

SQL

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

DDL (Data Definition Language) Befehle

ALTER TABLE – Ändern einer Relation

- es gibt viele mögliche Änderungen an das Relationenschema

Beispiele:

- ein neues Attribut hinzufügen

```
ALTER TABLE table-name  
ADD column-name type
```

Bsp.:

```
ALTER TABLE Studenten  
ADD erstesJahr INTEGER
```

Beachte: Werte des neuen Attributes bestehender Tupel werden mit Nullwerten belegt

ALTER TABLE – Ändern einer Relation

- Datentyp eines Attributes ändern

```
ALTER TABLE table-name
```

```
ALTER column-name datatype
```

- ein Attribut löschen

```
ALTER TABLE table-name
```

```
DROP COLUMN column-name
```

Referenz-Integritätsregel

ON DELETE/UPDATE:

- NO ACTION – Tupel wird nicht gelöscht (default Lösung)
- CASCADE – rekursives Löschen
- SET NULL/SET DEFAULT – Nullsetzen aller darauf verweisender Fremdschlüssel. Kann nur verwendet werden, wenn Null-Werte für das Attribut erlaubt sind

Referenz-Integritätsregel

```
CREATE TABLE Kurse
(KursId CHAR(5) PRIMARY KEY,
Titel VARCHAR(20),
ECTS int,
ProfessorId int,
FOREIGN KEY (ProfessorId) REFERENCES Professor
ON DELETE SET NULL
ON UPDATE CASCADE)
```

Default Constraint

Wenn das Wert eines Attributes nicht explizit einen Wert bekommt, wird anstatt einen Null-Wert ein Default-Wert dafür genommen.

```
ALTER TABLE table-name  
ADD CONSTRAINT constraint-name  
DEFAULT value FOR column
```

Bsp.

```
ALTER TABLE Enrolled  
ADD CONSTRAINT defaultNote  
DEFAULT 1 FOR Note
```

Integritätsregeln

```
CREATE TABLE Studenten
(MatNr CHAR(20),
Name CHAR(50),
Vorname CHAR(50),
Email CHAR(30),
Age INTEGER,
Gruppe INTEGER,
PRIMARY KEY (MatNr),
CONSTRAINT ageInterval
CHECK (age >= 18
      AND age <= 70))
```


Constraints löschen

```
ALTER TABLE table-name
```

```
DROP CONSTRAINT constraint-name
```

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
Ag	Algebra	5

Enrolled

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

JOIN Abfragen

Join Typ	Abfrage	Ergebnis																				
INNER JOIN	SELECT S.Name, K.Titel FROM Studenten S INNER JOIN Enrolled E ON S.MatrNr = E.MatrNr INNER JOIN Kurse K ON E.KursId = K.KursId		<table><tr><th>Name</th><th>Titel</th></tr><tr><td>Schmidt</td><td>Algorithmen1</td></tr><tr><td>Schmidt</td><td>Datenbanken1</td></tr><tr><td>Schmidt</td><td>Datenbanken2</td></tr><tr><td>Meisel</td><td>Algorithmen1</td></tr><tr><td>Krause</td><td>Datenbanken1</td></tr><tr><td>Rasch</td><td>Datenbanken2</td></tr></table>	Name	Titel	Schmidt	Algorithmen1	Schmidt	Datenbanken1	Schmidt	Datenbanken2	Meisel	Algorithmen1	Krause	Datenbanken1	Rasch	Datenbanken2					
Name	Titel																					
Schmidt	Algorithmen1																					
Schmidt	Datenbanken1																					
Schmidt	Datenbanken2																					
Meisel	Algorithmen1																					
Krause	Datenbanken1																					
Rasch	Datenbanken2																					
LEFT OUTER JOIN (Studenten, die nie für einen Kurs angemeldet waren) Alle Tupel aus der linken Relation, die keinen Join-Partner in der rechten Relation haben, werden trotzdem ausgegeben	SELECT S.Name,K.Titel FROM Studenten S LEFT OUTER JOIN Enrolled E ON S.MatrNr = E.MatrNr LEFT OUTER JOIN Kurse K ON E.KursId=K.KursId		<table><tr><th>Name</th><th>Titel</th></tr><tr><td>Schmidt</td><td>Algorithmen1</td></tr><tr><td>Schmidt</td><td>Datenbanken1</td></tr><tr><td>Schmidt</td><td>Datenbanken2</td></tr><tr><td>Meisel</td><td>Algorithmen1</td></tr><tr><td>Krause</td><td>Datenbanken1</td></tr><tr><td>Rasch</td><td>Datenbanken2</td></tr><tr><td>Schmidt</td><td>NULL</td></tr></table>	Name	Titel	Schmidt	Algorithmen1	Schmidt	Datenbanken1	Schmidt	Datenbanken2	Meisel	Algorithmen1	Krause	Datenbanken1	Rasch	Datenbanken2	Schmidt	NULL			
Name	Titel																					
Schmidt	Algorithmen1																					
Schmidt	Datenbanken1																					
Schmidt	Datenbanken2																					
Meisel	Algorithmen1																					
Krause	Datenbanken1																					
Rasch	Datenbanken2																					
Schmidt	NULL																					
RIGHT OUTER JOIN (Finde alle Noten, die falsch eingetragen wurden/zu keinem Studenten gehören)	SELECT S.Name,K.Titel FROM Studenten S RIGHT OUTER JOIN Enrolled E ON S.MatrNr = E.MatrNr RIGHT OUTER JOIN Kurse K ON E.KursId=K.KursId		<table><tr><th>Name</th><th>Titel</th></tr><tr><td>NULL</td><td>Algebra</td></tr><tr><td>Schmidt</td><td>Algorithmen1</td></tr><tr><td>Meisel</td><td>Algorithmen1</td></tr><tr><td>Schmidt</td><td>Datenbanken1</td></tr><tr><td>Krause</td><td>Datenbanken1</td></tr><tr><td>Schmidt</td><td>Datenbanken2</td></tr><tr><td>Rasch</td><td>Datenbanken2</td></tr></table>	Name	Titel	NULL	Algebra	Schmidt	Algorithmen1	Meisel	Algorithmen1	Schmidt	Datenbanken1	Krause	Datenbanken1	Schmidt	Datenbanken2	Rasch	Datenbanken2			
Name	Titel																					
NULL	Algebra																					
Schmidt	Algorithmen1																					
Meisel	Algorithmen1																					
Schmidt	Datenbanken1																					
Krause	Datenbanken1																					
Schmidt	Datenbanken2																					
Rasch	Datenbanken2																					
FULL OUTER JOIN (LEFT + RIGHT OUTER JOIN)	SELECT S.Name,K.Titel FROM Studenten S FULL OUTER JOIN Enrolled E ON S.MatrNr = E.MatrNr FULL OUTER JOIN Kurse K ON E.KursId=K.KursId		<table><tr><th>Name</th><th>Titel</th></tr><tr><td>Schmidt</td><td>Algorithmen1</td></tr><tr><td>Schmidt</td><td>Datenbanken1</td></tr><tr><td>Schmidt</td><td>Datenbanken2</td></tr><tr><td>Meisel</td><td>Algorithmen1</td></tr><tr><td>Krause</td><td>Datenbanken1</td></tr><tr><td>Rasch</td><td>Datenbanken2</td></tr><tr><td>Schmidt</td><td>NULL</td></tr><tr><td>NULL</td><td>Algebra</td></tr></table>	Name	Titel	Schmidt	Algorithmen1	Schmidt	Datenbanken1	Schmidt	Datenbanken2	Meisel	Algorithmen1	Krause	Datenbanken1	Rasch	Datenbanken2	Schmidt	NULL	NULL	Algebra	
Name	Titel																					
Schmidt	Algorithmen1																					
Schmidt	Datenbanken1																					
Schmidt	Datenbanken2																					
Meisel	Algorithmen1																					
Krause	Datenbanken1																					
Rasch	Datenbanken2																					
Schmidt	NULL																					
NULL	Algebra																					

NULL Werte

- Manchmal sind die Werte für bestimmte Attribute in einem Tupel unbekannt/unknown oder inapplicable (nicht anwendbar). Dann werden diese mit NULL bezeichnet.
- Wenn eine Tabelle NULL Werte enthält werden viele Sachen komplizierter:
 - man muss bestimmte Operatoren benutzen, um zu prüfen ob ein Wert Null ist oder nicht
 - Wie sollte die Bedingung `age>8` ausgewertet werden wenn age Null ist? Was passiert für AND, OR und NOT
- Lösung: wir brauchen 3-valued Logik : **true, false, unknown**
- Wir müssen manchmal die Nullwerte extra raussuchen, um sie zu beseitigen für eine Abfrage.
- Outer Joins können benutzt werden, um Null Werte rauszusuchen.

Aggregatfunktionen

- werden auf eine Menge von Tupeln angewendet
 - Verdichtung einzelnen Tupeln zu einem Gesamtwert
 - SUM, AVG, MIN, MAX können nur auf Zahlen angewendet werden
-
- **SUM (X)** → 12
 - **AVG(X)** → 3
 - **MAX(X)** → 6
 - **MIN(X)** → 1
 - **COUNT(X)** → 4
 - **Duplikat-Eliminierung:** **COUNT(DISTINCT X)** → 3
 - **Behandlung von Null-Werten:** **COUNT(X)** zählt jeweils nur die Anzahl von Werten in X, die von NULL verschieden sind

X
1
1
4
6

Aggregation - GROUP BY und HAVING

- Anwendung: wenn wir Tupeln gruppieren wollen um Aggregatfunktionen auf bestimmte Gruppen anzuwenden
- z.B. Finde das Alter des jüngsten Studenten aus jeder Gruppe
 - wir wissen nicht wie viele Gruppen es gibt
 - es muss generell funktionieren, nicht nur für die Gruppen, die jetzt in der Tabelle existieren

Anfragen mit GROUP BY und HAVING

- Basisschema:

```
SELECT [DISTINCT] target-list  
FROM relation-list  
WHERE condition  
GROUP BY grouping-list  
HAVING group-condition
```


Aufpassen!

- Alle Spalten bei **SELECT**, die nicht in einem Aggregat-Ausdruck (mit **SUM()**, **COUNT()** etc.) auftauchen, müssen in der **GROUP BY**-Klausel stehen
- D.h. `target-list` kann Folgendes enthalten:
 - Attribute, die auch in der `grouping-list` sind
 - Aggregationsfunktionen (z.B. `MIN(S.age)`)
- Ausdrücke im `group-condition` dürfen ein einziger Wert per Gruppe haben
 - eigentlich enthalten `group-condition` Attribute aus der `grouping-list` oder Aggregatfunktionen

Aufpassen!

- Intuitiv: jedes Tupel gehört zu einer Gruppe und diese Attribute (die wir für die Gruppierung benutzt haben) haben ein einziger Wert für die ganze Gruppe.
- Gruppe = Menge von Tupel, die denselben Wert haben für alle Attribute in der grouping-list

Group by konzeptuelle Evaluation

- das kartesische Produkt der Relationen wird berechnet
- Tupeln, für welche `condition` nicht wahr ist werden rausgeworfen
- für den Rest: Tupel mit gleichen Werten für die angegebenen Attribute (`grouping-list`) werden in Gruppen zusammengefasst
- Gruppen für welche `group-condition` nicht wahr ist werden rausgeworfen.
- Pro Gruppe erzeugt die Anfrage ein Tupel der Ergebnisrelation (Deshalb: Hinter der **SELECT**-Klausel sind nur Attribute mit einem Wert pro Gruppe zugelassen)

Gruppieren mit Ordnen

```
SELECT [DISTINCT] target-list  
FROM relation-list  
WHERE condition  
GROUP BY grouping-list  
HAVING group-condition  
ORDER BY attribute-list [ASC | DESC]
```

Finde das Alter des jüngsten Studenten mit Alter ≥ 20 für jede Gruppe mit wenigstens 2 solche Studenten

```
SELECT S.gruppe, MIN(S.age) AS Jungste
FROM Studenten S
WHERE S.age >= 20
GROUP BY S.gruppe
HAVING COUNT(*) > 1
```

Finde die Anzahl der angemeldeten Studenten und die Mittelwerte der Noten für jeden 6 ECTS Kurs

```
SELECT K.KursId, K.Titel, COUNT(*) as Anzahl,  
AVG(Note) as DurchschnittNote  
FROM Enrolled E, Kurse K  
WHERE E.KursId = K.KursId  
AND K.ECTS = 6  
GROUP BY K.KursId, K.Titel
```

BETWEEN

- eine Möglichkeit den Intervall für ein Attribut zu bestimmen ist BETWEEN

```
SELECT *  
FROM Enrolled  
WHERE NOT Note is NULL AND Note between 7 and 9
```

TOP

- TOP (expression) [PERCENT] [WITH TIES]

```
SELECT TOP(1) WITH TIES E.MatrNr  
FROM Enrolled E  
WHERE E.KursID='BD'  
ORDER BY E.Note DESC
```

- WITH TIES -> gibt alle Tupel aus, die denselben Wert für das Attribut, nachdem geordnet wurde, haben, auch wenn die totale Anzahl das angegebene Limit überschreitet
- Es ist ein guter Praxis ORDER BY zusammen mit TOP zu benutzen, um genau zu wissen, welchen Tupeln ausgegeben werden

Übungen

Datenbank

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

Kurse(KursId, Titel, ECTS)

Enrolled(MatrnNr, KursId, Note)

1. Gebe die Emails, Kurse (Titel) und Noten aller Studenten aus.

- 2 Methoden (mit INNER JOIN und ohne)
- Methode I.

```
SELECT s.email, k.titel, e.note
FROM Studenten s INNER JOIN Enrolled e
ON s.MatrNr = e.MatrNr
INNER JOIN Kurse k
ON e.KursId = k.KursId
```

- Methode II.

```
SELECT s.email, k.titel, e.note
FROM Studenten s, Enrolled e, Kurse k
WHERE s.MatrNr = e.MatrNr
AND e.KursId = k.KursId
```

2. Gebe alle Studenten aus mit Alter \geq allen anderen mit dem Namen 'A...N'(die am ältesten sind)

```
select *  
from Studenten s  
where s.age  $\geq$  ALL (select s2.age  
                      from Studenten s2  
                      where s2.name like 'A%N')
```

3. Anzahl von Studenten deren Namen mit „A“ anfängt für jede Altersgruppe

```
SELECT COUNT(*) as StudentenNr, S.age  
FROM Studenten S  
WHERE S.Name like 'A%'  
GROUP BY S.Age
```

4. Für jeder Student, der in wenigstens 2 Kurse angemeldet ist, gebe die Anzahl der Kurse aus, in denen dieser angemeldet ist

```
SELECT count(*), E.MatrNr  
FROM Enrolled E  
GROUP BY E.MatrNr  
HAVING count(*)>1
```

Datenbank

Kunden (Kundeld, Name, Vorname, Stadt, Land, Tel)

Order (OrderNr, Kundeld, OrderDatum, Preis)

5. Anzahl der Kunden aus jedem Land mit wenigstens 10 Kunden

```
SELECT COUNT(KundeId) , Land  
FROM Kunden  
GROUP BY Land  
HAVING COUNT(KundeId) >=10
```


6. Anzahl der Kunden aus jedem Land mit wenigstens 9 Kunden, außer USA, sortiert absteigend nach Anzahl der Kunden

```
SELECT COUNT(KundeId) , Land
FROM Kunden
where Land <> 'USA'
GROUP BY Land
Having COUNT(KundeId) >=9
ORDER BY Count(KundeId) DESC
```

7. Alle Kunden mit dem durchschnittlichen Preis von Orders zwischen 1000 und 1200

```
SELECT AVG(Preis), K.Name, K.Vorname  
FROM Order O INNER JOIN Kunden K  
ON O.KundeId = K.KundeId  
GROUP BY K.KundeId, K.Name, K.Vorname  
HAVING AVG(O.Preis) BETWEEN 1000 AND 1200
```