

JDBC

Veerle Ongenae



Overzicht

- **Wat is JDBC?**
- **Basiswerking**
 - Statements
 - PreparedStatements
 - CallableStatements
 - Transacties
- **JDBC uitgebreid**
 - DataSource
 - Scrollable en aanpasbare ResultSet
 - RowSets
 - SQL3-gegevenstypes

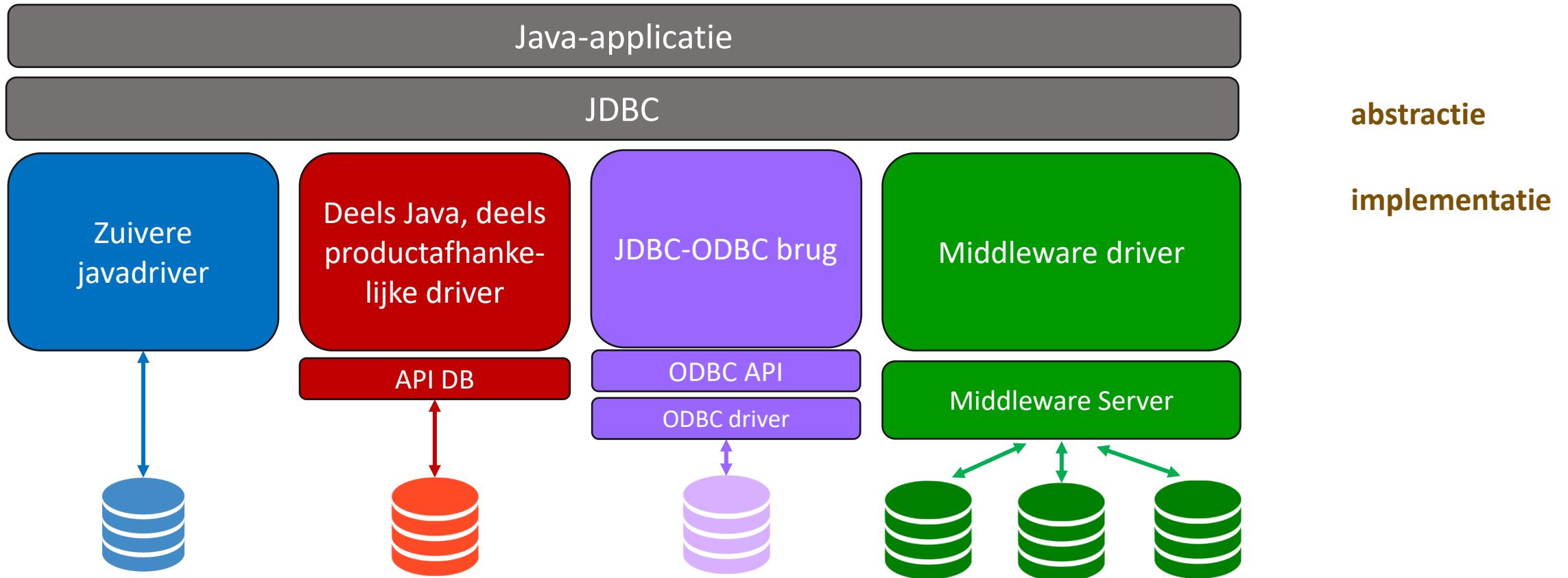


Java Database Connectivity (JDBC) API

- Toegang tot relationele data vanuit Java
- SQL-compatibele relationele gegevensbanken
- Een API om SQL-statements uit te voeren en de verkregen resultaten te verwerken
- Abstractie van productafhankelijke details
 - Veralgeming meest voorkomende gegevensbankfuncties
 - Applicatie bruikbaar met verschillende gegevensbanken



JDBC drivers - types



JDBC

- JDBC Core API: `java.sql`

- Gegevensbankverbinding via JDBC-drivers
- Basis SQL-operaties uitvoeren
- Bij alle versies van Java

- JDBC Optional Package API: `javax.sql`

- Voor application servers
 - Datasource ipv driver
- Beheer en pooling van JDBC-connecties, gedistribueerde transacties, rowsets,
 - ...
- Standaard bij JSR 223 vanaf versie 1.4



Overzicht

- Wat is JDBC?
- Basiswerking
 - Statements
 - PreparedStatements
 - CallableStatements
 - Transacties
- JDBC uitgebreid
 - DataSource
 - Scrollable en aanpasbare ResultSet
 - RowSets
 - SQL3-gegevenstypes



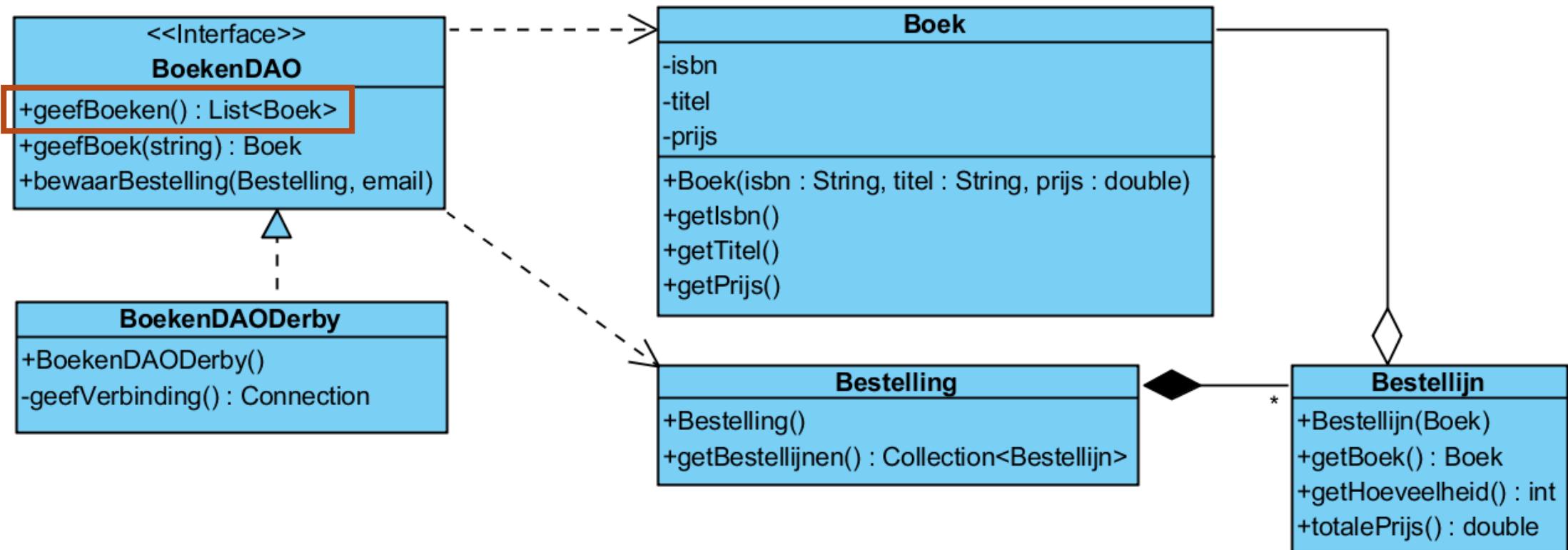
Voorbeeld

Tabel BOEKEN

ISBN	TITEL	PRIJS
isbn19789043025256	jQuery - de basis	12
isbn2	aan de slag met HTML5	15
isbn3	Head First Design Patterns	45
isbn1	Thinking in Java	58
isbn4	Java Servlet Programming	36
isbn5	Java Server Programming	81
isbn6	The Java Tutorial	65
isbn7	Java Swing	55



Voorbeeld



Verschillende stappen

- Configuratiegegevens inlezen
- Driver laden (optioneel vanaf JDK 1.6)
- Verbinding aanmaken
- Statement aanmaken
 - “gewoon”
 - PreparedStatement (parameters)
 - CallableStatement (stored procedure)
- Statement uitvoeren → eventueel ResultSet overlopen



Configuratie – driver laden

```
public class BoekenDAO Derby implements BoekenDAO {  
    private final ResourceBundle sqlOpdrachten; → type in java  
    private final ResourceBundle databaseConfig; → property bestand  
    → standaard methode ·  
    public BoekenDAO Derby() throws ClassNotFoundException { → klasse zoeken met naam  
        sqlOpdrachten = ResourceBundle.getBundle("be.tiwi.boekenwinkel.implDb.sql"); → sql en property ook  
        databaseConfig = ResourceBundle.getBundle("be.tiwi.boekenwinkel.implDb.database");  
  
        // enkel voor oudere versies van Java (voor JDK 1.6)  
        Class.forName(databaseConfig.getString("DRIVER"));  
    } → niet voor ons (voor oudere versies van Java  
    ...  
}
```



Configuratie

- .properties-bestand

sql.properties

```
# boek opvragen          commentaar
Q_BOEKEN = select * from boeken } sleutel-waarde paren
BOEK_ISBN = isbn
TITEL = titel
PRIJS = prijs
```



Configuratie

- Bestand gebruiken in programma

```
ResourceBundle databaseConfig =  
    ResourceBundle.getBundle("be.tiwi.boekenwinkel.implDb.database");
```

- Naam klasse of naam bestand
- PropertyResourceBundle → verstuft van deze type

locatie van package

```
String klasseDriver = databaseConfig.getString("DRIVER")
```

key
database.properties

```
# info databank  
DRIVER = org.apache.derby.jdbc.ClientDriver
```

...
Variables driver bevat deze value



Driverklasse laden

- Laden in JVM en registreren bij DriverManager
- Niet meer nodig vanaf JDK1.6
- Driverklasse implementeert de interface `java.sql.Driver`

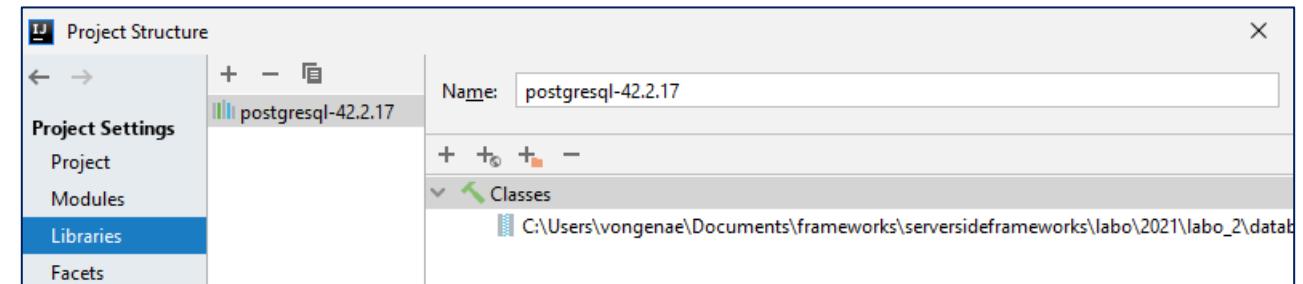
```
try {  
    Class.forName("org.apache.derby.jdbc.ClientDriver");  
} catch (ClassNotFoundException e) {  
    ...  
}
```

als klasse niet gevonden



Driverklasse toevoegen aan CLASSPATH

- De driverklasse behoort niet tot de standaard Java API
- Meegeleverd met de gegevensbank
 - jar-bestand
- Toevoegen aan CLASSPATH
 - Als omgevingsvariabele
 - Als optie bij het java-commando
 - IntelliJ: External Libraries
- CLASSPATH
 - Waar zoeken naar klassen?
 - Standaard: huidige map



Verschillende stappen

- Configuratiegegevens inlezen
- Driver laden (optioneel vanaf JDK 1.6)
- Verbinding aanmaken
- Statement aanmaken
 - “gewoon”
 - PreparedStatement (parameters)
 - CallableStatement (stored procedure)
- Statement uitvoeren → eventueel ResultSet overlopen



Verbinding aanmaken

- DriverManager
 - Toegang tot verschillende beschikbare drivers

```
private Connection geefVerbinding() throws SQLException {  
    return DriverManager.getConnection(  
        databaseConfig.getString("URL"), —> waar de server zich bevindt  
        databaseConfig.getString("LOGIN"), —> gebruiker username  
        databaseConfig.getString("PASWOORD")); —> wachtwoord  
}
```

database.properties

```
URL = jdbc:derby://localhost:1527/boekenwinkel  
LOGIN = iii  
PASWOORD = iiipwd
```



JDBC URL

- Vorm: `jdbc:<subprotocol>://<subname>`
- Subprotocol: identificeert gegevensbankdriver
 - MySql: mysql
 - Derby: derby
 - PostgreSQL: postgresql
- Subname: Afhankelijk van driver
 - MySql, Derby, PostgreSQL:
 - host:poort/database



Verschillende stappen

- Configuratiegegevens inlezen
- Driver laden (optioneel vanaf JDK 1.6)
- Verbinding aanmaken
- Statement aanmaken
 - “gewoon”
 - PreparedStatement (parameters)
 - CallableStatement (stored procedure)
- Statement uitvoeren → eventueel ResultSet overlopen



Opdracht aanmaken

uit DAO interface

```
public List<Boek> geefBoeken() throws BoekenNietBeschikbaarException {  
    ArrayList<Boek> boeken = new ArrayList<>();  
  
    try {  
  
        try (Connection conn = geefVerbinding();) → try with resources  
            Statement stmt = conn.createStatement(); → : als je externe bron  
            ... // opdracht uitvoeren → open, wordt na het  
        } → uitvoeren van opdr-  
    } catch (SQLException e) { → acht automatisch  
        throw new BoekenNietBeschikbaarException(e); → genoten -  
    }  
    return boeken;  
}
```



Voorbeeld

- Voor spel uitvoer?

```
try (Connection conn = ...;  
     Statement stmt = conn.createStatement()) {  
  
    System.out.println(conn.getClass());  
    System.out.println(stmt.getClass());  
  
} catch (SQLException e) {  
    Logger.getLogger(KlassenDB.class.getName())  
        .log(Level.SEVERE, null,  
             "probleem met ophalen info uit databank: " + e.getMessage());  
}
```

```
class org.postgresql.jdbc.PgConnection  
class org.postgresql.jdbc.PgStatement
```



Select-opdracht uitvoeren

```
try (Connection conn = geefVerbinding();
      Statement stmt = conn.createStatement()) {
    // opdracht uitvoeren
    ResultSet rs = stmt.executeQuery(sqlOpdrachten.getString("Q_BOEKEN"));
    while (rs.next()) {
        boeken.add(new Boek(
            rs.getString(sqlOpdrachten.getString("BOEK_ISBN")),
            rs.getString(sqlOpdrachten.getString("TITEL")),
            rs.getDouble(sqlOpdrachten.getString("PRIJS"))));
    }
}
```

een groot tabel

// opdracht uitvoeren

ResultSet rs = stmt.executeQuery(sqlOpdrachten.getString("Q_BOEKEN"));

volgende rij in tabel

rs.getString(sqlOpdrachten.getString("BOEK_ISBN")),

rs.getString(sqlOpdrachten.getString("TITEL")),

rs.getDouble(sqlOpdrachten.getString("PRIJS")));

naam van de columns



Resultaten ophalen

- Interface ResultSet

- Resultaat één keer van boven naar beneden overlopen
- Kolommen tellen vanaf 1 !!!!
- Resultaten worden mogelijks niet allemaal tegelijk opgehaald
- Conversie SQL-type naar Java-type

```
public boolean next() throws SQLException;  
public xxx getXxx(int columnIndex) throws SQLException  
public xxx getXxx(String columnName) throws SQLException
```

omgezet naar juiste JAVA type



JDBC Types

- Java ↔ SQL
 - String ↔ CHAR, VARCHAR
 - double ↔ DOUBLE
 - int ↔ INTEGER
 - float ↔ REAL
 - ...



Andere opdrachten uitvoeren

- delete, update, insert, DDL (Data Definition Language)

```
try (Connection conn = geefVerbinding();
     Statement stmt = conn.createStatement()) {

    // opdracht uitvoeren
    String opdr = "create table temp (nr numeric)";
    stmt.executeUpdate(opdr);

    String insertSql = "insert into temp values (1)";
    String updateSql = "update temp set nr=2 where nr=1";
    stmt.executeUpdate(insertSql);
    int updateCnt = stmt.executeUpdate(updateSql);

}
```



Overzicht Statement

```
try {
    Connection connection = DriverManager.getConnection(...,...,... );
    Statement stmt = conn.createStatement()
} {

    // delete-, insert-, update-opdracht of DDL
    int aantalRijenAangepast = stmt.executeUpdate(...);

    // select-opdracht
    ResultSet rs = stmt.executeQuery(...);
    while (rs.next()) {
        ... = rs.getXXX(kolomNaam);
        ... = rs.getXXX(kolomNummer);
    }
}
```



Verschillende stappen

- Configuratiegegevens inlezen
- Driver laden (optioneel vanaf JDK 1.6)
- Verbinding aanmaken
- Statement aanmaken
 - “gewoon”
 - PreparedStatement (parameters)
 - CallableStatement (stored procedure)
- Statement uitvoeren → eventueel ResultSet overlopen



Overzicht

- Wat is JDBC?
- Basiswerking
 - Statements
 - **PreparedStatement**s
 - CallableStatements
 - Transacties
- JDBC uitgebreid
 - DataSource
 - Scrollable en aanpasbare ResultSet
 - RowSets
 - SQL3-gegevenstypes



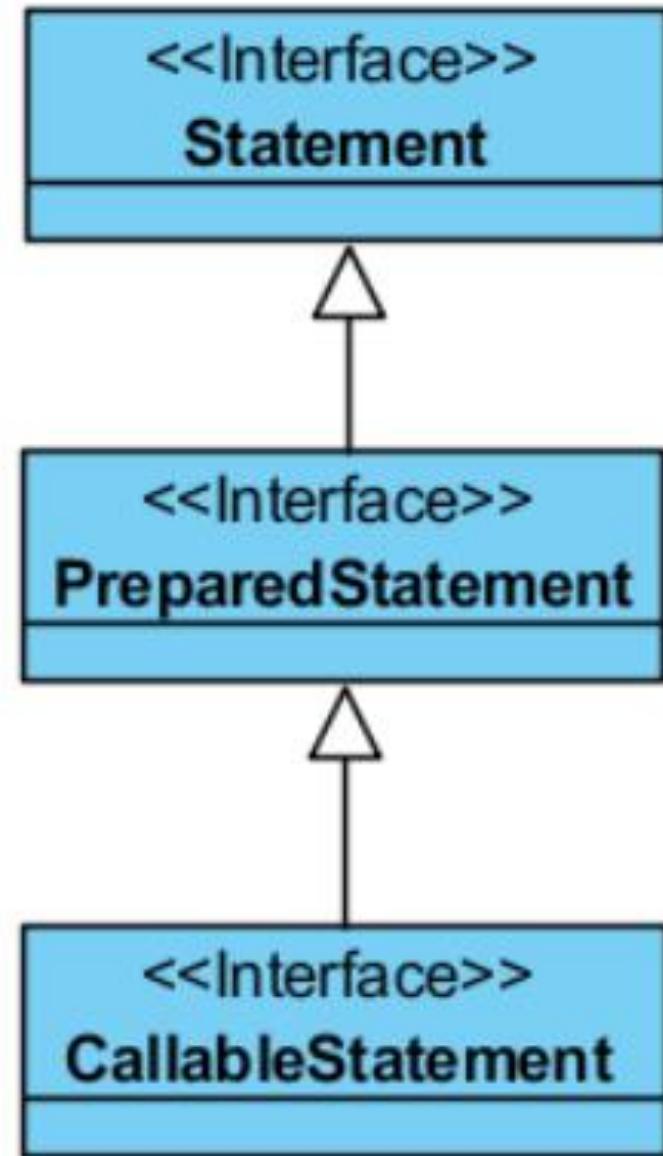
Prepared Statements

- SQL-opdrachten met parameters
- Voorgecompileerd in gegevensbank
- Opdrachten die verschillende keren uitgevoerd moeten worden
- Reductie uitvoertijd
- Veiligheid
 - Gegevensconversie
 - Voorkomen SQL-injectie

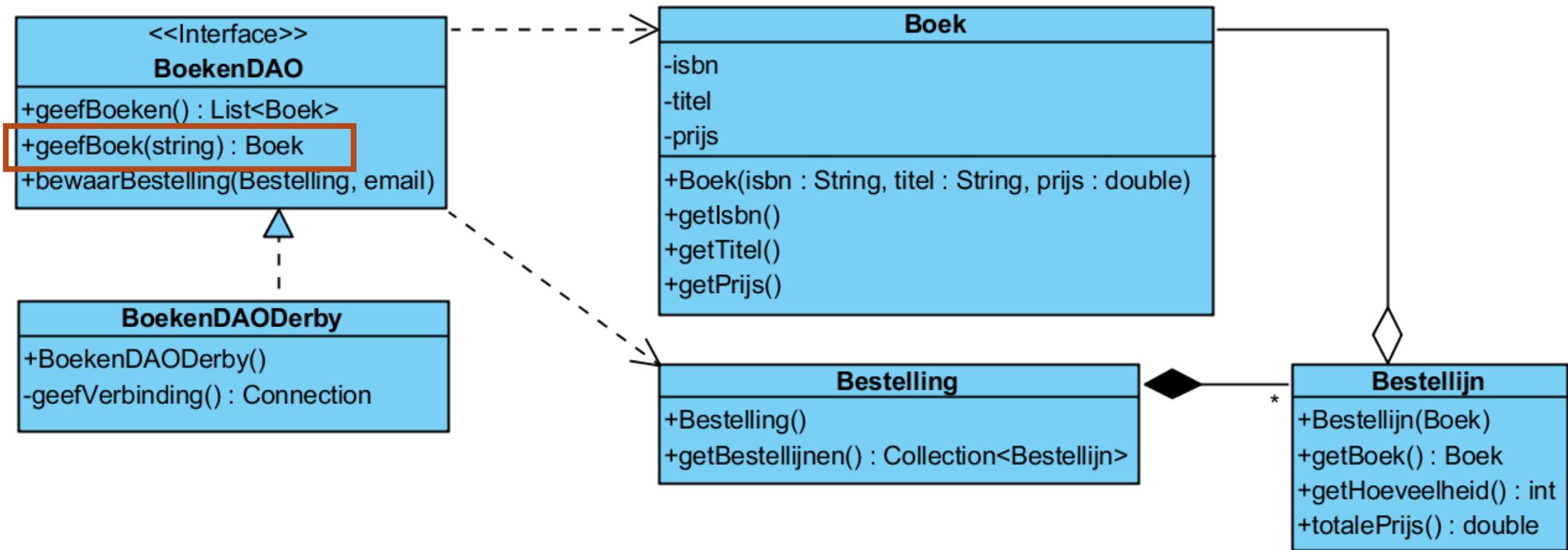


Prepared Statements

- Werkwijze
 - SQL-opdracht: parameters = ?
 - PreparedStatement aanmaken
 - Parameters invullen
 - Uitvoeren, eventueel resultaat ophalen



Voorbeeld



Voorbeeld

Tabel BOEKEN

ISBN	TITEL	PRIJS
isbn19789043025256	jQuery - de basis	12
isbn2	aan de slag met HTML5	15
isbn3	Head First Design Patterns	45
isbn1	Thinking in Java	58
isbn4	Java Servlet Programming	36
isbn5	Java Server Programming	81
isbn6	The Java Tutorial	65
isbn7	Java Swing	55



Voorbeeld PreparedStatement

```
public Boek geefBoek(String isbn) throws BoekenNietBeschikbaarException {  
    Boek boek = null;  
    try {  
        try (Connection conn = geefVerbinding();  
             PreparedStatement stmt = conn.prepareStatement(sqlOpdrachten.getString("Q_BOEK"))){  
            stellen vanaf 1 → teller vanaf 1  
            stmt.setString(1, isbn);  
            ResultSet rs = stmt.executeQuery();  
            if (rs.next()) { → want 1 rij  
                boek = new Boek(  
                    rs.getString(sqlOpdrachten.getString("BOEK_ISBN")),  
                    rs.getString(sqlOpdrachten.getString("TITEL")),  
                    rs.getDouble(sqlOpdrachten.getString("PRIJS")));  
            }  
        }  
    } catch (SQLException e) { ... }  
    return boek;  
}
```

resource bundle



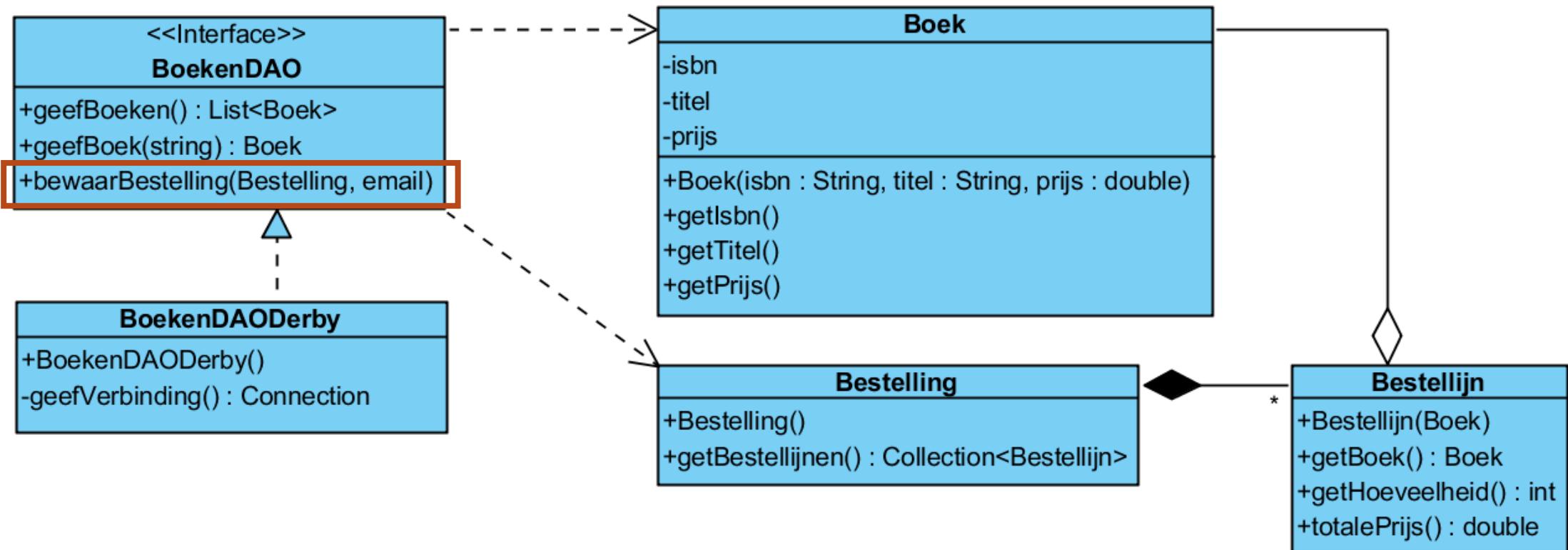
Voorbeeld PreparedStatement

sql.properties

```
# boek opvragen aan de hand van isbnnummer
Q_BOEK = select * from boeken where isbn = ?
# kolomnamen
BOEK_ISBN = isbn
TITEL = titel
PRIJS = prijs
```



Voorbeeld



Voorbeeld PreparedStatement

Tabel BESTELLINGEN		
AANTAL	KLANT	ISBNBOEK
4	veerle@hogent.be	isbn2
1	veerle@hogent.be	isbn3
1	test@test.be	isbn2
1	vo@gent.be	isbn5
2	vo@gent.be	isbn3
2	veerle@tiwi.be	isbn4
1	veerle@tiwi.be	isbn7



SQL-opdrachten

```
Q_BESTELLING = select aantal from bestellingen where klant = ? and isbnboek = ?  
AANTAL = aantal  
I_BESTELLING = insert into bestellingen (klant, isbnboek, aantal) values (?,?,?)  
U_BESTELLING = update bestellingen set aantal = ? where klant = ? and isbnboek = ?
```



```

try (Connection conn = geetVerbinding();
      PreparedStatement bestaatBestel = conn.prepareStatement(sqlOpdrachten.getString("Q_BESTELLING")))
{
    PreparedStatement voegBestelToe = conn.prepareStatement(sqlOpdrachten.getString("I_BESTELLING"));
    PreparedStatement pasBestelAan = conn.prepareStatement(sqlOpdrachten.getString("U_BESTELLING"));

    bestaatBestel.setString(1, email);
    voegBestelToe.setString(1, email);
    pasBestelAan.setString(2, email);

    for (Bestellijn lijn : bestelling.getBestellijnen()) {
        bestaatBestel.setString(2, lijn.getBoek().getIsbn()); → ISBN column in bepaalde tabel
        ResultSet bestaandeBestelling = bestaatBestel.executeQuery();
        if (bestaandeBestelling.next()) {
            int aantal = bestaandeBestelling.getInt(sqlOpdrachten.getString("AANTAL"));
            pasBestelAan.setString(3, lijn.getBoek().getIsbn());
            pasBestelAan.setInt(1, aantal + lijn.getHoeveelheid());
            pasBestelAan.executeUpdate();
        } else {
            voegBestelToe.setString(2, lijn.getBoek().getIsbn());
            voegBestelToe.setInt(3, lijn.getHoeveelheid());
            voegBestelToe.executeUpdate();
        }
    }
} catch (Exception e) {
    System.out.println(e.getMessage()); throw new BestellingMisluktExceptie(e);
}

```

[Handwritten notes]

bestaatBestel.setString(1, email); } => alle 3 tabellen hebben de kolom van de klant op verschillende tabel column. Bij sommigen is dat in column 1, anderan is dat een andere column nr.

voegBestelToe.setString(1, email); }

pasBestelAan.setString(2, email); }

for (Bestellijn lijn : bestelling.getBestellijnen()) {

bestaatBestel.setString(2, lijn.getBoek().getIsbn()); → ISBN column in bepaalde tabel

ResultSet bestaandeBestelling = bestaatBestel.executeQuery();

if (bestaandeBestelling.next()) {

int aantal = bestaandeBestelling.getInt(sqlOpdrachten.getString("AANTAL")); }

pasBestelAan.setString(3, lijn.getBoek().getIsbn()); }

pasBestelAan.setInt(1, aantal + lijn.getHoeveelheid()); }

pasBestelAan.executeUpdate(); }

} else {

voegBestelToe.setString(2, lijn.getBoek().getIsbn()); } → parameter uit tabel . bv:

voegBestelToe.setInt(3, lijn.getHoeveelheid()); } → 2e parameter in tabel was isbn nr. (nummer van de parameter).

voegBestelToe.executeUpdate(); }

}

}

}

catch (Exception e) {

System.out.println(e.getMessage()); throw new BestellingMisluktExceptie(e);



Parameter

- SQL-opdracht met parameter uitvoeren
- Herhaaldelijk uitvoer van dezelfde opdracht met andere waarden
- Gegevensconversie
- Voorkomen SQL-injectie



Gegevensconversie

- Aanhalingsteken in naam

```
String naam = "D'Haese";
```

- SQL-opdracht met knippen en plakken

```
String opdracht = "select * from studenten where naam = " + naam + ";"
```

- SQL-opdracht met parameter

```
String opdracht = "select * from studenten where naam = ?";
```



Gegevensconversie

- Aanhalingsteken in naam
 - Knippen en plakken geeft fout
- Gebruik Parameter
 - Opdracht wordt gecompileerd met parameter
 - Bij uitvoeren
 - Parameters doorsturen
 - Parameters worden geconverteerd naar juiste type
 - Parameters zijn argumenten voor gecompileerde functie



SQL-injectie

- Meestal in een webapplicatie
- Gebruiker
 - Parameter ingeven
 - Combineert parameter met SQL-opdracht
 - Injecteert SQL-opdrachten
 - Niet-toegankelijke gegevens bekijken
 - Gegevens verwijderen
 - ...



SQL-injectie: voorbeeld

- Ingave gebruiker: studentennummer
- SQL-opdracht

```
String opdracht = "select * from studenten where studnr = " + nummer;
```

- Malafide ingave

- 200200234 OR 1=1
- 200200234; DROP TABLE studenten
- 0 UNION SELECT username, password FROM users



SQL-injectie voorkomen → gebruik parameters!

- Opdracht wordt gecompileerd

```
select * from studenten where studnr = ?
```

- Instellen parameter converteert invoer naar een VARCHAR
- Deze VARCHAR is een parameter voor de voorgecompileerde SQL-opdracht



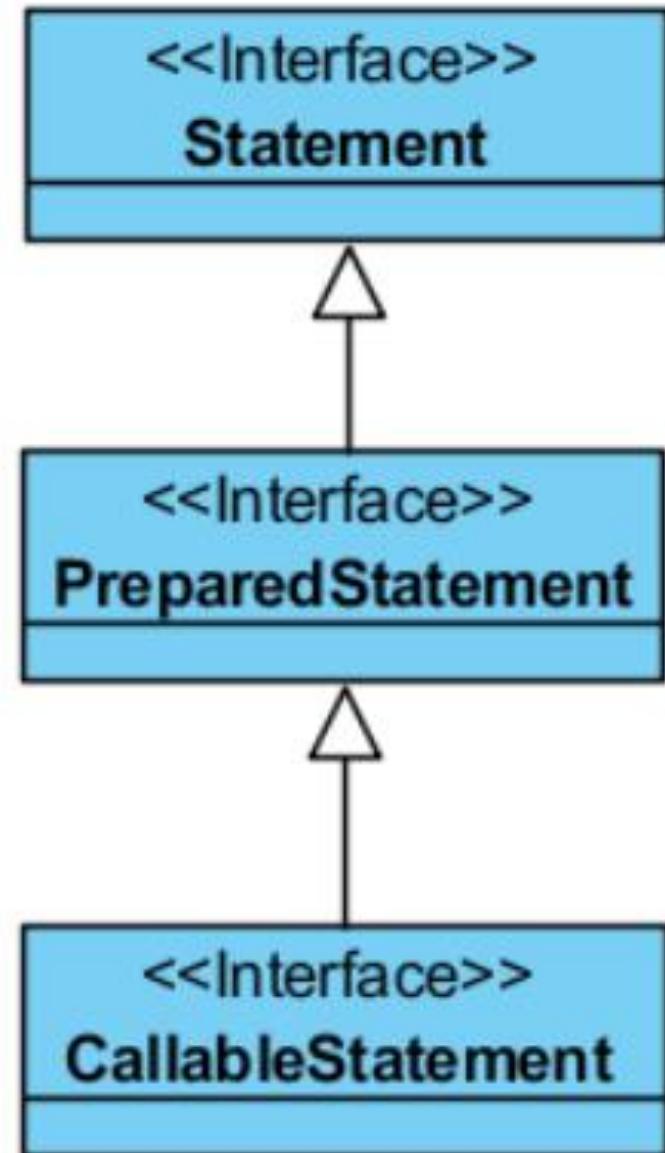
Overzicht

- Wat is JDBC?
- Basiswerking
 - Statements
 - PreparedStatements
 - CallableStatements
 - Transacties
- JDBC uitgebreid
 - DataSource
 - Scrollable en aanpasbare ResultSet
 - RowSets
 - SQL3-gegevenstypes



Prepared Statements

- Uitvoeren stored procedure
- Stored procedure
 - Functie of procedure
 - In gegevensbank
 - Geen standaardsyntax
 - Niet door alle systemen ondersteund
- JDBC API
 - Uniforme manier



Callable Statement

- Nodig
 - Naam procedure (bv. SELECT_BOEKEN)
 - Parameters
- Aanmaken CallableStatement

```
String opdracht = "{call SELECT_BOEKEN}";  
CallableStatement cstmt = conn.prepareCall(opdracht);
```

- Escape-syntax
 - Geen standaard SQL
 - Driver moet interpreteren of omzetten
- Verwijzing naar procedure in de databank



Voorbeeld CallableStatement

```
String opdracht = "{call BEWAAR_BESTEL(?, ?, ?)}";  
  
try (Connection conn = geefVerbinding();  
        CallableStatement cstmt = conn.prepareStatement(opdracht)) {  
  
    // parameters instellen  
    cstmt.setString(1, ...);  
    cstmt.setString(2, ...);  
    cstmt.setInt(3, ...);  
    cstmt.executeUpdate();  
  
}
```



Uitvoerparameters

- Corresponderen met uitvoerparameters van de stored procedure
- Methode
 - ? in de SQL-opdracht
 - Registeren
- Voorbeeld Opdracht
 - ? : aantal boeken

```
String opdracht = "{? = call SELECT_BOEKEN}";  
String opdracht = "{call SELECT_BOEKEN(?)}";
```



Registreren uitvoerparameter

- Opmerking

- Eerst ResultSet overlopen
- ? : bepaalt NIET het type parameter (in- of uit-)

```
CallableStatement cstmt = conn.prepareCall(opdracht);
cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
ResultSet rs = cstmt.executeQuery();
... // boeken ophalen
int aantalBoeken = cstmt.getInt(1);
```



Overzicht

- Wat is JDBC?
- Basiswerking
 - Statements
 - PreparedStatements
 - CallableStatements
 - **Transacties**
- JDBC uitgebreid
 - DataSource
 - Scrollable en aanpasbare ResultSet
 - RowSets
 - SQL3-gegevenstypes



Transacties

- Verschillende rijen en/of tabellen tegelijkertijd aanpassen
- Consistente gegevens *bv : geld afhalen en oversen op een andere rekening*
- Data integrity
 - Andere gebruikers zien pas aangepaste waarden als ze “definitief” zijn



Transacties

- Principe
 - Auto-commit-mode uitschakelen
 - Opdrachten uitvoeren
 - Geslaagd: commit → om permanent te maken
 - Mislukt: rollback
 - Auto-commit-mode inschakelen
- Auto-commit-mode
 - Standaard aan



Transacties

```
Connection conn = ...;
PreparedStatement stmt = ...;
try {
    stmt.setString(1,email);
    conn.setAutoCommit(false);
    for (Bestellijn lijn : bestelling.geefBestellijnen()) {
        stmt.setString(2,lijn.getBoek().getId());
        stmt.setInt(3,lijn.getHoeveelheid());
        stmt.executeUpdate();
    }
    conn.commit();
} catch (SQLException e) {    } => als net niet gelukt is
    conn.rollback();
} finally {
    conn.setAutoCommit(true);
}
```



Opmerkingen

- Let op: Connection-object lokaal
- Verschillende transacties
 - Verschillende Connection-objecten



Overzicht

- Wat is JDBC?
- Basiswerking
 - Statements
 - PreparedStatements
 - CallableStatements
 - Transacties
- JDBC uitgebreid
 - **DataSource**
 - Scrollable en aanpasbare ResultSet
 - RowSets
 - SQL3-gegevenstypes



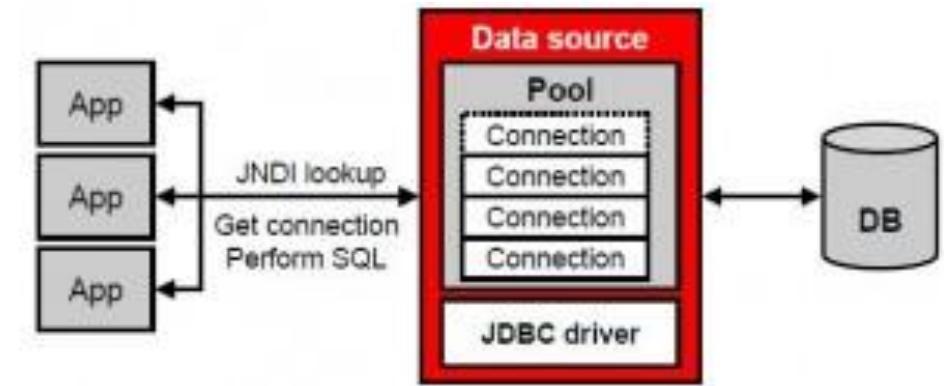
Driver versus DataSource

- java.sql
 - Gebruik Driver
- javax.sql
 - Gebruik DataSource
- DataSource
 - Applicatieservers of frameworks zoals Spring
 - Represeert een DBMS
 - Aangemaakt en ter beschikking gesteld via configuratie
 - Beschikbaar via JNDI (Java Naming and Directory Interface) of Dependency Injection



DataSource

- Beter alternatief voor JDBC-driver
 - Geen driverklasse, gebruikersnaam en wachtwoord in code
 - Gebruik connection pool
 - Ondersteunt gedistribueerde transacties
- Steunt op ofwel
 - JNDI: Java naming en directory services
 - Naming service: naam ~ object
 - Directory service uitbreiding naming service
 - Object heeft attributen
 - Zoeken op objecten mogelijk
 - Dependency Injection



bron: <http://jianmingli.com/wp/?p=5286>



Gebruik DataSource

- In applicatieserver bv. Glassfish
- Via framework bv. Spring
- Zorgen voor connection pool



Configuratie DataSource

- In configuratiebestand (Spring: application.properties)

- JDBC-URL
- Login
- Wachtwoord

```
spring.datasource.url=jdbc:postgresql://localhost:5432/iii
#spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.username=iii
spring.datasource.password=iiipwd
```

- Nodige drivers ter beschikking stellen bv. via pom.xml

Gebruik DataSource

- Ophalen via JNDI of Dependency Injection

```
private DataSource dataSource; → datasource object  
@Autowired  
public void setDataSource(DataSource dataSource) {  
    this.dataSource = dataSource;  
}
```

→ datasource object injection

Overzicht

- Wat is JDBC?
- Basiswerking
 - Statements
 - PreparedStatements
 - CallableStatements
 - Transacties
- JDBC uitgebreid
 - DataSource
 - Scrollable en aanpasbare ResultSet
 - RowSets
 - SQL3-gegevenstypes



Scrollable ResultSet

- Gegevensbank aanpassen mbv ResultSet-object
 - Cursor kan heen en weer verplaatst worden
 - Scrollable
 - Extra parameter bij aanmaken
 - TYPE_FORWARD_ONLY: enige mogelijke verplaatsing is één rij naar beneden
 - TYPE_SCROLL_INSENSITIVE: op een willekeurige wijze verplaatst worden
 - TYPE_SCROLL_SENSITIVE: op een willekeurige wijze verplaatst worden, weerspiegelt aanpassingen in gegevensbank
- om gegevens uit resultset te kunnen aanpassen*
- veranderingen wel ga kunnen zien*
- verandering niet gezien*
- van de dataset*



Aanpasbare ResultSet

- ResultSet gebruiken op gegevens aan te passen?

- Extra parameter bij aanmaken

- `ResultSet.CONCUR_READ_ONLY`

- `ResultSet.CONCUR_UPDATABLE` → de extra nodige parameter

- Niet door alle drivers ondersteund

```
Connection conn = ...;
String SQLZoekopdracht = ...;
Statement stmt = conn.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_READ_ONLY), → alleen info ophalen niet aanpassen
ResultSet rs = stmt.executeQuery(SQLZoekopdracht);
```



Verplaatsen cursor

- next
- previous
- first
- last
- beforeFirst
- afterLast
- relative(int rows)
- absolute(int row)



Rij aanpassen

```

public void modifyPrices(float percentage)
throws SQLException {
Statement stmt = null;
try (Connection con = ...;
      Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                              ResultSet.CONCUR_UPDATABLE)) {
ResultSet uprs = stmt.executeQuery("SELECT * FROM COFFEES");
while (uprs.next()) {
    float f = uprs.getFloat("PRICE");
    uprs.updateFloat("PRICE", f * percentage);    ↗ nieuwe waarde
    uprs.updateRow();                            ↗ naam van de kol
} catch (SQLException e) { ... }
}
  
```

inf from query
updateXxx(String/int, xxx)
van de col voor update!!

cancelRowUpdate()
updateRow()

naam van de kol



Rij toevoegen

```
public void insertRow(String coffeeName, int supplierID, float price,  
    int sales, int total) throws SQLException {  
    try (Connection con = ...;  
        Statement stmt = con.createStatement(  
            ResultSet.TYPE_SCROLL_SENSITIVE,  
            ResultSet.CONCUR_UPDATABLE) {  
        ResultSet uprs = stmt.executeQuery("SELECT * FROM COFFEES");  
        uprs.moveToInsertRow(); → verplaatst de cursor waar we nieuwe informatie willen toevoegen  
        uprs.updateString("COF_NAME", coffeeName);      updateXxx(String/int, xxx)  
        uprs.updateInt("SUP_ID", supplierID);  
        uprs.updateFloat("PRICE", price);  
        uprs.updateInt("SALES", sales);  
        uprs.updateInt("TOTAL", total);  
        uprs.insertRow(); → voeg het toe  
        uprs.moveToCurrentRow();  
    } catch (SQLException e) { ... }
```

(Handwritten notes)

→ tabel dat we willen aanpassen

→ verplaatst de cursor waar we nieuwe informatie willen toevoegen

updateXxx(String/int, xxx)

↓

col

↓

nieuwe waarde



Rij verwijderen

```
ResultSet uprs = ...;  
int rijNr ...;  
uprs.absolute(rijNr);  
uprs.deleteRow();
```

→ cursor verplaatsen naar die row



Zoekopdracht

- Aanpasbare ResultSet

- Eén tabel
- Geen ‘join’- of GROUP BY-clausule
- Primaire sleutel opgevraagd
- Lees- en schrijfrechten

- Toevoegen rij

- Alle kolommen die niet leeg mogen zijn, opgevraagd
- Alle kolommen zonder standaardwaarde, opgevraagd



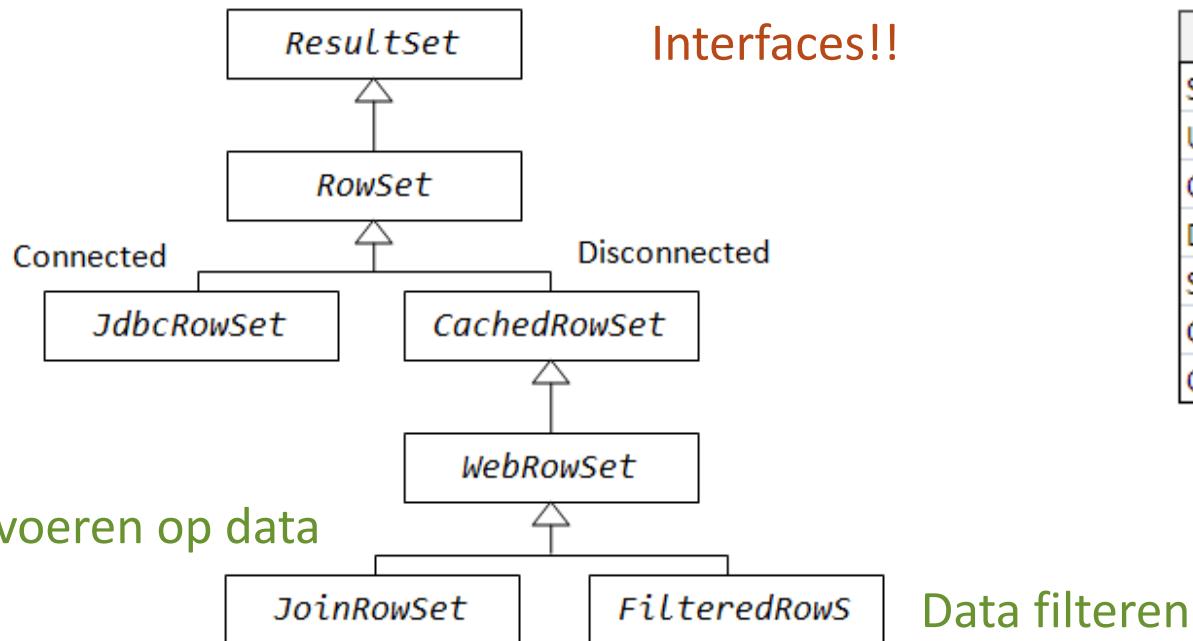
Overzicht

- Wat is JDBC?
- Basiswerking
 - Statements
 - PreparedStatements
 - CallableStatements
 - Transacties
- JDBC uitgebreid
 - DataSource
 - Scrollable en aanpasbare ResultSet
 - RowSets
 - SQL3-gegevenstypes



RowSet

- Scrollable Resultset niet ondersteund door alle drivers
- Scrollable ResultSet: connectie blijft open



Features	JdbcRowSet	CachedRowSet	WebRowSet
Scrollable	x	x	x
Updateable	x	x	x
Connected	x	x	x
Disconnected		x	x
Serializable		x	x
Generate XML			x
Consume XML			x



RowSet

- Functioneren als javabean
 - Properties
 - Events
 - Insert, update, delete, bewegen cursor, verandering inhoud
 - Registeren luisteraars
- Scrollable en Updatable
 - Indien niet ondersteund door driver



Aanmaak RowSet

```
RowSetFactory myRowSetFactory = RowSetProvider.newFactory();  
JdbcRowSet jdbcRs = myRowSetFactory.createJdbcRowSet();  
  
jdbcRs.setUrl("jdbc:myDriver:myAttribute");  
jdbcRs.setUsername("username");  
jdbcRs.setPassword("password");  
    setDatasourceName  
    SQL opdracht geven  
    ↗  
jdbcRs.setCommand("select * from COFFEES");  
jdbcRs.execute();  
  
...
```

Rowset aanmaken

createJdbcRowSet
createCachedRowSet
createFilteredRowSet
createJoinRowSet
createWebRowSet



Alternatief configuratie RowSet

```
RowSetFactory myRowSetFactory = RowSetProvider.newFactory();
CachedRowSet cRs = myRowSetFactory.createCachedRowSet();

Connection conn = ...;

cRs.setCommand("select * from COFFEES");
cRs.execute(conn);

...
```

NIET voor een
JdbcRowSet



CachedRowSet

- Ontkoppeld
- Bij aanmaak
 - SyncProvider → sync tussen database en rowset,
 - RowSetReader
 - RowSetWriter
- Sleutelkolommen specifiëren
 - Nodig voor update
- Explicit synchroniseren met de databank

```
CachedRowSet crs = ...;  
...  
crs.acceptChanges();
```

```
CachedRowSet crs = ...;  
int [] keys = {1};  
crs.setKeyColumns(keys);
```

primary key
bevatten

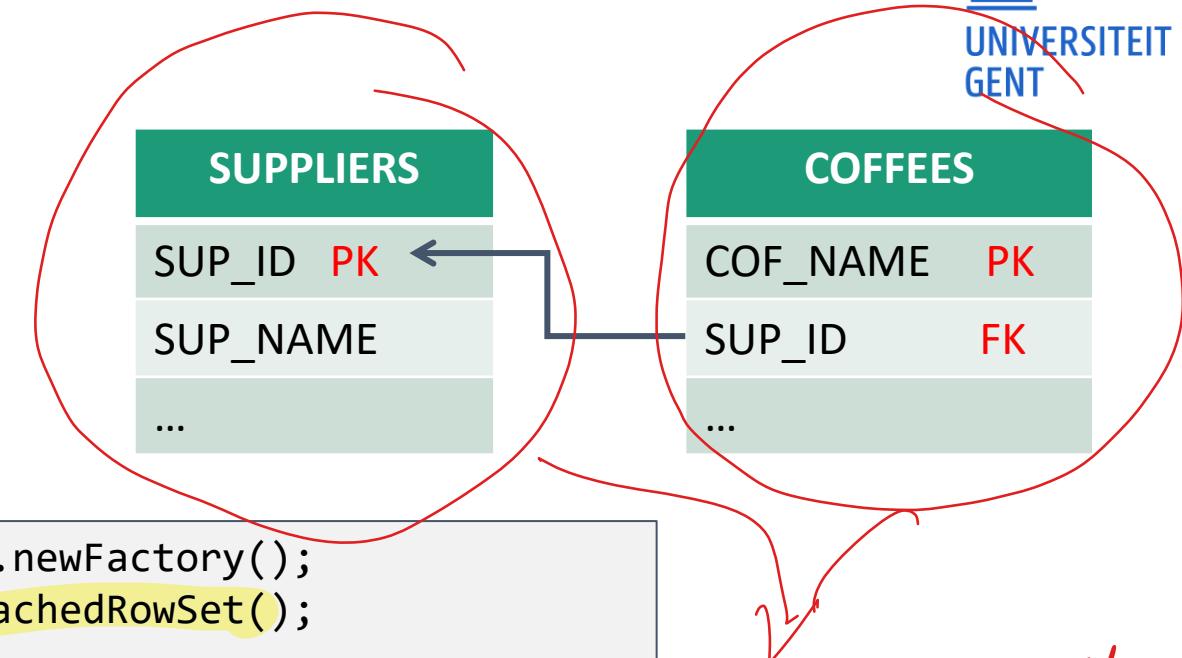
→ deze nummers bevatten
de primary key



→ als je informatie uit meerdere tabellen wilt combineren in 1 rowset
object

JoinRowSet aanmaken

- Informatie uit meerdere tabellen
- Combineren met "JOIN"
 - Standaard innerjoin



```
RowSetFactory myRowSetFactory = RowSetProvider.newFactory();
CachedRowSet coffees = myRowSetFactory.createCachedRowSet();
...
coffees.setCommand("select * from COFFEES");
coffees.execute();

CachedRowSet suppliers= myRowSetFactory.createCachedRowSet();
...
suppliers.setCommand("select * from SUPPLIERS ");
suppliers.execute();

JoinRowSet jrs = myRowSetFactory.createJoinRowSet();
jrs.addRowSet(coffees, "SUP_ID"); → tabel waarop de JOIN gevest (naam)
                                ↓ kolomnaam of nummer (vanaf 1)
```

cached rowset
elk apart
→ samen in een
joined rowset gecombineerd.



JoinRowSet gebruiken

```
System.out.println("Coffees bought from " + supplierName + ": ");  
  
while (jrs.next()) {  
    if (jrs.getString("SUP_NAME").equals(supplierName)) {  
        String coffeeName = jrs.getString(1);  
        System.out.println("      " + coffeeName);  
    }  
}
```

```
Coffees bought from Acme, Inc.:  
    Colombian  
    Colombian_Decaf
```



FilteredRowSet

- Zichtbare data beperken/filteren
- “WHERE”
- Filter op kolom(men)
 - Naam of volgnummer
 - Interface javax.sql.rowset.Predicate

```
boolean evaluate(Object value, int column);
boolean evaluate(Object value, String columnName);
boolean evaluate(ResultSet rs);
```



FilteredRowSet aanmaken

```
RowSetFactory myRowSetFactory = RowSetProvider.newFactory();
FilteredRowSet frs = myRowSetFactory.createFilteredRowSet();
...
frs.setCommand("SELECT * FROM COFFEE_HOUSES");
frs.execute();

String[] cityArray = { "SF", "LA" };
CityFilter myCityFilter = new CityFilter(cityArray, 2);
frs.setFilter(myCityFilter);
this.viewFilteredRowSet(frs);
```

COFFEE_HOUSES

STORE_ID PK

CITY

...

*) column waarop
je gaat filteren*



Voorbeeld Filter

```
public class CityFilter implements Predicate {  
  
    private String[] cities;  
    private String colName = null;  
    private int colNumber = -1;  
  
    public CityFilter(String[] citiesArg, String colNameArg) {  
        this.cities = citiesArg;  
        this.colNumber = -1;  
        this.colName = colNameArg;  
    }  
  
    public CityFilter(String[] citiesArg, int colNumberArg) {  
        this.cities = citiesArg;  
        this.colNumber = colNumberArg;  
        this.colName = null;  
    }  
    ...  
}
```

1 van de beide



Voorbeeld Filter

```
public class CityFilter implements Predicate {  
    ...  
  
    public boolean evaluate(Object valueArg, String colNameArg) {  
        if (colNameArg.equalsIgnoreCase(this.colName)) {  
            for (int i = 0; i < this.cities.length; i++) {  
                if (this.cities[i].equalsIgnoreCase((String)valueArg)) {  
                    return true;  
                }  
            }  
        }  
        return false;  
    }  
    ...  
}
```

true als valArg een van de opgegeven steden bevat anders false



Voorbeeld Filter

```
public class CityFilter implements Predicate {  
    ...  
    HIER OP VOLGENR  
    public boolean evaluate(Object valueArg, int colNumberArg) {  
        if (colNumberArg == this.colNumber) {  
            for (int i = 0; i < this.cities.length; i++) {  
                if (this.cities[i].equalsIgnoreCase((String)valueArg)) {  
                    return true;  
                }  
            }  
        }  
        return false;  
    }  
    ...  
}
```



Voorbeeld Filter

```
public class CityFilter implements Predicate {
    ...
    public boolean evaluate(ResultSet rs) {
        if (rs == null) return false;
        try {
            for (int i = 0; i < this.cities.length; i++) {
                String cityName = null;
                if (this.colNumber > 0) {
                    cityName = (String)rs.getObject(this.colNumber);
                } else if (this.colName != null) {
                    cityName = (String)rs.getObject(this.colName);
                } else {
                    return false;
                }
                if (cityName.equalsIgnoreCase(cities[i])) {
                    return true;
                }
            }
        } catch (SQLException e) { return false;}
        return false;
    }
}
```



Overzicht

- Wat is JDBC?
- Basiswerking
 - Statements
 - PreparedStatements
 - CallableStatements
 - Transacties
- JDBC uitgebreid
 - DataSource
 - Scrollable en aanpasbare ResultSet
 - RowSets
 - SQL3-gegevenstypes



SQL3-gegevenstypes

SQL3-type	Javatype	Omschrijving
BLOB	Blob	<i>Binary Large Object</i>
CLOB	Clob	<i>Character Large Object</i>
ARRAY	Array	Een tabel van waarden
gestructureerd SQL-type	Struct	Een zelfgemaakt gestructureerd SQL-type
REF	Ref	Een verwijzing naar een instantiatie van een zelfgemaakt gestructureerd SQL-type
DISTINCT	Javatype van het type waarvan het DISTINCT-type is afgeleid	Een nieuw type afgeleid van een basistype. Bijvoorbeeld een type <i>taalcode</i> of <i>ISBN-nummer</i> afgeleid van het type <i>VARCHAR</i>

