Teil 2: Das relationale Modell

Literatur:

- Elmasri/Navathe: Fundamentals of Database Systems, 3. Auflage, 1999.
  7.1 Relational Model Concepts
  7.2 Relational Constraints and Relational Database Schemas
  7.3 Update Operations and Dealing with Constraint Violations
- Kemper/Eickler: Datenbanksysteme, 4. Auflage, 2001.
  Abschnitt 3.1, "Definition des relationalen Modells"
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3. Auflage, 1999.
  Kapitel 3: Relational Model. Abschnitt 6.2: "Referential Integrity".
- Heuer/Saake: Datenbanken, Konzepte und Sprachen, Thomson, 1995.
- Date/Darwen: A Guide to the SQL Standard, 4. Auflage, Addison-Wesley, 1997.
- Lipeck: Skript zur Vorlesung Datenbanksysteme, Univ. Hannover, 1996.
- Codd: A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377–387, 1970. Reprinted in CACM 26(1), 64–69, 1983.
  Siehe auch: `http://www1.acm.org:81/classics/nov95/toc.html` (unvollständig)

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- Grundkonzepte des relationalen Modells erklären.
  Was ist ein Schema? Was ist ein Zustand für ein gegebenes Schema?
- Domains erklären und warum sie nützlich sind.
- Anwendungen/Probleme von Nullwerten erklären.
- die Bedeutung von Schlüsseln und Fremdschlüsseln erklären.
- verschiedene Notationen für relationale Schemata verstehen.

Inhalt

Relationenmodell: Bedeutung

- Relationale DBMS beherrschen derzeit den Markt. Die meisten neuen DB-Projekte nutzen sie.
  Alte Anwendungen laufen eventuell noch auf einem Netzwerk-DBMS oder hierarchischen DBMS (z.B. IMS von IBM).
- Objektorientierte DBMS: nur selten eingesetzt.
  Nur bei "Nicht-Standard-Anwendungen", z.B. CAD-Daten. Die meisten OODBMS haben nur eine Programmierschnittstelle, und keine mächtige Anfragesprache wie SQL (es gibt aber OQL-Vorschlag).
- Der Trend geht zu objektrelationalen DBMS.
  Alle großen Anbieter relationaler DBMS haben auch OR-Features. Es ist unklar, inwieweit diese Erweiterungen in der Praxis genutzt werden.
- XML-DBMS werden derzeit entwickelt.

Beispiel-Datenbank (1)

STUDENTEN			
SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Grau	NULL
103	Daniel	Sommer	···
104	Iris	Winter	···

BEWERTUNGEN			
SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

AUFGABEN			
ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

Beispiel-Datenbank (2)

- `STUDENTEN`: enthält eine Zeile für jeden Studenten.
  - ◇ `SID`: “Studenten-ID” (eindeutige Nummer).
  - ◇ `VORNAME`, `NACHNAME`: Vor- und Nachname.
  - ◇ `EMAIL`: Email-Adresse (kann NULL sein).
- `AUFGABEN`: enthält eine Zeile für jede Aufgabe.
  - ◇ `ATYP`: Typ/Kategorie der Aufgabe.

    Z.B. `'H'`: Hausaufgabe, `'Z'`: Zwischenklausur, `'E'`: Endklausur.
  - ◇ `ANR`: Aufgabennummer (innerhalb des Typs).
  - ◇ `THEMA`: Thema der Aufgabe.
  - ◇ `MAXPT`: Maximale/volle Punktzahl der Aufgabe.

Beispiel-Datenbank (3)

- `BEWERTUNGEN`: enthält eine Zeile für jede abgegebene Lösung zu einer Aufgabe.
  - ◇ `SID`: Student, der die Lösung abgegeben hat.

    Dies referenziert eine Zeile in `STUDENTEN`.
  - ◇ `ATYP`, `ANR`: Identifikation der Aufgabe.

    Zusammen identifiziert dies eine Zeile in `AUFGABEN`.
  - ◇ `PUNKTE`: Punkte, die der Student für die Lösung bekommen hat.
  - ◇ Falls es keinen Eintrag für einen Studenten und eine Aufgabe gibt: Aufgabe nicht abgegeben.

Datenwerte (1)

- Tabelleneinträge sind Datenwerte, die einer gegebenen Auswahl von Datentypen entnommen sind.
- Die möglichen Datentypen sind durch das RDBMS (oder den SQL-Standard) vorgegeben.

  Die DBMS unterscheiden sich in den unterstützten Datentypen.
- Z.B. Strings, Zahlen (verschiedener Maximallänge, Nachkommastellen), Datum/Zeit, binäre Daten.
- Das relationale Modell (RM) ist von der speziellen Auswahl an Datentypen unabhängig.

  Die Def. des RM erhält eine Menge von Datentypen als Parameter.

Datenwerte (2)

- Erweiterbare DBMS erlauben die Definition neuer Datentypen (z.B. geometrische Datentypen).

  Es gibt grundsätzlich zwei Arten, dies zu ermöglichen: (1) Man kann neue Prozeduren (z.B. in C geschrieben) ins DBMS einbinden. (2) Das DBMS hat eine eingebaute Programmiersprache (für serverseitig gespeicherte Prozeduren). Darin definierte Datentypen und Operationen können in Tabellendeklarationen und Anfragen verwendet werden. Echte Erweiterbarkeit sollte es auch ermöglichen, neue Indexstrukturen zu definieren und den Anfrageoptimierer zu erweitern.

- Diese Erweiterbarkeit ist eine wichtige Eigenschaft moderner objektrelationaler Systeme.

  "Universelle/r DB/Server": kann mehr als Zahlen und Zeichenketten speichern ("alle Arten elektronischer Information").

Datenwerte (3)

- In der Logik (Kap. 3) werden die gegebenen Datentypen in Form einer Signatur $\Sigma_\mathcal{D} = (\mathcal{S}_\mathcal{D}, \mathcal{P}_\mathcal{D}, \mathcal{F}_\mathcal{D})$ und einer $\Sigma_\mathcal{D}$-Interpretation $\mathcal{I}_\mathcal{D}$ spezifiziert.
- In den folgenden Definitionen brauchen wir nur
  - ◇ eine gegebene Menge $\mathcal{S}_\mathcal{D}$ von Datentyp-Namen

    Man sagt oft einfach "Datentyp" statt "Datentyp-Name".
  - ◇ und für jedes $D \in \mathcal{S}_\mathcal{D}$ eine Menge $val(D)$ möglicher Werte dieses Typs ($val(D) := \mathcal{I}_\mathcal{D}[D]$).
- Z.B. hat Datentyp "`NUMERIC(2)`" die Werte `-99..+99`.

Atomare Attributwerte (1)

- Das relationale Modell behandelt einzelne Tabelleneinträge als atomar.

  Statt Spalten sagt man auch "Attribute". Entsprechend sind Tabelleneinträge dann "Attributwerte".

- D.h. das klassische relationale Modell erlaubt nicht, strukturierte oder mehrwertige Spaltenwerte einzuführen.

  Jede Zelle kann nur einzelne Zahlen, Zeichenketten, etc. enthalten.

- Das NF^2-Datenmodell ("Non First Normal Form") erlaubt dagegen ganze Tabellen als Tabelleneintrag.

Atomare Attributwerte (2)

- Bsp. einer NF^2-Tabelle (im klassischen relationalen Modell nicht enthalten, hier nicht behandelt):

HAUSAUFGABEN

NR	THEMA	MAXPUNKTE	GELOEST_VON: STUDENT	GELOEST_VON: PUNKTE
1	ER	10	Lisa Weiss Michael Grau Daniel Sommer	10 9 5
2	SQL	10	Lisa Weiss Michael Grau	8 9

Atomare Attributwerte (3)

- Unterstützung von “komplexen Werten” (Mengen, Records, verschachtelte Tabellen) ist ein weiteres typisches Feature von objektrelationalen Systemen.

  Oracle8 (mit “Objekt”-Option) erlaubt jeden PL/SQL-Typ für Spalten, einschließlich verschachtelte Tabellen. PL/SQL ist Oracles Sprache für gespeicherte Prozeduren. Seit Oracle 8i wird Java als Alternative unterstützt.

- Manche Systeme erlauben beliebige Schachtelung der Konstrukte “Menge”, “Liste”, “Feld”, “Multimenge” (Menge mit Duplikaten) und “Record”.

  Eine Tabelle ist dann einfach der Spezialfall “Menge (oder Multimenge) von Records”.

Atomare Attributwerte (4)

- Natürlich können auch in klassischen Systemen, wenn z.B. `DATE` (Datum) ein gegebener Datentyp ist, die Datentyp-Operationen verwendet werden, um Tag, Monat oder Jahr zu extrahieren.

  Dann sind auch Zeichenketten (Strings) nicht richtig atomar, sondern eine Folge von Zeichen.

- Das geschieht jedoch auf der Ebene der gegebenen Datentypen, nicht auf der Ebene des Datenmodells.

  Z.B. kann man keine neuen strukturierten Datentypen einführen und wenn man Strings mit einer wichtigen inneren Struktur verwendet, wird man bald merken, dass es sinnvolle Anfragen gibt, die in SQL nicht mit Datentyp-Funktionen ausgedrückt werden können.

Relationale DB-Schemata (1)

- Ein relationales Schema ρ (Schema einer einzigen Relation) definiert
  - ◇ eine endliche Folge $A_1 \ldots A_n$ von Attributnamen (Spaltennamen) und

    Formal eine Abbildung von $\{1, \ldots, n\}$ auf Bezeichner (Namen). Die Namen müssen sich unterscheiden, d.h. $A_i \neq A_j$ für $i \neq j$.

  - ◇ für jedes Attribut A_i einen Datentyp D_i.

    Sei $dom(A_i) := val(D_i)$ (Menge möglicher Wert von A_i, Domain).

- Ein Schema einer Relation kann dann geschrieben werden als

$$\rho = (A_1 \colon D_1, \; \ldots, \; A_n \colon D_n).$$

Relationale DB-Schemata (2)

- Ein relationales DB-Schema $\mathcal{R}$ definiert
  - ◇ eine endliche Menge von Relationen-Namen $\{R_1, \ldots, R_m\}$,
  - ◇ für jede Relation R_i ein Schema $sch(R_i)$ und
  - ◇ eine Menge $\mathcal{C}$ von Integritätsbedingungen (später definiert).

    Z.B. Schlüssel und Fremdschlüssel.

- D.h. $\mathcal{R} = (\{R_1, \ldots, R_m\}, \; sch, \; \mathcal{C})$.

  Dies ist nur eine mögliche mathematische Formalisierung. In der Praxis schreibt man DB-Schema anders (übersichtlicher) auf. Es gibt viele verschiedene Notationen für solche Schemata, siehe unten.

Relationale DB-Schemata (3)

Konsequenzen der Definition:

- Spaltennamen in einer Tabelle eindeutig: keine Tabelle darf zwei gleichnamige Spalten haben.
- Verschiedene Tabellen können jedoch gleichnamige Spalten haben (z.B. `ANR` im Beispiel).

  Die Spalten können verschiedene Datentypen haben (schlechter Stil).
- Für jede Spalte (identifiziert durch die Kombination von Tabellen- und Spaltennamen) gibt es einen eindeutigen Datentyp.

  Natürlich können verschiedene Spalten den gleichen Datentyp haben.

Relationale DB-Schemata (4)

- Die Spalten einer Tabelle sind sortiert, d.h. es gibt eine erste, zweite, usw. Spalte.

  Das ist normalerweise nicht sehr wichtig, aber z.B. `SELECT * FROM` R gibt die Tabelle mit den Spalten in der gegebenen Reihenfolge aus.
- In einem DB-Schema müssen Tabellennamen eindeutig sein: keine gleichnamigen Tabellen.
- Ein DBMS-Server kann normalerweise mehrere DB-Schemata verwalten.

  Dann können verschiedene Schemata gleichnamige Tabellen haben. Z.B. sind in einem Oracle-System Tabellen eindeutig durch die Kombination von Schema(Nutzer)-Name und Tabellenname identifiziert.

Schemata: Notation (1)

- Betrachten Sie die Beispiel-Tabelle:

AUFGABEN			
ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

- Eine Art, ein Schema präzise zu definieren, ist über ein SQL-Statement (siehe Kapitel 10):

```
CREATE TABLE AUFGABEN(ATYP  CHAR(1),
                      ANR   NUMERIC(2),
                      THEMA VARCHAR(40),
                      MAXPT NUMERIC(2))
```


Schemata: Notation (2)

- Obwohl letztendlich ein `CREATE TABLE`-Statement für das DBMS benötigt wird, gibt es andere Notationen, um das Schema zu dokumentieren.
- Bei der Diskussion der DB-Struktur sind die Datentypen der Spalten oft nicht wichtig.
- Eine kurze Notation ist der Tabellenname, gefolgt von der Liste der Spaltennamen in Klammern:

  `AUFGABEN(ATYP, ANR, THEMA, MAXPT)`
- Wenn nötig, werden die Datentypen hinzugefügt:

  `AUFGABEN(ATYP: CHAR(1), ...)`

Schemata: Notation (3)

- Man kann auch den Kopf der Tabelle verwenden:

AUFGABEN			
ATYP	ANR	THEMA	MAXPT
⋮	⋮	⋮	⋮

- Oder eine Tabelle mit Spaltendefinitionen:

AUFGABEN	
Spalte	Typ
ATYP	CHAR(1)
ANR	NUMERIC(2)
THEMA	VARCHAR(40)
MAXPT	NUMERIC(2)

Tupel (1)

- Ein n-Tupel ist eine Folge von n Werten.

  Man kann auch nur "Tupel" statt n-Tupel sagen, wenn das n nicht wichtig ist oder vom Kontext her klar ist. Tupel werden verwendet, um Tabellenzeilen zu formalisieren, dann ist n die Anzahl der Spalten.

- Z.B. sind XY-Koordinaten Paare (X, Y) von reellen Zahlen. Paare sind Tupel der Länge 2 ("2-Tupel").

  3-Tupel werden auch Tripel genannt und 4-Tupel Quadrupel.

- Das kartesische Produkt $\times$ erstellt Mengen von Tupeln, z.B.:

$$\mathbb{R} \times \mathbb{R} := \{(X, Y) \mid X \in \mathbb{R},\ Y \in \mathbb{R}\}.$$

Tupel (2)

- Ein Tupel t für das Relationen-Schema

$$\rho = (A_1\colon D_1, \ldots, A_n\colon D_n)$$

  ist eine Folge $(d_1, \ldots, d_n)$ von n Werten, so dass $d_i \in val(D_i)$. D.h. $t \in val(D_1) \times \cdots \times val(D_n)$.

- Gegeben sei ein solches Tupel. Wir schreiben $t.A_i$ für den Wert d_i in der Spalte A_i.

  Alternative Notation: $t[A_i]$.

- Z.B. ist eine Zeile in der Beispieltabelle "AUFGABEN" das Tupel ('H', 1, 'ER', 10).

DB-Zustände (1)

Sei ein DB-Schema $(\{R_1, \ldots, R_m\}, sch, \mathcal{C})$ gegeben.

- Ein DB-Zustand $\mathcal{I}$ für dieses Schema definiert für jede Relation R_i eine endliche Menge von Tupeln für das Relationen-Schema $sch(R_i)$.
- D.h. wenn $sch(R_i) = (A_{i,1}\colon D_{i,1}, \ \ldots, \ A_{i,n_i}\colon D_{i,n_i})$, dann

$$\mathcal{I}[R_i] \subseteq val(D_{i,1}) \times \cdots \times val(D_{i,n_i}).$$

- D.h. ein DB-Zustand interpretiert die Symbole im DB-Schema.

  Er bildet Relationen-Namen auf Relation ab.

DB-Zustände (2)

- In der Mathematik wird der Begriff "Relation" als "Teilmenge eines kartesischen Produkts" definiert.
- Z.B. ist eine Ordnungsrelation wie "$<$" auf den natürlichen Zahlen formal $\{(X, Y) \in \mathbb{N} \times \mathbb{N} \mid X < Y\}$.
- Übung: Was sind Unterschiede zwischen Relationen in Datenbanken und Relationen wie "$<$"?
    1. ______________________________
    2. ______________________________
    3. ______________________________

DB-Zustände (3)

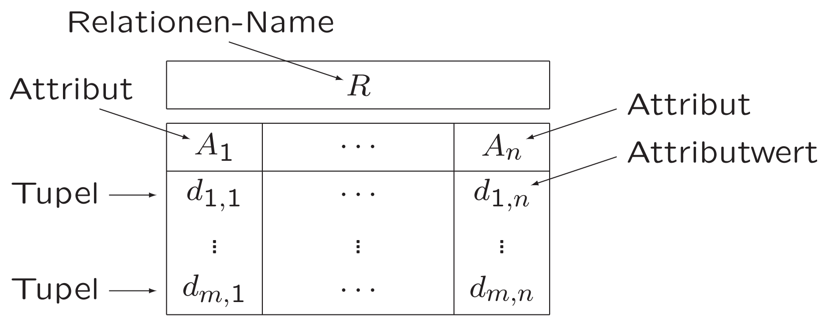
Relationen sind Mengen von Tupeln. Daher

- ist die Reihenfolge der Tupel nicht definiert.
    - ◇ Die Darstellung in einer Tabelle ist etwas irreführend. Es gibt keine erste, zweite, usw. Zeile.

      Die Speicherplatzverwaltung definiert, wo eine neue Zeile eingefügt wird (verwendet z.B. den Platz von gelöschten Zeilen).
    - ◇ Relationen können bei Ausgabe sortiert werden.
- gibt es keine Tupel-Duplikate.
    - ◇ Viele derzeitige Systeme erlauben doppelte Tupel, solange kein Schlüssel definiert ist (später).

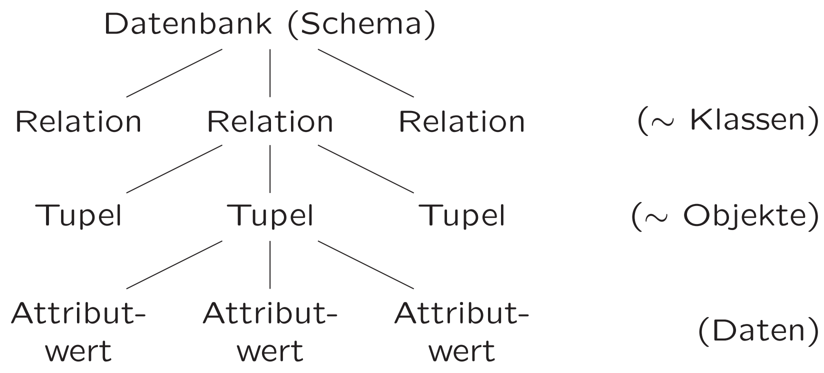
      Also wäre eine Formalisierung als Multimengen korrekt.

Zusammenfassung (1)



Synonyme: Relation und Tabelle.
Tupel, Zeile und Record.
Attribut, Spalte, Feld.
Attributwert, Spaltenwert, Tabelleneintrag.

Zusammenfassung (2)



Speicherstrukturen

- Offensichtlich kann eine Relation als Datei von Records gespeichert werden. Aber auch andere Datenstrukturen können ein relationales Interface bieten.

  Das relationale Modell erfordert keine spezifische Speicherstruktur. Tabellen sind nur die logische Sicht. Andere Speicherstrukturen könnten erlauben, gewisse Anfragen effizienter zu bearbeiten. Z.B. sind die `V$*`-Tabellen in Oracle ein Interface zu Datenstrukturen im Server.

- Übung: Definieren Sie ein relationales Interface zu

  ```
  Monatsnamen: array[1..12] of string;
  ```

  Was sind Unterschiede zwischen diesem Array und der Standard- "Datei von Records" für die Relation?

Update-Operationen (1)

- Updates ändern einen DB-Zustand $\mathcal{I}_{\text{alt}}$ in einen DB-Zustand $\mathcal{I}_{\text{neu}}$. Die grundlegenden Update-Operationen des relationalen Modells sind:
  - ◇ Einfügen (eines Tupels in eine Relation):

    $$\mathcal{I}_{\text{neu}}[R] := \mathcal{I}_{\text{alt}}[R] \cup \{(d_1, \ldots, d_n)\}$$

  - ◇ Löschen (eines Tupels aus einer Relation):

    $$\mathcal{I}_{\text{neu}}[R] := \mathcal{I}_{\text{alt}}[R] - \{(d_1, \ldots, d_n)\}$$

  - ◇ Änderung / Update (eines Tupels):

    $$\mathcal{I}_{\text{neu}}[R] := (\mathcal{I}_{\text{alt}}[R] - \{(d_1, \ldots, d_i, \ldots, d_n)\}) \cup \{(d_1, \ldots, d_i', \ldots, d_n)\}$$

Update-Operationen (2)

- Die Änderung entspricht einem Löschen gefolgt von einer Einfügung, aber ohne die Existenz des Tupels zu unterbrechen.

  Es könnte von Integritätsbedingungen verlangt werden, dass ein Tupel mit bestimmten Werten für die Schlüsselattribute existiert.

- SQL hat Befehle für das Einfügen, Löschen und die Änderung einer ganzen Menge von Tupeln (der gleichen Relation).
- Updates können auch zu Transaktionen kombiniert werden (atomar, d.h. "ganz oder gar nicht").

Inhalt

Nullwerte (1)

- Das relationale Modell erlaubt fehlende Attributwerte, d.h. Tabelleneinträge können leer sein.
- Formal wird die Menge der möglichen Attributwerte durch einen neuen Wert "Null" erweitert.
- Wenn R das Schema $(A_1: D_1, \ldots, A_n: D_n)$ hat, dann

  $$\mathcal{I}[R] \subseteq (val(D_1) \cup \{null\}) \times \cdots \times (val(D_n) \cup \{null\}).$$

- "Null" ist nicht die Zahl 0 oder der leere String! Es ist von allen Werten des Datentyps verschieden.

Nullwerte (2)

- Nullwerte werden in vielen verschiedenen Situationen verwendet, z.B.:
  - ◇ Ein Wert existiert, ist aber unbekannt.

    Angenommen, man will in der Tabelle `STUDENTEN` auch die Telefonnummer der Studenten speichern, aber man kennt möglicherweise nicht von jedem Studenten die Telefonnummer, obwohl wahrscheinlich alle irgendwie telefonisch zu erreichen sind.
  - ◇ Es existiert kein Wert.

    In einer Tabelle mit Vorlesungsdaten könnte es eine Spalte `URL` geben, aber nicht jede Vorlesung hat eine Web-Seite (wenn aber eine Webseite existiert, wäre sie normalerweise auch eingetragen).
  - ◇ Es könnte ein (unbekannter) Wert existieren, oder auch keiner.

Nullwerte (3)

- Anwendungen von Nullwerten, fortgesetzt:
  - ◇ Attribut ist auf dieses Tupel nicht anwendbar.

    Z.B. müssen an einer Universität in den USA nur ausländische Studenten einen Toefl-Test ablegen, um ihre Englischkenntnisse zu beweisen. Eine Spalte für die Toefl-Punktzahl in der Tabelle `STUDENTEN` ist für U.S.-Studenten nicht anwendbar. Selbst wenn diese Studenten früher einmal einen Toefl-Test gemacht haben (z.B. weil sie Immigranten sind), ist die Universität an dem Resultat nicht interessiert.
  - ◇ Wert wird später zugewiesen/bekannt gegeben.
  - ◇ Jeder Wert ist möglich.
- Ein Ausschuss fand 13 verschiedene Bedeutungen von Nullwerten.

Nullwerte (4)

Vorteile von Nullwerten:

- Ohne Nullwerte wäre es nötig, die meisten Relationen in viele aufzuspalten ("Subklassen"):
  - ◇ Z.B. `STUDENTEN_MIT_EMAIL`, `STUDENTEN_OHNE_EMAIL`.
  - ◇ Oder extra Relation: `STUD_EMAIL(SID, EMAIL)`.
  - ◇ Das erschwert Anfragen.

    Man braucht Verbunde und Vereinigungen (siehe Kapitel 4).
- Sind Nullwerte nicht erlaubt, werden sich die Nutzer Werte ausdenken, um die Spalten zu füllen.

  Das macht die DB-Struktur sogar noch unklarer.

Nullwerte (5)

Probleme:

- Da der gleiche Nullwert für verschiedene Zwecke genutzt wird, kann es keine klare Semantik geben.
- SQL benutzt dreiwertige Logik, um Bedingungen mit Nullwerten auszuwerten.

  Da man an die normale zweiwertige Logik gewöhnt ist, kann es Überraschungen geben — einige Äquivalenzen gelten nicht.
- Fast alle Programmiersprachen haben keine Nullwerte. Das erschwert Anwendungsprogramme.

  Wenn also ein Attributwert in eine Programmvariable eingelesen wird, muss er auf Nullwerte überprüft werden (→ Indikatorvariablen).

Nullwerte ausschließen (1)

- Da Nullwerte zu Komplikationen führen, kann für jedes Attribut festgelegt werden, ob Nullwerte erlaubt sind oder nicht.
- Es ist wichtig, genau darüber nachzudenken, wo Nullwerte gebraucht werden.
- Viele Spalten als "not null" zu deklarieren, vereinfacht Programme und verringert Überraschungen.
- Die Flexibilität geht jedoch verloren: Nutzer werden gezwungen, für alle "not null"-Attribute Werte einzutragen.

Nullwerte ausschließen (2)

- In SQL schreibt man `NOT NULL` hinter den Datentyp für ein Attribut, das nicht Null sein kann.

  Dies ist genau genommen eine Integritätsbedingung, aber man kann es auch als Teil des Datentyps ansehen. Die genaue Syntax der "`CREATE TABLE`" Anweisung wird in Kapitel 10 erklärt.
- Z.B. kann `EMAIL` in `STUDENTEN` Null sein:

```
CREATE TABLE STUDENTEN(
                SID         NUMERIC(3)  NOT NULL,
                VORNAME     VARCHAR(20) NOT NULL,
                NACHNAME    VARCHAR(20) NOT NULL,
                EMAIL       VARCHAR(80)         )
```


Nullwerte ausschließen (3)

- In SQL sind Nullwerte als Default erlaubt und man muss explizit "`NOT NULL`" verlangen.
- Oft können nur wenige Spalten Nullwerte haben.
- Daher ist es besser, in der vereinfachten Notation umgekehrt optionale Attribute zu markieren:

  STUDENTEN(SID, VORNAME, NACHNAME, $\mathtt{EMAIL}^{o}$)
- In dieser Notation werden Attribute, die Nullwerte enthalten können, mit einem kleinen "o" (optional) im Exponenten markiert.

  Dies ist nicht Teil des Spaltennamens. Alternative: "`EMAIL?`".

Nullwerte ausschließen (4)

- In der Tabellen-Notation kann die Möglichkeit von Nullwerten folgendermaßen dargestellt werden:

STUDENTEN		
Spalte	Typ	Null
SID	NUMERIC(3)	N
VORNAME	VARCHAR(20)	N
NACHNAME	VARCHAR(20)	N
EMAIL	VARCHAR(80)	J

STUDENTEN	SID	···	EMAIL
Typ	NUMERIC(3)	···	VARCHAR(80)
Null	N	···	J

Inhalt

Integritätsbedingungen (1)

- Integritätsbedingungen (IBen) sind Bedingungen, die jeder DB-Zustand erfüllen muss, siehe Kapitel 8.
- Damit wird die Menge der DB-Zustände weiter eingeschränkt.

  Zunächst konstruiert man mit den Tabellenstrukturen (Tabellennamen, Spalten, Datentypen) eine Menge von möglichen DB-Zuständen, dann wählt man über Integritätsbedingungen davon eine Teilmenge als die zulässigen DB-Zustände, die dann wirklich zur Repräsentation von Informationen verwendet werden soll.
- Das DBMS weist jede Zustandsänderung zurück, die zur Verletzung einer IB führen würde.

  Der initiale (leere) Zustand muss natürlich alle IBen erfüllen.

Integritätsbedingungen (2)

- Z.B. können im `CREATE TABLE` - Statement in SQL folgende Arten von IBen festgelegt werden:
  - ◇ `NOT NULL`: Verbot von Nullwerten in einer Spalte.
  - ◇ Schlüssel: Eindeutige Identifizierung.
  - ◇ Fremdschlüssel: Jeder Wert in einer Spalte muß als Schlüsselwert in anderer Tabelle auftauchen.
  - ◇ `CHECK`: Bedingung für Spaltenwerte.

    Die Bedingung kann sich auch auf mehrere Spalten derselben Zeile beziehen.
- In der Praxis braucht man zur Konstruktion der Zustandsmenge aber manchmal weitere IBen.

Integritätsbedingungen (3)

- Der SQL-92-Standard enthält eine Anweisung `CREATE ASSERTION`, die aber in den heutigen DBMS nicht implementiert ist.
- Man kann Integritätsbedingungen auch durch SQL-Anfragen formalisieren, die Verletzungen der Bedingungen ausgeben (oder als logische Formeln).

  Oder man kann die Integritätsbedingungen in natürlicher Sprache angeben. Das DBMS versteht dies zwar nicht und kann daher die Bedingung nicht erzwingen. Aber es ist dennoch eine nützliche Dokumentation für die Entwicklung von Anwendungsprogrammen (die Erfüllung der IBen muß in den Programmen zur Dateneingabe geprüft werden). Sind IBen als SQL-Anfragen formuliert (s.o.), so kann man sie von Zeit zu Zeit ausführen und ggf. Verletzungen finden.

Eindeutige Identifikation (1)

- Ein Schlüssel einer Relation R ist eine Spalte A, die die Tupel/Zeilen in R eindeutig identifiziert.

  Die Schlüsselbedingung ist in einem DB-Zustand $\mathcal{I}$ genau dann erfüllt, wenn für alle Tupel $t, u \in \mathcal{I}[R]$ gilt: wenn $t.A = u.A$, dann $t = u$.

- Wenn z.B. `SID` als Schlüssel von `STUDENTEN` deklariert wurde, ist dieser DB-Zustand verboten:

STUDENTEN			
SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
101	Michael	Grau	···
103	Daniel	Sommer	NULL
104	Iris	Winter	···

Eindeutige Identifikation (2)

- Wurde `SID` als Schlüssel von `STUDENTEN` deklariert, lehnt das DBMS ab, eine Zeile mit dem gleichen Wert für `SID` wie eine existierende Zeile einzufügen.
- Schlüssel sind Integritätsbedingungen: Sie müssen für alle DB-Zustände gelten, nicht nur für den aktuellen Zustand.
- Obwohl im obigen DB-Zustand (mit nur 4 Studenten) der Nachname (`NACHNAME`) eindeutig ist, würde dies allgemein zu einschränkend sein.

  Z.B. wäre das zukünftige Einfügen von "`Nina Weiss`" unmöglich.

Eindeutige Identifikation (3)

- Ein Schlüssel kann auch aus mehreren Attributen bestehen ("zusammengesetzter Schlüssel").

  Wenn A und B zusammen einen Schlüssel bilden, ist es verboten, dass es zwei Zeilen t und u gibt, die in beiden Attributen übereinstimmen (d.h. $t.A = u.A$ und $t.B = u.B$). Zwei Zeilen können in einem Attribut übereinstimmen, aber nicht in beiden.

- Der Schlüssel "`VORNAME, NACHNAME`" ist hier erfüllt:

STUDENTEN			
SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Weiss	···
103	Michael	Grau	···

Schlüssel: Minimalität (1)

- Jeder DB-Zustand $\mathcal{I}$,
  - der den Schlüssel "`NACHNAME`" erfüllt,
  - erfüllt auch den Schlüssel "`VORNAME, NACHNAME`".
- Ist `NACHNAME` allein schon eindeutig, so ist erst recht die Kombination mit `VORNAME` zusammen eindeutig.

  Gibt es keine zwei Zeilen, die im `NACHNAME` übereinstimmen, so gibt es auch nicht zwei Zeilen, die in `NACHNAME` und `VORNAME` übereinstimmen.
- Allgemein macht das Hinzufügen von Attributen Schlüssel schwächer (weniger einschränkend).

  Die Bedingung wird dann von mehr Zuständen erfüllt.

Schlüssel: Minimalität (2)

- Sei F_1 eine Formel, die "`NACHNAME`" als Schlüssel festlegt, und F_2 eine Formel, die dem zusammengesetzten Schlüssel "`VORNAME, NACHNAME`" entspricht.
  - Dann ist F_2 logische Folgerung aus F_1.

    Jeder DB-Zustand $\mathcal{I}$, der F_1 erfüllt, erfüllt auch F_2.
- Wenn also `NACHNAME` als Schlüssel deklariert wurde, ist es nicht mehr interessant, dass auch "`VORNAME, NACHNAME`" eindeutig identifizierend ist.
- Man wird nie zwei Schlüssel deklarieren, so dass einer eine Teilmenge des anderen ist.

Schlüssel: Minimalität (3)

- Nur minimale Schlüssel (in Bezug auf "$\subseteq$") sind interessant.

  Viele Autoren beziehen die Minimalitätsbedingung in die Definition des Schlüsselbegriffs ein. Dann sind Schlüssel aber keine Integritätsbedingungen im Sinne von Beschränkungen der Zustandsmenge mehr.
- Der Schlüssel "`NACHNAME`" ist jedoch im Beispiel-Zustand auf Folie 2-48 nicht erfüllt.
- Möchte der DB-Entwerfer diesen Zustand zulassen, so ist die Schlüsselbedingung "`NACHNAME`" zu streng.

  Das ist eigentlich keine freie Entscheidung: Der DB-Entwerfer muss Situationen in der realen Welt betrachten, um dies zu entscheiden.

Schlüssel: Minimalität (4)

- Weil der Schlüssel "`NACHNAME`" ausgeschlossen ist, kommt der zusammengesetzte Schlüssel "`VORNAME, NACHNAME`" wieder in Frage.

  Man muß natürlich auch prüfen, ob der Schlüssel "`VORNAME`" möglich ist, aber dieser ist im Beispielzustand ebenfalls nicht erfüllt.
- Der DB-Entwerfer muss nun herausfinden, ob es jemals zwei Studenten in der Vorlesung geben kann, die den gleichen Vor- und Nachnamen haben.

  Im Beispiel-Zustand gibt es solche Studenten nicht, aber die Integritätsbedingung muss für alle Zustände gelten.

Schlüssel: Minimalität (5)

- Natürliche Schlüssel können fast immer Ausnahmen haben. Sind diese Ausnahmen sehr selten, könnte man solche Schlüssel dennoch in Erwägung ziehen:
  - ◇ Nachteil: Tritt eine Ausnahme auf, muss man den Namen von einem der beiden Studenten in der DB ändern und alle Dokumente, die von der DB gedruckt werden, muss man wieder ändern.

    Nachdem ich 7 Jahre gelehrt hatte, trat das auf (in einer Vorlesung mit über 150 Studenten).
  - ◇ Vorteil: Man kann Studenten in Programmen durch ihren Vor- und Nachnamen identifizieren.

Schlüssel: Minimalität (6)

- Wenn der Entwerfer entscheidet, dass der Nachteil des Schlüssels “VORNAME, NACHNAME” größer als der Vorteil ist, könnte er versuchen, weitere Attribute hinzuzufügen.
- Aber die Kombination “SID, VORNAME, NACHNAME” ist uninteressant, weil “SID” schon ein Schlüssel ist.
- Entscheidet der Entwerfer jedoch, dass “VORNAME, NACHNAME” “eindeutig genug” ist, wäre dies minimal, auch wenn “SID” schon ein Schlüssel ist.

  Minimalität bezieht sich hier auf “$\subseteq$”, nicht die Elementanzahl.

Mehrere Schlüssel

- Eine Relation kann mehr als einen Schlüssel haben.
- Z.B. ist SID ein Schlüssel von STUDENTEN und “VORNAME, NACHNAME” ggf. ein weiterer Schlüssel.
- Ein Schlüssel wird zum “Primärschlüssel” ernannt.

  Der Primärschlüssel sollte aus einem kurzen Attribut bestehen, das möglichst nie verändert wird (durch Updates). Der Primärschlüssel wird in anderen Tabellen verwendet, die sich auf Zeilen dieser Tabelle beziehen. In manchen Systemen ist Zugriff über Primärschlüssel besonders schnell. Ansonsten ist die Wahl des Primärschlüssels egal.
- Die anderen sind “Alternativ-/Sekundär-Schlüssel”.

  SQL verwendet den Begriff UNIQUE für alternative Schlüssel.

Schlüssel: Notation (1)

- Die Primärschlüssel-Attribute werden oft markiert, indem man sie unterstreicht:

$$R(\underline{A_1\colon D_1}, \ldots, \underline{A_k\colon D_k}, A_{k+1}\colon D_{k+1}, \ldots, A_n\colon D_n).$$

STUDENTEN			
SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Grau	NULL
103	Daniel	Sommer	···
104	Iris	Winter	···

- Normalerweise werden die Attribute so angeordnet, daß der Primärschlüssel am Anfang steht.

Schlüssel: Notation (2)

- In SQL können Schlüssel folgendermaßen definiert werden:

```
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))
```

- Die genaue Syntax wird in Kapitel 10 behandelt.

Schlüssel und Nullwerte

- Der Primärschlüssel darf nicht Null sein, andere Schlüssel sollten nicht Null sein.

  In SQL-89 und DB2 muss `NOT NULL` für jedes Attribut einer `PRIMARY KEY`- oder `UNIQUE`-Bedingung festgelegt werden. In SQL-92 und Oracle impliziert die "`PRIMARY KEY`"-Deklaration automatisch "`NOT NULL`", aber "`UNIQUE`" (für alternative Schlüssel) tut dies nicht. In Oracle kann es mehrere Zeilen mit einem Nullwert in einem `UNIQUE`-Attribut geben. In SQL Server darf nur eine Zeile Null sein.
  SQL-92 definiert drei verschiedene Semantiken für zusammengesetzte Schlüssel, die nur in manchen Attributen Nullwerte haben. Wenn man nicht genau weiß, was man tut, sollte man Nullwerte in Schlüsseln vermeiden.

- Es ist nicht akzeptabel, wenn schon die "Objektidentität" des Tupels unbekannt ist.

Schlüssel und Updates

- Es wird als schlechter Stil angesehen, wenn Schlüsselattribute geändert werden (mit Updates).

  Das würde die "Objektidentität" ändern. Besser: Tupel zuerst löschen und dann das Tupel mit neuen Werten einfügen.

- Aber SQL verbietet dies nicht.

  Der Standard enthält sogar Klauseln, die festlegen, was mit Fremdschlüsseln passieren soll, wenn sich der referenzierte Schlüsselwert ändert.

Der schwächste Schlüssel

- Ein Schlüssel bestehend aus allen Spalten der Tabelle fordert nur, dass es nie zwei verschiedene Zeilen gibt, die in allen Spaltenwerten übereinstimmen.

  Theoretisch sind Relationen Mengen: Dann wäre dieser Schlüssel keine Einschränkung. In der Praxis sind Relationen jedoch zunächst Multimengen. Dieser Schlüssel verbietet nun doppelte Zeilen (und bringt damit Theorie und Praxis wieder zusammen).

- Empfehlung: Um Duplikate auszuschließen, sollte man immer mindestens einen Schlüssel für jede Relation festlegen.

  Gibt es keinen anderen Schlüssel, sollte der Schlüssel gewählt werden, der aus allen Attributen der Relation besteht.

Schlüssel: Zusammenfassung

- Bestimmte Spalten als Schlüssel zu deklarieren ist etwas einschränkender als die eindeutige Identifikations-Eigenschaft:
  - ◇ Nullwerte sind zumindest im Primärschlüssel ausgeschlossen.
  - ◇ Man sollte Updates vermeiden, zumindest beim Primärschlüssel.
- Die Eindeutigkeit ist jedoch die Hauptaufgabe eines Schlüssels. Alles andere ist sekundär.

Übungen (1)

- Wählen Sie einen Schlüssel aus:

REZEPT		
NAME	ZUTAT	MENGE
Lebkuchen	Eier	2
Lebkuchen	Zucker	200g
Mürbeteig	Butter	250g
Mürbeteig	Zucker	100g

- Geben Sie ein Beispiel für eine Einfügung an, die den Schlüssel verletzten würde:

- Könnte "MENGE" auch als Schlüssel dienen?

Übungen (2)

- Betrachten Sie meinen Terminkalender:

TERMINE				
DATUM	START	ENDE	RAUM	AUFGABE
22.11.05	10:15	11:45	307	DB I halten
22.11.05	14:00	15:00	313	Prüfung Herr Meier
22.11.05	16:00	18:00	507	Forschungstreffen

- Was wären korrekte Schlüssel?
- Beispiel für einen nicht-minimalen Schlüssel?
- Werden weitere Integritätsbedingungen benötigt?

  Kann es ungültige Zustände geben, auch wenn Schlüsselbed. erfüllt?

Inhalt

Fremdschlüssel (1)

- Das relationale Modell hat keine expliziten Verweise (Zeiger) oder Beziehungen zwischen Tupeln.
- Schlüsselattributwerte identifizieren ein Tupel.

  Sie sind "logische Adressen" der Tupel.
- Um sich in einer Relation S auf Tupel von R zu beziehen, fügt man den Primärschlüssel von R zu den Attributen von S hinzu.

  Solche Attributwerte sind "logische Zeiger" auf Tupel in R.
- Z.B. hat die Tabelle `BEWERTUNGEN` das Attribut `SID`, welches Primärschlüsselwerte von `STUDENTEN` enthält.

Fremdschlüssel (2)

`SID` in `BEWERTUNGEN` ist ein Fremdschlüssel, der `STUDENTEN` referenziert:

STUDENTEN			
<u>SID</u>	VORNAME	NACHNAME	···
101	Lisa	Weiss	···
102	Michael	Grau	···
103	Daniel	Sommer	···
104	Iris	Winter	···

BEWERTUNGEN			
<u>SID</u>	<u>ATYP</u>	<u>ANR</u>	PUNKTE
101	H	1	10
101	H	2	8
102	H	1	9
102	H	2	9
103	H	1	5
105	H	1	7

?
Fehler

Die hier benötigte Bedingung ist, dass jeder `SID`-Wert in `BEWERTUNGEN` auch in `STUDENTEN` auftaucht.

Fremdschlüssel (3)

- Die Fremdschlüsselbedingung

  "`BEWERTUNGEN.SID` $\rightarrow$ `STUDENTEN`"

  fordert, daß die Menge der Werte in der Spalte `SID` der Tabelle `BEWERTUNGEN` immer eine Teilmenge der Primärschlüsselwerte in `STUDENTEN` ist.

  Somit ist die Menge der `SID`-Werte in `STUDENTEN` eine Art "dynamische Domain" für `SID` in `BEWERTUNGEN`.
- In relationaler Algebra (Kapitel 4) liefert die Projektion $\pi_{\texttt{SID}}$ die Werte der Spalte `SID`. Dann lautet die Fremdschlüsselbedingung:

  $$\pi_{\texttt{SID}}(\texttt{BEWERTUNGEN}) \subseteq \pi_{\texttt{SID}}(\texttt{STUDENTEN}).$$

Fremdschlüssel (4)

- Die Fremdschlüsselbedingung stellt sicher, dass es für jedes Tupel t in `BEWERTUNGEN` ein Tupel u in `STUDENTEN` gibt, so dass $t.\texttt{SID} = u.\texttt{SID}$.

  Paare solcher Tupel t und u kann man durch eine Operation der relationalen Algebra ("Join", Kapitel 4) zusammenbringen. Das entspricht der Dereferenzierung von Zeigern in anderen Modellen. Ohne Fremdschlüsselbedingung könnte es "Zeiger" geben, die ins Nichts zeigen. SQL-Anfragen stürzen in diesem Fall jedoch nicht ab: Tupel ohne "Join-Partner" werden in einer Anfrage mit einem Join eliminiert.
- Die Schlüsselbedingung in `STUDENTEN` stellt sicher, dass es maximal ein solches Tupel u gibt.

  Zusammen folgt, dass jedes Tupel t in `BEWERTUNGEN` genau ein Tupel u in `STUDENTEN` referenziert.

Fremdschlüssel (5)

- Die Erzwingung von Fremdschlüsselbedingungen sichert die “referentielle Integrität” der Datenbank.

  Statt “Fremdschlüsselbedingung” kann man auch “referentielle Integritätesbedingung” sagen.

- Fremdschlüssel entsprechen “eins-zu-viele”-Beziehungen: Z.B. ein Student hat viele Aufgaben gelöst.
- Die Tabelle BEWERTUNGEN, die den Fremdschlüssel enthält, wird “Kindtabelle” der referentiellen Integritätsbedingung genannt und die referenzierte Tabelle STUDENTEN ist die “Elterntabelle”.

Fremdschlüssel (6)

Aufgabe:

- Es ist noch nicht lange her, dass MySQL Fremdschlüssel nur “syntaktisch” unterstützte.

  Möglicherweise gilt das noch immer für manche Speichermanager.

- “CREATE TABLE” Anweisungen mit Fremdschlüssel-Deklarationen wurden akzeptiert, aber die Fremdschlüssel nicht wirklich überwacht.

  Sondern einfach ignoriert.

- Wie können Sie ausprobieren, ob Ihre Datenbank Fremdschlüssel überwacht?

Fremdschlüssel (7)

- Die Tabelle BEWERTUNGEN enthält noch einen Fremdschlüssel, der die gelöste Aufgabe referenziert.
- Aufgaben werden durch eine Kategorie und eine Nummer (ATYP und ANR) identifiziert:

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
⋮	⋮	⋮	⋮

AUFGABEN

<u>ATYP</u>	<u>ANR</u>	···	MAXPT
H	1	···	10
H	2	···	10
Z	1	···	14

Fremdschlüssel (8)

- Eine Tabelle mit zusammengesetztem Schlüssel (wie AUFGABEN) muss mit einem Fremdschlüssel referenziert werden, der die gleiche Spaltenanzahl hat.
- Die zugehörigen Spalten müssen den gleichen Datentyp haben.
- Es ist nicht nötig, dass die zugehörigen Spalten den gleichen Namen haben.
- Im Beispiel erfordert der Fremdschlüssel, dass jede Kombination von ATYP und ANR, die in BEWERTUNGEN vorkommt, auch in AUFGABEN existiert.

Fremdschlüssel (9)

- Spalten werden nach der Position in der Deklaration zugeordnet: Ist (VORNAME, NACHNAME) der Schlüssel und (NACHNAME, VORNAME) der Fremdschlüssel, werden Einfügungen meist Fehler geben.

  Sind die Datentypen von VORNAME und NACHNAME sehr verschieden, kann der Fehler schon bei der Deklaration des Fremdschlüssels erkannt werden. Aber manche Systeme erfordern nur "kompatible" Datentypen und das ist bereits mit VARCHAR-Typen verschiedener Länge erfüllt.

- Nur (ganze) Schlüssel können referenziert werden.

  Nicht beliebige Attribute, und auch nicht nur Teile eines zusammengesetzten Schlüssels. Normalerweise sollte man nur den Primärschlüssel referenzieren, aber SQL erlaubt auch alternative Schlüssel.

Fremdschlüssel: Notation (1)

- In der Attributlisten-Notation können Fremdschlüssel durch einen Pfeil und den Namen der referenzierten Tabelle markiert werden. Bei zusammengesetzten Fremdschlüsseln braucht man Klammern:

```
BEWERTUNGEN(SID → STUDENTEN,
            (ATYP, ANR) → AUFGABEN, PUNKTE)
STUDENTEN(SID, VORNAME, NACHNAME, EMAIL)
AUFGABEN(ATYP, ANR, THEMA, MAXPT)
```

- Da normalerweise nur Primärschlüssel referenziert werden, ist es nicht nötig, die zugehörigen Attribute der referenzierten Tabelle anzugeben.

Fremdschlüssel: Notation (2)

- Obiges Beispiel ist untypisch, weil alle Fremdschlüssel Teil eines Schlüssels sind. Das ist nicht nötig:

```
MODULKATALOG(MNR, TITEL, BESCHREIBUNG)
MODULANGEBOT(ANR, MNR → MODULKATALOG, SEM,
             (DOZ_VORNAME, DOZ_NACHNAME)→DOZENT)
DOZENT(VORNAME, NACHNAME, BUERO, TEL)
```

  In diesem Beispiel sind auch die Namen von Fremdschlüssel-Attributen und referenzierten Attributen verschieden. Das ist möglich.

- Manche markieren Fremdschlüssel durch gestricheltes Unterstreichen oder einen Strich oben. Dann ist aber die referenzierte Tabelle nicht klar.

Fremdschlüssel: Notation (3)

- In SQL können Fremdschlüssel wie folgt deklariert werden:

```
CREATE TABLE BEWERTUNGEN(
             SID          NUMERIC(3)   NOT NULL,
             ATYP         CHAR(1)      NOT NULL,
             ANR          NUMERIC(2)   NOT NULL,
             PUNKTE       NUMERIC(4,1) NOT NULL,
             PRIMARY KEY(SID, ATYP, ANR),
             FOREIGN KEY(SID)
                          REFERENCES STUDENTEN,
             FOREIGN KEY(ATYP,ANR)
                          REFERENCES AUFGABEN)
```

Fremdschlüssel: Notation (4)

- In der Tabellen-Notation können Fremdschlüssel z.B. folgendermaßen deklariert werden:

BEWERTUNGEN	SID	ATYP	ANR	PUNKTE
Typ	NUMERIC(3)	CHAR(1)	NUMERIC(2)	NUMERIC(2)
Null	N	N	N	N
Referenz	STUDENTEN	AUFGABEN	AUFGABEN	

- Zusammengesetzte Fremdschlüssel sind wieder ein Problem.

  Sollte die obige Notation unklar sein, gibt man die Namen der referenzierten Spalten mit an oder verteilt die Information über Fremdschlüssel auf mehrere Zeilen. In seltenen Fällen können sich Fremdschlüssel auch überlappen. Dann sind immer mehrere Zeilen nötig.

Fremdschlüssel: Notation (5)

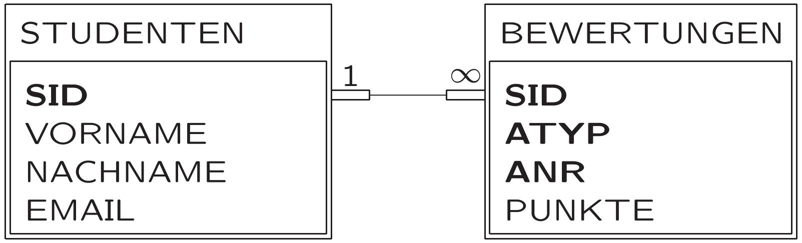
- In der Oracle-DBA-Prüfung wird folgende Notation für relationale Schemata verwendet, z.B. für MUSIC_PIECE(<u>PNO</u>, PNAME, CNO→COMPOSER):

Instance Chart for Table MUSIC_PIECE			
Column Name:	PNO	PNAME	CNO
Key Type:	PK		FK
Nulls/Unique:	NN, U	NN	
FK Table:			COMPOSER
FK Column:			CNO
Datatype:	NUMBER	VARCHAR	NUMBER
Length:	4	40	2

FK: "foreign key", NN: "not null", PK: "primary key", U: "unique".

Fremdschlüssel: Notation (6)

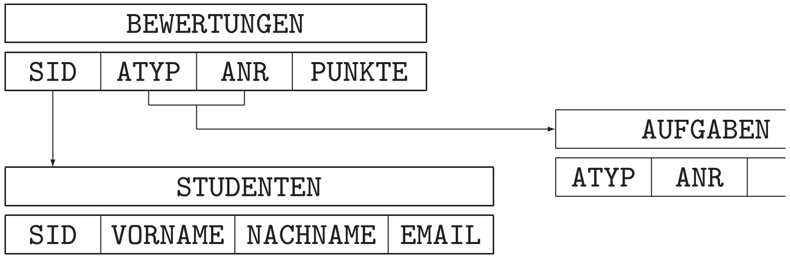
- MS Access stellt Fremdschlüssel in "Relationships" dar (Primärschlüsselattribute fettgedruckt):



- "1" / "∞" symbolisieren das eins-zu-viele-Relationship.

Fremdschlüssel: Notation (7)

- Natürlich kann man auch Pfeile verwenden:



- Achtung: Manche Leute zeichnen den Pfeil auch in die entgegengesetzte Richtung!

  Z.B. im Oracle-DBA-Examen. Man muss genau auf den gegebenen DB-Zustand achten.

Mehr über Fremdschlüssel (1)

Fremdschlüssel und Nullwerte:

- Solange kein “Not Null”-Constraint spezifiziert ist, können Fremdschlüssel Null sein.
- Die Fremdschlüsselbedingung ist sogar dann erfüllt, wenn die referenzierenden Attribute “Null” sind. Das entspricht einem “nil”-Zeiger.
- Wenn ein Fremdschlüssel (FS) mehrere Attribute hat, sollten entweder alle oder keines Null sein.

  Aber Oracle und SQL-92 erlauben teilweise definierte Fremdschlüssel. In Oracle ist die Bedingung erfüllt, wenn mindestens ein FS-Attribut Null ist. Der SQL-92-Standard definiert 3 verschiedene Semantiken.

Mehr über Fremdschlüssel (2)

Gegenseitige Referenzierung:

- Es ist möglich, dass Eltern- und Kindtabelle die gleiche sind, z.B.

  ANG(<u>ANGNR</u>, ANAME, JOB, CHEF^o→ANG, ABTNR→ABT)
  PERSON(<u>NAME</u>, MUTTER^o→PERSON, VATER^o→PERSON)
- Zwei Relationen können sich gegenseitig referenzieren, z.B.

  ANGESTELLTE(<u>ANGNR</u>, ..., ABT→ABTEILUNGEN)
  ABTEILUNGEN(<u>ABTNR</u>, ..., CHEF^o→ANGESTELLTE).
- Übung/Rätsel: Wie kann man Tupel einfügen?

Bitte merken:

- Fremdschlüssel (FS) sind selbst keine Schlüssel!

  Die Attribute eines Fremdschlüssels können Teil eines Schlüssels sein, aber das ist eher die Ausnahme. Die FS-Bedingung hat nichts mit einer Schlüsselbedingung zu tun. Für manche Autoren ist jedoch jedes Attribut, das Tupel identifiziert (nicht unbedingt in der gleichen Tabelle), ein Schlüssel. Dann wären FS Schlüssel, aber normale Schlüssel brauchen dann immer einen Zusatz (“Primär-/Alternativ-”).
- Nur Schlüssel einer Relation können referenziert werden, keine beliebigen Attribute.
- Enthält die referenzierte Relation zwei Attribute, muss der FS auch aus zwei Attributen bestehen (gleiche Datentypen und gleiche Reihenfolge).

FS und Updates (1)

Diese Operationen können Fremdschlüssel verletzen:

- Einfügen in Kindtabelle BEWERTUNGEN ohne passendes Tupel in Elterntabelle STUDENTEN.
- Löschen aus Elterntabelle STUDENTEN, wenn das gelöschte Tupel noch referenziert wird.
- Änderung des FS SID in der Kindtabelle BEWERTUNGEN in einen Wert, der nicht in STUDENTEN vorkommt.

  Wird normalerweise wie Einfügen behandelt.
- Änderung des Schlüssels SID in STUDENTEN, wenn der alte Wert noch referenziert wird.

FS und Updates (2)

Man beachte:

- Löschungen aus der Kindtabelle `BEWERTUNGEN` und Einfügungen in die Elterntabelle `STUDENTEN` können nie zu Verletzungen der Fremdschlüsselbedingung führen.

  Daher muß das DBMS bei diesen Operationen die Bedingung nicht überprüfen.

Reaktionen auf Einfügung, die FS-Bedingung verletzt:

- Das Einfügen wird abgelehnt (Fehlermeldung). Der DB-Zustand bleibt unverändert.

FS und Updates (3)

Reaktionen auf Löschen referenzierter Schlüsselwerte:

- Die Löschung wird abgelehnt. Zustand unverändert.
- Kaskadierendes (rekursives) Löschen: Alle Tupel in `BEWERTUNGEN`, die das gelöschte Tupel in `STUDENTEN` referenzierten, werden mit gelöscht.
- Der Fremdschlüssel wird auf Null gesetzt.

  In SQL-92 enthalten, in DB2 unterstützt, aber nicht in Oracle.
- Der Fremdschlüssel wird auf einen vorher definierten Default-Wert gesetzt.

  In SQL-92 enthalten, aber nicht in Oracle oder DB2.

FS und Updates (4)

Reaktionen auf Updates referenzierter Schlüsselwerte:

- Änderung wird abgelehnt. Der DB-Zustand bleibt unverändert.

  DB2 und Oracle unterstützen nur diese Alternative des SQL-92-Standards. Auf jeden Fall ist die Änderung von Schlüsselattributen schlechter Stil.
- Kaskadierender Update.

  D.h. das Attribut SID in `BEWERTUNGEN` wird genauso geändert, wie das Attribut SID in `STUDENTEN` geändert wurde.
- Fremdschlüssel wird Null gesetzt.
- Fremdschlüssel wird auf Default-Wert gesetzt.

FS und Updates (5)

- Bei der Definition eines Fremdschlüssels muss entschieden werden, welche Reaktion die beste ist.
- Default ist die erste Alternative ("Keine Aktion").
- Für das Löschen aus der Elterntabelle sollten alle Systeme kaskadierendes Löschen unterstützen.

  Das ist eine Art aktive Integritätserzwingung: Das System lehnt die Änderung nicht ab, sondern macht andere Änderungen, um den DB-Zustand zu reparieren.
- Andere Alternativen gibt es bis jetzt nur in wenigen Systemen.