DATENBANKEN & SQL

– Datenbanken erstellen –

Kurs am RRZK der Universität zu Köln

Rüdiger Voigt, M.A.

13.08.–17.08.2018

Übersicht

1. Normalformen
2. Storage Engine
3. Collation
4. Data Types
5. Geo-Daten
6. Index & Primary Key
7. Foreign Keys

Normalformen

Normalformen

Die Normalformen sind Regeln für die Struktur einer Datenbank.

Nichts erzwingt technisch ihre Einhaltung. Sie zu beachten eliminiert allerdings diverse Fehlerquellen und hilft Ihnen das Potential eines DBMS zu nutzen. Sobald die Datenbank nur etwas größer oder komplexer wird, sparen sie auch Arbeit. **Auf jeden Fall sollten Sie die ersten drei Normalformen anwenden!**

Erste Normalform: Beispiel

Klassisches Beispiel “Adresse”:

1. Ein einziges Feld mit allen Informationen ist sinnlos: sie können nicht nach einzelnen Bestandteilen filtern, oder sortieren. Wenn Sie Teilwerte brauchen, müssen Sie komplexe Verfahren anwenden.
2. Sie erfüllen die erste Normalform, wenn Sie dieses Feld in mehrere aufspalten, die sich logisch nicht weiter unterteilen lassen. Etwa: Vorname, Name, Straße, Hausnummer, Postleitzahl, Stadt und Land. Jetzt können Sie mit all diesen Informationen arbeiten und zum Beispiel alle Kunden in einem bestimmten Postleitzahlbereich heraussuchen.

Normalformen

Die Normalformen 1 bis 4 üben wir am praktischen Beispiel.

Storage Engine

Storage Engine

Die Storage Engine ist die Methode mittels welcher das DBMS Ihre Daten speichert.

Hier gibt es abhängig von Ihrem DBMS meist mehrere zur Auswahl, die unterschiedliche Vor- und Nachteile aufweisen:

- Es gibt Storage Engines, die Daten gleich komprimiert auf der Festplatte abspeichern. Das ist ideal für Daten, die Sie speichern müssen, aber selten bis nie wieder lesen – etwa alte Log-Daten.
- Andere speichern Daten extrem schnell, aber auch flüchtig im RAM-Speicher.
- ...

Storage Engine für MariaDB

Im Fall von MariaDB ist **fast immer XtraDB die richtige Wahl**. Das ist eine Weiterentwicklung von InnoDB und **gibt sich im Frontend als InnoDB aus** (aus Kompatibilitätsgründen).

Die Vorteile von XtraDB:

- relativ schnell beim Lesen als auch beim Schreiben von Daten
- unterstützt Foreign Key Constraints (wichtig!)
- unterstützt Transaktionen
- über Jahre erprobt mit sehr großen Datenmengen

Collation

Collation

Die Auswahl der Collation hat weitreichende Folgen. Sie besteht aus drei Bestandteilen:

1. dem Encoding
2. der Locale
3. der Information, ob sie case sensitive oder case insensitive ist, also ob sie zwischen Groß- und Kleinschreibung unterscheidet oder nicht.

Encoding I

Intern speichert der Computer Alles als eine Folge von Einsen und Nullen, zum Beispiel 10110001100101100000101001001 . . . .

Das Encoding definiert in welcher Art und Weise diese Folge gelesen werden muss um Sinn zu ergeben.

Also:

- Wieviele Bit (in diesem Fall Stellen) stehen für ein Zeichen?
- Gibt es Folgen mit besonderen Bedeutungen?
- Für welches Zeichen steht welche Folge?
- . . .

Encoding II

Aus historischen Gründen gibt es kein überall verwendetes Encoding. Nicht jedes Encoding unterstützt jedes Zeichen.

Versuch der Lösung: **UNICODE**. Ein Standard für prinzipiell jedes Zeichen der Welt angefangen von Maja-Runen bis hin zu Emojis. Noch nicht allgegenwärtig, aber weit verbreitet.

Es gibt verschiedene Encodings um Unicode zu speichern, aber die für den europäischen Sprachraum gebräuchlichste / effizienteste Variante ist utf8.

Locale

Die Locale speichert im Wesentlichen die Information über die besonderen Regeln eines Landes ab:

- Es gibt Länder, welche das lateinische Alphabet nutzen, aber eine andere Sortierreihenfolge haben als die deutsche Sprache.
- **extrem wichtig:** wird ein Dezimalpunkt, oder ein Dezimalkomma verwendet?
- ...

Collation: Empfehlung

Sinnvoll für die meisten Projekte:
utf8mb4_general_ci oder **utf8mb4_german2_ci**

Data Types

Generell

- Wenn Sie eine Tabelle neu anlegen, *müssen Sie* den Data Type jeder Spalte festlegen!
- **Der Data Type *determiniert* welche Inhalte die Spalte aufnehmen kann und wie das DBMS die Spalte bei Abfragen behandelt.**
- Auf diesen Folien finden Sie **nur die gebräuchlichsten** Data Types. MariaDB und die meisten anderen DBMS verfügen über erheblich mehr!

Data Types für Zahlen I

Zahlenwerte sind entweder als signed oder unsigned zu definieren. Letzteres bedeutet, dass negative Werte nicht zulässig sind. Mit signed können Sie entsprechend auch negative Zahlen speichern. Das geht dann allerdings zu Lasten der Größe der zu speichernden Zahlen.

- **int**
  Integer sind Ganzzahlen, also 1 oder 2 oder 3, aber nicht 1,543 oder ähnliche.
- **float / double**
  Eine Floating Point Number / Gleitkommazahl. Verwenden Sie bei MariaDB bevorzugt double (mehr Platz für Nachkommastellen), weil MariaDB mit floats intern so rechnet wie mit doubles.
- Viele DBMS haben spezielle Datentypen für Geldbeträge (maximal 2 Nachkommastellen, Verhalten beim Runden, . . .).

Data Types für Zahlen II

- Es gibt noch eine Reihe weiterer Data Types für Zahlen. Wenn Sie mit hoher Präzision rechnen / speichern müssen, schauen Sie sich unbedingt die Anleitung Ihres DBMS an!

Telefonnummern u.ä. speichern

Klassische Falle

Angenommen Sie möchten die Telefonnummer +49-(0)221470-0 speichern. Es ist intiuitives Vorgehen Sonderzeichen zu entfernen / zu ersetzen und die Nummer zum Beispiel so abzuspeichern:
INSERT INTO phoneNumbers (telNo) VALUES (00492214700)
...
Falls die Spalte telNo das Datenformat Integer hat, speichert die Datenbank allerdings den Wert
Falls die Spalte telNo das Datenformat float hat, speichert sie den Wert

Falls führende oder folgende Nullen eine Bedeutung haben, müssen Sie die Folge in einem Text- statt einem Zahlenformat abspeichern!

Verfälschte Werte ohne Fehlermeldung

Klassische Falle beim Speichern von Kommazahlen

Die meisten Collations erwarten Kommazahlen im amerikanischen Format, also mit **Dezimalpunkt statt Dezimalkomma**.
INSERT INTO numbers (floatCo) VALUES (3,4 , 2,7, 5,3)

INSERT INTO numbers (floatCo) VALUES ('3,4' , '2,7', '5,3')

Die Datenbank reagiert mit einer Warnung, aber nicht mit einem Fehler. Das heißt das DBMS speichert die Daten in veränderter Form. Warnungen werden in vielen Standardkonfigurationen Ihnen nicht angezeigt! Sie können also wochenlang falsche Daten speichern und finden es erst raus, wenn Sie die Daten abfragen!

Data Types für Text I

- **char**
  Eine Textfolge (string) mit fester Länge. Sie müssen angeben wie lang der Text maximal sein darf. Es werden so viele Byte Speicherplatz belegt, wie der Maximalwert brauchen würde.
- **varchar**
  Wie char, aber mit variabler Länge. Es wird nur soviel Speicherplatz belegt, wie der String tatsächlich braucht plus 2 Bytes.
- **text**
  Im Prinzip wie varchar, kann aber wesentlich längere Texte aufnehmen. Gibt es in verschiedenen Varianten hinsichtlich des Speicherplatz - longtext etwa kann bis zu 4 Gigabyte pro Zelle speichern. Ein Index auf eine solche Spalte ist meist eine schlechte Idee.

Data Types für Datum und Zeit

- **timestamp**
  Ein Datum (in der Regel im amerikanischen Format) gefolgt von einer Uhrzeit inklusive Sekunden (24-Stunden Format).
- **date**
  Nur das Datum. Einige DBMS speichern hier aber einfach im Format timestamp ab. Beim Umgang mit Datumswerten sind die Funktionen GETDATE() und DATEDIFF() oft hilfreich.

Geo-Daten

GIS Features

Geographische Daten sind ein Sonderfall:

- GIS-Features wurden in MySQL über Jahre stiefmütterlich behandelt.
- Formal standen die Datentypen zur Verfügung, aber zum Beispiel war es nicht möglich korrekt zu prüfen ob eine Koordinate in einem Staat lag. Anstatt der eigentlichen Geometrie der Grenzen wurde um das Layer-Element eine Box gezogen. Elementare Features wie Projektionen fehlten.
- Als MariaDB anfing die Features zu überarbeiten und zu erweitern, zog MySQL nach.

MariaDB versus PostgreSQL / PostGIS

- Die PostGIS Variante hat einen Entwicklungsvorsprung von mehreren Jahren.
- Bei einigen Features hinkt MariaDB der PostGIS Variante deutlich hinterher:
  - Aktuell noch immer keinen Index auf geographische Daten bei transaktionssicheren Storage Engines.
  - Indizes auf geographische Daten haben wohl auch ein Problem mit NULL Werten.
  - keine 3D-Distanz
  - ...
- Der Teil der MariaDB-Dokumentation mit Bezug zu GIS ist vielfach lange nicht mehr aktualsiert.

tatsächliche GIS Features

- grundlegende Features sind vorhanden:
  - Punkte (mit Koordinaten)
  - Shapes
  - Buffer
  - ...
- grundlegende Abfragen (innerhalb einer Geometrie, Distanz, ...) sind möglich
- Videos zum Thema:
  - von 2013, aber gut:
    `https://www.youtube.com/watch?v=8beAascKycY`
  - `https://www.youtube.com/watch?v=pO-4SfRHblg`

Fazit GIS Features

- Wenn Sie mit grundlegenden GIS-Features auskommen, können Sie einfach MariaDB nutzen.
- Für komplexe geographische Analysen empfiehlt sich eher PostgreSQL mit der PostGIS-Erweiterung:
  - Sie können Ihre in diesem Kurs erlangten Kenntnisse (mit kleinen Anpassungen) anwenden.
  - Wenn Sie auf unterschiedliche Ports[1] eingestellt werden, können Sie verschiedene Datenbanksysteme auf dem gleichen Rechner betreiben. Der Datenaustausch ist nur schwieriger als innerhalb des gleichen DBMS.

[1] `https://de.wikipedia.org/wiki/Port_(Protokoll)`

Index & Primary Key

Index: Sinn und Zweck

Vereinfacht gesprochen ist ein Index eine Art Inhaltsverzeichnis für eine Spalte.

Statements (SELECT, UPDATE, DELETE), welche mit einer WHERE Bedingung Werte einer Spalte prüfen, werden durch einen für diese Spalte vorhandenen Index meist *enorm* beschleunigt!

Falls das DBMS die gesamte Tabelle in den RAM laden kann ist der Effekt scheinbar klein – ob das geht ist aber abhängig von der aktuellen Last auf dem System!
Bei großen Datenbanken kann bei vorhandenem Index ein Befehl oft **in Sekundenbruchteilen statt in Minuten** angewendet werden. Bei sehr häufigen Zugriffen auf eine Datenbank (Subqueries!) wird die Beschleunigung sehr schnell relevant.

Definition: Primary Key

Ein Primary Key (PK) ist eine Sonderform eines Index. Wenn Sie eine Spalte als PK definiert haben, brauchen Sie keinen weiteren Index dafür zu definieren.
Alle Werte der PK-Spalte müssen unique sein – im Gegensatz zum normalen Index sind Dubletten unzulässig und werden vom DBMS verhindert.
Der Primary Key dient zur eindeutigen Identifikation einer Zeile! Wenn Sie mehrere Tabellen mit einem JOIN verbinden, matchen Sie diese in der Regel anhand des PK. Jede Tabelle kann nur einen PK haben.

Auswahl des Primary Key

In vielen Fällen drängt sich ein eindeutiges Merkmal zur Identifizierung auf. Sei es die Steueridentifikationsnummer, eine Kundennummer, die Matrikelnummer, ...

In allen anderen Fällen generieren Sie einfach eines. Legen Sie eine Spalte vom Typ Integer (zum Beispiel mit dem Namen id) an und definieren Sie diese als Auto increment und Primary Key. Das können Sie auch tun nachdem bereits Daten in der Tabelle sind. Jede Zeile erhält dann eine eindeutige Nummer, die kontinuierlich hochgezählt wird und nicht doppelt vorkommt.

Foreign Key Constraints

Noisy Data I

15 Varianten für USD aus einem realen sozialwissenschaftlichen Datensatz mit weniger als 1000 Zeilen:

Dollar, dollars, Dollars, US, U.S.Dollas, us dollars, US dollars, U.S.Dollars, U.S. Dollars, million $, million dollars, bln. USD, billion dollars, USD per year, $ (not sure if Canadian or US)

Daraus ergeben sich drei Probleme:

1. Ein SELECT müsste alle Varianten und Rechtschreibfehler abdecken.
2. Bei vielen Varianten ist nur zu raten, ob es sich um australische, kanadische oder US-amerikanische Dollar handelt
3. Faktoren und Zeiteinheiten werden mit der Währung gemischt. Ein ORDER BY auf den numerischen Wert ergibt ein falsches Ergebnis.

Noisy Data II

Eine solche Datenqualität trifft man leider immer wieder an, weil Datenbanken in einfachen Text- oder Excel-Dateien oder ohne wirksame Kontrollen angelegt wurden.

Oft haben dutzende oder hunderte Personen über Jahre Inhalte eingepflegt. Deren Fehler sind vielfältig und sie zu korrigieren kann Stunden oder auch Tage in Anspruch nehmen.

Bei Berechnungen mit solchen Daten gilt:
garbage in –> garbage out!

Noisy Data III

Maßnahmen:

1. Eingabeinterface so umgestalten, dass Endnutzer nur valide Daten eingeben können.
2. Aufräumen (Varianten zusammenführen, Zahlenwerte in die gleiche Einheit umrechnen, . . .)
3. **Foreign Key Constraints einführen!**

Foreign Key Constraints

Foreign Key Constraints / Fremdschlüssel stellen eine Verbindung zwischen einer Spalte einer <child table> zu einer Spalte einer <parent table> (in der Regel deren Primary Key, aber auf jeden Fall als unique definiert) her.

Folgen / Vorteile:

- In der entsprechenden Spalte der <child table> können nur noch Werte eingeben werden, die in der verknüpften Spalte der <parent table> vorhanden sind.
- Sie können definieren was passiert, wenn der Wert in der <parent table> geändert / gelöscht werden soll, aber Zeilen der <child table> diesen referenzieren.

Foreign Key Constraints

Konsistenz erzwingen

Fremdschlüsselbeziehungen / Foreign Key Constraints sind ein zentraler Mechanismus um Konsistenz innerhalb Ihrer Datenbank sicherzustellen. Richtig eingesetzt eliminieren Fremdschlüssel diverse Fehlerquellen.

Dabei können sich Fremdschlüssel auch auf Tabellen in anderen Datenbanken innerhalb des gleichen DBMS beziehen.

Wiederholung: Sonderfall NULL I

Warnung

1. NULL signalsiert eine leere Zelle. NULL hat nichts mit dem Zahlenwert 0 zu tun und ist auch kein String wie “NULL” oder ‘NULL’.
2. MariaDB: “CONCAT() returns NULL if any argument is NULL.”

Das können Sie mittels IFNULL() oder COALESCE() abfangen.

Tipp für den Import von Daten:
Definieren Sie für das Feld einen Default-Wert, den zum Beispiel R, STATA oder SPSS als Fehlercode versteht. Wichtig ist dabei, dass Sie den Datentyp nicht wechseln.

NULL erlauben oder unterbinden

Sie können / sollten für jede Spalte festlegen, ob Sie NULL in dieser Spalte erlauben möchten oder nicht.

Wenn Sie verhindern, dass eine Zelle NULL sein kann, haben Sie etwas, das man **Pflichtfeld** nennen könnte.
Idealerweise kombinieren Sie das mit einem Fremdschlüssel um sicherzustellen, dass **immer ein sinnvoller Wert** im Feld stehen muss.

Foreign Key Constraints: Beispiel 1

Sie haben eine <parent table> namens currency mit einer Spalte iso. Enthaltene Werte USD, EUR, GBP. Sie haben eine <child table> payments mit den Spalten value und currencyID.

Wenn Sie definieren, dass payment.currencyID eine Fremdschlüsselbeziehung zu currency.iso hat, werden bei einem INSERT in die Tabelle payment ausschließlich USD, EUR oder GBP als Eingabe für das Feld currencyID akzeptiert. Das DBMS weigert sich eine Zeile abzuspeichern, die für dieses Feld einen falschen Wert enthält. Auch durch UPDATE ist es nicht möglich payment.currencyID auf einen ungültigen Wert zu ändern.

ON DELETE / ON UPDATE I

Sie können verschiedene Aktionen definieren, die ausgeführt werden, wenn eine Zeile in der <parent table> gelöscht wird (ON DELETE) oder der Wert in der referenzierten Spalte der <parent table> geändert wird (ON UPDATE).

Die Aktionen werden nur ausgeführt, wenn es mindestens eine Zeile in der <child table> gibt, welche die betroffene Zeile in der <parent table> referenziert.

Mögliche Aktionen sind RESTRICT, CASCADE und SET NULL.

mögliche Aktionen bei ON DELETE

- RESTRICT:
  Das Löschen der Zeile in der <parent table> wird unterbunden, solange es in der <child table> eine referenzierende Zeile gibt.
- CASCADE:
  Wenn die Zeile in der <parent table> gelöscht wird, werden alle diese eine referenzierenden Zeilen in der <child table> ebenfalls gelöscht!
- SET NULL:
  Das Löschen in der <parent table> wird nicht unterbunden.

mögliche Aktionen bei ON UPDATE

- RESTRICT:
  Die Änderung der referenzierten Spalte in der <parent table> wird unterbunden, solange es in der <child table> eine referenzierende Zeile gibt.
- CASCADE:
  Wenn der Wert der Spalte in der <parent table> geändert wird, wird er auch in der <child table> geändert.
- SET NULL:
  Die Änderung in der <parent table> wird nicht unterbunden.

Foreign Key Constraints: Beispiel 2

Die Datenbank Ihres Unternehmens enthält die Tabellen vertraege und kunden. Die Spalte vertraege.kundenId ist als Fremdschlüssel auf kunden.id definiert. Ferner haben Sie eingestellt, dass vertraege.kundenId niemals NULL sein darf. Zusätzlich haben Sie die Optionen "ON DELETE RESTRICT" und "ON UPDATE CASCADE" gesetzt.

1. Jeder Vertrag wird einem Kunden zugeordnet. (NOT NULL)
2. Verträge können nur existierenden Kunden zugeordnet werden. (Foreign Key)
3. Sie können nicht versehentlich einen Kunden löschen, dem ein Vertrag zugeordnet sind. (ON DELETE RESTRICT)
4. Sie kommen niemals in die Verlegenheit, dass Sie einen Vertrag haben, aber diesen keinem Kunden zuordnen können. (Löschen des Kunden ist bereits abgefangen / Änderung der Kundennummer kaskadiert)

Anlegen von Foreign Key Constraints

1. **Auf den zu verbinden Spalten muss jeweils ein Index liegen!** Dabei ist die referenzierte Spalte der parent table in der Regel deren Primary Key (also bereits ein Index).
2. Wählen Sie eine Option für ON DELETE.
3. Wählen Sie eine Option für ON UPDATE.
4. Benennen Sie den Constraint. Wenn ein INSERT oder ein UPDATE einen Fehler wirft, lesen Sie diesen Namen in der Fehlermeldung und finden schneller die Ursache.

Den kompletten Foliensatz finden Sie in aktueller Version unter:

`https://www.ruediger-voigt.eu/kurs-datenbanken-und-sql.html`