Aufgabe 8

Ich modelliere zusätzlich zu den Entitäten *Hotel*, *Zimmer* und *Gast* auch die Relationen *IstKunde* und *Reserviert* als eigenständige Tabellen. Die Relation *Hat* wird über Fremdschlüsselbeziehungen modelliert. Das komposite Attribut *Adr* ersetze ich durch die beiden Attribute *Strasse* und *Ort*, es taucht außer im ER-Diagramm nicht mehr in der Datenbank auf.

0:1-Beziehungen habe ich über eindeutige Primärschlüssel modelliert, die aber auch Null sein dürfen (Gast - Bewohnt). Nicht Null ist erlaubt bei 1:1-Beziehungen, die auf einen eindeutigen Fremdschlüssel aufbauen (Zimmer - hat). 0:n basiert darauf, dass die Relation eine eigene Tabelle ist, die die Primärschlüssel der Tabellen, die miteinander verknüpft werden, über Fremdschlüssel als Primärschlüssel einsetzt (Gast - IstKunde), die einzeln nicht eindeutig sein müssen.

Man kann sich darüber streiten, ob es notwendig ist, die Relation *Reserviert* überhaupt einzuführen, da sie ja lediglich einem zukünftigen *Bewohnt* mit 0 Telefoneinheiten entspricht.

In den nachfolgenden Tabellen sind Primärschlüsselattribute grau hinterlegt und unterstrichen, Fremdschlüssel sind lediglich grau.

Hotel	**hid**	**Name**	**Strasse**	**Ort**

Zimmer	**zid**	**Nummer**	**Raucher**	**AnzBetten**	**Preis**	**hid**

Gast	**gid**	**Name**	**Strasse**	**Ort**	**Raucher**

IstKunde	**gid**	**hid**	**Rabatt**

Reserviert	**gid**	**zid**	**Von**	**Bis**

Bewohnt	**Start**	**Ende**	**Tel.Einheiten**	**gid**	**zid**

In den Tabellen ist der Datentyp der einzelnen Attribute nicht ersichtlich, genauso fehlen noch notwendige Bedingungen. Sie finden sich alle in den SQL-Statements wieder (ich habe MySQL-kompatible Datentypen benutzt):

```
CREATE TABLE Hotel (
      hid         INTEGER           NOT NULL,
      Name        CHARACTER(30)     NOT NULL,
      Strasse     CHARACTER(30),
      Ort         CHARACTER(30),

      PRIMARY KEY (hid) );


CREATE TABLE Zimmer (
      zid         INTEGER           NOT NULL,
      Nummer      INTEGER,
      Raucher     BIT,
      AnzBetten   INTEGER,
      Preis       NUMERIC(10,2),
      hid         INTEGER           NOT NULL,
```

```
      PRIMARY KEY (zid),
      FOREIGN KEY (hid) REFERENCES Hotel);


CREATE TABLE Gast (
      gid         INTEGER           NOT NULL,
      Name        CHARACTER(30)     NOT NULL,
      Strasse     CHARACTER(30),
      Ort         CHARACTER(30),
      Raucher     BIT,

      PRIMARY KEY (gid));


CREATE TABLE IstKunde (
      gid         INTEGER           NOT NULL,
      hid         INTEGER           NOT NULL,
      Rabatt      DECIMAL,

      PRIMARY KEY (gid, hid),
      FOREIGN KEY (gid) REFERENCES Gast,
      FOREIGN KEY (hid) REFERENCES Hotel);


CREATE TABLE Reserviert (
      Von         DATE              NOT NULL,
      Bis         DATE              NOT NULL,
      gid         INTEGER           NOT NULL,
      zid         INTEGER           NOT NULL,

      PRIMARY KEY (zid, Von, Bis),
      UNIQUE      (gid),
      FOREIGN KEY (gid) REFERENCES Gast,
      UNIQUE      (zid);
      FOREIGN KEY (zid) REFERENCES Zimmer);


CREATE TABLE Bewohnt (
      Start       DATE,
      Ende        DATE,
      TelEinheiten INTEGER,
      gid         INTEGER           NOT NULL,
      zid         INTEGER           NOT NULL,

      PRIMARY KEY (zid, Start, Ende),
      FOREIGN KEY (gid) REFERENCES Gast,
      FOREIGN KEY (zid) REFERENCES Zimmer);
```

Abschließend noch ein Vergleich der Datentypen:

MySQL	Aufgabenstellung
integer	int
character	char
bit	bool
date	date
numeric	float

Um eine Eindeutigkeit zu wahren, habe ich zusätzlich noch den Operator `UNIQUE` benutzt.

Aufgabe 9

Ich habe erneut alle SQL-Statements in MySQL umgesetzt, um mich von der Funktionsweise überzeugen zu können. Eventuell sind daher Abweichungen zu einer optimalen SQL-92-Lösung möglich.

Mir war es möglich, *alle* Anfragen in MySQL zu formulieren.

a) Aufgrund einer MySQL-Beschränkung für die SQL-Klammerung musste ich auf ein *right outer join* ausweichen, das die Kundentabelle mit dem *natural join* von AKRel und Auftrag verknüpft:

```
SELECT DISTINCT kunde.*, auftrag.auftragnr, bezeichnung
            FROM (akrel NATURAL JOIN auftrag)
                  RIGHT OUTER JOIN kunde ON kunde.kundennr=akrel.kundennr
            ORDER BY name ASC, auftragnr DESC, ort ASC;
```

b) Der LIKE-Operator erlaubt mit dem Platzhalter _ das Auftreten genau eines beliebigen Zeichens an der Stelle. Da es aber möglich ist, dass auch Regal12 existiert, d.h. es auch mehr als 1 Zeichen nach Regal folgen darf, benutze ich %:

```
SELECT DISTINCT lager, lagerplatz, artikelnr
            FROM lagerort
            WHERE lagerplatz LIKE "Regal%" AND lagerplatz<>"Regal1"
            ORDER BY lager, lagerplatz;
```

c) Eine GROUP-Klausel sorgt dafür, dass die Ergebnistabelle nach der Auftragnummer gruppiert wird. Davon interessiert mich jeweils der Artikel mit dem höchsten VK-Preis:

```
SELECT DISTINCT artikel.artikelnr, artikel.bezeichnung, MAX(vk_preis),
                auftrag.auftragnr
            FROM auftrag NATURAL JOIN auftragspositionen
                  NATURAL JOIN artikel
            GROUP BY auftrag.auftragnr;
```

d) Der UPDATE-Befehl verfügt über keine Besonderheiten:

```
UPDATE lagerort SET anzahl=9
            WHERE lager="Nord" AND lagerplatz="Regal1" AND artikelnr=567;
```

e) Das komplette Regal wird gelöscht:

```
DELETE FROM lagerort
            WHERE lager="Nord" AND lagerplatz="Regal4";
```

f) Ich überschreibe die Attribute Lager und Lagerplatz der gewünschten Kiste(n):

```
UPDATE lagerort SET lager="Süd", lagerplatz="Tresen"
            WHERE lager="Nord" AND lagerplatz="Regal3";
```

g) Im reinen SQL sind Sub-Selects möglich, für MySQL führte ich eine temporäre Variable artikelnr_kabelbruecke ein:

SQL:
```
INSERT INTO lagerort
            VALUES ("Süd", "Regal1",
                        (SELECT artikelnr FROM artikel
                                WHERE bezeichnung="Kabelbrücke"), 11);
```

MySQL:
```
SELECT @artikelnr_kabelbruecke := artikelnr FROM artikel
            WHERE bezeichnung="Kabelbrücke";
INSERT INTO lagerort
            VALUES ("Süd", "Regal1", @artikelnr_kabelbruecke, 11);
```