PreparedStatement

- Innehåller ett SQL-statement som är förkompilerat
- Eftersom denna typ av `Statement` är snabbare än vanliga `Statement` används detta för frågor som ofta exekveras
- Har möjlighet att ta emot parametrar
- Har inbyggt skydd mot SQL injections

```
String query = "SELECT * FROM Employee WHERE employeeNumber = ?";

// Create and prepare statement
try (Connection connection = DriverManager.getConnection(...);
     PreparedStatement statement = connection.prepareStatement(query)) {

     // Set parameters
     statement.setString(1, "1001");

     ResultSet result = statement.executeQuery();
     ...
}
```

PreparedStatement och INSERT

- Har möjlighet att ta emot parametrar

```
// Insert SQL-String with parameter placeholders
String insert = "INSERT INTO Employee (firstName, lastName, employeeNumber)
                 VALUES (?,?,?)"

// Prepare statement
try (Connection connection = DriverManager.getConnection(...);
     PreparedStatement statement = connection.prepareStatement(insert)) {

     // Set parameters
    statement.setString(1, "Watson");
    statement.setString(2, "Jones");
    statement.setString(3, "1001");
    statement.executeUpdate();
}
```

Transaktioner

- Transaktioner ger dig möjlighet att bestämma om ändringar i databasen skall appliceras eller inte
- Ser flera SQL-statments som *en* logisk enhet – om ett statment misslyckas gör hela transaktionen det
- Transaktioner i dess enklaste form innefattar stegen:

  1. Sätt **`autoCommit(false)`**

  2. Exekvera de statments som skall ingå i transaktionen

  3. Om något statment misslyckades anropas **`rollBack()`** – annars anropas **`commit()`**

- En transaktion kan köras i olika *Transaction Isolation Levels* - använd *default* tills du behöver ändra (Det finns fem olika levels och sätts på Connection-objektet)
- *Tänk på att arbeta med transaktioner kan innebära att vissa delar av en tabell (ibland en hel tabell) blir låst under tiden transaktionen körs. Detta för att ingen annan skall kunna ändra i berörd data eller se data som ännu inte är ”commitad”. Detta kan leda till att andra får vänta vilket inte alltid är att önska.*

Transaktioner forts.

- Exempel på en transaktion:

```
try (Connection connection = DriverManager.getConnection(...)) {

    // Turn of auto commit
    connection.setAutoCommit(false);

    // Create and prepare statement
    try (Statement statement = connection.createStatement()) {

      // Insert two valid employees
      statement.executeUpdate("INSERT INTO Employee (firstName, lastName, employeeNumber)
                               VALUES ('Luke','Skywalker','1001')");
      statement.executeUpdate("INSERT INTO Employee (firstName, lastName, employeeNumber)
                               VALUES ('Leia','Skywalker','1002')");

      // Insert employee with missing employeeNumber
      statement.executeUpdate("INSERT INTO Employee (firstName, lastName) VALUES
                              ('Darth','Vader')");

      // Try to commit ALL statements
      connection.commit();
  }
  catch (SQLException e) {
      connection.rollback();
      throw e;
  }
}
```

- Notera: Det går att använda **PreparedStatement** istället för **Statement**

Batch Processing

- Används för att skicka flera SQL-satser på en gång till databasen
- De SQL-satser som skickas in får inte returnera något `ResultSet` (används alltså inte till frågor)
- Slå alltid av **`autoCommit`** när du använder dig av batch processing om det är viktig att alla eller inga av SQL-satserna körs

```
try (Connection connection = DriverManager.getConnection(...)) {

   // Turn of auto commit
   connection.setAutoCommit(false);

   // Create and prepare statement
   try (Statement statement = connection.createStatement()) {

         // Add employees to batch
         statement.addBatch("INSERT INTO Employee (firstName, lastName, employeeNumber)
                             VALUES ('Luke','Skywalker','1001')");
         statement.addBatch("INSERT INTO Employee (firstName, lastName, employeeNumber)
                             VALUES ('Leia','Skywalker','1002')");
         statement.addBatch("INSERT INTO Employee (firstName, lastName, employeeNumber)
                             VALUES ('Obi-Wan','Kenobi','1003')");

         // Try to execute batch
         statement.executeBatch();

         // Commit transaction
         connection.commit();
   }
   catch (SQLException e) {
         connection.rollback();
         throw e;
   }
}
```

Hämta genererade nycklar

- Om Drivern stöder det går det att få tag på de ev. genererade primary keys som skapades vid INSERT
- För att få reda på detta:

```
boolean hasSupport = connection.getMetaData().supportsGetGeneratedKeys();
```

- För att få den genererade nyckeln görs följande:

```
 // Return generated keys
statement.executeUpdate("INSERT INTO Employee (firstName, lastName, employeeNumber)
                         VALUES ('Luke','Skywalker','1001')",
                         Statement.RETURN_GENERATED_KEYS);

// Retrieve result set with key(s)
ResultSet keys = statement.getGeneratedKeys();

if(keys.next()){
   // Get key
   long key = keys.getLong(1);
}
```