Datamanagement

Hochschule Luzern
Technik & Architektur

TA.BA_DMG

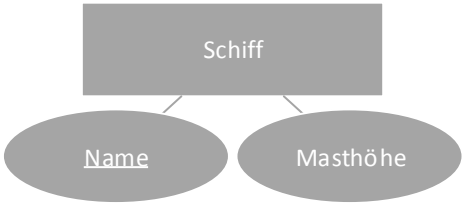
Inhalt

1 Datenbankmodellierung, ERM (ER-Modell)

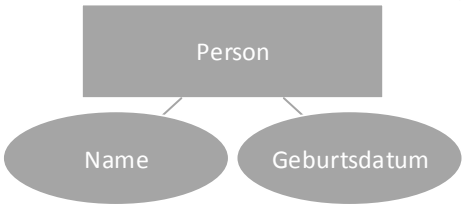
- Datenbankmodellierung
- ERM / ER-Modell

1.1 Entity-Mengen 1

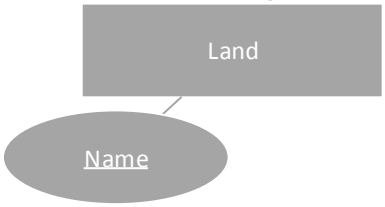
- Schiffe mit **eindeutigem** Namen und Masthöhe



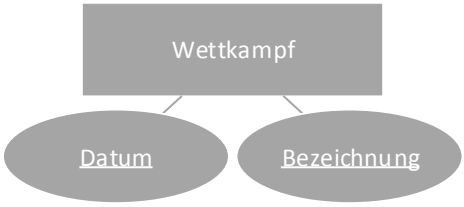
- Personen mit Namen, Geburtsdatum (Schiffseigner oder Mannschaftsmitglieder)



- Länder mit **eindeutigem** Namen



- Wettkämpfe mit Bezeichnung und Datum (**zusammen haben die Attribute Schlüsseleigenschaft**)

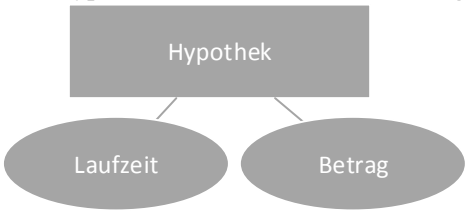


- Mannschaft

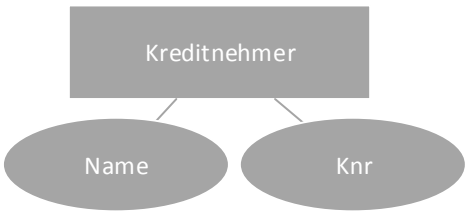
1.2 Entity-Mengen 2

- Banken mit **eindeutiger** Bezeichnung (Bez)

- • Hypotheken mit Laufzeit und Betrag



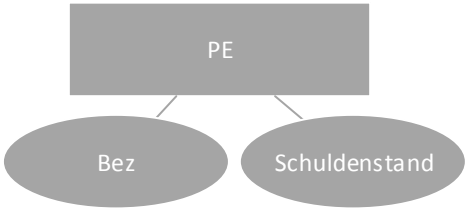
- • Kreditnehmer mit Name und KontoNummer (KNr)



- • Länder mit **eindeutiger** Bezeichnung (Bez), mit Schuldenstand (Schuldenstand wird in PE abgebildet)

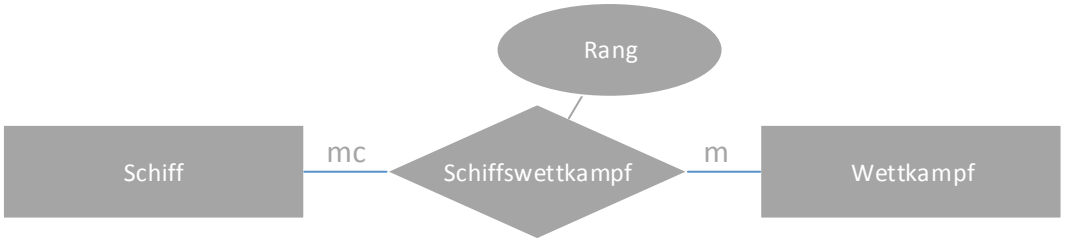
- • Gemeinden mit Bezeichnung (Bez) und Schuldenstand (Schuldenstand wird in PE abgebildet)

- • Politische Einheiten (PE) mit Bezeichnung (Bez) und Schuldenstand

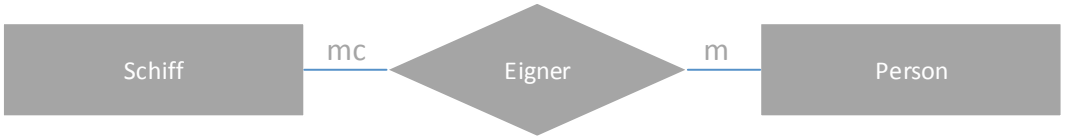


1.3 Beziehungsinformationen 1

- Schiffe **nehmen** an Wettkämpfen **teil** und belegen dort einen Rang



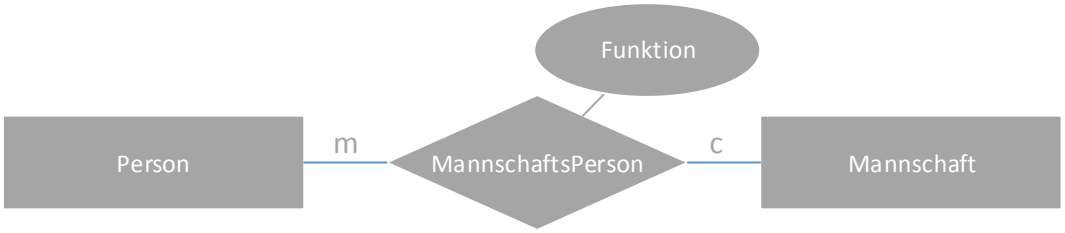
- Ein Schiff hat ein oder **mehrere** Personen als Eigner



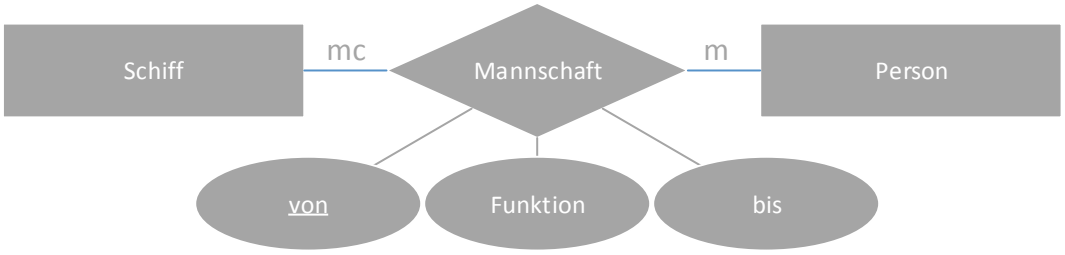
- Ein Schiff hat **eine** Mannschaft
- Eine Mannschaft fährt **genau auf einem** Schiff.

- Ein Schiff fährt für **genau ein** Land

- Personen. Eine Person gehört zu **höchstens einer** Mannschaft.
- Jede Person in einer Mannschaft hat **genau eine** Funktion.

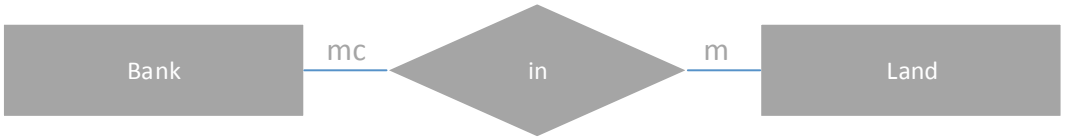


- Eine Person kann im Laufe der Zeit zu **mehreren** Mannschaften gehören. Zu einem **bestimmten Zeitpunkt** gehört sie nur zu **einer einzigen** Mannschaft.
  → Die Relationship Mannschaftsperson hat ausser dem Attribut Funktion noch die Attribute von und bis. Von ist Teilschlüssel. Die c-Beziehung wird zur mc-Beziehung.



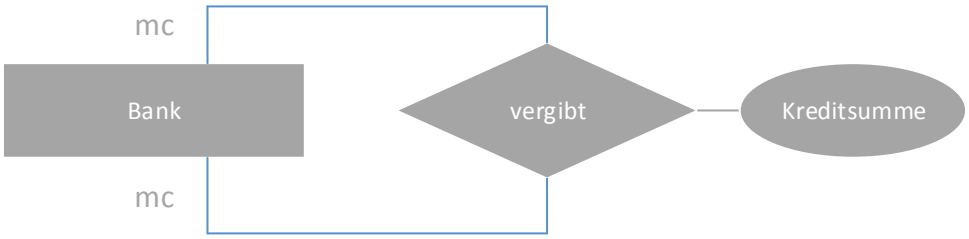
1.4 Beziehungsinformationen 2

- Jede Bank kann in **vielen** Ländern vertreten sein.

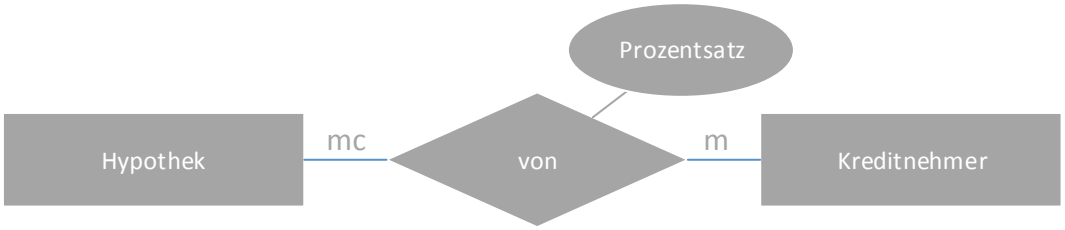


- Jede Bank vergibt **viele** Hypotheken. Jede Hypothek wird von **genau einer** Bank vergeben.

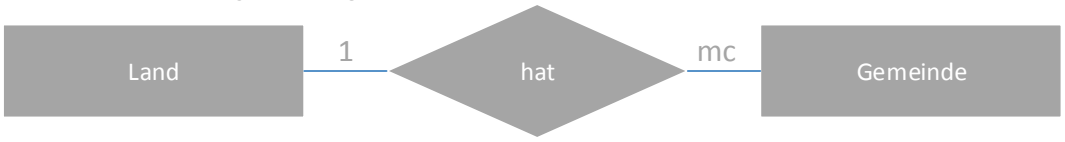
- Banken geben **einander** Kredite. Die **Kreditsumme** von einer Bank an eine andere ist **festzuhalten**



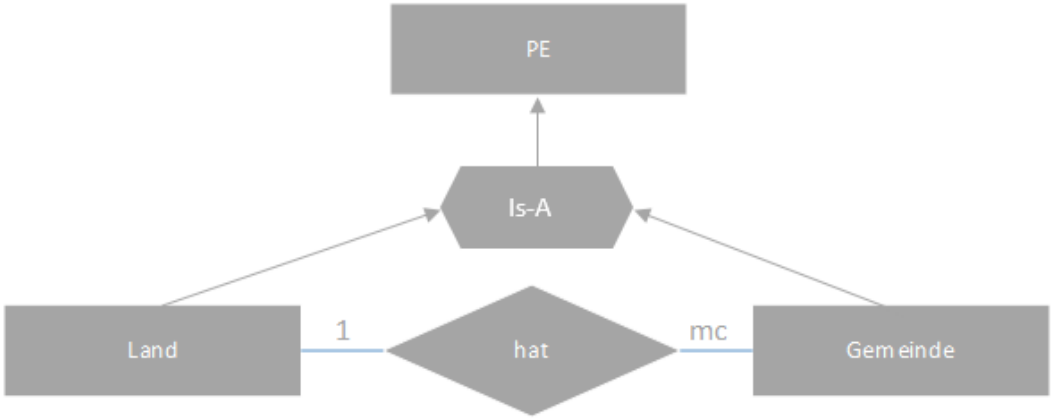
- Jede Hypothek hat **mindestens einen** Kreditnehmer – jeweils mit einem %-Satz für seinen Haftungsanteil.
- Ein Kreditnehmer kann **viele** Hypotheken haben.



- • Jede Gemeinde gehört zu **genau einem** Land.



- • Länder und Gemeinden **sind** „Politische Einheiten“



Aufgabe a)

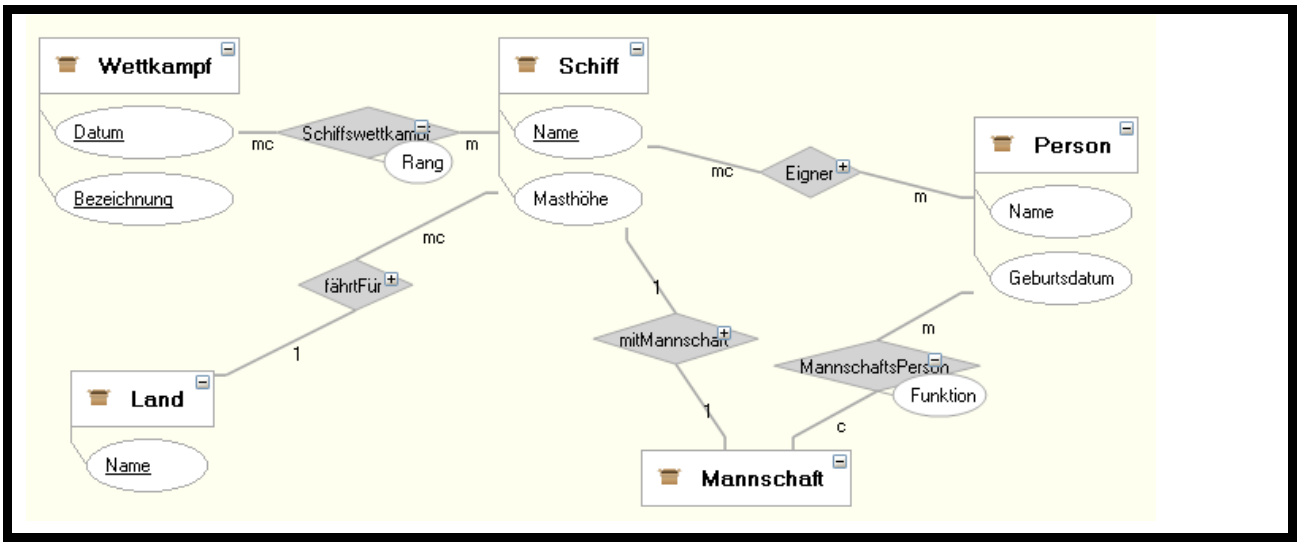
Entity-Mengen sind:

- Schiffe mit eindeutigem Namen und Masthöhe
- Personen mit Namen, Geburtsdatum (Schiffseigner oder Mannschaftsmitglieder)
- Länder mit eindeutigem Namen
- Wettkämpfe mit Bezeichnung und Datum (zusammen haben die Attribute Schlüsseleigenschaft)

Beziehungsinformationen sind:

- Ein Schiff hat ein oder mehrere Personen als Eigner
- Ein Schiff fährt für genau ein Land
- Ein Schiff hat eine Mannschaft – mehrere Personen. Eine Person gehört zu höchstens einer Mannschaft.
- Eine Mannschaft fährt genau auf einem Schiff.
- Jede Person in einer Mannschaft hat genau eine Funktion.
- Schiffe nehmen an Wettkämpfen teil und belegen dort einen Rang

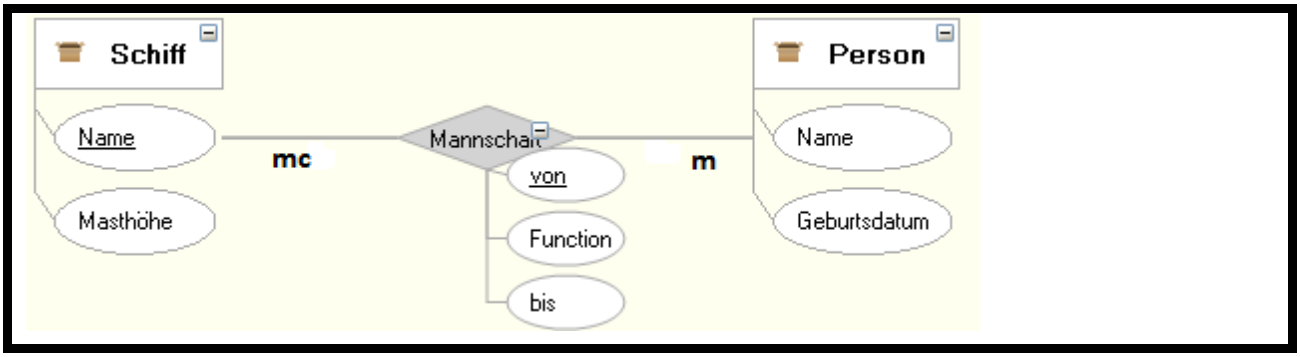
Lösung a)



Aufgabe b)

Die Relationship Mannschaftsperson hat ausser dem Attribut Funktion noch die Attribute von und bis. Von ist Teilschlüssel. Die c-Beziehung wird zur mc-Beziehung.

Lösung b)



2 ER-Modell → relational DB (create Table)

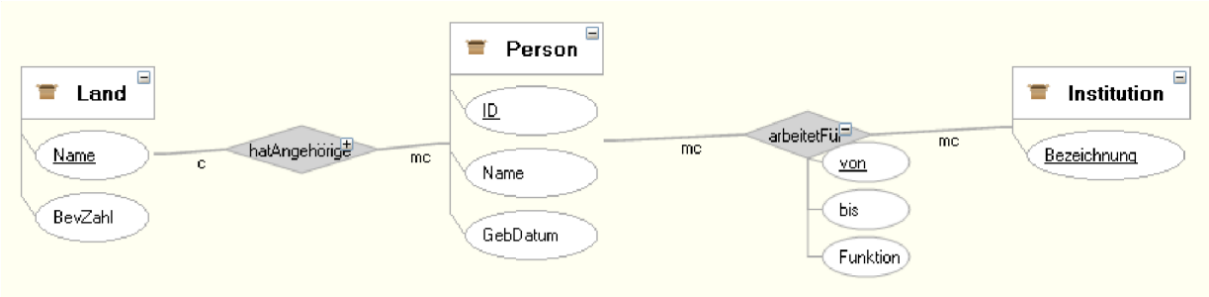
- create table

2.1 Syntax

Folgende Syntax wird angewendet, um eine Tabelle zu erstellen:

```
CREATE TABLE Tabellenname (
       Spaltenname1  Datentyp  Attribute,
       Spaltenname2  Datentyp  Attribute,
       Spaltenname3  Datentyp  Attribute REFERENCES FremdTabellenName ON ...
);
```

2.2 Prüfungsaufgabe



Ich habe unser Tool DataMagus genutzt. Die Relationships hatAngehörige und arbeitetFür gelten sind zwischen Entitymengen – die grafische Darstellung hat noch Optimierungspotenzial. Unterstrichene Attribute haben Schlüsseleigenschaft bzw. sind Teil vom Schüssel („von“ in Relationship arbeitetFür).

Zu den Beziehungstypangaben: Im Buch wird c als 1, mc als M oder N dargestellt

Attribut	**Datentyp**
Name, Bezeichnung, Funktion	VARCHAR(20)
GebDatum, von, bis	DATE
Bev(ölkerungs)Zahl	INT
ID	INT

Anforderungen:

• Jede Tabelle, die eine Entitätsmenge implementiert, soll einen technischen Primärschlüssel (ID) haben. Für Attribute/ Attributkombinationen mit Schlüsseleigenschaft sollen diese angegeben werden.

• Wenn eine Institution gelöscht wird, sollen ihre Beziehungsinformationen zu Person automatisch gelöscht werden. Entsprechendes gilt für die Löschung einer Person.

• Ein Land darf nur gelöscht werden, wenn es nicht referenziert wird.

a) Schreiben Sie die create-table-statements für das obige ER-Modell. Abkürzungen sind erlaubt. (7 Punkte)

b) Erscheint es Ihnen sinnvoll, für jede Tabelle (nicht nur für die Tabellen, die Entitätsmengen implementieren) einen technischen Primärschlüssel zu vergeben? Begründen Sie, bitte. (1 Punkt)

Lösung

```
CREATE TABLE Land (
	ID	INT	PRIMARY KEY	IDENTITY,
	Name	VARCHAR(20)	NOT NULL	UNIQUE,
	BevZahl	INT	NOT NULL
);

-- 1 zu n Beziehung zwischen Person und Land
-- (Eine Person gehört zu einem Land, ein Land hat mehrere Personen)
CREATE TABLE Person (
	ID	INT	PRIMARY KEY	IDENTITY,
	Name	VARCHAR(20)	NOT	NULL ,
	GebDatum	DATE	NOT	NULL
	LandId	INT	REFERENCES	Land	ON	DELETE	NO	ACTION;
);

CREATE TABLE Institution (
	ID	INT	PRIMARY KEY	IDENTITY,
	Bezeichnung	VARCHAR(20)	NOT	NULL	UNIQUE
);

-- n zu n Beziehung zwischen Person und Institution
CREATE TABLE arbeitetFür (
	von	DATE	NOT	NULL ,
	bis	DATE	NOT	NULL
	Funktion	VARCHAR(20)	NOT	NULL ,
	PersonId	INT	NOT	NULL	REFERENCES	Person	ON DELETE CASCADE ,
	InstitutionId	INT	NOT	NULL	REFERENCES	Institution	ON DELETE	CASCADE,
	PRIMARY	KEY	(von, PersonId, InstitutionId)
);
```

2.3 Datentypen

Nachfolgend wurden die wichtigsten Datentypen:

Datentyp	Beschreibung
INT	Speichert eine Zahl mit der Genauigkeit von -2147483648 bis 2147483647 ab.
BIGINT	Speichert eine Nummer mit der Genauigkeit von -9223372036854775808 bis 9223372036854775807 ab.
REAL	Speichert eine Gleitkommazahl mit der Genauigkeit von -3,40E+38 bis 3,40E+38 ab.
FLOAT	Speichert eine Gleitkommazahl mit der Genauigkeit von -1,79E+308 bis 1,79E+308 ab.
DATETIME	Speichert das Datum und die Zeit ab. (Genauigkeit: Millisekunden)
DATE	Speichert den Tag ab. (Genauigkeit: Tag)
CHAR (Anzahl)	Speichert eine Zeichenfolge ab. Im Gegensatz zu VARCHAR werden dabei immer alle Zeichen verwendet. Wenn die Zeichenfolge kleiner ist als die Grösse dieses Feldes, wird es mit Leerzeichen aufgefüllt.
VARCHAR (Anzahl)	Speichert eine Zeichenfolge ab. Mit VARCHAR(MAX) wird die Maximale Zeichenfolgengrösse genommen (2GB).
BINARY	Speichert Binäre Daten ab. Verhalten dabei gleich wie bei *CHAR*.
VARBINARY	Speichert Binäre Daten ab. Verhalten dabei gleich wie bei *VARCHAR*.

2.4 Referenzen

Um Tabellen miteinander zu verbinden wird das Schlüsselwort „REFERENCES“ benötigt. Dabei wird folgende Syntax verwendet.

Spaltenname Datentyp Attribute **REFERENCES FremdtabellenName*(Spaltenname)***
ON DELETE SET NULL ON UPDATE CASCADE

Der Spaltenname, auf den verwiesen wird, ist optional. Falls dieser nicht gesetzt wird, wird auf den Primärschlüssel der Fremdtabelle verwiesen. Weiter sind die ON DELETE und ON UPDATE Einschränkungen optional.

WICHTIG: Die Fremdtabelle muss bereits existieren, bevor mit REFERENCES auf diese verwiesen wird!

Folgende Einschränkungen sind dabei möglich:

Name	Beschreibung
`NO ACTION`	Erlaubt das Löschen bzw. Ändern eines Eintrages nicht, solange ein anderer Eintrag noch darauf verweist.
`CASCADE`	Sobald ein Eintrag gelöscht bzw. aktualisiert wird, werden auch alle Einträge welchen auf diesen Verweisen gelöscht bzw. aktualisiert.
`SET NULL`	Setzt den Wert auf NULL für alle Einträge, welchen auf aktualisierten bzw. gelöschten Eintrag verweisen
`SET Default`	Setzt den Standardwert für alle Einträge, welchen auf aktualisierten bzw. gelöschten Eintrag verweisen

2.5 Attribute

Folgende Attribute sind auf die Spalten anwendbar:

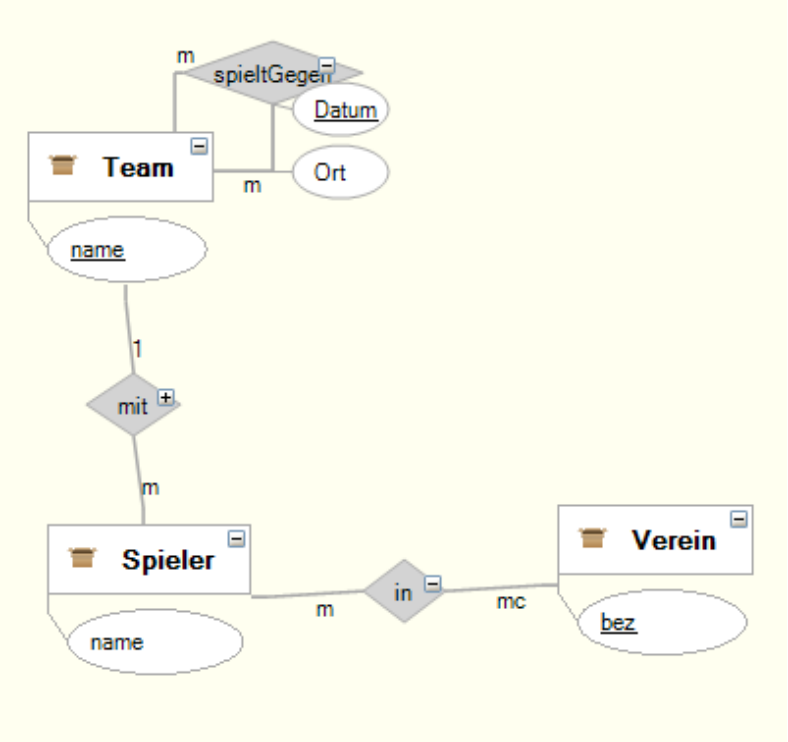
Wort	Bedeutung
`NULL`	Erlaubt Null Einträge in dieser Spalte
`NOT NULL`	Erlaubt keine Null Einträge in dieser Spalte
`PRIMARY KEY`	Markiert den Schlüssel als Fremdschlüssel. Der Wert dieser Spalte darf nicht mehrmals vorkommen (vgl. Unique). Weiter dürfen keine NULL Werte in diese Spalte eingefügt werden. Kann auch über mehrere Spalten gemacht werden: PRIMARY KEY (Spalte 1, Spalte 2, Spalte 3, …)
`UNIQUE`	Dieses Attribut stellt sicher, dass die Einträge dieser Spalte alle Eindeutig sind. Auch dieses Attribut kann über mehrere Spalten gesetzt werden: UNIQE(Spalte 1, Spalte 2, Spalte 3)
`IDENTITY`	Startet bei einem definierten Wert an und erhöht diesen für jeden Beitrag um einen definierten Schritt (Default: Anfangswert = 1, Erhöhungsschritt = 1)

3 ER → ODL

- ODL

3.1 Aufgabe

Folgender Ausschnitt des ER-Modells aus Aufgabe 2 soll objekt-orientiert implementiert werden.



Schreiben Sie für das obige ER-Modell die Klassen in ODL-Syntax. Schreiben Sie zu jeder relationship auch die inverse Angabe.

Abkürzungen sind erlaubt. Ohne eigene Erklärung können Sie abkürzen: relationship mit rel, attribute mit a, inverse mit i, class mit c.

3.2 Syntax

```
→ nur ein Key:  Class team  (extent alleTeams  key name);
→ mehrere Keys: Class spiel (extent alleSpiele key (team1, team2, date));

Class [Name] (extent [alleName] key ([key, key, …])) {
  // Attribute [Datentyp (String,Int,Date,..)] [name];
  Attribute String name;

  // 1:n Relationship [RefTabelle] [name] inverse [RefTabelle]::[RefFeld];
  Relationship Team team inverse Team::spieler;

  // n:n Relationship set<[RefTab]> [name] inverse [RefTabelle]::[RefFeld];
  Relationship set<Verein> inVerein inverse Verein::spieler;
}
```

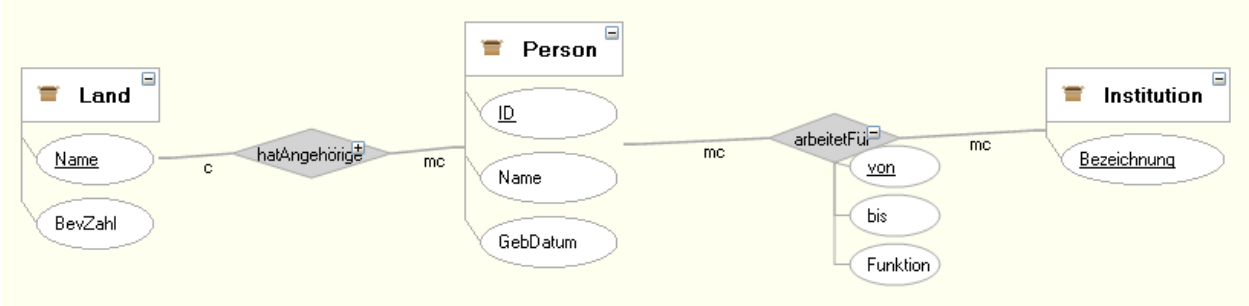
3.3 Lösung

```
Class team (extent alleTeams key name) {
  Attribute String name;
  Relationship set<Spiel> inSpielenTeam1 inverse Spiel::team1;
  Relationship set<Spiel> inSpielenTeam2 inverse Spiel::team2;
  Relationship set<Spieler> spieler inverse Spieler::team;
}

Class spieler(extent alleSpieler) {
  Attribute String name;
  Relationship Team team inverse Team::spieler;
  Relationship set<Verein> vereine inverse Verein::spieler;
}

Class verein(extent alleVereine key bez) {
  Attribute String bez;
  Relationship set<Spieler> spieler inverse Spieler::vereine;
}

Class spiel(extent alleSpiele key(team1, team2, datum)) {
  Attribute Date datum;
  Attribute String ort;
  Relationship Team team1 inverse Team::inSpielenTeam1;
  Relationship Team team2 inverse Team::inSpielenTeam2;
}
```

3.4 Zusätzliche Aufgabe ODL

Beschreiben Sie für dieses vereinfachte Modell die Klassen in ODL-Syntax. Benutzen Sie bei allen Relationships die inverse-Angabe. Geben Sie jeder Klasse einen extent-Namen.

Die „arbeitetFür-Relationship“ hat keine eigenen Attribute (von, bis, Funktion entfallen).



Lösung

```
Class Land (extent alleLänder) {
  attribute String name;
  attribute Int BevZahl;
  Relationship set<Person> angehöriger inverse Person::land;
}

Class Person(extent allePersonen) {
  attribute String name;
  attribute Date GebDatum;
  Relationship Land land inverse Land::angehöriger;
  Relationship set<Institution> institutionen inverse Institution::personen;
}

Class Institution (extent alleInstitutionen, key bezeichnung) {
  attribute String bezeichnung;
  Relationship set<Person> personen inverse Person::institutionen;
}
```

3.5 Hinweise:

extent:	„extent alleKlassennamen“
key:	„key Keyword“
attribute:	„attribute“
eine zu 1 oder zu c Beziehung:	„Relationship“
eine zu m oder zu mc Beziehung:	„Relationship set“
inverse:	bsp: „inverse Person::institutionen“

4 SQL Anfragen

- SQL
- select
- with th (rekursiv, rekursion, transitive hülle)

4.1 Aufgabe

Gegeben ist das physische Schema **Zug-Db** auf dem Extrablatt im Querformat. Sie brauchen dieses Schema auch für weitere Aufgaben. Hier einige Anmerkungen dazu.

- **Bez** ist das Abkürzung von Bezeichnung.
- Ein **Zug hat** eine Lok – wird von dieser gezogen. Eine Lok kann unterschiedliche Züge ziehen, wenn sich deren Fahrzeiten (in **Zughalt**) nicht überlappen!
- Mit **ZugWagen** wird festgehalten, welche Wagons in welchen Zügen eingesetzt werden. Ein Wagon kann in mehreren Zügen zum Einsatz kommen.
- Ein Hersteller kann für einen anderen Hersteller „arbeiten“ – ist dann **subHersteller von** diesem.
- In **Zughalt** sind auch Start- und Endbahnhof enthalten: beim Startbahnhof gibt es keine Ankunftszeit (Wert: null), beim Endbahnhof keine Abfahrtzeit (Wert: null).
- **Datum** als String: yyyy-mm-dd, z.B. 2012-06-29, **Time** als String: hh:mi, z.B. 08:30

4.2 Syntax

```
SELECT [ ALL | DISTINCT ]
[ TOP ( expression ) [ PERCENT ] [ WITH TIES ] ]
<select_list>
<select_list> ::=
    {
            *
      | { table_name | view_name | table_alias }.*
      | {
          [ { table_name | view_name | table_alias }. ]
              { column_name | $IDENTITY | $ROWGUID }
          | udt_column_name [ { . | :: } { { property_name | field_name }
            | method_name ( argument [ ,...n] ) } ]
          | expression
          [ [ AS ] column_alias ]
        }
      | column_alias = expression
    } [ ,...n ]
```

```
-- God Functions
SELECT TOP x
SELECT DISTINCT

-- Aggregate Functions
COUNT(*)
AVG(x)
SUM(x)
MIN(x)
MAX(x)
```

Aufgabe a)

Gesucht ist letzteWartung von der Lok des Zuges mit Bezeichnung AlpenExpress

Lösung a)

```
SELECT letzteWartung FROM lok JOIN zug ON lok.id=zug.lok WHERE bez='AlpenExpress'
```

Aufgabe b)

Gesucht ist die älteste letzteWartung der Wagons des Zuges mit Bezeichnung AlpenExpress

Lösung b)

```
SELECT MIN(letzteWartung) FROM Wagon WHERE id IN
  (SELECT wagon FROM ZugWagon WHERE zug = (SELECT id FROM Zug WHERE bez='AlpenExpress'));
```

Aufgabe c)

Erweitern Sie b), sodass auch die Id des Zuges ins Resultat kommt. Wahrscheinlich können Sie viel von b) übernehmen: unterstreichen Sie dort, was Sie übernehmen, und kennzeichnen Sie in der Lösung für c, wo es seinen Platz hat.

Lösung c)

```
SELECT z.id, (SELECT MIN(letzteWartung)
              FROM Wagon
              WHERE id IN (SELECT wagon FROM ZugWagon WHERE zug=z.id))
FROM zug AS z WHERE bez='AlpenExpress';
```

Aufgabe d)

Jeder Wagon soll mit seiner Id, mit der Bezeichnung seines Wagontyps und mit der Bezeichnung des Zuges, in dem er eingesetzt ist, ausgegeben werden. Ein Wagon kann für mehrere Züge eingesetzt werden. Dann gibt es mehrere Resultatzeilen für den Wagon. Wagons, die nicht eingesetzt werden (nicht in ZugWagon referenziert werden) sollen auch ausgegeben werden (ohne Bezeichnung eines Zuges)

Lösung d)

```
SELECT z.bez, w.id, wtyp.bez
FROM zug AS z
  RIGHT JOIN ZugWagon AS zw ON z.id=zw.zug
  RIGHT JOIN wagon AS w ON zw.wagon=w.id
  JOIN wagontyp AS wtyp ON w.wagontyp=wtyp.id;
```

Aufgabe e)

Gesucht sind die Namen der Hersteller, die den Wagontyp mit Bez='x' und/ oder den Loktyp mit Bez='y' herstellen. Ein Name soll nur 1x in die Ausgabe.

Lösung e)

```
SELECT DISTINCT(name)
FROM hersteller
WHERE id IN (SELECT hersteller FROM wagontyp WHERE bez='x')
   OR id IN (SELECT hersteller FROM loktyp WHERE bez='y')
```

Aufgabe f)

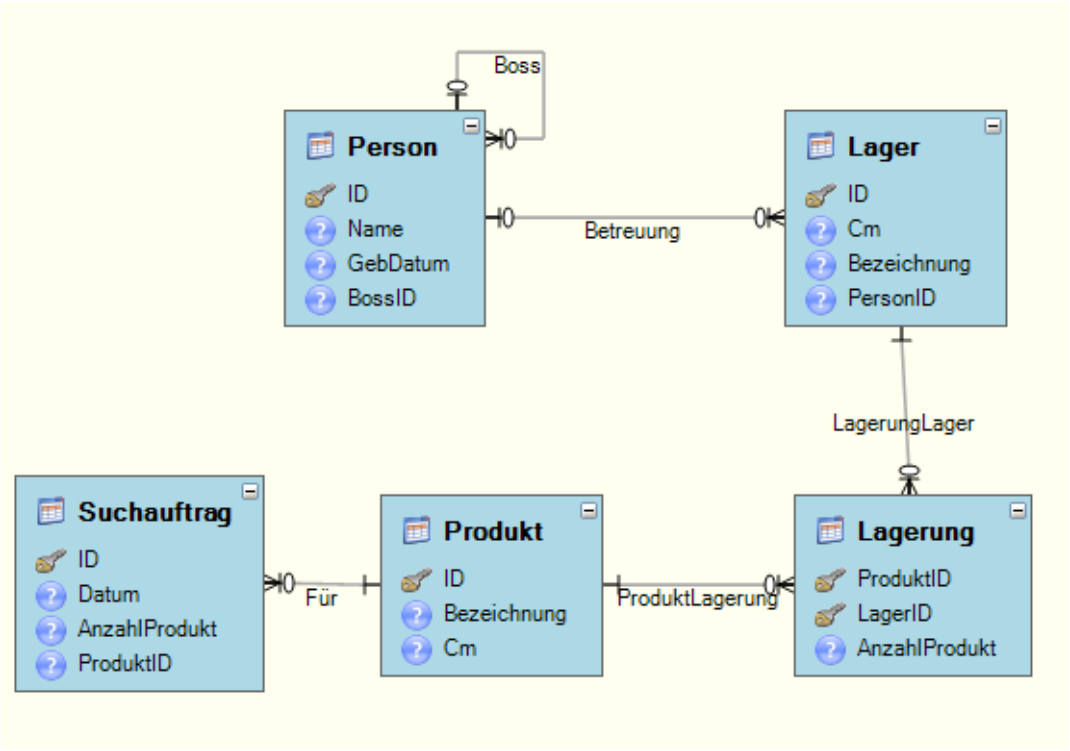
Zur Rekursion:

Gesucht sind die Ids **aller** Sub-Hersteller vom Hersteller mit Name='Siemens'. Damit sind nicht nur die direkten Sub-Hersteller gemeint, sondern auch die **indirekten** über alle Stufen (wir hatten das die transitive Hülle genannt).

Lösung f)

```
WITH TH(id) AS (
    -- Basis
    SELECT subHerstellerVon FROM hersteller AS Z
      JOIN hersteller AS h ON h.id = z.subHerstellerVon
    WHERE h.name='Siemens'
  UNION ALL
    -- Rekursionsvorschrift
    SELECT subHerstellerVon FROM hersteller AS h JOIN th ON h.id=th.id
)
SELECT * FROM th;
```

Nachfolgende Aufgaben basieren auf folgendem ERD



Aufgabe g)

Wie viele Lager gibt es für das Produkt mit der Bezeichnung ‚A380'?

Lösung g)

```
SELECT COUNT(*) FROM produkt AS p, lagerung AS l
WHERE p.bezeichnung='A380' AND p.id=l.produktid;
```

Aufgabe h)

Wie gross ist die Fläche aller Lager zusammen, die von der Person mit Namen ‚Amstutz' betreut werden? Unterschiedliche Personen mit demselben Namen werden nicht unterschieden.

Lösung h)

```
SELECT SUM(cm) FROM lager AS l, person AS p WHERE p.name='amstutz' AND p.id=l.personid;
```

Aufgabe i)

Erweiterung zu b): Weil es mehrere Personen mit Namen Amstutz geben kann, soll für jede einzelne Person mit ihrer ID die betreute Lagerfläche ausgegeben werden. Personen namens Amstutz, die keine Lager betreuen, werden unterdrückt.

Lösung i)

```
SELECT p.id, SUM(cm)
FROM lager AS l, person AS p
WHERE p.name='amstutz' AND p.id=l.personid
GROUP BY p.id;
```

Aufgabe j)

Für welches Lager/ welche Lager (ID in die Ausgabe) gab es den ersten (ältesten) Suchauftrag? Ein Suchauftrag s ist für ein Lager l, wenn das Produkt in s in l lagert/ gelagert wird. Es ist möglich, dass für den ersten (ältesten) Suchauftrag kein Lager (mehr) existiert – weil das Produkt nicht (mehr) gelagert wird.

Lösung j)

```
SELECT l.lagerid
FROM lagerung AS l, suchauftrag AS s
WHERE l.produktId = s.produktId
      AND s.datum = (SELECT MIN(datum) FROM suchauftrag);
```

Aufgabe k)

Gesucht ist die Anzahl der Suchaufträge pro Produkt, sofern diese grösser als 10 ist. In die Ausgabe kommen ProduktID und die Anzahl der Suchaufträge. Achtung: diese Anzahl hat nichts mit AnzahlProdukt zu tun!

Lösung k)

```
SELECT s.produktid, COUNT(*)
FROM suchauftrag AS s
GROUP BY s.produktId
HAVING COUNT(*)>10;
```

Aufgabe l)

Rekursion / transitive Hülle:

Für die Person mit ID=2020 sind die Boss-Personen gesucht – über alle Hierarchiestufen. Von den Boss-Personen sind ID, Name und BossID auszugeben

Lösung l)

```
WITH th (ID, Name, BossID) AS (
    -- Basis
    SELECT b.ID, b.Name, b.BossID
    FROM Person AS p
      JOIN Person AS b ON p.BossID = b.ID
    WHERE p.ID = 2020
  UNION ALL
    -- Rekursionsvorschrift
    SELECT b.ID, b.Name, b.BossID
    FROM th
      JOIN person AS b ON th.BossID=b.ID
)
SELECT * FROM th
```

5 OQL-Anfragen

- Anhand von ODL entsprechende OQL-Anfrage erstellen

5.1 Aufgabe

Ein Teil der Zug-Db ist objekt-orientiert implementiert.

```
Class LokTyp (extent alleLokTypen) {
  Attribute string bez;
  Attribute int antriebskraft;
  Relationship set<Lok> loks inverse Lok:: lokTyp;
}

Class Lok (extent alleLoks) {
  Attribute date herstellDatum;
  Attribute date letzteWartung;
  Relationship LokTyp loktyp inverse LokTyp::loks;
  Relationship set<Zug> ziehtZüge inverse Zug::gezogenVon;
}

Class Zug (extent alleZüge) {
  Attribute string bez;
  Relationship Lok gezogenVon inverse Lok:: ziehtZüge;
  Relationship set<Wagon> wagons inverse Wagon::fürZüge;
}

Class Wagon (extent alleWagons) {
  Attribute date herstellDatum;
  Attribute date letzteWartung;
  Relationship set<Zug> fürZüge inverse Zug::wagons;
}
```

Bezüglich der obigen Klassen: formulieren Sie folgende Anfragen als OQL-Ausdrücke. Alternativ können Sie auch mittels Java-Persistence-Query-Language formulieren. Attribute und Relationships entsprechen dann gleichnamigen Java Feldern.

5.2 Syntax

```
SELECT [X]
FROM [X] IN [alleTabelle] (,[y] in [x.refTab])
WHERE [x.fk(.fk)] = [Bedingung]
```

Schreiben Sie als OQL-Ausdruck oder JPA-Query (nur der select-String ist gefordert) die Anfragen

Aufgabe a)

Welche Züge (vollständige Zug-Objekte in die Ausgabe) werden von einer Lok gezogen, deren Loktyp die Bezeichnung (bez) 'BR 145' hat?

Lösung a)

```
select z from z in alleZüge where z.gezogenVon.loktyp.bez='BR 145'
```

Aufgabe b)

Welche Loks (vollständiges Lok-Objekt in die Ausgabe) ziehen Züge mit Wagons, für die letzteWartung < '2012-01-22' ist?

Lösung b)

```
select l
from l in alleLoks,
  z in l.ziehtZüge,
  w in z.wagons
where w.letzteWartung < '2012-01-22'
```

Alternative Aufgabe

```
Class Mitarbeiter (extent alleMitarbeiter) {
  relationship set(Maschine) bedient inverse Maschine::bedientVon;
  relationship set(Maschine) wartet inverse Maschine::gewartetVon;
}

Class Maschine(extent alleMaschinen) {
  relationship set(Mitarbeiter) bedientVon inverse Mitarbeiter::bedient;
  relationship Mitarbeiter gewartetVon inverse Mitarbeiter::wartet;
  relationship set(Teil) fertigt inverse Teil::gefertigtVon;
}

Class Teil( extent alleTeile) {
  attribute string bezeichnung;
  relationship Maschine gefertigtVon inverse Maschine::fertigt;
  relationship set(Teil) hatUnterteile inverse Teil::inOberteilen;
  relationship set(Teil) inOberteilen inverse Teil:: hatUnterteile;
}
```

Schreiben Sie als OQL-Ausdruck oder JPA-Query (nur der select-String ist gefordert) die Anfragen

Aufgabe

Wer wartet die Maschine, die das Teil mit bezeichnung = ‚VeloVS' fertigt? Resultat soll das vollständige Mitarbeiter-Objekt sein.

Lösung

```
// OQL
SELECT t.gefertigtVon.gewartetVon
FROM t in alleTeile
WHERE t. bezeichnung = ‚VeloVS';

// JPA-Query
SELECT t.gefertigtVon.gewartetVon
FROM Teile t
WHERE t. bezeichnung = ‚VeloVS';
```

6 JDBC

6.1 Wichtige Funktionen

JDBC Command	Beschreibung
void [Connection].**createStatement**();	Erzeugt ein neues Statement auf, welches als DB-Verbindung das Connection Objekt enthält, auf welchem diese Methode ausgeführt wurde.
ResultSet [Statement].**executeQuery**(string Query)	Führt ein Query auf einer Datenbank aus und liefert eine ResultSet Instanz zurück. In diesem ResultSet sind Einträge enthalten, welche dem Query entsprechen. (z.B. für SELECT)
bool **execute**(string Query)	Diese Methode führt ein Query auf einer Datenbank aus. Kann für alle Arten von Queries eingesetzt werden.
int **executeUpdate**(string Query, *[int autoGeneratedKeys])*	Diese Methode führt Mutationen auf einer Datenbank aus (INSERT, UPDATE, DELETE). Dabei kann Optional noch angegeben werden, ob die automatisch Generierten Schlüssel ausgelesen werden sollen mit dem Parameter autoGeneratedKeys (Konstante dafür ist Statement. RETURN_GENERATED_KEYS).
ResultSet [Statement].**getGeneratedKeys**()	Liefert ein ResultSet zurück, welches die automatisch generierten Schlüssel beinhaltet. Wird nach executeUpdate(Query, Statement.RETURN_GENERATED_KEYS) aufgerufen).
bool [ResultSet].**next**()	Mit dieser Methode wird von Eintrag zu Eintrag gesprungen. Dies ist auch die erste Methode, welche man aufrufen muss um die Einträge abzufragen.
int [ResultSet].**getInt**(int index) int [ResultSet].**getInt**(string columnName)	Liest das Feld mit dem übergebenen Index oder dem Spaltennamen als Integer aus.
int [ResultSet].**getFloat**(int index) int [ResultSet]. **getFloat** (string columnName)	Liest das Feld mit dem übergebenen Index oder dem Spaltennamen als Float aus.
int [ResultSet].**getDate**(int index) int [ResultSet].**getDate**(string columnName)	Liest das Feld mit dem übergebenen Index oder dem Spaltennamen als Datum aus.
int [ResultSet].**getTime**(int index) int [ResultSet]. **getTime** (string columnName)	Liest das Feld mit dem übergebenen Index oder dem Spaltennamen als Zeit (Time) aus.
int [ResultSet].**getString**(int index) int [ResultSet].**getString**(string columnName)	Liest das Feld mit dem übergebenen Index oder dem Spaltennamen als String aus.

Aufgabe 1

Gegeben ist die Datenbank mit dem Schema aus Aufgabe 3. Der Primärschlüssel von Suchauftrag (ID) wird automatisch vergeben (identity oder auto_increment).

Schreiben Sie eine Methode

static boolean handleSuchAuftrag(Connection conn, int *suchProduktID*, int *suchAnzahlProdukt*) throws SQLException

die einen neuen **Suchauftrag** mit existierender ProduktID=*suchProduktID*, anzahlProdukt = *suchAnzahl-Produkt* und Datum= current_timestamp einfügt. Der Suchauftrag darf allerdings nur eingefügt werden, wenn insgesamt mindestens *suchAnzahlProdukt* Exemplare dieses Produktes am Lager sind (Einträge in Lagerung). Nur solche Suchaufträge sind „realisierbar“ (Sprechweise für Aufgabe 6). Bei erfolgreicher Einfügung ist der Returnwert true, sonst false.

Lösung 1

```
static boolean handleSuchAuftrag(Connection conn, int suchProduktID,
                                 int suchAnzahlProdukt) throws SQLException {
   Statement s = conn.createStatement();
   ResultSet rs = s.executeQuery(“select sum(anzahlProdukt) from lagerung
                                    where produktId=” + suchProduktID);
   rs.next();

   if (rs.getInt(1) < suchAnzahlProdukt) return false;

   rs. close();
   return s.execute(“insert into suchauftrag(produktid, datum, anzahlProdukt)
                       values(“ + suchProduktID +”, current_timestamp,”
                         + suchAnzahlProdukt +”)” );
}
```

Aufgabe 2

Gegeben ist die Zug-Db. Es soll ein neuer Zug eingetragen werden – mit vorgegebener Lok (lokId), Bezeichnung (bez) und für ein vorgegebenes Zeitinterval vb (vonTime, bisTime). Der Zug darf nur eingetragen werden, wenn die vorgegebene Lok nicht innerhalb von vb für einen anderen Zug genutzt wird (für eine solche Unverträglichkeit gilt, dass ein bisheriger Zug mit der vorgegebenen Lok innerhalb von vb an- oder abfährt).

Bemerkung: Dies ist eine grobe Vereinfachung. Es müsste auch berücksichtigt werden, dass genügend Zeit bleibt, um die Lok von ihrem letzten Zielbahnhof an ihren neuen Startbahnhof zu bringen. Das soll hier aber nicht überprüft werden!

Für einen erfolgreichen Eintrag soll die Id des neuen Zuges zurückgegeben werden. Kann der Eintrag nicht gemacht werden, gibt es eine Exception oder -1 als Rückgabewert. Nehmen Sie an, dass der Java Datentyp time mit dem DB-Datentyp time (Typ von ankunft, abfahrt) verträglich ist und dass die Umwandlung zum String in Java automatisch passend gemacht wird.

Schreiben Sie eine Methode

static int insertBid(Connection conn, int lokId, string bez, time vonTime, time bisTime) throws SQLException

```
static int insertBid(Connection conn, int lokId,
                       time vonTime, time bisTime) throws SQLException {
   Statement stmt = conn.createStatement();
   ResultSet rs=stmt.executeQuery(
      “select count(*) from zug z join zughalt zh on z.id=zh.zug
       where lok=” + lokId
       + “ and ankunft between “ + vonTime + “and “ + bisTime
       + “ or abfahrt between “ + vonTime + “and “+ bisTime);

   rs.next();

   if (rs.getInt(1) > 0) return -1;

   stmt.executeUpdate(
      “insert into zug(bez,lok) values(‘”+bez+”’,”+lokId+”)” ,
      Statement.RETURN_GENERATED_KEYS);

   rs = stmt.getGeneratedKeys();
   rs.next();
   return rs.getInt(1);
}
```

7 Trigger

- ROW-Level Trigger

7.1 Syntax

Folgende Syntax wird verwendet, um einen Row-Level Trigger zu erstellen.

```
CREATE TRIGGER Name [BEFORE / AFTER / INSTEAD OF]  [INSERT / UPDATE / DELETE]
  ON Tabellenname
  FOR EACH ROW
  BEGIN
       -- Inhalt
 END
```

Prüfungsaufgabe (Formativer Test 13)

Sorgen Sie mit einem Trigger dafür, dass ein insert von „nicht realisierbaren“ Suchaufträgen verhindert wird (diese Sprechweise „nicht realisierbar“ wird in Aufgabe 5 eingeführt; die Lösung von Aufgabe 5 ist keine Voraussetzung für diese Aufgabe).

Beispiel: Für

- `insert into Suchauftrag(ProduktID , Datum , AnzahlProdukt) values(5, .., 100);` soll der Trigger rollback() aufrufen, wenn die Summe von AnzahlProdukt in Lagerung für das Produkt mit Id=5 unter 100 ist. Dies ist eine beispielhafte Erklärungen für „wann ist ein Suchauftrag nicht realisierbar“. Der Trigger soll für alle Produkte funktionieren.

Empfehlung/ Hinweise:

- Schreiben Sie einen row-level trigger, auch wenn es den bei MS SqlServer nicht gibt.
- Variable können Sie vereinbaren mit DECLARE <Name> <Type>;
- Ist das Resultat einer Anfrage nur eine Zeile, so können Sie Attribute direkt im select an Variable zuweisen. z.B. declare @myCnt int; select @myCnt = count(*) from studenten;

Lösung:

```
CREATE TRIGGER CheckSuchauftrag ON suchauftrag BEFORE INSERT
FOR EACH ROW
BEGIN
       DECLARE INT @cnt;

       SELECT @cnt = SUM(anzahlProdukt) FROM lagerung WHERE produktId = new.produktID;

       IF (@cnt < new.anzahlProdukt) rollback();
END;
```

7.2 Prüfungsaufgabe 2 (MEP 12)

Schreiben Sie einen Trigger, der ein insert in die Tabelle ZugHalt nur zulässt wenn

- Ankunft < Abfahrt ist oder eine der Zeiten null ist
- Und kein anderer (!) ZugHalt-Eintrag für den Bahnhof des neuen Eintrags mit identischer Ankunftszeit existiert. Damit soll ausgeschlossen werden, dass Züge gleichzeitig einfahren – das mag unsinnig erscheinen, akzeptieren Sie es trotzdem.

Lösung

```
CREATE TRIGGER checkZughalt BEFORE INSERT ON ZugHalt
FOR EACH ROW
BEGIN
	IF (new.abfahrt <= new.ankunft) {
		rollback();
		return;
	}

	IF (EXISTS(SELECT null FROM zughalt
			WHERE ankunft = new.ankunft
				AND bahnhof=new.bahnhof
				AND zug <> new.zug)
	   ) {
		ROLLBACK();
	}
END;
```

8 Normalform, BCNF

- Normalform
- BCNF (Boyce–Codd normal form)

8.1 Regeln

Eine Tabelle ist in Boyce-Codd-Normalform (BC-NF), wenn

1. **Alle Attribute atomare Wertebereiche haben**
   (die einzelnen Werte sind z.B. keine Listen, keine Mengen, keine Strukturen)
2. **Für jede funktionale Abhängigkeit A→B in T gilt auch: A hat Schlüsseleigenschaft**

8.2 Aufgaben

Prüfungsaufgabe 1

Zusätzliche Annahme: jeder Hersteller stellt **entweder** nur doppelstöckige Wagontypen (doppelGeschoss=1) **oder** nur einstöckige Wagontypen (doppelGeschoss=0) her, nie sowohl Typen mit doppelGeschoss = 0 als auch Typen mit doppelGeschoss = 1.

Mit dieser Annahme: ist die Tabelle WagonTyp in Boyce-Codd-Normalform?

Argumentieren Sie mit den formalen Definitionen! Eine Argumentation über die 3te Normalform (im Unterricht nicht behandelt) wird auch akzeptiert.

Lösung:

doppelGeschoss ist funktional abhängig von hersteller

hersteller hat keine Schlüsseleigenschaft

➔ Verletzung

Prüfungsaufgabe 2

In die Tabelle Suchauftrag kommt zusätzlich noch das Attribut ProduktBezeichnung – es ist die Bezeichnung des Produktes mit ID = ProduktID.

Suchauftrag: {[ID: int, ProduktID : int, ProduktBezeichnung : varchar(20), Datum : datetime, AnzahlProdukt : int]}

Weiterhin gilt, dass es viele Suchaufträge für ein Produkt geben kann.
Begründen Sie formal, warum die so erweiterte Tabelle in der Boyce-Codd-Normalform ist bzw. dagegen verstösst.

Lösung:

ProduktBezeichnung ist funktional abhängig von ProduktId. ProduktID (alleine) hat aber keine Schlüsseleigenschaft in Suchauftrag.

➔ Verstoss gegen BC-NF.

9 B*-Baum

- Blockgrösse
- Höhe des Baumes

9.1 Formel

Anz Einträge eines Indexblockes:

$$\frac{\boldsymbol{Indexblockgrösse}}{(\boldsymbol{Grösse\ des\ Index} + \boldsymbol{Grösse\ des\ Index\ zu\ den\ Basisdaten})} * \mathbf{80\%}\ \boldsymbol{gefüllt}$$

Höhe des Baumes:

$$\boldsymbol{Log_{AnzEinträge}(Anzahl\ Aufträge)}$$

9.2 Aufgabe 1

Häufig werden alle Suchaufträge für einen speziellen Tag gesucht:

(Q) SELECT * FROM Suchauftrag WHERE Datum >= '2007-06-23' AND Datum < '2007-06-24'

Bemerkung: Datum ist vom Typ DateTime und speichert also auch die Zeit (auf Millisekunden!).
Das Datumsformat stimmt und impliziert genau 00-Uhr.

Suchaufträge sind über mehrere Jahrzehnte gespeichert Es gibt 1.000.000 solcher Aufträge. Pro Tag (Teil der Datumsangabe) gibt es durchschnittlich 100 Einträge. Es gibt einen Index für *Suchauftrag.Datum* - als B*-Baum organisiert.

V-1: Die Blockgrösse sei 1KB – ich runde das zu 1000B.
V-2: Die Datumswerte brauchen 8B (Attributlänge).
V-3: Verweise innerhalb von einem Index (B*-Baum) und von den Blattknoten auf die Basissätze brauchen jeweils 4B.

Aufgabe a)

Mit den Vorgaben V-1,2,3: Wie viele Blockzugriffe braucht es für die Abarbeitung von (Q) über den B*-Baum? Zeigen Sie, wie Sie rechnen. Konkrete Zahlen als Ergebnis sind nicht notwendig. Sofern notwendig: machen Sie weitere Annahmen

Lösung a)

Einträge pro Block: $\frac{\mathbf{1000\ B}}{(\mathbf{8\ B+4B})} \cdot \mathbf{0.8} = \mathbf{66.6} \rightarrow \mathbf{66}$ **Einträge**

Höhe des Baumes: $\mathbf{log_{66}(1'000'000)} = \mathbf{3.297} \rightarrow \mathbf{4}$

Sie haben 100 relevante Suchaufträge. Diese passen nicht in diese 66 Einträge.

Somit müssen mindestens 2 Blattknoten des B*-Baumes gelesen werden.

→ 4 + 1 = 5 → Somit müssen im B*-Baum **5 Blöcke** gelesen werden.

Pro Tag gibt es 100 Einträge. Da wir nur einen Tag haben, bleibt es bei 100 Einträge. Daraus ergibt sich, dass nochmals 100 Blöcke gelesen werden müssen, ➔ **insgesamt also 105**!

Aufgabe b)

Statt der Annahmen V-1 bis V-3 gilt: in einem IndexBlock sind durchschnittlich 300 Indexeinträge. Ein solcher Eintrag besteht aus Datum und Verweis. Zeigen Sie, wie Sie unter dieser Annahme die Zahl der Blockzugriffe berechnen.

Lösung b)

Einträge pro Block: **300 Einträge**

Höhe des Baumes: $\mathbf{log_{300}(1'000'000) = 2.42 \rightarrow 3}$

Ich darf annehmen, dass die 100 relevanten Verweise alle in einem Block sind.

Somit sind 3 B*-Baum-Blöcke zu lesen.

Bei den Basisdaten bleibt es bei zusätzlichen 100 Blöcken. ➔ **insgesamt also 103**

9.3 Aufgabe 2

Nehmen Sie an, dass es 10.000 Wagons gibt und dass täglich 30 gewartet werden.

Es gibt einen Index über *letzteWartung* der Tabelle Wagon – organisiert als B*-Baum.

Die Indexblöcke sind 1KB gross, Datumsfelder und Verweise (innerhalb vom Index und vom Index zu den Basisdaten) sind jeweils 4B lang. Indexblöcke sind zu 80% gefüllt.

Die Basisdaten der Wagon-Tabelle: durchschnittlich sind 2 komplette Wagon-Sätze/Zeilen in einem Block – es gibt viel mehr Wagon-Attribute als im Schema sichtbar sind.

Berechnen Sie für

SELECT * FROM Wagon WHERE letzteWartung BETWEEN '2012-06-25' AND '2012-06-27' (Q)

die Anzahl der Blockzugriffe

a. wenn der Index für *letzteWartung* genutzt wird
b. wenn nur sequentiell in den Basisdaten (ohne Index) gesucht wird

Zeigen Sie mit den Antworten, wie Sie rechnen. Ein Zahlenergebnis ist nicht gefordert.

Lösung a)

Einträge pro Block: $\frac{1024\,B}{(4\,B + 4B)} \cdot 0.8 = 102.4 \rightarrow$ **102 Einträge**

Höhe des Baumes: $\mathbf{log_{102}(10'000) = 1.99 \rightarrow 2}$ **Indexblöcke**

Between '2012-06-25' and '2012-06-27' → 3 Tage → 3 Tage * 30 = **90 Sätze**

Da die 90 Blöcke in den 102 Einträgen Platz haben, bleibt es bei den 90 Blöcken.

90 Blöcke + die 2 Indexblöcke = **92 Blöcke.**

Lösung b)

Alle Blöcke der Basisdaten müssen gelesen werden (2 Zeilen in einem Block): $\frac{10'000}{2} = 5'000$

10 Optimierung von Anfragen

Gegeben ist wieder die Zug-Db. Folgende Anfrage wird ausgeführt:

```
SELECT DISTINCT WTyp.*
FROM Zug, ZugWagon ZW, Wagon W, WagonTyp WTyp
WHERE Zug.lok=3 AND Zug.Id=ZW.zug AND ZW.wagon=W.id AND W.wagonTyp=WTyp.id
```

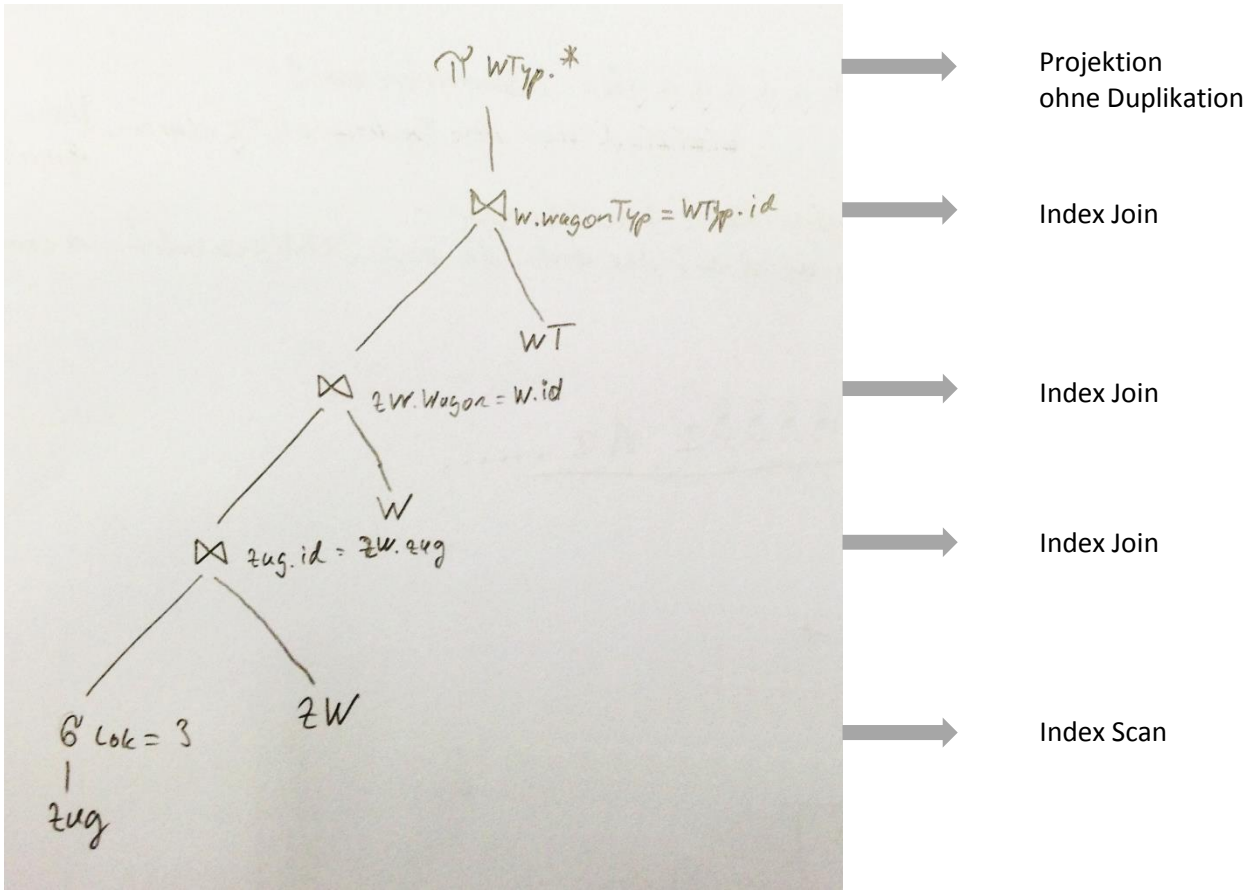
nur sehr wenige Züge diese Lok haben (von ihr gezogen werden).

Für die Tabelle ZugWagon ist der Primärschlüssel als primary key(zug, wagon) definiert.

Erstellen Sie einen „optimalen“ Auswertungsplan für die obige Anfrage. Geben Sie dafür die Ergebnisse für die beiden folgenden Optimierungsschritte in Baum-Form an:

- Ergebnis der logischen Optimierung
- Auswertungsplan, physische Optimierung

Präzisierungen der physischen Optimierung. In der Anfrage habe ich die Kurzformen als Alias-Namen für die längeren Tabellennamen definiert (Umbenennung). Sie können direkt die Kurzformen als Tabellennamen benutzen.



11 Transaktionen

- Transaktionen
- Deadlock

Aufgabe a)

- 2 inserts in die ZugHalt-Tabelle - mit dem Trigger wie in Aufgabe 7 beschrieben – laufen parallel.

Jedes insert (zusammen mit dem Trigger) ist Teil einer Transaktion. Kann es zu Inkonsistenzen kommen?

Inkonsistenz heisst hier: 2 Züge haben die gleiche Ankunftszeit im selben Bahnhof.

Argumentieren, begründen Sie für beide Isolationsstufen serializable und read committed

Lösung a)

Read committed:	der andere neue Eintrag wird ignoriert
	→ Inkonsistenz möglich
Serializable:	auf den anderen neuen Eintrag muss gewartet werden (commit)
	→ keine Inkonsistenz

Aufgabe b)

Transaktion A

- A1) Select * from Hersteller where id=1;
- A2) Select * from Hersteller where id=7;
- A3) Update Hersteller set subHersteller=1 where id=7;
- A4) Commit;

Transaktion B

- B1) Select * from Hersteller where id=7;
- B2) Update Hersteller set subHerstellerVon=1 where id=7;
- B3) Update Hersteller set anzahl=anzahl+1 where id=1;
- B4) Commit;

Geben Sie eine Folge von Ai und Bj Operationen, die in einem Deadlock enden, wenn beide Transaktionen unter der Isolationsstufe **serializable** laufen.

Lösung b)

A1, A2, B1, B2, A3

Aufgabe c)

Geben Sie ein konkretes Beispiel von 2 Transaktionen, so dass es zu einer Deadlock-Situation kommt. Nehmen Sie dazu an, dass die Transaktionen unter dem isolation level serializable laufen und dass für jede Operation die betroffenen Datensätze gesperrt werden (shared bzw. exclusive, pessimistic locking).

Konkret heisst: geben Sie die SQL-Statements an, die einen Deadlock herbeiführen (auf welchen Tabellen die Statements operieren, entscheiden Sie).

Lösung c)

Transaction-1	**Transaction-2**
`UPDATE konto SET saldo = saldo – 100 WHERE id=1;`	
	`SELECT * FROM konto WHERE id = 2;`
`UPDATE konto SET saldo = saldo + 100 WHERE id = 2;` **(→ wartet)**	
	`SELECT * FROM konto WHERE id = 1;` **(→ deadlock)**

12 Transaktionen Recovery

- Recovery
- Transaktion Log / DB Dump

Aufgabe a)

Welche Informationen (Daten) muss das DBMS haben, damit ein Plattencrash „vollständig“ repariert werden kann?

Lösung a)

Backup, redo-log (Archiv)

Aufgabe b)

Welche Änderungen werden mittels Recovery nicht wieder hergestellt?

Lösung b)

Änderungen von non-committed Transaktionen (losers)

13 Datenschutz (View, Grant)

- Datenschutz
- View
- Grant
- Revoke

13.1 Aufgabe 1

Als DB-Benutzer B1 haben Sie die Tabelle Person(Id, Name, Gehalt, AbteilungsNr) kreiert.

Der DB-Benutzer B2 soll auf die Personen mit der AbteilungsNr 7 lesend zugreifen können.

Schreiben Sie dafür die entsprechenden SQL-Statements.

```
CREATE VIEW Person7 AS
  SELECT *
  FROM Person
  WHERE AbteilungsNr = 7;

GRANT SELECT ON Person7 TO B2;
```

Leider kann man im grant-statement direkt keinen Filter mitgeben – nur via view.

13.2 Aufgabe 2

An Bahnhöfen soll für den Db-Benutzer ruhigeSBB Nachtruhe herrschen. Er sieht keine Zughalt-Einträge mit Ankunftszeiten > '22:00' oder Abfahrtzeiten < '05:00', zudem kann er auch keine Änderungen und Einfügungen ausserhalb dieser Restriktionen machen.

Der Benutzer ruhigeSBB hatte bisher select-, insert-, update-Rechte auf der Tabelle Zughalt. Diese Rechte sollen ihm entzogen werden. Stattdessen soll er diese Rechte auf einer View bekommen, die die Nachtruhe „erzwingt“ – zumindest aus Db-Sicht für diesen Benutzer.

Schreiben Sie die entsprechenden SQL-Statements.

Hier eine Erinnerung dazu:

- Wenn eine View-Definition mit “WITH CHECK OPTION” beendet wird, werden nur Änderungen akzeptiert, die die Bedingungen der View erfüllen.

```
-- Alle Rechte entfernen
REVOKE INSERT, SELECT, UPDATE ON Zughalt FROM ruhigeSBB;

-- oder
REVOKE ALL ON Zughalt FROM ruhigeSBB;

-- View erstellen
CREATE VIEW ruhigerZughalt AS
  SELECT * FROM Zughalt
  WHERE (ankunft IS NULL OR ankunft <= '22:00')
        AND (abfahrt IS NULL OR abfahrt >= '05:00')
WITH CHECK OPTION;

-- Rechte auf die View vergeben
GRANT INSERT, SELECT, UPDATE ON ruhigerZughalt TO ruhigeSBB;
```

14 OLAP

Die Daten der Zug-Db sind in einem Data-Warehouse denormalisiert, aber auch mit zusätzlicher Information gespeichert. Es gibt die Faktentabelle ZugFact, die tagesgenau die view

```
SELECT z.id, z.lok, w.id, w.herstellDatum, w.letzteWartung, w.wagontyp
FROM (Zug z JOIN ZugWagon zw ON z.id=zw.zug) JOIN Wagon w ON zw.wagon=w.id
```

materialisiert.

Tagesgenau heisst: es kommt noch ein date-Feld **einsatzdatum** hinzu – von Tag zu Tag kann sich die Wagon-Zuordnung ändern.

ZugFact hat also die Attribute/ Felder zugId, einsatzdatum, lok, wagonId, herstellDatum, letzteWartung, wagontyp zugId hat jetzt nur zusammen mit einsatzdatum und wagonId Schlüsseleigenschaft.

Es gibt weiterhin die Tabellen Lok und WagenTyp. Zusätzlich gibt es eine Zeittabelle Z mit einem Date-Feld als Primärschlüssel (id), der als Fremdschlüssel in ZugFact genutzt wird für die dortigen Date-Felder.

Die Z-Tabelle hat u.a. ein int-Feld Jahr (mit dem Jahr aus dem Id-Feld) und ein bit-Feld fussballTag, das mit 0 bzw. 1 für angibt, ob am Tag des Id-Feldes wichtige Fussballspiele stattfanden.

Aufgabe a)

Nennen Sie die Dimensionstabellen der Faktentabelle ZugFact

Lösung a)

Z, Lok, Wagentyp

Aufgabe b)

Geben Sie ein select-statement, das die Anzahl der eingesetzten Wagons pro WagonTyp an jedem Tag mit fussballTag = 1 ausgibt (TagessummeProTyp). Weiter soll darauf basierend

- die Tagessumme unabhängig vom Wagontyp
- die Summe pro Jahr der eingesetzten Wagons
- letztlich die Summe der eingesetzten Wagons über alle Jahre

im Ergebnis sein.

Lösung b)

```
SELECT Z.Id, Z.jahr, wagentyp, COUNT(*)
FROM ZugFact, Z
WHERE einsatzdatum = Z.id
GROUP BY CUBE(Z.Id, Z.jahr, wagentyp);
```

15 XQuery

Teile der Zug-Db sind in folgender XML-Struktur gespeichert:

```
<SBB>
   <Bhf Name="Zh Hbf" AnzGleise="30">
      <ZHlt ZugNr="z1" Abfahrt="05:13" />
      <ZHlt ZugNr="z2" Ankunft="06:11" Abfahrt="06:17" />
      <!-- viele weitere Zughalte in Zh Hbf -->
      <ZHlt ZugNr="z321" Ankunft="22:13" />
   </Bhf>
   <Bhf Name="Thalwil" AnzGleise="8">
      <ZHlt ZugNr="z1" Ankunft="05:28" Abfahrt="05:30" />
      <ZHlt ZugNr="z2" Ankunft="06:32" Abfahrt="06:34" />
      <!-- viele weitere Zughalte in Thalwil -->
   </Bhf>
   <Zug ZugId="z1" Bez="S1 Pfeil" Lok="l1" Zw="w1 w2" />
   <!-- viele weitere Züge -->
   <WTyp Bez="komfort W1" AnzSitzplätze="100" Doppelgeschoss="0">
      <W WId="w1" HerstellDatum="23.1.2007" LetzteWartung="13.11.2011" />
      <!-- viele weitere Wagons dieses Typs -->
   </WTyp>
   <!-- viele weitere Wagontypen mit ihren Wagons -->
</SBB>
```

Diese Daten sind in der Tabelle myXml im XML-Attribut sbbData gespeichert. Für die folgenden Anfragen können Sie davon ausgehen, dass myXml nur einen Satz (eine Zeile) enthält.

Schreiben Sie folgende Anfragen als SQL-select-statements mit XQuery-Ausdrücken bezüglich der Tabelle myXml.

Aufgabe a)

Welche Züge (ZugNr in die Ausgabe) kommen in Horw (Name von Bhf) um 08:13 (Ankunft von ZHlt) an?

Lösung a)

```
select sbbData.query('for $b in //Bhf[@Name=="Horw"]/ZHlt where $b/@Abfahrt="05:13"
                      return <a>{$b/@ZugNr}</a> ') from myxml;
```

Aufgabe b)

Welche Wagons (W-Element in die Ausgabe) vom Typ X (Bez von WTyp) sind im Zug mit ZugId= z1?

Lösung b)

```
select sbbData.query('for $w in //WTyp[@Bez="komfort W1"]/W,
                          $z in //Zug[@ZugId="z1"] where contains($z/@Zw,$w/@WId)
                      return $w') from myxml;
```

16 Relationale Algebra

16.1 Optimierung

Wichtig ist beim Erstellen von Optimierungen für Abfragen ist, dass man die Filterung der Daten möglichst früh macht.

16.2 Aufgabe 1

Gegeben ist ein Teil der Datenbank aus Aufgabe 3 mit für diese Aufgabe geeigneteren Namen für den Primärschlüssel (Änderung kursiv dazugesetzt).

```
Person: {[PersonID : int, Name : varchar(30), GebDatum : datetime]}
Lager: {[LagerID : int, cm : float, PersonID : int, bezeichnung : varchar(20)]}
Lagerung: {[ LagerID : int, ProduktID : int, AnzahlProdukt : int]}
Produkt: {[ ProduktID : int, bezeichnung : varchar(50) unique, cm : float]}
```

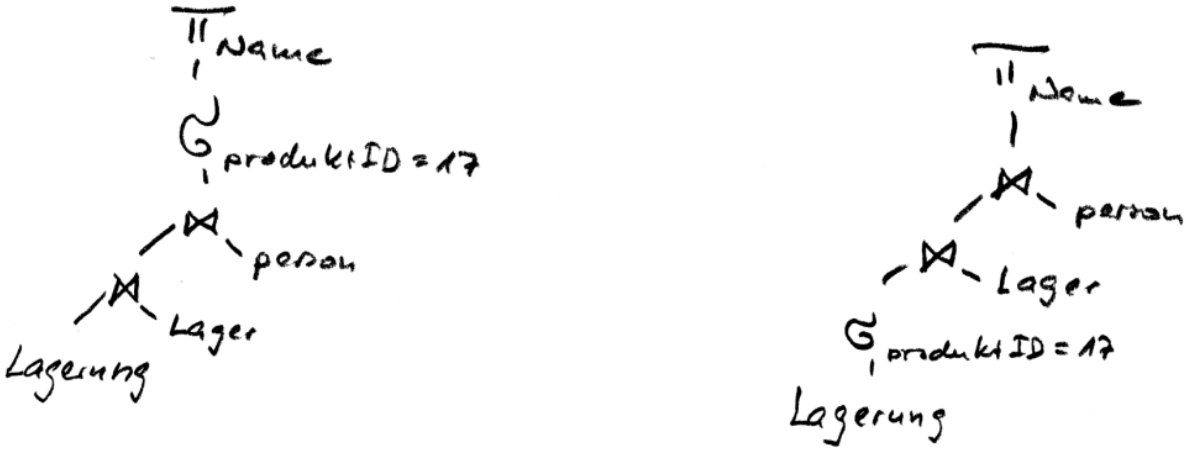
Q: Von welchen Personen (Name in die Ausgabe) werden die Lager betreut, in denen das Produkt mit der ID 17 gelagert ist?

a) Wie lauten zwei äquivalente, aber syntaktisch verschiedene relational algebraische Ausdrücke für Q?

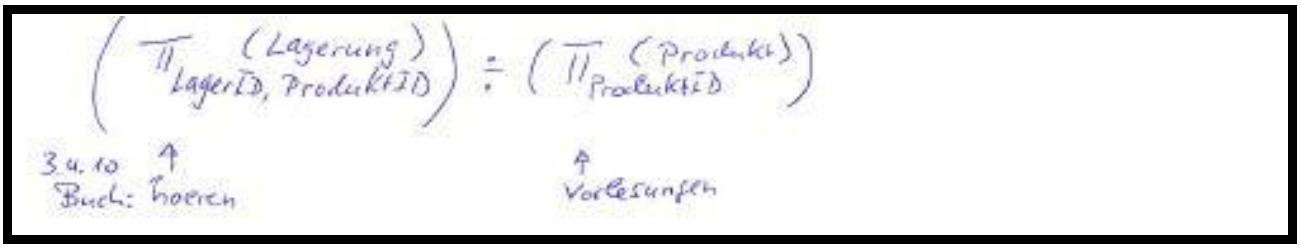
$$\pi_{Name}(\sigma_{ProduktId=17}((Lagerung \bowtie Lager) \bowtie Person))$$

$$\pi_{Name}(((\sigma_{ProduktId=17}(Lagerung)) \bowtie Lager) \bowtie Person)$$

b) Zeichen Sie für die beiden relational algebraischen Ausdrücke von a) auch die Baumdarstellung auf (2 Baumdarstellungen).



c) In welchen Lagern sind (entsprechend Lagerung-Tabelle) alle Produkte? Von diesen Lagern soll nur die LagerID in die Ausgabe.

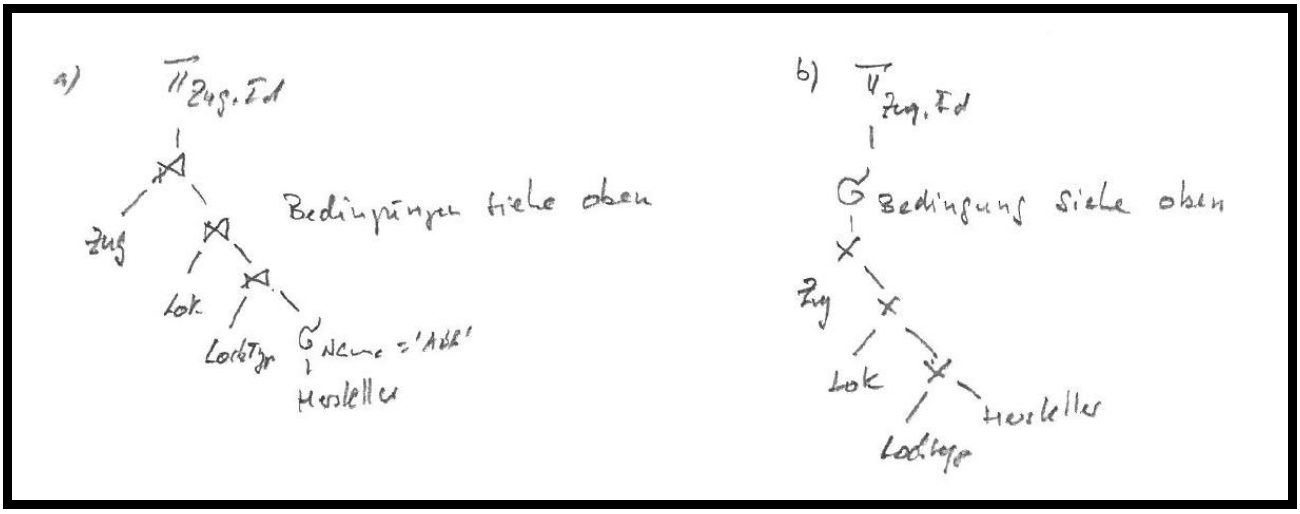


16.3 Aufgabe 2

Wieder ist der Bezug auf die Zug-Db.

Anfrage Q:
Welche Züge (Id in die Ausgabe) haben Loks (ihre IDs sind in Zug.lok) mit einem Loktyp vom Hersteller ABB (Name des Herstellers) ?

a) Zeichen Sie für jeden relational algebraischen Ausdruck aus a) die Baumdarstellung (insgesamt 2 Baumdarstellungen). (3 Punkte)



b) (Division, 3 Punkte):
Schreiben Sie als relational algebraischen Ausdruck:
Welche Züge (nur Id in die Ausgabe) halten an allen grossen Bahnhöfen (ein grosser Bahnhof hat mehr als 20 Gleise)