Datenbanken und Informationssysteme Panikzettel

Mathis Müller, Luca Oeljeklaus, Tobias Polock,
Julian Schakib-Ekbatan, Philipp Schröer

5. April 2024

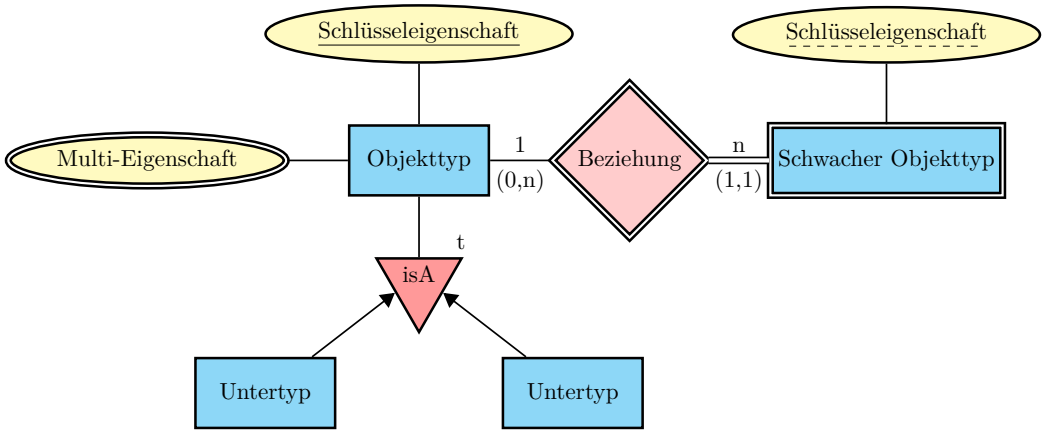
Inhaltsverzeichnis

1 Einführung

Dieser Panikzettel ist eine mehr oder weniger informelle Zusammenfassung der Vorlesung Datenbanken und Informationssysteme bei Prof. Matthias Jarke und Prof. Stefan Decker im Sommersemester 2017. Es werden hier die wichtigsten Aussagen, Tipps und Erklärungen gesammelt, die hoffentlich in dem unvermeidbaren Moment der Panik bei Hausaufgaben oder beim Lernen helfen.

2 Entity-Relationship-Modell

Das ER-Modell nutzt *Objekttypen* mit *Beziehungen* zwischen diesen. Beide können *Eigenschaften* haben. Als Notation für das ER-Modell gibt es ER-Diagramme:



Schwache Objekttypen sind von der Existenz des übergeordneten Typs abhängig.

Die Vererbung mit `isA` kann in vier verschiedenen Formen vorkommen:

- *Disjunkt*, wenn die Pfeile von dem `isA` weg zeigen. Oder *nicht disjunkt*, wenn Pfeile auf das `isA` zeigen.
- *Total*, wenn ein `t` an dem `isA` steht. Dann sind die gelisteten Untertypen alle Untertypen. Sonst ist die Vererbung *partiell*, markiert mit einem `p`.

Wir haben zwei Notationen für Kardinalitäten von Beziehungen:

- Im Diagramm bedeutet die `m:n`-Notation, dass `1 Objekttyp` in `Beziehung` zu `n schwachen Objekttypen` steht.

- Die `(min,max)`-Notation bedeutet, dass genau einem `Objekttyp` zwischen `0` und `n schwachen Objekttypen` zugeordnet werden können.

3 Relationales Datenmodell

Im relationalen Datenmodell werden Objekte und Beziehungen zwischen Objekten als Relationen modelliert.

Eine *Relation* R mit *Grad* k ist eine Teilmenge des kartesischen Produktes von k Mengen (auch *Wertebereich* oder *Domain* genannt) $D_1, \ldots, D_k$. Dann ist $R \subseteq D_1 \times \ldots \times D_k$.

Ein *Relationenschema* $\mathcal{R} = (X, \Sigma_X)$ besteht aus einem Relationssymbol R, einer endlichen Menge X von Attributen und einer Menge Σ_X von intrarelationalen Abhängigkeiten über X.

Eine Relation r mit Schema $\mathcal{R} = (X, \Sigma_X)$ ist eine konkrete Ausprägung des Relationenschemas. Sie heißt *konsistent/gültig*, falls r alle Abhängigkeiten aus Σ_X erfüllt.

Ein *Datenbankschema* $\mathcal{D} = (\mathcal{R}, \Sigma_\mathcal{R})$ besteht aus einer Menge $\mathcal{R}$ an Relationenschemata und einer Menge $\Sigma_\mathcal{R}$ an interrelationalen Abhängigkeiten über $\mathcal{R}$

Eine *Datenbank* $D = r_1, \ldots, r_n$ mit Schema $(D) = (\mathcal{R}, \Sigma_\mathcal{R})$ ist eine Menge an Relationen $r_1, \ldots, r_n$. Die Datenbank heißt konsistent oder gültig, falls sowohl inter- als auch intrarelationale Abhängigkeiten erfüllt sind.

3.1 Vom ER-Diagramm in das Relationale Modell

Wenn man ein ER-Diagramm in ein Relationales Modell überführt gilt es verschiedene Fälle zu beachten. Um die Form der Relation zu verdeutlichen, werden hier Beispiele aus der Luft gegriffen. Grundsätzlich gilt:

- Für jedes **Objekt** eine Relation anlegen: Student(Matrikelnummer, AnzahlAnstehendeKlausuren, Panik, ...).
- Für jede **n:m-Beziehung** (zwischen Student-Objekten und Panikzettel-Objekten):
  - Relation: Liest(Matrikelnummer, PanikzettelID, ...), es können noch mehr Attribute an der Beziehung beteiligt sein.
  - Interrelationen: Liest[Matrikelnummer] $\subseteq$ Student[Matrikelnummer] und Liest[PanikzettelID] $\subseteq$ Panikzettel[PanikzettelID].

Es gibt weitere Arten von Beziehungen:

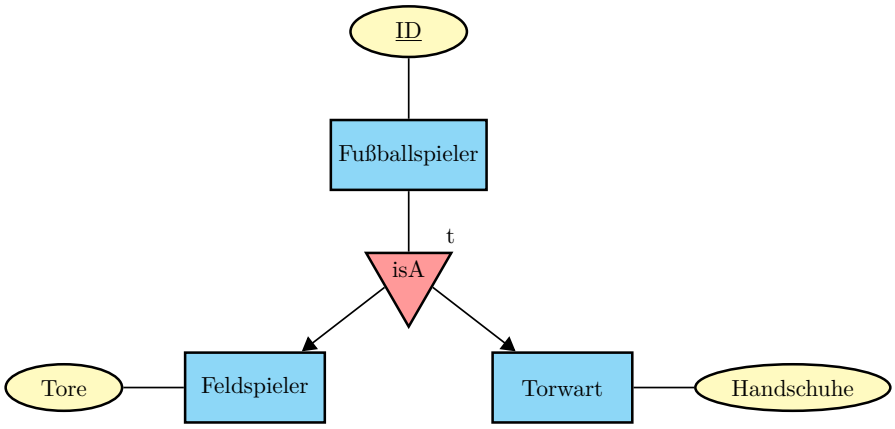
- **1:n Beziehung**: Keine eigene Relation für die Beziehung anlegen, sondern Primärschlüssel des Objekts, welches nur einmal auftritt, als Attribut in die Relation des n-mal auftretenden Objekts aufnehmen. Also z.B. n-Studenten, die auf genau eine Webseite zugreifen:
  - Relation ändern: Student(Matrikelnummer, ..., URL), sei URL hier Primärschlüssel der Webseite.
  - Interrelation: Student[URL] $\subseteq$ Webseite[URL].
- **Rekursive Beziehung**: Habe etwa in einem Repository jede Version Vorgänger und Nachfolger, die selbst wieder Versionen sind:

  - Relation (neben der Relation für *Version*): Voraussetzen(Vorgänger, Nachfolger).
  - Interrelation: Voraussetzen[Vorgänger] $\subseteq$ Version[VersionsID] und Voraussetzen[Nachfolger] $\subseteq$ Version[VersionsID].

- **n-stellige Beziehungen**: Sind mehr als zwei Objekte an der Beziehung beteiligt, hat die Beziehungsrelation trotzdem nur zwei Primärschlüssel. Die restlichen Primärschlüssel als einfache Attribute in die Beziehung mit aufnehmen.
- **1:1 Beziehung**: Gegebenenfalls zu einem Objekt verschmelzen, dabei nur einen der Primärschlüssel als Primärschlüssel behalten.

Folgende Fälle muss man auch noch beachten:

- **Mehrwertiges Attribut**: Eigene Relation für das Attribut mit Fremdschlüssel zum dazugehörigen Objekt und eine Interrelation zu diesem.
  - Student(Matrikelnummer, Name, ...).
  - Telefonnummern(Nummer, Matrikelnummer).
  - Telefonnummern[MatrikelNummer] $\subseteq$ Student[Matrikelnummer].
- **Schwache Objekttypen**: Fremdschlüssel des starken Objekts als Attribut in der Relation des Schwachen und Interrelation zu diesem (wie beim mehrwertigen Attribut).
- **Generalisierung/Spezialisierung**: (Betrachte untenstehendes ER-Diagramm)
  - Relationen für die Objekte formulieren, wobei jede Spezialisierung einen Fremdschlüssel auf die Generalisierung hat (in diesem Beispiel ID).
  - Wenn die Beziehung total ist, erstellen wir keine Relation für die Generalisierung sondern nur: Feldspieler(ID, Tore) und Torwart(ID, Handschuhe).
  - Wenn die Spezialisierungen disjunkt ist, verdeutlichen wir das durch: Feldspieler[ID] $\cap$ Torwart[ID] $= \varnothing$.



3.2 Relationale Algebra, Relationenkalkül, Tupelkalkül und Domänenkalkül

Wie Anfragen an eine Datenbank formuliert werden ist klar festgelegt. In konkreten Datenbankimplementierungen werden hierfür Anfragesprachen genutzt. Die Grundlage dafür sind die *Relationale Algebra* und das *Relationale Kalkül*, bei welchem man zwischen dem Tupel- und Bereichskalkül unterscheidet.

3.2.1 Relationale Algebra

Die relationale Algebra arbeitet auf Relationen (also Mengen) und definiert verschiedene Operationen zur Manipulation der Relationen.

Es existieren folgende Basisoperationen:

- **Vereinigung**: $R \cup S = \{t \mid t \in R \vee t \in S\}$.
- **Differenz**: $R - S = \{t \mid t \in R \wedge t \notin S\}$.
  Für Vereinigung und Differenz muss $\text{Sch}(R) = \text{Sch}(S)$ gelten.
- **Kartesisches Produkt**: $R \times S = \{(a_1, \ldots, a_r, b_1, \ldots b_s) \mid (a_1, \ldots, a_r) \in R \wedge (b_1, \ldots, b_s) \in S\}$.
- **Selektion**: $\sigma_F(R) = \{t \mid t \in R \wedge t \text{ erfüllt } F\}$.
- **Projektion**: $\pi_{A_{i_1},\ldots,A_{i_m}}(R) = \{(a_{i_1}, \ldots, a_{i_m}) \mid (a_1, \ldots, a_k) \in R\}$, wobei R den Grad k hat und $i_1, \ldots i_m \in [1, k]$ gilt.
- **Umbenennung**: $\rho_S(R)$ von Relation R in S.
- **Umbenennung**: $\rho_{A' \leftarrow A}(R)$ von Attribut A in A'.

Aus den Basisoperationen ableitbare Operationen:

- **Durchschnitt**: $R \cap S = \{t \mid t \in R \wedge t \in S\}$. Es muss $\text{Sch}(R) = \text{Sch}(S)$ gelten.
- **Quotient**: $R \div S = \{\, (a_1, \ldots, a_m) \mid \forall (b_1, \ldots, b_n) \in S : (a_1, \ldots, a_m, b_1, \ldots b_n) \in R \,\}$. Hier muss offensichtlich $\text{Sch}(S) \subseteq \text{Sch}(R)$ und $S \neq \emptyset$ gelten.
- **Natural Join**: Sei k die Anzahl der gemeinsamen Attribute von R und S, $A_1, \ldots, A_m, B_1, \ldots, B_k$ die Attribute von R und $B_1, \ldots, B_k, C_1,$. Außerdem muss $\text{Sch}(R) \cap \text{Sch}(S) \neq \emptyset$ sein. Dann ist

  $$R \bowtie S = \pi_{A_1,\ldots,A_m,B_1,\ldots,B_k,C_1,\ldots,C_n}(\sigma_{R.B_1=S.B_1 \wedge \ldots \wedge R.B_k=S.B_k}(R \times S))$$

  der Natural-Join zwischen R und S.
- **Theta-Join**: $R \bowtie_\theta S = \sigma_\theta(R \times S) = \{\, t \cup s \mid s \in R \wedge t \in S \wedge t \cup s \text{ erfüllt } \theta \,\}$. $t \cup s$ stellt das Tupel dar, welches alle Einträge von s und t in dieser Reihenfolge enthält.
- **Left-Outer-Join**: $R \mathbin{⟕} S$. Hier werden auch die Tupel aus R aufgenommen, die keinen Join-Partner in S finden, indem sie um den Grad von S an "`NULL`"-Einträgen am Ende erweitert werden.
- **Right-Outer-Join**: $R \mathbin{⟖} S$. Äquivalent zum Left-Outer-Join werden Tupel aus S ohne Join-Partner aufgenommen und um den Grad von R an "`NULL`"-Einträgen am Anfang erweitert.
- **Full-Outer-Join**: $R \mathbin{⟗} S = R \mathbin{⟕} S \cup R \mathbin{⟖} S$.
- **Left-Semi-Join**: $R \ltimes S = \sigma_R(R \bowtie S)$. Es bleiben also alle Tupel aus R übrig, die einen Join-Partner in S gefunden haben.
- **Right-Semi-Join**: $R \rtimes S = \sigma_S(R \bowtie S)$.

Da man in der relationalen Algebra immer auch die Operationen angibt, welche zur Bearbeitung einer Anfrage notwendig sind, ist sie eine *prozedurale* (formale) Sprache. Sie beschreibt *wie* eine Operation erfüllt wird.

3.2.2 Relationenkalkül

Dieser zur relationalen Algebra gleich mächtige Kalkül ist eine *deklarative* (formale) Sprache. Ausdrücke beschreiben die Eigenschaften der Tupel des Ergebnisses. Er beschreibt *was* ein zulässiges Tupel der Ergebnisrelation ist.

Der Kalkül gliedert sich in das Tupel- und das Domänenkalkül. Diese unterscheiden sich lediglich in ihrer Darstellungsweise der Tupel. Im Tupelkakül stellen Variablen immer Tupel dar, während im Domänenkalkül Variablen stets Werte von Attributen verschiedener Domänen repräsentieren.

Ein Ausdruck des Tupelkalküls hat die Form $\{ t \mid \varphi(t) \}$ wobei t eine sog. *Tupelvariable* ist und $\varphi(t)$ eine prädikatenlogische Formel erster Stufe ist.

Ist das Schema $(A_1 : D_1, \ldots, A_k : D_k)$ einer Tupelvariable t nicht aus dem Zusammenhang klar, oder möchte man eine neue Relation erzeugen, so gibt man das Schema folgendermaßen an:

$$\{ t \in (A_1 : D_1, \ldots, A_k : D_k) \mid \varphi(t) \}.$$

Als Kurzschreibweise hat sich der Tupelkonstruktor etabliert:

$$\{ [t.A_1, \ldots, t.A_n] \mid \varphi(t) \} = \{ t \in (A_1 : D_1, \ldots, A_k : D_k) \mid \varphi(t) \}.$$

Zur Auswertung müssen alle Tupelvariablen einer Formel $\varphi(t)$ durch konkrete Belegungen interpretiert werden. In der Ergebnisrelation landen dann alle Tupel r mit dem durch die Tupelvariable festgelegten Schema, welche bei der Interpretation der Formel eine 'wahre' Aussage liefern. Für die also gilt $I(\varphi(r|t)) = \mathbf{true}$, wobei alle Vorkommen von t durch r ersetzt werden und der Aussage dann durch I ein Wahrheitswert zugewiesen wird.

Bereichsvariablen sind im Domänenkalkül das Pendant zu den Tupelvariablen. Ein Ausdruck mit k Bereichsvariablen $x_1 : D_1, \ldots x_k : D_k$ hat die Form $\{ x_1, \ldots, x_k \mid \varphi(x_1, \ldots, x_k) \}$. Außerdem findet hier der Tupelkonstruktor ebenfalls seine Verwendung. Beispiel:

$$\{ m \mid \exists n([m, n] \in \mathit{Student} \wedge \ldots) \}.$$

4 SQL

SQL ist eine relationale Datenbanksprache, die auf einer Mischung der relationalen Algebra und des relationalen Kalküls aufbaut. Wir behandeln hier zwei Teilsprachen von SQL, die *Data Definition Language* (DDL) für das Schema und die *Data Manipulation Language* (DML) für das Bearbeiten und Anfragen von Daten.

Wir haben verschiedene Datentypen, unter anderem `NUMBER`, `DECIMAL`, `CHAR(n)` (Zeichenkette mit fester Länge), `VARCHAR(n)` (Zeichenkette mit maximaler Länge), `DATE` oder `BOOLEAN`.

4.1 Die DDL von SQL

Relation anlegen

Eine Relation `Panikzettel` erstellen mit zwei Attributen, die beide nicht `NULL` sein dürfen. Der Modulname ist der primäre Schlüssel.

```
CREATE TABLE Panikzettel (
    Modul VARCHAR (500) NOT NULL PRIMARY KEY,
    Kuerzel VARCHAR (50) NOT NULL
  )
```

Wir können auch mit *Foreign Keys* im Schema sichern, dass Referenzen auf andere Relationen erhalten bleiben: Mit dem Spaltenattribut `REFERENCES AndereTabelle(Attr1, Attr2)`.

Schlüssel können auch über mehrere Attribute spannen. Dazu schreibt man nach dem Attributen z.B.: `FOREIGN KEY (Modul, Kuerzel) REFERENCES AndereTabelle(Modul, Kuerzel)`.

Relation ändern
`ALTER TABLE Name` gefolgt von `ADD Attributname Typ` oder `DROP COLUMN Attributname`.

Relation löschen
`DROP TABLE Name`.

Sichten
Sichten sind Relationen, die über Anfragen definiert sind. `CREATE VIEW NurKuerzel AS SELECT Kuerzel FROM Panikzettel`. Löschen mit `DROP VIEW Name`.

4.2 Die DML von SQL

Wir wollen an dieser Stelle einfach die Basisoperationen der relationalen Algebra in SQL übersetzen.

Nehmen wir an, dass wir folgendes Schema haben: $R(A, B, C, D), S(E, F, G)$ und $T(A, B, C, D)$.

- **Vereinigung**: $R \cup T$: `SELECT * FROM R UNION SELECT * FROM T`.
- **Differenz**: $R \setminus T$: `SELECT * FROM R MINUS SELECT * FROM T`.
- **Kreuzprodukt**: $R \times T$: `SELECT * FROM R CROSS JOIN T`.
- **Selektion**: $\sigma_{B=b}(R)$: `SELECT * FROM R WHERE B = ‘b‘`.
- **Projektion**: $\pi_{A,C}(R)$: `SELECT DISTINCT A, C FROM R`.
- **Umbenennung**: $\rho_V(R)$: `SELECT V.A FROM R AS V`.
- **Umbenennung**: $\rho_{K \leftarrow C}(R)$: `SELECT A, B, C AS K, D FROM R`.
- **Relation**: R: `SELECT * FROM R`.
- **Theta-Join**: $R \bowtie_{\theta(B,F)} S =$ `SELECT * FROM R, S WHERE` θ`(B, F)`.
- **Natural Join**: $R \bowtie U =$ `SELECT * FROM R, U`.

`DISTINCT` löscht Duplikate. Mit Funktionen kann man auch Mengen von Werten weiterverrechnen. Dazu gibt es etwa `COUNT`, `MIN`, `MAX`, `SUM` und `AVG`.
z.B.: `SELECT COUNT(DISTINCT Name) FROM Student`

Mithilfe von `GROUP BY` können Tupel zu Gruppen zusammengefasst werden. Man könnte eine Anfrage der Form `SELECT .. FROM .. GROUP BY attr1, attr2 HAVING bedingung` schreiben.

Mit `ORDER BY attr1, attr2, ... [ASC|DESC]` bei einem `SELECT` können dann die Tupel sortiert werden. Die Sortierungsrichtung ist optional.
z.B.: `SELECT Name FROM Student ORDER BY Matrikelnummer ASC`

Joins

Klassischer Stil:

```
SELECT K.KName, Ware
FROM Kunde K, Auftrag
WHERE K.KName = Auftrag.KName
```

SQL92:

```
SELECT KName, Ware
FROM Kunde NATURAL JOIN Auftrag
```

Theta-Join mit expliziter Join-Bedingung:

```
SELECT Kunde.KName, Ware
FROM Kunde JOIN Auftrag
ON Kunde.KName = Auftrag.KName
```

Analog andere Joins:
`[LEFT|RIGHT|FULL] (OUTER) JOIN`.

Analog zum `SELECT` gibt es auch `INSERT INTO`, `DELETE FROM` und `UPDATE ... SET`.
z.B.: `INSERT INTO Konto(Kontostand, Name) VALUES (0, 'DouglasAdams')`
und `UPDATE Konto SET Kontostand = 42 WHERE Name = 'DouglasAdams'`

5 Relationale Entwurfstheorie

5.1 Funktionale Abhängigkeiten

Eine *funktionale Abhängigkeit* (FD) $\alpha \to \beta$ in einer Relation mit Attributmenge $X \supseteq \alpha \cup \beta$ besagt, dass wenn alle Werte der Attribute aus α bekannt sind, die Werte der Attribute aus β eindeutig bestimmt sind. An den funktionalen Abhängigkeiten lässt sich die Qualität eines Relationsschemas erkennen, daraus ergeben sich die Normalformen (5.4).

Formal sagt man, dass $\alpha \to \beta$ in R gilt, wenn

$$r.\alpha = r'.\alpha \implies r.\beta = r'.\beta$$

für alle möglichen Ausprägungen $r, r' \in R$.

Ein Superschlüssel ist eine Menge $\kappa \subseteq X$, sodass $\kappa^+ = X$ ist. Ein Schlüsselkandidat ist ein (bzgl. $\subseteq$) minimaler Superschlüssel.

Man kann funktionale Abhängigkeiten auseinander ableiten. Man sagt: Eine funktionale Abhängigkeit f folgt aus einer Menge funktionaler Abhängigkeiten F, geschrieben $F \models f$, wenn in jeder Relation, die alle Abhängigkeiten aus F erfüllt, auch f erfüllt ist. Um solche Implikationen syntaktisch zu prüfen, gibt es das Armstrong-Kalkül.

5.2 Armstrong-Kalkül

Die Axiome sind:

Reflexivität:

$$\frac{\beta \subseteq \alpha}{\alpha \to \beta}\, A_1$$

Verstärkung:

$$\frac{\alpha \to \beta}{\alpha\gamma \to \beta\gamma}\, A_2$$

Transitivität:

$$\frac{\alpha \to \beta \qquad \beta \to \gamma}{\alpha \to \gamma}\, A_3$$

Die (geschlossene) Hülle einer Menge funktionaler Abhängigkeiten ist

$$F^+ := \{\, f \mid F \vdash f \,\} = \{\, f \mid F \models f \,\}$$

Die Attributhülle einer Menge von Attributen α ist

$$\alpha^+ = \left\{\, x \;\middle|\; \alpha \to \{\, x \,\} \in F^+ \,\right\}$$

Wir haben die folgenden weiteren Regeln eingeführt:

Vereinigung:

$$\frac{\alpha \to \beta \qquad \alpha \to \gamma}{\alpha \to \beta\gamma}\, A_4$$

Dekomposition:

$$\frac{\alpha \to \beta\gamma}{\alpha \to \beta}\, A_5$$

Pseudotranistivität:

$$\frac{\alpha \to \beta \qquad \beta\gamma \to \delta}{\alpha\gamma \to \delta}\, A_6$$

5.3 Kanonische Überdeckung

Statt der Hülle (mit vielen Redundanzen) verwendet man auch die *kanonische Überdeckung*. Dies ist eine Menge F_C mit den folgenden Eigenschaften:

- $F_C^+ = F^+$.
- In jedem $\alpha \to \beta \in F_C$ kann man keine $A \in \alpha$ oder $B \in \beta$ weglassen.
- Jede linke Seite ist eindeutig.

Um die kanonische Überdeckung zu erhalten, nutzt man den folgenden Algorithmus:

1. Linksreduktion: Entferne unnötige Elemente der linken Seiten.
   - Für $\alpha \to \beta \in F$: Prüfe für alle $A \in \alpha$, ob gilt: $\beta \subseteq \text{AttrHülle}(F, \alpha \setminus A)$
2. Rechtsreduktion: Entferne unnötige Elemente der rechten Seiten.
   - Für $\alpha \to \beta \in F$: Prüfe für alle $B \in \beta$, ob gilt: $B \in \text{AttrHülle}((F_c \setminus (\alpha \to \beta)) \cup (\alpha \to \beta \setminus B), \alpha)$
3. Entferne FDs der Form $\alpha \to \emptyset$.
4. Fasse gleiche linke Seiten zusammen.

Beispiel Ausgehend von den FDs:

$$A \to BE \qquad A \to D \qquad F \to A$$
$$AC \to F \qquad BC \to E \qquad C \to A$$

1. Schritt: Linksreduktion

1. a) Prüfe, ob A in der Hülle von C ist.
   b) $C \to F$, dann $F \to A$.
   c) Bedingung a) ist erfüllt.
   d) Lösche A aus $AC \to F$.
2. a) Prüfe, ob B in der Hülle von C ist.
   b) $C \to A$, dann $A \to BE$.
   c) Bedingung a) ist erfüllt.
   d) Lösche B aus $BC \to E$.

Zwischenergebnis:

$$A \to BE$$
$$A \to D$$
$$F \to A$$
$$C \to F$$
$$C \to E$$
$$C \to A$$

2. Schritt: Rechtsreduktion

1. B und E tauchen sonst nicht in der Hülle von A auf.
2. D auch nicht.
3. A nicht in der Hülle von F.
4. F nicht in der Hülle von C.
5. Durch $C \to A$ und $A \to BE$ ist E schon in der Hülle von C. Löschen.
6. Durch $C \to F$ und $F \to A$ ist A es auch. Löschen.

Zum Schluss noch zusammenkürzen ergibt:

$$A \to BED$$
$$F \to A$$
$$C \to F$$

5.4 Normalformen

Zur Vermeidung von Redundanzen ist es wünschenswert, ein Relationenschema in eine verlustlose und wenn möglich abhängigkeitserhaltende Zerlegung zu bringen, die einer gewissen Form entspricht.

Verlustlos heißt, dass alle Informationen erhalten bleiben, d.h. dass der Natural Join der neuen Tabellen genau die alte Tabelle liefert.

Abhängigkeitserhaltend heißt, dass sich alle funktionalen Abhängigkeiten des alten Schemas aus den funktionalen Abhängigkeiten der neuen Schemata ergeben.

Es gilt 1NF $\supseteq$ 2NF $\supseteq$ 3NF $\supseteq$ BCNF($\supseteq$ 4NF $\supseteq$ 5NF).

Das heißt, um z.B. ein Relationsschema in 2NF zu bringen müssen wir sie zuerst in 1NF bringen.

5.4.1 Erste Normalform (1NF)

Attribute von Relationen in 1NF haben einen atomaren Wertebereich. Das heißt sie sind selbst keine Mengen oder Tupel, etc.

nicht in 1NF:

Vater	Mutter	Kind
Johann	Martha	{Else, Lucia}
Johann	Maria	{Theo, Jose}
Heinz	Martha	{Cleo}

in 1NF:

Vater	Mutter	Kind
Johann	Martha	Else
Johann	Martha	Lucia
Johann	Maria	Theo
Johann	Maria	Jose
Heinz	Martha	Cleo

5.4.2 Zweite Normalform (2NF)

Die 2NF Form erreicht man, indem man Attribute, die verschiedenen Konzepten angehören, aufspaltet, und nur noch die Schlüsselkandidaten der Relation zusammenhält. Dabei müssen die Nichtschlüssel-Attribute einem Schlüsselkandidaten zugeordnet werden, von dem sie funktional abhängig sind. Sei im folgenden die Relation der Vorlesungen und des Professors, der sie hält, gegeben:

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS

Da Name, Rang und Raum von PersNr funktional abhängig sind und Titel und SWS von VorlNr abhängen, teilt sich die Relation wie folgt auf:

PersNr	Name	Rang	Raum

PersNr	VorlNr

VorlNr	Titel	SWS

Das heißt, wir haben für zwei Konzepte getrennte Relationen, und eine weitere Relation, die diese über ihre Schlüsselkandidaten verknüpft.

5.4.3 Dritte Normalform (3NF)

Ein Relationsschema (X, F) ist in 3NF, wenn für alle Nichtschlüsselattribute $B \in X$ und $\alpha \to \{ B \} \in F^+$ die Menge α ein Superschlüssel ist. (Ein Nichtschlüsselattribut kommt in keinem Schlüsselkandidaten vor.)

Um eine Relation in 3NF zu bringen, wendet man den Synthese-Algorithmus an. Um ein Relationsschema $\mathcal{R} = (X, F)$ zu zerlegen:

1. Berechne kanonische Überdeckung F_C.
2. Für jedes $\alpha \to \beta \in F_C$:

   Erstelle Relationsschema der in $\alpha \to \beta$ vorkommenden Attribute:
   $(\alpha \cup \beta, \{ \alpha' \to \beta' \in F_C \mid \alpha', \beta' \subseteq \alpha \cup \beta \})$.
3. Falls kein Schema einen Schlüsselkandidaten von $\mathcal{R}$ enthält:

   Wähle einen Schlüsselkandidaten κ und erstelle Schema $(\kappa, \{ \kappa \to \kappa \})$.
4. Lösche jede Relation, deren Attributmenge in der Attributmenge einer anderen Relation enthalten ist.

5.4.4 Boyce-Codd Normalform (BCNF)

Nicht notwendigerweise abhängigkeitserhaltend! (5.4)

Ein Relationsschema (X, F) ist in BCNF wenn für alle nichttrivialen $\alpha \to \beta \in F^+$ α ein Superschlüssel ist.

Um eine Relation in BCNF zu bringen, gibt es den Dekompositionsalgorithmus:

Solange ein $\mathcal{R}_i = (X_i, F_i)$ nicht in BCNF ist:

1. Finde nichttriviale Abhängigkeit $\alpha \to \beta \in F_i^+$ mit $\alpha \cap \beta = \emptyset$ und $\alpha \nrightarrow X_i$.
2. $X_{i1} := \alpha \cup \beta$, $X_{i2} := X_i \setminus \beta$
3. Ersetze $\mathcal{R}_i$ durch $\mathcal{R}_{i1}$ und $\mathcal{R}_{i2}$ mit Attributmengen X_{i1} bzw. X_{i2} und FDs durch Einschränkung von F_i^+.

6 Relationale Anfragebearbeitung

6.1 Anfrageoptimierung

Wenn wir mit dem Kanonischen Auswertungsplan Anfragen auswerten, haben wir meistens ziemlich miese Performance. Der Restrukturierungsalgorithmus kann den Kanonischen Auswertungsplan optimieren.

6.1.1 Kanonischer Auswertungsplan

Einfache SQL-Anfragen der Form links werden übersetzt in die Formel auf der rechten Seite.

1. Kartesisches Produkt von $R1, R2, \ldots$
2. Selektionen der Bedingungen
3. Projektion der Ergebnistupel

```
SELECT A1, A2, ...
FROM R1, R2, ...
WHERE B1 AND B2, ...
```

$$\downarrow$$

$$\pi_{A_1,A_2}(\sigma_{B_2}(\sigma_{B_1}(R_1 \times R_2)))$$

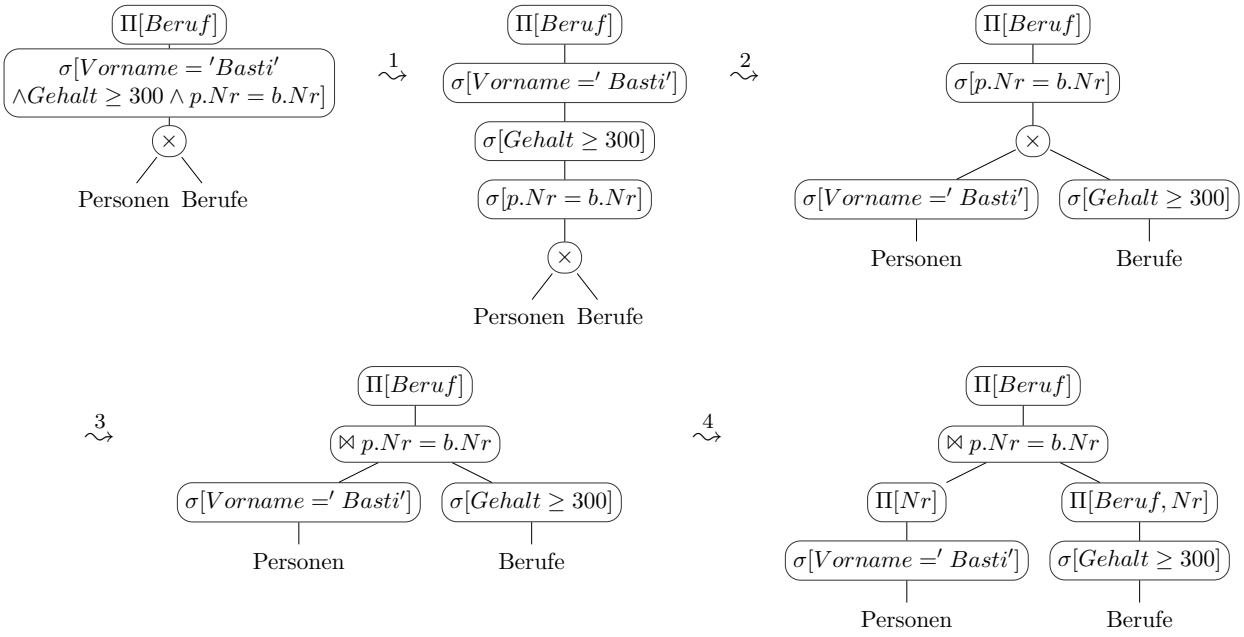
6.1.2 Restrukturierungsalgorithmus

Der Restrukturierungsalgorithmus wendet einfache Regeln an, um Auswertungspläne zu vereinfachen.

1. Aufbrechen von Selektionen ($\sigma_{a \wedge b} = \sigma_a \cdot \sigma_b$).
2. Alle Selektionen so weit wie möglich nach innen ziehen.
3. Selektionen und Kreuzprodukte zu Joins zusammenfassen.
4. Projektionen so weit wie möglich innen einfügen oder nach innen verschieben.
5. Einzelne Selektionen wieder zusammen fassen.

Beispiel:

```
SELECT Beruf FROM Personen AS p, Berufe AS b WHERE Vorname = 'Basti' AND Gehalt >= 300 AND p.Nr == b.Nr
```



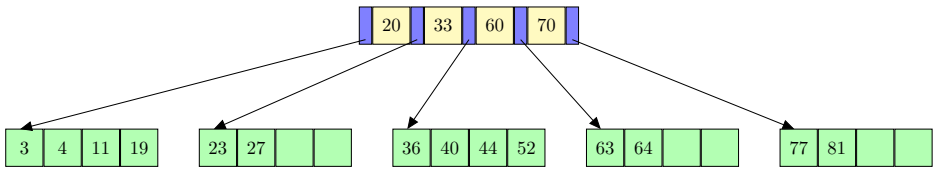
6.2 Indexstrukturen

Indexstrukturen dienen der effizienten Implementierung der Selektion.

6.2.1 B-Bäume

Ein *B-Baum* ist ein Baumstruktur mit Parameter $M \in 2\mathbb{N}$ (und $m = M/2$) wobei für einen nicht-leeren B-Baum folgende Eigenschaften gelten:

- Jeder Knoten enthält höchstens M Schlüssel.
- Die Wurzel enthält mindestens einen Schlüssel.
- Jeder Knoten außer der Wurzel enthält mindestens $m = M/2$ Schlüssel.
- Ein innerer Knoten mit b Schlüsseln hat genau b+1 Kinder.
- Alle Blätter befinden sich auf der selben Höhe.



Um den Baum nach einem Element zu **durchsuchen**, wird beginnend in der Wurzel auf den jeweiligen Knoten eine binäre Suche ausgeführt und immer im entsprechenden Teilbaum gesucht. Komplexität: $\mathcal{O}(\log_2 N)$.

Zur Erfüllung einer **Bereichsanfrage** (min, max) wird der kleinste Wert, welcher größer oder gleich min ist, rekursiv gesucht. Dann wird eine in-order Traversierung bis zum größten Element, welches kleiner oder gleich max ist, durchgeführt.

Beim **Einfügen** eines neuen Elements wird erst ein zum Objekt passender Knoten gesucht und dort eingefügt. Falls das Blatt deswegen überläuft, so wird der Knoten gesplittet; hierbei wird das mittlere Objekt dem Vorgängerknoten hinzugefügt und die beiden Hälften in zwei neue Knoten aufgeteilt. Wenn der Vorgänger dadurch überläuft splitte rekursiv ggf. bis zur Wurzel. Komplexität: $\mathcal{O}(\log_m N)$

Löschen: Erst einmal wird der Knoten gesucht, welcher den zu löschenden Schlüssel beinhaltet. Es treten drei Fälle auf.

- Ist der Knoten ein Blatt, so wird der Schlüssel gelöscht. Hat der Knoten nun weniger als $M/2$ Schlüssel, so findet eine Verschmelzung statt.
- Ist der Knoten ein innerer, so wir der zu löschende Schlüssel durch den größten Schlüssel l im linken Teilbaum ersetzt. Dann wird l aus seinem Blatt gelöscht und ggf. findet auch hier eine rekursive Verschmelzung statt.
- Handelt es sich beim Knoten um die Wurzel, wird das Objekt einfach gelöscht.

Verschmelzen: Existiert ein Nachbarknoten mit mehr als $M/2$ Schlüsseln, so findet ein Ausgleich statt. Die Schlüssel der beiden Knoten werden gleichmäßig auf beide Knoten verteilt und ein neuer *Trennschlüssel* bestimmt. Sonst gibt es keinen Nachbarknoten mit genügend Schlüsseln, aber mindestens einen mit genau $M/2$ Schlüsseln. Beide Knoten verschmelzen ihre Elemente und den Trennschlüssel im Vorgänger miteinander zu einem neuen Knoten mit M Elementen.

B+-Bäume: B+-Bäume sind B-Bäume, deren Knoten nur Schlüssel enthalten und wo die Blätter verkettet sind um schnelle Iteration zu ermöglichen. Die echten Daten werden dann an einem anderen Ort gespeichert.

7 XML

XML ist ein kompaktes und effizientes Format in einem passenden Anwendungskontext. XML-Dokumente bestehen aus Bäumen, wobei die *Elementknoten* auch *Attribute* haben können.

7.1 XML Schema

Mit XML Schema können wir für XML-Dokumente ein Schema in XML definieren. Man verwendet `http://www.w3.org/2001/XMLSchema` als Namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
①    xmlns="http://panikzettel.philworld.de.de/Panikzettel"
②    xmlns:xs="http://www.w3.org/2001/XMLSchema"
③    targetNamespace="http://panikzettel.philworld.de.de/Panikzettel"
     elementFormDefault="qualified">
④    <xs:complexType name="PanikzettelType">
         <xs:sequence>
             <xs:element name="Titel" type="xs:string" />
             <xs:element name="Datum" type="xs:date" />
         </xs:sequence>
```

```
        <xs:attribute name="URL" type="xs:string" />
    </xs:complexType>
⑤   <xs:complexType name="PanikzettelListeType">
        <xs:sequence>
            <xs:element name="Panikzettel" minOccurs="0"
    maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
⑥   <xs:element name="PanikzettelListe" type="PanikzettelListeType"/>
</xs:schema>
```

1. Der Standardnamespace ist die angegebene URI.
2. Wir importieren XML Schema als `xs`.
3. Wir deklarieren gerade für die angegebene URI.
4. Ein Knotentyp, wo jeder Knoten dann `Titel` und `Datum` in der angegebenen Reihenfolge enthält und entweder davor oder danach einen `URL`-Knoten.
5. Ein Knotentyp, der beliebig viele Elemente mit Namen `Panikzettel` enthalten darf.
6. Der Wurzelknoten jedes Dokumentes, was dieses neue Schema verwendet, muss `PanikzettelListe` heißen und ist vom Typ `PanikzettelListeType`.

7.2 XPath

Mit XPath können wir Knoten in XML-Bäumen suchen. Diese Suche wird “Lokalisierung” genannt. Eine XPath-Anfrage kann dann durch `/` getrennte Lokalisierungsschritte enthalten; diese Ausdrücke nach dem `/` selektieren dann weiter auf die zutreffenden Knoten von zuvor.

Ein einzelner Lokalisierungsschritt hat die Struktur `Achse::Test[Prädikat]`. Die *Achsen* machen aus jedem Knoten eine bestimmte andere Menge von Knoten, siehe dazu Beispiele. Ein *Knotentest* prüft eine einfache Bedingung, meist etwa einen Knotennamen. Das *Prädikat* prüft dann komplexere Bedingungen.

Mögliche Achsen sind: `self`, `attribute`, `child` oder `parent` oder `ancestor`. Knotentests können unter anderem sein: `node()` (immer wahr), `*` (wahr für jeden Knoten der Achse), `element(Name)`, `attribute(Name)`.

In Pfadausdrücken gibt es Abkürzungen wie `.` (aktueller Knoten), `..` (Vaterknoten), `/` am Anfang (für den Wurzelknoten), `@` (Attribute), `//` (alle Nachfahren und der Knoten selber) und `[n]` für das n-te Element.

Ein Beispiel: `/Universität/Fakultäten/Fakultät[FakName="Informatik"]//Vorlesung/@Name` (das Name-Attribut aller Informatik-Vorlesungen).

8 RDF

RDF ist ein Graph-basiertes Datenmodell mit Knoten und Kanten. RDF an sich gibt keine Serialisierung, d.h. auch keine Sprache vor. Alle RDF-Daten sind Tripel (Subjekt, Prädikat, Objekt). Subjekt und Objekt sind Knoten, das Prädikat ist eine (gerichtete) Kante.

Als Typen gibt es URIs, Literale (Werte wie Text oder Zahlen) oder *Blank Nodes* (Knoten nur mit einer Identität). Als Subjekte können URIs und Blank Nodes auftreten, als Prädikate nur URIs und Objekte können von jedem Typ sein. Bei Literalen kann ein Datentyp angegeben werden.

8.1 Turtle

Turtle ist ein "knappes" Datenformat für RDF.

- URIs: `<http://panikzettel.philworld.de/>`.
- Literale: `"Hallo Welt"@DE` (mit Sprache) oder `"31"^^xsd:int` (mit Typ).
- Tripel werden durch einen Punkt getrennt, die Tripeleinträge werden nacheinander aufgeschrieben. Mehrere Tripel mit gleichem Objekt oder zusätzlich gleichem Prädikat können zusammengefasst werden.
- Namespace-Abkürzung deklarieren: `@prefix rdfs <http://www.w3.org/2000/01/rdf-schema>` kürzt die lange URI als `rdfs` ab.
- Blank Nodes: `_:lokalerName`.

8.2 SPARQL

SPARQL kann man als SQL für RDF verstehen. Anfragen mit SPARQL bestehen aus vier Teilen, in denen Variablen `?name` auftauchen können:

1. Definition von Namespaces, etwa `PREFIX ex: <http://example.com/resources/>`.
2. Anfrageklausel (Projektion): `SELECT`, `ASK`, `CONSTRUCT`, `DESCRIBE`.
3. `WHERE`-Klausel.
4. Anfragemodifikatoren, etwa `ORDER BY`.

Eine Anfrage kann dann etwa so aussehen:

```
SELECT ?person ?job
  WHERE {
    ?person foaf:knows :Panikzettel.
    ?person foaf:name ?name.
    FILTER (str(?name) == "Basti").
    OPTIONAL { ?person foaf:job ?job }
  }
```

Dabei ist `FILTER` ein SPARQL-Filterausdruck, der das Ergebnis weiter einschränkt. `OPTIONAL` erlaubt es, möglicherweise vorhandenene Werte mit aus zu geben.

8.3 RDF Schema (RDFS)

Das *RDF Schema* ist ein RDF Vokabular, um in RDF ein Schema für RDF-Daten zu definieren. Der Namespace ist `http://www.w3.org/2000/01/rdf-schema#`, meist abgekürzt als `rdfs`.

In RDF selber gibt es *Klassen* und Zugehörigkeit zu Klassen kann man mit `ex:EineAInstanz rdf:type ex:A` festlegen. Eine Entität kann Instanz mehrerer Klassen sein.

Mit RDF Schema kann man nun auch etwa *Unterklassen* definieren: `ex:A rdfs:subClassOf ex:OberA`. Die Typinformationen sind transitiv, also implizit gilt nun auch schon `ex:EineAInstanz rdf:type ex:OberA`.

Prädikate können ähnlich definiert werden: `ex:haben rdf:type rdf:Property`. Dann können wir dies nutzen: `ex:Jon ex:haben ex:keineAhnung`. Mit `rdfs:domain` und `rdfs:range` können erlaubte Typen für Subjekt und Objekt respektive für ein Prädikat angegeben werden.

Man möchte an dieser Stelle bemerken, wie `rdf` und `rdfs` drunter und drüber verwendet werden. Das ist leider Absicht.

9 Transaktionsverwaltung

9.1 Transaktionskonzept

Transaktionen werden durch das ACID-Prinzip charakterisiert.

Atomicity:
Der Effekt einer Transaktion kommt entweder ganz oder gar nicht zu tragen.

Consistency:
Eine konsistente Datenbank ist auch nach einer Transaktion konsistent.

Isolation:
Veränderungen anderer Nutzer werden nicht wahrgenommen.

Durability:
Der Effekt einer Transaktion ist dauerhaft.

Definitionen:

- $\text{DB} = \{ x, y, z, ... \}$ unsere Datenbank, eine Menge von Objekten.
- p_i ein beliebiger Zugriff auf eine Objekt.
- s eine für eine Menge von Transaktionen definierte Reihenfolge von Zugriffen, auch Schedule genannt.
- $r(x)$ und $w(x)$, $x \in \text{DB}$ die Lese- und Schreibzugriffe auf Objekte.
- $t = p_1, ..., p_n$ eine Transaktion aus endlich vielen Zugriffen.
- c_i und a_i die `commit` bzw. `abort` Pseudoaktionen von t_i, die eine Transaktion abschließen und sich ausschließen.

Wir notieren nebenläufige Transaktionen und ihre Zugriffe mit Indizes:

$$t_i = r_i(x)w_i(y) \text{ und } t_j = w_j(x)r_j(y)$$

9.2 Synchronisation

Falls verschiedene Transaktionen nicht isoliert sind kann es zu Anomalien kommen:

- *Lost Updates*: Änderungen einer Transaktion können von einer anderen überschrieben werden. In dem Schedule $r_1(x)r_2(x)w_2(x)w_1(x)$ berücksichtigt t_1 die Veränderung von x von t_2 nicht.
- *Dirty read/write*: So bezeichnen wir das Lesen bzw. Überschreiben von Daten einer nicht abgeschlossenen Transaktion.
- *Nicht-reproduzierbares Lesen*: Ein Wert wird zwischen zwei Lesezugriffen einer Transaktion verändert.
- *Phantomproblem*: Ähnlich zum nicht-reproduzierbaren Lesen, auch *Inconsistent read* genannt, da zwischen Lesezugriffen bearbeitete Werte inkonsistent sind.

Sei $T = \{ t_1, ..., t_n \}$ eine Menge von Transaktionen. Dann ist:

- *shuffle*(T): Die Menge aller Folgen von Aktionen, die $t_1, ..., t_n$ als Teilfolgen, und keine weiteren Aktionen enthält.

- *Vollständiges Schedule*: Eine Folge $s \in \text{shuffle}(T)$, die die Aktionen c_i oder a_i für $t_i \in T$ einschließt.
- *Serielles Schedule*: Ist ein vollständiges Schedule, für das eine Permutation ρ von $\{ 1, ..., n \}$ gibt, so dass $s = t_{\rho(1)}, ..., t_{\rho(n)}$

Wir definieren folgende Mengen:

$\texttt{op}(s)$
Menge aller Aktionen in s

$\texttt{trans}(s)$
Menge aller Transaktionen in s

$\texttt{commit}(s)$
Menge der bestätigten Transaktionen

$\texttt{abort}(s)$
Menge der abgebrochenen Transaktionen

$$\texttt{active}(s) = \texttt{trans}(s) \setminus (\texttt{commit}(s) \cup \texttt{abort}(s))$$
Menge der aktiven Transaktionen

Sei s ein Schedule mit zwei Transaktionen t und t'. Dann ist:

- *Konflikt*: zwei Zugriffe p und q, von denen mindestens eins ein Schreibzugriff ist, arbeiten auf demselben Objekt.
- *Menge der Konfliktbeziehungen*: $C(s) = \{ (p, q) \mid p \text{ steht vor } q \text{ und befinden sich im Konflikt} \}$
- $conf(s)$ bezeichnet die Menge der Konfliktbeziehungen in s nach Bereinigung von abgebrochenen Transaktionen.

Konfliktserialisierbarkeit (CSR)

Zwei Schedules s und s' sind *konfliktäquivalent*, auch $s \approx_c s'$ geschrieben falls:

- $op(s) = op(s')$
- $conf(s) = conf(s')$

Ein vollständiger Schedule s ist *konfliktserialisierbar*, falls ein serieller Schedule s' mit $s \approx_c s'$ existiert.
Man schreibt $s \in \text{CSR}$.

Wir definieren den *Konfliktgraph* $G(s) = (v, E)$ mit

- $V = commit(s)$
- $E = \{ (t, t') \mid t \neq t' \wedge \exists p \in t, \exists q \in t' : (p, q) \in conf(s) \}$

$$s \in \text{CSR} \Leftrightarrow G(s) \text{ ist azyklisch}$$

Durch CSR werden Lost Update und das Phantomproblem vermieden, nicht aber dirty read/writes.

"liest von"-Notation

In einem Schedule s heißt $p <_s q$, dass p vor q stattfindet. Wir sagen t_i *liest* x *von* t_j in s wenn:

- $w_j(x) <_s r_i(x)$
- $a_j \not<_s r_i(x)$
- $w_j(x) <_s w_k(x) <_s r_i(x) \rightarrow a_k <_s r_i(x)$

Rücksetzbarkeit (RC)

s ist *zurücksetzbar* wenn gilt:
t_i liest von t_j in s
$\wedge c_i \in s \rightarrow c_j <_s c_i$

verm. kaskadierende Aborts (ACA)

s vermeidet kaskadierende Aborts, falls:
t_i liest x von t_j in $s \rightarrow c_j <_s r_i(x)$

Striktheit (ST)

s ist strikt, falls:
$w_j(x) <_s p_i(x) \rightarrow$
$a_j <_s p_i(x) \vee c_j <_s p_i(x)$

Ein Schedule heißt *korrekt*, falls es in der Klasse CSR und in einer der Klassen RC, ACA oder ST liegt.

9.3 Protokolle zur Synchronisation

Mit *Sperren* synchronisieren wir Zugriffe auf Objekte. Ein Objekt wird vor einem Zugriff gesperrt und danach wieder freigegeben. Bei mehreren simultanen Transaktionen sind Lesesperren untereinander kompatibel, Schreibzugriffe jedoch nicht.

Lesesperren:

- $rl_i(x)$ (lock)
- $ru_i(x)$ (unlock)

Schreibsperren:

- $wl_i(x)$ (lock)
- $wu_i(x)$ (unlock)

Es gelten folgende Regeln für Sperren:

1. Falls t_i eine Aktion $p_i(x)$ enthält, dann davor (irgendwann) $pl_i(x)$ und danach (irgendwann) $pu_i(x)$.
2. Für jedes von t_i verwendete x existiert genau ein $rl_i(x)$ bzw. $wl_i(x)$.
3. Unlocks sind nicht redundant.

Für ein Schedule s bezeichnen wir mit $\mathrm{DT}(s)$ die Inklusion der Aktionen r, w, a und c.

2-Phasen Sperrprotokoll (2PL)

- Phase eins: *lock*-Aktionen
- Phase zwei: *unlock*-Aktionen

Nach der ersten *unlock*-Aktion einer Transaktion darf keine *lock*-Aktion mehr stattfinden. 2PL erfüllt CSR, auch genannt *s-sicher*.

Konservatives/statisches 2PL (C2PL)

Alle Sperren einer Transaktion werden vor dem ersten tatsächlichen Zugriff gesetzt.

- Kein Abbruch von Transaktionen.
- Alle Zugriffe müssen im voraus bekannt sein.
- Ggf. muss die Transaktion warten.

Strenges/dynamisches 2PL (S2PL)

Schreibsperren einer Transaktion werden erst nach dem letzten Zugriff aufgehoben.

- S2PL erfüllt CSR und ST und ist daher sicher.
- Werden auch Lesesperren gehalten sprich man von starken S2PL (SS2PL).

Nachteile von 2PL-Protokollen sind:

- Bei großen Objekten gibt es wenige Sperren zu verwalten, aber mehr Konflikte zwischen Transaktionen.
- Bei kleinen Objekten gibt es viele Sperren zu verwalten, dafür weniger Konflikte zwischen Transaktionen.

9.4 Recovery

Potenzielle Fehler:

- *Transaktionsfehler*: z. B. Abbruch der Anwendung, Verletzung von Zugriffsrechten, Rollback vom Benutzer, Transaktionskonflikt.
  Lösung: *Rücksetzen*: Den Anfangszustand vor der Transaktion sichern und wiederherstellen.
- *Systemfehler*: z. B. Stromausfall, Ausfall der CPU, Absturz des OS.
  Lösung: *Warmstart*: UNDO aller nicht abgeschlossenen Transaktionen.
- *Medienfehler*: Datenverlust durch Schaden am Speichermedium.
  Lösung: *Kaltstart*: Aufsetzen eines Backups. REDO aller bereits commiteten Transaktionen. UNDO aller nicht abgeschlossenen Transaktionen.

Techniken:

- *Duplizierung*: z. B. Bänder, Spiegelplatten, weitere Rechenzentren.
- *Logging*: Protokollierung aller Vorgänge.

9.5 Datenschutz

Schutz der Daten vor unberechtigter Manipulation. Maßnahmen:

- *Identifikation*: z. B. Passwort oder Personal.
- Schutz vor physischen Übergriffen: Diebstahl von Speichermedien, Anzapfen von Leitungen.
- *Verschlüsselung.*

DBS-spezifische Maßnahmen

- *DAC*: Sicherheitssubjekte (wer Zugriff hat), und Sicherheitsobjekte (auf welche Daten), dazu Rechte (Einfügen, Entfernen, Modifizieren).
  Datenerzeuger sind für Sicherheit verantwortlich, Granularität der Objekte erhöht stark die Verwaltung der Rechte.
- *MAC*: Sicherheitseinstufung von Daten (Sensitivität) und Personen (Vertrauenswürdigkeit) (streng geheim, unklassifiziert etc.). Zugriff erlaubt solange Vertrauenswürdigkeit $>$ Sensitivität.
  Potentiell höhere Sicherheit, Zusammenarbeit von Benutzern mit unterschiedlichen Vertrauenswürdigkeiten schwer.