add more resources (CPU, RAM, SSD) to one server.
- **Horizontal Scaling (Sharding)**: Splitting database into smaller shards across multiple servers.
  - Example: Users A–M on one shard, N–Z on another.
- Improves performance for very large datasets.

5. ACID Properties in MySQL

- **Atomicity**: Transactions execute fully or not at all.
- **Consistency**: DB moves from one valid state to another.
- **Isolation**: Multiple transactions execute independently.
- **Durability**: Data persists even after crash.
- **InnoDB Engine** provides ACID compliance with `START TRANSACTION`, `COMMIT`, `ROLLBACK`.

6. Backup & Recovery Strategies

- **Logical Backup**: Using `mysqldump` to export schema/data.
- **Physical Backup**: Copying database files directly.
- **Point-in-time Recovery**: Use binary logs (`mysqlbinlog`).
- **Replication-based backup**: Use a replica server for recovery.

7. Preventing SQL Injection

- Use **Prepared Statements/Parameterized Queries**:
- `cursor.execute("SELECT * FROM users WHERE id = %s", (user_id,))`
- Never concatenate user input directly.
- Use ORM frameworks.
- Apply least-privilege principle for DB users.

8. HAVING vs WHERE

- **WHERE**: Filters rows **before** grouping.
- **HAVING**: Filters groups **after** aggregation.
- Example:
- `SELECT course_id, COUNT(*) AS total`
- `FROM students`
- `GROUP BY course_id`
- `HAVING COUNT(*) > 5;`

9. Concurrency & Locking

- **Locks** prevent conflicts during simultaneous transactions.
- **Types**:
  - **Table-level lock**: Entire table locked (MyISAM).
  - **Row-level lock**: Only specific rows locked (InnoDB).
- **Isolation Levels**:
  - READ UNCOMMITTED
  - READ COMMITTED
  - REPEATABLE READ (default in InnoDB)
  - SERIALIZABLE

10. Triggers in MySQL

- A **trigger** is a set of SQL statements automatically executed in response to `INSERT`, `UPDATE`, or `DELETE`.
- Use cases: