Logischer DB-Entwurf

Entwurf eines relationalen DB-Schemas

- Ziel:
  - Theoretische Grundlage für den Entwurf eines “guten“ relationalen DB-Schemas (Normalisierungslehre, Entwurfstheorie)
- Güte eines DB-Schemas:
  - Leichte Handhabbarkeit, Übersichtlichkeit
  - Entwurfstheorie beschreibt die “Güte“
- Was macht eine schlechten DB-Entwurf aus?
  - Redundanzen
  - Löschanomalien
  - Potentielle Inkonsistenz (Änderungsanomalien)
  - Einfügeanomalien
    - Oft hervorgerufen durch “Vermischung“ von Entities
- Normalisierung von Relationen
  - Hilft, einen gegebenen Entwurf zu verbessern
- Synthese von Relationen
  - Zielt auf die Konstruktion eines “optimalen“ DB-Schemas

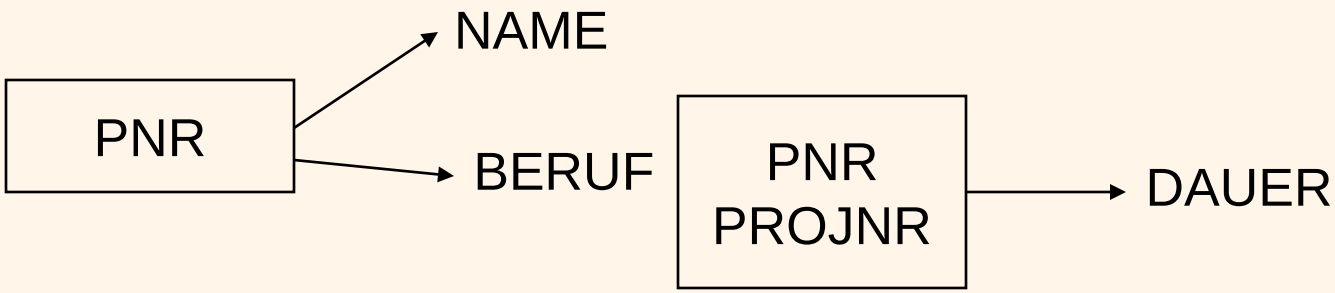
Funktionale Abhängigkeiten

- Funktionale Abhängigkeit (*Functional Dependency*) FD

  Die FD $X \rightarrow Y$ gilt (X bestimmt Y funktional), wenn für alle Relationen r des Relationenschemas R gilt: Zwei Tupel, deren Komponenten in X übereinstimmen, stimmen auch in Y überein. X und Y sind Mengen von Attributen.

  Formal: $\forall u \in R \;\; \forall v \in R \;\; (u[X] = v[X]) \Rightarrow (u[Y] = v[Y])$

Graphische Notation Abhängigkeitsdiagramme:



Grundbegriffe bei funktionalen Abhängigkeiten

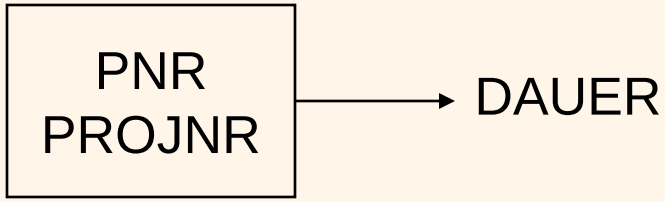
Triviale FD: X $\rightarrow$ X

Volle funktionale Abhängigkeit:

A1, A2, ..., An $\rightarrow$ B1, B2, ..., Bm

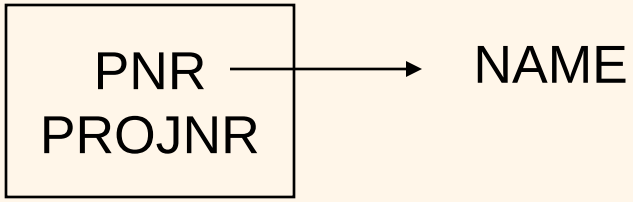
B = { B1, B2, ..., Bm } ist voll funktional abhängig von A = {A1, A2, ..., An}, wenn B funktional abhängig von A, aber nicht funktional abhängig von einer echten Teilmenge von A ist.

Beispiel:



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

Beispiel:



Ableiten von funktionalen Abhängigkeiten

- Mit einer Menge von gegebenen FDs können meist weitere abgeleitet werden:
  - angestellter $\rightarrow$ stufe, stufe $\rightarrow$ gehalt impliziert angestellter $\rightarrow$ 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!

Ableiten von funktionalen Abhängigkeiten (Forts.)

- 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 $XW \rightarrow Z$
- Beispiel: Contracts(*cid,sid,jid,did,pid,qty,value*) mit:
  cid = Vertrags-ID (*Contract*), sid = Lieferanten-ID (Supplier),
  jid = Projekt-ID (*Job*), did = Abteilungs-ID (*Department*),
  pid = Teil-ID (*Part*), qty = Menge (*Quantity*), value = Preis (*Value*)
  - C ist Schlüssel: $C \rightarrow CSJDPQV$
  - Projekt J kauft jedes Teil P in einem einzigen Contract: $JP \rightarrow C$
  - Abteilung D kauft höchstens ein Teil von einem Lieferanten: $SD \rightarrow P$
- $JP \rightarrow C$, $C \rightarrow CSJDPQV$ impliziert $JP \rightarrow CSJDPQV$
- $SD \rightarrow P$ impliziert $SDJ \rightarrow JP$
- $SDJ \rightarrow JP$, $JP \rightarrow CSJDPQV$ impliziert $SDJ \rightarrow CSJDPQV$

Ableiten von funktionalen Abhängigkeiten (Forts.)

- Berechnung der Hülle einer Menge von FDs kann sehr teuer werden (Größe der Hülle wächst exponentiell mit der Anzahl Attribute)
- Typische Frage: Prüfen, ob eine gegebene FD $X \rightarrow Y$ in der Hülle einer Menge von FDs *F* ist. Effiziente Prüfung:
  - Berechne die *Attributhülle* von X (bezeichnet als X^+) bezogen auf *F:*
    - Menge aller Attribute A, so daß $X \rightarrow A$ in F^+ ist
    - Es gibt einen Algorithmus zur Berechnung
  - Prüfen, ob Y in X^+ enthalten ist
- Gegeben F = { $A \rightarrow B$, $B \rightarrow C$, $CD \rightarrow E$ } Impliziert dies $A \rightarrow E$?
  - d.h. ist $A \rightarrow E$ in der Hülle F^+ ? Äquivalent: Ist E in A^+ ?
- Spezialfall: Bestimmung von Schlüsselkandidaten - ebenfalls mit einem solchen Algorithmus möglich

Unnormalisierte Relation

- Beispiel:

PRÜFUNGSGESCHEHEN

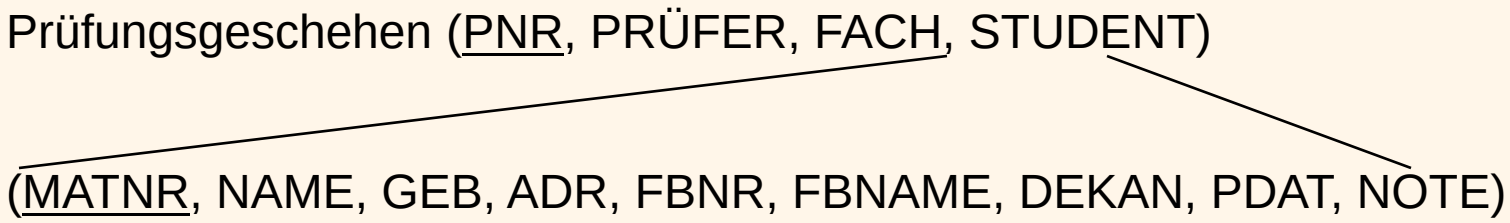
(PNR,	PRÜFER,	FACH,	STUDENT	(MATNR,	NAME, ...))
1	MEIER	GI		1234	MÜLLER
				5678	MAIER
2	ALT	DB		5678	MAIER
				0070	BONTE ....

enthält Attribute, die wiederum Relationen sind

Darstellung von komplexen Objekten (hierarchische Sichten)

- Vorteile:
  - Clusterbildung
  - Effiziente Verarbeitung in einem hierarchischen Objekt
- Nachteile
  - Redundanzen (bei n:m-Beziehungen)
  - Asymmetrie (nur eine Richtung der Beziehung)
  - Anomalien bei Aktualisierung

Überführung in 1NF



Normalisierung (Überführung in 1NF):

1. Starte mit der übergeordneten Relation (Vaterrelation)
2. Nimm ihren Primärschlüssel und erweitere jede unmittelbar untergeordnete Relation damit zu einer selbständigen Relation
3. Streiche alle nicht-einfachen Attribute (untergeordnete Relation) aus der Vaterrelation
4. Wiederhole diesen Prozeß ggf. rekursiv

Regeln:

- Nicht-einfache Attribute bilden neue Relationen
- Primärschlüssel der übergeordneten wird an untergeordnete Relation angehängt (*copy down the key*)

Erste Normalform (1NF)

Relationenschema in 1NF:

PRÜFUNG (PNR, PRÜFER, FACH)

PRÜFLING (PNR, MATNR, NAME, GEB, ADR, FBNR, FBNAME, DEKAN, PDAT, NOTE)

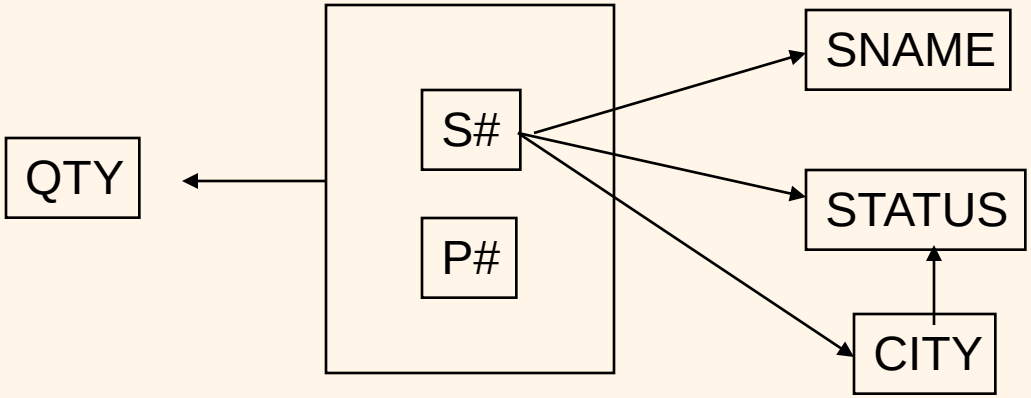
- Definition 1NF:

  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.

- 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

Separierung verschiedener Entity-Mengen in eigene Relationen

Beispiel-Relationenschema



- Relationenschema F (S#, SNAME, STATUS, CITY, P#, QTY)
  - S# = Lieferanten-Identifier
  - SNAME = Name des Lieferanten
  - STATUS = Status des Lieferanten
  - CITY = Ort des Lieferanten
  - P# = Teile-Identifier
  - QTY = Liefermenge
- Anomalien:
  - INSERT: Lieferantendaten können nur gespeichert werden, wenn eine Lieferung besteht
  - DELETE: Löschen der letzten Lieferung eines Lieferanten löscht alle Daten für den Lieferanten
  - UPDATE: Redundanz kann zu Inkonsistenzen beim Update führen

Zweite Normalform (2NF)

- Definition Primärattribut:

  Ein Primärattribut (Schlüsselattribut) eines Relationenschemas ist ein Attribut, das zu mindestens einem Schlüsselkandidaten des Schemas gehört.

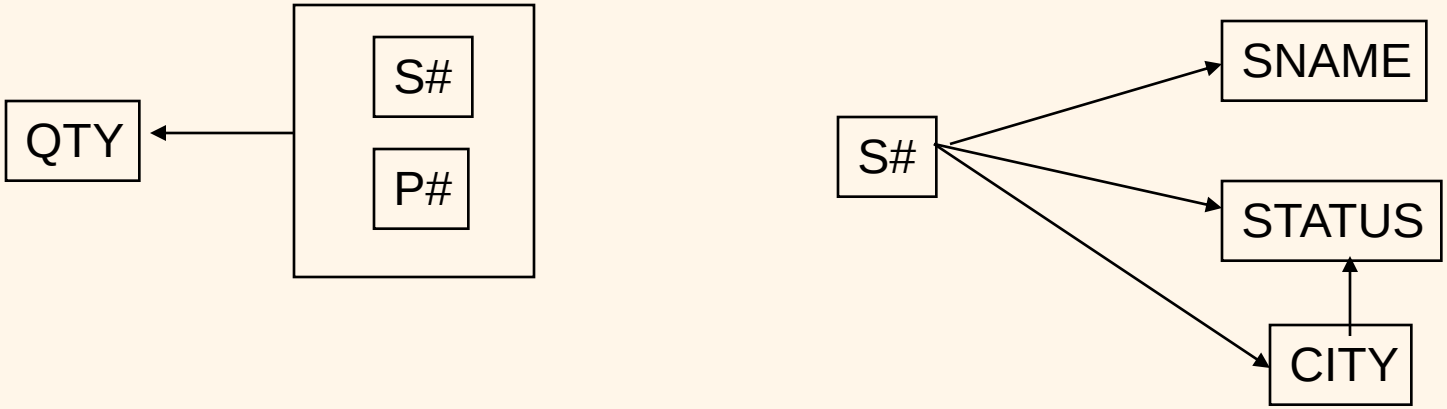
- Definition 2NF:

  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)

Überführung in 2NF

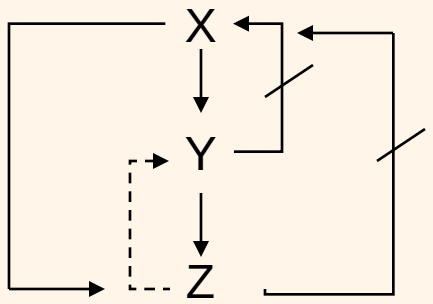


- Auftrennen des Relationenschemas F in 2 Schemas G und H
  - G: (S#, SNAME, STATUS, CITY)
  - H: (S#, P#, QTY)
- Die transitive Abhängigkeit von S#, STATUS und CITY führt zu neuen Anomalien:
  - Obwohl STATUS von CITY abhängig ist (ein Status ist in unserem Beispiel für eine Stadt spezifisch, muß man erst einen Lieferanten speichern, bevor man den Statuscode einer Stadt speichern kann.
  - Es treten die gleichen Anomalien auf, wie vorhin beschrieben

Dritte Normalform (3NF)

- 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:
  - $X \rightarrow Y$, $Y \rightarrow Z$, $\neg (Y \rightarrow X)$, $Z \not\subset Y$

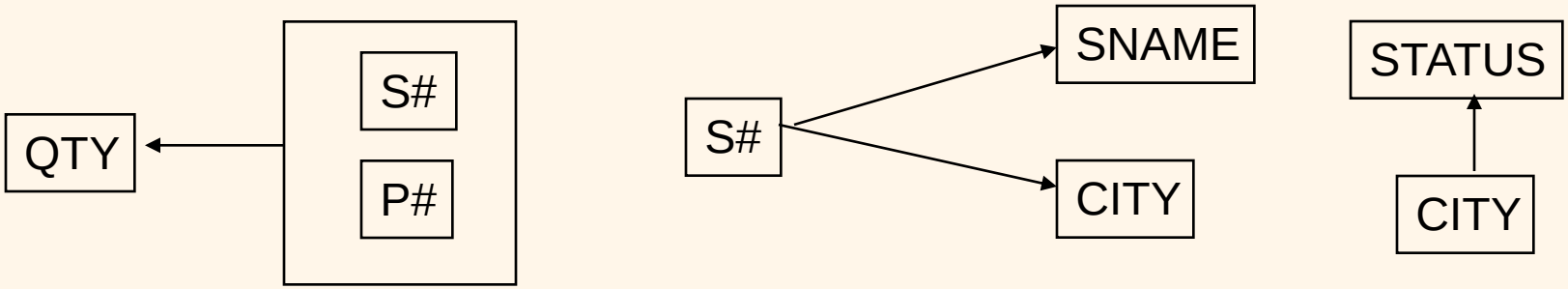


$Z \rightarrow Y$ zulässig
Strikte Trans.: unzulässig

- 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



- Relationenschema in 3NF

  H: (S#, P#, QTY)

  Aufbrechen von G in I und J

  I: (S#, SNAME, CITY)

  J: (CITY, STATUS)

- Normalisierung hängt von der Semantik der Daten ab (als FDs ausgedrückt); andere Interpretation führt zu anderen Schemata
  - Beispiel: Status als Zuverlässigkeit eines Lieferanten interpretiert macht die FD CITY→ STATUS sinnlos. Ohne diese FD wäre G bereits in 3NF

Boyce/Codd-Normalform (BCNF)

- Definition der 3NF hat gewisse Schwächen bei Relationen mit mehreren Schlüsselkandidaten, wenn die Schlüsselkandidaten
  - zusammengesetzt sind und
  - sich überlappen
- „Each field must represent a fact about the key, the whole key and nothing but the key“ [Bill Kent, HP Palo Alto]

  PRÜFUNG (PNR, MATNR, FACH, NOTE)

  PRIMARY KEY (PNR, MATNR)

  UNIQUE (FACH, MATNR)
- Definition BCNF:
  Ein Relationenschema R ist in BCNF, wenn jeder Determinant ein Schlüsselkandidat von R ist.
  Ein Attribut (oder eine Gruppe von Attributen), von dem andere voll funktional abhängen, heißt Determinant.

Überführung in BCNF

- Im Relationenschema F: (S#, SNAME, STATUS, CITY, QTY) sind S#, CITY und (S#,P#) Determinante
- Das Relationenschema G: (S#, SNAME, CITY, STATUS) ist nicht in BCNF, da der Determinant CITY nicht ein Schlüsselkandidat für die gesamte Relation ist
- Dagegen ist H: (S#, P#, QTY) in BCNF, da (S#,P#) der einzige Schlüsselkandidat ist
- Aufbrechen von G in
  - I: (S#, SNAME, CITY) und
  - J: (CITY, STATUS)
    erzeugt Relationenschemas in 3NF als auch BCNF
- Zusatzdefinition der Attributsemantik im Beispiel:
  - SNAME ist einzig für Lieferanten
  - STATUS bedeutet Zuverlässigkeit des Lieferanten und ist nicht vom Standort des Lieferanten abhängig
  - Darauf folgt:

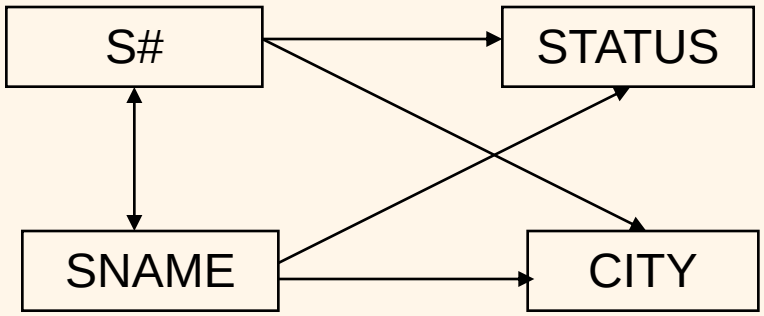
    SNAME $\rightarrow$ S#

    SNAME $\rightarrow$ STATUS

    SNAME $\rightarrow$ CITY

    und CITY $\rightarrow$ STATUS gilt nicht mehr

Überführung in BCNF (Forts.)



Abhängigkeitsdiagramm

- S# und SNAME als Primärschlüssel austauschbar, d.h. SNAME ist ein echter Schlüsselkandidat
- Nach Definition ist K: (S#, SNAME, STATUS, CITY) in BCNF
  Beide Determinanten S# und SNAME sind Schlüsselkandidaten
- Mit gleichen FDs, Relationenschema L: (S#, SNAME, P#, QTY)
  - Ist L in 3NF?
    Ja, da keine transitive FD existiert und 3NF kein volle Abhängigkeit verlangt
  - Ist L in BCNF?
    Nein! Determinanten: S#, SNAME, (S#, P#), (SNAME, P#)
    S# und SNAME sind Determinanten, da sie gegenseitig funktional abhängig sind
    S# oder SNAME keine Schlüssel für gesamte Relation $\Rightarrow$ L nicht in BCNF
  - Aufbrechen in
    M: (S#, SNAME)
    N: (S#, P#, QTY) oder O: (SNAME, P#, QTY)

Projektionskriterien

- Relationenschema G: (S#, SNAME, CITY, STATUS)
  mit FD CITY $\rightarrow$ STATUS kann auf zwei Arten zerlegt werden, jeweils gültig unter BCNF:
  - I: (S#, SNAME, CITY), J: (CITY, STATUS)
  - P: (S#, SNAME, CITY), Q: (S#; STATUS)
- Unabhängige Projektion
  - Projektion in I und J ist vorzuziehen, da I und J unabhängig voneinander verändert werden können
  - Projektion in P und Q erfordert hingegen gleichzeitiges Ändern von P.CITY und Q.STATUS
    - CITY $\rightarrow$ STATUS ist in der Projektion P,Q zum *inter*relationalen Constraint geworden

Zerlegung eines Relationenschemas

- Gegeben sei ein Relationenschema R mit Attributen *A1 ... An.*
- Eine *Dekomposition* von R bedeutet, R durch zwei oder mehr Relationenschemas zu ersetzen:
  - Jedes neue Relationenschema besteht aus einer Teilmenge von Attributen von R (keine Attribute, die nicht in R erscheinen) und
  - Jedes Attribut von R erscheint als ein Attribut in einem oder mehreren Relationenschemas
- Dekomposition von R bedeutet: Instanzen der erzeugten Relationenschemas werden gespeichert anstelle von Instanzen von R
- Beispiel:
  - Relationenschema Emp (E, R, W, H)
    - mit E = Angestellten-Nr., R = Einstufung, W = Stundenlohn, H = Wöchentliche Arbeitszeit
    - Funktionale Abhängigkeiten E $\rightarrow$ RWH und R $\rightarrow$ W
    - Somit ist Emp nicht in 3NF (Mehrfachspeicherung der Beziehung R-W), Ausweg: Dekomposition
    - Mögliche Zerlegung von ERWH: ERH und RW

Probleme bei Dekomposition

- Dekomposition sollte nur angewandt werden, wenn wirklich benötigt
- Die zu speichernde Information besteht aus ERWH-Tupeln. Entstehen beim Speichern der Projektion dieser Tupel in ERH und RW irgendwelche Probleme?
- Drei potentielle Probleme zu beachten:
  1. *Performance*: Einige Anfragen werden sehr teuer.
     - z.B. Wieviel verdient Jörg? (Gehalt = W*H, erfordert Join)
  2. *Verlustfreiheit*: Mit den Instanzen der zerlegten Relationen kann es passieren, daß wir die korrespondierende Instanz der Original-Relation nicht wieder rekonstruieren können.
     - Nicht in unserem Beispiel.
  3. *Abhängigkeitswahrung*: Die Prüfung einiger Abhängigkeiten kann einen Join der Instanzen der zerlegten Relationen erfordern.
     - Nicht in unserem Beispiel.
- *Tradeoff*: Abwägen dieser Probleme gegenüber redundanter Speicherung von Daten

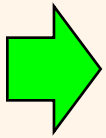
Verlustfreie Dekomposition

- Dekomposition von R in X and Y ist *verlustfrei (lossless-join)* in Bezug auf eine Menge von FDs F, wenn für jede Instanz r, die F erfüllt, gilt:
  - $\pi_X(r) \bowtie \pi_Y(r) = r$
- Generell gilt: $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$
  - Im allgemeinen gilt die Umkehrung nicht (Join von zerlegten Relationen produziert mehr Tupel als in der Original-Relation). Anderenfalls liegt verlustfreie Zerlegung vor.
- Definition läßt sich auf 3 oder mehr Relationen einfach verallgemeinern
- *Es ist wichtig, daß alle Dekompositionen, die zur Beseitigung von Redundanz eingeführt werden, verlustfrei sind - Vermeidet somit Problem (2)*

Verlustfreie Dekomposition (Forts.)

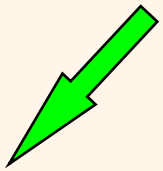
- Die Dekomposition von R in X und Y ist verlustfrei bezüglich F genau dann wenn die Hülle von F, F^+, folgende FDs enthält:
  - $X \cap Y \rightarrow X$, oder
  - $X \cap Y \rightarrow Y$
  - Interpretation: Gemeinsame Attribute von X und Y müssen Schlüssel für X oder Y sein (Bedingung im Beispiel Emp erfüllt, R ist Schlüssel in RW)
- 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)

A	B	C
1	2	3
4	5	6
7	2	8



A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8



A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Abhängigkeitsbewahrende Dekomposition

- Betrachte CSJDPQV mit
  C = Vertrags-ID (*Contract*), S = Lieferanten-ID (Supplier),
  J = Projekt-ID (*Job*), D = Abteilungs-ID (*Department*),
  P = Teil-ID (*Part*), Q = Menge (*Quantity*), V = Preis (*Value*)
  C ist Schlüssel, JP $\rightarrow$ C und SD $\rightarrow$ P
  - BCNF Dekomposition: CSJDQV und SDP
  - Problem: Prüfung von JP $\rightarrow$ C erfordert einen Join!
- Abhängigkeitsbewahrende Dekomposition (Intuitiv):
  - 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. *- Vermeidet Problem (3)*

- *Projektion einer Menge von FDs F*: Wenn R zerlegt wird in X, ... dann ist die Projektion von F auf X (bezeichnet als F_X) die Menge von FDs U $\rightarrow$ V in F^+ (*Hülle von F*) so daß U, V in X sind

Abhängigkeitsbewahrende Dekomposition (Forts.)

- Dekomposition von R in X und Y ist *abhängigkeitsbewahrend* wenn $(F_X \cup F_Y)^+ = F^+$
  - 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
  - $F^+ = F \cup \{ A \rightarrow C, B \rightarrow A, C \rightarrow B \}$
  - 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: $C \rightarrow A$
    $\Rightarrow$ Dekomposition ist abhängigkeitsbewahrend
  - Algorithmus ist exponentiell von der Größe von F abhängig
- Abhängigkeitsbewahrend impliziert keineswegs eine verlustfreie Zerlegung:
  - Beispiel: ABC, $A \rightarrow B$, zerlegt in AB und BC (nicht verlustfrei!)

Dekomposition in BCNF

- Betrachte Relationenschema R mit FDs F: Wenn durch $X \rightarrow Y$ die BCNF-Eigenschaft verletzt ist, zerlege R in R - Y and XY.
  - Wiederholte Anwendung dieser Idee resultiert in einer Sammlung von Relationen, mit den Eigenschaften BCNF/ verlustfreie Zerlegung (Algorithmus terminiert)
  - Beispiel: CSJDPQV, Schlüssel C, JP $\rightarrow$ C, SD $\rightarrow$ P, J $\rightarrow$ S
  - Zur Behandlung von SD $\rightarrow$ P, Zerlegung in SDP, CSJDQV.
  - Zur Behandlung von J $\rightarrow$ S, Zerlegung von CSJDQV in JS and CJDQV
- Im allgemeinen wird BCNF durch verschiedene Abhängigkeiten verletzt. Die Reihenfolge, wie diese behandelt werden, kann zu ganz unterschiedlichen Mengen von Relationen führen!

BCNF und Abhängigkeitsbewahrung

- Im allgemeinen gibt es keine abhängigkeitsbewahrende Dekomposition in BCNF
  - Beispiel: Relationenschema SBD (S=Segler, B=Boot, D=Tag)
    Es gilt: SB $\rightarrow$ D (ein Segler kann ein Boot höchstens einen Tag reservieren); D $\rightarrow$ B (an einem bestimmten Tag, kann nur ein Boot reserviert werden)
  - SBD nicht in BCNF (D ist kein Schlüssel), bei Dekomposition kann die Abhängigkeit SB $\rightarrow$ D nicht bewahrt werden
- Ähnliches Beispiel: Dekomposition von CSJDQV in SDP, JS und CJDQV ist nicht abhängigkeitsbewahrend (in Bezug auf die FDs JP $\rightarrow$ C, SD $\rightarrow$ P and J $\rightarrow$ S)
  - Es ist jedoch eine verlustfreie Dekomposition.
  - In diesem Fall ermöglicht das Hinzufügen von JPC zu der Menge der Relationen eine abhängigkeitsbewahrende Dekomposition
    - JPC Tupel werden nur gespeichert, um die FD JP $\rightarrow$ C überprüfen zu können (*Redundanz!*)

Dekomposition in 3NF

- Offenkundig kann der Algorithmus für eine verlustfreie Zerlegung in BCNF auch genutzt werden, um eine verlustfreie Zerlegung in 3NF zu erzielen (kann früher stoppen)
- Idee, um die Bewahrung der Abhängigkeiten zu sichern:
  - Wenn $X \rightarrow Y$ nicht erhalten wird, füge Relation XY hinzu
  - Problem ist, daß XY die 3NF verletzen kann!
    z.B. Hinzufügen von CJP, um $JP \rightarrow C$ zu ‘bewahren‘: Wenn es auch eine FD $J \rightarrow C$ gibt?
- Verfeinerung:
  - Anstelle einer gegebenen Menge von FDs F, nutze eine *minimale Hülle für F*.

Minimalabdeckung für eine Menge von FDs*

- *Minimalabdeckung* G für eine Menge von FDs F erfüllt folgende Bedingungen:
  1. Hülle F^+ = Hülle G^+
  2. Jede FD in G ist in der Form $X \rightarrow A$, wobei A ein einzelnes Attribut ist
  3. Wenn wir G modifizieren durch das Löschen einer FD oder durch das Löschen von Attributen aus einer FD in G, verändert sich die Hülle von G.
- Intuitiv: Jede FD in G wird benötigt und ist *"so klein wie möglich"*, um die gleiche Hülle wie F zu liefern
- Beispiel: $A \rightarrow B$, $ABCD \rightarrow E$, $EF \rightarrow GH$, $ACDF \rightarrow EG$ hat die folgende minimale Abdeckung:
  - $A \rightarrow B$, $ACD \rightarrow E$, $EF \rightarrow G$ und $EF \rightarrow H$
- Allgemeiner Algorithmus:
  1. Bringe FDs mit Hilfe der Ableitungsregeln in die Standard-Form (d.h. nur einzelne Attribute auf der rechten Seite)
  2. Minimiere linke Seite jeder FD (prüfe, welches Attribut gelöscht werden kann)
  3. Lösche redundante FDs

Zusammenfassung

- Wenn eine Relation in BCNF ist, ist sie frei von Redundanzen, die entdeckt werden können durch Nutzung von funktionalen Abhängigkeiten.
- Wenn alle Relationen in BCNF sind, hat man somit eine gute Heuristik.
- Wenn eine Relation nicht in BCNF ist, können wir versuchen, sie in eine Menge von BCNF-Relationen zu zerlegen.
  - Es muß beachtet werden, ob alle FDs gewahrt werden. Falls eine verlustfreie und abhängigkeitsbewahrende Dekomposition in die BCNF nicht möglich (oder unpraktisch wegen typischer Anfragen) ist, sollte eine Dekomposition in 3NF vorgezogen werden.
  - Dekompositionen sollten immer durchgeführt bzw. überprüft werden unter Performance-Gesichtssichtspunkten (☞ausführlich in Datenbanken 2).