

3. Normalization

Einführung

Anomalien in Datenbanksystemen

- Anomalien sind unerwünschte Nebeneffekte, die bei der Datenmanipulation (Einfügen, Ändern, Löschen) in schlecht entworfenen Datenbanken auftreten können. Sie führen zu Dateninkonsistenzen und Datenverlust.

Hauptquellen für Anomalien

- Redundanz:** Informationen sind mehrfach in der Datenbank gespeichert.
- Speicherplatzverschwendungen:** Durch redundante Daten oder unnötige Null-Werte (*NULL*) wird Speicherplatz ineffizient genutzt.
- Unvollständige Beziehungen:** Existenz von Entitäten ohne die dazugehörigen abhängigen Informationen, z.B. Professor:innen ohne Lehrveranstaltungen (LVAs) oder LVAs ohne zugeordnete Professor:innen.

courseOffers						
empID	teacher	rank	office	courseID	title	ects
2125	Socrates	C4	226	5041	Robotics	4
2125	Socrates	C4	226	5049	Ethics	2
2125	Socrates	C4	226	4052	Logic	4
2133	Curie	C3	52	5259	Chemistry	2
2137	Curie	C4	7	4630	Physics	4
⊥	⊥	⊥	⊥	5432	Economy	4
...

Arten von Anomalien

1. Änderungsanomalien (*Update Anomaly*)

- Problem:** Wenn eine Information, die mehrfach in der Datenbank vorkommt, an einer Stelle geändert wird, aber nicht an allen anderen Stellen, führt dies zu Inkonsistenzen.
- Beispiel:** Sokrates zieht von Raum 226 in Raum 338. Wenn diese Änderung nur für einige seiner Lehrveranstaltungen (aber nicht für alle) aktualisiert wird, sind die Daten inkonsistent.

2. Einfügeanomalien (*Insertion Anomaly*)

- Problem:** Eine neue LVA (Lehrveranstaltung) kann nicht eingefügt werden, ohne gleichzeitig Null-Werte (*NULL values*) für Attribute zu setzen, die eigentlich nicht unbekannt sein sollten (z.B. ein zugehöriger Professor:in).

- **Beispiel:** Ein neuer Kurs kann nicht erfasst werden, wenn der zugehörige Professor noch nicht existiert oder zugewiesen ist, obwohl der Kurs selbst schon bekannt ist.

3. Löschanomalien (*Deletion Anomaly*)

- **Problem:** Das Löschen einer bestimmten Information führt zum Verlust von weiteren, eigentlich unabhängigen Informationen.
- **Beispiel:** Wenn die letzte LVA von einem: einer Professor:in gelöscht wird, gehen alle Informationen über diese:n Professor:in (Name, ID, Rang, Büro) verloren.

Ziel des Datenbankentwurfs

- **Vermeidung von Redundanz:** Informationen sollen nur einmal gespeichert werden.
- **Vermeidung von Null-Werten (*NULL values*):** Minimierung von Attributen, die bei der Erfassung leer bleiben müssen.
- **Vermeidung von Anomalien:** Sicherstellen der Konsistenz und Integrität der Daten bei allen Operationen.
- **Abbildung von Beziehungen:** Alle Beziehungen zwischen Attributen müssen korrekt und eindeutig abgebildet sein.

Strategie zur Problemlösung

- **Transformation funktionaler Abhängigkeiten:** Alle gegebenen funktionalen Abhängigkeiten sollen in Schlüsselabhängigkeiten transformiert werden.
 - **Ziel:** Dabei dürfen keine semantischen Informationen (Bedeutung der Daten und ihrer Beziehungen) verloren gehen.
 - (Funktionale Abhängigkeiten beschreiben, wie Werte eines Attributs die Werte anderer Attribute bestimmen. Schlüsselabhängigkeiten bedeuten, dass die Abhängigkeit über einen Schlüssel läuft, was Redundanz reduziert)

Gute und schlechte Zerlegungen

Gute Zerlegung

Grundidee zur Vermeidung von Redundanz, Anomalien etc.

- Zerlege *courseOffers* in:
 - *course* (*courseID, title, ects, empID*)
 - *instructor* (*empID, teacher, rank, office*)

course			
courseID	title	ects	empID
5041	Robotics	4	2125
5049	Ethics	2	2125
4052	Logic	4	2125
5259	Chemistry	2	2133
4630	Physics	4	2137

instructor			
empID	teacher	rank	office
2125	Socrates	C4	226
2133	Curie	C4	36
2137	Curie	C4	7

$\pi_{empID, teacher, rank, office}(courseOffers)$

$\pi_{courseID, title, ects, empID}(courseOffers)$

courseOffers						
empID	teacher	rank	office	courseID	title	ects
2125	Socrates	C4	226	5041	Robotics	4
2125	Socrates	C4	226	5049	Ethics	2
2125	Socrates	C4	226	4052	Logic	4
2133	Curie	C3	52	5259	Chemistry	2
2137	Curie	C4	7	4630	Physics	4

$courseOffers =$
 $instructor \bowtie course$

Diese Zerlegung ist im Sinne der Normalisierung sinnvoll, da so Redundanzen vermieden werden.

Anforderungen an eine gute Zerlegung

- Alle Attribute des originalen Schemas müssen in der Zerlegung vorkommen:

$$R = R_1 \cup R_2 \cup \dots \cup R_n$$

- Das bedeutet, dass die Vereinigung aller Attribute der zerlegten Relationen R_i wieder alle Attribute der ursprünglichen Relation R ergeben muss. Keine Information darf verloren gehen.

- Zerlegung mit verlustfreiem Join:

Für Relationen R mit Schema R gilt:

$$R = \pi_{R_1}(R) \bowtie \pi_{R_2}(R)$$

- Wenn die ursprüngliche Relation R durch den *natürlichen Join* (\bowtie) ihrer Projektionen (π) auf die zerlegten Schemata R_1 und R_2 wiederhergestellt werden kann, spricht man von einem verlustfreien Join. Dies ist entscheidend, um die ursprünglichen Daten exakt wiederherstellen zu können und somit keine Informationen durch die Zerlegung zu verlieren.

Schlechte Zerlegung

course			
courseID	title	ects	teacher
5041	Robotics	4	Socrates
5049	Ethics	2	Socrates
4052	Logic	4	Socrates
5259	Chemistry	2	Curie
4630	Physics	4	Curie

instructor			
empID	teacher	rank	office
2125	Socrates	C4	226
2133	Curie	C4	36
2137	Curie	C4	7

$\pi_{courseID, title, ects, teacher}(courseOffers) \ \pi_{empID, teacher, rank, office}(courseOffers)$

courseOffers						
empID	teacher	rank	office	courseID	title	ects
2125	Socrates	C4	226	5041	Robotics	4
2125	Socrates	C4	226	5049	Ethics	2
2125	Socrates	C4	226	4052	Logic	4
2133	Curie	C3	36	5259	Chemistry	2
2133	Curie	C3	36	4630	Physics	4
2137	Curie	C4	7	5259	Chemistry	2
2137	Curie	C4	7	4630	Physics	4
...

$courseOffers =$
 $instructor \bowtie course$

Funktionale Abhängigkeiten

Definition

Symbole

- Schema $\mathcal{R} = \{A, B, C, D\}$ (Menge aller Attribute, die in einer Relation vorkommen können)
- Ausprägung/Instanz R (Eine konkrete Tabelle/Beziehung, die dem Schema \mathcal{R} entspricht)
- Attributmengen $\alpha \subseteq \mathcal{R}$ und $\beta \subseteq \mathcal{R}$ (Teilmengen der Attribute aus dem Schema)

Definition einer funktionalen Abhängigkeit

Eine **funktionale Abhängigkeit** $\alpha \rightarrow \beta$ in R liegt genau dann vor, wenn für alle legalen Ausprägungen R von \mathcal{R} gilt:

$$\forall r, s \in R : r. \alpha = s. \alpha \Rightarrow r. \beta = s. \beta$$

- Das bedeutet: Wenn zwei Tupel (Zeilen) r und s in einer Relation R in ihren Werten für die Attributmenge α übereinstimmen ($r. \alpha = s. \alpha$), dann müssen sie auch in ihren Werten für die Attributmenge β übereinstimmen ($r. \beta = s. \beta$).
- Die α -Werte identifizieren die β -Werte eindeutig.
- Bzw.: Die α -Werte bestimmen die β -Werte funktional.

Triviale funktionale Abhängigkeit

Eine funktionale Abhängigkeit $\alpha \rightarrow \beta$ wird **trivial** genannt, wenn $\beta \subseteq \alpha$.

- **Beispiel:** Wenn wir die Abhängigkeit $\{A, B\} \rightarrow A$ betrachten, ist dies trivial, da A bereits Teil von $\{A, B\}$ ist. Dies ist immer wahr und liefert keine neue Information über die Datenintegrität.

Beispiele

Postleitzahl und Stadt

Wenn ich die Postleitzahl kenne, dann kenne ich die Stadt.

- Die Postleitzahl 1040 gibt mir genau eine Stadt: Wien
- Die Postleitzahl 3400 gibt mir nicht {Wien, Klosterneuburg} (Dies würde bedeuten, dass 3400 mal Wien und mal Klosterneuburg sein könnte, was der Definition der funktionalen Abhängigkeit widerspricht.)
- **Notation:** {Postleitzahl} \rightarrow {Stadt} (Die Postleitzahl bestimmt die Stadt eindeutig.)

Sozialversicherungsnummer und Name

Die Sozialversicherungsnummer (VSNR) gibt mir einen Namen.

- VSNR 8476120478 gibt mir Sean Connery
- **Notation:** $VSNR \rightarrow Vorname, Nachname$ (Die Sozialversicherungsnummer bestimmt Vorname und Nachname eindeutig.)

Welche funktionalen Abhängigkeiten (FDs, Functional Dependencies) sind erfüllt?

R			
A	B	C	D
a4	b2	c4	d3
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c3	d2
a3	b2	c4	d3

Analyse der funktionalen Abhängigkeiten basierend auf der gegebenen Tabelle (Instanz) R:

- $\{A\} \rightarrow \{A\}$: **ja** (Trivial, da A in A enthalten ist.)
- $\{A\} \rightarrow \{B\}$: **ja** (Wenn A gleich, dann B gleich. Prüfen: (a1,b1) und (a1,b1) sind identisch in A und B.)
- $\{A\} \rightarrow \{C\}$: **ja** (Wenn A gleich, dann C gleich. Prüfen: (a1,c1) und (a1,c1) sind identisch in A und C.)
- $\{A\} \rightarrow \{D\}$: **nein** (Prüfen: Für $A = a1$ gibt es $D = d1$ (Zeile 2) und $D = d2$ (Zeile 3). Der A-Wert $a1$ bestimmt den D-Wert nicht eindeutig.)
- $\{A\} \rightarrow \{C, D\}$: **nein** (Wenn A gleich, dann C und D gleich. Da $A \rightarrow D$ nicht gilt, kann auch $A \rightarrow \{C, D\}$ nicht gelten.)
- $\{B\} \rightarrow \{D\}$: **nein** (Prüfen: Für $B = b2$ gibt es $D = d3$ (Zeile 1) und $D = d2$ (Zeile 4). Der B-Wert $b2$ bestimmt den D-Wert nicht eindeutig.)
- $\{B\} \rightarrow \{A\}$: **nein** (Prüfen: Für $B = b2$ gibt es $A = a4$ (Zeile 1) und $A = a2$ (Zeile 4). Der B-Wert $b2$ bestimmt den A-Wert nicht eindeutig.)
- $\{C, D\} \rightarrow \{A\}$: **nein** (Prüfen: Für $\{C, D\} = \{c4, d3\}$ gibt es $A = a4$ (Zeile 1) und $A = a3$ (Zeile 5). Die Kombination der C- und D-Werte $\{c4, d3\}$ bestimmt den A-Wert nicht eindeutig.)
- $\{C, D\} \rightarrow \{B\}$: **ja** (Prüfen:
 - $(c4, d3) \rightarrow b2$ (Zeile 1)
 - $(c1, d1) \rightarrow b1$ (Zeile 2)
 - $(c1, d2) \rightarrow b1$ (Zeile 3)
 - $(c3, d2) \rightarrow b2$ (Zeile 4)

- $(c4, d3) \rightarrow b2$ (Zeile 5)
In allen Fällen, wenn $\{C, D\}$ gleich ist, ist auch B gleich. *Beispiel: Für $\{c4, d3\}$ kommt nur $b2$ als B-Wert vor.)*
- **Vereinfachte Notation:** $CD \rightarrow B$

Semantische Konsistenzbedingungen

Funktionale Abhängigkeiten sind **semantische Konsistenzbedingungen**, die sich aus der **jeweiligen Anwendungssemantik** und **nicht aus der aktuellen Ausprägung einer Relation** ergeben. Sie müssen zu jedem (gültigen) Datenbankzustand eingehalten werden.

- **Wichtig:** Eine funktionale Abhängigkeit wird *nicht* dadurch definiert, dass sie in einer *einzelnen* momentanen Dateninstanz gilt. Sie muss vielmehr eine **grundlegende Regel** sein, die für alle *denkbaren und gültigen* Datenzustände der Datenbank zutrifft, basierend auf der Bedeutung der Daten in der realen Welt. Die Überprüfung an einer Instanz dient lediglich zur Illustration oder zum Finden von Verletzungen.

Schlüssel

Superschlüssel

- $\alpha \subseteq R$ ist ein **Superschlüssel** wenn $\alpha \rightarrow R$.
 - D.h., α bestimmt alle anderen Attributwerte in der Relation.
 - Ein Superschlüssel ist also eine Attributmenge, die die gesamte Relation eindeutig identifizieren kann.
- Die Menge aller Attribute bildet einen Superschlüssel: $R \rightarrow R$.
 - Wenn man alle Attribute kennt, kann man natürlich auch die gesamte Relation eindeutig identifizieren.
- Superschlüssel sind **nicht notwendigerweise minimal!**
 - Ein Superschlüssel kann also überflüssige Attribute enthalten, die nicht zwingend zur Eindeutigkeit der Identifizierung notwendig wären.

Volle funktionale Abhängigkeit

β ist **voll funktional abhängig** von α wenn:

- $\alpha \rightarrow \beta$ (Es besteht eine funktionale Abhängigkeit von α zu β .)
- und α kann nicht mehr verkleinert (=linksreduziert) werden, d.h.

$$\forall A \in \alpha : (\alpha - \{A\}) \not\rightarrow \beta$$

- Das bedeutet, dass kein Attribut aus α entfernt werden kann, ohne dass die funktionale Abhängigkeit zu β verloren geht. α ist also die **minimale** Menge von Attributen, die β bestimmt.

Schlüsselkandidat

$\alpha \subseteq \mathcal{R}$ ist ein **Schlüsselkandidat**, wenn \mathcal{R} **voll funktional abhängig** von α ist.

- Ein Schlüsselkandidat ist also ein *minimaler* Superschlüssel. Das bedeutet, er ist ein Superschlüssel, und wenn man irgendein Attribut aus ihm entfernt, ist er kein Superschlüssel mehr.
- Ein Schlüsselkandidat wird als **Primärschlüssel** ausgewählt!
 - Aus der Menge aller Schlüsselkandidaten wird in der Regel einer als Primärschlüssel für eine Relation bestimmt. Dieser Primärschlüssel dient dann zur eindeutigen Identifikation der Tupel in der Relation.

Ableitung funktionaler Abhängigkeiten

Bestimmung funktionaler Abhängigkeiten

Hier sind einige Beispiele für funktionale Abhängigkeiten und wie daraus weitere abgeleitet werden können:

- $\{empID\} \rightarrow \{empID, name, rank, office, city, street, zip, dialCode, region, inhabitants, government\}$
 - Dies ist ein **Schlüsselkandidat**. (Da die `empID` alle anderen Attribute eindeutig bestimmt, ist es ein Superschlüssel. Wenn es sich um einen minimalen Superschlüssel handelt, ist es ein Schlüsselkandidat.)
 - Dies impliziert, dass eine Mitarbeiter-ID ausreicht, um alle Informationen über einen Mitarbeiter und eventuell zugehörige Standortdaten (Stadt, Region etc.) eindeutig zu identifizieren.
- $\{city, region\} \rightarrow \{inhabitants, dialCode\}$
 - Die Kombination aus Stadt und Region bestimmt eindeutig die Einwohnerzahl und die Vorwahl.
- $\{zip\} \rightarrow \{region, city, inhabitants\}$
 - Eine Postleitzahl bestimmt eindeutig die Region, die Stadt und die Einwohnerzahl.
- $\{region, city, street\} \rightarrow \{zip\}$
 - Die Kombination aus Region, Stadt und Straße bestimmt eindeutig die Postleitzahl. (Dies ist die Umkehrung der obigen Abhängigkeit und zeigt, dass die PLZ durch detailliertere Adressinformationen bestimmt werden kann.)
- $\{region\} \rightarrow \{government\}$
 - Eine Region bestimmt eindeutig die jeweilige Regierung (z.B. Bundeslandregierung).
- $\{office\} \rightarrow \{empID\}$
 - Ein Büro (als Entität oder spezifischer Standort) bestimmt eindeutig eine Mitarbeiter-ID. (Dies könnte bedeuten, dass ein Büro nur von einem spezifischen Mitarbeiter geleitet wird, oder dass die Büro-ID eine eindeutige Beziehung zu einer Mitarbeiter-ID hat.)

Davon können weitere abgeleitet werden

Basierend auf den oben genannten funktionalen Abhängigkeiten können durch Inferenzregeln (z.B. Transitivität) weitere Abhängigkeiten abgeleitet werden:

- $\{office\} \rightarrow \{empID, name, rank, office, city, street, zip, dialCode, inhabitants, government\}$
 - **Ableitung:** Wir wissen, dass $\{office\} \rightarrow \{empID\}$ gilt.
 - Wir wissen auch, dass $\{empID\} \rightarrow \{empID, name, rank, office, city, street, zip, dialCode, region, inhabitants, g\}$ gilt.
 - Durch die Transitivitätsregel ($A \rightarrow B$ und $B \rightarrow C \implies A \rightarrow C$) können wir ableiten, dass $\{office\}$ alle Attribute bestimmt, die von $\{empID\}$ bestimmt werden.
 - (Anmerkung: Das Attribut 'city' ist hier zweimal aufgeführt, dies ist wahrscheinlich ein Tippfehler in den Originalfolien.)
- $\{zip\} \rightarrow \{government\}$
 - **Ableitung:** Wir wissen, dass $\{zip\} \rightarrow \{region, city, inhabitants\}$ gilt.
 - Wir wissen auch, dass $\{region\} \rightarrow \{government\}$ gilt.
 - Da $\{zip\}$ die `region` bestimmt, und die `region` die `government` bestimmt, folgt durch Transitivität, dass $\{zip\} \rightarrow \{government\}$.

Herleitung weiterer FDs

- Aus einer Menge von FDs F sind **weitere FDs herleitbar**.
 - F^+ beinhaltet **alle FDs**, die aus F abgeleitet werden können, d.h., alle FDs, die von FDs in F **logisch impliziert** werden.
 - F^+ wird **Hülle (closure)** von F genannt.
 - **Inferenzregeln** (Armstrong-Axiome) beschreiben die Herleitung.
-

Die Armstrong-Axiome

Seien $\alpha, \beta, \gamma, \delta$ Teilmengen der Attribute aus R .

- **Reflexivität:**
 - Falls $\beta \subseteq \alpha$, dann $\alpha \rightarrow \beta$.
 - Insbesondere: $\alpha \rightarrow \alpha$ (Eine Menge von Attributen determiniert sich selbst).
- **Erweiterung/Verstärkung:**
 - Falls $\alpha \rightarrow \beta$, dann $\alpha\gamma \rightarrow \beta\gamma$. (Wenn α β determiniert, dann determiniert α zusammen mit weiteren Attributen γ auch β zusammen mit γ).
- **Transitivität:**
 - Falls $\alpha \rightarrow \beta$ und $\beta \rightarrow \gamma$, dann $\alpha \rightarrow \gamma$. (Wenn α β determiniert und β γ determiniert, dann determiniert α auch γ).
- Die Armstrong-Axiome sind **korrekt und vollständig**:

- Sie sind **korrekt** in dem Sinne, dass sich mit ihnen nur korrekte funktionale Abhängigkeiten herleiten lassen.
- Sie sind **vollständig** in dem Sinne, dass sich mit ihnen **alle möglichen FDs (F^+) von F** herleiten lassen.

Weitere Ableitungsregeln

Nicht zwingend notwendig, aber oftmals komfortabel für Herleitungen.

- **Vereinigung:**
 - Wenn $\alpha \rightarrow \beta$ und $\alpha \rightarrow \gamma$, dann auch $\alpha \rightarrow \beta\gamma$. (Wenn α sowohl β als auch γ determiniert, dann determiniert α auch die Vereinigung von β und γ).
- **Dekomposition:**
 - Wenn $\alpha \rightarrow \beta\gamma$, dann $\alpha \rightarrow \beta$ und $\alpha \rightarrow \gamma$. (Wenn α die Vereinigung von β und γ determiniert, dann determiniert α auch β und γ einzeln).
- **Pseudotransitivität:**
 - Wenn $\alpha \rightarrow \beta$ und $\gamma\beta \rightarrow \delta$, dann auch $\alpha\gamma \rightarrow \delta$.

Beispiel

Gegeben seien die folgenden FDs F , leite mithilfe der Armstrong-Axiome weitere her.

- $A \rightarrow BC$
- $CD \rightarrow E$
- $B \rightarrow D$
- $E \rightarrow A$

Hergeleitete FDs

- $E \rightarrow A$ und $A \rightarrow BC$, dann $E \rightarrow BC$ (Transitivität)
- $B \rightarrow D$, dann $CB \rightarrow CD$ (Erweiterung/Verstärkung)
- $CB \rightarrow CD$ und $CD \rightarrow E$, dann $CB \rightarrow E$ (Transitivität)

Algorithmus zur Bestimmung der Attributhülle

Algorithmus `attrClosure(F , α)`

Input:

- Eine Menge von FDs F
- Eine Menge von Attributen $\alpha \subseteq R$ (Teilmenge aller Attribute des Schemas R)

Algorithmus-Schritte:

Algorithmus attrClosure(F , α):

```

result =  $\alpha$ 
repeat
  for each  $\beta \rightarrow \gamma$  in  $F$  do
    if  $\beta \subseteq \text{result}$  then
      result = result  $\cup \gamma$ 
    end if
  end for
until result changes no more

```

Korrektheit und Terminierung des Algorithmus

- Der Algorithmus berechnet die Attributhülle von α korrekt.
 - Bei Terminierung: Alle Teile der Attributhülle von α sind in `result`.
- Der Algorithmus terminiert (da die Menge der Attribute endlich ist und `result` bei jeder Änderung wächst).

Anwendungen des Algorithmus zur Attributhülle

- Testen, ob eine funktionale Abhängigkeit $\alpha \rightarrow \beta$ gilt.
 - Berechne α^+ (die Attributhülle von α) mit `attrClosure(F, α)`.
 - Wenn $\beta \subseteq \alpha^+$ (d.h. alle Attribute von β in der Attributhülle von α enthalten sind), dann gilt die funktionale Abhängigkeit $\alpha \rightarrow \beta$.
- Testen, ob eine Menge von Attributen $\kappa \subseteq R$ ein Superschlüssel ist.
 - Ein *Superschlüssel* ist eine Menge von Attributen, die eindeutig alle anderen Attribute einer Relation bestimmt.
 - Rufe `attrClosure(F, κ)` auf, um κ^+ (die Attributhülle von κ) zu erhalten.
 - Falls $\kappa^+ = R$ (d.h. die Attributhülle von κ alle Attribute des Schemas R enthält), dann ist κ ein Superschlüssel von R .

Beispiel

Relation R : $\{[A, B, C, D]\}$ $\alpha = \{A, D\}$

Menge an FDs: $F = \{A \rightarrow B, A \rightarrow C, CD \rightarrow A\}$

$\alpha^+ = result = \{$

A, D, B, C, da $A \rightarrow C$

}

Is $\{ A, D \}$ a super key? ja

Is $\{ A \}$ a super key? nein

Is $\{ D \}$ a super key? nein

Kanonische Überdeckung

Äquivalente FD-Mengen

- Zwei Mengen von FDs F und G werden als *äquivalent* angesehen ($F \equiv G$), wenn deren Hüllen gleich sind, d.h., $F^+ = G^+$.
- Beide Mengen erlauben die *gleiche Menge von FDs*.

Beobachtung:

- F^+ kann *riesig* sein.
- Viele *redundante Abhängigkeiten* (d.h. Abhängigkeiten, die aus anderen in F abgeleitet werden können).
- In der Praxis unübersichtlich.

Ziel:

- Finde die *kleinstmögliche Menge* F_c für F , so dass $F_c^+ = F^+$.
- Es gibt möglicherweise *mehrere minimale Mengen*!

Kanonische Überdeckung F_c (minimale Überdeckung)

Eine *minimale Überdeckung* F_c ist eine *kanonische Darstellung* einer Menge F von funktionalen Abhängigkeiten.

Eigenschaften:

1. $F_c \equiv F$, also $F_c^+ = F^+$ (Äquivalenz, falls die Hüllen gleich sind).
 2. In F_c existieren *keine FDs* $\alpha \rightarrow \beta$, bei denen α oder β überflüssige Attribute enthalten.
- D.h. es muss gelten:

- (a) $\forall A \in \alpha: (F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\} \not\equiv F_c$ (Das Entfernen eines Attributs aus der linken Seite oder das Ersetzen von α durch $\alpha - A$ verändert die Hülle von F_c)
- (b) $\forall B \in \beta: (F_c - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - B)\} \not\equiv F_c$ (Das Entfernen eines Attributs aus der rechten Seite oder das Ersetzen von β durch $\beta - B$ verändert die Hülle von F_c)
- **Überprüfung mit Hilfe der Attributhülle, ob $A \in \alpha$ in $\alpha \rightarrow \beta$ überflüssig ist:**
 - $A \in \alpha$ ist überflüssig, wenn $\beta \subseteq \text{attrClosure}((F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}, \alpha)$ (d.h., wenn β auch ohne A auf der linken Seite in Kombination mit den restlichen FDs abgeleitet werden kann).
- **Überprüfung mithilfe der Attributhülle, ob $B \in \beta$ in $\alpha \rightarrow \beta$ überflüssig ist:**
 - $B \in \beta$ ist überflüssig, wenn $B \in \text{attrClosure}((F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - B)\}, \alpha)$ (d.h., wenn B auch dann abgeleitet werden kann, wenn es nicht auf der rechten Seite von $\alpha \rightarrow \beta$ steht, sondern aus den verbleibenden FDs in F und der modifizierten FD $\alpha \rightarrow (\beta - B)$).

3. Jede linke Seite einer funktionalen Abhängigkeit in F_c ist **einzigartig**.

- Durch Anwendung der **Vereinigungsregel** wird $(\alpha \rightarrow \beta)$ und $(\alpha \rightarrow \gamma)$ durch $\alpha \rightarrow \beta\gamma$ ersetzt. (Dies konsolidiert alle FDs mit gleicher linker Seite).

Algorithmus minimale Überdeckung

Hauptschritte:

1. **Führe FDs mit der gleichen linken Seite zusammen:**

- $\alpha \rightarrow \beta_1, \dots, \alpha \rightarrow \beta_n$ wird zu $\alpha \rightarrow (\beta_1 \cup \dots \cup \beta_n)$ (Anwendung der Vereinigungsregel, um die linke Seite einzigartig zu machen).

2. **Für jede FD $(\alpha \rightarrow \beta) \in F$ führe Linksreduktion durch:**

- Überprüfe für jedes $A \in \alpha$, ob A überflüssig ist, d.h., ob $\beta \subseteq \text{attrClosure}(F, \alpha - A)$.
 - (**Erklärung:** A ist überflüssig, wenn die Attribute auf der rechten Seite β immer noch aus α (ohne A) abgeleitet werden können, unter Berücksichtigung aller anderen FDs in F).
- Wenn ja, entferne A , indem $\alpha \rightarrow \beta$ durch $(\alpha - A) \rightarrow \beta$ ersetzt wird.

3. **Für jede FD $(\alpha \rightarrow \beta) \in F$ führe Rechtsreduktion durch:**

- Überprüfe für jedes $B \in \beta$, ob B überflüssig ist, d.h., ob $B \in \text{attrClosure}((F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - B)\}, \alpha)$.
 - (**Erklärung:** B ist überflüssig, wenn B auch dann in der Attributhülle von α ist, wenn die FD $\alpha \rightarrow \beta$ durch $\alpha \rightarrow (\beta - B)$ (also ohne B auf der rechten Seite) ersetzt wird. Dies bedeutet, dass B durch andere FDs impliziert wird oder bereits in α enthalten ist.)
- Wenn ja, entferne B , indem $\alpha \rightarrow \beta$ durch $\alpha \rightarrow (\beta - B)$ ersetzt wird.

4. **Entferne FDs der Form $\alpha \rightarrow \emptyset$** (sind möglicherweise in Schritt 3 entstanden, wenn alle Attribute auf der rechten Seite als überflüssig entfernt wurden).

5. **Gehe zu Schritt 1** (Wiederhole die Schritte, da eine Änderung in einem Schritt Auswirkungen auf die Überflüssigkeit in einem anderen haben kann).

- Im Allgemeinen ist eine Runde (Schritte 1-4) plus ein weiteres Mal Schritt 1 genug.

Beispiel

Initiale Menge von Abhängigkeiten:

$$F = \{A \rightarrow B, B \rightarrow C, AB \rightarrow C\}$$

Momentane Menge von Abhängigkeiten:

$$F_c = \{A \rightarrow B, B \rightarrow C\}$$

- Schritt 1: Einzigartige linke Seiten
Keine Zusammenführung in diesem Beispiel notwendig
- Schritt 2: Linksreduktion
Ersetzte $AB \rightarrow C$ durch $A \rightarrow C$
- Schritt 3: Rechtsreduktion
Ersetzte $A \rightarrow C$ durch $A \rightarrow \emptyset$
- Schritt 4: Entferne Regeln der Form $\alpha \rightarrow \emptyset$
Entferne $A \rightarrow \emptyset$

Normalisierung durch Zerlegung der Relation

- Zerlege ein Relationenschema R in mehrere Relationsschemata R_1, \dots, R_m , um Probleme des originalen Entwurfs zu beheben.

Normalformen

- Normalformen beschreiben die Güte des Entwurfs.
- **1NF, 2NF, 3NF, BCNF, 4NF, ...**
- Sie verbieten bestimmte funktionale Abhängigkeiten in einer Relation, um **Redundanz**, **Null-Werte** und **Anomalien** zu verhindern.
- Gute ER-Modelle führen typischerweise direkt zur **3NF** (oder höhere NF).
- Normalisierung verhindert Probleme, welche durch funktionale Abhängigkeiten zwischen den Attributen eines Entitytyps ausgelöst werden.

Gültige und verlustlose Zerlegungen

Eine Zerlegung ist **gültig**, falls $R = R_1 \cup R_2$, d.h., wenn alle Attribute aus R in der Zerlegung enthalten bleiben.

- $R_1 := \pi_{R_1}(R)$
- $R_2 := \pi_{R_2}(R)$

Eine Zerlegung von R in R_1 und R_2 ist **verlustlos**, wenn Folgendes für alle möglichen Ausprägungen R von R gilt:

$$R = R_1 \bowtie R_2$$

Die in der ursprünglichen Ausprägung R des Schemas R enthaltenen Informationen müssen aus den Ausprägungen R_1, \dots, R_n der neuen Schemata R_1, \dots, R_n durch einen natürlichen Join rekonstruierbar sein.

Verlustlose Zerlegung

Formale Charakterisierung verlustloser Zerlegungen

Gegeben:

- Eine Zerlegung von R in R_1 und R_2
- F_R ist die Menge der Funktionalen Abhängigkeiten (FDs) in R

Eine Zerlegung ist **verlustlos**, wenn mindestens eine der folgenden FDs herleitbar ist:

- $(R_1 \cap R_2) \rightarrow R_1 \in F_R^+$ (die gemeinsamen Attribute bestimmen R_1)
- oder

- $(R_1 \cap R_2) \rightarrow R_2 \in F_R^+$ (die gemeinsamen Attribute bestimmen R_2)

Beispielzerlegung

beerDrinkers		
pub	guest	beer
Kowalski	Kemper	pils
Kowalski	Eickler	wheat beer
Innsteg	Kemper	wheat beer

zerlegt in

customer		drinks	
pub	guest	guest	beer
Kowalski	Kemper	Kemper	pils
Kowalski	Eickler	Eickler	wheat beer
Innsteg	Kemper	Kemper	wheat beer

Rekonstruktion: $R_1 \bowtie R_2$

beerDrinkers		
pub	guest	beer
Kowalski	Kemper	pils
Kowalski	Kemper	wheat beer
Kowalski	Eickler	wheat beer
Innsteg	Kemper	pils
Innsteg	Kemper	wheat beer

- Die Beziehung zwischen guest, pub und beer ging verloren.
- Die Verletzung der Verlustlosigkeit kann manchmal auch bedeuten, dass bei der Wiederherstellung zusätzliche Tupel entstehen.
- Also keine verlustlose Zerlegung.

Abhängigkeitsbewahrung

Zweites Kriterium für eine gute Zerlegung

Die für R geltenden Abhängigkeiten müssen in den neuen Schemata R_1, \dots, R_n verifizierbar sein.

- Wir können alle Abhängigkeiten lokal in R_1, \dots, R_n überprüfen.
- Wir vermeiden, dass wir den Join $R_1 \bowtie \dots \bowtie R_n$ berechnen müssen, um überprüfen zu können, ob eine FD verletzt wird.

Eine Zerlegung bewahrt Abhängigkeiten, wenn

$F_R^+ = (F_{R_1} \cup \dots \cup F_{R_n})^+$ (Die Abschlussmenge der FDs im Originalschema ist gleich der Abschlussmenge der vereinigten FDs der Teilschemata).

F_{R_i} sind die funktionalen Abhängigkeiten, die sich über R_i effizient überprüfen lassen.

Überprüfen, ob eine Zerlegung Abhängigkeiten bewahrt, ohne F^+ berechnen zu müssen:

- ① Können FDs effizient über dem zerlegtem Schema getestet werden?
 - Weise FDs den Relationen zu, welche alle involvierten Attribute enthalten.
Falls ja: die Zerlegung bewahrt Abhängigkeiten
- ② Falls nicht: verwende modifizierte Version von attrClosure um jede $\alpha \rightarrow \beta$ in F zu testen
Falls $\beta \subseteq result$, dann ist die FD bewahrt
Wenn alle FDs bewahrt werden, dann bewahrt die Zerlegung Abhängigkeiten

$result = \alpha$

repeat

```
for each  $R_i$  in the decomposition do
   $t = (result \cap R_i)^+ \cap R_i$ 
   $result = result \cup t$ 
end for
```

until result changes no more

Beispiele

zipCatalog: {[street, city, region, zip]}

FDs

- {zip} \rightarrow {city, region}
- {street, city, region} \rightarrow {zip}

Zerlegung

- streets: {[zip, street]}
- cities: {[zip, city, region]}

Ist diese Zerlegung verlustlos?
Bewahrt diese Zerlegung Abhängigkeiten?

zipCatalog			
city	region	street	zip
Frankfurt	Hesse	Goethestraße	60313
Frankfurt	Hesse	Galgenstraße	60437
Frankfurt	Brandenburg	Goethestraße	15234

 $\pi_{zip, street}$ $\pi_{city, region, zip}$

street	
zip	street
15234	Goethestraße
60313	Goethestraße
60437	Galgenstraße

cities		
city	region	zip
Frankfurt	Hesse	60313
Frankfurt	Hesse	60437
Frankfurt	Brandenburg	15234

Die Zerlegung bewahrt die Abhängigkeiten **nicht**:

$\{street, city, region\} \rightarrow \{zip\}$ kann nicht über der Zerlegung überprüft werden und wird nicht durch andere FDs garantiert.

Gegeben

- $R = (A, B, C)$
- $F = \{A \rightarrow B, B \rightarrow C\}$
- $R_1 = (A, B), R_2 = (B, C)$

Ist das eine verlustlose Zerlegung?

Hinweis (gemeinsame Attribute sind Superschlüssel in einer der Relationen):

- $(R_1 \cap R_2) \rightarrow R_1 \in F_R^+$
- $(R_1 \cap R_2) \rightarrow R_2 \in F_R^+$

JA:

- $R_1 \cap R_2 = \{B\}$ und $B \rightarrow BC$

Bewahrt diese Zerlegung Abhängigkeiten?

Hinweis (wir können alle Abhängigkeiten lokal überprüfen):

$$(F_1 \cup F_2)^+ = F^+$$

JA:

- $F_1 = \{A \rightarrow B, A \rightarrow AB\} \cup \{\text{viele triviale Abhängigkeiten}\}$
- $F_2 = \{B \rightarrow C, B \rightarrow BC\} \cup \{\text{viele triviale Abhängigkeiten}\}$
- $(F_1 \cup F_2)^+ = F^+$

Gegeben

- $R = (A, B, C)$
- $F = \{A \rightarrow B, B \rightarrow C\}$
- $R_1 = (A, B), R_2 = (\textcolor{red}{A}, C)$

Ist das eine verlustlose Zerlegung?

Hinweis (gemeinsame Attribute sind Superschlüssel in einer der Relationen):

- $(R_1 \cap R_2) \rightarrow R_1 \in F_R^+$
- $(R_1 \cap R_2) \rightarrow R_2 \in F_R^+$

JA:

- $R_1 \cap R_2 = \{A\}$ und $A \rightarrow AB$

Gegeben

- $R = (A, B, C)$
- $F = \{A \rightarrow B, B \rightarrow C\}$
- $R_1 = (A, B), R_2 = (\textcolor{red}{A}, C)$

Bewahrt diese Zerlegung Abhängigkeiten?

Hinweis (wir können alle Abhängigkeiten lokal überprüfen):

$$(F_1 \cup F_2)^+ = F^+$$

NEIN:

- $F_1 = \{A \rightarrow B, A \rightarrow AB\} \cup \{\text{viele triviale Abhängigkeiten}\}$
- $F_2 = \{\text{viele triviale Abhängigkeiten}\}$
- $(F_1 \cup F_2)^+ \neq F^+$
- $B \rightarrow C$ kann nicht überprüft werden ohne $R_1 \bowtie R_2$ zu berechnen

Zusammenfassung funktionaler Abhängigkeiten

Eigenschaften eines guten Datenbankentwurfs

- Keine Redundanz
- Keine Änderungsanomalien
- Alle Informationen können dargestellt werden

Warum Zerlegung?

- Verwandelt einen schlechten Entwurf in einen guten
 - Zerlege große Relationsschemata in mehrere kleinere

Eine gute Zerlegung

- ist verlustlos
- bewahrt Abhängigkeiten

Normalformen

Normalformen:

- legen Eigenschaften von Relationsschemata fest
- verbieten bestimmte Kombinationen von funktionalen Abhängigkeiten in Relationen
- sollen Redundanzen und Anomalien vermeiden
- Richtlinie, um gute Zerlegungen zu erhalten

Beispiele für Normalformen: **1NF**, **2NF**, **3NF**, **BCNF**, **4NF**, ...

Erste Normalform (1NF)

Eine Relation R ist in **1NF**, wenn alle Attribute atomar sind (keine zusammengesetzten, mengenwertigen oder relationenwertige Domänen).

1NF:

Nicht 1NF:

parents		
father	mother	child
Johann	Martha	{Else, Lucie}
Johann	Maria	{Theo, Josef}
Heinz	Martha	{Cleo}

parents		
father	mother	child
Johann	Martha	Else
Johann	Martha	Lucie
Johann	Maria	Theo
Johann	Maria	Josef
Heinz	Martha	Cleo

Zweite Normalform (2NF)

Ein Relationsschema R mit FDs F ist in **2NF**, falls jedes Nicht-Primattribut $A \in R$ voll funktional abhängig von jedem Kandidatenschlüssel der Relation ist.

- $(\kappa_j \rightarrow A) \in F^+$ und κ_j ist linksreduziert (**voll funktional abhängig**).

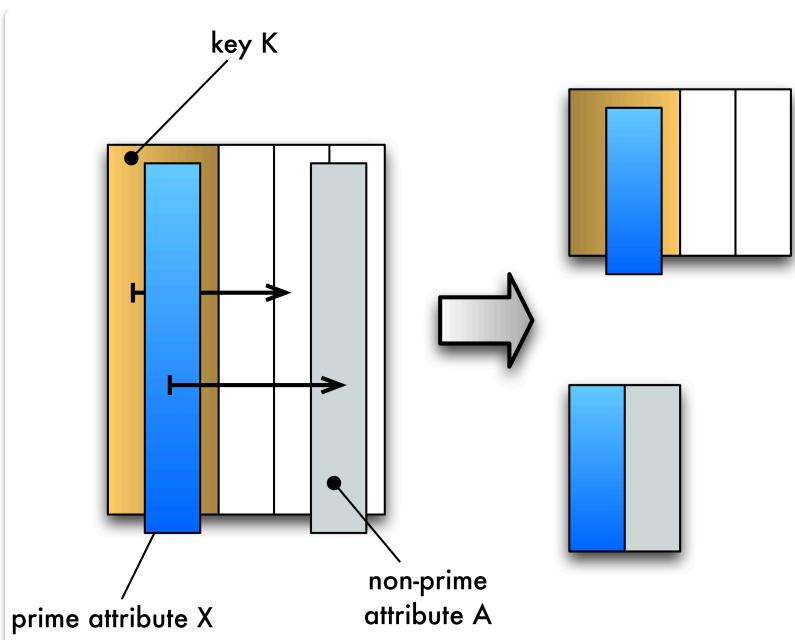
Definitionen:

- Ein Attribut A ist **prim**, wenn $A \in (\kappa_1 \cup \dots \cup \kappa_i)$ (Teil eines Kandidatenschlüssels).
- Ein Attribut A ist **nicht prim**, wenn $A \in R - (\kappa_1 \cup \dots \cup \kappa_i)$ (nicht Teil eines Kandidatenschlüssels).

2NF verhindert partielle Abhängigkeiten:

- Es gibt keine funktionale Abhängigkeit eines Nicht-Prim-Attributs von einer Teilmenge eines Schlüssels.

Eliminierung partieller Abhängigkeiten



studentCourses			
studID	courselD	name	semester
26120	5001	Fichte	10
27550	5001	Schopenhauer	6
27550	4052	Schopenhauer	6
28106	5041	Carnap	3
28106	5052	Carnap	3
28106	5216	Carnap	3
28106	5259	Carnap	3
...

Zerlegung in zwei Relationen:

- takes: {[studID, courselD]}
- student: {[studID, name, semester]}

Beide Relationen sind in 2NF.

Beispiel

studentCourses			
studID	courselD	name	semester
26120	5001	Fichte	10
27550	5001	Schopenhauer	6
27550	4052	Schopenhauer	6
28106	5041	Carnap	3
28106	5052	Carnap	3
28106	5216	Carnap	3
28106	5259	Carnap	3
...

- Kandidatenschlüssel: $\{ \{ \text{studID}, \text{courselD} \} \}$
- Schlüsselattribute (prim): $\{ \text{studID}, \text{courselD} \}$
- Nicht-Schlüsselattribute (nicht prim): $\{ \text{name}, \text{semester} \}$
- FDs: $\{ \text{studID} \} \rightarrow \{ \text{name} \}$ und $\{ \text{studID} \} \rightarrow \{ \text{semester} \}$

Partielle Abhängigkeit: studentCourses ist nicht in 2NF

Dritte Normalform (3NF)

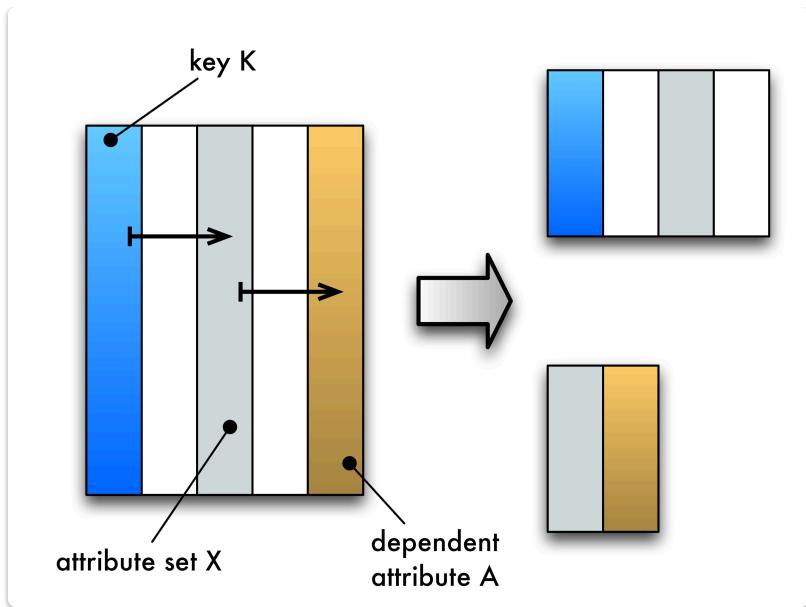
Ein Relationsschema R ist in **Dritter Normalform (3NF)**, wenn für jede für R geltende FD der Form $\alpha \rightarrow B$ mit Attribut $B \in R$ mindestens eine von drei Bedingungen gilt:

1. $B \in \alpha$, d.h. die FD ist **trivial** (Das Determinant enthält das Dependant).
2. α ist ein **Superschlüssel** von R .
3. B ist **prim** (Schlüsselattribut, also Teil eines Kandidatenschlüssels).

Eigenschaften der 3NF

- Nicht-Prim-Attribute dürfen keine anderen Attribute bestimmen.
- 3NF verhindert **partielle** und **transitive** Abhängigkeiten.
- **Ausnahme (Fall 3):** transitive Abhängigkeit mit Prim-Attributen als Endpunkte sind okay (wenn B prim ist, darf es auch transitiv von α abhängen).
- 3NF "beinhaltet" 2NF (wenn eine Relation in 3NF ist, ist sie automatisch auch in 2NF).

Die 3NF verhindert **transitive Abhängigkeiten** und beinhaltet 2NF.



- Nur Prim-Attribute können Endpunkte von transitiven Abhängigkeiten sein!

Beispiel

studentCourses			
studID	courselD	name	semester
26120	5001	Fichte	10
27550	5001	Schopenhauer	6
27550	4052	Schopenhauer	6
28106	5041	Carnap	3
28106	5052	Carnap	3
28106	5216	Carnap	3
28106	5259	Carnap	3
...

- Kandidatenschlüssel: $\{ \{ \text{studID}, \text{courselD} \} \}$
 - FDs: $\{ \text{studID} \} \rightarrow \{ \text{name} \}$, $\{ \text{studID} \} \rightarrow \{ \text{semester} \}$
- Hinweise ($\text{FD } \alpha \rightarrow B$)
- $B \in \alpha$, i.e., die FD ist trivial – beide FDs sind nicht trivial
 - α ist ein Superschlüssel von R – $\{ \text{studID} \}$ ist kein Superschlüssel
 - B ist prim – name und semester sind nicht prim

⇒ Die Relation ist nicht in 3NF

Boyce-Codd-Normalform (BCNF)

Ein Relationsschema R ist in **BCNF**, wenn für jede für R geltende FD der Form $\alpha \rightarrow B$ mit Attribut $B \in R$ mindestens eine von zwei Bedingungen gilt:

- $B \in \alpha$, d.h. die FD ist **trivial**.
- α ist **Superschlüssel** von R .

Eigenschaften der BCNF

- **Unterschied zu 3NF:** Die dritte Bedingung der 3NF (B ist prim) fällt weg.
- Die Boyce-Codd-Normalform ist eine weitere Verschärfung der 3NF (d.h., BCNF "beinhaltet" 3NF).
- **Eliminierung transitiver Abhängigkeiten auch zu Prim-Attributen** (im Gegensatz zur 3NF, wo transitive Abhängigkeiten zu Prim-Attributen erlaubt sein können).

Beispiel 3NF vs. BCNF

cities: $\{[\text{city}, \text{state}, \text{governor}, \text{inhabitants}]\}$

FDs

- ✓ $\{\text{city}, \text{state}\} \rightarrow \{\text{inhabitants}\}$
- ✓ $\{\text{state}\} \rightarrow \{\text{governor}\}$
- ✓ $\{\text{governor}\} \rightarrow \{\text{state}\}$

Kandidatenschlüssel:

$\{\text{city}, \text{state}\}$ and $\{\text{city}, \text{governor}\}$

Ist die Relation in 3NF? **JA**

- Trivialität? (Bedingung 1) – Keine der FDs ist trivial
- Ist die linke Seite ein Superschlüssel? (Bedingung 2) – FD1
- Ist die rechte Seite prim? (Bedingung 3) – FD2 und FD3

Ist die Relation in BCNF? **NEIN**

- Trivialität? (Bedingung 1) – Keine der FDs ist trivial
- Ist die linke Seite ein Superschlüssel? (Bedingung 2) – FD1
- ~~Ist die rechte Seite prim? (Bedingung 3)~~ – FD2 und FD3

BCNF Zerlegung

Es ist immer möglich, ein Relationsschema R mit FDs F in:

- **3NF Relationsschemata** R_1, \dots, R_n zu zerlegen, so dass die Zerlegung:
 - **verlustfrei** ist
 - **Abhängigkeiten bewahrt**
- **BCNF Relationsschemata** R_1, \dots, R_n zu zerlegen, so dass die Zerlegung:
 - **verlustfrei** ist

Wichtig: Es ist nicht immer möglich, eine BCNF-Zerlegung R_1, \dots, R_n von R zu erstellen, die alle Abhängigkeiten bewahrt.

Zerlegungsalgorithmus für BCNF

Der Algorithmus dient dazu, ein Relationenschema in BCNF zu überführen.

- Initialisierung: `result` ist eine Menge, die das ursprüngliche Relationenschema R enthält: $\text{result} = \{R\}$. Berechne die *Closure* (den Abschluss) aller funktionalen Abhängigkeiten F^+ (aller FDs).
 - **Schleife:** Solange es ein Relationenschema $R_i \in \text{result}$ gibt, das **nicht in BCNF** ist, führe folgende Schritte aus:
 - **Grund für Nicht-BCNF:** Es existiert eine für R_i geltende nicht-triviale funktionale Abhängigkeit (FD) $\alpha \rightarrow \beta$ mit:
 - $\alpha \cap \beta = \emptyset$ (Dies bedeutet, die Attribute in α und β sind disjunkt).
 - $\alpha \rightarrow R_i \notin F^+$ (Dies bedeutet, α ist **kein Superschlüssel** von R_i).
 - (*Erklärung: Ein Superschlüssel ist eine Menge von Attributen, die alle anderen Attribute in einem Relationenschema eindeutig identifiziert. Wenn α ein Superschlüssel wäre, wäre die Relation in BCNF bzgl. dieser FD.*)
 - **Zerlegung:** Zerlege R_i in zwei neue Relationenschemata R_{i1} und R_{i2} :
 - $R_{i1} := \alpha \cup \beta$
 - $R_{i2} := R_i - \beta$ (Alle Attribute von R_i , außer denen in β)
 - **Aktualisierung des Ergebnisses:** Ersetze R_i in der `result`-Menge durch die beiden neuen Relationenschemata:
 - $\text{result} := (\text{result} - R_i) \cup R_{i1} \cup R_{i2}$
 - **Hinweis zur Superschlüsselprüfung:** Um zu überprüfen, ob α ein Superschlüssel ist, kann auch α^+ (der Attributabschluss von α) berechnet werden, anstatt F^+ . Wenn α^+ alle Attribute von R_i enthält, ist α ein Superschlüssel von R_i .
-

Beispiel

cities: {[city, state, governor, inhabitants]}

Kandidatenschlüssel: {city, state} und {city, governor}

FDs:

- ① {state} → {governor}
- ② {city, state} → {inhabitants}
- ③ {governor} → {state}

\mathcal{R}_{i1} governments: {[state, governor]}

\mathcal{R}_{i2} cities: {[city, state, inhabitants]}

Diese Zerlegung ist verlustfrei und bewahrt Abhängigkeiten.

- FD1 und FD3 können governments zugewiesen werden.
- FD2 kann cities zugewiesen werden.

Zerlegung in BSNF

zipCatalog: {[street, city, region, zip]}

Kandidatenschlüssel: {street, city, region}

FDs

- | | |
|----------------------------------|---------------|
| ① {street, city, region} → {zip} | OK |
| ② {zip} → {city, region} | verletzt BCNF |

Zerlegung anhand von FD2:

- cities: {[zip, city, region]}
- streets: {[zip, street]}

Diese Zerlegung

- ist verlustfrei
- bewahrt aber nicht Abhängigkeiten! FD1 kann nicht überprüft werden, ohne den Join zu berechnen

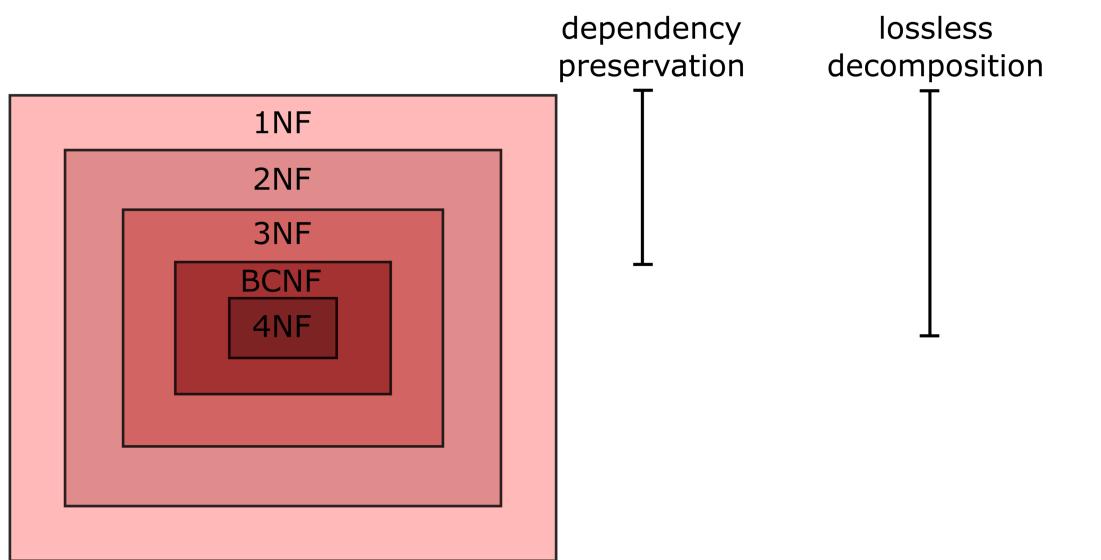
Übersicht über die Normalformen

Normalform	Kurzcharakteristik
1NF	nur atomare Attribute
2NF	keine partielle Abhangigkeit
3NF	keine transitive Abhangigkeit (Ausnahme: Primattribute)
BCNF	keine transitive Abhangigkeit

In der Praxis gibt man sich statt einer BCNF-Zerlegung, die zu einem Verlust der Abhangigkeitsbewahrung fuhren wurde, mit der dritten Normalform zufrieden.

Normalformen

- **Verlustlosigkeit** ist fur alle Zerlegungsalgorithmen in alle Normalformen garantiert.
- Die **Abhangigkeitsbewahrung** kann nur bis zur **dritten Normalform (3NF)** garantiert werden.



- **dependency preservation (Abhangigkeitsbewahrung):** Wenn wir eine Relation in kleinere Relationen zerlegen, bleiben alle ursprunglichen funktionalen Abhangigkeiten erhalten und konnen aus den zerlegten Relationen abgeleitet werden.
- **lossless decomposition (verlustlose Zerlegung):** Wenn wir eine Relation in kleinere Relationen zerlegen und diese spater wieder zusammenfugen, verlieren wir keine Informationen und erhalten die ursprungliche Relation zuruck.

Zusammenfassung

Eigenschaften eines „guten“ relationalen Entwurfs

- Vermeidung von Redundanz (mehrfaches Speichern gleicher Informationen).
- Vermeidung von Anomalien (Probleme beim Einfugen, Loschen oder Andern von Daten).

- Vermeidung von Null-Werten (leere Felder, die oft auf unvollständige Informationen hindeuten).
- Informationen können dargestellt werden (alle notwendigen Informationen sind im Schema abbildbar).

Funktionale Abhängigkeiten (FDs)

- Das wichtigste Konzept beim relationalen Datenbankentwurf.
- **Armstrong-Axiome:** Eine Menge von Regeln, um aus einer gegebenen Menge von FDs alle weiteren FDs (die Hülle einer Menge von FDs) zu bestimmen.
- **Kanonische/minimale Überdeckung:** Eine minimale Menge von FDs, die die gleiche Hülle wie die ursprüngliche Menge an FDs hat. Das bedeutet, es gibt keine redundanten FDs in der minimalen Überdeckung.
- FDs sind ein Werkzeug, um einen „guten“ relationalen Entwurf zu garantieren.

Zerlegung

- Hilft, Relationenschemata zu verbessern.
- Versucht, funktionale Abhängigkeiten in Schlüsselabhängigkeiten zu verwandeln.
 - **Intuitivere Erklärung:** Das Ziel ist es, dass jede funktionale Abhängigkeit in einer Relation durch den Primärschlüssel der Relation erklärt wird. Das heißt, wenn wir den Schlüssel kennen, kennen wir auch alle anderen Attribute.
- Ein gutes ER-Modell und die Abbildung auf Relationen führt oft direkt zu Relationen in 3NF (oder höhere NF).

3NF (Dritte Normalform)

- **Verlustfrei** und **abhängigkeitsbewahrend**.
- Kann **immer erreicht werden**.

BCNF (Boyce-Codd-Normalform)

- **Verlustfrei**, aber **nicht immer abhängigkeitsbewahrend**.
 - **Intuitivere Erklärung:** Obwohl BCNF eine stärkere Normalform ist (besser in Bezug auf Redundanzvermeidung), kann es sein, dass bei einer Zerlegung in BCNF einige der ursprünglichen funktionalen Abhängigkeiten nicht mehr direkt aus den zerlegten Relationen abgeleitet werden können, ohne sie wieder zusammenzufügen.

Denormalisierung

- Prozess, bei dem die Normalisierung „zurückgenommen“ wird, um Anfragen schneller bearbeiten zu können.
 - **Intuitivere Erklärung:** Manchmal wird bewusst eine Redundanz in der Datenbank zugelassen, um die Performance bei Lesezugriffen zu verbessern, auch wenn dies zu

den Nachteilen der Normalisierung (Redundanz, Anomalien) führen kann. Dies ist ein Kompromiss zwischen Datenintegrität und Abfragegeschwindigkeit.