Joins

Sr. No.	Question	Pg. No.
1.	What are the different types of SQL joins (INNER, LEFT, RIGHT, FULL, CROSS, etc.) and when would you use each?	
2.	What is the difference between a CROSS JOIN and a FULL OUTER JOIN?	
3.	Write a SQL query to retrieve the first and last names of employees along with the names of their managers (given Employees and Managers tables).	
4.	Write a SQL query to find the average salary for each department, given tables Employees (with DepartmentID) and Departments (with DepartmentName).	
5.	Write a SQL query to list all products that have never been ordered (products in a Product table with no matching rows in the Orders table).	
6.	Write a SQL query to list all employees who are also managers (for example, employees who appear as managers in the same table).	
7.	What is a self-join, and when might you use it? Provide an example scenario.	
8.	How would you join more than two tables in a single SQL query? What factors affect the performance when joining multiple tables?	
9.	Explain how an OUTER JOIN works when one side has no matching rows. How does this differ from an INNER JOIN in practice?	

Indexing and Keys

Sr. No.	Question	Pg. No.
1.	What is a SQL index and what are different types of indexes (clustered, non-clustered, unique, etc.)?	1
2.	What is the difference between a heap (no clustered index) and a table with a clustered index, and how can you identify a heap table?	2
3.	What is the difference between a PRIMARY KEY and a UNIQUE KEY (or unique index) in SQL?	
4.	What are index “forwarding pointers” in a heap table, and how do they affect query performance?	
5.	What is a composite index, and how do you choose the order of columns in it for optimal performance?	
6.	When should you use a covering index, and how does it improve the performance of a query?	
7.	How does the existence of an index on a column affect INSERT, UPDATE, and DELETE performance on a table?	
8.	What is index selectivity, and why is it important for query optimization?	

Sr. No.	Question	Pg. No.
9.	How many clustered indexes can a table have, and why?	
10.	What is index fragmentation, and how can it be resolved or mitigated in a large database?	

Normalization and Schema Design

Sr. No.	Question	Pg. No.
1.	What is database normalization and what are the normal forms (1NF, 2NF, 3NF, BCNF)?	
2.	Given a table with repeating groups of data (e.g. Customer and multiple phone numbers), how would you normalize the table to 3NF?	
3.	Explain how denormalization can be used for performance, and what the trade-offs are (e.g. redundancy vs. speed).	
4.	What kinds of data anomalies (insertion, update, deletion anomalies) are prevented by normalization?	
5.	How would you design a table schema to handle a many-to-many relationship, for example between Students and Courses?	
6.	Explain the concept of denormalization with an example scenario in a large database.	
7.	What is a surrogate key vs a natural key, and when would you choose each in table design?	
8.	How do you implement a 1-to-1 relationship versus a 1-to-many relationship in table design?	
9.	How do you enforce referential integrity without foreign key constraints? (e.g. using triggers)	
10.	How would you handle schema migrations or changes in a production database environment?	

Stored Procedures and Functions

Sr. No.	Question	Pg. No.
1.	What is the difference between a stored procedure and a user-defined function in SQL (aside from return value)?	
2.	What are the advantages and disadvantages of using stored procedures?	
3.	Can you perform INSERT/UPDATE/DELETE operations inside a SQL function? Why or why not?	
4.	What is a table-valued function and when would you use one in a query?	

Sr. No.	Question	Pg. No.
5.	When would you use a stored procedure instead of inline SQL queries in an application?	
6.	How do you pass parameters to and receive results from stored procedures?	
7.	How would you debug or test a slow or failing stored procedure in production?	
8.	How do you grant a user permission to execute a specific stored procedure?	

Views

Sr. No.	Question	Pg. No.
1.	What is a database view, and can you update data in the base tables through it?	
2.	What is the difference between a standard view and a materialized (or indexed) view?	
3.	What happens if a materialized view is being refreshed (complete refresh) and a user queries it at the same time?	
4.	When would you use a view in a database design? What benefits do views provide (e.g. security, abstraction)?	
5.	Can you create an index on a view? If so, what are the implications (e.g. indexed view in SQL Server)?	
6.	How do you modify or drop a view if the underlying table schema changes?	
7.	What is the difference between a view and a temporary table?	

Triggers

Sr. No.	Question	Pg. No.
1.	What is a trigger in SQL, and when would you use one? Give an example use case.	
2.	What is the difference between an AFTER trigger and an INSTEAD OF trigger (e.g. in SQL Server)?	
3.	What are the “inserted” and “deleted” magic tables in SQL Server triggers?	
4.	How can triggers be used to enforce business rules or data integrity (e.g. auditing changes, simulating foreign keys)?	
5.	What are the potential drawbacks of using triggers (such as performance impact or hidden logic)?	
6.	How do INSTEAD OF triggers on a view work?	
7.	Can triggers call stored procedures, and are there any limitations to doing that?	

Transactions and Concurrency