1.) Wissensfragen

a) Wieso nutzt man prepared Statements in JDBC in bezug auf Sicherheit
   - Um SQL-Injections zu vermeiden.

b) Ein Update Statement will auf eine Tabelle mit einem Trigger zugreifen.
   Fehlermeldung: “Tabelle wird gerade geändert, Trigger bemerkt evtl. nicht was passiert
   “Was kann diese Fehlermeldung bedeuten?
   - Es wird momentan ein Zeilenbasiertert Trigger angewendet.

c) Was ist statisches SQL?
   - Statisches SQL ist, wenn SQL in die Programmiersprache eingebettet wird. Es wird zur Compilerzeit kompiliert.

d) Wofür werden intentionelle Sperren genutzt, nenne ein Beispiel!
   - Um je nach Transaktion die Feinheit der Sperre zu ändern, um performanter zu sein.
   - Zu Performance Aspekten, es lohnt sich mehr, wenn man nicht Alle Tupel sondern einzelne Sperren kann, wenn nicht mehr notwendig ist. Oder die ganze Datenbank oder die ganze Tabelle. Z.B.

e) Wie kann man mit JDBC SQL Injections vermeiden?
   - Mit PreparedStatments. Durch die Set-Methoden wird dies angepasst.

f) Ein Update Statement will auf eine Tabelle mit einem Trigger zugreifen.
   Fehlermeldung: "Tabelle wird geändert, Trigger bemerkt Änderung evtl. nicht."
   Auf was weist die Fehlermeldung hin?
   ---

g) Kann statisches mit dynamischen SQL gemischt werden? Begründen Sie.
   - Ja, in Pro*C wird dies gemacht.
   - Bei statischem SQL übersetzt der spezielle Compiler das statische SQL in dynamisches SQL.

h) Wie viele Relationen gibt es bei der Vererbungsstrategie Universalrelation pro Vererbungshierarchie? Nennen Sie jeweils 2 Vor- und Nachteile dieser Vererbungsstrategie.
   - Es entsteht nur eine Relation.
   - VT:
     - Performance Aspekte, es kann sehr schnell auf alle Daten zugegriffen werden und man braucht auch keine Joins
     - Die Tabelle kann Mehrfachvererbt werden
   - NT:
     - Viele NULL-Werte
     - Keine NULL-Constrains moeglich

2.) SQL Queries (15 Punkte)

a.)

```
CREATE TABLE teams(
      team_id number primary key,
      team_name varchar
      liga_name varchar,
      Punkte number,
      Tordifferenz number
)
```

```sql
--a)    Wählen Sie alle Teams mit einer Punktzahl über 40 aus
SELECT *
FROM teams
WHERE punkte > 40;

--b)    Wählen Sie die Anzahl der Teams pro Liga und den Namen der Lega aus
SELECT liga_name, count(*) as Anzahl
FROM teams
GROUP BY liga_name;

--c)    Wählen Sie alle Teams
SELECT *
FROM teams;

--d)    Wählen Sie Teamname, Liganame, Punkte pro Team und durchschnittliche Punkte pro Liga (verwenden Sie
analytische Funktionen)
SELECT team_name, liga_name, punkte, AVG(punkte) OVER (PARTITION BY liga_name) AS PUNKTE_PRO_LIGA
FROM teams;
```

```
b.) CREATE TABLE Zootiere(
          tier_id number(8) primary key name varchar(20)
          art varchar(20)
          gewicht number(8) //in kg
```

```sql
-- a)   Geben Sie alle Tiernamen aus die ein Gewicht von mehr als 100kg haben
SELECT name
FROM Zootiere
WHERE gewicht > 100;

-- b)   Geben Sie die Tierarten an bei denen das Gewicht durchschnittlich über 100kg ist
SELECT art
FROM Zootiere
GROUP BY art
HAVING AVG(gewicht) > 100;

-- c)   Geben Sie alle Tiernamen aus, bei denen die Tierart durchschnittlich mehr als 100kg wiegt  (Ohne
Analytische Funktion!)
SELECT name
FROM Zootiere z JOIN (SELECT art, AVG(gewicht) as avgg
                  FROM Zootiere
                  GROUP BY art) x ON (z.art = x.art)
WHERE x.avgg > 100;


-- d)   Geben Sie alle Tiernamen aus, bei denen die Tierart durchschnittlich mehr als 100kg wiegt  (mit
Analytische Funktion!)
SELECT z.name
FROM    (SELECT name, AVG(gewicht) OVER (PARTITION BY art) as x
        FROM Zootiere z) z
WHERE z.x >100;
```

2.1 Analytische Funktionen

```
a) ROW_NUMBER()     OVER (ORDER BY WERT1)
b) RANK()           OVER (ORDER BY WERT1)
c) DENSE_RANK()     OVER (ORDER BY WERT1)
d) SUM(WERT2)       OVER (PARTITION BY WERT1)
e) SUM(WERT2)       OVER (PARTITION BY WERT1 ORDER BY ID)
```

ID	WERT1	WERT2	a.)	b.)	c.)	d.)	e.)
1	3	2	4	4	3	2	2
2	5	8	9	9	5	12	8
3	4	10	5	5	4	34	10
4	1	1	1	1	1	1	1
5	2	8	2	2	2	16	8
6	5	4	10	9	5	12	12
7	4	9	6	5	4	34	19
8	4	9	7	5	4	34	28
9	4	6	8	5	4	34	34
10	2	8	3	2	2	16	16

3.) JDBC (15 Punkte)

a.)

- Der Bonus darf nicht negativ sein und auch nicht 50% des Gehaltes übersteigen. Denken Sie an Fehlerbehandlung und werfen Sie exceptions wo nötig
- Denken Sie an das Schließen von Ressourcen
- Es herrscht mehrfachbetrieb
- Die Datenbank ist auf Read-commited eingestellt
- Für die Transaktionssteuerung gibt es das statische Connection conn Element.

```
create table mitarbeiter (
      id number primary key,
      name, varchar,
      gehalt number,
      bonus number
)
```

Gesucht wird eine JDBC Funktion
`void bestimme_bonus(int mitarbeiter_id, double bonus)`

```java
public void bestimmte_bonus(int m_id, double bonus) {
    boolean ok = false;
    String SQL = "SELECT gehalt FROM mitarbeiter WHERE id = ?";
    double gehalt;
    try {
        try (PreparedStatement stmt = conn.prepareStatement(SQL)) {
            stmt.setLong(1, m_id);
            try (ResultSet rs = stmt.executeQuery()) {
                rs.next();
                gehalt = rs.getDouble("gehalt");
            }
            if( bonus < 0 && (gehalt*0.5) > bonus ) {
                throw new SQLException("Exception message");
            }
        }

        SQL = "UPDATE mitarbeiter SET bonus = ? WHERE m_id = ?";
        try (PreparedStatement stmt = conn.prepareStatement(SQL)) {
            stmt.setString(1, bonus);
            stmt.setLong(2, m_id);
        }
        conn.commit();
        ok = true;
    } finally {
        if (!ok)
            conn.rollback();
    }
}
```

b.)

- `Kunde (PK-Kunden_id)`
- `Fahrrad(PK-Fahrrad_id, FK-Kunden_id)`
- Sie soll prüfen, ob das Fahrrad verfügbar ist (in Tabelle Fahrrad ist entsprechende Kunde_id=NULL).
- Wenn Fahrrad verfügbar, dann soll der Kunde hinzugefügt werden, sonst soll eine Exception geworfen werden.

Schreibe eine Methode in JDBC die das Ausleihen handelt und Transaktion mit händelt.
`void ausleihen(int fahrrad_id, int kunde_id) {...}`

```java
public void ausleihen(int fahrrad_id, int kunde_id) {
    boolean ok = false;
    String SQL = "SELECT K_id FROM Fahrrad WHERE F_id = ?";
    try {
        try (PreparedStatement stmt = conn.prepareStatement(SQL)) {
            stmt.setLong(1, fahrrad_id);
            try (ResultSet rs = stmt.executeQuery()) {
                rs.next();
                rs.getInt("K_id");

                if (!(rs.wasNull())) {
                    throw new SQLException("Fahrrad ist nicht verfuegbar!");
                }
            }
        }
        SQL = "UPDATE Fahrrad SET FK-K_id = ? WHERE PK-F_id = ?";

        try (PreparedStatement stmt = conn.prepareStatement(SQL)) {
            stmt.setString(1, kunde_id);
            stmt.setLong(2, fahrrad_id);
        }
        conn.commit();
        ok = true;
    } finally {
        if(!ok)
            conn.rollback();
    }
}
```

4.) JPA (20 Punkte)

a.) JAVA CODE MIT PROJEKT, KUNDE UND GESCHÄFTSKUNDE

Fügen Sie nur die nötigen Annotationen ein. Wenn nötig, streichen Sie Zeilen.

```
@Entity
@TABLE(name = "Projekt");
public class Projekt{
      @ID @GeneratedValue
      @Column(name= "test")
      private int id;
      @Column
      private String name;
      @ManyToOne
      private Kunde kunde;
}
@Entity
@TABLE(name = "Kunde");
public class Kunde{
      @ID @GeneratedValue
      @Column
      private int id;
      @Column
      private String name;
      @Column
      @OneToMany(mappedby = „
      private List<Projekt> hatBeauftragt = new ArrayList<Projekt>();
}

public class Geschäftskunde extends Kunde{
      ~~private int id;~~
      @Column(name = „test")
      private String rechtsform;
}
```

- Welche Abbildungsmöglichkeiten gibt es für Klassen auf Entitäten?
    - Horizontale Vererbung
    - Vertikale Vererbung
    - Universal Vererbung

- Was sind "Detached" Objekte? Wozu sind die gut?
    - Detached Objekte sind Objekte die nicht mehr mit der Datenbank verbunden sind.
    - Die sind gut um die in der API darzustellen. Es macht keinen Sinn wenn wir die Objekte verbunden lassen, da die sonst niemand mehr bearbeiten kann.

b.) Gegeben sind 2 Entities in einem ER-Diagramm.

Kunde <-1----------0...n-> Journey

- int ID — int ID
- string name — string name
- <nochwas> — <nochwas>

- Schreiben Sie den Java Code mit allen nötigen Annotationen für JPA, damit das ER-Diagramm so genau wie möglich abgebildet wird.
- Methoden können vernachlässigt werden
- Alle Entitäts-Tabellen sollen in der Datenbank mit "E" beginnen.
- Alle Attribute sollen in der Datenbank mit "A" beginnen.

```java
@Entity
@Table(name = "E-Kunde")
public class Kunde{

    @Id @GeneratedValue @Column(name = "A-K_ID")
    private int id;

    @Column(name = "A-name")
    private string name;

    @OneToMany(mappedBy = "kunde") @Column(name = "A_journeys")
    private Set<Journy> journeys = new HashSet<Journy>();

}

@Entity
@Table(name = "E-Journey")
public class Journey{

    @Id @GeneratedValue @Colunm(name = "A-J_ID")
    private int id;

    @ManyToOne @Colunm(A-name)
    string name;
}
```

c) Gegebenes UML Diagramm
Klassen:

- Politiker (Name: string)
- Gremium (Gremienname: string)
- Partei (Parteiname: string)

Beziehungen:
Zwischen Politiker und Gremium 0..* jeweils beidseitig Zwischen Politiker und Partei: Politiker 0..* - 1 Partei

```
@Enitity
@Table(name = "politiker")
public class Politiker{
    @Id @GeneratedValue @Column(name = "dd")
    private int id;

    @Column(name = "name")
    private string name;

    @ManyToMany(mappedBy "politiker")
    Set<Gremium> gremien = new HashSet<Gremium>();

    @ManyToOne
    Patrei pratei = new Partei();

}

@Enitity
@Table(name = "Gremium")
public class Gremium{
    @id @GeneratedValue @Column(name = "id")
    private int id;

    @ManyToMany
    Set<Politiker> politiker = new HashSet<Politiker>();
}

@Enitity
@Table(name = "Partei")
public class Partei{
    @id @GeneratedValue @Column(name = "id")
    private int id;

    @OneToMany(mappedBy = "partei")
    Set<Poltiker> pol = new HashSet<Poltiker>();
}
```

- Schreiben sie eine JPA Methode um einen neuen Poltiker hinzuzufügen

```
void neuerPolitiker(String Poltikername,String Parteiname){
    EntityManager em = emf.getEntityManager();
    EntityTransaction tx = em.getTransaction();
    try {
        tx.begin();
        Politiker pol = new Politiker();
        pol.setName(PolitikerName);
        pol.getPartei().setName(Parteiname);
        em.persist(pol);
        tx.commit();
    } finally {
        if (tx.isActive()) tx.rollback();
        em.close();
    }
}
```

5.) Transaktion (20 Punkte)

a.) Gegeben sind zwei Schedules mit je zwei bzw. drei Transaktionen

- Sind die Schedules Konfliktserialisierbar? Begründen Sie mit einem Abhängigkeitsgraphen.
- Sie die Schedules Seriell? Begründen Sie kurz

- Geben Sie den Ablauf der Schedules an, wenn sie ein Striktes Zwei Phasen Sperrprotokoll anwenden
- Was ist ein Lost Update und wieso wird es durch serialisierbarkeit verhindert?
    - Ein Lost Update passiert, wenn sich Transaktionen ineinander verschachteln.
    - Beide Transaktionen lesen R(a) und während der andere a überschreibt, überschreibt die andere Transaktion auch direkt a ohne das Update des vorherigen Überschreibens mitbekommen zu haben, da er vorher schon gelesen hat. Und so wird ein Update verworfen.
    - Dies kann man verhindern, indem der Schedules serialisierbar ist. Denn dies bedeutet das das Ergebnis des Schedules das gleiche ist, wie wenn der Schedules seriell ist und nicht verschachtelt ist.

---

b.) Gegeben ist ein Schedule mit 3 Transaktionen.

- Prüfen Sie ob der Schedule konflikt-serialisierbar ist.
  (Zeichnen Sie den Abhängigkeitsgraphen.)

- Füllen Sie die Tabelle für das Zeitstempelverfahren mit dem gegebenen Schedule durch.
  (Klappt das? Begründen.)

- Füllen Sie die Tabelle für das strikte 2PL Protokoll mit Mehrfachmodi-Sperren durch.
  (Klappt das? Begründen.)

- Gegeben ist folgender Schedule.
  Zeichnen Sie ein an welchen Stellen Aborts/Commits eingefügt werden müssen, um
    a) einen kaskadierenden Abbruch zu verursachen
    b) den Schedule nicht rücksetzbar zu machen

6.) Trigger (10 Punkte)

a.)

a) Beantworten sie kurz und präzise was der Unterschied zwischen einem Zeilentrigger und einem Statement-trigger ist
   - Ein Zeilenbasierte Trigger greift auf jede Zeile der Tabelle zu mittels einer Schleife, dazu kann er auch auf :old und :new zugreifen.
   - Eine Statement-Trigger greift nur ein mal auf die Tabelle zu und kann nicht auf :old und :new zugreifen.

b) Wenn zwei Trigger auf das gleiche Ereignis ausgelöst werden: In welcher Reihenfolge werden die Trigger abgearbeitet? Wie wirkt sich das auf das Programm aus?
   - Die Reihenfolge ist nicht deterministisch. Daher kann man nicht wirklich sagen wie sich dies auf das Programm auswirkt. Aber das Programm sollte so geschrieben worden sein, dass dies keine Auswirkung hat.

---

b.) Erklären sie was hierbei genau passiert!

```
CREATE OR REPLACE TRIGGER trg_projekt
AFTER DELETE OR UPDATE OF projektnr ON Projekte_Angestellte
DECLARE anzAngest NUMBER;
BEGIN
     SELECT min(cnt) INTO anzAngest FROM
          (SELECT COUNT(*) cnt, abtnr FROM Projekte_Angestellte GROUP BY
projektnr);
     IF (anzAngest < 2) THEN
          RAISE_APPLICATION_ERROR(num => -20000,
               msg => 'Angest2Min verletzt!'
          );
     END IF;
END;
```

- Es wird geprueft ob ein Projekt mindestens 2 Angestellte hat, falls dies nicht der Fall ist wird eine Error ausgeloest.

7.) Sicherheit (10)

a.) Geben Sie die GRANT Befehle für die Situationen.
Wenn ein GRANT nicht genügt, geben Sie eine Lösungsskizze an

a) Die Rolle Mitarbeiter soll die Tabelle Bestellung Lesen können
   - `GRANT SELECT ON BESTELLUNG TO MITARBEITER`

b) Die Rolle Mitarbeiter soll die Tabelle X lesen können, aber ohne die Spalte Y
   - `CREATE VIEW VIEW_BESTELLUNG AS (...)`
   - `GRANT SELECT ON VIEW_BESTELLUNG TO MITARBEITER`

c) Welche Zwei Anmeldemöglichkeiten gibt es bei einer Datenbank? Nennen Sie je einen Vorteil
   - Username-Passwort: VT: - Einfach zu merken
   - Zertifikate VT: - deutlich Sichereres Verfahren

---

b.) Geben Sie den Befehl an, um folgende SQL-Berechtigungen zu ermöglichen.
Falls es keinen geeigneten Befehl gibt, skizzieren Sie einen Lösungsweg.

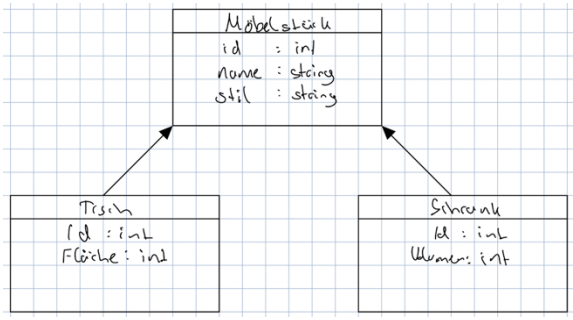
a) Die Rolle Mitarbeiter soll Leserechte auf die Tabelle Angestellte haben.
   - `GRANT SELECT ON ANGESTELLTE TO MITARBEITER`

b) Die Rolle Mitarbeiter soll alle Spalten außer der Spalte Gehalt lesen können.
   - `CREATE VIEW VIEW_A AS (...)`
   - `GRANT SELECT ON VIEW_A TO ANGESTELLTE`

c) Die Rolle Mitarbeiter soll alle Aufträge updaten dürfen, außer Aufträge, die im Status "Abgeschlossen" sind.
   - `CREATE OR REPLACE PROCEDURE A_UPDATE …`
   - `GRANT EXECUTE ON A_UPDATE TO MITARBEITER`

Vertikale & Horizontale Partitionierung

a) Wandeln sie Das UML-Diagramm mit Vertikaler Partitionierung um und schreiben sie die Create table anweisungen

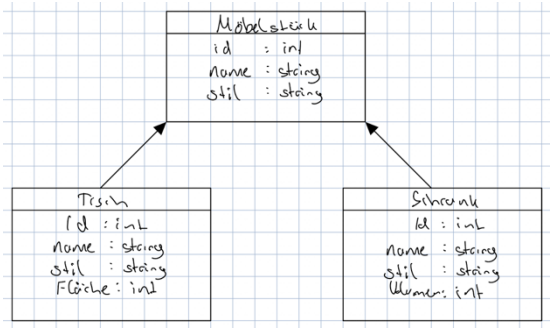


```
CREATE TABLE Moebelstueck (
    name VARCHAR(100),
    stil VARCHAR(100),
    id NUMBER primary key
);

CREATE TABLE Tisch(
    id NUMBER primary key,
    flaeche NUMBER,
    foreign key (id) references Moebelstueck(id)
);

CREATE TABLE Schrank(
    id NUMBER primary key,
    volumen NUMBER,
    foreign key (id) references Moebelstueck(id)
);
```

b) Wandeln sie Das UML-Diagramm mit Horizontaler Partitionierung um und schreiben sie die Create table anweisungen



```
CREATE TABLE Moebelstueck (
    id NUMBER primary key,
    name VARCHAR(100),
    stil VARCHAR(100)
);

CREATE TABLE Tisch(
    id NUMBER primary key,
    name VARCHAR(100),
    stil VARCHAR(100),
    flaeche NUMBER
);

CREATE TABLE Schrank(
    id NUMBER primary key,
    name VARCHAR(100),
    stil VARCHAR(100),
    volumen NUMBER
);
```