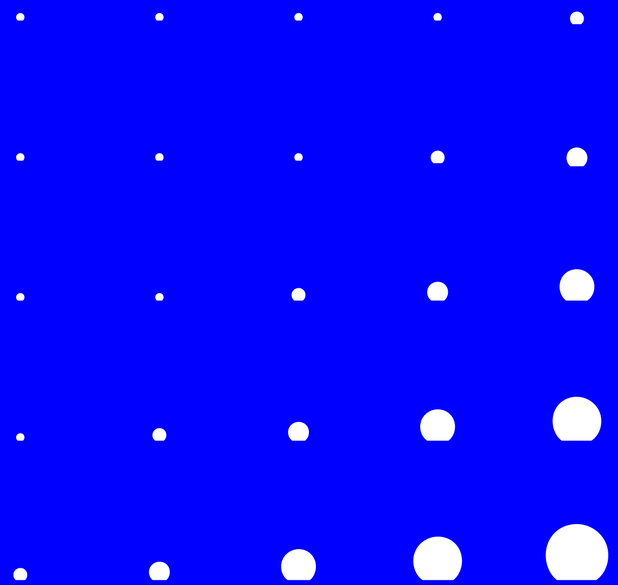
Technische Hochschule
mannheim

SQL-Einführung

siehe auch: https://www.w3schools.com/sql/

Technische Hochschule Mannheim | Prof. Dr. Frank Dopatka



Was ist SQL?

Technische Hochschule mannheim

- SQL ist eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten Abfragen von darauf basierenden Datenbeständen.
- Die Sprache
  - basiert auf der relationalen Algebra,
  - ihre Syntax ist relativ einfach aufgebaut
  - und semantisch an die englische Umgangssprache angelehnt.
- Ein gemeinsames Gremium standardisiert die Sprache unter Mitwirkung nationaler Normungsgremien.
- Fast alle gängigen Datenbanksysteme unterstützen SQL; allerdings in unterschiedlichem Umfang und leicht voneinander abweichenden Dialekten.

MariaDB Datentypen: Ganzzahlen

Technische Hochschule mannheim

- TINYINT, 1 Byte
  - Ganzzahlen von 0 bis 255 oder von -128 bis 127
- SMALLINT, 2 Bytes
  - Ganzzahlen von 0 bis 65.535 oder von -32.768 bis 32.767
- MEDIUMINT, 3 Bytes
  - Ganzzahlen von 0 bis 16.777.215 oder von -8.388.608 bis 8.388.607
- INT, 4 Bytes
  - Ganzzahlen von 0 bis ~4,3 Mill. oder von ~ -2,1 Mill. bis ~ 2,1 Mill.
- BIGINT, 8 Bytes
  - Ganzzahlen von 0 bis 2^64-1 oder von -(2^63) bis (2^63)-1

- Die Zahlen sind standardmäßig „SIGNED“, also mit einem Vorzeichen behaftet.
- Durch den SQL-Befehl „UNSIGNED“ kann der Wertebereich rein positiv verwendet werden, indem das Vorzeichenbit mit genutzt wird.

MariaDB Datentypen: Kommazahlen

Technische Hochschule mannheim

- FLOAT, 4 Bytes
  - Fließkommazahl mit Vorzeichen
- DOUBLE, 8 Bytes
  - Fließkommazahl mit Vorzeichen und doppelter Genauigkeit
- DECIMAL
  - Festkommazahl mit Vorzeichen, die exakte numerische Datenwerte speichert

MariaDB Datentypen: Datumswerte

Technische Hochschule mannheim

- DATE, 3 Bytes
  - Datum im Format `YYYY-MM-DD`
  - Wertebereich von 01.01.1000 bis 31.12.9999
- DATETIME, 8 Bytes
  - Datumsangabe im Format `YYYY-MM-DD hh:mm:ss`
  - Wertebereich entspricht DATE
- TIMESTAMP, 4 Bytes
  - Zeitstempel mit Wertebereich: 1.1.1970 bis 19.01.2038
- TIME, 3 Bytes
  - Zeit zwischen -838:59:59 und 839:59:59, Ausgabe: hh:mm:ss
- YEAR, 1 Byte
  - Jahr zwischen 1901 bis 2155

MariaDB Datentypen: Zeichen, Zeichenketten & Binärdaten

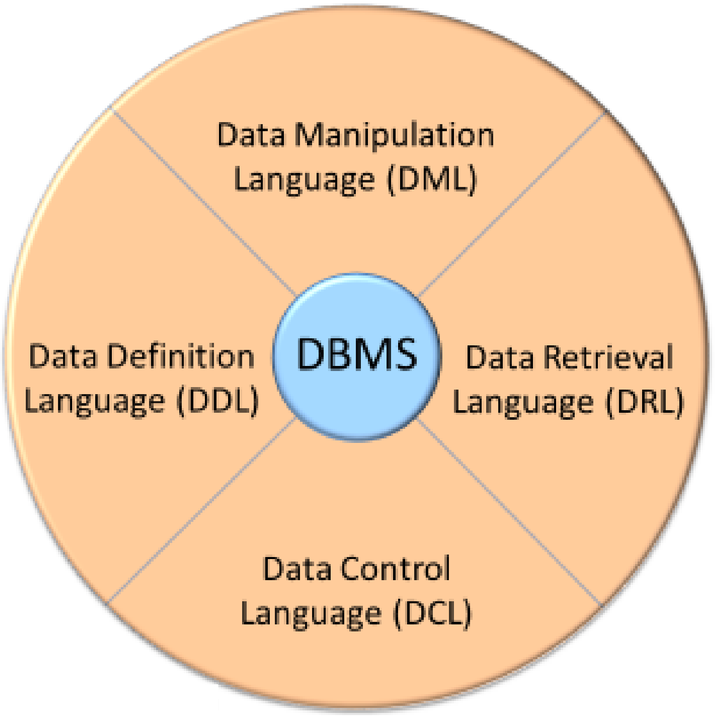
Technische Hochschule mannheim

- CHAR
  - Zeichenkette fester Länge.
- VARCHAR
  - Zeichenkette variabler Länge
- BLOB, 65535 Bytes
  - Binäres Objekt mit variablen Daten.
  - Weitere Typen dazu sind
    - TINYBLOB (255 Bytes),
    - MEDIUMBLOB (16.777.215 Bytes) und
    - LONGBLOB (4.294.967.295 Bytes).

SQL-Befehlskategorien

Technische Hochschule mannheim

- SQL-Befehle lassen sich in 4 Kategorien unterteilen:
- DML:
  Befehle zur Datenmanipulation wie
  Ändern, Einfügen und Löschen
- DDL:
  Befehle zur Definition des Datenbankschemas
- DCL:
  Befehle für die Rechteverwaltung
  und Transaktionskontrolle
- DRL:
  Befehle für lesende Abfragen auf bestehenden Datenbeständen;
  manchmal auch als Data Query Language (DQL) bezeichnet



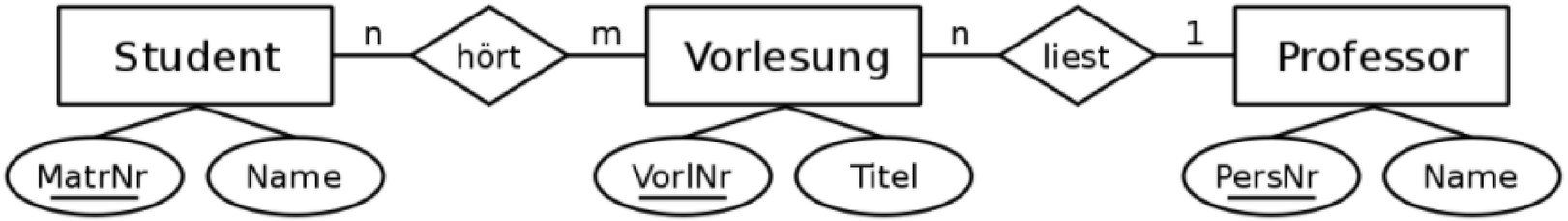
Technische Hochschule
mannheim

DRL/DQL - Einfache Abfragen mit SELECT

DRL - Beispielhafter Datenbestand

Technische Hochschule mannheim

ER-Diagramm:



Relationen:

Student

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

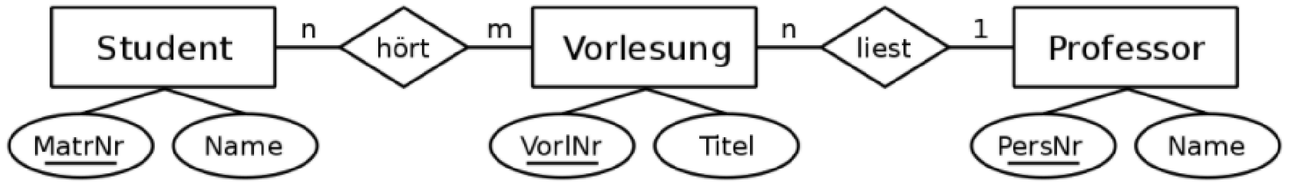
DRL – SELECT...FROM: Einfachste Abfrage

Technische Hochschule mannheim

- **`SELECT * FROM Student;`** listet alle Spalten und alle Zeilen der Tabelle Student auf.
- Das Ergebnis lautet hier:

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

ER-Diagramm:



Relationen:

Student

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

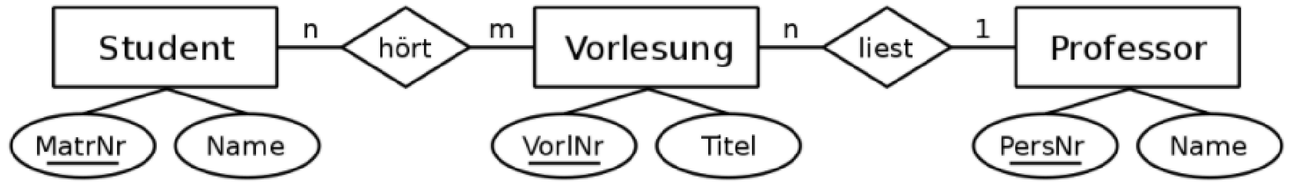
DRL – SELECT...FROM: Abfrage mit Spaltenauswahl

Technische Hochschule mannheim

- **`SELECT VorlNr, Titel FROM Vorlesung;`**
  listet die Spalten VorlNr und Titel aller Zeilen der Tabelle Vorlesung auf.
- Das Ergebnis lautet hier:

VorlNr	Titel
5001	ET
5022	IT
5045	DB

ER-Diagramm:



Relationen:

Student

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

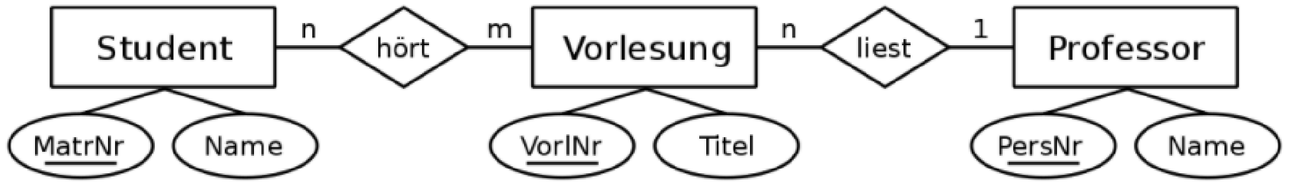
DRL – SELECT...FROM: Abfrage mit Spaltenauswahl

Technische Hochschule mannheim

- **`SELECT DISTINCT MatrNr FROM hört;`**
  listet nur unterschiedliche Einträge der Spalte MatrNr aus der Tabelle „hört“ auf.
- Dies zeigt die Matrikelnummern aller Studenten, die mindestens eine Vorlesung hören, wobei mehrfach auftretende Matrikelnummern nur einmal ausgegeben werden.
- Das Ergebnis lautet hier:

MatrNr
25403
26120

ER-Diagramm:



Relationen:

Student

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

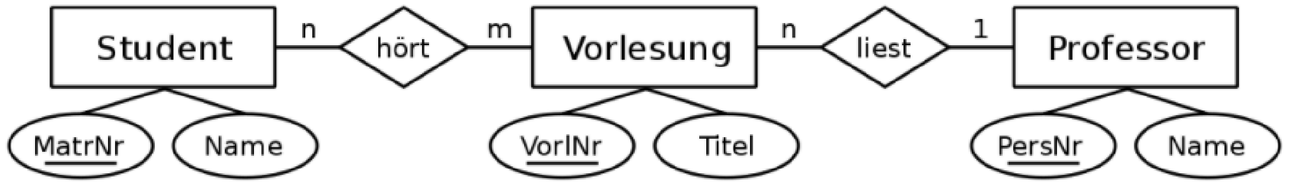
DRL – SELECT...AS...FROM: Abfrage mit Umbenennung

Technische Hochschule mannheim

- **`SELECT MatrNr AS Matrikelnummer, Name FROM Student;`**
  listet die Spalten MatrNr und Name aller Zeilen der Tabelle Student auf.
- MatrNr wird beim Anzeigeergebnis als Matrikelnummer aufgeführt.
- Das Ergebnis lautet hier:

Matrikelnummer	Name
26120	Fichte
25403	Jonas
27103	Fauler

ER-Diagramm:



Relationen:

Student

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

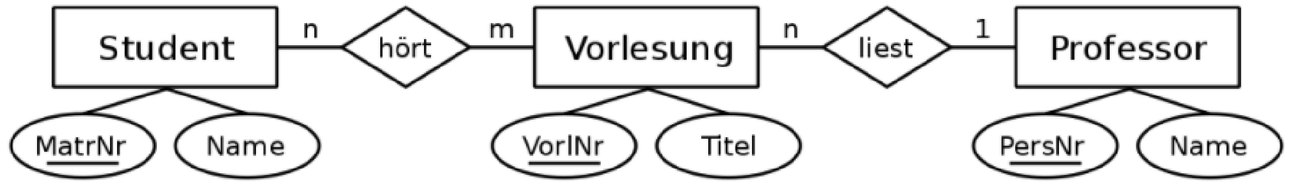
DRL – SELECT...FROM...WHERE: Abfrage mit Filter

Technische Hochschule mannheim

- **`SELECT VorlNr, Titel FROM Vorlesung WHERE Titel=`ET`;`**
  listet VorlNr und Titel aller derjenigen Zeilen der Tabelle Vorlesung auf, deren Titel `ET` ist.
- Diese häufig verwendete Anweisung wird nach den Anfangsbuchstaben auch als SFW-Block bezeichnet.
- Das Ergebnis lautet hier:

VorlNr	Titel
5001	ET

ER-Diagramm:



Relationen:

Student

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

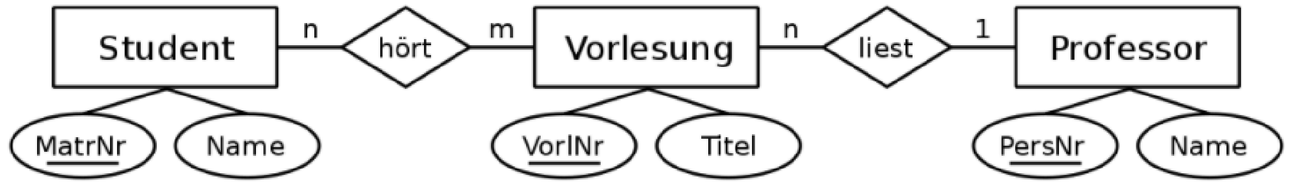
DRL – SELECT...FROM...WHERE...LIKE: Abfrage mit Filter nach Inhalt

Technische Hochschule mannheim

- **`SELECT Name`**
  `FROM Student`
  `WHERE Name LIKE `F%`;`
  listet die Namen aller Studenten auf, deren Name mit F beginnt.
- LIKE kann mit verschiedenen Platzhaltern belegt werden:
  - _ steht für ein fehlendes Zeichen
  - % steht für eine beliebige Zeichenfolge.
- So können mit der Abfrage auch Felder nach Inhalt durchsucht werden.
- Das Ergebnis lautet hier:

Name
Fichte
Fauler

ER-Diagramm:



Relationen:

Student

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

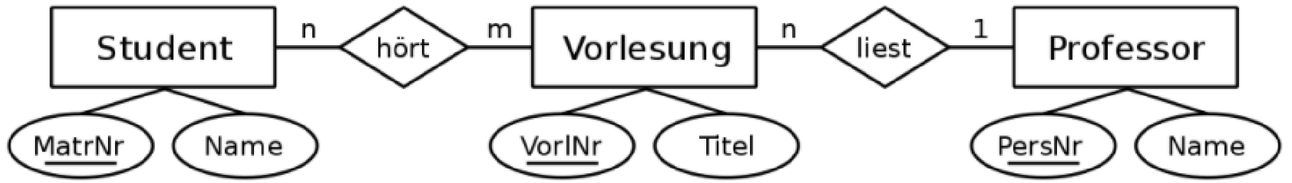
F...

DRL – ...ORDER BY: Abfrage mit Sortierung

Technische Hochschule mannheim

- **`SELECT *`**
  `FROM Student`
  `ORDER BY Name ASC;`
  listet alle Daten aller Studenten sortiert nach Name auf.
- Normalerweise wird aufsteigend sortiert, so dass die Angabe von ASC optional ist.
- Möchte man absteigend sortieren, so muss man statt dessen DESC verwenden.

ER-Diagramm:



Relationen:

Student

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

Technische Hochschule
mannheim

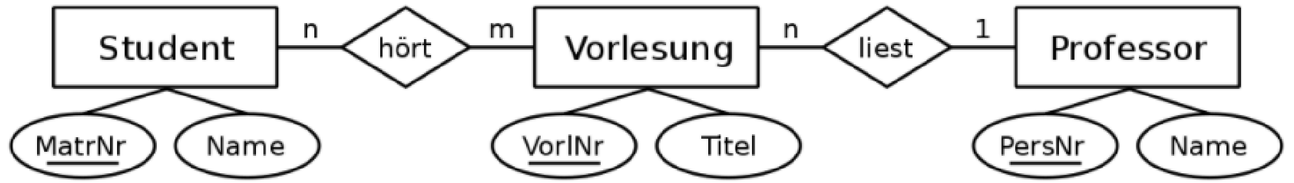
DRL
Komplexere Abfragen über mehrere Tabellen mit verschachteltem SELECT

DRL – SELECT...FROM...WHERE: Abfrage mit verknüpften Tabellen

Technische Hochschule mannheim

- **SELECT**
  v.VorlNr, v.Titel,
  p.PersNr, p.Name
  FROM
  Professor p, Vorlesung v
  WHERE p.PersNr = v.PersNr;

ER-Diagramm:



Relationen:

Student

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

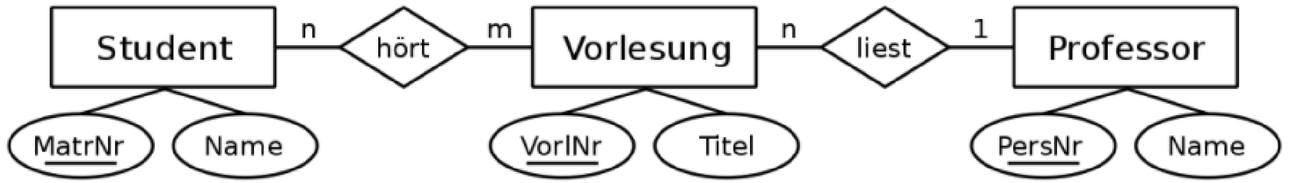
- Die Aufzählung hinter FROM legt die beiden Datenquellen fest.
- Dies sind alle Datensätze aus den Tabellen Professor und Vorlesung, die den gleichen Wert im Feld PersNr haben.
- Professoren ohne Vorlesung und Vorlesungen ohne Professor werden damit nicht angezeigt.

DRL – SELECT...FROM...WHERE: Abfrage mit verknüpften Tabellen

Technische Hochschule mannheim

- **SELECT**
  v.VorlNr, v.Titel,
  p.PersNr, p.Name
  FROM
  Professor p, Vorlesung v
  WHERE p.PersNr = v.PersNr;

ER-Diagramm:



Relationen:

Student

MatrNr	Name
26120	Fichte
25403	Jonas
27103	Fauler

hört

MatrNr	VorlNr
25403	5001
26120	5001
26120	5045

Vorlesung

VorlNr	Titel	PersNr
5001	ET	15
5022	IT	12
5045	DB	12

Professor

PersNr	Name
12	Wirth
15	Tesla
20	Urlauber

- Eine solche Struktur über mehrere Tabellen nennt man JOIN:
  - Dabei werden mehrere Tabellen über Schlüsselfelder verknüpft, wodurch deren Daten zusammengeführt und angezeigt werden.
  - In diesem Fall spricht man von einem inneren natürlichen Verbund NATURAL JOIN als Spezialfall des INNER JOIN.
  - JOINs werden im nächsten Kapitel genauer betrachtet...

DRL am Beispiel SVS 1.x mit MariaDB und Java

Technische Hochschule mannheim

- Das StudierendenVerwaltungsSystem SVS bestand in der Version 1.x aus Veranstaltungen, Teams und Studierenden.
  - In einer Veranstaltung gibt es beliebig viele Studierende/Benutzer.
  - In einer Veranstaltung gibt es beliebig viele Teams.
  - Ein Team besteht aus beliebig vielen Studierenden/Benutzern.
  - Studierende/Benutzer können auch Teil einer Veranstaltung sein, aber nicht eines Teams.

DRL am Beispiel SVS 1.x mit MariaDB und Java: Tabelle benutzer

Technische Hochschule mannheim

Name	Typ	Kollation	Attribute	Null	Standard
id_benutzer 🔑	varchar(20)	latin1_german2_ci		Nein	*kein(e)*
geschlecht	varchar(1)	latin1_german2_ci		Ja	*NULL*
vorname	varchar(50)	latin1_german2_ci		Ja	*NULL*
nachname	varchar(50)	latin1_german2_ci		Ja	*NULL*
nummer_matrikel	varchar(10)	latin1_german2_ci		Ja	*NULL*
mail	varchar(100)	latin1_german2_ci		Ja	*NULL*
kennwort	varchar(100)	latin1_german2_ci		Ja	*NULL*
aktiv	tinyint(1)			Nein	0
aktivierungscode	varchar(30)	latin1_german2_ci		Ja	*NULL*

DRL am Beispiel SVS 1.x mit MariaDB und Java: Tabelle veranstaltung & student_in_veranstaltung

Technische Hochschule mannheim

Name	Typ	Kollation	Attribute	Null	Standard	Kommentare	Extra
id_veranstaltung	bigint(20)			Nein	*kein(e)*		AUTO_INCREMENT
name	varchar(50)	latin1_german2_ci		Ja			
sichtbar	tinyint(1)			Nein	0		
anz_blatt	int(11)			Nein	5		
anz_test	int(11)			Nein	5		
punkte_max	int(11)			Nein	*kein(e)*		
punkte_bestanden	int(11)			Nein	*kein(e)*		
kennwort	varchar(10)	latin1_german2_ci		Ja	*NULL*		

Name	Typ	Kollation	Attribute	Null	Standard
id_benutzer	varchar(20)	latin1_german2_ci		Nein	*kein(e)*
id_veranstaltung	bigint(20)			Nein	*kein(e)*

DRL am Beispiel SVS 1.x mit MariaDB und Java: Tabelle team & student_in_team

Technische Hochschule mannheim

Name	Typ	Kollation	Attribute	Null	Standard	Kommentare	Extra
id_team 🔑	bigint(11)			Nein	*kein(e)*		AUTO_INCREMENT
id_veranstaltung	bigint(20)			Nein	*kein(e)*		
block	varchar(1)	latin1_german2_ci		Ja	*NULL*		
nummer	tinyint(4)			Nein	*kein(e)*		
kommentar_team	varchar(5000)	latin1_german2_ci		Nein			
kommentar_admin	varchar(5000)	latin1_german2_ci		Nein			

Name	Typ	Kollation	Attribute	Null	Standard
id_benutzer 🔑	varchar(20)	latin1_german2_ci		Nein	*kein(e)*
id_team 🔑	bigint(11)			Nein	*kein(e)*

Aufteilung von SQL-Statements in Java

Technische Hochschule mannheim

- Durch die Aufteilung von verschachtelten SQL-Statements in der Anwendungslogik…
  - kann jedes SQL-Statement einzeln getestet werden.
  - bleibt der SQL-Code übersichtlicher und besser wartbar.
  - erhält man vielleicht nicht das effizienteste Ergebnis aus Sicht der Performance und der Server-Last.

- Die Wartbarkeit verringert aber die Anzahl der notwendigen Personal-Stunden.
  - In der Regel sind Personal-Stunden heutzutage teurer als Hardware-Kapazität.
  - Erst bei hoch skalierten Systemen kann dies anders herum sein.

DRL – SELECT...WHERE...IN: Verschachtelte SELECTs am Beispiel SVS 1.x

Technische Hochschule mannheim

```
getTeamDesStudierenden(int id_veranstaltung,String id_benutzer){
  String sql1,sql2;
  // Team-ID dieses Studenten holen
  sql1= "SELECT id_team
        FROM student_in_team
        WHERE id_benutzer="+id_benutzer;
  // und von diesem Team alle Daten
  sql2= "SELECT DISTINCT *
        FROM team
        WHERE id_veranstaltung="+id_veranstaltung+
        " AND id_team IN ("+sql1+");";
  ...
```

DRL – SELECT...WHERE...IN:
Verschachtelte SELECTs am Beispiel SVS 1.x

Technische Hochschule mannheim

```
getStudierendeDesTeams(int id_veranstaltung,String team){
  String sql1,sql2;
  int id_team=getId(id_veranstaltung,team);
  // hole die Benutzer-IDs der Studenten aus diesem Team
  sql1= "SELECT DISTINCT id_benutzer
        FROM student_in_team
        WHERE id_team="+id_team;
  // und von diesen IDs alle Benutzerdaten
  sql2= "SELECT DISTINCT *
        FROM benutzer
        WHERE id_benutzer IN ("+sql1+ ")
        ORDER BY nachname,vorname;";
  ...
```

DRL – SELECT...WHERE...IN:
Verschachtelte SELECTs am Beispiel SVS 1.x

Technische Hochschule mannheim

```
getVeranstaltungenDesStudierenden(String id_benutzer){
  String sql1,sql2;
  // alle Veranstaltungen des Studenten
  sql1= "SELECT DISTINCT id_veranstaltung
        FROM student_in_veranstaltung
        WHERE id_benutzer="+id_benutzer;
  // und davon alle Daten
  sql2= "SELECT DISTINCT *
        FROM veranstaltung
        WHERE id_veranstaltung IN ("+sql1+")
        ORDER BY name;";
  ...
```

DRL – SELECT...WHERE...IN: Verschachtelte SELECTs am Beispiel SVS 1.x

Technische Hochschule mannheim

```
getAndereVeranstaltungenOhneDiesenStudierenden(String id_benutzer){
  String sql1,sql2;
  // alle Veranstaltungen des Studierenden
  sql1="SELECT DISTINCT id_veranstaltung
        FROM student_in_veranstaltung
        WHERE id_benutzer="+id_benutzer;
  // und von den anderen Veranstaltungen alle Daten
  sql2="SELECT DISTINCT * FROM veranstaltung
        WHERE id_veranstaltung NOT IN ("+sql1+")
        ORDER BY name;";
  ...
```

DRL – SELECT...WHERE...IN: Verschachtelte SELECTs am Beispiel SVS 1.x

Technische Hochschule mannheim

```
getStudierendeAusAnderenVeranstaltungen(int id_veranstaltung){
  String sql1,sql2,sql3;
  // alle Studenten
  sql1="SELECT DISTINCT id_benutzer FROM benutzer";
  // alle Studenten aus dieser Veranstaltung
  sql2="SELECT DISTINCT id_benutzer
        FROM student_in_veranstaltung
        WHERE id_veranstaltung="+id_veranstaltung;
  // alle Studentendaten aus anderen Veranstaltungen holen,
  // die nicht auch in dieser Veranstaltung sind
  sql3="SELECT DISTINCT *
        FROM benutzer
        WHERE id_benutzer IN ("+sql1+")
        AND id_benutzer NOT IN ("+sql2+")
        ORDER BY nachname,vorname;";
  ...
```

DRL – SELECT...WHERE...IN: Verschachtelte SELECTs am Beispiel SVS 1.x

Technische Hochschule mannheim

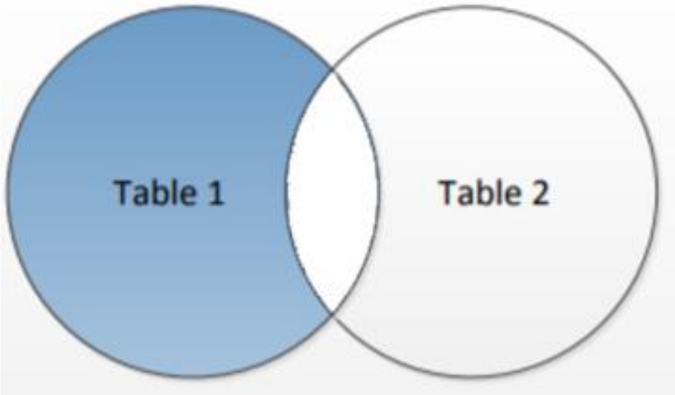
```
getStudierendeDerVeranstaltungOhneTeam(int id_veranstaltung){
  String sql1,sql2,sql3,sql4;
  // alle Teams dieser Veranstaltung
  sql1="SELECT DISTINCT id_team
       FROM team
       WHERE id_veranstaltung="+id_veranstaltung;
  // alle Studenten, die in dieser Veranstaltung in einem Team sind
  sql2="SELECT DISTINCT id_benutzer
       FROM student_in_team
       WHERE id_team IN ("+sql1+")";
  // alle Studenten dieser Veranstaltung, die noch kein Team haben
  sql3="SELECT DISTINCT id_benutzer
       FROM student_in_veranstaltung
       WHERE id_veranstaltung="+id_veranstaltung+
       " AND id_benutzer NOT IN ("+sql2+")";
  // von diesen Studenten alle holen
  sql4="SELECT DISTINCT * FROM benutzer
       WHERE id_benutzer IN ("+sql3+")
       ORDER BY nachname,vorname;";
  ...
```

DRL – SELECT...WHERE...IN: Verschachtelte SELECTs am Beispiel SVS 1.x

Technische Hochschule mannheim

```
getStudierendeAusAnderenTeamsDerVeranstaltung(int id_veranstaltung,String
team){
  String sql1,sql2,sql3;
  // alle Teams dieser Veranstaltung ausser diesem Team
  sql1= "SELECT DISTINCT id_team
        FROM team
        WHERE id_veranstaltung="+ id_veranstaltung+
        " AND id_team<>`"+team+"`";
  // alle Studenten, die in dieser Veranstaltung,
  // in einem dieser Team sind
  sql2= "SELECT DISTINCT id_benutzer
        FROM student_in_team
        WHERE id_team IN ("+sql1+")";
  // und von diesen Studenten alle Daten holen
  sql3= "SELECT DISTINCT *
        FROM benutzer
        WHERE id_benutzer IN ("+sql2+")
        ORDER BY nachname,vorname;";
  ...
```

DRL: Die Differenz



- Bei der Differenz werden aus der ersten Relation alle Tupel entfernt, die auch in der zweiten Relation vorhanden sind.

A	B	C
1	2	3
4	5	6

Table 1

A	B	C
7	8	9
4	5	6

Table 2

A	B	C
1	2	3

Differenz

DRL: Die Differenz im SQL-Standard

Technische Hochschule mannheim

A	B	C
1	2	3
4	5	6

R

A	B	C
7	8	9
4	5	6

S

A	B	C
1	2	3

Differenz

- Im SQL-Standard:
  - **`SELECT * FROM R EXCEPT (SELECT * FROM S);`**

  oder
  - **`SELECT * FROM R MINUS (SELECT * FROM S);`**
- MariaDB unterstützt die Operatoren EXCEPT und MINUS aber leider nicht...
- Also muss die Differenz anders hergeleitet werden...

DRL: Die Differenz
Umsetzung in MariaDB-SQL über 2 SELECTs

Technische Hochschule mannheim

A	B	C
1	2	3
4	5	6

R

A	B	C
7	8	9
4	5	6

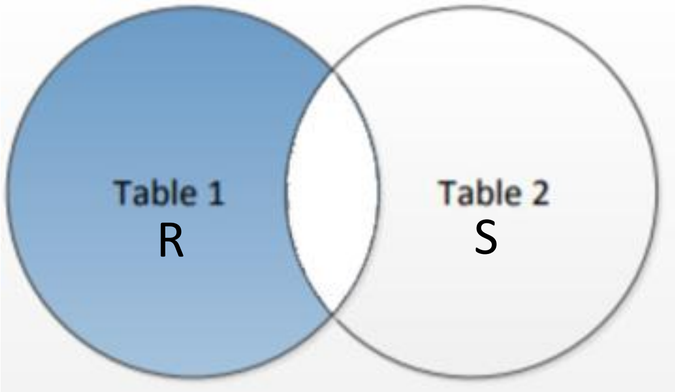
S

A	B	C
1	2	3

Differenz

```
SELECT * FROM R WHERE NOT EXISTS (SELECT * FROM S WHERE R.A = S.A);
```

- …wobei A jeweils der der jeweiligen Tabelle Primärschlüssel ist.



Technische Hochschule
mannheim

Aggregatfunktionen

Was sind Aggregatfunktionen?

Technische Hochschule mannheim

- Ihren Namen haben die Aggregatfunktionen vom englischen Verb to aggregate, was auf deutsch anhaufen, vereinigen oder zusammenballen heisst.
- Aggregatfunktionen bilden genau einen Ergebniswert aus einer Vielzahl von Eingabewerten.
- Die fünf wichtigsten SQL-Aggregatfunktionen sind
  - COUNT als Anzahl,
  - SUM als Summe,
  - AVG als arithmetischer Mittelwert,
  - MAX als Maximum sowie
  - MIN als Minimum einer Spalte.

Aggregatfunktionen am Beispiel

Technische Hochschule mannheim

Tabelle Wetter

stadt	temp_min	temp_max
FR	-1	19
S	-5	12
MA	-2	13
KA	-2	14

- `SELECT MAX(temp_min) as MAX FROM wetter;`
- ```
  SELECT *
  FROM wetter
  WHERE temp_min = (SELECT MAX(temp_min) FROM wetter);
  ```
  ```

Aggregieren und Gruppieren

Technische Hochschule mannheim

- Durch das GROUP BY Statement ist es möglich, eine Ergebnismenge zu gruppieren.
- Dieser SQL-Befehl wird häufig in Kombination mit den Aggregatfunktionen verwendet.

Auto	KM-Stand	Baujahr
Auto 1	30 000km	2002
Auto 2	10 000km	2010
Auto 3	20 000km	2010
Auto 4	30 000km	2001

```
SELECT Baujahr, COUNT(Baujahr) AS AnzahlAutos
FROM Auto
WHERE Baujahr='2010'
GROUP BY Baujahr
```

Baujahr	AnzahlAutos
2010	2