PERSONAL - A Cornucopia of Queries

DQL (Data Query Language)

- Studenten, die Fichte aus VL kennen

```
SELECT distinct s1.name
FROM studenten s1, studenten s2, hoeren h1, hoeren h2
WHERE s1.matrnr != s2.matrnr
AND s1.matrnr = h1.matrnr
AND s2.matrnr = h2.matrnr
AND h1.vorlnr = h2.vorlnr
AND s2.name = 'Fichte'
```

- Assistenten von Professoren, die Carnap unterrichtet haben

```
SELECT DISTINCT a.*
FROM assistenten a, professoren p, vorlesungen v, hoeren h, studenten s
WHERE s.name = 'Carnap'
AND s.matrnr = h.matrnr
AND h.vorlnr = v.vorlnr
AND v.gelesenvon = p.persnr
AND p.persnr = a.boss
```

- Name der Professoren, die Theophrastos aus VL kennen

```
SELECT DISTINCT p.name
FROM professoren p, vorlesungen v, hoeren h, studenten s
WHERE p.persnr = v.gelesenvon
AND h.vorlnr = v.vorlnr
AND s.matrnr = h.matrnr
AND s.name = 'Theophrastos'
```

- Vorlesungen, die von Studenten im Bachelorstudium gehört werden

```
SELECT DISTINCT v.*
FROM vorlesungen v, hoeren h, studenten s
WHERE v.vorlnr = h.vorlnr
AND s.matrnr = h.matrnr
AND s.semester < 7 -- or s.semester between 1 and 6
```

- Anzahl der VL, die jeder Student hört

```
SELECT s.matrnr, count(vorlnr) -- alt: SELECT s.name
FROM studenten s LEFT JOIN hoeren h ON s.matrnr = h.matrnr
GROUP BY s.matrnr -- alt: GROUP BY s.name
ORDER BY count DESC
```

- Anzahl der Hörer jeder VL

```
SELECT v.vorlnr, count(matrnr) AS hoerer
FROM vorlesungen v LEFT JOIN hoeren h ON v.vorlnr = h.vorlnr
GROUP BY v.vorlnr
ORDER BY hoerer DESC
```

- Anzahl der VL, die jeder Professor hält

```
SELECT p.name, count(vorlnr)
FROM professoren p LEFT OUTER JOIN vorlesungen v ON p.persnr = v.gelesenvon
GROUP BY p.name
```

- Durchschnittliches Semester der Studenten, die VL von Sokrates hören

```
WITH vl_von_sokrates AS (SELECT *
                         FROM vorlesungen v, professoren p
                         WHERE v.gelesenvon = p.persnr AND p.name =
'Sokrates'),
     studenten_von_sokrates AS (SELECT DISTINCT s.* -- CTE recommended for
distinct selection
                                FROM studenten s, hoeren h, vl_von_sokrates
v
                                WHERE s.matrnr = h.matrnr AND h.vorlnr =
v.vorlnr)

SELECT AVG(semester)
FROM studenten_von_sokrates
```

- Durchschnittliches Semester der Studenten (decimal & int)

```
SELECT avg(semester)
FROM studenten;

SELECT CAST(avg(semester) AS integer)
FROM studenten;
```

- Studenten, die noch nicht geprüft worden sind

```
WITH studentNoExam AS ((SELECT matrnr
                        FROM studenten)
                       EXCEPT
```

```
                        (SELECT matrnr
                         FROM pruefen))

SELECT s.*
FROM studenten s, studentNoExam n
WHERE s.matrnr = n.matrnr

-- with NOT EXISTS
SELECT *
FROM studenten s
WHERE NOT EXISTS(SELECT *
                 FROM pruefen p
                 WHERE s.matrnr = p.matrnr)
```

- Studenten, die geprüft worden sind

```
SELECT DISTINCT s.*
FROM studenten s, pruefen p
WHERE s.matrnr = p.matrnr
```

- Studenten, die geprüft worden sind (*fancy edition*)

```
WITH n AS ((SELECT matrnr -- matrnr der studenten
            FROM studenten)
           INTERSECT
           (SELECT matrnr
            FROM pruefen))

SELECT s.* -- namen der studenten
FROM studenten s, n
WHERE s.matrnr = n.matrnr
```

- Vorgänger zweiter Stufe von 5216 / 5052

```
SELECT v1.vorgaenger
FROM voraussetzen v1, voraussetzen v2
WHERE v2.nachfolger = 5216
AND v2.vorgaenger = v1.nachfolger;

SELECT DISTINCT v1.vorgaenger -- beide 5001 als Vorgänger
FROM voraussetzen v1, voraussetzen v2
WHERE v2.nachfolger = 5052
AND v2.vorgaenger = v1.nachfolger
```

- *REKURSIV*: Alle Vorgänger der Vorlesung mit ID 5259 / mit Namen "Der Wiener Kreis"

```
with recursive vl_hierarchy as ((select nachfolger, vorgaenger, 1 as lvl
                                 from voraussetzen
                                 where nachfolger = 5259)
                                 union
                                 (select v.nachfolger, v.vorgaenger, h.lvl +
1 as lvl
                                 from vl_hierarchy h join voraussetzen v on
h.vorgaenger = v.nachfolger))

select * from vl_hierarchy
```

```
-- title version
with recursive vl_hierarchy as ((select vs.nachfolger, vs.vorgaenger, 1 as
lvl
                                 from voraussetzen vs, vorlesungen vl
                                 where vs.nachfolger = vl.vorlnr and vl.titel
= 'Der Wiener Kreis')
                                 union
                                 (select v.nachfolger, v.vorgaenger, h.lvl +
1 as lvl
                                 from vl_hierarchy h join voraussetzen v on
h.vorgaenger = v.nachfolger))

select * from vl_hierarchy
```

```
-- alternative
WITH RECURSIVE vor AS ((SELECT vorgaenger, 1 as lvl
                        FROM voraussetzen
                        WHERE nachfolger = (SELECT vorlnr
                                            FROM vorlesungen
                                            WHERE titel = 'Der Wiener Kreis'))
UNION
                       (SELECT v.vorgaenger, rec.lvl + 1 as lvl
                        FROM voraussetzen v, vor rec
                        WHERE rec.vorgaenger = v.nachfolger
                       ))

SELECT v.*
FROM vorlesungen v, vor
WHERE vor.vorgaenger = v.vorlnr
```

- *REKURSIV*: Vorgänger der Vorlesung "Der Wiener Kreis" mit Level 3 (*drei Semester vorher*)

```
with recursive vl_hierarchy as ((select vs.nachfolger, vs.vorgaenger, 1 as
lvl
```

```
                                from voraussetzen vs, vorlesungen vl
                                where vs.nachfolger = vl.vorlnr and vl.titel
= 'Der Wiener Kreis')
                                union
                                (select v.nachfolger, v.vorgaenger, h.lvl +
1 as lvl
                                from vl_hierarchy h join voraussetzen v on
h.vorgaenger = v.nachfolger))

select * from vl_hierarchy where lvl = 3
```

- Vorlesungen, die erst ab dem 3. Semester gehört werden sollen

```
SELECT *
FROM vorlesungen v, voraussetzen v1, voraussetzen v2
WHERE v.vorlnr = v1.nachfolger AND v1.vorgaenger = v2.nachfolger
```

- *REKURSIV*: Vorlesungen, die erst ab dem 3. Sem. gehört werden sollen

```
with recursive hierarchy as ((select v1.vorgaenger, v1.nachfolger, 2 as sem
-- finde vl aus dem 1 sem
                              from voraussetzen v1 left outer join
voraussetzen v2 on v1.vorgaenger = v2.nachfolger
                              where v2.vorgaenger is null)
                              union
                             (select v.vorgaenger, v.nachfolger, h.sem + 1
as sem
                              from voraussetzen v, hierarchy h
                              where h.nachfolger = v.vorgaenger))

select distinct nachfolger, sem
from hierarchy
where sem > 2
```

- *REKURSIV*: Nachfolger aller Vorlesungen ab dem 1. Sem.

```
WITH RECURSIVE nach AS ((SELECT nachfolger, 1 as lvl
                         FROM voraussetzen
                         WHERE vorgaenger = 5001)
                        UNION
                        (SELECT v.nachfolger, nach.lvl + 1 as lvl
                         FROM voraussetzen v, nach
                         WHERE nach.nachfolger = v.vorgaenger))
```

```
SELECT *
FROM nach
```

- REKURSIV: Minimales Semester für "Der Wiener Kreis"

```
WITH RECURSIVE vor AS (SELECT v.vorgaenger, 1 as lvl
                       FROM voraussetzen v, vorlesungen vl
                       WHERE v.nachfolger = vl.vorlnr
                       AND vl.titel = 'Der Wiener Kreis'
                       UNION
                       SELECT v.vorgaenger, vor.lvl + 1 as lvl
                       FROM vor, voraussetzen v
                       WHERE vor.vorgaenger = v.nachfolger
                      )

SELECT max(lvl)
FROM vor
```

- Vorlesungen, die keine Hörer haben

```
SELECT *
FROM vorlesungen v
WHERE NOT EXISTS(SELECT *
                 FROM hoeren h
                 WHERE v.vorlnr = h.vorlnr)
```

- Männer-/Frauenanteil pro Fakultät

```
WITH anz AS (SELECT s.fakultaet, count(*) AS totalAnz
             FROM studentengf s
             GROUP BY s.fakultaet),
     anzW AS (SELECT s2.fakultaet, count(*) AS anzWeib
              FROM studentengf s2
              WHERE s2.geschlecht = 'W'
              GROUP BY s2.fakultaet),
     anzM AS (SELECT s3.fakultaet, count(*) AS anzMann
              FROM studentengf s3
              WHERE s3.geschlecht = 'M'
              GROUP BY s3.fakultaet)

SELECT anz.Fakultaet, anz.totalAnz, anzw.anzWeib, (1.00 * anzw.anzWeib /
anz.totalAnz * 100) AS procWeib
FROM anz, anzw
WHERE anz.Fakultaet = anzw.Fakultaet;
```

```
SELECT anz.Fakultaet, anz.totalAnz, coalesce(anzm.anzMann, 0) AS anzMann,
coalesce((cast(anzm.anzMann as decimal(5,2))/anz.totalAnz * 100), 0) AS
procMann
FROM anz LEFT JOIN anzm ON anz.fakultaet = anzm.fakultaet

-- quick method
SELECT Fakultaet, (sum(case when Geschlecht = 'M' then 1.00 else 0.00 end))
/ count(*) * 100
FROM StudentenGF
GROUP BY Fakultaet
```

- Zehnkämpfer, die in ALLEN Disziplinen besser als Bolt waren

```
-- with "for all" -> "not exists"
SELECT DISTINCT a.name
FROM zehnkampfd a
WHERE NOT EXISTS(
  SELECT *
  FROM zehnkampfd a2
  WHERE a.name = a2.name
  AND EXISTS(
    SELECT *
    FROM zehnkampfd b
    WHERE b.name = 'Bolt' AND b.disziplin = a2.disziplin AND b.punkte >=
a2.punkte))

-- with counting
WITH disziplinen(anzahl) AS (SELECT count(distinct disziplin)
                             FROM zehnkampfd),
     betterThanBolt AS (SELECT a.name, a.disziplin
                        FROM zehnkampfd a, zehnkampfd b
                        WHERE b.name = 'Bolt' AND a.disziplin = b.disziplin
AND a.punkte > b.punkte)

SELECT b.name
FROM betterThanBolt b
GROUP BY name
HAVING count(name) = (SELECT anzahl FROM disziplinen)
```

- Silbermedalliengewinner

```
WITH gesamtpunkte AS (SELECT name, sum(punkte) as punkte
                      FROM zehnkampfd
                      GROUP BY name)
```

```
SELECT name
FROM gesamtpunkte silber
WHERE EXISTS(
  SELECT *
  FROM gesamtpunkte gold
  WHERE gold.name != silber.name AND gold.punkte > silber.punkte
  AND NOT EXISTS(
    SELECT *
    FROM gesamtpunkte rest
    WHERE rest.name != gold.name AND rest.punkte > silber.punkte))

-- counting
WITH gesamtpunkte AS (SELECT name, sum(punkte) as punkte
                      FROM zehnkampfd
                      GROUP BY name)

SELECT silber.name
FROM gesamtpunkte silber, gesamtpunkte gold
WHERE silber.punkte < gold.punkte
GROUP BY silber.name
HAVING count(gold.name) = 1 -- can also be gold.punkte, or *
```

- Studenten, die mehr als die durchschnittliche Anzahl von SWS belegt haben

```
WITH SWSproStudent AS (SELECT s.matrnr, coalesce(sum(sws),0) as summation
                       FROM studenten s
                       LEFT OUTER JOIN hoeren h ON s.matrnr = h.matrnr
                       LEFT OUTER JOIN vorlesungen v ON h.vorlnr = v.vorlnr
                       GROUP BY s.matrnr)

SELECT s.*
FROM studenten s JOIN SWSproStudent sws ON s.matrnr = sws.matrnr
WHERE sws.summation > (SELECT avg(summation) FROM SWSproStudent)
```

DML (Data Manipulation Language)

- Alle Professoren, die den Rang C3 haben, werden auf den Rang C4 befördert

```
UPDATE Professoren
SET Rang = 'C4' WHERE Rang = 'C3'
```

- Löschen aller Assistenten mit dem Fachgebiet "Planetenbewegung"

```
DELETE FROM Assistenten WHERE fachgebiet = 'Planetenbewegung'
```

- Erstellen einer neuen VL mit dem Namen „Grundlagen: Datenbanken“, ID 5278, gehalten von Curie, mit "Logik" als Voraussetzung, 4 SWS und mit Student mit MatrNr 28106 als Hörer.

```
INSERT INTO Vorlesungen
SELECT 5278, 'Grundlagen: Datenbanken', 4, persnr
FROM professoren
WHERE name = 'Curie';

INSERT INTO voraussetzen
SELECT vorlnr, 5278
FROM vorlesungen
WHERE titel = 'Logik';

INSERT INTO hoeren VALUES(28106, 5278);
```

DDL (Data Definition Language)

```
-- example: twitter
CREATE TABLE twt_user(
    id integer not null primary key,
    name varchar(20) not null unique
);

CREATE TABLE tweet(
    id integer not null primary key,
    user_id integer references twt_user on delete set null,
    twt_date timestamp not null,
    text varchar(300) not null,
    check(user_id IS NOT NULL or text = 'removed')
);

CREATE TABLE follows(
    follower_id integer not null references twt_user on delete cascade,
    following_id integer not null references twt_user on delete cascade,
    primary key(follower_id, following_id)
);

CREATE TABLE likes(
    user_id integer not null references twt_user on delete cascade,
    twt_id integer not null references tweet on delete cascade,
    like_date timestamp not null,
    primary key(user_id, twt_id)
    check exists(
```

```
        SELECT *
        FROM Tweet t
        WHERE t.id = twt_id AND t.twt_date <= like_date
    )
);
```