Einführung in Datenbanken

Kapitel 13: Einführung in den Datenbank-Entwurf

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2019/20

http://www.informatik.uni-halle.de/~brass/db19/

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.

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

Inhalt

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, die auf den vorhandenen Daten beruhen, entwickelt.
  - 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.
  - Größe: DB-Schemata können sehr groß sein.
- Wie jede schwierige Aufgabe wird der DB-Entwurf in mehreren Schritten durchgeführt.

Datenbank-Entwurf (5)

- Es gibt drei Phasen des DB-Entwurfs:
  - Konzeptioneller DB-Entwurf erstellt ein Modell der Miniwelt in einem konzeptionellen Datenmodell (z.B. dem Entity-Relationship Modell).

    Statt "konzeptionell" kann man auch "konzeptuell" sagen.
  - Logischer DB-Entwurf transformiert das Schema in das Datenmodell des DBMS.

    Dies ist heute fast immer das relationale Modell.
  - Physischer DB-Entwurf hat Verbesserung der Performance des endgültigen Systems zum Ziel.

    Indexe und Speicherparameter werden in dieser Phase gewählt.

Datenbank-Entwurf (6)

Warum verschiedene Entwicklungsphasen?

- Probleme können getrennt und nacheinander abgearbeitet werden.
- Z.B. muss man während der konzeptionellen Entwicklung nicht über die Performance oder über Einschränkungen verschiedener DBMS nachdenken.

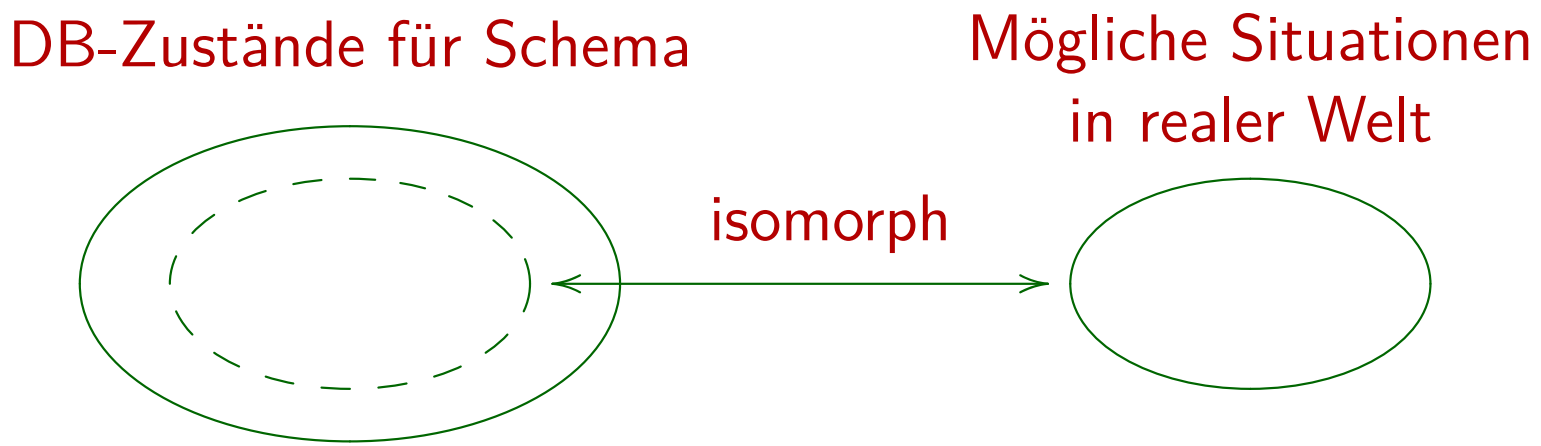
  Im Mittelpunkt: Entwicklung eines korrekten Modells der realen Welt.
- DBMS-Features beeinflussen den konzeptionellen Entwurf nicht (und nur teilweise den logischen).

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

Inhalt

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 (Entities, Attribute und Relationships) erlaubt oft zu viele (bedeutungslose, illegale) DB-Zustände:



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	2006	Köln	···

- Kundennummern müssen eindeutig sein.
- Das Geburtsjahr muss größer als 1880 sein.
- Kunden müssen mindestens 18 Jahre alt sein.

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 (man sichere Annahmen über Integritätsbedingungen).

  Z.B. der Programmierer nimmt an, dass keine Nullwerte auftreten (siehe Kapitel 9). 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).

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”.
- Triviale/implizierte Constraints nicht angeben!

  Erhöht Komplikationen, Menge der gültigen Zustände wird nicht geändert.

Zusammenfassung (1)

Warum Integritätsbedingungen (IBen) festlegen?

- Etwas Schutz vor Eingabefehlern.
- IBen 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 ist eine allgemeine Erfahrung, dass es Ausnahmesituationen gibt, in denen das DBS wegen der festgelegten IBen als zu unflexibel erscheint.

- 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 Verteilung der Daten ist auch eine nützliche Information für physischen Entwurf.

Inhalt

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.