Database Connectivity with Java

Prativa Nyaupane

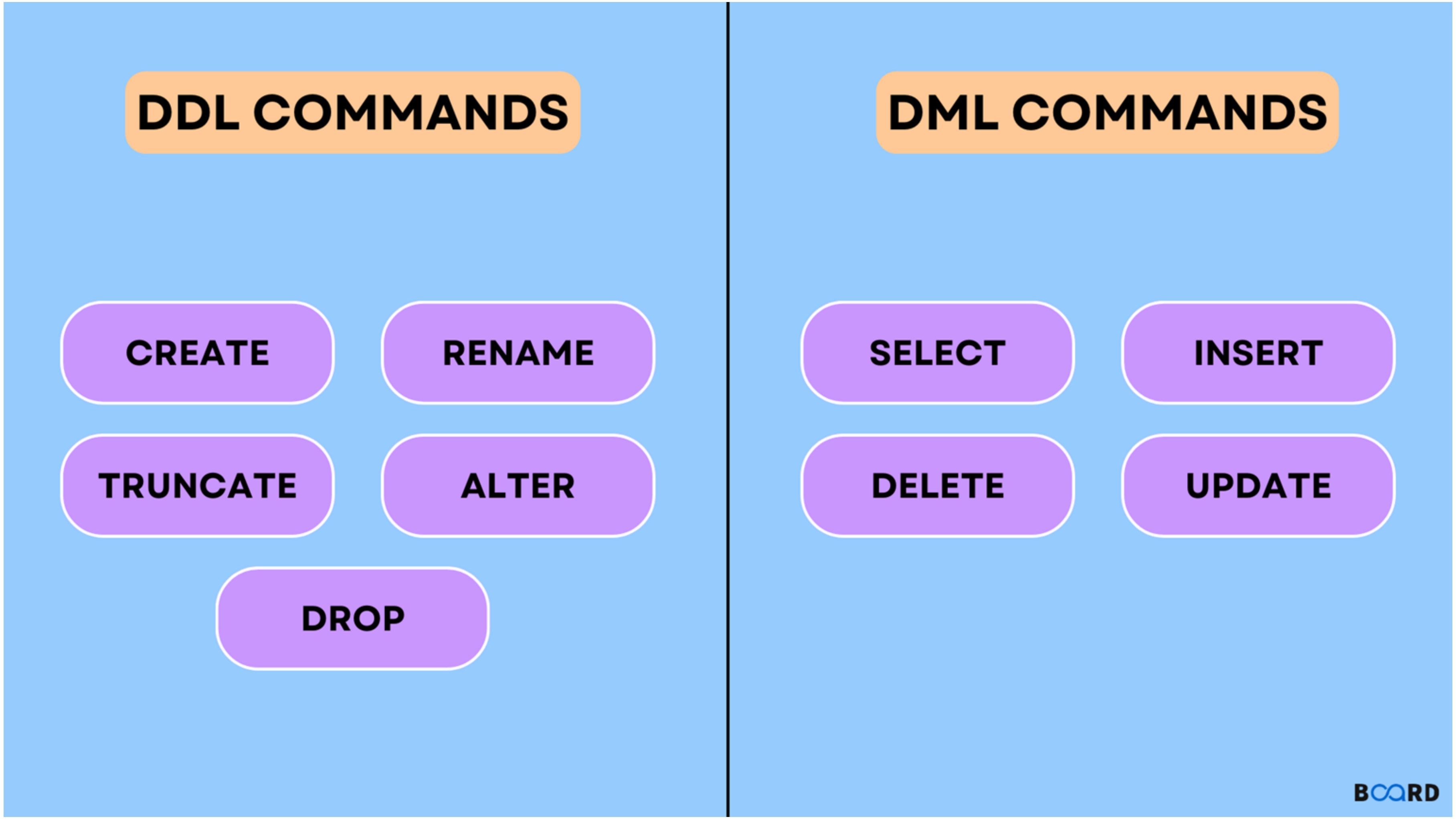
Recap

- JDBC Driver Types and Configuration
- Managing Connections and Statements
- Result Sets and Exception Handling

Outline

DDL and DML Operations

- DDL (Data Definition Language) Operations:
    - It is a programming language for creating and modifying database objects such as tables, indices and users. DDL deals with the structure of the database.
    - Operations include creating, altering, and deleting database objects like tables, indexes, and views.
    - Examples: CREATE TABLE, ALTER TABLE, DROP TABLE.
    - `DROP TABLE employees;`
- DML (Data Manipulation Language) Operations:
    - It is a programming language for adding(inserting), deleting and modifying data in a database. DML deals with the data stored in the database.
    - Operations include inserting, updating, and deleting data in tables.
    - Examples: INSERT INTO, UPDATE, DELETE FROM.
    - `INSERT INTO employees (first_name, last_name, fname) VALUES ('John', 'Capita', 'xcapit00');`



DDL COMMANDS

- CREATE
- RENAME
- TRUNCATE
- ALTER
- DROP

DML COMMANDS

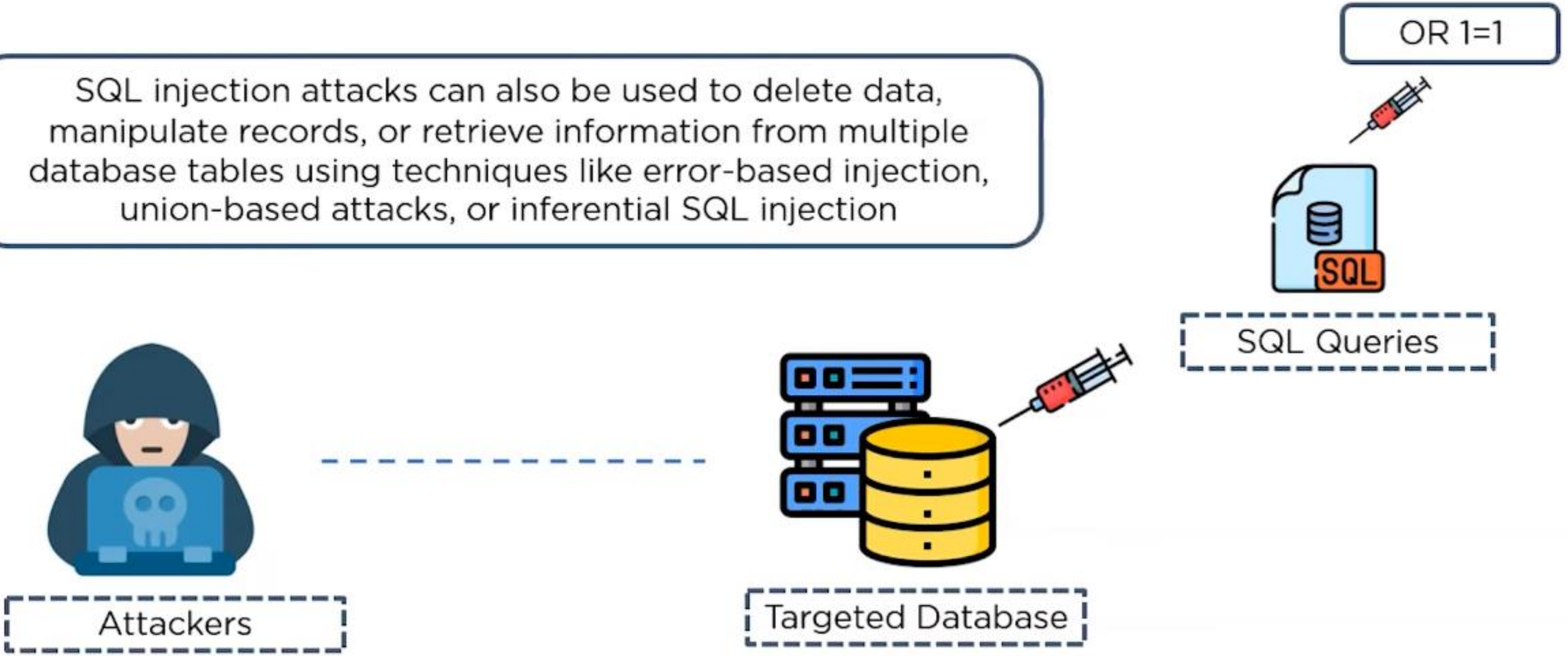
- SELECT
- INSERT
- DELETE
- UPDATE

SQL Injection

- SQL Injection is a code injection technique where attackers insert malicious SQL statements into input fields or queries.
- This can lead to unauthorized access to databases, extraction of sensitive information, or even the deletion or manipulation of data.
- SQL Injection can result in unauthorized access, data exfiltration, data manipulation, or even complete compromise of a database.
- Three types of SQL Injection are:
  - Classic SQL Injection
  - Blind SQL injection
  - Time-base Blind SQL Injection

How Does a SQL Injection Attack Work?

SQL injection attacks can also be used to delete data, manipulate records, or retrieve information from multiple database tables using techniques like error-based injection, union-based attacks, or inferential SQL injection



Classic SQL Injection

Attackers inject malicious SQL code through user inputs like forms or URL parameters. The injected SQL code is typically used to extract data from the database or manipulate its behavior. The attacker can often see the results of their injected SQL code directly in the application's response, such as retrieving usernames, passwords, or other sensitive information.

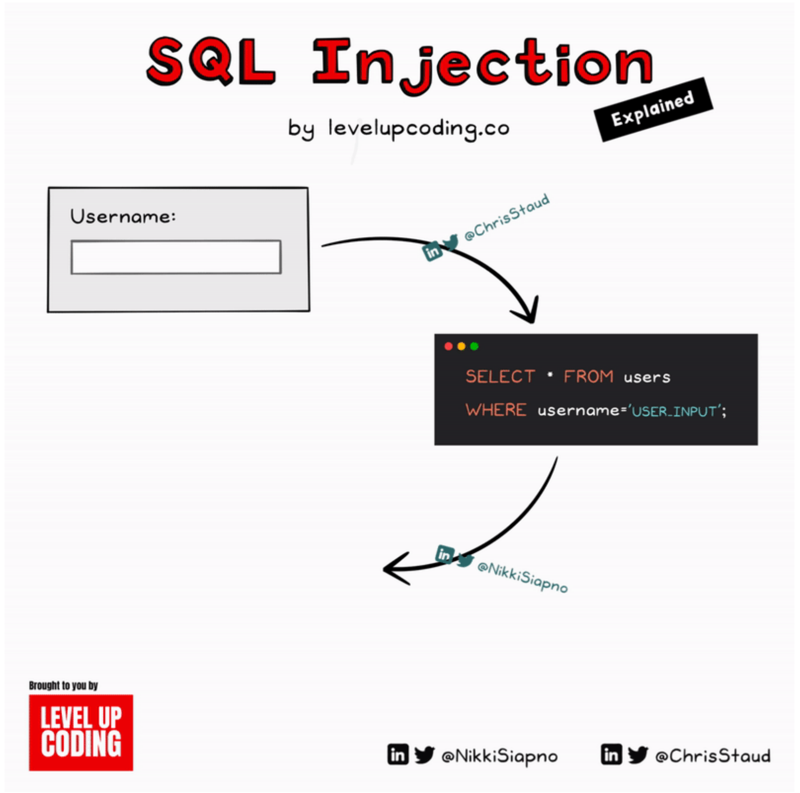
Example: Imagine an app that returns all your information after logging in. That query may look like the following:

SELECT * FROM users WHERE username = 'USER_INPUT';

If an attacker were to submit a malicious input, the query could change to the following:

SELECT * FROM users WHERE username = '' OR '1'='1';

This query will return all users as '1'='1' will always return true.



```
SELECT * FROM users
WHERE username='USER_INPUT';
```

- Blind SQL Injection:
  - Attackers infer the success or failure of injected queries without directly retrieving results.
  - Example: Modifying a query to cause a delay, and then observing if the page takes longer to load. Suppose we have a simple table called **users** with columns **username** and **password**, and we want to check if a user with a given username and password exists. We can use the following query:

    SELECT * FROM users WHERE username = 'username' AND password = 'password'

  - To perform a blind SQL injection, an attacker might exploit this query to bypass authentication without knowing valid credentials. They can inject a condition that is always true. For instance:

    SELECT * FROM users WHERE username = 'admin' AND '1'='1' -- ' AND password = 'anything'

  - In this injection, '1'='1' -- ' is always true due to the equality condition. The -- at the end comments out the rest of the original query to avoid syntax errors. The attacker doesn't need to know the actual password because the injected condition always evaluates to true, granting access to the account associated with the username 'admin'.

Time-Based Blind SQL Injection

- It is a type of blind/inferential injection attack.
- Delays the server's response to infer the success of a query.
- In a time-based attack, an attacker sends an SQL command to the server with code to force a delay in the execution of the queries.
- The response time indicated whether the result of the query is true or false.
- Example: Adding a sleep function to a query and observing if the delay occurs.

```
/* Resulting query (with malicious SLEEP injected). */
        SELECT * FROM table  WHERE id=1-SLEEP(15)
```

- When an attacker tries to use these functions in the query and if he is successful in slowing the response, it proves SQL injection is possible and the server is using MySQL as a database.

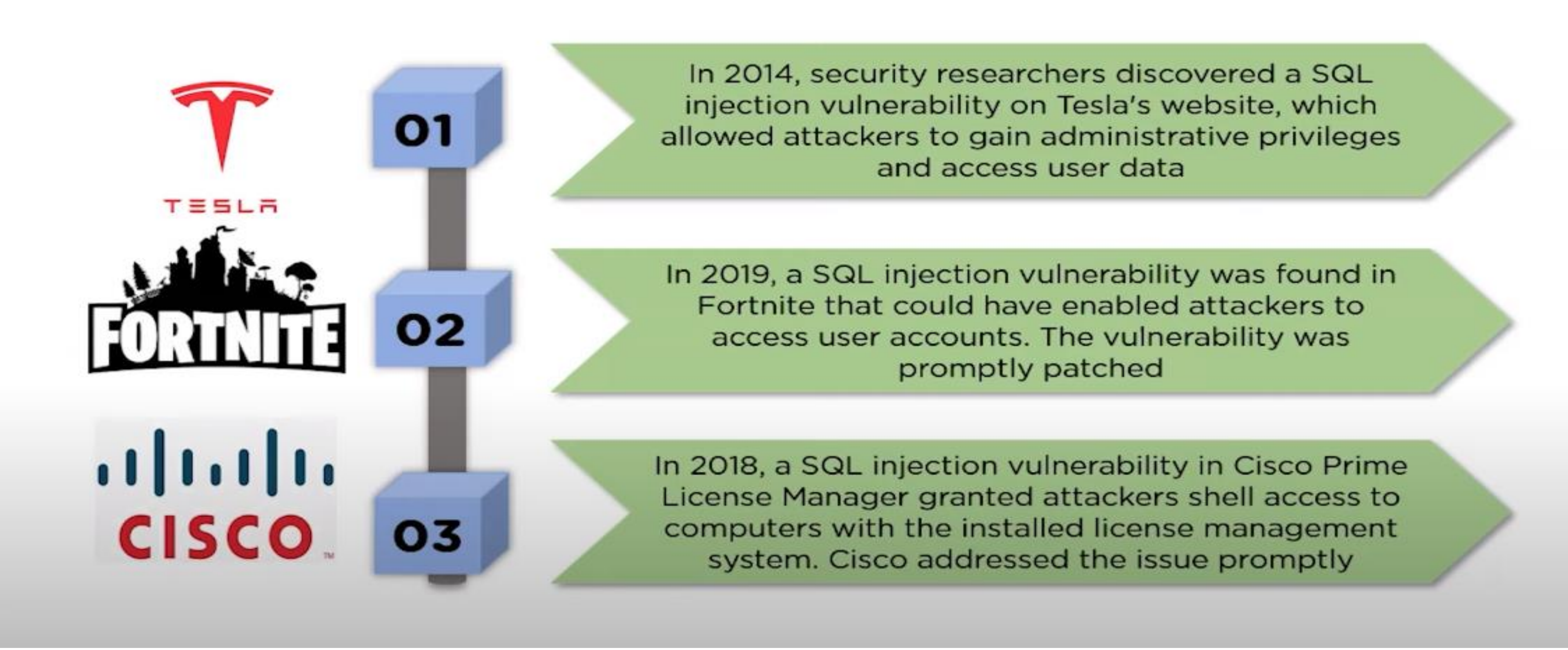
Detecting and Preventing SQL Injection Attacks

Train employees on prevention methods	01	Educate your IT teams, including developers and system administrators, on SQL injection attack vectors and prevention techniques
Implement input validation and parameterized queries	02	Validate and filter user input using an allowlist approach, parameterized queries to separate SQL code from user input
Regular security scans	03	Perform regular security scans to identify and address potential vulnerabilities in web applications
Keep software up to date	04	Ensure that all software, including databases and programming languages, are regularly updated to include the latest security patches and protections against SQL injection

Impact of SQL Injection Attacks

- Unauthorized access to sensitive information and resources
- Potential data breaches and exposure of confidential data
- Data manipulation or deletion
- Network infiltration and system compromise
- Loss of customer trust and decreased revenue
- Reputational damage and long-term consequences for the business

Real-Life SQL Injection Examples



01

In 2014, security researchers discovered a SQL injection vulnerability on Tesla's website, which allowed attackers to gain administrative privileges and access user data

02

In 2019, a SQL injection vulnerability was found in Fortnite that could have enabled attackers to access user accounts. The vulnerability was promptly patched

03

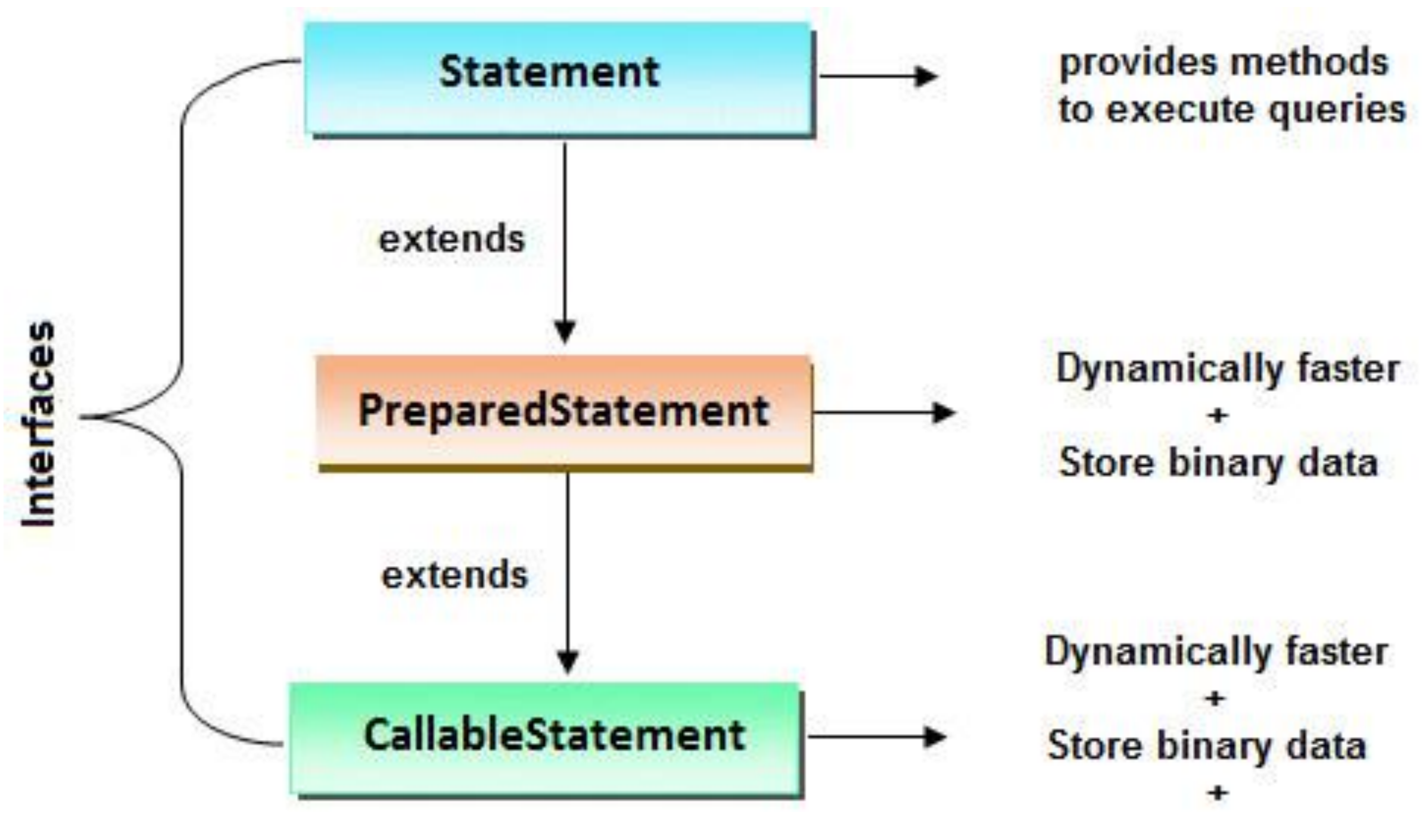
In 2018, a SQL injection vulnerability in Cisco Prime License Manager granted attackers shell access to computers with the installed license management system. Cisco addressed the issue promptly

Prepared Statements

- In DBMS, a prepared statement, parameterized statement or parameterized query is a feature where the database pre-compiles SQL code and stores the results, separating it from data.
  - It provides efficiency, as they can be used repeatedly without re-compiling.
  - It enhances security by reducing or eliminating SQL injection attacks.
- Parameterized queries created and compiled once, then reused with different parameters.

Using Prepared Statement

- A Prepared Statement is a feature of JDBC that allows the creation of parameterized SQL statements.
- It enables the application to execute the same SQL statement repeatedly with different parameter values, improving performance and security.
- Prepared Statements are precompiled and cached by the database, reducing the need for repetitive parsing and optimizing query execution.


Interfaces
Statement
extends
PreparedStatement
extends
CallableStatement
provides methods to execute queries
Dynamically faster + Store binary data
Dynamically faster + Store binary data +

Example of Prepared Statement

```
Connection con = DriverManager.getConnection("----", "system", "oracle");


PreparedStatement stmt =

con.prepareStatement("insert into Emp values(?,?)");


stmt.setInt(1,101);//1 specifies the first parameter in the query

stmt.setString(2,"Ratan");


int i=stmt.executeUpdate();


System.out.println(i+" records inserted");
```

Creating Database Connection

Prepare statement to replace ‘?’ with required information

Replacing ‘?’ with required value in the query.

Execute the prepared statement

Methods of PreparedStatement interface

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Row Sets and Transactions

- Row Sets:
    - A disconnected set of rows from a ResultSet.
    - Allows for more flexibility in data manipulation, especially in disconnected or distributed environments.
    - Types include CachedRowSet, WebRowSet, and FilteredRowSet.
- Transactions:
    - A sequence of one or more SQL statements executed as a single unit of work.
    - Ensures data consistency by either committing all changes or rolling back to the initial state.
    - JDBC supports transaction management using commit and rollback operations.

SQL Escapes

- Mechanism for representing special characters or reserved keywords in SQL queries.
- Helps prevent syntax errors or conflicts in SQL statements.
- Examples include using double quotes for identifiers or escaping special characters like apostrophes.

  SELECT * FROM my_table WHERE **column_name** LIKE **'O''Connor'** ;

  We're searching for records where the value in column_name is "O'Connor".

  The single quote character within the string "O'Connor" is **escaped by doubling it ('')**. This is the standard way to escape single quotes in SQL strings.

Thank You