Working with Databases

Relational Databases (case MySQL)

SQL Language

JDBC

This week

- You will learn to understand basic concepts of relational databases
- You will learn SQL-language (to sufficient level)
- You will learn how to use JDBC and Spring Data JDBC to connect to a database from Java Application

- And we will also take a brief look to NoSQL-databases (MongoDB) and other data formats commonly used

Relational Databases

Overview

RDBMS

- On the background there is the server machine that has the database product installed
  - The product may be Oracle, IBM DB2, Microsoft SQL Server, MySQL, PostgreSQL….
  - Products are often called DBMS or RDBMS (Database Management System, Relational Database Management System)
    - Sometimes the term Database Engine is also used to refer to DBMS
- Later on, we start developing server applications. Then the DBMS may be installed on
  - The same machine the application runs (rather rare case)
  - Behind the network on separate server
  - Big installations on server cluster
    - Several servers holding exactly the same applications and configurations to offer better response times and provide fault tolerance
- Developers may have the DBMS installed on their development machine
  - Or they may operate on development server over the network

Table

- Techinally the table is the basic unit of storage under the DBMS
  - Think of Excel spreadsheet having columns and rows, each column also having a title
- Logically the table is also the basic relation
  - Individual items on each row have a relation to each other, together they describe some kind of data entity
  - On person table each row holds data items that belong to single person
- Most RDBMS products group tables to logical databases
  - Single application often operates to a single logical database

SQL Language

- SQL Language is used to operate on RDBMS
  - Standard even though each RDBMS has its own dialect
- Sometimes a distinction is made between
  - DDL – Data Definition language
    - Statements used to create and modify tables
  - DML – Data Manipulation Language
    - Statements used to maintain and query data in the database
- DML gives you CRUD
  - Create
  - Review
  - Update
  - Delete

Transaction

- SQL statements are always run by the database engine within a transaction
  - It may contain single sql statement
  - Or group of sql statements
- During the transaction the engine ensures that no other user accesses the same data
  - Someone writes while you are reading => you get partially written data, this never happens with databases
  - Database engine “locks” the data you access
    - No one can write the data while you read, no one read while you write
    - Sometimes if complex operations are executed this may result in dead-locks, no one can read or write….
- Transactions are ACID
  - Atomicity
  - Consistency
  - Isolation
  - Durability

A few more terms

- OLTP – Online Transaction Processing
  - The typical database –based system where a user manipulates data
    - Enter data, queries some listings etc
  - This is what we will be implementing
- OLAP – Online Analytical Processing
  - Database constructs are designed especially for reporting
  - Often “multidimensional” you can query sales report
    - By sales person
    - By sales area
    - By fiscal quarter
    - By customer group
    - Or any combination of above
- DW – Datawarehouse
  - A huge storage of business data
  - Often a source of data for OLAP system

Designing a database

- You should start with a conceptual model
  - What data entities are manipulated: Person, Company….
    - These will become tables of database
  - Often you will have similar classes in your application and the fields of those classes translate to columns in the tables
- For a good database design there are quite a few guidelines
  - Study “database normalization”
- But let’s stick to few basic ones
  - Each table should have a “primary key”
    - Must be unique on each row
    - Often id of type integer
    - Not part of actual data otherwise belonging to that data
      - Even if we have some kind of userid (perhaps email) for each person it should not be used as primary key

Designing databases, foreign key

- Foreign key is used to describe one-to-many relation
  - Or many-to-one, which ever way you want to look at it
  - Company may have several employees, group of employees belongs to the same company
- The “many end” of relation holds the foreign key
  - An extra field that has a value of primary key that identifies the item at “one end” of relation
  - Employee table would hold company_id that refers to id of the company

Designing databases, many-to-many

- Purchase has several products, single product may be used on several purchases
- A separate “binding table” is needed

Product	
id	**Name**
1	Cheese
2	Milk

Purchase	
id	**Customer**
1	John
2	Mike

Id	Product_id	Purchase_id
1	1	1
2	1	2
3	2	2

Cheese was bought by John and Mike

Mike bought cheese and milk

Some other rules

- Columns should be undividable
  - Name is in most cases a bad idea, use firstname, lastname instead
- Columns should not repeat
  - No extra1, extra2, extra3, use extrainfo-table with foreign key to original table instead
- Naming
  - Choose a naming convention and stick to it
  - Avoid camel casing, not all RDBMS:s support it
    - Use all lower case
  - Table names should be in singular form
    - company, person NOT companies, persons

Datatypes

- Each column in the table has a name and datatype
  - Like variables at program code
- The types available vary slightly between database products. Commonly available are
  - int,float and double
  - varchar(32), where 32 is the length of the field, the actual length may be up to 32 characters
  - char(6), where 6 is again the length, the string always has 6 characters
  - date and datetime
- For mysql you may study
  - https://dev.mysql.com/doc/refman/8.0/en/data-types.html

Null

- The value in a column may be null
  - When creating the table you can describe if the null value is allowed for specific column
- The behaviour of null varies slightly between RDBM:s
- Null testing is most often done with “is” instead of “=“ in sql
- There are also database specific function that can be used in queries to replace a null value with empty string for example
  - select ifnull(email,'not available') from person;

SQL Language

Most important SQL statements

- Select – Query data
  - We'll mostly work with queries
- DML, these are rather straightword, we'll just take few examples
  - Insert
  - Update
  - Delete
- DDL
  - Create (database, table….)
  - Alter (database, table….)
  - Drop (database, table….)

SELECT basics

- Select basic syntax

```
SELECT    [column list]
FROM      [table list]
WHERE     [criteria]
GROUP BY   [column list]
ORDER BY   [column list];
```

Optional

- Get all data from customer-table

```
SELECT * FROM customer;
```

- Get name and price of all products

```
SELECT name,price FROM product;
```

- Get name and price for products where there are over 1000 items in stock

```
SELECT name,price
FROM product
WHERE in_stock > 1000
ORDER BY price desc,name;
```

???

Few notes

- Asterisk (*) is a wild card in column list = selects all columns
- You may define alias names for columns
  - Especially important if functions or calculations are used in column list
- Distinct makes sure all the rows returned are unique
- Tables may also have alias names
  - Important later on as we query data from several tables
- SQL keywords are not case sensitive but depending on the database engine the table – and column names may be

```
SELECT distinct p.id,
    concat(p.firstname,' ',p.lastname) as name
    salary*1.1 as increased_salary
    from person p;
```

Where criteria

The search criteria may become quite complex. It may hold

- Logical operators
- Boolean operators
- NULL –testing
- BETWEEN...AND –operators
- Complex text-searches
- IN –operator

```
=,
<>, !=
<, >, <=, >=
[NOT] LIKE
[NOT] EXISTS
[NOT] IN
[NOT] BETWEEN ... AND
IS [NOT] NULL
```

Equality

- All females

```
SELECT * from person
WHERE gender='F';
```

- All with specific last name

```
SELECT * from person
WHERE lastname='Jones';
```

- Specific lastname but no woment

```
SELECT * from person
WHERE lastname='Jones' AND gender<>'F';
```

- Lastname begins with something, firstname begins with T followed by any character and m

```
SELECT * from person
WHERE lastname like 'Jo%' AND firstname like 'T_m';
```

IS NULL -operator

- Null value operates differently on different DBMS:s
  - Sorting: either first or last
  - Testing equality: always matches or never matches

- But is null / is not null works consistently between DBMS:s

```
SELECT   * from person
WHERE    salary IS NULL;

SELECT   * from person
WHERE    salary IS NOT NULL;
```

BETWEEN...AND –operator

- In most cases you could use a>=b AND c<=d also
- People whose salary is between 5000 and 5500

```
SELECT * FROM person
WHERE salary BETWEEN 5000 AND 5500;
```

- Less than 5000 or greater than 5500

```
SELECT * FROM person
WHERE salary NOT BETWEEN 5000 AND 5500;
```

IN –operator

- Is the value in a list
  - List is often a subquery

```
SELECT * FROM sales
WHERE salesdate IN (
        '01.12.2011', '01.01.2012',
        '01.02.2012', '01.03.2012'
       );
```

- All persons that haven't bought something

```
SELECT * FROM person
WHERE id NOT IN (select personid from sales);
```

Joining data from several tables

- If table list has several tables by default all possible combinations are produced
  - WHERE clause must be used to limit the result
- Most often foreign keys are used to find matching values from other tables
  - But other information may be used also
- To consider
  - Do we want only to get information if the join criteria is satisfied
    - Inner join
  - Do we want to get all the information from one of the tables even if there is no matching data on second table
    - Outer join

Inner join

- Most often used join type
- Only returns rows where there is a binding

- Get all data from person table + the name of the company but only for people for whom the company is specified

```
SELECT p.*,c.name
FROM person p, company c
WHERE p.companyid=c.id
```

left (outer) join

- Get all information from the table specified in the FROM clause
  - Even if the data doesn't exist on the joined table
- You can limit the final result with WHERE-clause after the joins

- Everything from person table, c.name is null if company is not specified for the person

```
SELECT p.*,c.name FROM person p
LEFT JOIN company c ON (c.id=p.companyid)
```

Right (outer) join

- Instead of left join you can also use right join
- Works the other way around
  - Everything from the joined table even if there is no match on the table specified in the FROM-table
- Recommendation is to use left joins when possible
  - They are easier to understand

```
SELECT p.* , c.name FROM company c
RIGHT JOIN person p ON (p.companyid=c.id)
```

Some functions

- Functions are always DBMS-specific
  - All have similar functions but their names and parameter lists may differ
  - Study the documentation of DBMS you end up working with
  - For MySQL https://dev.mysql.com/doc/refman/8.0/en/functions.html

- Few examples
  - String concatenation: concat(firstname,' ',lastname)
  - Replace null value: ifnull(email,'not specified')
  - Date part of current timestamp: date(now())

Group by

- Most commonly group by –clause is used in association with aggregate function
  - Sum, Avg, Count ….
- You must specify all columns not used in aggregate function to group by list of columns

- How many employees company has

```
select c.name,count(p.id) as empl_count
from company c, person p
where p.company_id=c.id
group by c.name;
```

Subqueries and Views

Subquery

- You use a separate query as a part of original query
  - You can use the result of subquery as table (alias name must be specified) in the FROM of JOIN list of tables
  - You can use the result of single column query with WHERE … IN (…) -clause

Products are produced on different locations, our departments also work on different locations. Couple of ways to use this correlation

```
SELECT d.name as dname,p.name as pname
FROM product p, (SELECT name,location FROM department) d
WHERE d.location=p.location;
```

```
SELECT * FROM product
WHERE location in (SELECT location FROM department)
```

Views

- Views are predefined queries that may be used as tables on other queries
  - They don’t store data
  - They just help build queries that would otherwise contain several subqueries

- View is created with CREATE VIEW name_of_view AS SELECT ….. Command
  - After that you can use the view in other queries as it was a table
  - Some DBMS:s allow you to modify data through view as long as it uniquely identifies the underlying table

Data validation, constraints

- Table declarations may include validation rules. Invalid updates will not go through.
- Of those rules we just use
  - Primary key
  - Foreign key
  - not null for column (null value is not allowed)
- But there are more ways to validate data
  - Check https://dev.mysql.com/doc/refman/8.0/en/create-table-check-constraints.html

JDBC

Connecting to databases from Java application

JDBC

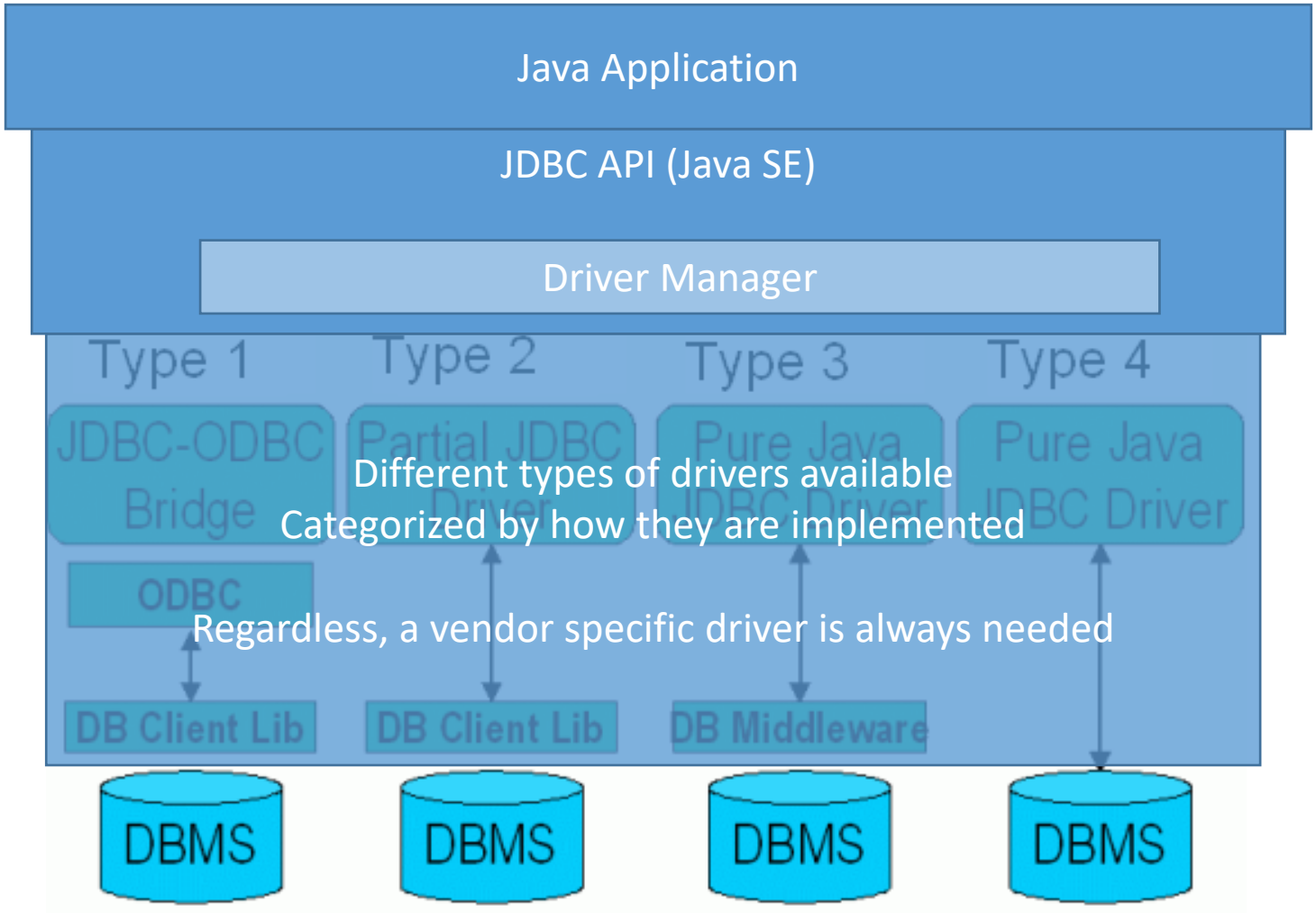
- Java Database Connectivity → JDBC
  - A standard for database-independent connectivity between the Java programming language and a wide range of databases
  - An interface for accessing database management systems (DBMS)

- "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data
  - No vendor-specific database (SQL) queries
  - Based on X/Open SQL CLI (like ODBC).

- Wide range of data sources possible:
  - SQL databases
  - Other tabular data sources, such as spreadsheets or flat files

- The JDBC API makes it possible to do three things:
  - Establish a connection with a database / tabular data source
  - Execute SQL statements
  - Process the results

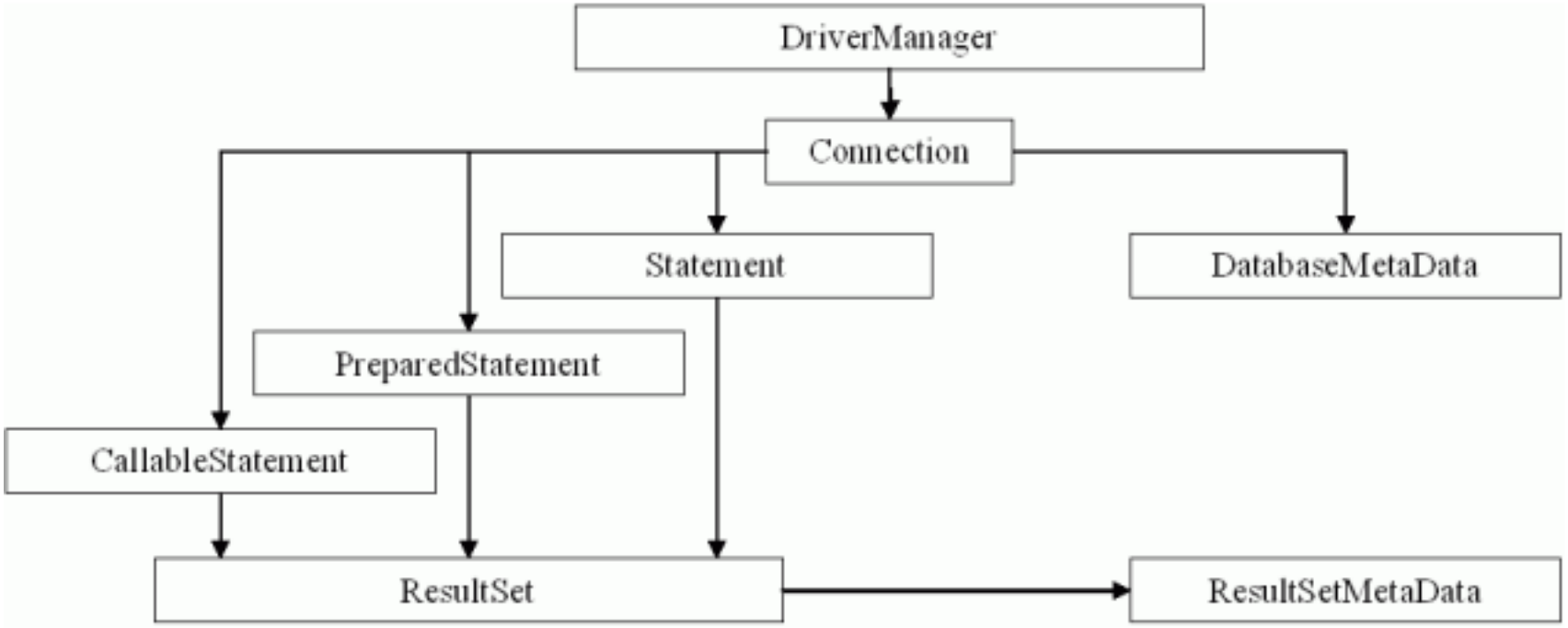
JDBC Architecture



Different types of drivers available
Categorized by how they are implemented

Regardless, a vendor specific driver is always needed

JDBC Object Model



Establishing a Connection

- Two steps: loading the driver and making the connection

1. Making sure the driver is available
   - Not an absolutely necessary step, but may help to pinpoint why the initial connection fails
   - Easy, just one line of code that basically loads the driver into memory:

     ```
     Class.forName("com.mysql.jdbc.Driver");
     ```

   - Diver class is different for each database
     - Driver is usually implemented by the database provider
     - Check from your database documentation which class name to use

Establishing a Connection

2. Making the connection
   - The JDBC `DriverManager` class manages the JDBC drivers that are installed on the system
   - It also provides factory functions for creating connections using the chosen driver

```
Connection conn = DriverManager.getConnection(
        "db url", "loginName", password");
```

   - Note that the database url depends on the driver used
   - For MySQL:
     `jdbc:mysql://localhost:3306/myJdbcDB`

Establishing a Connection

```
Connection conn = null;
try {
     Class.forName("com.mysql.jdbc.Driver");
     String url = "jdbc:mysql://localhost:3306/myJdbcTest";
     conn = DriverManager.getConnection(url ,"myUser","myPw");

     // ...use connection...
     // Remember to close the connection when done using it!
     conn.close();       // Might throw SQLException – omitted here

}catch(ClassNotFoundException cne){
   System.out.println("Couldn't find driver!");
}catch(SQLException se){
  System.out.println("Couldn't establish connection");
}
```

Exercise

- Copy instructor pointed DBCreator.java to a location of your choosing
  - Note the package name in the file
- Compile the code from command line
- Run the code

- You need the driver, copy the instructor pointed jar to your “project” folder
- Run the code
  - java –cp .;.\xxxx.jar jdbc.DBCreator

Creating JDBC Statements

- A `Statement` object is what sends your SQL statemenst to the DBMS.
  - Simply create a `Statement` object using your `Connection` object and then execute it

```
Statement stmt = conn.createStatement();
String sql = "INSERT INTO table1 (one, two) VALUES ('one', 'two')"
stmt.executeUpdate(sql);
```

- For SQL statements that create or modify tables, the method to use is `executeUpdate()`
- For a SELECT statement, the method to execute the query is `executeQuery()`

Creating JDBC Prepared Statements

- The `PreparedStatement` object contains a pre-compiled SQL statement
  - When executed, the DBMS can run the `PreparedStatement`'s SQL statement without having to compile it first
  - Much faster for statements that get executed multiple times than using regular `Statement` objects
  - Easier to avoid SQL injection attacks as inserted values are automatically escaped

```
String sql = "UPDATE table1 set one = ?, two = ?";
PreparedStatement prepStmt = conn.prepareStatement(sql);
prepStmt.setInt(1, 123);                    // First ?-mark
prepStmt.setString(2, "myNewValue2");       // Second ?-mark
prepStmt.executeUpdate();
```

Exercise

- Modify DBCreator slightly
  - Use Visual Studio Code for this exercise

- Create a new company
- Query a company by id

JDBC datatypes

- Data types in SQL and Java are not identical
  - JDBC defines a standard mapping from the JDBC database types to Java types
  - For example, a JDBC `INTEGER` is normally mapped to a Java `int`
  - This enables a simple interface for reading and writing JDBC values as simple Java types

- JDBC defines a set of generic SQL type identifiers in the class `java.sql.Types`
  - These types have been designed to represent the most commonly used SQL types, for example:

`VARCHAR/CHAR/LONGVARCHAR`	`java.lang.String`
`DATE`	`java.sql.Date`
`INTEGER`	`int`
`TIME`	`java.sql.Time`

JDBC ResultSet

- The `executeQuery()` method returns a `ResultSet` object
  - Both for `Statement` and `PreparedStatement`
- Provides access to the result of the query – sort of like a table
  - Table rows are retrieved in sequence
  - A `ResultSet` maintains a cursor pointing to its current row of data
- The `next()` method is used to iterate through the rows of the tabular results
  - Cursor is initially positioned just above the first row of a `ResultSet` object
  - The first call to `next()` moves the cursor to the first row and makes it the current row
  - Call `next()` again to move to the next row

JDBC ResultSet Example

```
String sql = "SELECT * FROM customer";
Statement stmt = conn.createStatement();

// Execute query to get the customers in a ResultSet
ResultSet rs = stmt.executeQuery(sql);
String id, name, address;
// Iterate over the ResultSet using next()
while(rs.next()){
    // Use the appropriate getter functions to access columns
    id = rs.getString("id");
    name = rs.getString("name");
    address = rs.getString("address");
    System.out.println(id + " " + name + " " + address);
}

stmt.close();    // Closes also the ResultSet object
```

Updates

- You can always pass SQL INSERT,UPDATE and DELETE statements to stm.executeUpdate
- But, especially for updates you can also update the data in the result set

```
public void update(Person p){
    String q="SELECT * FROM Person where ID=?";
    try{
        Connection con=getConnection();
        PreparedStatement stm=con.prepareStatement(q,
                ResultSet.TYPE_FORWARD_ONLY,
                ResultSet.CONCUR_UPDATABLE);
        stm.setInt(1, p.getId());
        ResultSet rs=stm.executeQuery();
        if (rs.next()){
            rs.updateInt("company_id", p.getCompanyId());
            rs.updateString("name",p.getName());
            rs.updateString("email", p.getEmail());
            rs.updateDate("contact_date", p.getContactDate());
            rs.updateRow();
        }
        rs.close();
        stm.close();
        con.close();
    }
    catch(Exception ex){
        ex.printStackTrace();
    }
}
```

JDBC transactions

- When a connection is created, it is in auto-commit mode
  - Each individual SQL statement is treated as a transaction and will be automatically committed right after it is executed

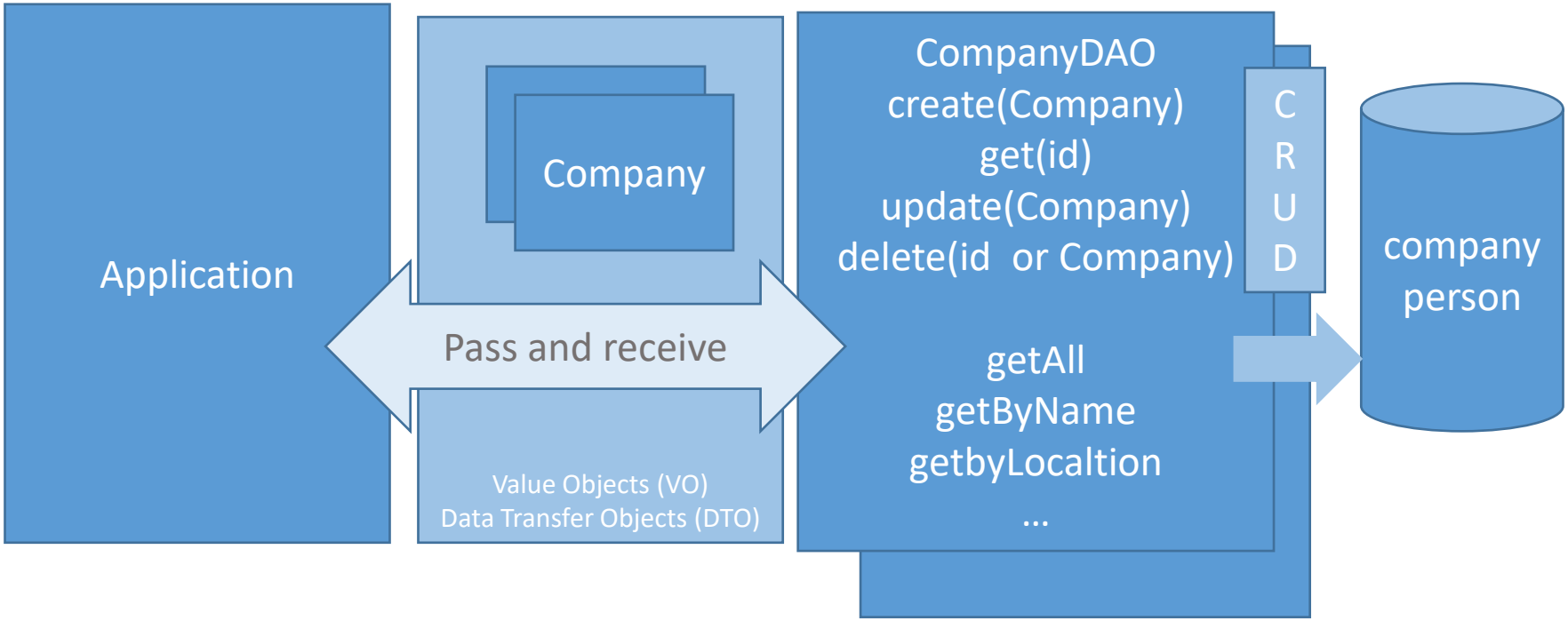
- Way to allow two or more statements to be grouped into a transaction is to disable auto-commit mode

```
Connection conn = ...;
conn.setAutoCommit(false);
```

- Once auto-commit mode is disabled, no SQL statements will be committed until you call `commit()` explicitly
  - All statements executed after the previous call to `commit()` will be included in the current transaction and will be committed together as a unit

DAO-pattern

- Database has tables (company,person)
- We work with object of Person class and Company class
- Let's implement a class against each table that translates table data into objects



Exercise

- Create a new book application “BookJdbc”
- Add a runtime dependency to MySQL-connector
  - Right click dependencies on Project-view
  - Check the pom.xml after adding the dependency
- Start by creating Author-class
  - Same fields as you have at author table
- Then create AuthorDAO-class with
  - Connection getConnection();
  - Author authorFromResultset(ResultSet rs)
  - Author get(int id)
    - Uses the first two to return author that has been fetched from the database
- Test in the main

Exercise

- To the AuthorDAO implement
  - List<Author> getAll()
  - Author create(Author a)
    - The id must change into the id of newly created author
  - Author update(Author a)
    - Return author as it was saved to the database
  - boolean delete(int id)

- To the mainclass implement showAllAuthors
  - That calls AuthorDAO.getAll and then displays all authors
  - This method just helps your testing

Exercise

- Create BookDAO
  - It should hold the CRUD methods as AuthorDAO did but it is not necessary to implement all those for this exercise
    - If you have time, you may do that, otherwise just add a comment that they should be there

- getBooksOfAuthor(int authorid)
  - Does this belong to BookDAO or AuthorDAO?
- getAuthorOfBook(int book)
  - Does this belong to BookDAO or AuthorDAO
- getBooksWithAuthor()
  - Returns some information about book and at least the name of the author

Exercise

- To your Book-class add Author author –field
- To you Author-class add ArrayList<Book> books –field

- To the BookDAO add void getAuthor(Book book)
  - That reads the author of the book and places it to the author-field

- To the AuthorDAO add void getBooks(Author author)
  - That reads the books from the author and places them to books-field of author object

Exercise

- LoanDAO
  - What methods should go in there?