Relationale Datenbanken - Allgemein

- Anomalien
- Normalisierungsformen (NF1 - NF3)
- String in einer Tabelle ersetzen
- Benutzer in Postgresql anlegen

Anomalien

Einleitung

Wenn wir Datenbanken anlegen oder länger verwenden, können bei falsch programmierten Tabellen sogenannte Anomalien entstehen. Diese Anomalien gilt es zu vermeiden. Hier finden wir einige Anomalien, die in unserer Datenbank auftreten können.

Mutationsanomalie

In der Mutationsanomalie wird ein Wert durch einen Fehler **Falsch geschrieben**. Dies kann einfach durch einen Wert entstehen, für den es mehrere Schreibweisen gibt. z.B. **Straße** und **Strasse**.

Dadurch befinden sich dann später verschiedene Schreibweisen von Daten in der Datenbank. Dadurch können wir nicht zuverlässig nach Daten suchen und daher sind die Daten dann nicht richtig verwendbar.

Vorname	Nachname	Straße	PLZ	Ort
Peter	Petersen	Hauptstraße 30	24944	Flensburg
Gerda	Petersen	Hauptstrasse 30	24944	Flensburg

Einfüge Anomalie

Bei der einfüge Anomalie handelt es sich um einen Fehler, indem Daten, die notwendig sind mit "Dummy Daten" gefüllt werden. Also ungewollten falsch Informationen. Dies können **Null** Werte sein, als auch Werte mit den nicht erwarteten Werten.

Vorname	Nachname	Straße	PLZ	Ort
Peter	Petersen	Hauptstraße 30	?	Flensburg
Keine Daten	Petersen	Hauptstrasse 30	24944	Flensburg

Löschanomalie

Bei der Löschanomalie handelt es sich um einen Fehler, der entsteht, wenn Daten ungewollt gelöscht werden und so wichtige Teilinformationen fehlen.

Ein Beispiel z.B. wäre, wenn Daten, die sich in einer anderen Tabelle befinden, gelöscht werden, kann über den Fremdschlüssel einer Tabelle nicht mehr auf die entsprechenden Daten zugegriffen werden. So entsteht ein Informationsverlust.

Vorname	**Nachname**	**Straße**	**PLZ-ID**
Peter	Petersen	Hauptstraße 30	1
Gerda	Petersen	Hauptstrasse 30	1

Änderungsanomalie

Bei der Änderungsanomalie handelt es sich um eine Anomalie, die entsteht, wenn eine Änderung an einem Datensatz weitere Änderungen nach sich zieht.

Im Beispiel entsteht dies, wenn der Peter Petersen einen neuen Nachnamen erhält, muss an jeder Stelle, in der dieser Nachname auftaucht, der Name per Hand geändert werden. Dies gilt es auch zu vermeiden!

Normalisierungsformen (NF1 - NF3)

Einleitung

Wenn wir Datenbanken einrichten, wollen wir die Daten haben, die zu keinem Problem führen. Wenn wir eine Datenbank erstellen, deren Tabellen alle 3 Normalisierungsformen erfüllen, haben wir Tabellen, die uns die höchste Performance und Zuverlässigkeit bietet. Daher sollte man versuchen, die Datenbanktabellen nach diesen Regeln anzulegen.

Die Regeln müssen nach und nach erfüllt werden, das bedeutet es muss erst die **NF1**, dann die **NF2** und zum Schluss die **NF3** erfüllt sein. Also eine Tabelle, die sich in der **Normalisierungsform 2** befindet, hat automatisch die Anforderungen der **Normalisierungsform 1** erfüllt.

Normalform 0 (0NF)

Eine Tabelle liegt in der **"Nullten Normalform"** vor, wenn die Daten sich einfach in der Tabelle befinden. Das bedeutet, die Daten sind nicht atomar. Das bedeutet wir haben **Redundanzen** in der Datenbank ohne **signifikanten Informationsgewinn**. Dadurch ist unsere Tabelle durch sogenannte Anomalien gefährdet.

In dem ersten Beispiel haben wir eine Tabelle mit Kundendaten mit den Bestellungen, die diese getätigt haben. Es gibt in der Tabelle Redundanzen, z.B. **DHL** => **Deutsche Post DHL Group** als auch im Merkmal / Attribut **Adresse** bei der Postleitzahl mit der Ortsangabe. Im Weiteren haben wir Probleme, wenn wir z.B. Kunden nach Nachnamen sortieren wollen. Dies ist laut unserer aktuellen Tabelle nicht möglich.

Name	Adresse	Geburtstag	Bestellte Artikel	Spediteur	Spediteur Name
Peter Petersen	Mühlenstraße 30, 24944 Flensburg	01. Jan 1990	5x Stifte, 3x Bücher	DHL	Deutsche Post DHL Group
Hans Hansen	Hauptstraße 60, 24943 Flensburg	25.06.2000	30x Eddings, 6x Kopierpapier	Hermes	Hermes Germany GmbH
Hans Petersen	Apenrader Straße 90, 25001 Berlin	30.05.1999	25x Kinderriegel, 2x Radiergummi	DHL	Deutsche Post DHL Group
Petra Hansen	Bundesstraße 90, 24943 Flensburg	05. Jun 1975	50x Klebezettel, 1x Collegeblock	DHL	Deutsche Post DHL Group

Normalform 1 (1NF)

Die Tabelle in der **ersten Normalform** existiert, wenn folgende Bedingungen erfüllt sind:

1. Jedes Merkmal atomar ist
2. Nicht weiter unterteilbar ist
3. Listen aufgelöst sind

Es befinden sich in der Tabelle dennoch weiterhin Redundanzen, da diese noch nicht entfernt werden. In der 1. Normalisierungsform werden die Daten nur aufgeteilt, dass wir eine reine Tabelle erhalten, in dem wir die entsprechenden Daten in einzelnen Merkmalen aufgeteilt haben.

> Ab wann ein Wert **atomar** ist, hängt vom Nutzungskontext ab. Wenn man gewisse Teilungen nicht benötigt, ist dies nicht unbedingt nötig. In der Praxis empfiehlt sich trotzdem alles möglichst klein aufzuteilen.

Vorname	Nachname	Straße	Hausnummer	PLZ	Ort	Geburtstag	Anzahl Artikel	Bestellte Artikel	Spediteur	Spediteur Name
Peter	Petersen	Mühlenstraße	30	24944	Flensburg	01.01.1990	5	Stifte	DHL	Deutsche Post DHL Group
Peter	Petersen	Mühlenstraße	30	24944	Flensburg	01.01.1990	3	Bücher	DHL	Deutsche Post DHL Group
Hans	Hansen	Hauptstraße	60	24943	Flensburg	25.06.2000	30	Eddings	Hermes	Hermes Germany GmbH
Hans	Hansen	Hauptstraße	60	24943	Flensburg	25.06.2000	6	Kopierpapier	Hermes	Hermes Germany GmbH
Hans	Petersen	Apenrader Straße	90	10179	Berlin	30.05.1999	25	Kinderriegel	DHL	Deutsche Post DHL Group
Hans	Petersen	Apenrader Straße	90	10179	Berlin	30.05.1999	2	Radiergummi	DHL	Deutsche Post DHL Group
Petra	Hansen	Bundesstraße	90	24943	Flensburg	05.06.1975	50	Klebezettel	DHL	Deutsche Post DHL Group
Petra	Hansen	Bundesstraße	90	24943	Flensburg	05.06.1975	1	Collegeblock	DHL	Deutsche Post DHL Group

Normalform 2 (2NF)

In der zweiten Normalform muss die Tabelle in der ersten Normalform vorliegen, und alle Nichtschlüsselmerkmale voll funktional vom Primärschlüssel abhängen. Unterm Strich bedeutet dies, dass jedes Nichtschlüsselmerkmal aus dem Primärschlüssel ableitbar ist. Merkmale, die von einem Teilschlüssel abhängig sind, müssen in eine eigene Tabelle geschrieben werden. Hier teilen wir also die Tabellen in einzelne Tabellen auf.

Um jetzt eine Tabelle in die zweite Normalform zu übernehmen, müssen wir folgende Schritte durchführen:

1. Alle Nichtschlüsselmerkmale, die von einem Teilschlüssel funktional abhängig sind, bestimmen.
2. Aus den Teilschlüsseln mit allen funktional abhängigen Nichtschlüsselmerkmalen eigene Tabellen bilden.
3. Im letzten Schritt entfernen wir alle nicht voll funktional abhängigen Nichtschlüsselmerkmale.

tblKunden

ID_Kunde	Vorname	Nachname	Straße	Hausnumme	PLZ	Ort	Geburtstag
1	Peter	Petersen	Mühlenstraße	30	24944	Flensburg	01.01.1990
2	Hans	Hansen	Hauptstraße	60	24943	Flensburg	25.06.2000
3	Hans	Petersen	Apenrader Straße	90	10179	Flensburg	30.05.1999
4	Petra	Hansen	Bundesstraße	90	24943	Flensburg	05.06.1975

tblSpediteur

ID_Spediteur	Spediteur	Spediteur Name
1	DHL	Deutsche Post DHL Group
2	Hermes	Hermes Germany GmbH

tblBestellungen

ID_Bestellungen	ID_Kunde	ID_Spediteur
1	1	1
2	2	2
3	3	1
4	4	1

tblBestellteArtikel

ID_BestellteArtike	ID_Bestellung	Menge	Artikel
1	1	5	Stifte
2	1	3	Bücher
3	2	30	Eddings
4	2	6	Kopierpapier
5	3	25	Kinderriegel
6	3	2	Radiergummi
7	4	50	Klebezettel
8	4	1	Collegeblock

Normalform 3 (3NF)

Im letzten Schritt entfernen wir die letzten Redundanzen. Laut der Definition ist eine Tabelle in der Normalform 3, wenn jedes Schlüsselmerkmal nicht transitiv vom Primärschlüssel abhängig ist.

Im Weiteren bedeutet **Transitiv**, dass ein Merkmal einen Umweg nutzen kann, um funktional abhängig von einem anderen Merkmal abhängig ist. Unterm Strich sollen alle Merkmale nur von einem Primärschlüssel abhängig sein.

Um eine Tabelle in die dritte Normalform zu übernehmen, verwenden wir folgende Schritte:

1. Alle Nichtschlüsselmerkmale, die transitiv vom Schlüssel abhängen, bestimmen.
2. Im nächsten Schritt sollen aus diesen transitiv abhängigen Nichtschlüsselmerkmalen und den Nichtschlüsselmerkmalen, von denen diese funktional abhängig sind, eigene Tabellen bilden.
3. Im letzten Schritt entfernen wir alle transitiv abhängigen Nichtschlüsselmerkmale aus der Ursprungstabelle.

tblKunden

ID_Kunde	Vorname	Nachname	Straße	Hausnumme	PLZ	Geburt
1	Peter	Petersen	Mühlenstraße	30	24944	01.01.19
2	Hans	Hansen	Hauptstraße	60	24943	25.06.20
3	Hans	Petersen	Apenrader Straße	90	10179	30.05.19
4	Petra	Hansen	Bundesstraße	90	24943	05.06.19

tblSpediteur

ID_Spediteur	Spediteur	Spediteur Name
1	DHL	Deutsche Post DHL Group
2	Hermes	Hermes Germany GmbH

tblBestellungen

ID_Bestellungen	ID_Kunde	ID_Spediteur
1	1	1

String in einer Tabelle ersetzen

Einleitung

In diesem Beitrag gehe ich drauf ein, wie wir mit einem SQL Befehl einen String in einem Tabellensatz durch einen anderen String ersetzen können. Dabei wird jedoch nur der entsprechende Teil ersetzt, also es wird nicht der ganze Inhalt der Zelle gelöscht.

String ersetzen

Um den String in einer Tabelle zu ersetzen, müssen wir zuerst in die **Datenbank** wechseln, in dem sich die entsprechenden Tabellen befinden. Dazu verwenden wir den Befehl `use`.

```
use _production;
```

Im nächsten Schritt verwenden wir den **SQL Befehl** `REPLACE`, um den String zu setzen, wir müssen in dem Befehl angeben, in welchem Attribut sich die zu ersetzenden Werte befinden, und geben im Anschluss den zu suchenden Wert und den Wert ein, der dann eingetragen werden soll.

```
UPDATE <Tabelle>
SET
<Attribut> = REPLACE(<Attribut>, "<String-Suchen>", "<String-Ersetzten>"
```

Wenn wir dies auf ein Praxisbeispiel anwenden, könnte der Befehl so aussehen:

```
UPDATE customer
SET
street = REPLACE(street, "Strasse", "StraÃŸe");
```

Benutzer in Postgresql anlegen

Einleitung

In diesem Beitrag erläutere ich kurz, wie wir unter **Postgresql** einen Benutzer erstellen können und die entsprechenden Berechtigungen auf eine Datenbank zuweisen können.

Benutzer anlegen

Im ersten Schritt müssen wir einen Benutzer anlegen, dazu überlegen wir uns einen Benutzernamen als auch ein Kennwort für den Benutzer. Mit diesem Kennwort meldet sich der Benutzer in der Zukunft an.

```
CREATE USER <Benutzername> WITH PASSWORD '<Kennwort>';
```

Datenbank anlegen

Jetzt im zweiten Schritt müssen wir eine Datenbank anlegen. Bei dem Namen der Datenbank denken wir uns wieder einen Namen aus. Sinn macht ein Name, der eindeutig auf ein Projekt oder Programm zuweisbar ist. So weiß man immer, welche Daten in der Datenbank liegen.

```
CREATE DATABASE <Datenbank>;
```

Berechtigungen erteilen

Im letzten Schritt müssen wir jetzt nur noch dem Benutzer Zugriff auf die Datenbank erteilen. Wir erteilen dem Benutzer mit dem folgenden Befehl alle Berechtigungen **NUR** für diese Datenbank. Da wir davon ausgehen, dass der Benutzer Tabellen in der Datenbank erstellen, löschen und bearbeiten soll.

```
GRANT ALL PRIVILEGES ON DATABASE <Datenbank> to <Benutzer>;
```