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.

Aufgabe 1 (ca. 15 - 20min)

Ein Marktforschungsinstitut hat erhoben,

- welche Baumärkte welche Produkte verkaufen,
- welche Personen bei welchem Baumarkt einkaufen und
- welche Personen bevorzugt welche Produkte im Baumarkt statt in anderen Läden wie Supermärkten

Dazu wurden die folgenden drei Tabellen in einer Datenbank angelegt und es werden auszugsweise Daten dargestellt:

Produktangebot

Baumarkt	**Produkt**
Hornbach	Schrauben
Hornbach	Werkzeug
OBI	Werkzeug
OBI	Möbel
TOOM	Lampen
TOOM	Blumen
Bauhaus	Schrauben

Baumarktkunden

Kunde	**Baumarkt**
Müller	Hornbach
Schulze	Hornbach
Müller	OBI
Schmidt	Marktkauf
Horst	TOOM

Bevorzugt

Kunde	**Produkt**
Schulze	Schrauben
Müller	Werkzeug
Müller	Blumen
Schmidt	Möbel
Schmidt	Lampen
Horst	Blumen
Meier	Werkzeug

Aufgabe 1:
Es ist folgendes bekannt: OBI will Marktanteile von den anderen Baumärkte erobern und deshalb alle möglichst alle Kunden aggressiv bewerben, die bei der Konkurrenz einkaufen. Dazu wird eine Tabelle Werbung mit den Namen der Kunden aufgebaut, in die alle potentiellen Kunden aufgenommen werden sollen. Wann immer ein Kunde eines anderen Baumarktes in die Tabelle Baumarktkunden hinzukommt, soll dieser in die Werbungstabelle übernommen werden.

Erklären Sie, mit welchem Instrument dies in der Datenbank implementiert werden kann und Schreiben sie dazu auch noch entsprechenden PL/SQL-Sourcecode.

Aufgabe 2:
Schreiben Sie mit Hilfe eines Cursors eine PL/SQL Prozedur, welche als Eingabeparameter den Namen eines Baumarktes erhält. Die Prozedur soll alle Produkte der Kunden ermitteln, die nicht in diesem Baumarkt einkaufen. Zum Schluss soll die Prozedur noch ausgeben, wie viele Produkte ausgegeben wurden.

Aufgabe 3 (15min)

Schreiben Sie eine PL/SQL-Funktion, die in einer Schleife von 1 bis zu einem übergebenen n die entsprechenden Einträge in die Tabelle TRAININGSPLAN nach dem nachfolgenden Schema einträgt und die Summe der zu laufenden Kilometer zurückgibt:

LAUFTAG	KM	ZEIT
1	1	60
2	1,5	80
3	2	100
4	2,5	120
…	…	…

Aufgabe 4 (15min)

Gegeben seien folgende zwei Tabellen:

Abteilung

AbteilungsNr	AbteilungsName	Ort
1	Personalabteilung	Erdgeschoss
2	Controlling	1. Stock
3	Marketing	1. Stock

Mitarbeiter

MitarbeiterNr	Nachname	Vorname	AbteilungsNr
1	Andersen	Andreas	1
2	Hansen	Helga	3
3	Hirsch	Harry	3

Weiterhin sei folgender View MitarbeiterUndAbteilung gegeben:

```
CREATE VIEW MitarbeiterUndAbteilung AS
      SELECT MitarbeiterNr, Nachname, Vorname, AbteilungsName, Ort
      FROM Mitarbeiter
      LEFT JOIN Abteilung USING (AbteilungsNr);
```

Wenn ich nun in den View einen neuen Datensatz einfügen möchte (z.B. mit dem Befehl `INSERT INTO MitarbeiterUndAbteilung VALUES ('4','Einstein','Albert','Forschung','2.Stock');`) erhalte ich eine Fehlermeldung.

a) Wieso erhalte ich diese Fehlermeldung?

b) Schreiben Sie einen PL/SQL-Trigger, der ein INSERT auf den View MitarbeiterUndAbteilung ermöglicht. Sie können davon ausgehen, dass sowohl die neue Abteilung als auch der neue Mitarbeiter noch nicht existieren. Achten Sie jedoch auf die Primär- und Fremdschlüssel-Beziehungen!

Hilfsmittel Syntaxsammlung PL/SQL

Anonymer Block

DECLARE (optional)
Variablen, Cursor, benutzerdefinierte Exceptions
BEGIN (obligatorisch)
– SQL-Anweisungen
– PL/SQL-Anweisungen
EXCEPTION (optional)
Aktionen, die ausgeführt werden sollen, wenn Fehler auftreten
END; (obligatorisch)

PL/SQL Prozedur

```
CREATE [OR REPLACE] PROCEDURE <Prozedurname> [(<Parameterliste>)] IS
<LokaleVariablen>
BEGIN
<Prozedurrumpf>
END;
```

PL/SQL Funktion

```
CREATE [OR REPLACE] FUNCTION <Funktionsname>
(<Parameterliste>) RETURN <Ergebnistyp> IS
<LokaleVariablen>
BEGIN
<Funktionsrumpf>
RETURN <variable>
END;
```

IF Kontrollstruktur

```
IF <Bedingung>
THEN <Block>
[ ELSIF <Bedingung> THEN <Block> ]
...
[ ELSIF <Bedingung> THEN <Block> ]
[ ELSE <Block>]
END IF;
```

Zählschleife

```
FOR <Laufvariable> IN [REVERSE] <Start> .. <Ende>
LOOP
<Block>
END LOOP;
```

While-Schleife

```
WHILE <Bedingung>
LOOP
<Block>
END LOOP;
```

Exception abfangen

```
WHEN <Exceptiontyp1>
        THEN <Block1>
...
[WHEN <ExceptiontypN>
        THEN <BlockN>
        WHEN OTHERS
THEN <Block>]
```

Cursor definieren

```
CURSOR <Cursorname> [ (Parameterliste)] IS <Datenbankanfrage>;
```

Cursur iterieren

```
DECLARE
        CURSOR emp_cur IS SELECT ename FROM EMP;
BEGIN
        FOR myrec IN emp_cur
                LOOP

        dbms_output.put_line(myrec.ename);
         END LOOP;
END;
```

%Type und %ROWTYPE

Das Attribut %TYPE wird verwendet für die Variablendeklaration gemäß der Definition einer Datenbankspalte.

```
variable table.column%TYPE
```

Mit %ROWTYPE wird die Struktur einer Datenbanktabelle komplett übernommen. Dadurch kann die Variablendeklaration in einem Schritt erfolgen:

```
variable tabellenname%ROWTYPE;
```

Trigger

```
CREATE [OR REPLACE] TRIGGER <Triggername>
{BEFORE | AFTER}
{INSERT | DELETE | UPDATE} [OF {Spaltenliste}]
[OR {INSERT | DELETE | UPDATE} [OF {Spaltenliste}]]
...
[OR {INSERT | DELETE | UPDATE} [OF {Spaltenliste}]]
ON <Tabellenname>
[FOR EACH ROW]
[WHEN <Bedingung>]
<PL/SQL-Block>;
```

: NEW.Feldname und :OLD. Feldname : Alter und neuer Wert der entsprechenden Felder

Instead-of-Trigger

```
CREATE [OR REPLACE] TRIGGER <Triggername>
INSTEAD OF
{INSERT | DELETE | UPDATE} [OF {Spaltenliste}]
ON <Viewname>
[FOR EACH ROW]
<PL/SQL-Block>;
```

Ergebnis einer Abfrage (einzelner Wert) in PL/SQL Variable übertragen
Beispiel:

```
SELECT Tier.Tname
INTO ergebnis
FROM Tier
WHERE Tier.Gattung=gat;
```

Ergebnis einer ganzen Abfragen in PL/SQL TABLE übertragen
Beispiel:

```
DECLARE
        TYPE nr_liste IS TABLE OF emp.empno%TYPE;
        nr nr_liste;
        num number;
BEGIN
SELECT empno BULK COLLECT INTO nr FROM emp WHERE deptno = 20;
        FORALL i IN nr.FIRST .. nr.LAST
          loop
          UPDATE emp SET sal = sal* 1.1 WHERE empno = nr(i);
          end loop;
END;
```

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);
```