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

```
String query = "SELECT email FROM users WHERE username = ?";

PreparedStatement pstmt = con.prepareStatement(query);
pstmt.setString(1, "anca01");

ResultSet rs = pstmt.executeQuery();
```

- Observera att vissa databaser kompilerar om dessa statements efter varje commit (tas upp längre fram) så i dessa fall är detta att använda PreparedStatements prestandasänkande

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:

```
// Skapar ett Statement
Statement stmt = con.createStatement();

// Slår av autocommit
con.setAutoCommit(false);

try
{
stmt.executeUpdate("INSERT INTO `users` VALUES (null, 'Anders', 'Carlsson', 'anca01', 'secret')");
        stmt.executeUpdate("INSERT INTO `users` VALUES (null, 'Elin', 'Carlsson', 'elca01','kl4ddk4k4')");

 // Detta kommer inte att gå då det saknas ett värde för primärnyckeln
stmt.executeUpdate("INSERT INTO `users` VALUES ('Felicia', 'Carlsson', 'feca01', 'weeeeee')");

 // Gör alla INSERT beständiga (kommer i detta fall inte att exekveras)
con.commit();
}
catch (SQLException e)
{
// "Rullar tillbaka" de två första INSERT eftersom det tredje misslyckades
 con.rollback();
}
…Stäng resurser som vanligt
```

- **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

```
// Slår av autoCommit
con.setAutoCommit(false);
try
{
// Lägger till INSERT-satser till Statments batch
stmt.addBatch("INSERT INTO `users` VALUES (null, 'Anders', 'Carlsson', 'anca01', 'secret')");
stmt.addBatch("INSERT INTO `users` VALUES (null, 'Elin', 'Carlsson', 'elca01', 'kl4ddk4k4')");
stmt.addBatch("INSERT INTO `users` VALUES (null, 'Felicia', 'Carlsson', 'feca01', 'weeeeee')");

// Exekverar alla INSERT-satser på en gång
stmt.executeBatch();

 // Om allt gick bra görs en commit
 con.commit();
}
catch (SQLException e)
{
System.err.println("Batch processing gick inte igenom. Ingen commit() gjordes.”);
 con.rollback();
}

… Stäng resurser som vanligt
```

Övning

- Lägg till `PreparedStatments` till övningen som delades ut i onsdags
- Använd en transaktion för att lägga in en `User` och dennes `Address`. Om något gick fel skall en rollback göras.
- Skapa en metod som lägger till många `Users` på en gång m.h.a. batch processing.