Datenbanken

Normalisierung von Datenbanken

Unter Normalisierung einer relationalen Datenbank versteht man die Aufteilung von Attributen in mehrere Relationen (Tabellen) mit Hilfe der Normalisierungsregeln. Insgesamt gibt es fünf Normalformen, aber im Alltag reicht eine Normalisierung bis zur dritten Normalform vollkommen aus. Die vierte und fünfte Normalform sind für Spezialfälle wie Atomkraftwerke usw. notwendig.

Die einzelnen Normalformen bauen aufeinander auf, d.h. die dritte Normalform ist nur erfüllt wenn auch die zweite Normalform erfüllt ist und die zweite Normalform ist nur erfüllt, wenn auch die erste Normalform erfüllt ist.

Ziel der Normalisierung:

- Beseitigung von Redundanzen
- Vermeidung von Anomalien
- Erstellen eines klar strukturierten Datenbankmodells

Erste Normalform (1NF)

Die **erste Normalform** ist dann erfüllt, wenn alle Informatione einer Tabelle **atomar** vorliegen. Atomar bedeutet, dass jede Information eine eigene Spalte bekommt.

Beispiel:

- nicht normalisierte Form

R-Nr	Datum	Name	Straße	Ort	Artikel	Anzahl	Preis
187	01.01.2022	Max Mustermann	Musterweg. 1	12345 Musterstadt	Bleistift	5	1.00€

- erste Normalform

R-Nr	Datum	Name	Vorname	Straße	Hnr	PLZ	Ort	Artikel	Preis	Währung
187	01.01.2022	Mustermann	Max	Musterstr.	1	12345	Musterstadt	5	1.00	Euro

Zweite Normalform (2NF)

Die **zweite Normalform** ist dann erfüllt, wenn jedes nicht-Schlüsselattribut **voll funktional** vom Schlüssel abhängig ist.

Beispiel:

- zweite Normalform

Rechnung		
R-Nr	Datum	Knr
187	01.01.2022	007

Kunde						
Knr	Name	Vorname	Straße	Hnr	PLZ	Ort
007	Mustermann	Max	Musterstr.	1	12345	Musterstadt

Rechnungsposition			
R-P-NR	R-Nr	Art-Nr	Anzahl
1	187	69	5

Artikel		
Art-Nr	Artikel	Preis
69	Bleistift	1.00

Dritte Normalform (3NF)

Die **dritte Normalform** ist dann erfüllt, wenn kein Nichtschlüsselattribut **transitiv** von einem Kandidatenschlüssel abhängt. Einfacher gesagt, es gibt immer nur einen möglichen Primärschlüssel und kein Attribut die auch Primärschlüssel sein können werden zusätzliche Tabellen ausgelagert und als Fremdschlüssel eingebunden.

Beispiel:

- dritte Normalform

Kunde					
Knr	Name	Vorname	Straße	Hnr	PLZ
007	Mustermann	Max	Musterstr.	1	12345

Postleitzahl	
PLZ	Ort

12345	Musterstadt

Manipulation von Datenbanken

Eine Datenbank ist eine organisierte Sammlung von strukturierten Informationen oder Daten, die elektronisch in einem Computer gespeichert werden. Eine Datenbank wird in der Regel von einem Datenbankmanagementsystem (DBMS) gesteuert. Die Daten und das DBMS werden zusammen mit den damit verbundenen Anwendungen als Datenbanksystem bezeichnet, oft auch nur als Datenbank.

Die Daten in den heute gebräuchlichsten Arten von Datenbanken werden in der Regel in Zeilen und Spalten in einer Reihe von Tabellen dargestellt, um eine effiziente Verarbeitung und Datenabfrage zu ermöglichen. Die Daten können dann leicht abgerufen, verwaltet, geändert, aktualisiert, kontrolliert und organisiert werden. Die meisten Datenbanken verwenden eine strukturierte Abfragesprache (SQL) zum Schreiben und Abfragen von Daten.

Die `INSERT INTO` Anweisung wird verwendet, um einen neuen Datensatz (Zeile) in eine Tabelle einzufügen.
 	
Es gibt zwei Varianten:
• In Spalten der Reihe nach einfügen
• Einfügen in Spalten nach Namen
	`-- In Spalten der Reihe nach einfügen:`
	`INSERT INTO table_name`
	`VALUES (value1, value2);`
	
	`-- Einfügen in Spalten nach Namen:`
	`INSERT INTO table_name (column1, column2)`
	`VALUES (value1, value2);`

Die `DELETE` Anweisung wird verwendet, um Datensätze (Zeilen) in einer Tabelle zu löschen. Die `WHERE` Anweisung gibt an, welcher Datensatz oder welche Datensätze gelöscht werden sollen. Wenn `WHERE` weggelassen wird, werden alle Datensätze gelöscht.	`DELETE FROM table_name` `WHERE some_column = some_value;`

Die `UPDATE` Anweisung wird verwendet, um Datensätze (Zeilen) in einer Tabelle zu bearbeiten. Sie enthält eine `SET` Anweisung, die die zu bearbeitende Spalte angibt, und eine `WHERE` Anweisung, die den oder die Datensätze spezifiziert.	`UPDATE table_name`
	`SET column1 = value1, column2 = value2`
	`WHERE some_column = some_value;`

Datenbank Abfragen

Mit dem `AND` Operator können mehrere Bedingungen kombiniert werden. Die Datensätze müssen beiden Bedingungen entsprechen, die mit `AND` verknüpft sind, um in die Ergebnisse aufgenommen zu werden. Die angegebene Abfrage zeigt alle Autos an, die blau sind und nach 2014 hergestellt wurden.	`SELECT model`
	`FROM cars`
	`WHERE color = 'blue'`
	`  AND year > 2014;`

Mit dem `OR` Operator können mehrere Bedingungen kombiniert werden. Datensätze, die einer der beiden Bedingungen entsprechen, die durch das `OR` verbunden sind, werden in die Ergebnisse aufgenommen. Die angegebene Abfrage findet Kunden, deren Staat entweder „CA“ oder „NY“ ist.	`SELECT name`
	`FROM customers`
	`WHERE state = 'CA'`
	`OR state = 'NY';`

`WHERE` wird verwendet, um Datensätze (Zeilen) zu filtern, die eine bestimmte Bedingung erfüllen. Die angegebene Abfrage wählt alle Datensätze aus, bei denen das pub_year gleich 2017 ist.	`SELECT title`
	`FROM library`
	`WHERE pub_year = 2017;`

Mit `LIKE` kann innerhalb einer `WHERE` ein bestimmtes Muster abgefragt werden. Die angegebene Abfrage findet jeden Film, dessen Titel mit „Star“ beginnt.	`SELECT name`
	`FROM movies`
	`WHERE name LIKE 'Star%';`

Der Platzhalter `%` kann mit `LIKE` verwendet werden, um keine oder mehrere nicht spezifizierte Zeichen zu finden. Die angegebene Abfrage findet jeden Film, der mit „The“ beginnt, gefolgt von keinem oder mehreren beliebigen Zeichen.	`SELECT name`
	`FROM movies`
	`WHERE name LIKE 'The%';`

Der Platzhalter `_` kann mit `LIKE` verwendet werden, um ein beliebiges einzelnes, nicht spezifiziertes Zeichen zu finden. Die angegebene Abfrage findet jeden Film, der mit einem einzelnen Zeichen beginnt, gefolgt von „ove“.	`SELECT name`
	`FROM movies`
	`WHERE name LIKE '_ove';`

Die `SELECT *` Anweisung gibt alle Spalten aus der angegebenen Tabelle in der Trefferliste zurück. Die angegebene Abfrage holt alle Spalten und Datensätze (Zeilen) aus der Tabelle „Filme“.	`SELECT *` `FROM movies;`

`ORDER BY` kann verwendet werden, um die Ergebnisse einer Spalte alphabetisch oder numerisch zu sortieren. Es kann auf zwei Arten sortiert werden:
• Mit `DESC` werden die Ergebnisse in absteigender Reihenfolge sortiert.
• Mit `ASC` werden die Ergebnisse in aufsteigender Reihenfolge sortiert (Standard).
	`SELECT *`
	`FROM contacts`
	`ORDER BY birth_date DESC;`

Der Operator `BETWEEN` kann zum Filtern nach einem Wertebereich verwendet werden. Bei dem Wertebereich kann es sich um Text, Zahlen oder Datumsdaten handeln. Die angegebene Abfrage findet alle Filme, die zwischen den Jahren 1980 und 1990 gedreht wurden.	`SELECT *`
	`FROM movies`
	`WHERE year BETWEEN 1980 AND 1990;`

Die Bedingung `LIMIT` wird verwendet, um die Ergebnisse auf eine bestimmte Anzahl von Zeilen zu begrenzen. Die angegebene Abfrage begrenzt die Ergebnismenge auf 5 Zeilen.	`SELECT *`
	`FROM movies`
	`LIMIT 5;`

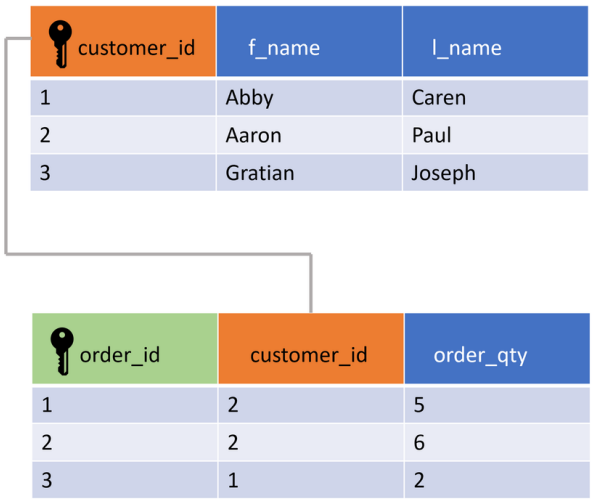
Spalteninhalte können `NULL` sein oder keinen Wert haben. Diese Datensätze können mit den Operatoren `IS NULL` und `IS NOT NULL` in Kombination mit `WHERE` abgeglichen werden. Die angegebene Abfrage wird alle Adressen anzeigen, bei denen die Adresse einen Wert hat oder nicht `NULL` ist.	`SELECT address`
	`FROM records`
	`WHERE address IS NOT NULL;`

Erstellen von Tabellen

Nach `CREATE TABLE` folgt der Tabellenname. Innerhalb der Klammern werden die Spalten und deren Datentyp aufgelistet. Der `PRIMARY KEY` gibt dabei an, in welcher Spalte die eindeutige Identifikationsnummer steht. Getrennt werden die einzelen Einträge mit Komma. Gibt es einen Verweis auf eine andere Tabelle, dann erfolgt dies durch den sogenannten `FOREIGN KEY`. Dieser gibt an in welcher Spalte die Referenz steht und auf welche Spalte einer anderen Tabelle referenziert wird.	`CREATE TABLE Personen {`
	`PersonenID INTEGER PRIMARY KEY,`
	`Nachname varchar(255),`
	`Vorname varchar(255),`
	`GeburtsOrtID INTEGER,`
	`FOREIGN KEY (GeburtsOrtID) REFERENCES Ort(OrtsID)`
	`}`

Foreign Key

Ein Fremdschlüssel ist ein Verweis von Datensätzen einer Tabelle auf den Primärschlüssel einer anderen Tabelle. Um mehrere Datensätze für eine bestimmte Zeile zu erhalten, spielt die Verwendung von Fremdschlüsseln eine wichtige Rolle. Um z. B. alle Bestellungen eines bestimmten Kunden zu verfolgen, kann die Tabelle Bestellung (unten im Bild) einen Fremdschlüssel enthalten.



customer_id	f_name	l_name
1	Abby	Caren
2	Aaron	Paul
3	Gratian	Joseph

order_id	customer_id	order_qty
1	2	5
2	2	6
3	1	2

Primary Key

Ein Primärschlüssel in einer SQL-Tabelle wird verwendet, um jeden Datensatz in dieser Tabelle eindeutig zu identifizieren. Ein Primärschlüssel kann nicht `NULL` sein. In diesem Beispiel ist `customer_id` der Primärschlüssel. Derselbe Wert kann in einer Spalte nicht noch einmal vorkommen. Primärschlüssel werden oft in `JOIN` Operationen verwendet.

customer_id	f_name	l_name
1	Abby	Caren
2	Aaron	Paul
3	Gratian	Joseph

Mehrere Tabellen verbinden

Bei einem Outer Join werden Zeilen aus verschiedenen Tabellen kombiniert, auch wenn die Join-Bedingung nicht erfüllt ist. Bei einem `LEFT JOIN` wird jede Zeile der linken Tabelle in die Ergebnismenge zurückgegeben. Wenn die Join-Bedingung nicht erfüllt ist, wird `NULL` verwendet, um die Spalten der rechten Tabelle aufzufüllen.

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
  ON table1.column_name =
table2.column_name;
```

`JOIN` ermöglicht die Rückgabe von Ergebnissen aus mehr als einer Tabelle, indem sie diese mit anderen Ergebnissen auf der Grundlage gemeinsamer Spaltenwerte verbindet, die mit `ON` angegeben werden. `INNER JOIN` ist der Standard `JOIN` und gibt nur Ergebnisse zurück, die der durch `ON` angegebenen Bedingung entsprechen.	``` SELECT * FROM books JOIN authors ON books.author_id = authors.id; ```

Created 2022-08-01 08:55:34 UTC by Joshua Lieder
Updated 2022-11-02 13:04:51 UTC