Bind Variables has 2 advantages  
 1) Performance  
 2) Avoids SQL Injection

To understand bind variables, consider an application that generates thousands of SELECT statements against a table; for example:

SELECT fname, lname, pcode FROM cust **WHERE id = 674;**  
SELECT fname, lname, pcode FROM cust **WHERE id = 234;**  
SELECT fname, lname, pcode FROM cust **WHERE id = 332;**

Each time the query is submitted, Oracle first checks in the shared pool to see whether this statement has been submitted before. If it has, the execution plan that this statement previously used is retrieved, and the SQL is executed. If the statement cannot be found in the shared pool, Oracle has to go through the process of parsing the statement, working out the various execution paths and coming up with an optimal access plan before it can be executed. This process is know as a «**hard parse**»

* In a **soft parse**, the SQL statement already exists in a shared pool, so very little processing is required for access rights and session verification.

The way to get Oracle to reuse the execution plans for these statements is to use bind variables. **Bind variables are «substituion» variables that are used in place of literals** (such as 674, 234, 332) and that have the effect of sending exactly the same SQL to Oracle every time the query is executed. For example, in our application, we would just submit

SELECT fname, lname, pcode FROM cust **WHERE id = :cust\_no;**

**SQL injection:**

"SELECT \* FROM emp WHERE emp\_name = '" + empName + "';"

If empName is set from a web application's form field, the attacker could enter the following in the empName field:

' or '1'='1

When this code is executed, it will return all the data from the table emp, because the '1'='1' in the WHERE clause is always true. The attacker will successfully retrieve data about every employee in the database.

**Preventing SQL injection attacks**

When a bind variable is passed as an argument to an SQL prepared statement, it is automatically escaped by the JDBC driver. The resulting escaped strings treat the variable as user data and cannot be interpreted by the SQL database server as an SQL statement. Therefore, any user-supplied data must be escaped before it is added to an SQL statement. Listing 19 shows the bind variable user ID being added to an SQL statement:

**Listing 19. Prepared statements with bind variables**

String selectStatement = "SELECT \* FROM User WHERE userId = ? ";

PreparedStatement prepStmt = con.prepareStatement(selectStatement);

prepStmt.setString(1, userId);

ResultSet rs = prepStmt.executeQuery();