Q1

SQL injection is possible whenever unsanitized user input or data is placed into SQL queries and reaches the SQL interpreter. This allows attackers to use special characters (like ‘) in order to insert their own SQL commands into queries and leak data from database.

Q2

- The most important way to prevent SQLi is **sanitizing and validating** user inputs. Sanitizing includes:
    1. Escaping special characters
    2. Ensuring that user input is in expected format, length, and data type
    3. Check for NULL inputs and blank strings whenever needed
    4. Using regex for input format validation
- Turn off the **visibility of database errors** on production sites. Database errors can be used with SQL Injection to gain information about your database.
- Use **Parameterized Statements**, the purpose of parameterized queries is to provide parameters, then connect values to those parameters, and then execute the query. Queries executed in such a way will not be susceptible to injection attacks because the query and the parameters will be sent over for execution separately.
- **Limit the privileges** of database accounts used by different parts and sites of your application. Avoid using accounts with unnecessary permissions. For example, if the product list is only SELECTing the rows of database, it should not be able to modify or delete them. You can also use views, SQL procedures and functions and other features of databases for this purpose.

Q3

- **In-Band SQLi**
  when an attacker is able to use the same communication channel to both launch the attack and gather results

    - **Error-Based SQLi**
      the attacker performs actions that cause the database to produce error messages
    - **Union-Based SQLi**
      this technique takes advantage of the UNION SQL operator, which fuses multiple select statements generated by the database to get a single HTTP response

- **Inferential (Blind) SQLi**
  The attacker cannot directly see the responses of the injected queries in Inferential SQLi because the data is not transferred between the web applications. Instead, these kinds of vulnerabilities are exploited by observing the behavior of the application in order to enumerate the database. These attacks are typically slower, but as harmful.

    - **Boolean-Based SQLi**
      the attacker sends an SQL query to the database which forces the application to return a different result depending on whether the query returns a TRUE or FALSE result.
    - **Time-Based SQLi**
      the attacker sends a query that forces the application to wait for a specific duration before returning a response. The attacker uses the response time to determine whether the result of a query is TRUE or FALSE.

- **Out-of-Bound SQLi**
  this attack is performed when the attacker can't use the same channel to launch the attack and gather information, and is very uncommon because certain features must be enabled on the database server. These techniques count on the capacity of the server to create DNS or HTTP requests to transfer data to an attacker, for example `xp_dirtree` in Microsoft SQL Server which can be used to send DNS requests to a server controlled by the attacker.

Q4

- Using automated scanning tools such as
  - SQLMap
  - Nessus
  - Acunetix
  - OWASP ZAP
- Conducting thorough code reviews - specially on parts of the application that perform SQL queries - and looking for poor written codes and queries (which lack proper anti-sqli techniques such as sanitization) can be very useful for detecting SQLi vulnerabilities.
- Conducting SQL injection tests as a malicious user.
  The tester has to make a list of all input fields whose values could be used in crafting a SQL query, including the hidden fields of POST requests, and then test them separately, trying to interfere with the query and to generate an error. Consider also HTTP headers and Cookies. This can be started by injecting a special character such as ' or ` or -- into input fields and observing the behavior of the app.

Q5

- Leakage of sensitive data from the database
- modifying database data (insert/update/delete)
- executing administration operations on the database (such as shutdown the DBMS)

- recovering the content of a given file existing on the DBMS file system or write files into the file system
- issuing commands to the operating system
- executing resource-intensive SQL queries and causing DoS