Janek Schoffit *13. Mai 2018*

SAE Stoffsammlung

Gliederung

1 Normalformen

1.1 1. Normalform

Jedes Attribut der Relation muss einen atomaren Wertebereich haben, und die Relation muss frei von Wiederholungsgruppen sein.
`Atomar` bedeutet, dass ein Attribut nur eine Information enthalten darf, also dürfen bspw. Straße und Hausnummer nicht in einem Attribut gespeichert werden, sondern müssen aufgeteilt werdenn.
Mit `Wiederholdungsgruppen` sind zum einen wiederholende Werte in einer Spalte gemeint (also bspw. in der Spalte Ort mehrmals der Wert Kirchheim) und zum anderen sich wiederholende Spalten (bspw. Spalten Standort1, Standort2, Standort3).

Durch diese Normalform werden Abfragen der Datenbank überhaupt erst möglich gemacht. Ohne atomare Daten könnte nicht jeder Aspekt der Daten einzeln abgefragt werden.

1.2 2. Normalform

1. Normalform muss erfüllt sein. Ein Nichtprimärattribut darf nicht funktional von einer Teilmenge eines Schlüsselkandidaten abhängen.
Das bedeutet: Attribute die nur von einem Teil eines Schlüssels und nicht von allen Schlüsseln in einer Tabelle (bei zusammengesetzten Schlüsseln) abhängig sind müssen ausgelagert werden, damit sie voll abhängig von ihrem Schlüssel werden.

Durch diese Normalform modelliert jede Relation nur einen Sachverhalt.

1.3 3. Normalform

2. Normalform muss erfüllt sein. Kein Nichtschlüsselattribut darf transitiv von einem Schlüsselkandidaten abhängig sein.
Das bedeutet: Wenn aus einem Attribut ein anderes Attribut ersichtlich wird, welches aber nicht durch den Schlüssel ersichtlich wird, ist es transitiv von dem ersten Attribut anhängig.

Bsp.:

CD_ID	Albumtitel	Interpret	Gründungsjahr
1	Not That Kind	Anastacia	1999
2	Wish you were here	Pink Floyd	1965
3	Freak of Nature	Anastacia	1999

Das Gründungsjahr ist durch den Interpreten ersichtlich und nicht durch die CD_ID. Das heißt, Interpret und Gründungsjahr sollten in eine Tabelle ausgelagert werden.

2 SQL Befehle

2.1 Tabelle erstellen

```
CREATE TABLE tabellenname
(id INT NOT NULL AUTO_INCREMENT,
wert VARCHAR(255) NOT NULL, PRIMARY (id));
```

2.2 Werte eintragen

```
INSERT INTO tabellenname (id, wert)
VALUES (1, 'Test');
```

2.3 Bestehende Werte ändern

```
UPDATE tabellenname
SET wert = 'Update TEST'
WHERE id = 1;
```

2.4 Werte aus Tabelle löschen

```
DELETE FROM tabellenname
WHERE id = 1;
```

2.5 Komplette Tabelle löschen

```
DROP tabellenname;
```

2.6 Alle Werte aus Tabelle löschen

```
TRUNCATE tabellenname;
```

2.7 Werte anzeigen

2.7.1 Alle anzeigen

```
SELECT * FROM tabellenname;
```

2.7.2 Nur bestimmte Spalten anzeigen

```
SELECT werte FROM tabellenname;
```

2.7.3 Spalten Beschriftungen

```
SELECT werte AS "Text"
FROM tabellenname;
```

2.7.4 Zeilen zählen

Bei 3 Einträgen in Werte:

```
SELECT COUNT(werte) AS "Anzahl Texte"
FROM tabellenname;
```

Ergebnis: Anzahl Texte 3

2.7.5 Werte aufsummieren

Angenommen die Tabelle enthält in der Spalte anzahl die Werte: 1, 3, 3, 7

```
SELECT SUM(anzahl) AS "Summe Anzahl"
FROM tabellenname;
```

Ergebnis wäre dann: Summe Anzahl 14

2.7.6 Weitere Rechenfunktionen

Außer COUNT() und SUM() gibt es noch:

- AVG: Average, Durchschnitt über Werte bilden
- MIN: Den kleinsten Wert ausgeben
- MAX: Den größten Wert ausgeben
- ROUND(spalte, dezimalstellen): Gerundete Werte ausgeben

2.7.7 Werte miteinander verrechnen

Angenommen die Tabelle enthält in der Spalte anzahl die Werte: 4, 2 und in der Spalte preis die Werte 5, 4.

```
SELECT (anzahl * preis) AS "Preis gesamt"
FROM tabellenname;
```

Die Ergebnisse wären dann: 20, 8

2.8 Werte sortieren

Eine unsortierte Liste bspw.: 7, 3, 1, 3

```
SELECT werteunsortiert
FROM tabellenname
ORDER BY werteunsortiert ASC;
```

Aufsteigend sortiert wird mit `ASC`.
Absteigend sortiert wird mit `DESC`.
Ergibt nach dem sortieren mit ASC: 1, 3, 3, 7
Nach dem sortieren mit DESC: 7, 3, 3, 1

2.8.1 Werte zufällig anordnen

Manchmal kann es sinnvoll sein, Werte in eine zuällige Reihenfolge zu bringen.

```
SELECT werteunsortiert
FROM tabellenname
ORDER BY RAND();
```

2.9 Werte gruppieren

Angenommen es gibt die Kategorien: 1, 3, 3, 7. Es sollen die einzelnen Kategorien und ihre Häufigkeit angezeigt werden (optional).

```
SELECT werte, COUNT(werte)
FROM tabellenname
GROUP BY werte;
```

Die Ergebnisse: 1 ⇒ 1x, 3 ⇒ 2x, 7 ⇒ 1x

2.9.1 HAVING bei GROUP BY

Mit WHERE kann man vor einer Gruppierung filtern. Soll aber nach einem GROUP BY gefiltert werden (damit die Gruppierung nicht verfälscht wird) geht das nicht mit WHERE sondern nur mit HAVING. Die Syntax für das filtern bleibt dabei die gleiche wie bei WHERE.

```
SELECT werte, COUNT(werte)
FROM tabellenname
GROUP BY werte
HAVING werte > 1;
```

2.10 Die ersten X Zeilen ausgeben

Wenn bspw. bei einer Highscore Liste nur die ersten 5 Kandidaten relevant sind, können diese mit LIMIT beschränkt werden.

```
SELECT werte
FROM tabellenname
ORDER BY werteunsortiert DESC
LIMIT 5;
```

Hiermit werden die 5 größten Werte ausgegeben.

2.11 Tabellen miteinander verknüpfen - JOINS

2.11.1 INNER JOIN

Tabelle Person (id, name, fk_ort)
Tabelle Ort (id, name, plz)

```
SELECT Person.name, Ort.plz
FROM Person INNER JOIN Ort
ON Person.fk_ort = Ort.id;
```

Jetzt werden alle Personen ausgegeben, die einen Ort zugewiesen haben. Personen die keinen Ort zugewiesen werden, fallen weg.

2.11.2 LEFT JOIN

Funktioniert genau gleich wie INNER JOIN, jedoch wird die linke Schnittmenge verwendet. Das bedeutet bei FROM Person LEFT JOIN Ort ist Person die linke Tabelle. Das heißt es werden alle Personen angezeigt, auch wenn diese keinen Ort eingetragen haben.

2.11.3 RIGHT JOIN

Genau das selbe Prinzip wie LEFT JOIN, bloß diesesmal wird die rechte Schnittmenge verwendet. Es würden in diesem Beispiel also alle Orte angezeigt werden, auch wenn dazu keine Personen zugeordnet sind (in diesem Beispiel eher weniger sinnvoll). Jedes RIGHT JOIN kann in ein LEFT JOIN umgeschrieben und umgekehrt. Es ist nur eine Darstellungssache, wenn man bspw. aus einem ERM Modell die Darstellung beibehalten möchte.

3 Programmierung

3.1 Dateien lesen

Dateien können mit einem FileReader geöffnet werden und mit einem Scanner ausgelesen werden.
Beispiel:

```
FileReader datei = new FileReader("dateipfad/datei.txt");
Scanner scn = new Scanner(datei);

while (scn.hasNextLine()) {
        ....
}
scn.close();
```

3.2 Dateien schreiben

Mit dem FileWriter können Dateien geöffnet werden. Der BufferedWriter schreibt dann in die Dateien.
Beispiel:

```
FileWriter filew = new FileWriter("dateipfad/datei.txt");
BufferedWriter bufferedw = new BufferedWriter(filew);

bufferedw.write("Das ist ein Test");
bufferedw.newLine();
bufferedw.write("Das ist ein zweiter Test");
bufferedw.close();
```