SQL

DDL (Data Definition Language) Befehle und DML(Data Manipulation Language)

DDL (Data Definition Language) Befehle

CREATE TABLE- Anlegen einer Relation

```
CREATE TABLE table-name (column-definition-list)
```
wobei `column-definition-list = attribut-name type [NOT NULL]`

Beispiel:

```
CREATE TABLE Kurse
(KursId VARCHAR(20),
 Titel VARCHAR(50),
 ECTS INTEGER)

CREATE TABLE Enrolled
(MatrNr CHAR(20),
 KursId CHAR(20),
 Note REAL)
```

DROP TABLE – Relationenschema löschen

```
DROP TABLE table-name
```

Bsp.

```
DROP TABLE Kurse
```

Primärschlüssel und Fremdschlüssel

- beim Anlegen einer Relation können auch Primärschlüssel und Fremdschlüssel angegeben werden

```
CREATE TABLE Enrolled
(MatrNr CHAR(20),
 KursId CHAR(20),
 Note REAL,
 PRIMARY KEY (MatrNr, KursId),
 CONSTRAINT FK_Enrolled_Studenten FOREIGN KEY (MatrNr)
REFERENCES Studenten,
 FOREIGN KEY (KursId) REFERENCES Kurse)
```

- In SQL können auch Kandidatschlüssel definiert werden mit Hilfe von UNIQUE.

DML(Data Manipulation Language)
Einfügen, Löschen und Ändern von Tupeln

Einfügen von Tupeln

```
INSERT INTO table-name [(column-list)]
VALUES (values-list)
```

Bsp.

```
INSERT INTO Kurse (KursId, Titel, ECTS)
VALUES ('Alg1', 'Algebra 1', 5)
```

Bulk – Einfügen

```
INSERT INTO table-name [(column-list)]
<select statement>
```

Bsp.

```
INSERT INTO Enrolled (MatrNr, KursId, Note)
SELECT MatrNr, 'Alg1', 10
FROM Studenten
```

Einfügen

Spalten und Werte müssen nicht angegeben werden, wenn :

- NULL-Werte erlaubt sind oder
- DEFAULT-Werte gesetzt sind oder
- AUTO-INCREMENT angegeben wurde

Bsp.

- Titel kann NULL sein
- ECTS nicht angegeben → Default-Wert 0

```
INSERT INTO Kurse (KursId)
VALUES (‘Alg3’)
```

Beispiel AUTO-INCREMENT

- die Tabelle Reviews hat einen AUTO-INCREMENT Id
- AUTO-INCREMENT in MS SQL Server → IDENTITY(1,1)

```
CREATE TABLE Reviews
(ID int IDENTITY(1,1) PRIMARY KEY,
ReviewContent char(500))
```

Löschen und Aktualisieren
DELETE – Löschen von Tupeln

```
DELETE FROM table-name
[WHERE condition]
```

Bsp.

```
DELETE FROM Studenten S
WHERE S.Name = ‘Schmidt’
```

UPDATE – Verändern von Tupeln

```
UPDATE table-name
SET column-name = expression
[,column-name2 = expression2]
[WHERE condition]
```

Bsp.

```
UPDATE Studenten S
SET S.age = S.age + 1
WHERE S.MatrNr = 123
```

DML(Data Manipulation Language)
SQL Abfragen

Studenten

MatrNr	Name	Vorname	Email	Age	Gruppe
1234	Schmidt	Hans	schmidt@cs.ro	21	331
1235	Meisel	Amelie	meisel@cs.ro	22	331
1236	Krause	Julia	krause@cs.ro	21	332
1237	Rasch	Lara	rasch@cs.ro	21	331
1238	Schmidt	Christian	schmidtC@cs.ro	22	332

Kurse

KursId	Titel	ECTS
Alg1	Algorithmen1	6
DB1	Datenbanken1	6
DB2	Datenbanken2	5

Enrolled

MatrNr	KursId	Note
1234	Alg1	7
1235	Alg1	8
1234	DB1	9
1234	DB2	7
1236	DB1	10
1237	DB2	10

SELECT

```
SELECT *
FROM Studenten S
WHERE S.Age = 21
```

- gibt alle 21-jährige Studenten aus:

1234	Schmidt	Hans	schmidt@cs.ro	21	331
1236	Krause	Julia	krause@cs.ro	21	332
1237	Rasch	Lara	rasch@cs.ro	21	331

SELECT

- Um nur die Namen und Email Adressen auszugeben:

```
SELECT * → SELECT S.Name, S.Vorname, S.Email
```

Schmidt	Hans	schmidt@cs.ro
Krause	Julia	krause@cs.ro
Rasch	Lara	rasch@cs.ro

Wir wollen üben!

- https://www.sqlclimber.com

- Übung - Basic select: **Select specified columns 1**

Was gibt die folgende Abfrage aus?

```
SELECT S.Name, E.KursId
FROM Studenten S, Enrolled E
WHERE S.MatrNr=E.MatrNr AND E.Note=10
```

Krause	DB1
Rasch	DB2

SELECT Klausel

```
SELECT [DISTINCT] target-list
FROM relation-list
WHERE qualification
```

wobei:

- *relation-list* – eine Liste von Relationen
- *target-list* – Liste von Attributen aus der Relation in *relation-list*
- *qualification* – Bedingungen mit Vergleichsoperatoren (<,>,=,...) und logischen Operatoren (AND, OR, NOT)
- *DISTINCT* – ist optional, eliminiert Duplikate aus dem Ergebnis

Achtung!

Die zwei Abfragen

```
SELECT S.Name, E.KursId
FROM Studenten S, Enrolled E
WHERE S.MatrNr=E.MatrNr AND E.Note=10
```

und

```
SELECT Name, KursId
FROM Studenten, Enrolled
WHERE Studenten.MatrNr=Enrolled.MatrNr AND  Enrolled.Note=10
```

sind äquivalent.

- *Range variables* (S,E) sind nötig wenn die gleiche Relation zwei mal in der FROM Klausel erscheint. Aber, es ist gut immer *range variables* zu benutzen.

Wir wollen üben!

- https://www.sqlclimber.com

- Übung - Basic select: **Column alias 1**

Finde Studenten, die wenigstens eine Note haben.

```
SELECT S.MatrNr
FROM Studenten S, Enrolled E
WHERE S.MatrNr=E.MatrNr
```

- Würde **DISTINCT** einen Unterschied machen?
- Was passiert, wenn wir anstatt MatrNr den Namen ausgeben? Brauchen wir **DISTINCT** dann?

Eine Abfrage mit LIKE-Bedingung und arithmetische Ausdrücke

```
SELECT S.age, age1 = S.age-5, 2*S.age AS age2
FROM Studenten S
WHERE S.Name LIKE 'B_%B'
```

- Die **LIKE**-Bedingung vergleicht Zeichenketten „ungenau“. Dazu werden Wildcards benutzt:
  - Der Unterstrich '_' steht für ein beliebiges einzelnes Zeichen, das an der betreffenden Stelle vorkommen kann
  - Das Prozentzeichen '%' steht für eine beliebige Zeichenkette mit 0 oder mehr Zeichen
- S.Name LIKE ‘B_%B’ – der Name beginnt und endet mit B und enthält wenigstenst 3 Buchstaben
- „=“ und „AS“ haben die gleiche Rolle hier

Wir wollen üben!

- https://www.sqlclimber.com

- Übung - Basic select: **Condition LIKE 2**

UNION

- Vereinigung zweier Relationen, die kompatibel Wertebereiche haben; Duplikate werden eliminiert
- Z.B. Geben sie Studenten aus, die Noten in einem 5 ECTS oder in einem 6 ECTS Kurs haben

```
SELECT E.MatrNr
FROM Enrolled E, Kurse K
WHERE E.KursId = K.KursId
AND K.ECTS = 5
UNION
SELECT E.MatrNr
FROM Enrolled E, Kurse K
WHERE E.KursId = K.KursId
AND K.ECTS = 6
```

Alternative Abfrage:

```
SELECT E.MatrNr
FROM Enrolled E,Kurse K
WHERE E.KursId = K.KursId
AND (K.ECTS = 5 OR
     K.ECTS = 6)
```

INTERSECT

- Was passiert wenn wir “oder” mit “und” ersetzen?
- Gebe die Studenten aus, die Noten in einem 5 ECTS und in einem 6 ECTS Kurs haben
- INTERSECT = Durchschnitt zweier Relationen, die kompatibel Wertebereiche haben

INTERSECT

```
SELECT E.MatrNr
FROM Enrolled E, Kurse K
WHERE E.KursId = K.KursId
AND K.ECTS = 5
INTERSECT
SELECT E.MatrNr
FROM Enrolled E, Kurse K
WHERE E.KursId = K.KursId
AND K.ECTS = 6
```

Alternative:

```
SELECT E1.MatrNr
FROM Kurse K1, Enrolled E1,
      Kurse K2, Enrolled E2
WHERE E1.MatrNr = E2.MatrNr AND
E1.KursId = K1.KursId AND
E2.KursId = K2.KursId AND
K1.ECTS = 5 AND
K2.ECTS = 6
```

EXCEPT

Gibt alle Studenten aus, die in ‘Datenbank II’ angemeldet sind, aber nicht in ‘Datenbank I’:

```
SELECT E.MatrNr
FROM Enrolled E, Kurse K
WHERE E.KursId = K.KursId
AND K.Titel = ‘Datenbanken II’
EXCEPT
SELECT E.MatrNr
FROM Enrolled E, Kurse K
WHERE E.KursId = K.KursId
AND K.Titel = ‘Datenbanken I’
```

Nested Queries (Verschachtelte Abfragen)

- Eine WHERE Klausel kann in einer anderen Abfrage enthalten sein.
- Gebe die Namen der Studenten aus, die für den Kurs ‘Alg1’ angemeldet sind

```
SELECT S.Name
FROM Studenten S
WHERE S.MatrNr IN (SELECT E.MatrNr
                   FROM Enrolled E
                   WHERE E.KursId = ’Alg1’)
```

Nested Queries (Verschachtelte Abfragen)

- Alternative Abfrage:

```
SELECT S.Name
FROM Studenten S
WHERE EXISTS (SELECT *
              FROM Enrolled E
              WHERE E.MatrNr = S.MatrNr
              AND E.KursId = 'Alg1')
```

- **EXISTS** und **IN** sind Vergleichsoperatoren für Mengen.
- Wir können auch **NOT IN** und **NOT EXISTS** benutzen.

ANY, ALL

- **ANY** – das Ergebnis ist True (wahr) wenn die Bedingung True ist für **wenigstens ein** Element aus der Ergebnis der Unterabfrage (sub-query)
- **ALL** – das Ergebnis ist True (wahr) wenn die Bedingung True ist für **alle** Elemente aus der Ergebnis der Unterabfrage (sub-query)

Gebe alle Studenten aus, die älter sind als ein Student mit dem Namen „Hans“

```
SELECT *
FROM Studenten S
WHERE S.age > ANY (SELECT S2.age
                   FROM Studenten S2
                   WHERE S2.Name='Hans')
```

Gebe die Studenten aus, die Noten in einem 5 ECTS und in einem 6 ECTS Kurs haben

```
SELECT E.MatrNr
FROM Enrolled E, Kurse K
WHERE E.KursId = K.KursId
AND K.ECTS = 5
INTERSECT
SELECT E.MatrNr
FROM Enrolled E, Kurse K
WHERE E.KursId = K.KursId
AND K.ECTS = 6
```

- **INTERSECT** Abfragen können mit **IN** umgeschrieben werden:

```
SELECT E.MatrNr
FROM Enrolled E, Kurse K
WHERE E.KursId = K.KursId
AND K.ECTS = 5
AND E.MatrNr IN (SELECT E2.MatrNr
                 FROM Enrolled E2, Kurse K2
                 WHERE E2.KursId = K2.KursId
                 AND K.ECTS = 6)
```

- Ähnlich kann man **EXCEPT** Abfragen mit **NOT IN** umschreiben.

EXCEPT

Gibt alle Studenten aus, die in ‘Datenbank II’ angemeldet sind, aber nicht in ‘Datenbank I’:

```
SELECT E.MatrNr
FROM Enrolled E, Kurse K
WHERE E.KursId = K.KursId
AND K.Titel = ‘Datenbanken II’
AND E.MatrNr NOT IN (SELECT E.MatrNr
                     FROM Enrolled E, Kurse K
                     WHERE E.KursId = K.KursId
                     AND K.Titel = ‘Datenbanken I’ )
```

Übungen

Datenbank

Studenten(MatrNr, Name, Vorname, Email, Age, Gruppe)

Kurse(KursId, Titel, ECTS)

Enrolled(MatrNr, KursId, Note)

1. Gebe alle Studenten aus der Gruppe 331 aus.

```
select *
from Studenten s
where s.gruppe = 331
```

2. Gebe alle Studenten aus, die in der Gruppe 331 oder 332 sind.

```
select *
from Studenten s
where s.gruppe = 331 OR s.gruppe= 332
```

3. Gebe alle Studenten aus, deren Name mit „An“ anfängt.

```
select *
from Studenten s
where s.name like 'An%'
```

- Gebe alle Studenten aus, deren Name mit „An“ anfängt und genau vier Buchstaben hat.

```
select *
from Studenten s
where s.name like 'An__'
```

4. Gebe alle Studenten aus (MatrNr), die in dem Kurs “Alg1” und “DB1” angemeldet sind.

- Methode I.

```
select e1.MatrNr
from Enrolled e1
where e1.kursId = ‘Alg1’
INTERSECT
select e2.MatrNr
from Enrolled e2
where e2.kursId = ‘DB1’
```

- Methode II.

```
select e1.MatrNr
from Enrolled e1, Enrolled e2
where e1.KursId = ‘Alg1’ and
e2.KursId = ‘DB1’ and
e1.MatrNr = e2.MatrNr
```

5. Gebe alle Studenten aus mit Alter >= allen anderen mit dem Namen ‘A....N‘(die am ältesten sind)

```
select *
from Studenten s
where s.age >= ALL (select s2.age
                    from studenten s2
                    where s2.name like 'A%N')
```

6. Gebe die Emails aller Studenten aus dem Kurs „DB1“ aus. – 3 Methoden

- Methode I.

```
select s.email
from studenten s
where s.matrNr in (SELECT e.MatrNr
                   from enrolled e
                   where e.KursId = 'DB1')
```

- Methode II.

```
select s.email
from studenten s
where exists (select *
       from enrolled e
       where e.MatrNr = s.MatrNr
       and e.KursId = 'DB1')
```

6. Gebe die Emails aller Studenten aus dem Kurs „DB1“ aus. – 3 Methoden

- Methode III.

```
select s.email
from studenten s, enrolled e
where s.MatrNr = e.MatrNr and e.KursId ='DB1'
```