Here are 10 questions for each difficulty level in databases:

**Easy:**

1. You've written a SQL query to fetch data from a table, but it's returning no results. What are some common reasons why this might happen?

2. What is SQL injection, and how can it be prevented?

3. Explain the difference between a database index and a database constraint.

4. What is a foreign key in a relational database, and why is it important?

5. How would you check if a table exists in a database using SQL?

6. What is a database transaction, and why is it important?

7. How do you connect to a MySQL database using Python?

8. What is the purpose of the GROUP BY clause in SQL?

9. Explain the difference between the WHERE and HAVING clauses in SQL.

10. How do you create a backup of a database in SQL Server Management Studio?

**Medium:**

1. You're getting a "deadlock detected" error in your database. What does this mean, and how would you resolve it?

2. What are the advantages and disadvantages of using an ORM (Object-Relational Mapping) tool?

3. Explain the concept of database normalization and provide an example.

4. How would you handle a situation where a database query is taking too long to execute?

5. Describe the difference between a left outer join and a right outer join.

6. What are database triggers, and when would you use them?

7. How can you improve the performance of a SQL query that's running slowly?

8. What is a composite primary key, and when would you use one?

9. Explain the concept of database transactions isolation levels.

10. How do you handle database schema changes in a production environment without causing downtime?

**Hard:**

1. You're working on a distributed database system, and you notice that some data is inconsistent across nodes. How would you diagnose and fix this issue?

2. Describe the process of database replication, including its benefits and potential challenges.

3. Explain the difference between optimistic and pessimistic concurrency control in databases.

4. What strategies would you use to implement high availability and fault tolerance in a database system?

5. You're tasked with designing a database schema for a multi-tenant SaaS application. What considerations would you take into account?

6. How would you design a database schema for a social networking platform like Facebook or Twitter?

7. Describe the concept of data partitioning in databases. When is it beneficial, and how would you implement it?

8. What are materialized views in databases, and how do they differ from regular views?

9. How would you handle a situation where a database query is causing excessive locking and blocking other queries?

10. Explain the concept of database normalization anomalies and how to avoid them during database design.

Sure, here are examples of SQL debugging questions classified by difficulty level:

**### Easy:**

1. \*\*Missing semicolon\*\*: Given the following SQL statement, identify and correct the syntax error:

```sql

SELECT \* FROM Customers WHERE City = 'New York'

SELECT \* FROM Orders WHERE Quantity > 10;

```

2. \*\*Incorrect table alias\*\*: Fix the error in this SQL query:

```sql

SELECT o.OrderID, c.CustomerName

FROM Orders o

INNER JOIN Customers c ON o.CustomerID = c.CustomerID

WHERE o.OrderDate > '2023-01-01';

```

3. \*\*Invalid column name\*\*: Identify and fix the error in this SQL query:

```sql

SELECT ProductName, Price FROM Products

WHERE Category = 'Electronics' AND Brand = 'Samsung';

```

**### Medium:**

4. \*\*Data type mismatch\*\*: Find and resolve the data type mismatch error in this SQL query:

```sql

SELECT CustomerName, SUM(OrderAmount)

FROM Orders

GROUP BY CustomerName;

```

5. \*\*Ambiguous column reference\*\*: Correct the ambiguous column reference error in this SQL query:

```sql

SELECT OrderID, CustomerName

FROM Orders

JOIN Customers ON Orders.CustomerID = Customers.CustomerID

JOIN Products ON Orders.ProductID = Products.ProductID;

```

6. \*\*Using aggregate functions incorrectly\*\*: Identify and fix the issue with this SQL query:

```sql

SELECT CustomerID, COUNT(OrderID), SUM(OrderAmount)

FROM Orders

GROUP BY CustomerID;

```

**### Hard:**

7. \*\*Subquery optimization\*\*: Optimize the following SQL query to improve performance:

```sql

SELECT CustomerID, CustomerName

FROM Customers

WHERE CustomerID IN (SELECT DISTINCT CustomerID FROM Orders WHERE OrderAmount > 1000);

```

8. \*\*Recursive CTE\*\*: Implement a recursive common table expression (CTE) to retrieve all the ancestors of a given node in a hierarchical data structure stored in a table.

9. \*\*Deadlock resolution\*\*: Analyze and resolve a deadlock situation occurring between two concurrent transactions accessing the same set of tables.

These debugging questions require a deeper understanding of SQL syntax, database design principles, and optimization techniques, making them suitable for intermediate to advanced developers.