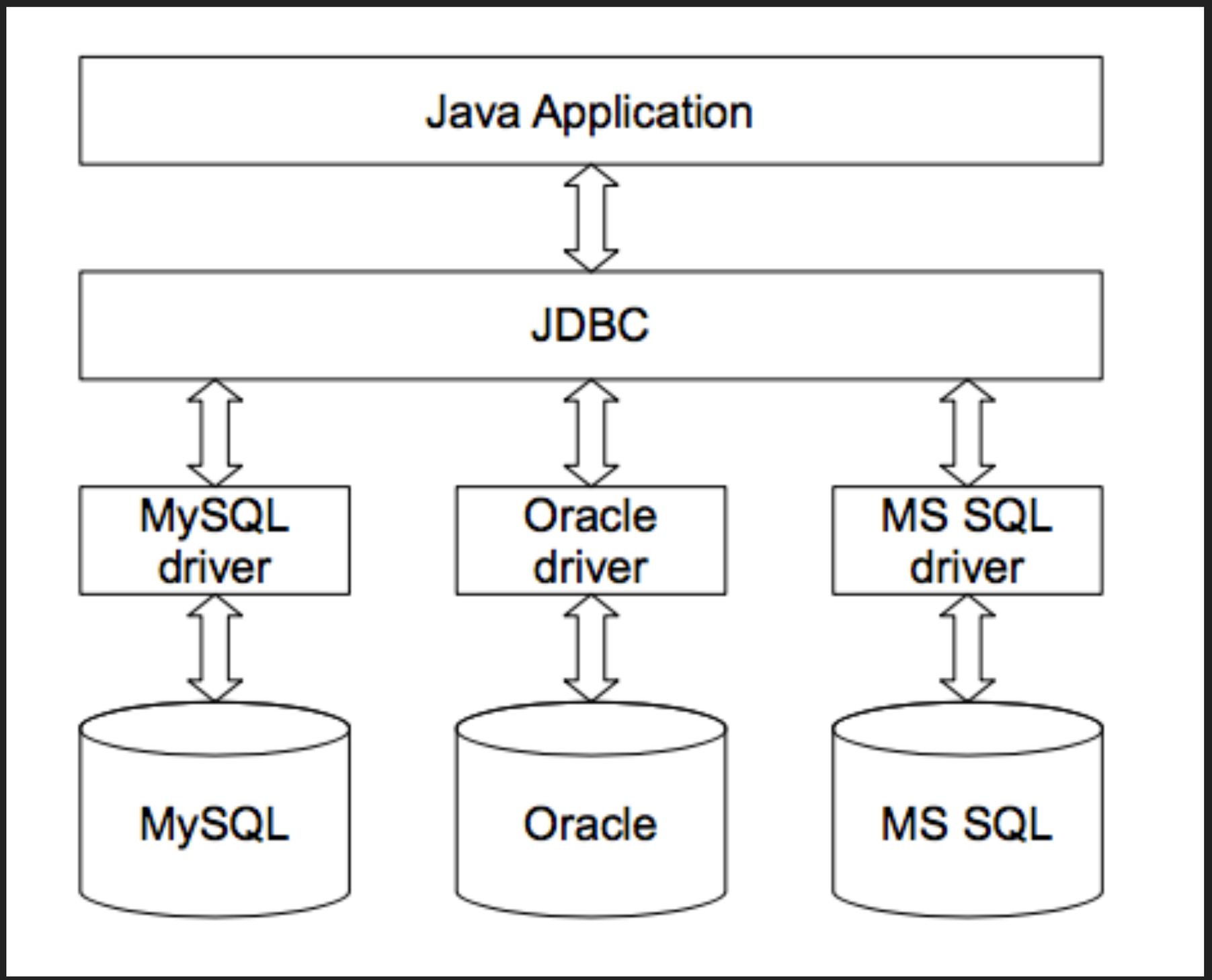
PROGRAMMING ADVANCED

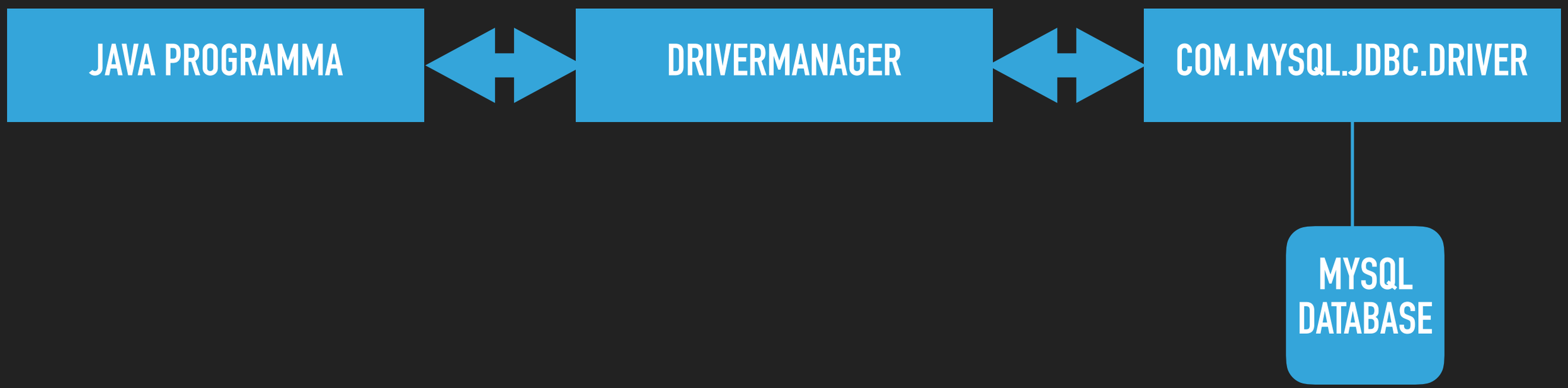
JDBC

WAT IS HET?

- Java DataBase Connectivity
- communicatie met relationele databases
- uniforme interface voor verschillende database producenten
- vereist installatie van een driver(stuurprogramma)


Java Application
JDBC
MySQL driver
Oracle driver
MS SQL driver
MySQL
Oracle
MS SQL

CONNECTEREN



CONNECTEREN

- Zorg dat de driver zich op het classpath bevindt
- Maak een connectie
  - DriverManager.getConnection(url, login, password)
    - url -> jdbc:subprotocol:subname
      - bvb. jdbc:mysql://noelvaes.eu/StudentDB

CONNECTIE

- class.forName(“com.mysql.jdbc.Driver”)
- try-with-resource

```
package jdbc;

import java.sql.*;
public class ConnectDB {
   public static void main(String[] args) {
      try (Connection con = DriverManager.getConnection(
            "jdbc:mysql://noelvaes.eu/StudentDB","student",
            "student123")) {
         System.out.println("Connection OK");
      }
      catch (Exception ex) {
         System.out.println("Oops, something went wrong!");
         ex.printStackTrace(System.err);
      }
   }
}
```

STATEMENTS

- Na het maken van een connectie kunnen we communiceren met de database
- Dit doen we met behulp van SQL commando's
- Via JDBC kan je met behulp van een Statement communiceren met de database
  - bvb. Statement s = connection.createStatement()

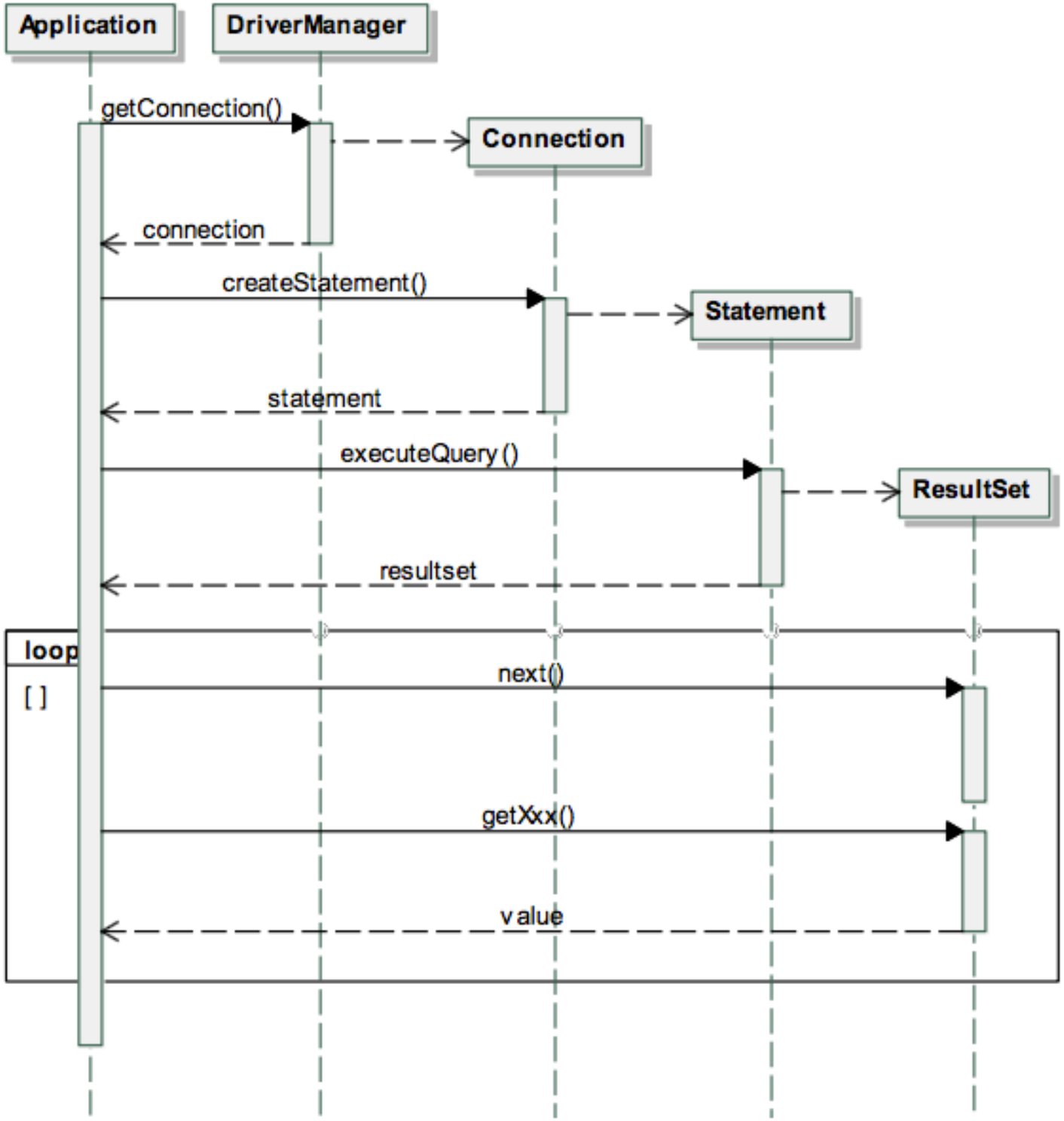
STATEMENTS

- executeUpdate(String sql)
  - Create, Insert, Update, Delete
    - geeft integer(aantal affected rows) als resultaat
- executeQuery(String sql)
  - geeft tabel met resultaten terug
- execute()
  - voert eender welk sql commando uit

RESULTSET

- executeQuery(String sql) geeft een resultSet terug
  - dit kan je vergelijken met een array
- Typisch iterate je over alle elementen in de resultSet
  - while(resultSet.next())
- en haal je de waarde per kolom op (zie cursus p.59 voor alle mogelijkheden)
  - resultSet.getString(1) -> haalt een String waarde op uit kolom 1
  - resultSet.getString(“password”) -> haalt een String waarde op uit kolom paswoord

RESULTSET



Opdracht 2: Gegevens opvragen

PREPARED STATEMENTS

- queries worden telkens gecompileerd (=traag)
- prepared statements worden op voorhand gecompileerd en kunnen meermaals herbruikt worden (=sneller)
- het beschermt ons tegen sql-injection
  - zie http://stackoverflow.com/questions/4333015/does-the-preparedstatement-avoid-sql-injection

PREPARED STATEMENTS ZONDER PARAMETERS

- PreparedStatement ps = connection.prepareStatement(sql)
- ps.executeQuery()

PREPARED STATEMENTS MET PARAMETERS

- one-based ipv zero-based!

```
String sql = "UPDATE Beers SET Price = ? WHERE Name = ?";
try (Connection con = DriverManager.getConnection(
     "jdbc:mysql://noelvaes.eu/StudentDB","student","student123");
     PreparedStatement stmt = con.prepareStatement(sql)) {
   stmt.setFloat(1,3.5F);
   stmt.setString(2,"Zulte");
   int result = stmt.executeUpdate();
}
```

GEVAREN – SQL INJECTION

- http://www.veracode.com/security/sql-injection
- http://www.w3schools.com/sql/sql_injection.asp
- PENDING uitleggen input sanitization

TRANSACTIES

- Een aantal commando’s als 1 blok uitvoeren ipv 1 per 1
  - ofwel lukt alles, ofwel wordt niéts uitgevoerd
  - bvb. overschrijven van geld tussen 2 rekeningen
    - geld gaat af van rekening A
    - geld komt bij op rekening B

TRANSACTIES

```
try (Connection con = DriverManager.getConnection("url",
                                                   "login","password")) {
   try (Statement stmt = con.createStatement()) {
      con.setAutoCommit(false);
      stmt.executeUpdate(SQL1);    // Transaction start
      stmt.executeUpdate(SQL2);
      con.commit();                // Transaction commit
   } catch (Exception e) {
      con.rollback();              // Transaction rollback
   }
}
```

SAVEPOINTS

- Tijdens een transactie kan men een savePoint instellen. Dit maakt het mogelijk om de transactie terug te draaien tot op dat punt

Methode	***Omschrijving***
`setAutoCommit(false)`	Schakelt transacties in.
`executeUpdate()` `executeQuery()`	SQL-commando's worden opgestapeld.
`setSavePoint()`	Stelt een *savepoint* in.
`rollback()`	Opgestapelde commando's worden geannuleerd.
`commit()`	Opgestapelde commando's worden als geheel definitief gemaakt.
`setAutoCommit(true)`	Schakelt transacties terug uit.

TRANSACTIES – READS

- Tijdens een transactie worden gegevens uit de database vergrendeld zodat andere gebruikers(andere trx) er tijdelijk geen wijzigingen in kunnen aanbrengen. Pas na het afsluiten van de transactie door commit of rollback wordt de vergrendeling opgeheven
  - dirty read: leest gegevens van transaction die nog niet gemocht is
  - non-repeatable read: transactie leest gegevens opnieuw die intussen gewijzigd zijn door andere net gecommitte transactie
  - phantom read: query uitgevoerd op intussen veranderde data

TRANSACTION ISOLATION LEVEL

- Bepaald door transaction isolatie level

Level	*Dirty read*	*Non-repeatable read*	*Phantom read*
TRANSACTION_NONE	Geen transacties		
TRANSACTION_READ_UNCOMMITTED	X	X	X
TRANSACTION_READ_COMMITTED	O	X	X
TRANSACTION_REPEATABLE_READ	O	O	X
TRANSACTION_SERIALIZABLE	O	O	O

(EXTRA) TRANSACTIONS – OPTIMISTIC VS PESSIMISTIC LOCKING

- https://en.wikipedia.org/wiki/Lock_(database)
- https://en.wikipedia.org/wiki/Isolation_(database_systems)

DAO'S

- PENDING