Innehåll

Användbara Statement metoder
- addbatch(*sql string*)
- executeBatch

Användbara ResultSet metoder som inte nämnts
- moveToInsertRow
- moveToCurrentRow
- updateXX
- updateRow
- deleteRow

Säkerhet
- SQL Injection
- Cross scripts (XSS)
- PreparedStatement vs Statement
- Kryptering

PreparedStatement
- Paramatrisering (1-indexerat från vänster)
- För-kompilerat

CallableStatement (Curly brackets (ODBC standard), i huvudsak en funktion)
- Stored procedure
- inout
- in
- out

Transactions
- autoCommit
- rollback
- savepoints

Hantering av resurser (try-with)
- Try-with-catch
- Finally
- close

Sammanfattning

- addBatch tillåter en att lägga flera SQL kommandon i rad
- moveToInsertRow flyttar pekaren till en färdig buffrad rad som är reserverad åt att lägga in nya rader, moveToCurrentRow flyttar tillbaks pekaren
- updateXX måste följas av updateRow för att ge effekt

Säkerhet
Alla SQL kommandon behandlas som strängar, detta öppnar upp en möjlighet för SQL Injection attack, dvs. att klienten skriver över SQL kommandot.
Cross script är liknande men det är när klienten kör script igenom strängarna, ex. där ett användarnamn ska stå på en hemsida kan klienten skriva in en <script></script> tag och på så vis köra javascript kod utan att ha tillgång till källkoden.

PreparedStatement förhindrar SQL Injections attack genom att parametrisera alla värden, detsamma gäller för CallableStatement.

Det finns många sätt att kryptera data på men principen är ungefär densamma för alla när den krypterade datan ska lagras i en databas.

1. För att lagra: Lössenord ----> Kryptera ----> Skicka krypterat lössenord till databasen med krypterings nyckeln
2. För att låsa upp: Hämta krypterat lössenord och nyckel från databasen ----> Ta in ett nytt lösenord från användaren ----> Kryptera nya lössenord ----> Jämför med det lagrade krypterade lössenordet

En krypterings metod bör varken vara för snabb eller långsam.
För snabb -> Lätt att brute-force:a
För långsam -> Klienten behöver vänta

Se nätverks attaker live: http://map.norsecorp.com/#/

PreparedStatement
Tidigare nämnt, preparedStatement förhindrar SQL injections men gör också:

- För-kompilering, dvs. det tar tid första gången men går fort vid flera körningar
- Parametriserar alla värden, istället för att veta om alla värden när frågan skapas så förs värdena in dynamiskt med setXX metoder.

CallableStatement
Körs på liknande sätt som preparedStatement men används för att kalla på stored procedures
Det finns tre typer av parametrar i en preparedStatement Query, dessa är:

- IN, det värdet du skickar
- INOUT, ett värde du både skickar och förväntar dig tillbaks
- OUT, ett värde du förväntar dig tillbaks

Kolla på https://www.youtube.com/watch?v=_2sJls8rnBU
6.1 - 6.4

Transaktioner
För att kunna försäkra dig om att all data verkligen överförs använder du transaktioner. Det första du måste göra är att kalla på connection.setAutoCommit(false) så att du själv bestämmer när du vill uppdatera databasen.
Om du behöver köra en rollback (återställa data) så återställa allt tills senaste savepointen eller då du öppnade anslutningen.

Resurshantering
När en resurs öppnas så överförs och/eller utförs arbeta i bakgrunden, därför är det viktigt att du alltid stänger ner en resurs med close metoden. Detta görs vanligen i finally blocket

```
try{
        ….
}
catch(Exception e){
        ….
}
finally {
        ...close();
}
```
finally blocket körs **alltid** när programmet lämnar try:en.

I java 7 infördes try-with-catch som kan används till alla resurser som implementerar java.lang.AutoCloseable. try-with-catch anropar close metoden när try-catch blocket är färdigt.