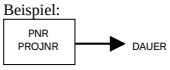
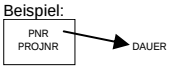
Grundbegriffe bei funktionalen Abhängigkeiten

Triviale FD: X → X
Volle funktionale Abhängigkeit:
A1, A2, ..., An → B1, B2, ..., Bm
B = { B1, B2, ..., Bm } ist voll funktional abhängig von A = {A1, A2, ..., An}, wenn B funktional abhängig von A, aber nicht funktional abhängig von einer echten Teilmenge von A ist.
Beispiel:



A → B ist eine *partielle Abhängigkeit*, wenn ein Attribut Ai in A existiert, so daß (A - {Ai}) → B gilt:
Beispiel:



1. Normalform

- Definition 1NF:
  Ein Relationenschema R ist in 1NF genau dann, wenn all seine Wertebereiche nur atomare Werte besitzen. Atomar heißt hier: keine Mengen oder Tupel von Werten.
- Überführung in 1NF:
  Mit Primärschlüssel der Vaterrelation eigene Relation erzeugen und diese aus der Vaterrelation streichen
- 1NF verursacht immer noch viele Änderungsanomalien, da verschiedene Entity-Mengen in einer Relation gespeichert werden können bzw. aufgrund von Redundanz innerhalb einer Relation (Bsp. PRÜFLING: wiederholte Speicherung von immer gleichen Fachbereichsdaten).
- 2NF vermeidet einige der Anomalien dadurch, indem nicht voll funktional (partiell) abhängige Attribute eliminiert werden

2. Normalform

- Definition Primärattribut:
  Ein Primärattribut (Schlüsselattribut) eines Relationenschemas ist ein Attribut, das zu mindestens einem Schlüsselkandidaten des Schemas gehört.
- Definition 2NF:
  Ein Relationenschema R ist in 2NF, wenn es in 1NF ist und jedes Nicht-Primärattribut voll funktional von jedem Schlüsselkandidaten von R abhängt.
- Überführung in 2NF:
  1. Bestimme funktionale Abhängigkeiten zwischen Nicht-Primärattributen und Schlüsselkandidaten.
  2. Eliminiere partiell abhängige Attribute und fasse sie in eigener Relation zusammen (unter Hinzunahme der zugehörigen Primärattribute)

3. Normalform

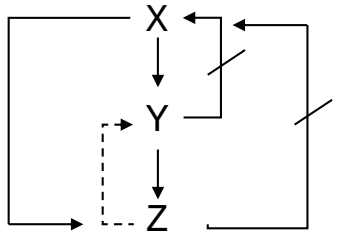
- Definition 3NF:
  Ein Relationenschema R ist in 3NF, wenn es in 2NF ist und jedes Nicht-Primärattribut von keinem Schlüsselkandidaten von R transitiv abhängig ist.
- Überführung in 3NF:
  Aufbrechen in zwei Relationen, hängt von der Semantik der Daten ab

BCNF

- Definition BCNF:
  Ein Relationenschema R ist in BCNF, wenn jeder Determinant (linke Seite einer FD) ein Schlüsselkandidat von R ist. Ein Attribut (oder eine Gruppe von Attributen), von dem andere voll funktional abhängen, heißt Determinant.
- Überführung in BCNF:
  1. Determinante bestimmen
  2. Verletzte BCNF-Terme in den FDs finden
  3. Aufbrechen der Relationen mit verletzten FDs

Transitive Abhängigkeit

- Definition Transitive Abhängigkeit
  Eine Attributmenge Z von Relationenschema R ist transitiv abhängig von einer Attributmenge X in R, wenn gilt:
  - X und Z sind disjunkt
  - Es existiert eine Attributmenge Y in R, so daß gilt:
  - X → Y, Y → Z, ¬ (Y → X), Z ⊄ Y



Z → Y zulässig
Strikte Trans.: unzulässig

TRK-Beispiele

Finde die Namen und Alter aller Segler mit einem Rating größer 7.

$$\{P \mid \exists S \in Sailors \ (S.rating > 7 \wedge P.sname = S.sname \wedge P.age = s.age)\}$$

Finde die Namen der Segler, die ein rotes Boot reserviert haben.

$$\{P \mid \exists S \in Sailors \ \exists R \in Reserves \ \exists B \in Boats \ (R.sid = S.sid \wedge B.bid = R.bid \wedge B.color = 'red' \wedge P.sname = S.sname)\}$$

Finde die Namen der Segler, die mindestens zwei Boote reserviert haben.

$$\{P \mid \exists S \in Sailors \ \exists R1 \in Reserves \ \exists R2 \in Reserves \ (S.sid = R1.sid \wedge R1.sid = R2.sid \wedge R1.bid \neq R2.bid \wedge P.sname = S.sname)\}$$

Finde die Namen der Segler, die alle Boote reserviert haben.

$$\{P \mid \exists S \in Sailors \forall B \in Boats \ (\exists R \in Reserves \ (S.sid = R.sid \wedge R.bid = B.bid \wedge P.sname = S.sname))\}$$

Finde die Segler, die alle roten Boote reserviert haben.

$$\{P \mid \exists S \in Sailors \ \forall B \in Boats \ (B.color = 'red' \Rightarrow (\exists R \in Reserves \ (S.sid = R.sid \wedge R.bid = B.bid)))\}$$

Andere Schreibweise:
p⇒ q ist logisch äquivalent to ¬p∨q

$$\{P \mid \exists S \in Sailors \ \forall B \in Boats \ (B.color \neq 'red' \vee (\exists R \in Reserves \ (S.sid = R.sid \wedge R.bid = B.bid)))\}$$

Ableiten von funktionalen Abhängigkeiten

- Mit einer Menge von gegebenen FDs können meist weitere abgeleitet werden:
  angestellter → stufe, stufe → gehalt impliziert angestellter → gehalt
- Eine FD *f wird impliziert* durch eine Menge von FDs *F*, wenn *f* gilt wenn auch immer alle FDs in *F* gelten.
  F^+ = *Hülle (closure) von F* ist die Menge aller FDs, die durch F impliziert werden
- Armstrongsche Axiome (X, Y, Z sind Mengen von Attributen):
  Reflexivität: Wenn $X \subseteq Y$, dann $Y \rightarrow X$
  Augmentation: Wenn $X \rightarrow Y$, dann $XZ \rightarrow YZ$ für beliebige Z
  Transitivität: Wenn $X \rightarrow Y$ und $Y \rightarrow Z$, dann $X \rightarrow Z$
- Armstrongsche Axiome sind *klare* und *vollständige* Inferenzregeln für FDs!
- Zusätzliche Regeln (abgeleitet aus den Armstrong-Axiomen)
  Additivität (Union): Wenn $X \rightarrow Y$ und $X \rightarrow Z$, dann gilt $X \rightarrow YZ$
  Projektivität (*Dekomposition*) Wenn $X \rightarrow YZ$, dann gilt $X \rightarrow Y$ und $X \rightarrow Z$
  Pseudotransitivität Wenn $X \rightarrow Y$ und $WY \rightarrow Z$ dann gilt $XW \rightarrow Z$

Verlustfreiheit/Abhängigkeitsbewahrung

Verlustfreiheit:

- Die Dekomposition von R in X und Y ist verlustfrei bezüglich F genau dann wenn die Hülle von F folgende FDs enthält:
  - $X \cap Y \rightarrow X$ oder $X \cap Y \rightarrow Y$
  - Interpretation: Gemeinsame Attribute von X und Y müssen Schlüssel für X oder Y sein
  - Verlustfreie Zerlegung von ABC in AB und BC, wenn gilt: B→A oder B→C
- Insbesondere gilt: Die Zerlegung von R in UV und R - V ist verlustfrei wenn $U \rightarrow V$ in R gilt (U = gemeinsame Attribute in beiden Teilrelationen)

Abhängigkeitsbewahrung:

- Abhängigkeitsbewahrend, wenn R in X, Y und Z zerlegt wird, und alle FDs eingehalten werden, die in X, in Y und in Z gelten, dann müssen alle FDs, die in R gegeben waren, auch gelten.
- Dekomposition von R in X und Y ist *abhängigkeitsbewahrend* wenn $(F_X \quad F_Y)^+ = F^+$
- d.h., wenn wir nur Abhängigkeiten in der Hülle F^+ betrachten, die in X geprüft werden können ohne Inanspruchnahme von Y, und in Y ohne Inanspruchnahme von X, impliziert dies alle Abhängigkeiten in F^+
- Wichtig ist in dieser Definition F + zu betrachten, nicht F:
- Beispiel: ABC, $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow A$, zerlegt in AB und BC
  - $F_{AB} = \{A \rightarrow B\}$, $F_{BC} = \{B \rightarrow C\}$, $C \rightarrow A$ fehlt in beiden Mengen
  - $F^+ = F \quad \{ A \rightarrow C, B \rightarrow A, C \rightarrow B \}$
  - Daraus folgt: $B \rightarrow A$ in F_{AB} $C \rightarrow B$ in F_{BC}
  - Aus Transitivität von $C \rightarrow B$ und $B \rightarrow A$ folgt: $C \rightarrow A$
    ⇒ Dekomposition ist abhängigkeitsbewahrend
- Algorithmus ist exponentiell von der Größe von F abhängig

Abbildung ERD auf Relationenmodell

- 1:n-Beziehung: Fremdschlüssel
- 1:1-Beziehung: 1 Tabelle oder Fremdschlüssel in einer Tabelle (bei 0:1, siehe Abbildungsregeln)
- m:n-Beziehung: eigene Tabelle mit Fremdschlüsseln
- m:n:p-Beziehung: eigene Tabelle mit Fremdschlüsseln
- ISA-Beziehung: 3 Relationen mit Fremdschlüssel in den ISA-Relationen auf die Vaterrelation oder 2 Relationen (Flachklopfen, nur bei totaler ISA) oder 1 große Relation – aber mit vielen NULL-Werten
- Schwache Entities: siehe 1:1-Beziehung

- ISA-Eigenschaften:
  p=partiell (nicht alle Elemente erfasst), t=total (alle Elemente erfasst)
  n=nicht disjunkt (die eine Relation kann Tupel der anderen enthalten), d=disjunkt (keine Schnittmenge)

Abbildungsregeln ERD auf Relationenmodell

... nur "E0"	(1,1) (1,1)	E0 (K1, K2, A) oder E0 (K2, K1, A)
bleibt E1 + E2	(0,1) (1,1) (1,*) (1,1) (0,*) (1,1)	E1 (K1, ...) E2 (K2, ..., A, *K1*)
entsteht ein neues E3	(0,1) (0,1)	E3 (*K1*, *K2*, A) oder E3 (*K2*, *K1*, A)
	(1,*) (0,1) (0,*) (0,1)	E3 (*K2*, *K1*, A)
	(1,*) (1,*) (1,*) (0,*)	E3 (*K1, K2*, A)

Relationenalgebra

Basis-Operationen:
Selektion (σ) Auswahl einer Teilmenge von Tupeln aus der Relation
Projektion (π) Auswahl von Spalten aus einer Relation
Kreuzprodukt (×) Kombination zweier Relationen
Mengendifferenz (-) Menge der Tupel aus Relation 1, aber nicht in Relation 2
Vereinigung (∪) Menge der Tupel aus Relation 1 und Relation 2

Logischer DB-Entwurf

Nachweis von FDs:

- Mit Armstrongschen Axiomen
- Mit Hüllenberechnung F^+: $(F)^0$ =A, $(F)^1$=AB, …, $(F)^n$=ABCD…=$(F)^+$
  ⇨ wenn $(F)^+$=U, dann ist F Schlüsselkandidat

Schlüsselkandidaten ermitteln:
1.-n. Schritt ⇨ untersuche alle n-elementigen Mengen: bilde Hüllen, wenn F^+=U, dann liegt ein Schlüsselkandidat vor; beachte: wegen Minimalität darf im i. Schritt keiner der bisher ermittelten Schlüsselkandidaten enthalten sein

Heuristik:

- Attribute, die nur rechts stehen, gehören zu keinem Schlüsselkandidaten
- Attribute, die nur links stehen, müssen zu jedem Schlüsselkandidaten gehören

Datentypen in SQL

- String-Datentypen
  - CHARACTER[(n)] (Abk. CHAR)
  - VARCHAR(n) (Abk. VARCHAR)
  - BIT[(n)]
  - BIT VARYING[(n)]
- Numerische Datentypen
  - DECIMAL(p,q) (Abk. DEC)
  - INTEGER (Abk. INT)
  - SMALLINT halbes Wort
  - FLOAT(p) p=Präzision
  - REAL
  - DOUBLE PRECISION
- Besondere Typen
  - DATE Datum (Format wählbar)
  - TIME Datum und Uhrzeit
  - TIMESTAMP Zeitstempel mit Mikrosek.-Präzision
  - INTERVAL Datums- und Zeitintervalle
- Besonderheiten der Typsysteme in den einzelnen DBMS beachten (z.B. VARCHAR2 oder NUMBER = 40stellige Nummer in Oracle 8)

GROUP BY und HAVING

SELECT [DISTINCT] *target-list*
FROM *relation-list*
WHERE *qualification*
GROUP BY *grouping-list*
HAVING *group-qualification*

- Die *target-list* enthält (i) Attributnamen (ii) Terme mit Aggregat-Operatoren (z.B. MIN (*S.age*)).
  - Die Liste der Attribute (i) muß eine Teilmenge der Liste in *grouping-list* sein. Jedes Ergebnistupel korrespondiert mit einer *Group*.
  - Eine *Group* ist eine Menge von Tupeln, die die gleichen Werte in den Attributen hat, die in *grouping-list* genannt sind
- Die Ausdrücke in *group-qualification* müssen einen einzelnen Wert (Skalar) pro Group liefern. Durch die HAVING-Klausel wird entschieden, ob ein Ergebnistupel für eine Group generiert wird.
- Wenn GROUP BY fehlt, wird die gesamte Tabelle als eine einzige Gruppe behandelt.
- Auswahl der Tupel durch die WHERE-Klausel
  Das Kreuzprodukt von *relation-list* wird berechnet, Tupel, die die WHERE-Bedingung (*qualification*) nicht erfüllen, werden entfernt, *`überflüssige'* Attribute werden gelöscht.
- Bildung von Gruppen durch die GROUP BY Klausel
  Die verbleibenden Tupel werden in Gruppen partitioniert, bestimmt durch die Werte der Attribute in *grouping-list*.
- Auswahl der Gruppen, die die HAVING-Klausel erfüllen
  Die HAVING-Klausel (*group-qualification*) wird angewandt, um einige Gruppen zu eliminieren. Ausdrücke in *group-qualification* müssen einen skalaren Wert pro Gruppe liefern!
- Ein Attribut in *group-qualification*, das nicht Argument einer Standardfunktion ist, erscheint auch in der *grouping-list*. (SQL nutzt hier keine Primärschlüssel-Semantik!)
- Ein Antwort-Tupel wird pro Gruppe generiert, die die HAVING-Bedingung erfüllt.

Operationen auf Tupel und Sichten

Einfügen:
```
INSERT INTO table [(attribute-list)]
        { VALUES (value-list) |
          table-expr |
          DEFAULT VALUES }
```

Löschen:
```
DELETE [FROM] table
WHERE qualification
```

Ändern:
```
UPDATE table
SET update-assignment-list
WHERE qualification
```

Erstellen von Sichten: (Ändern wie bei Relationen)
```
CREATE VIEW view [(attribute-list)]
AS table-exp
```

Löschen von Sichten:
```
DROP VIEW view
```

Aufbau von Triggern

```
CREATE TRIGGER trigger-name
BEFORE | AFTER
INSERT | UPDATE [OF column1, column2, ... ] | DELETE
        ON table-name
[ FOR EACH ROW ]
WHEN [ predicate ]
[ sql-block ]
```

SQL-Beispiele (aus P2004+SQL-Übung)

1.) Gesucht sind für Heinz Becker die Namen aller Restaurants (ob er sie besucht oder nicht), die keines seiner Lieblingsbiere im Sortiment haben.

```
SELECT S.Rname
FROM Sortiment S
WHERE S.Rname NOT IN
  (SELECT S2.Rname
   FROM Sortiment S2
   WHERE S2.Bsorte IN
     (SELECT L.Bsorte
      FROM Libelingsbier L
      WHERE L.Bname='Becker' and L.Bvorname='Heinz'));
```

2.) Welcher Passagier hat das meiste Geld für Reisen ausgegeben?

```
SELECT PsgNr
FROM HAT_GEB
GROUP BY PsgNr
HAVING SUM(Preis) =
  (SELECT MAX(SUM(PREIS))
   FROM HAT_GEB
   GROUP BY PsgNr);
```

3.) Welche Seriennummern kommen bei mehreren Flugzeugtypen vor?

```
SELECT F.Seriennr
FROM Flugzeug F
GROUP BY F.Seriennr
HAVING COUNT(*)>1;
```

SQL-Queries (+Beispiele)

Einfache Query:
```
SELECT [DISTINCT] target-list FROM relation-list WHERE qualification
```

Ausdrücke und Zeichenketten:
```
SELECT S.age, age1=S.age-5, 2*S.age AS age2
FROM Sailors S
WHERE S.sname LIKE 'B_%B'
```

- Illustriert den Gebrauch von arithmetischen Ausdrücken und Mustervergleichen: *Finde Tripel (Alter des Seglers und zwei Attribute, definiert durch Ausdrücke) für die Segler, deren Namen mit einem 'B' beginnen und enden und die mindestens drei Zeichen enthalten.*
- **AS** und **=** sind zwei Varianten, um Attribute im Resultat zu benennen
- **LIKE** wird verwendet für String-Vergleiche (Matching). **`_'** repräsentiert ein beliebiges Zeichen und **'%'** steht für 0 oder mehr beliebige Zeichen.

Mengenoperatoren:
- UNION: für die Berechnung der Vereinigung zweier vereinigungs-verträglicher Mengen von Tupeln (welche selbst Ergebnis von SQL-Anfragen sind)
- INTERSECT: Für die Berechnung des Durchschnitts von zwei beliebigen vereinigungsverträglichen Tupelmengen
- EXCEPT: Für die Berechnung der Differenz von zwei beliebigen vereinigungsverträglichen Tupelmengen

Mengenvergleichsoperatoren:
IN bzw. NOT IN:
```
SELECT S.sname
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                FROM Reserves R
                WHERE R.bid=103)
```

EXISTS ist ein weiterer Mengenvergleichsoperator, wie IN (Test, ob eine Menge leer ist)

ANY und ALL:
```
SELECT *
FROM Sailors S
WHERE S.rating >
          ANY (SELECT S2.rating
          FROM Sailors S2
          WHERE S2.sname='Horatio')

SELECT *
FROM Sailors S
WHERE S.rating > =
          ALL (SELECT S2.rating
          FROM Sailors S2)
```

Erzeugen/Ändern/Löschen von Objekten (Beispiele)

```
CREATE TABLE Reserves
( sid       INTEGER ,
  bid       INTEGER,
  day       DATE NOT NULL,
  PRIMARY KEY (sid,bid),
  FOREIGN KEY (sid) REFERENCES Sailors,
  FOREIGN KEY (bid) REFERENCES Boats(bid));

ALTER TABLE Sailors
[MODIFY telefon varchar(10) |
ADD telephone |
DROP phone]

DROP TABLE Boats
```

Kopie einer Tabelle erstellen mit veränderter Spalte (Beispiel aus P2004):
```
CREATE TABLE Sortiment2
AS (SELECT Rname, Bsort, (AnLagerü100) as LagerNeu
    FROM Sortiment);
```

Aggregations-Operatoren

COUNT (*)	Anzahl Tupel
COUNT ([DISTINCT] A)	Anzahl (verschiedener) Werte in A
SUM ([DISTINCT] A)	Summe (verschiedener) Werte in A
AVG ([DISTINCT] A)	Durchschnitt (versch.) Werte in A
MAX (A)	Maximal-Wert in Spalte A
MIN (A)	Minimal-Wert in Spalte A

GRANT/REVOKE

GRANT privileges	INSERT (col-name), UPDATE (col), DELETE, REFERENCES, SELECT
ON object	
TO users	Liste von Benutzern oder PUBLIC
[WITH GRANT OPTION]	dyn. Weitergabe von Rechten

REVOKE [GRANT OPTION FOR]	Entziehen der Rechteweitergabe möglich
Privileges	siehe GRANT
ON object	
FROM users	Benutzerliste oder PUBLIC
{RESTRICT | CASCADE}	entweder dürfen keine Abhängigkeiten mehr existieren oder die Rechte werden kaskadierend entzogen