Scrollable ResultSet

- Ett `ResultSet` har default bara möjlighet att stegas igenom från första till sista raden en gång
- `ScrollableResultSet` gör det möjligt att stega fram och tillbaka i ett `ResultSet` flera gånger
- Är inget nytt interface eller klass utan en egenskap hos det `ResultSet` som returneras från `Statment`

Finns tre olika typer:

`TYPE_FORWARD_ONLY:`
Känner inte av ändringar i databasen vid genomstegning. Kan bara stega framåt.

`TYPE_SCROLL_INSENSITIVE:`
Känner inte av ändringar i databasen vid genomstegning. Kan stega både framåt och bakåt.

`TYPE_SCROLL_SENSITIVE:`
Känner av ändringar i databasen vid genomstegning. Kan stega både framåt och bakåt.

Notera:
Drivern till MySQL 5 stöder inte `TYPE_SCROLL_SENSITIVE`

Scrollable ResultSet forts.

- **TYPE_SCROLL_SENSITIVE** och **TYPE_SCROLL_INSENSITIV** kan använda följande metoder:

  next() – flyttar cursor till nästa rad (som vanligt)

  previous() – flyttar cursor ett steg bakåt

  beforeFirst() – flyttar cursor till innan första raden

  afterLast() – flyttar cursor till efter sista raden

  first() – flyttar cursor till första raden

  last() - flyttar cursor till sista raden

  relative(int row) – flyttar cursor till en specifik rad relativt till aktuell rad (negativt värde stegar bakåt)

  absolut(int row) – flyttar cursor till en specifik rad räknat från första raden om värdet är positivt annars räknat från sista raden

Scrollable ResultSet forts.

- Exempel:

```
// Skapar ett Statement som skapar Scrollable ResultSets
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);

// Exekverar en fråga mot tidigare skapat Statement och spara resultatet i ett ResultSet
ResultSet rs = stmt.executeQuery("SELECT * FROM users");

// Flyttar cursor till rad 3 sett från början
rs.absolute(3);

System.out.println("Firstname: " + rs.getString("firstname"));
System.out.println("Lastname: " + rs.getString("lastname"));
System.out.println("Username: " + rs.getString("username"));
System.out.println("Password: " + rs.getString("password"));

…Stäng resurser som vanligt
```

Updateable ResultSet

- ResultSet kan vara **CONCUR_READ_ONLY** och **CONCUR_UPDATEABLE**
- CONCURE_UPDATEABLE – ger möjlighet att uppdatera data i ett ResultSet så att det slår igenom i databasen
- För att ett ResultSet skall gå att uppdatera måste det:

  1. I frågan hämta alla kolumner som utgör primärnyckeln
  2. Vara en fråga som bara hämtar kolumner från en tabell

- De olika **updateXXX() –** metoderna används för att uppdatera aktuell rad
- Databasen uppdateras först när **updateRow()** anropas
- Det går även att ta bort och lägga till rader till ett Updateable ResultSet
- **deleteRow()** tar bort aktuell rad direkt
- **moveToInsertRow()** – flyttar cursor till *insert row* (fungerar som en buffer tills **insertRow()** anropats)

Updateable ResultSet forts.

- Exempel på uppdatering:

```
// Skapar ett Statement som i sin tur kan skapa Updateable ResultSet
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);

// Exekverar en fråga mot tidigare skapat Statement. Notera att primärnycklen hämtas
ResultSet rs = stmt.executeQuery("SELECT id, password FROM users");

// Flyttar cursor till rad 3 sett från början
rs.absolute(3);

// Uppdaterar rad 3 “password” –kolumn
rs.updateString("password", "k0k0smj0lk");

// Gör ändringen bestående i databasen
rs.updateRow();

…Stäng resurser på vanligt vis
```

Updateable ResultSet forts.

- Exempel då ny rad skapas:

```
// Skapar ett Statement som i sin tur kan skapa Updateable ResultSet
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);

// Exekverar en fråga mot tidigare skapat Statement. Notera att primärnycklen hämtas
ResultSet rs = stmt.executeQuery("SELECT * FROM users");

// Flyttar till insert row
rs.moveToInsertRow();

// Lägger till en till user
rs.updateString("firstname", "James");
rs.updateString("lastname", "Bond");
rs.updateString("username", "jabo10");
rs.updateString("password", "007");

// Updaterar databasen
rs.insertRow();

…Stäng resurser på vanligt vis
```

Notera:
Inget värde för primärnyckeln ges. Detta fungerar eftersom kolumnen *id* i tabellen users är satt till AUTO_INCREMENT

Updateable ResultSet forts.

- Exempel då en rad tas bort:

```
// Skapar ett Statement som i sin tur kan skapa Updateable ResultSet
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);

// Exekverar en fråga mot tidigare skapat Statement. Notera att primärnycklen hämtas
ResultSet rs = stmt.executeQuery("SELECT * FROM users");

// Loopar igenom resultatet
while (rs.next()) {
String username = rs.getString("username");
  if("jabo01".equals(username)){
        // Tar bort raden. Obs! Detta slår igenom direkt i databasen
  rs.deleteRow();
  }
}
```

…Stäng resurser på vanligt vis

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:

```
// supportGenKeys kommer att vara true om det stöds
boolean supportGenKeys = con.getMetaData().supportsGetGeneratedKeys();
```

- För att få den genererade nyckeln görs följande:

```
Statement stmt = con.createStatement();
// Säger till att denna executeUpdate skall spara ev. genererad primary key
stmt.executeUpdate("INSERT INTO users values (null, 'anca01','secret‘)",
Statement.RETURN_GENERATED_KEYS);

// Hämtar primary key som genererades
ResultSet rs = stmt.getGeneratedKeys();

// I detta fall är det bara en primary key som returneras
rs.next();

// Hämtar primary key för den rad som skapades
int key = rs.getInt(1);
```

Övning

- Skapa en metod som uppdaterar en anställde med hjälp av ett UpdatableResultSet
- Skapa en metod som tar bort en anställde från ett UpdateableResultSet
- Skapa en metod som, om anställde inte existerar, lägger till den med hjälp av ett UpdatableResultSet
- Skapa en metod som returnerar den genererade nyckel som genereras när en anställde sparas