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
```

- ein Attribut löschen

```
ALTER TABLE table-name
DROP COLUMN column-name
```

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

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

```
CREATE TABLE Enrolled
(MatrNr CHAR(20),
 KursId CHAR(20),
 Note REAL,
 PRIMARY KEY (MatrNr, KursId),
 FOREIGN KEY (MatrNr) REFERENCES Studenten
 ON DELETE CASCADE
 ON UPDATE SET NULL,
 FOREIGN KEY (KursId) REFERENCES Kurse)
```

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 0 FOR Note
```

Integritätsregeln

```
CREATE TABLE Studenten
(MatrNr CHAR(20),
 Name CHAR(50),
 Vorname CHAR(50),
 Email CHAR(30),
 Age INTEGER,
 Gruppe INTEGER,
 PRIMARY KEY (MatrNr),
 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

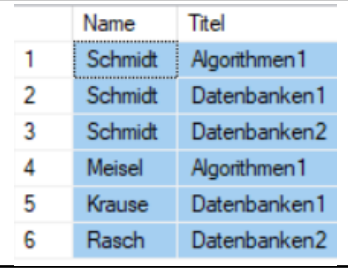
KursId	Titel	ECTS
Alg1	Algorithmen1	6
DB1	Datenbanken1	6
DB2	Datenbanken2	5

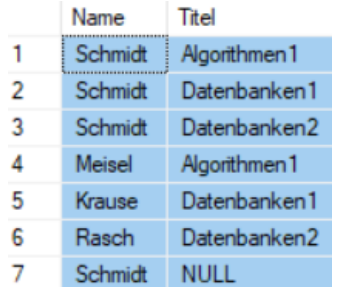
Enrolled

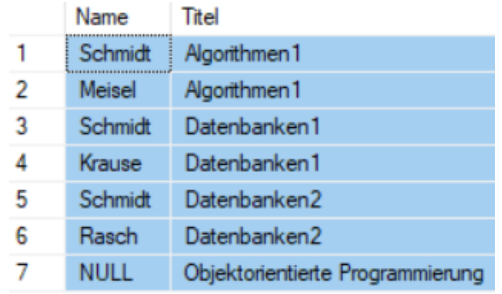
MatrNr	KursId	Note
1234	Alg1	7
1235	Alg1	8
1234	DB1	9
1234	DB2	7
1236	DB1	10
1237	DB2	10

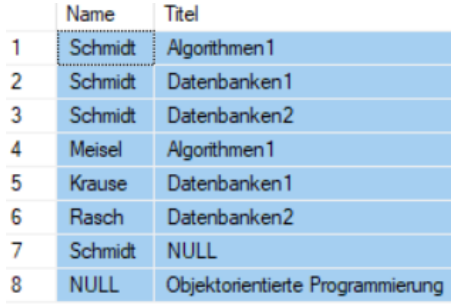
JOIN Abfragen

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`
		

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`
		

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`
		

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`
		

	Name	Titel
1	Schmidt	Algorithmen1
2	Schmidt	Datenbanken1
3	Schmidt	Datenbanken2
4	Meisel	Algorithmen1
5	Krause	Datenbanken1
6	Rasch	Datenbanken2

	Name	Titel
1	Schmidt	Algorithmen1
2	Schmidt	Datenbanken1
3	Schmidt	Datenbanken2
4	Meisel	Algorithmen1
5	Krause	Datenbanken1
6	Rasch	Datenbanken2
7	Schmidt	NULL

	Name	Titel
1	Schmidt	Algorithmen1
2	Meisel	Algorithmen1
3	Schmidt	Datenbanken1
4	Krause	Datenbanken1
5	Schmidt	Datenbanken2
6	Rasch	Datenbanken2
7	NULL	Objektorientierte Programmierung

	Name	Titel
1	Schmidt	Algorithmen1
2	Schmidt	Datenbanken1
3	Schmidt	Datenbanken2
4	Meisel	Algorithmen1
5	Krause	Datenbanken1
6	Rasch	Datenbanken2
7	Schmidt	NULL
8	NULL	Objektorientierte Programmierung

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.

Wir wollen üben!

- SqlClimber – Multiple Join 2
- https://www.sqlclimber.com/assignment/gu5t4x/multiple-joins-2

- SqlClimber – Left Join 2
- https://www.sqlclimber.com/assignment/hy3j6a/left-join-2

Aggregatfunktionen

- werden auf eine Menge von Tupeln angewendet
- Verdichtung einzelner Tupeln zu einem Gesamtwert
- SUM, AVG, MIN, MAX können nur auf Zahlen angewendet werden

X
1
1
4
6

- **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

Aggregation - GROUP BY und HAVING

- Anwendung: wenn wir Tupeln gruppieren wollen um Aggregatfunktionen auf bestimmte Gruppen anzuwenden
- z.B. Finde den 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 einziges 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 einziges Wert für die ganze Gruppe.
- Gruppe = Menge von Tupels, 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, der älter als 20 ist, und der zu einer Studengruppe mit wenigstens 2 solche Studenten gehört

```
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 alle 6 ECTS Kurse

```
SELECT K.KursId, 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
```

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
```

Wir wollen üben!

- SqlClimber – Between 1
- https://www.sqlclimber.com/assignment/p63xjk/condition-between-1

- SqlClimber – Left Join 3
- https://www.sqlclimber.com/assignment/68mpwd/left-join-3

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

Wir wollen üben!

- SqlClimber – Select the first 5 records
- https://www.sqlclimber.com/assignment/3zwzn2/select-the-first-5-records

Übungen

Datenbank

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

Kurse(KursId, Titel, ECTS)

Enrolled(MatrNr, KursId, Note)

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

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