Probeklausur Datenbanksysteme Q3/2020 (Stoff des 1. Semesters)

Hinweis: Bei den folgenden Aufgaben handelt es sich um Beispielaufgaben, mit denen der Inhalt geübt und wiederholt werden kann. Es ist kein Rückschluss auf eventuelle Klausuraufgaben möglich, insbesondere kein Schluss darauf, dass nicht genannte Themen nicht in der Klausur vorkommen oder dass genannte Themen nur in dieser Form abgefragt werden können. Insbesondere können sich Schwerpunkte verschieben.

1. Allgemeine Fragen zu Datenbanken

Wie kann man in einer Tabelle für eine bestimmte Spalte zählen, wie oft der Wert NULL in dieser Spalte steht? (3 Punkte)

Was versteht man unter "Löschweitergabe"? Wie wird dies bei SQL umgesetzt? (3 Punkte)

In welchen Fällen müssen Sie Left oder Right Joins verwenden? Konstruieren Sie ein Beispiel. (3 Punkte)

Was sind rekursive Beziehungen in einem ER-Modell? Nennen Sie praktische Beispiele. (3 Punkte)

Grenzen Sie die drei Begriffe "Datenbank", "Datenbank-Management-System" und "Datenbanksystem" voneinander ab. Was bedeuten sie jeweils? (3 Punkte)

2. ER-Modellierung

Ihr Unternehmen entwickelt eine Datenbank zur nächsten Fußball-Weltmeisterschaft. Folgende fachliche Informationen wurden dazu bereits zusammengetragen:

Die Spiele werden in verschiedenen Stadien ausgetragen, die einen Namen besitzen, an einem Ort stehen und eine gewisse Zuschauerkapazität haben. In jedem Stadion werden mehrere Spiele ausgetragen. Die Spiele sind durchnummeriert. Sie finden jeweils zu einem bestimmten Datum zu einer bestimmten Zeit statt. An jedem Spiel nehmen zwei Mannschaften teil. Jede Mannschaft vertritt eine andere Nation und hat einen Rang in der FIFA-Weltrangliste. Jeder Spieler einer Mannschaft hat einen eindeutigen Namen, eine Position und eine Rückennummer und darf nur für eine Nation antreten. Jedes Spiel wird von einem Schiedsrichter geleitet, der einen eindeutigen Namen besitzt und aus einem anderen Land als die beiden teilnehmenden Mannschaften kommt. Ein Schiedsrichter kann mehrere Spiele leiten. Zuschauer werden aus Sicherheitsgründen schon beim Ticketkauf mit einer ID versehen und einer Gefährdungskategorie zugeordnet. Sie können mehrere Spiele besuchen und haben für jedes Spiel einen individuellen Eintrittspreis bezahlt. Es dürfen nicht mehr Zuschauer Spiele besuchen, als die Zuschauerkapazität des jeweiligen Stadions erlaubt.

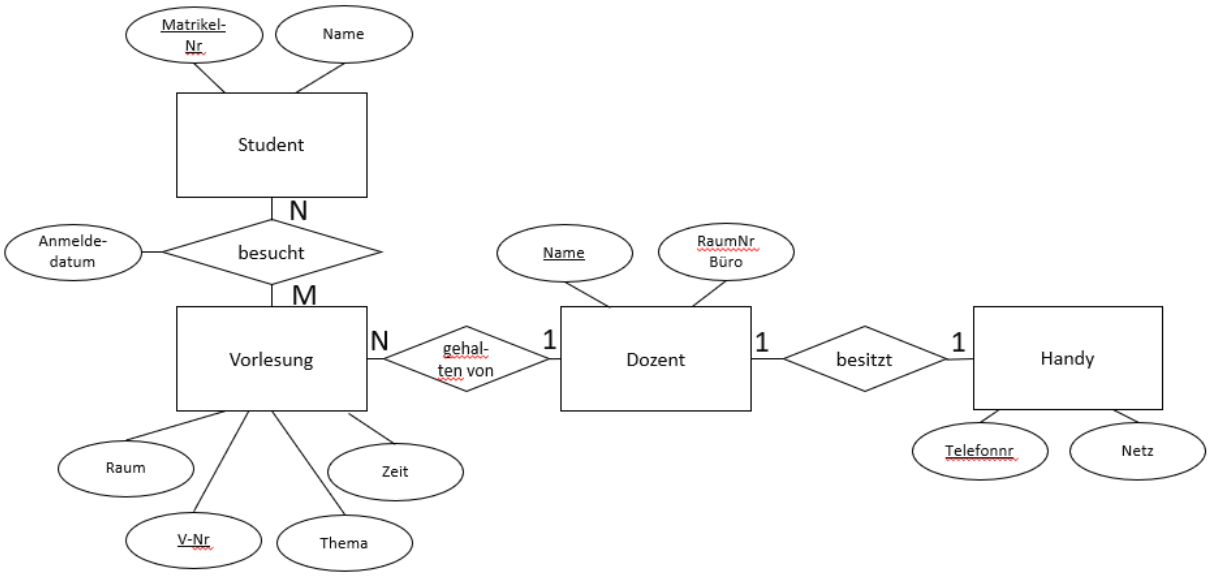
2.a) In welcher Phase des Datenbankentwurfs werden Entity-Relationship-Modelle erstellt? (1 Punkt)

2.b) Welche dieser fachlichen Informationen können Sie in einem ER-Modell nicht sinnvoll abbilden? Nennen Sie zwei Beispiele und begründen Sie, warum das so ist. (9 Punkte)

2.c) Erstellen Sie auf Grundlage der abbildbaren Informationen ein grafisches ER-Modell in der bekannten Notation. Versuchen Sie, alle gegebenen Informationen möglichst vollständig zu erfassen. Geben Sie insbesondere alle Attribute an. Wählen Sie möglichst fachlich motivierte Schlüsselattribute für die Entitäten und unterstreichen Sie sie im Modell. Versehen Sie alle Beziehungen mit Kardinalitäten. (25 Punkte)

3. Abbildung auf Relationenmodell

Gegeben sei das folgende ER-Diagramm:



Leiten Sie aus dem Diagramm Tabellen ab und markieren Sie einen Schlüsselkandidaten in jeder Tabelle. Vermeiden Sie die Ableitung überflüssiger Einzeltabellen. (25 Punkte)

4. Relationen und Normalformen

Die Autovermietung "Elmshorn-Cars" vermerkt ihre Buchungen in der folgenden kleinen Tabelle:

BuchungsNr	KundenNr	Mietbeginn	Name	Rabatt	Kennzeichen	Autotyp	Tage
78	12	04.08.2020	Straßberger	Ja	PI-NA 123	VW Polo	2
79	21	06.08.2020	Meier AG	Ja	PI-NA 911	MB Sprinter	4
79	21	06.08.2020	Meier AG	Ja	PI-NA 912	MB Sprinter	4
80	79	06.08.2020	Schubert	Nein	PI-NA 456	Smart	14

4.a) Zwischen den Attributen der Tabelle gibt es folgende vollständig funktionalen Abhängigkeiten:

{BuchungsNr} → {KundenNr, Mietbeginn, Name, Rabatt}
{KundenNr} → {Name, Rabatt}
{Kennzeichen} → {Autotyp}
{BuchungsNr, Kennzeichen} → {AnzahlTage}

Identifizieren sie die Schlüsselkandidaten der Tabelle. (1 Punkte)

4.b) Erläutern sie, ob sich die Tabelle in der ersten Normalform (1NF) befindet. Falls nicht, überführen sie die Tabelle in die 1NF. Nehmen sie dabei nur die unbedingt notwendigen Veränderungen vor. Markieren sie die Schlüsselkandidaten der Tabellen. (3 Punkte)

4.c) Erläutern sie, ob sich die Tabelle in der zweiten Normalform (2NF) befindet. Falls nicht, überführen sie die Tabelle in die 2NF. Nehmen sie dabei nur die unbedingt notwendigen Veränderungen vor. Markieren sie die Schlüsselkandidaten der Tabellen. (3 Punkte)

4.d) Erläutern sie, ob sich die Tabelle in der dritten Normalform (3NF) befindet. Falls nicht, überführen sie die Tabelle in die 3NF. Nehmen sie dabei nur die unbedingt notwendigen Veränderungen vor. Markieren sie die Schlüsselkandidaten der Tabellen. (3 Punkte)

5. SQL-Anfragen

Gegeben seien folgende Tabellen zur Studentenverwaltung (Schlüssel sind unterstrichen, leere Felder stehen für NULL-Werte):

Firma

Name	Betreuer
Super	Mai
Klasse	
Doll	Juni

Student

MatNr	Name	Firma	Jahrgang
1	Ute	Super	01
2	Udo	Super	02
3	Uwe	Klasse	02

Pruefung

MatNr	Fach	Note
1	DB	1.0
1	BWL	5.0
2	DB	5.0

Formulieren sie die folgenden Textzeilen jeweils als SQL-Anfragen.

5.a) Geben sie die Fächer aus, in denen schon mindestens zwei Prüfungen stattgefunden haben. (3 Punkte)

5.b) Geben sie die Namen der Studenten aus, die zurzeit keinen Betreuer in ihrer Firma haben. (3 Punkte)

5.c) Geben sie für jede Firma die Anzahl der zurzeit eingetragenen Studenten an (Ausgabe: Name der Firma, Anzahl). (3 Punkte)

5.d) Geben sie für jede Firma und jedes Prüfungsfach die aktuelle Durchschnittsnote (der Studenten dieser Firma in diesem Fach) an, insofern schon mindestens zwei Studenten dieser Firma eine Prüfung in diesem Fach abgelegt haben. (Ausgabe: Name der Firma, Name des Fachs, Durchschnittsnote). (3 Punkte)

5.e) Geben sie die Namen aller Studierenden aus, die noch nie durchgefallen sind und schon mindestens eine Prüfung absolviert haben. (3 Punkte)

SQL-Kurzreferenz

Select-Anweisungen

```
SELECT [DISTINCT] {* | Spalte1
[ [AS ] "Alias" ], ... }
FROM Tabellenname;
```

Arithmetische Operatoren
```
SELECT Spalte1, Spalte2 + Wert
FROM Tabellenname;
```

Alias Definition
```
SELECT Spalte1 [AS] Alias,
FROM Tabellenname  TabAlias;
```

Bedingungen mit WHERE

```
SELECT [DISTINCT] {* | Spalte [ [AS ]
" Alias ], ... } FROM Tabelle
[ WHERE Bedingung(en) ] ;
```

Mögliche Operatoren
```
=; >; >=; <; <=; <>;
IS NULL; IS NOT NULL;
BETWEEN ... AND ... ;
IN (Liste); LIKE
```

Verwendung
```
WHERE Spalte IS NULL
WHERE Spalte BETWEEN ... AND ...
WHERE Spalte IN (Eintr.1, Eintr.2,...)
WHERE Spalte LIKE '[%][_]String[%][_]'
% beliebig viele Zeichen (auch null)
_ ein beliebiges Zeichen
```

Logische Operatoren (AND, OR, NOT)
```
WHERE Bedingung1 AND Bedingung2
WHERE Spaltenname NOT IN (Liste)
Reihenfolge der Wichtigkeit: Klammern;
Vergleichsoperatoren; NOT; AND; OR
```

Sortierung mit ORDER BY
```
SELECT * FROM Tabellenname
[ WHERE Bedingung(en) ]
[ ORDER BY { Spaltenname | Ausdruck |
Aliasname [ ASC | DESC ] } ]
Aufsteigend (asc)(Default)
Absteigend (desc)
```

JOINS

Equijoin
```
SELECT {Alias1.Spalte1,
Alias1.Spalte2, Alias2.Spalte1, ...}
FROM Tab1 Alias1, Tab2 Alias2, ...
WHERE Alias1.Spalte1 = Alias2.Spalte1
[AND Alias2.Spalte2 = Alias3.Spalte1];
```

Alternativ:
```
SELECT {Alias1.Spalte1,
Alias1.Spalte2, Alias2.Spalte1, ...}
FROM (Tab1 Alias1 INNER JOIN Tab2
Alias2 ON Alias1.Spalte1 =
Alias2.Spalte1)
INNER JOIN Tab3 Alias3
ON Alias2.Spalte2 = Alias3.Spalte1
WHERE (...);
```

Outer-Join

```
SELECT {Alias1.Spalte1,Alias1.Spalte2,
Alias2.Spalte1, ...}
FROM (Tab1 Alias1 {LEFT|RIGHT|FULL|
OUTER} JOIN Tab2 Alias2
ON Alias1.Spalte1 = Alias2.Spalte2)
WHERE (...);
```

LEFT JOIN orientiert sich an der Tabelle 1 und ergänzt fehlende Informationen mit NULL-Datensätzen der Tabelle 2.

RIGHT JOIN orientiert sich an der Tabelle 2 und ergänzt fehlende Informationen mit NULL-Datensätzen der Tabelle1.

Self-Join

```
SELECT {Alias1.Spalte1,
Alias1.Spalte2, Alias2.Spalte1, ...}
FROM Tab1 Alias1, Tab1 Alias2
WHERE Alias1.Spalte1 = Alias2.Spalte2;
```

```
alternativ:
SELECT {Alias1.Spalte1,
Alias1.Spalte2, Alias2.Spalte1, ...}
FROM (Tab1 Alias1 INNER JOIN Tab1
Alias2 ON Alias1.Spalte1 =
Alias2.Spalte2) WHERE (...);
```

Gruppenfunktionen

```
SELECT Gruppenfkt.(Spaltenname), ...
FROM Tabelle [WHERE Bedingung(en)]
[ORDER BY {Spaltenname|Ausdruck|
Aliasname} [ASC|DESC] ];

AVG (Spaltenname) : Durchschnitte
SUM (Spaltenname) : Summe
MIN (Spaltenname) : Minimum
MAX (Spaltenname) : Maximum
COUNT (Spaltenname) : Anzahl
NULL-Werte werden von den Funktionen
nicht berücksichtigt

COUNT (*) (Zählt Zeilen mit NULL mit)
```

Datengruppen mit GROUP BY
```
SELECT Spalte1,
Gruppenfunktion(Spalte2), ...
FROM Tabelle
[ WHERE Bedingung(en) ]
[ GROUP BY Spaltenname1 [, ...] ]
[ HAVING Gruppenbedingung ]
[ ORDER BY {Spaltenname1 | Ausdruck |
Aliasname} [ ASC | DESC ] ];

HAVING dient der Einschränkung
Gruppenergebnisse ein.
```

Unterabfragen

SELECT-Unterabfragen
```
SELECT Spalten FROM Tabelle
WHERE Spaltenname Operation
(Select-Statement) [ AND ... ];
```

Select darf nur einen Wert als Vergleichswert zurückliefern. Unterabfragen, die mehrere Werte zurückliefern müssen die Operatoren IN; ANY; ALL; EXISTS verwenden.

Beispiel:
```
SELECT A.A_NR FROM ARTIKEL As A
WHERE EXISTS
  (SELECT B.UMSATZ_NR FROM UMSATZ As B
       WHERE B.A_NR = A.A_NR)
```

Beispiel ALL / ANY:
```
SELECT * FROM Waggons
       WHERE waggon_id < [ALL|ANY]
  (SELECT waggon_id FROM Kunden);
```
Alle ids aus Kunden müssen größer als waggon_id sein. Bei ANY muss Übereinstimmung nicht bei allen Elementen der Ergebnismenge vorliegen.

UPDATE Unterabfragen
```
UPDATE Tabelle Alias SET Spalte =
(SELECT expr FROM Tabelle alias2
WHERE Alias.Spalte = "5"
```

DELETE Unterabfragen
```
DELETE FROM Tab1 Alias1 WHERE Spalte
Operator (SELECT expr FROM Tab)
```

Mengenoperationen

Anzahl und Typ der SELECT-Anweisungen müssen übereinstimmen.

Vereinigung
```
SELECT Spalten FROM Tabelle
[WHERE Bedingung(en)]
UNION SELECT Spalten
FROM Tabelle [WHERE Bedingung(en)]
```

Durchschnitt
```
SELECT Spalten FROM Tabelle
[WHERE Bedingung(en)]
INTERSECT SELECT Spalten FROM Tabelle
[WHERE Bedingung(en)] ;
```

Differenz
```
SELECT Spalten FROM Tabelle
[WHERE Bedingung(en)]
MINUS SELECT Spalten FROM Tabelle
[WHERE Bedingung(en)];
```

Tabelleninhalt bearbeiten

Datensätze einfügen
```
INSERT INTO Tab[(Spalte1,Spalte2,...)]
VALUES (Wert1, "Wert2",...);
```

Datensätze ändern
```
UPDATE Tabelle SET Spalte1 = Wert1,
[Spalte2 = Wert2, ...]
[WHERE Bedingung(en)];
```

Datensätze löschen
```
DELETE FROM Tabelle
[WHERE Bedingung(en)];
```

DDL-Data Definition Language

Datenbank erstellen / löschen
```
CREATE DATABASE datenbankname;
DROP DATABASE datenbankname;
```

Tabelle erstellen
```
CREATE TABLE tabellenname
(spaltenname datentyp [NOT NULL],
 [...,]
 spaltenname datentyp[NOT NULL]);
```

Datentypen: CHAR(n), INT, SMALLINT, NUMBER, FLOAT(n), REAL, DOUBLE PRECISION, DEC(m, [n]), DATE

Tabelle löschen
```
DROP TABLE tabellenname
```

Spalten hinzufügen
```
ALTER TABLE tabellenname
    ADD spalte datentyp [NOT NULL],
    [...,];
```

Spalte löschen
```
ALTER TABLE tabellenname
    DROP (spalte,[... ,], spalte);
```