Teil 4: 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

Bedeutung des rel. Modells

- Relationale DBMS (RDBMS) beherrschen derzeit den Markt.

  Z.B. Oracle, IBM DB2, MS SQL Server, Sybase, Informix, CA Ingres.

- Die meisten neuen DB-Projekte nutzen RDBMS.

  Es gibt noch Systeme, die auf einem Netzwerk- oder Hierarchie-DBMS beruhen. Z.B. das hierarchische System IMS von IBM.

- Objektorientierte DBS hauptsächlich für "nicht-Standard-Anwendungen" (z.B. CAD-Daten).

  OODBMS haben einige der Vorteile der RDBMS nicht. Trend geht zu objektrelationalen DBMS. Alle großen Anbieter haben OR-Features.

- XML-DBMS werden derzeit entwickelt.

Beispiel-Datenbank (1)

STUDENTEN			
SID	VORNAME	NAME	EMAIL
101	Ann	Smith	···
102	Michael	Jones	(NULL)
103	Richard	Turner	···
104	Maria	Brown	···

RESULTATE			
SID	KAT	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			
KAT	ANR	THEMA	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
Z	1	SQL	14

Beispiel-Datenbank (2)

- STUDENTEN: eine Zeile für jeden Studenten.
  - ◇ SID: "Studenten-ID" (eindeutige Zahl).
  - ◇ VORNAME, NAME: Vor-und Nachname.
  - ◇ EMAIL: Email-Adresse (kann NULL sein).
- AUFGABEN: eine Zeile für jede Aufgabe.
  - ◇ KAT: Kategorie der Aufgabe.

    Z.B. 'H': Hausaufgabe, 'Z': Zwischenklausur, 'E': Endklausur.
  - ◇ ANR: Aufgabennummer (innerhalb der Kategorie).
  - ◇ THEMA: Thema der Aufgabe.
  - ◇ MAXPT: Max. Punktzahl (Wieviele Punkte ist sie wert?).

Beispiel-Datenbank (3)

- RESULTATE: eine Zeile für jede zu einer Aufgabe abgegebenen Lösung.
  - ◇ SID: Student, der die Lösung abgegeben hat.

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

    Zusammen identifiziert dies eine Zeile in AUFGABEN.
  - ◇ PUNKTE: Punkte, die der Student für die Lösung bekommen hat.
  - ◇ Eine fehlende Zeile bedeutet, dass der Student noch keine Lösung zur Aufgabe abgegeben hat.

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 Länge und Präzision), Datum und Zeit, Geld, 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)

- Erweiterungsfähige DBMS erlauben die Def. 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 Erweiterungsfähigkeit sollte auch ermöglichen, neue Indexstrukturen zu definieren und den Anfrageoptimierer zu erweitern.

- Diese Erweiterungsfähigkeit ist eine wichtige Eigenschaft moderner objektrelationaler Systeme.

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

Datenwerte (3)

- Wie in Kapitel 2 erklärt, sind 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`.

Domains (1)

- Die Spalten `ANR` in `RESULTATE` und `ANR` in `AUFGABEN` sollten den gleichen Datentyp haben (beides sind Aufgabennummern). Das gleiche gilt für `AUFGABEN.MAXPT` und `RESULTATE.PUNKTE`.
- Man kann anwendungsspezif. "Domains" als Name/Abkürzung für Standarddatentypen definieren:

```
CREATE DOMAIN AUFG_NUMMER AS NUMERIC(2)
```

- Man kann sogar den Constraint, dass die Zahl positiv sein muss, hinzufügen.

```
CREATE DOMAIN AUFG_NUMMER AS NUMERIC(2) CHECK(VALUE > 0)
```


Domains (2)

- Dann wird der Spalten-Datentyp indirekt über die Domain definiert:

Spalte "`AUFGABEN.ANR`" →
Spalte "`RESULTATE.ANR`" →
Domain "`AUFG_NUMMER`" → Datentyp "`NUMERIC(2)`"

- Wenn je nötig sein sollte, die Menge möglicher Aufgabennummern zu erweitern, z.B. zu `NUMERIC(3)`, wird durch diese Struktur keine Spalte vergessen.

Domains (3)

- Domains sind nützlich, um zu dokumentieren, dass zwei Spalten die selbe Art Inhalt haben und dass Vergleiche zwischen ihnen Sinn machen.
- Auch wenn z.B. "`PUNKTE`" den gleichen Datentyp "`NUMERIC(2)`" hat, macht diese Anfrage wenig Sinn:

  "Welche Aufgabe hat eine Nummer, die der Anzahl ihrer Punkte entspricht?"
- SQL verbietet jedoch keine Vergleiche von Werten verschiedener Domains.

  Man würde z.B. "Subdomains" brauchen (hat SQL nicht). Vergleiche zwischen verschiedenen Domains sind auf jeden Fall merkwürdig.

Domains (4)

- SQL-92 Standard enthält Domain-Definitionen, bis jetzt unterstützen sie aber nur wenige Systeme.

  Oracle 8i, IBM DB2 V5 und MS SQL Server 7 unterstützen alle `CREATE DOMAIN` nicht. Aber z.B. nutzer-definierte Datentypen in SQL Server (`sp_addtype`) sind sehr ähnlich.
- Domains sind zumindest ein nützlicher Kommentar, um die Verbindung von Spalten besser zu verstehen.
- Auch wenn das RDBMS Domains nicht unterstützt, sollte man sie während des DB-Entwurfs definieren.

  Oracle Designer unterstützt Domains und ersetzt sie durch die Datentypen, wenn `CREATE TABLE`-Statements erstellt werden.

Domains (5)

- Oft können Domain-Namen direkt als Spaltennamen verwendet werden.

  Z.B. hieß in einer alten Version der Beispiel-DB die Aufgabennummer in `AUFGABEN` "`NR`" und in `RESULTATE` "`ANR`". Die neue Version erscheint klarer. Spaltennamen werden automatisch einheitlicher sein, wenn man die Domain-Namen als Spaltennamen verwendet.
- Auch wenn Domains in realen Systemen noch etwas exotisch sind, sind sie ein nützliches Tool, um die Struktur der DB zu verstehen und um Uniformität/Konsistenz im DB-Entwurf sicherzustellen.

  Domains sind ein etwas "höheres Level" als die gegebenen Datentypen.

Atomare Attributwerte (1)

- Das relationale Modell behandelt einzelne Tabelleneinträge als atomar.
- 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 (Beispiel siehe nächste Folie).

Atomare Attributwerte (2)

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

HAUSAUFGABEN				
NR	THEMA	MAXPUNKTE	GELOEST_VON	
			STUDENT	PUNKTE
1	Rel. Alg.	10	Ann Smith	10
			Michael Jones	9
2	SQL	10	Ann Smith	8
			Michael Jones	9
			Richard Turner	10

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 dem Level der gegebenen Datentypen, nicht auf dem Level 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 Datentypfunktionen ausgedrückt werden können.

Relationale DB-Schemata (1)

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

    Die Namen müssen sich unterscheiden, d.h. $A_i \neq A_j$ für $i \neq j$.
  - ◇ für jedes Attribut A_i einen Datentyp/Domain D_i.

    Sei $dom(A_i) := val(D_i)$ (Menge möglicher Wert von A_i).
- 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 $\mathcal{C}$ con Integritätsbedingungen (später definiert).

    Z.B. Schlüssel und Fremdschlüssel.
- D.h. $\mathcal{R} = (\{R_1, \ldots, R_m\},\ sch,\ \mathcal{C})$.

  Es gibt viele verschiedene Notationen für solche Schemata, siehe unten. Verglichen mit den Definitionen in Kapitel 2 sind die Attributnamen neu. Ansonsten sind Relationen nichts anderes als Prädikate.

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).

  Spalten können sogar versch. 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			
KAT	ANR	THEMA	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
Z	1	SQL	14

- Eine Art, ein Schema präzise zu definieren, ist über ein SQL-Statement (siehe Kapitel 9):

```
CREATE TABLE AUFGABEN(KAT   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:

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

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

Schemata: Notation (3)

- Man kann auch den Kopf der Tabelle verwenden:

AUFGABEN			
KAT	ANR	THEMA	MAXPT
⋮	⋮	⋮	⋮

- Oder eine Tabelle mit Spaltendefinitionen:

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

Übung

Definieren Sie ein relationales DB-Schema für eine Sammlung von Kochrezepten.

- Für jedes Rezept muss eine eindeutige Nummer, der Name des Gerichts, eine kurze Erklärung, was zu machen ist und die Backzeit und -temperatur gespeichert werden.
- Für jedes Rezept muss auch eine Menge von Zutaten und für jede Zutat die vorgeschriebene Menge gespeichert werden.

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 in Bezug auf 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, sodass $d_i \in val(D_i)$. 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, 'Rel. Algeb.', 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 in Bezug auf 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.

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)

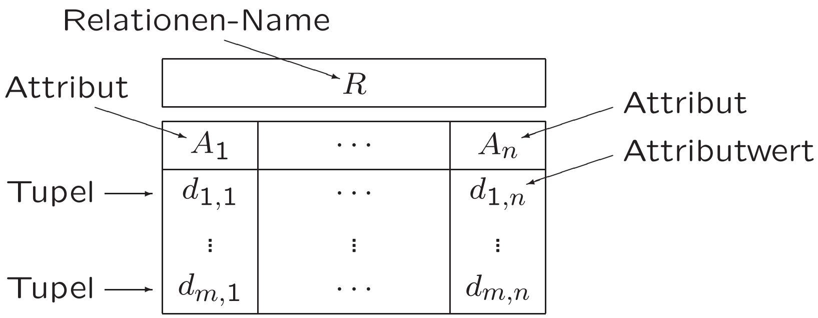
Relationen sind Mengen von Tupeln. Daher

- 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).

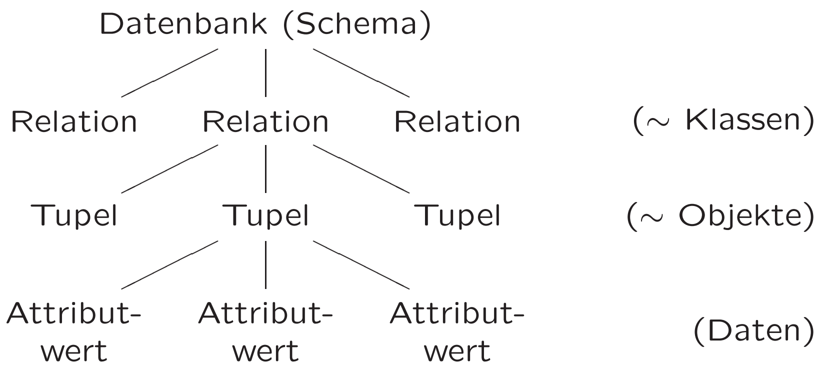
    Also wäre eine Formulierung mit Multimengen korrekt.

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 Feld 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 RM 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 Constraints 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 sein.

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.

    Stellen Sie sich vor, man will in der Tabelle `STUDENTEN` z.B. auch die Telefonnummer der Studenten speichern, aber man kennt möglicherweise nicht von jedem Studenten die Telefonnummer, obwohl wahrscheinlich die meisten eine haben.

  - ◇ Es existiert kein Wert.

    Nicht jeder Student hat z.B. eine zweite Adresse für die Dauer des Semesters (verschieden von Heimatanschrift). Die Tabelle `STUDENTEN` könnte eine solche Spalte jedoch enthalten. Oder eine Tabelle `VORLESUNGEN`: Es könnte eine Spalte `URL` geben, aber nicht jede Vorlesung hat eine Web-Seite.

Nullwerte (3)

- Anwendungen von Nullwerten, fortgesetzt:
  - ◇ Attribut bei diesem Tupel nicht anwendbar.

    Z.B. müssen 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 6).
- 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 hat Drei-Werte-Logik, um Bedingungen mit Nullwerten auszuwerten (wahr/falsch/unbekannt).

  Für diejenigen, die an Zwei-Werte-Logik gewöhnt sind (die meisten), kann es Überraschungen geben – viele 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 nachzuenken, wo Nullwerte gebraucht werden.
- Viele Spalten als "not null" zu erklären, 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.

  Das ist eine Art "Spalten-Constraint" (siehe unten). Zwischen den Datentyp und "NOT NULL" kann man noch andere Spalten-Constraints schreiben (z.B. CHECK), aber meist wird "NOT NULL" zuerst geschrieben. Auf diese Art kann man es auch als Teil des Datentyps sehen.

- Z.B. kann EMAIL in STUDENTEN Null sein:

```
CREATE TABLE STUDENTEN(
              SID        NUMERIC(3)  NOT NULL,
              VORNAME    VARCHAR(20) NOT NULL,
              NAME       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, bei Verwendung der vereinfachten Notationen anders herum vorzugehen:

  STUDENTEN(SID, VORNAME, NAME, $\mathtt{EMAIL}^o$)

- In dieser Notation werden Attribute, die Nullwerte haben 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
NAME	VARCHAR(20)	N
EMAIL	VARCHAR(80)	J

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

Inhalt

Constraints: Übersicht (1)

- Integritätsbedingungen sind Bedingungen, die jeder DB-Zustand erfüllen muss, siehe Kapitel 3.
- Z.B. können im SQL-`CREATE TABLE`-Statement folgende Arten von Constraints festgelegt werden:
  - ◇ `NOT NULL`: Eine Spalte darf nicht null sein.
  - ◇ Schlüssel: Jeder Schlüsselwert nur einmal.
  - ◇ Fremdschlüssel: Werte einer Spalte müssen auch als Schlüsselwert in einer anderen Tabelle auftauchen.
  - ◇ `CHECK`: Spaltenwerte müssen eine Bedingung erfüllen.

    Bedingung kann mehrere Spalten derselben Zeile einbeziehen.

Constraints: Übersicht (2)

- Der SQL-92-Standard enthält ein Statement `CREATE ASSERTION`, das jedoch nicht in den heutigen Datenbank-Systemen implementiert ist.
- Man kann Constraints auch durch SQL-Anfragen, die die Verletzungen ausgeben, oder als Formeln formalisieren.

  Oder man kann die Constraints in natürlicher Sprache angeben. Das DBMS versteht dies zwar nicht und kann daher die Bedingung nicht erzwingen. Aber es kann trotzdem eine nützliche Dokumentation für die Entwicklung von Anwendungsprogrammen sein (Constraints kann man programmieren). Sind Constraints als SQL-Anfragen, die Verletzungen ausgeben, formuliert, kann man sie von Zeit zu Zeit ausführen.

Eindeutige Identifikation (1)

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

  Schlüsselbedingung 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 illegal:

STUDENTEN

<u>SID</u>	VORNAME	NAME	EMAIL
101	Ann	Smith	···
101	Michael	Jones	···
103	Richard	Turner	(null)
104	Maria	Brown	···

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.
- Alle Schlüssel sind Constraints: Sie beruhen auf allen möglichen DB-Zuständen, nicht nur auf dem derzeitigen Zustand.
- Auch wenn im obigen DB-Zustand (mit nur 4 Studenten) der Nachname (`NAME`) als Schlüssel dienen könnte, würde dies zu einschränkend sein.

  Z.B. wäre das zukünftige Einfügen von "`John Smith`" unmöglich.

Eindeutige Identifikation (3)

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

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

- Diese Tabelle erfüllt den Schlüssel `VORNAME`, `NAME`:

STUDENTEN			
SID	<u>VORNAME</u>	<u>NAME</u>	EMAIL
101	Ann	Smith	···
102	John	Smith	···
103	John	Miller	···

Schlüssel: Minimalität (1)

- Sei F eine Formel, die `NAME` als Schlüssel festlegt.
- Sei G eine Formel, die den zusammengesetzten Schlüssel `VORNAME`, `NAME` festlegt.
- Dann $F \vdash G$, d.h. jeder DB-Zustand, der den Key `NAME` erfüllt, erfüllt auch den Key `VORNAME, NAME`.

  Allgemein macht das Hinzufügen von Attributen Schlüssel schwächer (die Bedingung wird von mehr Zuständen erfüllt).

- Wenn `NAME` also als Schlüssel deklariert wurde, ist es nicht interessant, dass `VORNAME, NAME` auch die eindeutige Identifikationseigenschaft hat.

Schlüssel: Minimalität (2)

- Man wird nie zwei Schlüssel deklarieren, sodass einer eine Teilmenge des anderen ist.

  Nur minimale Schlüssel (in Bezug auf "$\subseteq$") sind interessant. Viele Autoren beziehen sogar die Minimalitätsbedingung in die Definition eines Schlüssel ein.

- Der Schlüssel "`NAME`" ist jedoch im Beispiel-Zustand auf Folie 4-53 nicht erfüllt.
- Möchte der DB-Designer diesen Zustand zulassen, so ist die Schlüsselbedingung "`NAME`" zu streng.

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

Schlüssel: Minimalität (3)

- Da der Schlüssel "`NAME`" ausgeschlossen ist, wird der zusammengesetzte Schlüssel "`VORNAME, NAME`" wieder interessant.

  Man wird auch prüfen, dass der Schlüssel "`VORNAME`" aus dem gleichen Grund nicht möglich ist.

- Der DB-Designer 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 (4)

- 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 (5)

- Wenn der Designer entscheidet, dass der Nachteil des Schlüssels "`VORNAME, NAME`" größer als der Vorteil ist, könnte er versuchen, weitere Attribute hinzuzufügen.
- Aber die Kombination "`SID, VORNAME, NAME`" ist uninteressant, weil "`SID`" schon ein Schlüssel ist.
- Sollte der Designer jedoch entscheiden, dass "`VORNAME, NAME`" "eindeutig genug" ist, wäre dies minimal, auch wenn "`SID`" schon ein Schlüssel ist.

  Anzahl der Spalten eines Schlüssels für Minimalität nicht wichtig.

Mehrere Schlüssel

- Eine Relation kann mehr als einen Schlüssel haben.
- Z.B. ist `SID` ein Schlüssel von `STUDENTEN` und `VORNAME, NAME` könnte ein weiterer Schlüssel sein.
- 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	NAME	EMAIL
101	Ann	Smith	···
102	Michael	Jones	(null)
103	Richard	Turner	···
104	Maria	Brown	···

- Gewöhnlich werden die Attribute so sortiert, 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,
             NAME       VARCHAR(20)  NOT NULL,
             EMAIL      VARCHAR(80),
             PRIMARY KEY(SID),
             UNIQUE(VORNAME, NAME))
```

- Die genaue Syntax wird in Kapitel 9 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 `PRIMÄRSCHLÜSSEL`- oder `UNIQUE`-Bedingung festgelegt werden. In SQL-92 und Oracle impliziert die "`PRIMÄRSCHLÜSSEL`"-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. Man sollte dies jedoch alles 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 Befehle, die vorschreiben, was mit Fremdschlüsseln passieren soll, wenn der referenzierte Schlüsselwert geändert wurde.

Der schwächste Schlüssel

- Ein Schlüssel bestehend aus allen Spalten der Tabelle erfordert nur, dass es nie zwei verschiedene Tupel gibt, die in allen Attributen übereinstimmen.

  Theoretisch sind Relationen Mengen: Dann ist es keine Einschränkung. In der Praxis sind Relationen jedoch Multimengen und dieser Schlüssel schützt vor doppelten Zeilen.

- 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:

GELOEST		
STUDENT	HA	PUNKTE
John Smith	1	10
John Smith	2	12
Maria Brown	1	9

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

- Könnte "PUNKTE" auch als Schlüssel dienen?

Übungen (2)

- Betrachten Sie einen Terminkalender:

TERMINE				
DATUM	START	ENDE	RAUM	WAS
Jan. 19	10:00	11:00	IS 726	Michael
Jan. 19	14:00	15:00	IS 726	Siripun
Jan. 19	18:00	20:50	IS 501	INFSCI 2710

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

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

Inhalt

Fremdschlüssel (1)

- Das relationale Modell hat keine expliziten Relationships, Verknüpfungen oder Zeiger.
- 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 `RESULTATE` das Attribut `SID`, welches Primärschlüsselwerte von `STUDENTEN` enthält.

Fremdschlüssel (2)

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

STUDENTEN

SID	VORNAME	NAME	···
101	Ann	Smith	···
102	Michael	Jones	···
103	Richard	Turner	···
104	Maria	Brown	···

RESULTATE

SID	KAT	ANR	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 `RESULTATE` auch in `STUDENTEN` auftaucht.

Fremdschlüssel (3)

- Wenn `SID` in `RESULTATE` ein Fremdschlüssel ist, der `STUDENTEN` referenziert, lehnt das DBMS jeden Versuch ab, eine Lösung für einen nicht existierenden Studenten einzufügen.
- Somit ist die Menge der `SID`-Werte in `STUDENTEN` eine Art "dynamische Domain" für `SID` in `RESULTATE`.
- In relationaler Algebra (Kapitel 6) liefert die Projektion $\pi_{\mathtt{SID}}$ die Werte der Spalte `SID`. Dann lautet die Fremdschlüsselbedingung:

$$\pi_{\mathtt{SID}}(\mathtt{RESULTATE}) \subseteq \pi_{\mathtt{SID}}(\mathtt{STUDENTEN}).$$

Fremdschlüssel (4)

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

  Paare solcher Tupel t und u kann man durch eine Operation der relationalen Algebra ("Join", Kapitel 6) 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 `RESULTATE` genau ein Tupel u in `STUDENTEN` referenziert.

Fremdschlüssel (5)

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

  D.h. Fremdschlüsselbedingung und referentielle Integritätesbedingung sind Synonyme.

- Fremdschlüssel implementiert "eins-zu-viele"-Relationship: Ein Student hat viele Aufgaben gelöst.
- Die Tabelle `RESULTATE`, die den Fremdschlüssel enthält, wird "Kindtabelle" der referentiellen Integritätsbedingung genannt und die referenzierte Tabelle `STUDENTEN` ist die "Elterntabelle".

Fremdschlüssel (6)

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

RESULTATE

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

AUFGABEN

KAT	ANR	···	MAXPT
H	1	···	10
H	2	···	10
Z	1	···	14

Fremdschlüssel (7)

- 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 `KAT` und `ANR`, die in `RESULTATE` vorkommt, auch in `AUFGABEN` existiert.

Fremdschlüssel (8)

- Spalten werden nach der Position in der Deklaration zugeordnet: Ist z.B. (`VORNAME, NAME`) der Schlüssel und (`NAME, VORNAME`) der Fremdschlüssel, werden Einfügungen wahrscheinlich meist Fehler geben.

  Sind die Datentypen von `VORNAME` und `NAME` 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 Schlüssel können referenziert werden: Ein Teil eines Schlüssels oder ein beliebiges Attribut nicht.

  Normalerweise sollte man nur den Primärschlüssel referenzieren, aber SQL erlaubt auch, alternative Schlüssel zu referenzieren.

Fremdschlüssel: Notation (1)

- In der Attributlisten-Notation können Fremdschlüssel durch einen Pfeil und den Namen der referenzierten Tabelle markiert werden. Zusammengesetzte Fremdschlüssel in Klammern:

```
RESULTATE(SID → STUDENTEN,
          (KAT, ANR) → AUFGABEN, PUNKTE)
STUDENTEN(SID, VORNAME, NAME, EMAIL)
AUFGABEN(KAT, 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:

```
VORLESUNGSKATALOG(NR, TITEL, BESCHREIBUNG)
VORLANGEBOT(VN, VNR → VORLESUNGSKATALOG, SEM,
            (DOZ_VORNAME, DOZ_NAME) → DOZENT)
DOZENT(VORNAME, NAME, BUERO, TEL)
```

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

- Manche markieren Fremdschlüssel durch gestricheltes Unterstreichen oder einen Strich oben. Das wird nicht empfohlen, weil die referenzierte Tabelle fehlt.

Fremdschlüssel: Notation (3)

- In SQL können Fremdschlüssel wie folgt deklariert werden:

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


Fremdschlüssel: Notation (4)

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

RESULTATE	SID	KAT	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)

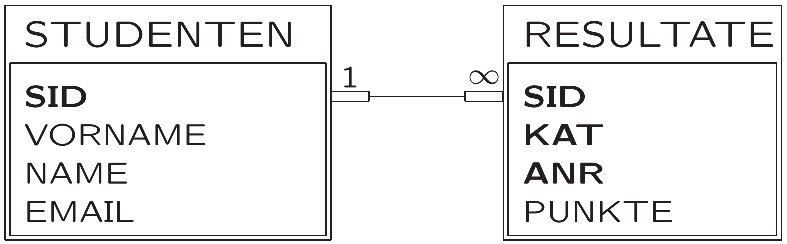
- Im Oracle-DBA-Examen wird folgende Notation verwendet:

Instance Chart für Tabelle MUSIKSTUECK			
Spaltenname:	SNR	SNAME	KNR
Schlüsselart:	PS		FS
Null/Unique:	NN, U	NN	
FS-Tabelle:			KOMPONIST
FS-Spalte:			KNR
Datentyp:	NUMBER	VARCHAR	NUMBER
Länge:	4	40	2

- "FS" = "Fremdschlüssel", "NN" = "Not Null", "PS" = "Primärschlüssel", "U" = "Unique".

Fremdschlüssel: Notation (6)

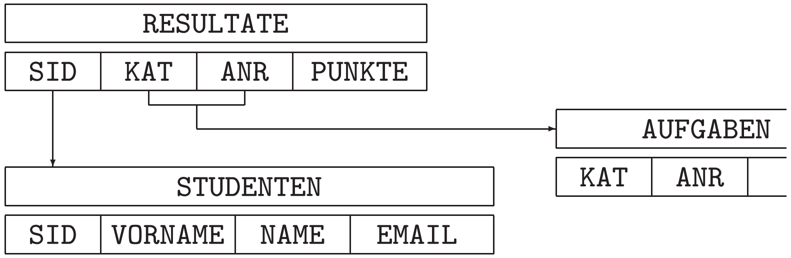
- MS Access stellt Fremdschlüssel in "Relationships" dar (Primärschlüsselattribute fettgedruckt):



- "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(ANGNR, ANAME, JOB, CHEF^o→ANG, ABTNR→ABT)
  PERSON(NAME, MUTTER^o→PERSON, VATER^o→PERSON)
- Zwei Relationen können sich gegenseitig referenzieren, z.B.
  ANGESTELLTE(ANGNR, ..., ABT→ABTEILUNGEN)
  ABTEILUNGEN(ABTNR, ..., 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 irgendeinen 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 `RESULTATE` ohne passendes Tupel in Elterntabelle `STUDENTEN`.
- Löschen aus Elterntabelle `STUDENTEN`, wenn das gelöschte Tupel noch referenziert wird.
- Änderung des FS `SID` in der Kindtabelle `RESULTATE` 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:

- Löschen aus `RESULTATE` (Kindtabelle) und Einfügen in `STUDENTEN` (Elterntabelle) können nie zu Verletzungen der Fremdschlüsselbedingung führen.

  Das heißt, dass das DBMS bei diesen Operationen die Bedingung nicht überprüfen muss.

Reaktionen auf unerlaubtes Einfügen:

- Das Einfügen wird abgelehnt.
  Der DB-Zustand bleibt unverändert.

FS und Updates (3)

Reaktionen auf Löschen referenzierter Schlüsselwerte:

- Löschen wird abgelehnt.
- Die Lösch-Kaskade: Alle Tupel aus `RESULTATE`, die das gelöschte Tupel in `STUDENTEN` referenzierten, werden auch gelöscht.
- Der Fremdschlüssel wird 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.
- Die Update-Kaskade.

  D.h. das Attribut SID in `RESULTATE` wird genauso geändert, wie das Attribut SID in `STUDENTEN` geändert wurde.
- Der 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").
- Wenigstens für das Löschen aus der Elterntabelle sollten alle Systeme Lösch-Kaskade 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.

Übung (1)

Definieren Sie ein rel. DB-Schema für ein Hotel:

- Informationen über Gäste:
  Vorname, Nachname und Adresse.
- Informationen über Zimmer: Einzel- oder Doppelzimmer? Wie hoch ist die offizielle Zimmermiete? Wann zuletzt renoviert?
- Informationen über Aufenthalte: Welches Zimmer von welchem Gast von wann bis wann? Wieviel Zimmermiete wurde von ihm/ihr verlangt?

  Es kann weniger als die offizielle Zimmermiete sein.

Übung (2)

Bitte definieren Sie Folgendes:

- Tabellen- und Spaltennamen.
- Schlüssel.
- Fremdschlüssel.
- Null-Bedingungen.

Beschreiben Sie außerdem weitere Integritätsbedingungen, die notwendig sein könnten.