# Intec Brussel Exam 06 : JDBC

| Evaluatie categorieën | Score |
|---|---|
| Analyseren | __ / 35 EP |
| SQL broncode | __ / 65 EP |
| Totaal | 00 / 100 EP |
| Score | |
| | |

## Inschrijvingsformulier

| Cursist | |
|---|---|
| Voornaam | |
| Familienaam | |
| Klascode | Java Juni 21 |
| Notities | |

## Algemeen info over het examen:

| Het examensdatum | Geadviseerde deadline | De laatste deadline |
|---|---|---|
| 25/10/2021 (start om 10:00) | 25/10/2021 Ma 17:00 | 28/10/21 Do 17:00 |
| | | |
| **Contact info van de instructeur** | yilmaz.mustafa@intecbrussel.be | +32 467 71 17 09 |

### Vragen & Antwoorden voor de start

| Vraag | Antwoord |
| --- | --- |
| Moet ik alle code zelf programmeren zonder een oplossing te zoeken via zoekmachines? | Nee, je mag alle resources gebruiken om te je vragen te antwoorden. Probeer echter niet te veel tijd te besteden aan het zoeken. Soms kost het meer tijd om tijd te besteden aan het vinden van een bestaand antwoord dan het zelf te schrijven. |
| Mag ik hulp of codestuks vragen aan de andere cursisten? | Nee, het is een volledig individueel examen. |

| | |
| --- | --- |
| **Als ik de eerste deadline mis, mag ik dan doorgaan met het examen?** | **Natuurlijk is er hieronder een scoretabel |
| die de berekening van scores per dag laat zien.** | |
| Geadviseerde datum | Je krijgt 100% van je score |
| Na 1 dag | Je krijgt 85% van je score |
| Na 2 dag | Je krijgt 70% van je score |
| Na 3 dag | Je krijgt 60% van je score |

---

## De oplossingscode moet in Java en SQL gecodeerd worden.

## Gebruik hiervoor MySQL v8+

## Het examen is open om iets van online te zoeken, van je boek te bekijken, maar mogen jullie niet elkaar antwoorden vragen.

## JDK11 en Maven moet geïnstalleerd worden.

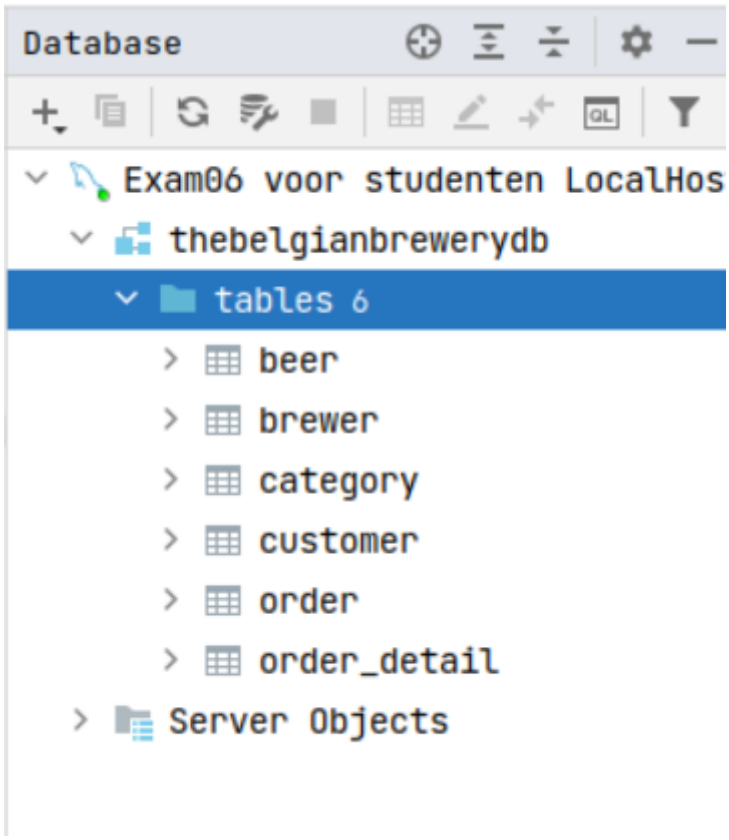
---

**(Het begin van de vragen. Veel succes allemaal!)**

TIP: **Vergeet niet** om jullie **SQL-oplossingscode** hier toe te voegen.

1. Creëer een gebruiker met de gebruikersnaam 'customer' en met een wachtwoord 'P@ssw0rd' (0 als getal zero)
2. Creëer de database 'thebelgianbrewerydb': (SQL queries voor het genereren van mock data worden in het zip file toegevoegd.)

1. Geef een volledig toegang van de database 'thebelgianbrewerydb' naar 'customer'@'localhost'.
2. Voer de volgende SQL queries uit:




```sql
create table `customer`
(
   `id`            int unique auto_increment,
   `first_name`    varchar(100)                not null,
   `last_name`     varchar(100)                not null,
   `email`         nvarchar(255) unique        not null,
   `passcode`      nvarchar(255)               null,
   `registry_date` date    default '2000-01-01' null,
   `active`        boolean default true        null,

   primary key (`id`)

) charset = utf8;

create table `category`
(
   `id`    int unique auto_increment,
   `title` varchar(255)        not null,
   `slug`  varchar(255) unique not null,

   primary key (`id`)

) charset = utf8;

create table `brewer`
(
   `id`       int unique auto_increment,
   `name`     varchar(50)   null,
```

```
    `address`  varchar(50)   null,
    `postcode` varchar(10)   null,
    `city`     varchar(50)   null,
    `turnover` int default 0 null,

    primary key (`id`)

) charset = utf8;

create table `beer`
(
    `id`          int unique auto_increment,
    `name`        varchar(100)       null,
    `brewer_id`   int                not null,
    `category_id` int                not null,
    `price`       float default 0.00 null,
    `stock`       int   default 0    null,
    `alcohol`     float default 0.00 null,
    `version`     int   default 0    null,
    `image_url`   varchar(255)       null,
    constraint brewer_to_beer_fk foreign key (brewer_id) references brewer (id),
    constraint category_to_beer_fk foreign key (category_id) references category
(Id)
) charset = utf8;


create table `order`
(
    `customer_id` int      not null,
    `brewer_id`   int      not null,
    `quantity`    int      null default 1,
    `discount`    float    null default 0.00,
    `payment`     float    null default 0.00,

    constraint brewer_to_order_fk foreign key (`brewer_id`) references brewer
(`id`),
    constraint customer_to_order_fk foreign key (`customer_id`) references
customer (`id`),

    primary key (`customer_id`, `brewer_id`)

) charset = utf8;
```

# Insert Statements:

Om **willekeurige** records te **genereren** (insert), heb ik al enkele sql-bestanden voor je gemaakt.
Voor meer info navigeer [ src → main → sql → thebelgianbrewerydb_***.sql ]

# Toepassingsvereisten:

Alle bronnen die je nodig hebt om SQL-opdrachten uit te voeren, worden toegevoegd aan het zip-bestand.
Tijdens het examen bent je verantwoordelijk voor het maken van een database, het maken van de vereiste mysql-gebruiker die wordt vermeld in de volgende taken

en het één voor één uitvoeren van alle SQL-query's om records aan te maken.
Het volgen van deze stappen is onderdeel van het examen.
Je kunt altijd begeleiding vragen, maar de instructeur kan je daar geen oplossing voor geven.

Frontend-app-ontwikkelaars wachten tot je werk voltooid is.
Aan de Java-kant werk je **alleen** aan het pakket '**repositories**'.
Alle andere klassen zijn ontwikkeld door ch€f. Je hoeft **niet** een opnieuw klas aan te maken.
Het is een veel voorkomende benadering om problemen in de software-industrie op te lossen.

# Zakelijke vereisten:

**DBB: De Belgische Brouwerij**

De applicatie-GUI is nog in ontwikkeling.
Het Belgische Brouwerijbedrijf vraagt je een verbinding te schrijven tussen java- en mysql-codes (repository codes).
**Wijzig** daarom **GEEN** enkele code in het '**views**'-pakket. Het zijn slechts testdoeleinden.
De volgende maven-commando is voldoende om je code opnieuw in een *.jar te pakken en te testen:


```
mvn clean javafx:run -Pbrewer
```

In het bovenstaande voorbeeld is -Pbrewer een dynamische parameter. Als u Brewer-weergave wilt starten, moet u een opdrachtregelparameter -Pbrewer doorgeven. Als u wilt zien hoe de weergave van de klant eruitziet, moet u deze vervangen door de opdracht -Pcustomer. Bekijk uw pom.xml om meer details en opties te zien.

```
mvn clean javafx:run -Pcustomer
```

of om de front-end-app voor bieren te bekijken:

```
mvn schoon javafx:run -Pbeer
```

Voor elk repository-klasse heeft ch€f fields (variabelen) op klassenniveau gemaakt:
Connection connection, PreparedStatement prepareStatement, en ResultSet results.

Je hoeft deze klassen **NIET** voor elke keer **apart te declareren**. Initialiseer opnieuw ze voor elke methode.
Bijvoorbeeld: als je een verbinding met de database wilt maken of een PreparedStatement-instantie wilt genereren, moet je de volgende code gebruiken in elk repository-methode (create(), update(), read(), delete() of all andere methoden die gaan verbinden naar de DB):

## TIP: Als je deze 3 variabelen verwijderen, en eigen logica implementeren, het is ook aanvaardbaar.

Hoe je de code schrijft, of hoe je problemen oplost, is niet beperkt tot één techniek.
Je kunt je eigen stijl kiezen om alle TODO's in het examen op te lossen.

```
connection = getConnection();
statement = connection.prepareStatement(query);
```

# OPDRACHTEN:

Alle taken worden vermeld in de java-codes. Je zult veel TODO's zien.
Dit zijn de codeblokken die je moet zelf-coderen en laten werken.
Wanneer alle TODO's zijn voltooid, moet je de app testen en een **schermafbeelding** (screenshot) krijgen van BeerGUI-, BrewerGUI- en CustomerGUI- views.

## OPDRACHTEN VOOR de klas 'ConnectionFactory'

```
package be.intecbrussel.javajuni21.exams.exam06.repositories;


import java.sql.*;


public class ConnectionFactory {

    // TODO: creëer een database genoemd thebelgianbrewerydb

    // TODO: maak een mysql gebruiker met naam ' customer ' en wachtwoord '
P@ssw0rd ' aan.

    String connectionUrl = "jdbc:mysql://localhost:3306/thebelgianbrewerydb";
    String dbUser = "customer";
    String dbPwd = "P@ssw0rd";

    private static ConnectionFactory connectionFactory = null;

    private ConnectionFactory() {

    }

    public Connection getConnection() throws SQLException {
        Connection conn = null;
        conn = DriverManager.getConnection(connectionUrl, dbUser, dbPwd);
        return conn;
    }

    public static ConnectionFactory getInstance() {
        if (connectionFactory == null) {
            connectionFactory = new ConnectionFactory();
```

```
        }
        return connectionFactory;

    }
}
```

## OPDRACHTEN VOOR de klas 'BeerRepository'

```
package be.intecbrussel.javajuni21.exams.exam06.repositories;

import be.intecbrussel.javajuni21.exams.exam06.models.entities.Beer;

import java.sql.*;
import java.util.*;

import be.intecbrussel.javajuni21.exams.exam06.exceptions.*;

public class BeerRepository {

    private Connection connection = null;
    private PreparedStatement statement = null;
    private ResultSet results = null;

    private Connection getConnection() throws SQLException {
        Connection conn;
        conn = ConnectionFactory.getInstance().getConnection();
        return conn;
    }

    public int create(Beer newBeer) throws BeerException {

        int noOfRecordsEffected = 0;

        try {

            // TODO: voeg je code hier toe ..

        } catch (SQLException sqlException) {
            BeerException ex = new BeerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BeerException ex = new BeerException(exception.getMessage());
                throw ex;
            }
```

```
        }
        return noOfRecordsEffected;
    }

    public Beer read(int id) throws BeerException {

        Beer result = new Beer();

        try {

            // TODO: voeg je code hier toe ..

        } catch (SQLException sqlException) {
            BeerException ex = new BeerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BeerException ex = new BeerException(exception.getMessage());
                throw ex;
            }

        }

        return result;

    }

    public List<Beer> read(Beer example) throws BeerException {

        List<Beer> beerList = new ArrayList<>();

        try {

           // TODO: voeg je code hier toe ..

        } catch (SQLException sqlException) {
            BeerException ex = new BeerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BeerException ex = new BeerException(exception.getMessage());
                throw ex;
```

```
            }

        }

        return beerList;

    }

    public List<Beer> read() throws BeerException {

        List<Beer> beerList = new ArrayList<>();

        try {

            // TODO: voeg je code hier toe ..

        } catch (SQLException sqlException) {
            BeerException ex = new BeerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BeerException ex = new BeerException(exception.getMessage());
                throw ex;
            }

        }

        return beerList;

    }

    public int update(long id, Beer existingBeer) throws BeerException {

        if (id < 0) {
            throw new BeerException("Beer ID is required.").requiredFields("id");
        }

        if (existingBeer == null) {
            throw new BeerException("Beer is required.").nullBeerException();
        }

        int noOfRecordsEffected = 0;

        try {

            // TODO: voeg je code hier toe ..

        } catch (SQLException sqlException) {
            BeerException ex = new BeerException(sqlException.getMessage());
            throw ex;
```

```
        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BeerException ex = new BeerException(exception.getMessage());
                throw ex;
            }

        }

        return noOfRecordsEffected;

    }

    public int delete(long id) throws BeerException {

        int noOfRecordsEffected = 0;

        try {

            // TODO: voeg je code hier toe ..

        } catch (SQLException sqlException) {
            BeerException ex = new BeerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BeerException ex = new BeerException(exception.getMessage());
                throw ex;
            }

        }

        return noOfRecordsEffected;
    }

}
```

## OPDRACHTEN VOOR de klas 'BrewerRepository'

```
package be.intecbrussel.javajuni21.exams.exam06.repositories;

import java.sql.*;
import java.util.*;

import be.intecbrussel.javajuni21.exams.exam06.exceptions.*;
import be.intecbrussel.javajuni21.exams.exam06.models.entities.Brewer;

public class BrewerRepository {

    private Connection connection = null;
    private PreparedStatement statement = null;
    private ResultSet results = null;

    private Connection getConnection() throws SQLException {
        Connection conn;
        conn = ConnectionFactory.getInstance().getConnection();
        return conn;
    }

    public int create(Brewer newBrewer) throws BrewerException {

        int noOfRecordsEffected = 0;

        try {

            // TODO: voeg je code hier toe..

        } catch (SQLException sqlException) {
            BrewerException ex = new BrewerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BrewerException ex = new BrewerException(exception.getMessage());
                throw ex;
            }

        }
        return noOfRecordsEffected;
    }

    public Brewer read(long id) throws BrewerException {

        Brewer result = new Brewer();

        try {
```

```
            // TODO: voeg je code hier toe..

        } catch (SQLException sqlException) {
            BrewerException ex = new BrewerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BrewerException ex = new BrewerException(exception.getMessage());
                throw ex;
            }

        }

        return result;

    }

    public List<Brewer> read(Brewer example) throws BrewerException {

        try {

            // TODO: voeg je code hier toe..

        } catch (SQLException sqlException) {
            BrewerException ex = new BrewerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BrewerException ex = new BrewerException(exception.getMessage());
                throw ex;
            }

        }

        // return empty collection if fails
        return Collections.emptyList();

    }

    public List<Brewer> read() throws BrewerException {
```

```
        List<Brewer> brewerList = new ArrayList<>();

        try {

            // TODO: voeg je code hier toe ..

        } catch (SQLException sqlException) {
            BrewerException ex = new BrewerException(sqlException.getMessage());
            throw ex.notFound();

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BrewerException ex = new BrewerException(exception.getMessage());
                throw ex;
            }

        }

        return brewerList;
    }

    public int update(long id, Brewer existingBrewer) throws BrewerException {

        int noOfRecordsEffected = 0;

        try {

            // TODO: voeg je code hier toe ..

        } catch (SQLException sqlException) {
            BrewerException ex = new BrewerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BrewerException ex = new BrewerException(exception.getMessage());
                throw ex;
            }

        }

        return noOfRecordsEffected;

    }
```

```
    public int delete(long id) throws BrewerException {

        int noOfRecordsEffected = 0;

        try {

           // TODO: voeg je code hier toe ..

        } catch (SQLException sqlException) {
            BrewerException ex = new BrewerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                BrewerException ex = new BrewerException(exception.getMessage());
                throw ex;
            }

        }

        return noOfRecordsEffected;
    }

}
```

## OPDRACHTEN VOOR de klas 'CategoryRepository'

```
package be.intecbrussel.javajuni21.exams.exam06.repositories;

import be.intecbrussel.javajuni21.exams.exam06.models.entities.Category;
import java.sql.*;
import java.util.*;
import be.intecbrussel.javajuni21.exams.exam06.exceptions.*;

public class CategoryRepository {

    private Connection connection = null;
    private PreparedStatement statement = null;
    private ResultSet results = null;

    private Connection getConnection() throws SQLException {
            Connection conn;
            conn = ConnectionFactory.getInstance().getConnection();
            return conn;
    }
```

```
    public int create(Category newCategory) throws CategoryException {

        int noOfRecordsEffected = 0;

        try {

            // TODO: voeg je code hier ..

        } catch (SQLException sqlException) {
            CategoryException ex = new
CategoryException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                CategoryException ex = new
CategoryException(exception.getMessage());
                     throw ex;
            }

        }
        return noOfRecordsEffected;
    }

    public Category read(long id) throws CategoryException {

        Category result = new Category();

        try {

            // TODO: voeg je code hier ..

        } catch (SQLException sqlException) {
            CategoryException ex = new
CategoryException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                CategoryException ex = new
CategoryException(exception.getMessage());
                     throw ex;
            }
```

```
        }

        return result;

    }

    public List<Category> read(Category example) throws CategoryException {

    List<Category> categoryList = new ArrayList<>();

        try {

            // TODO: voeg je code hier ..

        } catch (SQLException sqlException) {
            CategoryException ex = new
CategoryException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                CategoryException ex = new
CategoryException(exception.getMessage());
                     throw ex;
            }

        }

        return categoryList;

    }

    public List<Category> read() throws CategoryException {

        List<Category> categoryList = new ArrayList<>();

        try {

            // TODO: voeg je code hier..

        } catch (SQLException sqlException) {
            CategoryException ex = new
CategoryException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
```

```
                    connection.close();

            } catch (Exception exception) {
                CategoryException ex = new
CategoryException(exception.getMessage());
                    throw ex;
            }

        }

        return categoryList;

    }

    public int update(long id, Category existingCategory) throws
CategoryException {

        if(id < 0) {
            throw new CategoryException("Category ID is
required.").requiredFields("id");
        }

        if (existingCategory == null) {
            throw new CategoryException("Category is
required.").nullCategoryException();
        }

        int noOfRecordsEffected = 0;

        try {

            // TODO: voeg je code hier ..

        } catch (SQLException sqlException) {
            CategoryException ex = new
CategoryException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                CategoryException ex = new
CategoryException(exception.getMessage());
                    throw ex;
            }

        }

        return noOfRecordsEffected;

    }
```

```
    public int delete(long id) throws CategoryException {

        int noOfRecordsEffected = 0;

        try {

            // TODO: voeg je code hier ..

        } catch (SQLException sqlException) {
            CategoryException ex = new
CategoryException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                CategoryException ex = new
CategoryException(exception.getMessage());
                     throw ex;
            }

        }

        return noOfRecordsEffected;
    }

}
```

## OPDRACHTEN VOOR de klas 'CategoryRepository'

```
package be.intecbrussel.javajuni21.exams.exam06.repositories;

import be.intecbrussel.javajuni21.exams.exam06.models.entities.Customer;

import java.sql.*;
import java.util.*;

import be.intecbrussel.javajuni21.exams.exam06.exceptions.*;

public class CustomerRepository {

    private Connection connection = null;
    private PreparedStatement statement = null;
    private ResultSet results = null;

    private Connection getConnection() throws SQLException {
        Connection conn;
```

```
        conn = ConnectionFactory.getInstance().getConnection();
        return conn;
    }

    public int create(Customer newCustomer) throws CustomerException {

        int noOfRecordsEffected = 0;

        try {

            // TODO: voeg je code hier ..

        } catch (SQLException sqlException) {
            CustomerException ex = new
CustomerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                CustomerException ex = new
CustomerException(exception.getMessage());
                throw ex;
            }

        }
        return noOfRecordsEffected;
    }

    public Customer read(long id) throws CustomerException {

        Customer result = new Customer();

        try {

        // TODO: voeg je code hier ..

        } catch (SQLException sqlException) {
            CustomerException ex = new
CustomerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
```

```
                CustomerException ex = new
CustomerException(exception.getMessage());
                throw ex;
            }

        }

        return result;

    }

    public List<Customer> read(Customer example) throws CustomerException {

        List<Customer> customerList = new ArrayList<Customer>();

        try {

            // TODO: voeg je code hier

        } catch (SQLException sqlException) {
            CustomerException ex = new
CustomerException(sqlException.getMessage());
            throw ex.notFound();

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                CustomerException ex = new
CustomerException(exception.getMessage());
                throw ex;
            }

        }

        // return empty collection if fails
        return customerList;

    }

    public List<Customer> read() throws CustomerException {

        List<Customer> customerList = new ArrayList<>();

        try {

            // TODO: voeg je code hier ..

        } catch (SQLException sqlException) {
            CustomerException ex = new
CustomerException(sqlException.getMessage());
            throw ex;
```

```
        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                CustomerException ex = new
CustomerException(exception.getMessage());
                throw ex;
            }

        }

        return customerList;

    }

    public int update(long id, Customer existingCustomer) throws
CustomerException {

        if (id < 0) {
            throw new CustomerException("Customer ID is
required.").requiredFields("id");
        }

        if (existingCustomer == null) {
            throw new CustomerException("Customer is
required.").nullCustomerException();
        }

        int noOfRecordsEffected = 0;

        try {

           // TODO: voeg je code hier ..

        } catch (SQLException sqlException) {
            CustomerException ex = new
CustomerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                CustomerException ex = new
CustomerException(exception.getMessage());
                throw ex;
            }
```

```
        }

        return noOfRecordsEffected;

    }

    public int delete(long id) throws CustomerException {

        int noOfRecordsEffected = 0;

        try {

            // TODO: voeg je code hier ..

        } catch (SQLException sqlException) {
            CustomerException ex = new
CustomerException(sqlException.getMessage());
            throw ex;

        } finally {

            try {
                if (statement != null)
                    statement.close();
                if (connection != null)
                    connection.close();

            } catch (Exception exception) {
                CustomerException ex = new
CustomerException(exception.getMessage());
                throw ex;
            }

        }

        return noOfRecordsEffected;
    }

}
```

## OPDRACHTEN VOOR de klas 'OrderRepository'

Maak zelf een opnieuw OrderRepository-klasse aan.
OrderRepository moet de interface TemplateRepository implementeren.

## BEERS-ZOEKENGINE

Ontwikkel een andere methode in de repository-klassen genoemd 'search(String keyword)'.
Voeg hiervoor de juiste oplossing om iets van een tabel te zoeken.
Je kunt kiezen wat je wilt zoeken of waar je wilt zoeken,
bijvoorbeeld een methode maken die alle records op naam doorzoekt.

## CUSTOMER-LOGIN-LOGOUT (OPTIONEEL)

Maak 2 methoden aan in de klantentabel aan, namelijk: inloggen (String e-mail, String wachtwoord) en uitloggen (String e-mail).
Schrijf hier eigen code om een user-validatie mechanisme te ontwikkelen.
Als email en wachtwoord matchen met de database records dan returneer terug true,
als er geen user in de database zit, of wachtwoord onjuist is dan retourneer false.
om een logout mechanisme te ontwerpen moet je een extra field genoemd 'logged' naar customers-tabel toevoegen.
De data-typ van de column moet boolean zijn. 2de stap na je een COLUMN aanmaakt is om een field toe te voegen aan de klas Customer.
Daarna moet je verzorgen dat CustomerRepository methoden gaat ook implementatie hebben om nieuwe data te beheren.

## TIP: pom.xml-bestand blijft ongewijzigd.

Als je nog een afhankelijkheid wilt importeren, zal ik je niet stoppen met experimenteren. Als je experimentele code werkt, dan krijg je een extra score.

**(Het einde van de vragen. Veel succes allemaal!)**

---

# PROJECT ALS ZIP

Maak een zip-bestand van je voltooide project.
Vergeet niet om screenshots toe te voegen aan je zip-bestand.
src, pom.xml en alle andere vereiste bronnen moeten in de zip zitten.
Aan de andere kant mag elk bestand met de extensie .iml
of elke map met de naam .idea NOOIT aan de zip toegevoegd worden.
Dit zijn instellingen (configs) van Intellij-applicaties die speciaal zijn aangepast voor je PC's.

Als je klaar bent, uploader je de code als zip-bestand naar "Aankondigingen → Files → Student Exams".
Vergeet **NIET** dat een **juiste naamgevingsconventie** volgen essentieel is;
bijvoorbeeld: "Exam06 Yilmaz Mustafa.zip". Gebruik geen extra underscore, speciaal karakters of iets anders.

Zorgen ervoor dat jullie het mark-down-bestand (***.md) aan de zip hebben toegevoegd.
Bovendien, vergeten dan niet om
een wachtwoord in te stellen voor het zip-bestand. Proberen het minst 4 karakters te zetten in jullie wachtwoord.

# RESOURCES:

Intellij Idea heeft al een plug-in waarmee je een markdown-bestand kunt openen en bewerken.
Daarom is het installeren van de Typora-app slechts een optie voor jou.
Als je al vertrouwd bent met het werken aan markdown-bestanden via Intellij,
hoeft je deze niet te installeren.

- Typora Markdown Reference | Markdown Guide

## Het bericht van de ch€f:

> Begin waar je bent. Gebruik wat je hebt. Doe wat je kan.