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 Supermärkte welche Produkte verkaufen. Dazu wurde auch noch erhoben, welche Personen bei welchem Supermarkt einkaufen und welche Personen welche Produkte bevorzugen. Dazu wurden die folgenden drei Tabellen in einer Datenbank angelegt und es werden auszugsweise Daten dargestellt:

Produktangebot

Supermarkt	**Produkt**
Aldi	Leberwurst
Aldi	Butter
Lidl	Butter
Aldi	Murmeln
Penny	Salami
Penny	Brot
Marktkauf	Leberwurst

Supermarktkunden

Kunde	**Supermarkt**
Müller	Aldi
Schulze	Aldi
Müller	Lidl
Schmidt	Marktkauf
Horst	Penny

Bevorzugt

Kunde	**Produkt**
Schulze	Leberwurst
Müller	Butter
Müller	Brot
Schmidt	Murmeln
Schmidt	Salami
Horst	Brot
Meier	Butter

a) Es ist folgendes bekannt: Der Marketingleiter von Aldi geht davon aus, dass wenn eine Kundenumfrage ergibt, dass ein beliebiger Kunde angibt ein Produkt zu bevorzugen und dieses in der Datenbank gespeichert wird, dann sollte dieses Produkt auch entsprechend angeboten werden. Dieses soll von der Datenbank berücksichtigt werden. Erklären Sie, mit welchem Instrument dies in der Datenbank implementiert werden kann. Schreiben sie dazu auch noch entsprechenden PL/SQL-Sourcecode. (6 Punkte)

Mittels Trigger.

```
CREATE OR REPLACE TRIGGER T
AFTER INSERT ON Bevorzugt
FOR EACH ROW
WHEN (:NEW.Supermarkt <> 'Aldi')
num NUMBER :=0;
BEGIN
  SELECT count(*) INTO num FROM Produktangebot WHERE Supermarkt='Aldi' AND
Produkt=:NEW.Produkt;
  IF(num=0)
    INSERT INTO Produktangebot VALUES ('Aldi', :NEW.Produkt);
 END IF;
END;
```

b) Schreiben Sie eine PL/SQL Prozedur, welche als Eingabeparameter den Namen eines Kunden erhält. Die Prozedur soll alle Produkte ermitteln, die der Kunde bevorzugt und jeweils einmal auf dem Bildschirm ausgeben. Zum Schluss soll die Prozedur noch ausgeben, wie viele Produkte ausgegeben wurden. (6 Punkte)

```
CREATE OR REPLACE PROCEDURE P
(Name VARCHAR2(30))
IS
CURSOR Produkte_cur IS SELECT DISTINCT Produkt
                       FROM Supermarktkunden BK, Bevorzugt B
                       WHERE BK.Supermarkt <> SupermarktName
                       AND BK.Kunde = B.Kunde
zaehler NUMBER := 0;
BEGIN
  FOR Produkt_rec IN Produkte_cur
  LOOP
    DBMS_OUTPUT.PUT_LINE(Produkt_rec.Produkt);
    zaehler := zaehler +1;
  END LOOP
  DBMS_OUTPUT.PUT_LINE('Es wurden ' || zaehler || ' Produkte ausgegeben');
END;
```

Aufgabe 2 (ca. 20min)

- Was versteht man unter "Löschweitergabe"? Wie wird dies bei SQL umgesetzt?
- Aus welchen syntaktischen Bausteinen (auch optional) besteht ein PL/SQL-Block?

- Welche verschiedenen Blocktypen kennen Sie in PL/SQL und worin unterscheiden sie sich?
- Was ist eine Materialized View und wozu benötigt man sie? Geben Sie ein Beispiel.
- Benennen Sie die Ebenen des DB-Tunings und erläutern Sie auf welchen Ebenen die größten Effekte zu vermuten sind?
- Erläutern die die Begriffe JDBC und ODBC und erklären Sie die konzeptionellen Unterschiede.
- Wie wird in Java eine Verbindung zu einer h2 Datenbank aufgebaut. Erläutern Sie die notwendigen Schritte und geben dazu auszugsweise die notwendigen Java-Klassen und die Methodenaufrufe an.
- Wie kann mit der Klasse ResultSet das Ergebnis einer Datenbankabfrage in Java verarbeitet werden? Wie können Sie auf die einzelnen Ergebnisse zugreifen. Erläutern Sie die notwendigen Schritte und geben dazu auszugsweise die notwendigen Java-Klassen und die Methodenaufrufe an.

Die Lösung können Sie jeweils dem Skript entnehmen.

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
…	…	…

```
CREATE OR REPLACE FUNCTION TrainingsplanFuellen
gesamtTage INTEGER
RETURN NUMBER(9, 2)
IS
Kilometer NUMBER(3,1) := 1;
Zeit NUMBER := 60;
summeKilometer NUMBER(9, 2);
BEGIN
  FOR Lauftag IN 1..gesamtTage
  LOOP
    INSERT INTO Trainingsplan VALUES (Lauftag, Kilometer, Zeit);
    Kilometer := Kilometer + 0.5;
    Zeit := Zeit + 20;
  END LOOP;
  SELECT SUM(KM) INTO summeKilometer FROM Trainingsplan;
   RETURN summeKilometer;
END;
```

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?

Die Fehlermeldung erscheint, weil der VIEW mehrere Tabellen miteinander verbindet und ich hier keine Datensätze einfügen kann. Oracle-Fehlermeldung: Über eine Join-View kann nicht mehr als eine Basistabelle modifiziert werden

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!

```
CREATE OR REPLACE TRIGGER aufgabe4
 INSTEAD OF INSERT ON MitarbeiterUndAbteilung
 FOR EACH ROW
   DECLARE neueAbteilungsNr NUMBER;
   BEGIN
    SELECT max(AbteilungsNr)+1
    INTO neueAbteilungsNr
    FROM Abteilung;
   INSERT INTO Abteilung VALUES (neueAbteilungsNr,:NEW.AbteilungsName,:NEW.Ort);
   INSERT INTO Mitarbeiter VALUES
(:NEW.MitarbeiterNr,:NEW.Nachname,:NEW.Vorname,neueAbteilungsNr);
 END;
    /
```

Aufgabe 5: SQL (20 Punkte)

Eine Datenbank für den Bestand von Bäumen hält fest, welche Baumtypen in welchem Park gepflanzt sind. Dabei sind die folgenden Tabellen gegeben:

Tabelle Park:

PARK_ID	NAME	FLAECHE
1	Nordpark	15000
2	Südpark	100000
3	Westpark	200000
4	Ostpark	150000

Tabelle Baum:

TYP_ID	SORTE	WISS_NAME
1	Apfelbaum	Malus domesticus
2	Birnbaum	Pyrus communis
3	Vogel-Kirsche	Prunus avium
4	Amerikanische Buche	Fagus grandifola
5	Stechpalme	Ilex aquifolium
6	Oregon-Eiche	Quercus garryana

Tabelle Pflanzung

PARK_ID	TYP_ID	MENGE	JAHR
1	1	10	2011
1	1	22	2012
1	2	5	2010
1	2	25	2012
1	4	2	2012
3	2	240	2011
3	3	230	2012
3	4	150	2011
3	1	442	2011
3	1	100	2012
2	1	100	2010
2	1	122	2011
2	2	12	2010
2	4	44	2012
4	1	400	2010
4	2	200	2012
4	4	50	2010
4	4	101	2012

Aufgaben:
Geben Sie in einem SQL-Statement folgende Informationen an:

a) Die Summe aller gepflanzten Apfelbäume
(4 Punkte)

```
SELECT SUM(Pflanzung.MENGE) FROM Baum, Pflanzung where Baum.typ_ID =
Pflanzung.Typ_ID and Baum.Sorte = 'Apfelbaum';
```

b) Die Baumsorten, die nie gepflanzt wurden (4 Punkte)

```
Select Baum.Sorte from Baum where not exists(
select Park_ID from Pflanzung where Pflanzung.Typ_id = baum.Typ_ID);
```

c) Geben Sie die Baumsorte, das Jahr und die Summe der in diesem Jahr gepflanzten Bäume pro Baumsorte und Jahr an. Die Ausgabe soll nach der Baumsorte und anschließend nach dem Jahr sortiert sein. (4 Punkte)

Es müssen nur gepflanzte Baumsorten und Jahre aufgeführt werden, in denen eine Baumsorte gepflanzt wurden.

```
Select B.Sorte, P.Jahr, sum(p.Menge)
from Baum B, Pflanzung P where
B.Typ_ID = P.Typ_ID
group by b.Sorte, P.Jahr
Order by b.Sorte, p.Jahr;
```

d) Die Namen der Parks mit der Summe der gepflanzten Bäume für die Parks, in denen mehr als 500 Bäume gepflanzt wurden.
Die Ausgabe soll nach der Summe der gepflanzten Bäume absteigend sortiert sein. (4 Punkte)

```
select Park.Name, Sum(Pflanzung.Menge)
from Park, Pflanzung
where Park.Park_ID = Pflanzung.Park_ID  group by Park.name
having sum(Pflanzung.Menge) > 500
order by sum(pflanzung.Menge) desc;
```

e) Eine Liste **aller** Baumsorten (nach Baumsorten aufsteigend sortiert) mit deren gepflanzten Mengen. Bäume, die nicht gepflanzt wurden, müssen in der Liste aufgeführt werden. (4 Punkte)

```
Select Baum.Sorte, Sum(Pflanzung.Menge)
from Baum left join Pflanzung
on Baum.Typ_ID = Pflanzung.Typ_ID
group by Baum.Sorte
order by Baum.Sorte asc;
```

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);
```