Einführung in Datenbanken

Kapitel 16: Einführung in den Datenbank-Entwurf

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2020/21

http://www.informatik.uni-halle.de/~brass/db20/

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Die drei Phasen des Datenbank-Entwurfs erklären.

  Wozu sind verschiedene Phasen nützlich?

- Die Korrektheit von Datenbank-Schemata für eine Anwendung diskutieren.

- Bedeutung und Nutzen von Integritätsbedingungen erläutern.

  Und erkennen, wo in eigenen Schemata Integritätsbedingungen nötig sind.

- `CHECK`-Constraints im `CREATE TABLE` verwenden.

- Grundlegende Aspekte der Qualität von DB-Schemata diskutieren und bei der Entwicklung eigener Schemata berücksichtigen.

Inhalt

1. DB-Entwurf
2. Entwurfs-Phasen
3. Integritätsbedingungen
4. CHECK-Constraints in SQL
5. Schema-Qualität

Datenbank-Entwurf (1)

- Ziel: Entwicklung von Programmen,
  um gegebene Aufgaben der realen Welt zu bearbeiten.
- Diese Programme benötigen persistente Daten.
- Verwendung von Software Engineering Methoden
  (aber spezialisiert auf datenintensive Programme).
- DB-Entwurf ist der Prozess der Entwicklung eines
  DB-Schemas für eine gegebene Anwendung.

Es ist eine Teilaufgabe des allgemeinen Software Engineering.

Datenbank-Entwurf (2)

- Die Anforderungen an Programme und Daten sind verflochten, beide hängen voneinander ab:
  - Das Schema muss die Daten enthalten, die von den Programmen benötigt werden.
  - Die Programme sind meist leicht zu spezifizieren, wenn die zu verwaltenden Daten festliegen.
- Daten sind aber eine unabhängige Ressource:
  - Oft werden später zusätzliche Programme entwickelt, die auf den vorhandenen Daten beruhen.
  - Auch ad-hoc-Anfragen sind möglich.

Datenbank-Entwurf (3)

- Während des DB-Entwurfs muss ein formales Modell von Teilen der realen Welt („Miniwelt“) erstellt werden.

  Fragen über die reale Welt sollen von der Datenbank beantwortet werden.
  Eine Liste solcher Fragen kann ein wichtiger Input für den DB-Entwurf sein.

- Die reale Welt ist ein Maß für die Korrektheit des Schemas:
  - Die Menge der Datenbank-Zustände zu dem Schema sollte genau
  - den möglichen Situationen der realen Welt entsprechen.

    Es wäre z.B. schlecht, wenn Situationen in der realen Welt auftreten, die in der Datenbank nicht abgespeichert (repräsentiert) werden können.
    Umgekehrt sollten auch Datenbank-Zustände ausgeschlossen werden (mit Hilfe von Integritätsbedingungen), die keiner möglichen Situation der realen Welt entsprechen.

Datenbank-Entwurf (4)

Datenbank-Entwurf ist nicht leicht:

- Der Entwickler muss den Anwendungsbereich verstehen.

  U.a. führt man dazu Gespräche (Interviews) mit Anwendungsexperten (z.B. spätere Nutzer der Datenbank). Diese erwähnen aber eventuell Dinge nicht, die für sie selbstverständlich sind.

- Ausnahmen: Die reale Welt ist sehr flexibel.

  Und häufig denkt der Anwendungsexperte, der dem Datenbank-Spezialisten die Anwendung erklärt, nicht an die Ausnahmen. Sie sind eben selten.

- Größe: DB-Schemata können sehr groß sein.

  Nach [https://www.berater-wiki.de/Tabellen] soll es in SAP ca. 112.000 Tabellen geben. Ich hatte früher von 15.000 Tabellen gehört.

- Man muss aus einem oder wenigen Zuständen, die man beobachten kann, auf alle möglichen Zustände extrapolieren.

Inhalt

1. DB-Entwurf
2. Entwurfs-Phasen
3. Integritätsbedingungen
4. CHECK-Constraints in SQL
5. Schema-Qualität

Entwurfs-Phasen (1)

Die klassischen drei Phasen des Datenbank-Entwurfs:

- Konzeptioneller DB-Entwurf: In dieser Phase wird ein Modell der Miniwelt in einem konzeptionellen Datenmodell erstellt (z.B. dem Entity-Relationship Modell).

  Statt „konzeptionell“ kann man auch „konzeptuell“ sagen.

  Der Fokus liegt hier auf einer Beschreibung der realen Welt, soweit sie für das Projekt wichtig ist, und nicht auf der Datenbank.

- Logischer DB-Entwurf: Hier wird das Schema aus dem konzeptionellen Datenmodell in das Datenmodell des DBMS transformiert (heute fast immer das relationale Modell).

- Physischer DB-Entwurf: Ziel ist die Erfüllung der Leistungsanforderungen an das endgültige System.

  Indexe und Speicherparameter werden in dieser Phase gewählt.

Entwurfs-Phasen (2)

Warum verschiedene Entwicklungsphasen?

- Probleme können getrennt und nacheinander abgearbeitet werden.

  Man kann sich auf jeweils eine Schwierigkeit konzentrieren.

  Entscheidungen werden möglichst so in eine Reihenfolge gebracht, dass sie nicht wechselseitig von einander abhängen.

- Z.B. muss man während der konzeptionellen Entwicklung nicht über die Performance oder über Einschränkungen verschiedener DBMS nachdenken.

  Im Mittelpunkt: Entwicklung eines korrekten Modells der realen Welt.

- DBMS-Features beeinflussen den konzeptionellen Entwurf nicht (und nur teilweise den logischen).

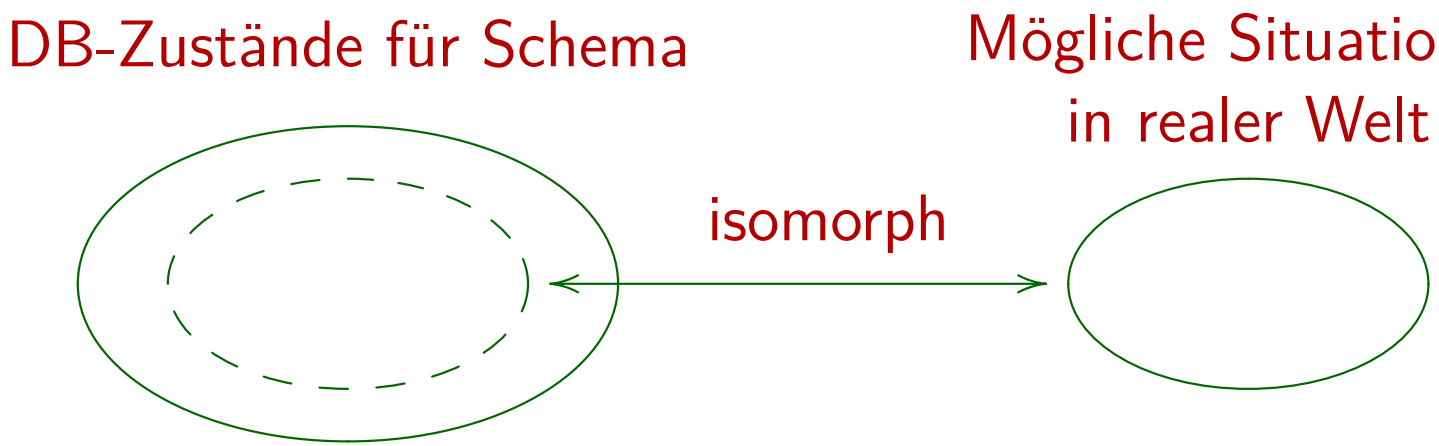
  Somit wird der konzeptionelle Entwurf nicht ungültig, wenn später ein anderes DBMS verwendet wird.

Inhalt

1. DB-Entwurf
2. Entwurfs-Phasen
3. Integritätsbedingungen
4. CHECK-Constraints in SQL
5. Schema-Qualität

Gültige DB-Zustände (1)

- Ziel des DB-Entwurfs: Die DB sollte ein Abbild der relevanten Teilmenge der realen Welt sein.
- Aber die Definition der Grundstruktur (Tabellen und Spalten bzw. Entities, Attribute und Relationships) erlaubt oft zu viele DB-Zustände (die sinnlos/illegal sind):



Gültige DB-Zustände (2)

KUNDE				
Kund_Nr	Name	Geb_Jahr	Stadt	···
1	Müller	1936	Berlin	···
2	Schmidt	1965	Hamburg	···
3	Schneider	64	München	···
3	Fischer	2007	Köln	···

- Kundennummern müssen eindeutig sein.

  Das ist für die Datenstruktur wichtig.

- Das Geburtsjahr muss größer als ca. 1880 sein.

  Das ist ein naturwissenschaftliches Gesetz.

- Kunden müssen mindestens 18 Jahre alt sein.

  Das ist eine Geschäftsregel bzw. eine gesetzliche Vorschrift.

Gültige DB-Zustände (3)

- Zwei Fehlerarten müssen unterschieden werden:
  - Eingabe falscher Daten, d.h. der DB-Zustand entspricht zu einer anderen Situation der realen Welt als der tatsächlichen aktuellen Situation.

    Z.B. 8 Punkte für Hausaufgabe 1 in der DB vs. 10 in der realen Welt. Dann ist der DB-Zustand falsch, aber nicht das Schema. Übung: Was kann man machen, um solche Fehler zu vermeiden?
  - Eingabe von Daten, die keinen Sinn machen oder die illegal sind.

    Z.B. -5 Punkte für eine Aufgabe oder zwei Einträge für die gleiche Aufgabe und den gleichen Studenten. Wenn solche unmöglichen Daten eingegeben werden können, ist das DB-Schema falsch.

Gültige DB-Zustände (4)

- Wenn die DB illegale oder sinnlose Daten enthält, wird sie mit unserem allgemeinen Verständnis der realen Welt inkonsistent.
- Sind Annahmen des Programmierers über die Daten verletzt, so kann das unvorhersehbare Auswirkungen haben.

  Z.B. der Programmierer nimmt an, dass keine Nullwerte auftreten (siehe Kapitel 12). Also verwendet er keine Indikatorvariable beim Abfragen der Daten. Dies funktioniert, solange keine Nullwerte auftreten. Wenn aber das Schema Nullwerte zulässt und jemand irgendwann einen Nullwert eingibt, wird das Programm abstürzen (mit einer unverständlichen Fehlermeldung).
- Man sichere Annahmen über Integritätsbedingungen!

Integritätsbedingungen (1)

- Integritätsbedingungen („Constraints“, IBen)
  sind Bedingungen, die jeder DB-Zustand erfüllen muss.
- Sie schränken die Menge möglicher DB-Zustände ein.

  Idealerweise zu Abbildern von Situationen der realen Welt.
- Integritätsbedingungen können als Teil des DB-Schemas definiert werden.
- Das DBMS lehnt jede Änderung ab, die eine Bedingung verletzen würde.

  Somit werden ungültige Zustände ausgeschlossen. Heutige DBMS erlauben keine beliebigen Bedingungen, nur spezielle Fälle, siehe unten.

Integritätsbedingungen (2)

- Jedes Datenmodell unterstützt bestimmte Arten von Constraints speziell (u.a. besondere Notation).

  Z.B. haben im ER-Modell Schlüssel, Kardinalitätsbedingungen und die Einschränkungen bei schwachen Entities eine spezielle graphische Syntax. Diese Arten von Integritätsbedingungen werden unten erklärt.

- Werden Integritätsbedingungen benötigt, die das Datenmodell bzw. DBMS nicht unterstützt, sollte man sie dennoch beim DB-Entwurf dokumentieren.

  Z.B. als logische Formel oder in natürlicher Sprache. Man kann auch eine Anfrage schreiben, die die Verletzungen der Bedingung ausgibt (d.h. das Anfrageergebnis muss immer leer sein). Zukünftige Systeme werden wahrscheinlich auch allgemeinere Constraints unterstützen.

Integritätsbedingungen (3)

- Sicherstellung allgemeiner Integritätsbedingungen:
  - Die zur Dateneingabe verwendeten Anwendungsprogramme überprüfen die Bedingungen.
  - Änderungen nur über gespeicherte Prozeduren im DBMS, die die Bedingungen überprüfen.
  - Überprüfung der Integritätsbedingung in Triggern.

    Trigger sind im DBMS gespeicherte Prozeduren, die bei definierten Events, hier der Änderung einer bestimmten Tabelle, automatisch aufgerufen werden.
  - Von Zeit zu Zeit wird eine Anfrage ausgeführt, die alle Verletzungen der Bedingung findet.

    Eventuell wurden die fehlerhaften Daten aber schon verwendet.

Triviale/Implizierte IBen

- Eine triviale Bedingung ist eine Bedingung, die immer erfüllt ist (logisch äquivalent zu „wahr“).
- Eine Bedingung A impliziert eine Bedingung B, wenn aus A ist wahr folgt, dass auch B wahr ist.

  D.h. die Zustände, die A erfüllen, sind eine Teilmenge der Zustände, die B erfüllen. Das ist die Standarddefinition der logischen Implikation.

- Z.B. A: „Jeder Dozent hält 1 oder 2 Vorlesungen“.
  B: „Dozenten können max. 4 Vorl. halten“.
- Man sollte keine trivialen oder implizierten Constraints spezifizieren.

  Erhöht Komplikationen, Menge der gültigen Zustände wird nicht geändert.

Zusammenfassung (1)

Warum Integritätsbedingungen (IBen) festlegen?

- Etwas Schutz vor Eingabefehlern.
- Integritätsbedingungen enthalten Wissen über DB-Zustände.
- Erzwingung von Gesetzen/Unternehmensstandards.
- Schutz vor Inkonsistenz bei redundant gespeicherten Daten (aus anderen Daten berechenbar).
- Anfragen/Programme werden einfacher, wenn der Programmierer weniger Fälle behandeln muss (die IBen reduzieren ja die Menge möglicher Zustände).

  Z.B. keine Indikatorvariable bei Spalten ohne Nullwerte.

Zusammenfassung (2)

Integritätsbedingungen und Ausnahmen:

- Integritätsbedingungen lassen keine Ausnahmen zu.

  Jeder Versuch, ungültige Daten einzugeben, wird abgelehnt.

- Es kommt manchmal vor, dass ein Datenbanksystem in Ausnahmesituationen wegen der festgelegten Integritätsbedingungen als zu unflexibel erscheint.

  Wenn in der realen Welt Situationen vorkommen, die in der Datenbank nicht abgebildet werden können, ist das DB-Schema strenggenommen falsch.

- Nur Bedingungen, die ohne Zweifel immer gelten müssen, sollten als Integritätsbedingung festgelegt werden.

  „Normalerweise“-Bedingungen könnten nützlich sein, aber nicht als IBen.
  Z.B. können Programme nachfragen, wenn ein neuer Kunde über 100 ist.
  Die Daten-Verteilung ist auch nützlich für den physischen Entwurf.

Inhalt

1. DB-Entwurf
2. Entwurfs-Phasen
3. Integritätsbedingungen
4. CHECK-Constraints in SQL
5. Schema-Qualität

CHECK-Constraints in SQL (1)

- In der `CREATE TABLE`-Anweisung in SQL ist es möglich, Bedingungen anzugeben, die für alle Tabellenzeilen gelten müssen:

```
CREATE TABLE STUDENTEN(
                SID         NUMERIC(3)  NOT NULL,
                VORNAME     VARCHAR(20) NOT NULL,
                NACHNAME    VARCHAR(20) NOT NULL,
                EMAIL       VARCHAR(80),
                PRIMARY KEY(SID),
                UNIQUE(VORNAME, NACHNAME),
                CHECK(SID > 0))
```

- Versucht man, mit `INSERT` eine Zeile mit negativer `SID` einzufügen, gibt es eine Fehlermeldung.

  Und die Zeile wird natürlich nicht eingefügt.

CHECK-Constraints in SQL (2)

- Man kann jede `WHERE`-Bedingung ohne Unteranfragen als `CHECK`-Constraint benutzen.

  D.h. auch `AND`, `OR`, `NOT`, `BETWEEN`, `LIKE`, `IS NULL`, `IN (`⟨Werteliste⟩`)`.

- D.h. erlaubt sind quantorenfreie Formeln.

  Sie sind implizit allquantifiziert über allen Zeilen der Relation.

- Dadurch, dass sich die Formel auf jeder Zeile einzeln auswerten läßt, reicht es, sie für eingefügte bzw. durch Updates veränderte Zeilen zu prüfen.

  Da die Tabelle anfangs leer ist, gilt die Bedingung da natürlich für alle Zeilen (Induktionsanfang). Wenn die Bedingung vor der Änderung gilt (Induktionsvoraussetzung), und sie für die eingefügten bzw. modifizierten Zeilen geprüft wurde, gilt sie auch nach der Änderung (Induktionsschritt).

- Nach dem SQL-Standard wären auch Unteranfragen erlaubt, diese sind bisher aber in keinem DBMS implementiert.

Inhalt

1. DB-Entwurf
2. Entwurfs-Phasen
3. Integritätsbedingungen
4. CHECK-Constraints in SQL
5. Schema-Qualität

Geeignete Namen

- Namen sollten selbstdokumentierend sein.

  Der Zusammenhang zur realen Welt muss klar sein. Wenn nötig, fügt man Kommentare, Erklärungen oder Beispieldaten hinzu.

- Namen sollten nicht zu lang sein.

  Namen mit mehr als 15–20 Buchstaben werden unhandlich.

- Die Wahl guter Namen verlangt einige Überlegung.
  Aber die investierte Zeit wird sich später auszahlen.

  Es kann helfen, die Namen mit anderen Leuten zu diskutieren.

- Man sollte immer versuchen, konsistent zu bleiben!

  Abkürzungen konsistent verwenden. „_“ konsistent verwenden.
  In einem Team sollten alle den gleichen Stil verwenden.

Redundante Information

- Redundante Informationen in einem DB-Schema sind schlecht: Sie verkomplizieren das Schema und die Anwendungs-Programmierung, sowie ggf. später notwendige Änderungen.

  Man braucht mindestens eine Integritätsbedingung, die sicherstellt, dass beide Kopien der Information konsistent bleiben.

- Redundante Information kann im physischen Entwurf aus Performancegründen eingeführt werden.

- Außerdem kann man später Sichten definieren, die abgeleitete Informationen enthalten.