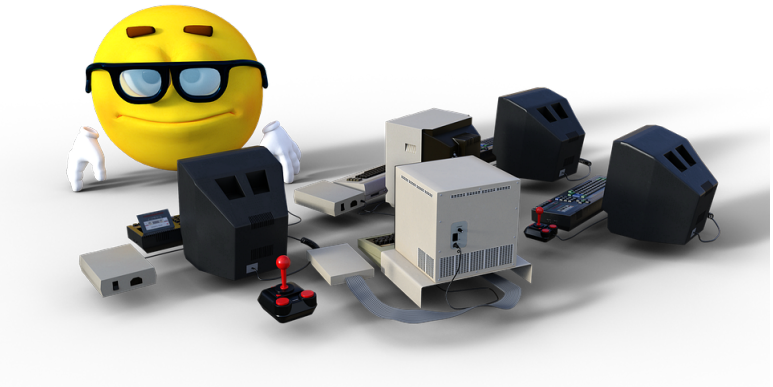
Westfälische Hochschule

DBA

Alles wichtige auf einen Blick

DBA Semesterzusammenfassung



Author(s):
-

Supervisor(s):
Prof. -

4. November 2025

Vorwort

Dieses Dokument ist zu der Vorlesung aus dem Jahr 2025/26 entstanden. Inhalte können im Vergleich zu anderen Jahren abweichen. Es wird keine Garantie auf Richtigkeit der Inhalte gegeben. Dieses Dokument ist eine ergänzung für die Vorlesungen und kein vollständiger ersatz.

Inhaltsverzeichnis

Teil I

Semesterzusammenfassung

1. Motivation und Grundlagen von Datenbanken

1.1 Motivation: Warum Datenbanken?

Daten können in einer Textdatei gespeichert werden. Wird dies getan, gibt es einige Vor- und Nachteile, die in Tabelle 1.1 gegenübergestellt sind.

Tabelle 1.1: Vor- und Nachteile der Datenspeicherung in reinen Textdateien

Vorteile	**Nachteile**
• Einfach verfügbar und verständlich • Einfach zugreifbar, portabel • Wenig Overhead-Daten • Große Kontrolle über die Struktur (Exportierbarkeit) • Keine (DBMS-spezifischen) Fehlermeldungen	• Unstrukturierte Daten (z.B. 70 Schauspieler in einer Spalte) • Analysemethoden stark eingeschränkt • Änderungen an gleichen Daten schwierig • Keine Integritätssicherung (z.B. Datum "35.12.2017"möglich) • Keine Datenkontrolle oder Typisierung • Kein Mehrbenutzerbetrieb (Concurrent Access)

1.2 Historische Entwicklung von Datenbanken

Die Nachteile von einer einfachen Textdatei sind offensichtlich. Aus diesem Grund wurde Ende der 60er Jahre ein Dateiverwaltungssystem eingeführt. Hierdurch waren Datenbanken zwar geräteunabhängig, allerdings gab es keinen Schutz vor Redundanz und Inkonsistenz.

Aus diesem Grund wurde über die gesamte Datenbank ein Datenbankmanagementsystem (DBMS) eingebaut. Daten und Metadaten sind auf der Datenbank gespeichert. Das Datenbankmanagementsystem ist softwarebasiert und liegt auf einem Server.

Abbildung 1.1: Entwicklungsschritt 1: Dateiverwaltungssystem (vgl. Folie 5)

Abbildung 1.2: Entwicklungsschritt 2: Datenbankmanagementsystem (DBMS) (vgl. Folie 6)

1.2.1 Moderne Architekturen (Nicht klausurrelevant)

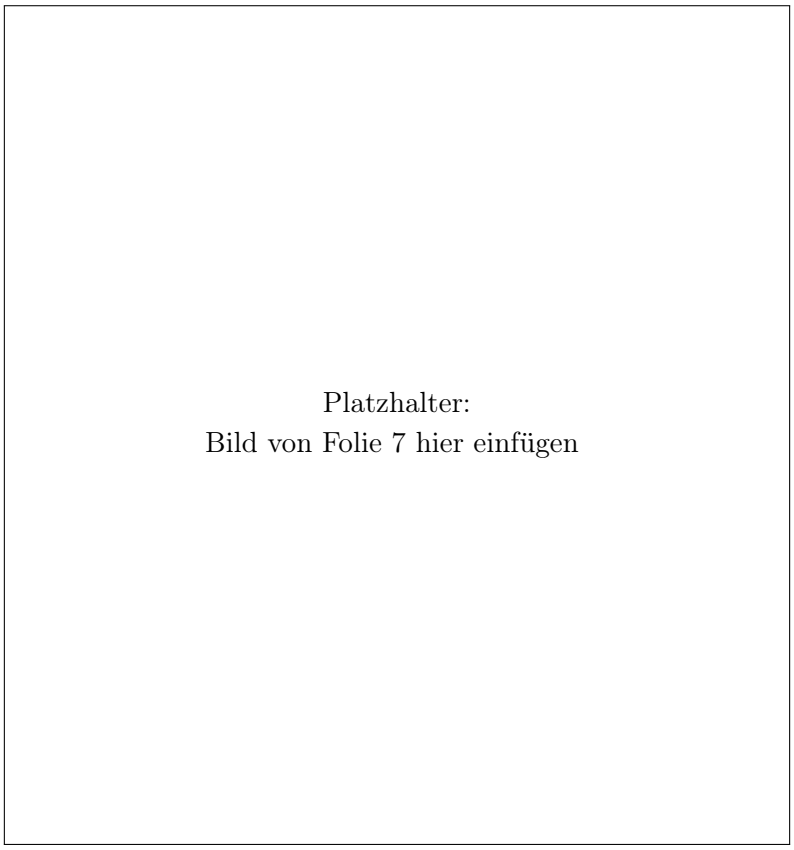


Abbildung 1.3: Moderne Datenbankarchitekturen (vgl. Folie 7)

Es gibt mehrere Datenbanken mit jeweils einem eigenen Datenbankmanagementsystem, welche über eine Cloud oder On-Premise-Systemen verbunden sind. Die Anwendungen kommunizieren dann nicht direkt mit der Datenbank, sondern mit diesen Schnittstellen bzw. "Datentöpfen". Die Datentöpfe müssen keine konsistenten Daten haben. Unter Umständen können andere Datenbanken (wie z.B. Cassandra) zwischen den Datenbanken und den Schnittstellen geschaltet werden, um die Geschwindigkeit zu erhöhen.

1.3 Grundlegende Merkmale von Datenbanken

Daten werden in einer Datenbank in einem **logischen Datenmodell** und mit einem **Datenschema** gespeichert. Es stehen **Operationen zur Datenmanipulation** zur Verfügung.

Wichtige Merkmale von Daten in einem DBMS sind:

- **Persistenz**: Daten bleiben dauerhaft gespeichert.
- **Konsistenz**: Daten sind in sich widerspruchsfrei.
- **Effizienz**: Schneller Zugriff und effiziente Speicherung.

1.4 Aufgaben eines DBMS nach Codd

Um die Funktionalität zu gewährleisten, haben sich im Laufe der Jahre Basisfunktionen herausgestellt, welche von einem DBMS erfüllt sein müssen.

1.4.1 Datenintegration

Es wird ein **logisches Datenmodell** benutzt, um den Inhalt der Datenintegration für verschiedene Anwendungen zu beschreiben und zu kommunizieren.

Es gibt einige verschiedene Datenmodelle:

Hierarchisches Modell

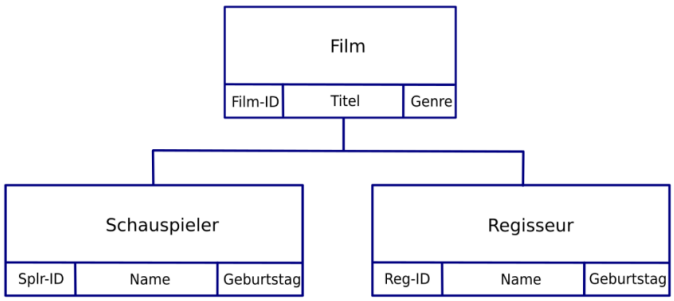


Abbildung 1.4: Beispiel eines hierarchischen Datenmodells

Dieses Modell ist vergleichbar mit einem Baum. Es sind nur einfache 1:n-Beziehungen möglich.

Netzwerkmodell

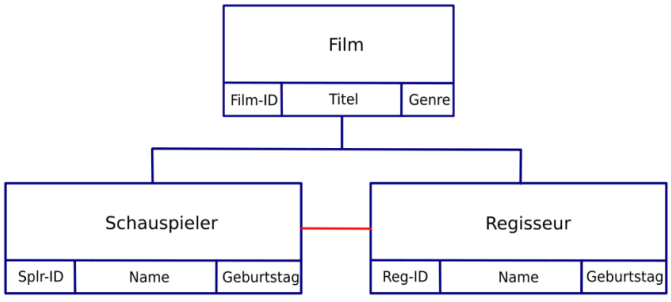


Abbildung 1.5: Beispiel eines Netzwerkmodells

Es ist keine strenge Hierarchie erforderlich. Es sind auch komplexe n:m-Beziehungen erlaubt.

Relationales Modell

Objekte und Beziehungen werden in Tabellen modelliert. Informationen in den Tabellen können mit Operationen ausgewertet werden.

XML-Modell

Objekte und Beziehungen werden in einer Baumstruktur modelliert. Diese kann durch Schemata beschrieben werden.

Objektorientiertes Modell

Daten und Funktionen werden in einem Objekt gespeichert. Selten in der Praxis genutzt.

1.4.2 Operationen und Anfragesprachen

Operationen und Abfragesprachen werden zur Datenverarbeitung verwendet und müssen auf reale Daten mit mehreren Milliarden Zeilen und Tausenden Tabellen anwendbar sein. Der Standard für relationale Datenbanken ist SQL.

1.4.3 Datenkatalog

1.4.4 Benutzersichten

1.4.5 Konsistenz

1.4.6 Zugriffskontrolle

1.4.7 Transaktionskonzept

1.4.8 Synchronisation

1.4.9 Datensicherung

1.5 DBMS im Software Stack

2. SQL

2.1 CREATE TABLE

Tabellen können über den `CREATE TABLE`-Befehl erstellt werden.

> `CREATE TABLE` *tabellenname* (
> *spaltenname_1 datentyp_1* [`NOT NULL`],
> *spaltenname_n datentyp_n* [`NOT NULL`],
> [`PRIMARY KEY` (*spalte_a, spalte_b, ...*)],
> [`FOREIGN KEY` (*spalte_c*) `REFERENCES`
> *andere_tabelle*(*spalte_d*), ...]
>)

Zwischen eckigen Klammern `[]` stehende Teile sind optional.

2.1.1 Beispiel 1

Listing 2.1: Beispiel 1: Erstellung der Tabelle 'film'

```
1 CREATE TABLE film (
2     fid INTEGER NOT NULL,
3     titel VARCHAR(50),
4     erscheinungsdatum DATE,
5     genre VARCHAR(30),
6     PRIMARY KEY (fid)
7 );
```

2.1.2 Beispiel 2

Listing 2.2: Beispiel 2: Zusammengesetzter Primärschlüssel

```
1 CREATE TABLE film2 (
2     titel VARCHAR(50),
3     erscheinungsdatum DATE,
4     genre VARCHAR(30),
5     PRIMARY KEY (titel, erscheinungsdatum)
6 );
```

(`titel`, `erscheinungsdatum`) ist ein zusammengesetzter Primärschlüssel. In dieser Modellierung können keine zwei Filme mit dem gleichen Titel am selben Tag erscheinen.

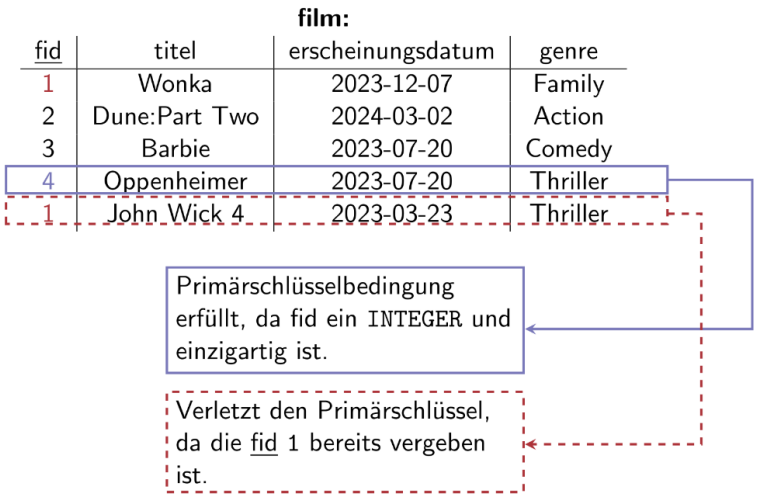


Abbildung 2.1: Resultierende Tabellenstruktur für 'film'

2.1.3 Schlüsselgenerierung

Schlüssel können durch verschiedene Strategien generiert werden:

GENERATED AS IDENTITY

Varianten:

- `GENERATED ALWAYS AS IDENTITY` (Immer generiert, manuelles Einfügen verboten)
- `GENERATED BY DEFAULT AS IDENTITY` (Standardmäßig generiert, manuelles Einfügen erlaubt)

Listing 2.3: Schlüsselgenerierung mit `IDENTITY`

```
1 CREATE TABLE film (
2     fid INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
3     titel TEXT,
4     ...
5 );
```

Hinweis: Bei `BY DEFAULT` kann manuell ein Wert gesetzt werden.

Manuelle Sequenzen

Volle Kontrolle über Startwert und Inkrement:

Listing 2.4: Manuelle Sequenzenerstellung und -nutzung

```
1 CREATE SEQUENCE user_id_seq START 100;
2
3 CREATE TABLE schauspieler (
4     sid INT PRIMARY KEY DEFAULT nextval('user_id_seq'),
5     vorname TEXT,
6     ...
7 );
```

Vorteil: Flexibel, z. B. anpassbare Startwerte.
Nachteil: Verwaltung der Sequenz liegt in der Hand des Entwicklers.

UUIDs

Globale Eindeutigkeit ohne Sequenzen (benötigt ggf. Erweiterung):

Listing 2.5: UUIDs als Primärschlüssel (PostgreSQL)

```
1  CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
2
3  CREATE TABLE schauspieler (
4      sid UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
5      name TEXT,
6      ...
7  );
```

Vorteile: Keine Kollisionen bei verteilten Systemen, kein zentraler Zähler nötig.
Nachteil: IDs sind länger (16 Bytes) und schwerer lesbar.

Eigene Generatoren / Trigger

Komplexe ID-Logik mit Funktionen (z.B. in `plpgsql`) möglich.

Listing 2.6: Eigener ID-Generator mit `plpgsql` (PostgreSQL)

```
1  CREATE SEQUENCE film_seq;
2
3  CREATE FUNCTION generate_film_id() RETURNS TEXT AS $$
4  BEGIN
5      RETURN 'FID-' || to_char(NOW(), 'YYYYMMDD')
6             || '-' || nextval('film_seq');
7  END;
8  $$ LANGUAGE plpgsql;
9
10 CREATE TABLE film (
11     fid TEXT PRIMARY KEY DEFAULT generate_film_id(),
12     titel TEXT,
13     ...
14 );
```

2.1.4 Beispiel 3: Fremdschlüssel

Listing 2.7: Beispiel 3: Definition von Fremdschlüsseln

```
1  CREATE TABLE spielt_in (
2      fid INTEGER NOT NULL,
3      sid INTEGER NOT NULL,
4      FOREIGN KEY (fid) REFERENCES film(fid),
5      FOREIGN KEY (sid) REFERENCES schauspieler(sid)
6  );
```

Ein Fremdschlüssel verknüpft Datensätze zwischen Tabellen. Er verweist auf den Primärschlüssel einer anderen Tabelle und stellt sicher, dass keine Datensätze verwaisen (Referenzielle Integrität).

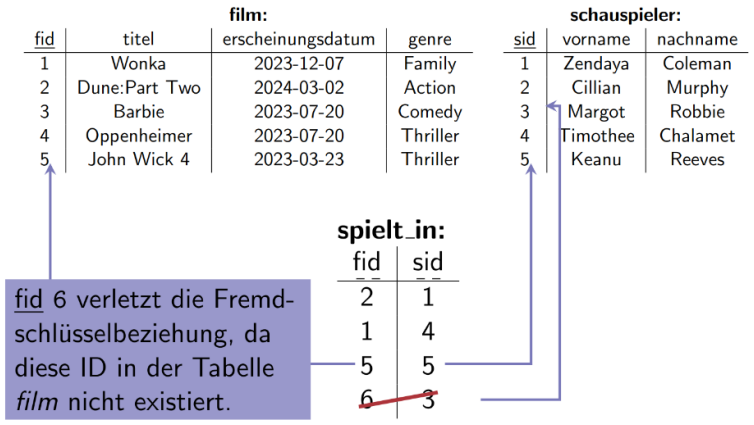


Abbildung 2.2: Visualisierung eines Fremdschlüssels

Referenzielle Aktionen

Definieren, was bei Lösch- oder Update-Operationen passieren soll:

Listing 2.8: Referenzielle Aktionen definieren

```
1 FOREIGN KEY (kunde_id) REFERENCES kunden(id)
2     ON DELETE CASCADE
3     ON UPDATE SET NULL;
```

Wichtige Varianten:

ON DELETE CASCADE Löscht abhängige Datensätze automatisch mit.

ON DELETE SET NULL Setzt Fremdschlüssel in abhängigen Datensätzen auf `NULL`.

ON DELETE RESTRICT Verhindert die Löschung, wenn Abhängigkeiten bestehen (Standard).

ON UPDATE CASCADE Aktualisiert die Fremdschlüsselwerte, wenn der referenzierte Schlüssel sich ändert.

2.1.5 Datentypen

PostgreSQL unterstützt eine Vielzahl von Datentypen.

Numerische Typen

SMALLINT 2 Byte, von -32.768 bis 32.767

INTEGER 4 Byte, Standard-Ganzzahl

BIGINT 8 Byte, für sehr große Ganzzahlen

REAL 4 Byte Fließkommazahl (einfache Genauigkeit)

DOUBLE PRECISION 8 Byte Fließkommazahl (doppelte Genauigkeit)

NUMERIC(p, s) Exakte Dezimalzahl (z.B. `NUMERIC(10, 2)`: 10 Stellen gesamt, 2 Nachkommastellen). Ideal für Geldbeträge.

Zeichenketten

CHAR(n) Feste Länge (wird mit Leerzeichen aufgefüllt)

VARCHAR(n) Variable Länge, auf n Zeichen begrenzt

TEXT Variable, unbegrenzte Länge

Datums- und Zeit-Typen

DATE Nur Datum, z.B. `'2025-11-04'`

TIME Nur Uhrzeit, z.B. `'08:30:00'`

TIMETZ Uhrzeit mit Zeitzone, z.B. `'08:30:00+01'`

TIMESTAMP Datum und Uhrzeit, z.B. `'2025-11-04 08:30:00'`

TIMESTAMPTZ Datum und Uhrzeit mit Zeitzone (bevorzugt für globale Anwendungen)

INTERVAL Zeitspanne, z.B. `'1 day'`, `'3 hours'`

Sonstige Typen

BOOLEAN `TRUE`, `FALSE` oder `NULL`

UUID Universally Unique Identifiers

INET IP-Adressen (IPv4 oder IPv6)

BYTEA Binärdaten (z.B. Bilder, Dateien)

Strukturierte Datentypen

Arrays Jeder Datentyp kann als Array gespeichert werden, z.B. `INT[]`.

JSON/JSONB Speicherung von strukturierten Daten. `JSONB` ist die binäre, indizierbare Variante und wird meist bevorzugt.

Benutzerdefinierte Datentypen

Enumerations (ENUM) `CREATE TYPE status AS ENUM ('neu', 'aktiv', 'archiviert');`

Domänen (Domains) `CREATE DOMAIN positive int AS INT CHECK (VALUE > 0);`

Zusammengesetzte Typen `CREATE TYPE adresse AS (strasse TEXT, plz INT);`

Vorteil: Eigene Typen fördern Konsistenz und Wiederverwendung.

2.2 DROP TABLE

Tabellen können über `DROP TABLE` gelöscht werden.

```
1 DROP TABLE tabellenname;
```

Der Befehl wird zurückgewiesen, wenn noch Abhängigkeiten (Sichten, Fremdschlüssel) existieren. Wird eine Tabelle gelöscht, gehen auch alle Daten in der Tabelle unwiderruflich verloren.

2.3 ALTER TABLE

Tabellen können über `ALTER TABLE` *`tabelle modifikation`* geändert werden.

Als *modifikation* stehen Anweisungen zur Verfügung wie:

- `ADD COLUMN` *`spaltendefinition`*
- `DROP COLUMN` *`spaltenname`*
- `ALTER COLUMN` *`spaltenname typ ...`*
- `RENAME COLUMN` *`alt`* `TO` *`neu`*
- `ADD CONSTRAINT ...`

Listing 2.9: Beispiel 1: Spalte hinzufügen

```
1 ALTER TABLE schauspieler
2 ADD COLUMN geburtsdatum DATE;
```

Listing 2.10: Beispiel 2: Spalte mit Standardwert hinzufügen

```
1 ALTER TABLE schauspieler
2 ADD COLUMN hat_oskar BOOLEAN NOT NULL DEFAULT false;
```

2.4 INSERT

Fügt neue Datensätze (Zeilen) in eine Tabelle ein.

Listing 2.11: Syntax: INSERT

```
1 INSERT INTO tabelle (attribut_1, ..., attribut_n)
2 VALUES (attributwert_1, ..., attributwert_n);
```

Attribute müssen in derselben Reihenfolge wie die Werte angegeben werden.

Listing 2.12: Beispiel: INSERT

```
1 INSERT INTO film (fid, titel, erscheinungsdatum, genre)
2 VALUES (6, 'Kung␣Fu␣Panda␣4', '2024-03-14', 'Family');
```

2.5 UPDATE

Bearbeitet Zeilen, die einer Bedingung entsprechen.

Listing 2.13: Syntax: UPDATE

```
1 UPDATE tabelle
2 SET spalte1 = value1, spalte2 = value2, ...
3 WHERE bedingung;
```

Listing 2.14: Beispiel: UPDATE

```
1 -- Erhöht die Dauer aller Filme um 10 Minuten
2 UPDATE film
3 SET dauer = dauer + 10;
```

2.6 DELETE

Löscht Zeilen, die einer Bedingung entsprechen.

Listing 2.15: Syntax: DELETE

```
1 DELETE FROM tabelle
2 WHERE bedingung;
```

Listing 2.16: Beispiel: DELETE

```
1 DELETE FROM film
2 WHERE dauer > 120;
```

Achtung: Wird keine `WHERE`-Bedingung angegeben, werden **alle** Zeilen aus der Tabelle gelöscht!

2.7 SELECT

Über die `SELECT`-Anweisung können Daten aus der Datenbank gelesen (äbgefragt") werden. Im Allgemeinen ist der `SELECT`-Befehl aufgeteilt in:

- **SELECT** Projektionsliste (Spalten, die angezeigt werden sollen)
- **FROM** Gibt die verwendeten Tabellen an
- **WHERE** Filtert die Zeilen basierend auf Bedingungen

2.7.1 Beispiel

Eine einfache `SELECT`-Anweisung:

Listing 2.17: Allgemeine SELECT-Struktur

```
1 SELECT A1, ..., An
2 FROM R1, ..., Rm
3 WHERE C;
```

Mit einem Sternchen (`*`) können alle Spalten aus der Datenbank geholt werden.

2.7.2 DISTINCT

`DISTINCT` entfernt alle Duplikate aus der Ergebnismenge.

```
1 SELECT DISTINCT fid
2 FROM spielt_in;
```

2.7.3 WHERE

Über das `WHERE` können Bedingungen gesetzt werden, welche erfüllt werden müssen. Typische Operatoren:

- Logische Vergleiche: =, <> (oder `!=`), <, >, <=, >=
- Bereichsprüfungen: `BETWEEN a AND b`
- Wertelisten: `IN (...)`, `NOT IN (...)`
- Mustervergleiche: `LIKE` (mit `%` als Platzhalter)
- Null-Prüfung: `IS NULL` / `IS NOT NULL`

Listing 2.18: Beispiele für WHERE-Bedingungen

```
1 WHERE name LIKE 'A%'; -- Beginnt mit A
2 WHERE ort IN ('Berlin', 'Hamburg');
3 WHERE datum BETWEEN '2025-01-01' AND '2025-12-31';
4 WHERE aktiv IS TRUE;
```

Bedingungen können mit `AND`, `OR` und `NOT` kombiniert werden.

2.7.4 Aliase

Zur besseren Lesbarkeit können in einer SQL-Anfrage Aliase (Umbenennungen) verwendet werden:

Listing 2.19: Aliase für Tabellennamen

```
1 SELECT DISTINCT titel, rid
2 FROM film AS f, fuehrt_Regie AS r
3 WHERE f.fid = r.fid;
```

Haben in einer Abfrage mehrere Tabellen Spalten mit dem selben Namen (z.B. `genre`), muss durch Voranstellen des Alias (z.B. `f.genre`) explizit angegeben werden, welche Spalte gemeint ist.

Aliase können auch für Spalten, arithmetische Ausdrücke oder Konstanten verwendet werden:

Listing 2.20: Spalten-Aliase

```
1 SELECT fid, titel,
2        dauer + 10 AS kinolaenge,
3        '3D' AS vorstellung
4 FROM film;
```

Mögliche Ergebnismenge (formatiert als `verbatim`):

```
 fid | titel          | kinolaenge | vorstellung
-----+----------------+------------+-------------
 1   | Wonka          | 127        | 3D
 2   | Dune:Part Two  | 176        | 3D
 3   | Barbie         | 124        | 3D
 4   | Oppenheimer    | 190        | 3D
 5   | John Wick 4    | 179        | 3D
```

2.7.5 WHERE-Klausel (Impliziter JOIN)

In der `WHERE`-Klausel können Bedingungen angegeben werden, die bestimmte Tupel aus den selektierten Tabellen filtern.

Listing 2.21: Veralteter ïmpliziter"JOIN-Stil

```
1 SELECT DISTINCT titel, rid
2 FROM film AS f, fuehrt_Regie AS r
3 WHERE f.fid = r.fid;
```

Wird keine `WHERE`-Klausel (oder `JOIN ON`) angegeben, wird das **Kartesische Produkt** (jede Zeile aus Tabelle A mit jeder Zeile aus Tabelle B) ausgewertet.

2.7.6 JOIN (Expliziter JOIN)

Die moderne und bevorzugte Art, Tabellen zu verknüpfen:

Listing 2.22: Expliziter JOIN (bevorzugt)

```
1 SELECT * FROM film AS f
2 JOIN fuehrt_regie AS r ON f.fid = r.fid;
```

Die Verwendung einer `JOIN ...ON`-Bedingung ist lesbarer und trennt Verknüpfungs-Logik von der Filter-Logik (die in `WHERE` bleibt).

2.7.7 IN und NOT IN

Über `IN` und `NOT IN` wird getestet, ob ein Wert (nicht) in einer (Unter-)Abfrage oder Liste enthalten ist.

Listing 2.23: NOT IN mit Unterabfrage

```
1 -- Findet alle Schauspieler, die in keinem Film spielen
2 SELECT sid, vorname, nachname
3 FROM schauspieler
4 WHERE sid NOT IN (
5     SELECT sid FROM spielt_in
6 );
```

Es werden also alle Schauspieler ausgegeben, die in keinem Film spielen.

2.8 Unterabfragen

Eine Unterabfrage (Subquery) ist eine SQL-Abfrage, die in eine andere Abfrage (Hauptabfrage) verschachtelt ist. Diese können oft durch `JOIN`s ersetzt werden, was meist performanter ist.

Listing 2.24: Verschachtelte Unterabfragen

```
1 -- Findet alle Kunden, deren Gesamtsumme über dem ↩
     Durchschnitt liegt
2 SELECT k.name
3 FROM kunden k
4 WHERE (
```

```
5       -- 1. Berechne Summe pro Kunde (korrelierte ↩
            Unterabfrage)
6       SELECT SUM(b.preis)
7       FROM besuche b
8       WHERE b.kunde_id = k.id
9   ) > (
10      -- 2. Berechne Durchschnitt aller Kundensummen
11      SELECT AVG(summe)
12      FROM (
13          -- 2a. Berechne Summe für JEDEN Kunden
14          SELECT SUM(b2.preis) AS summe
15          FROM besuche b2
16          GROUP BY b2.kunde_id
17      ) AS durchschnitts_tabelle
18  );
```

3. Datenbankentwurf

3.1 Datenbankentwurf Phasenmodell

3.1.1 Vorbereitung

Zuallererst müssen sich einige Fragen gestellt werden. Die zentrale Frage ist: *Welche Daten müssen gespeichert werden?* Darüber hinaus sind weitere Überlegungen wichtig:

- Wie sieht die Kosten-Nutzen-Analyse der Planung aus?
- Daten sollten möglichst effizient gespeichert werden.
- Es sollten nur notwendige Daten gespeichert werden (keine Redundanzen).
- Das Datenmodell muss zur Anwendung passen.

3.1.2 Phasenmodell

Das Phasenmodell beschreibt eine Abfolge von Entwurfsdokumenten. Jede Phase soll die existierenden Informationen erhalten bzw. konsistent zu den anderen Phasen sein.

Anforderungsanalyse

Anforderungen an den Endanwender (Kunden) werden gesammelt und analysiert. Das Fachproblem soll informell beschrieben werden.

Konzeptioneller Entwurf

Eine formale Beschreibung des Datenmodells. Es soll also ein ERM (Entity-Relationship-Modell) erstellt werden. Am Ende soll ein integriertes Datenmodell (Datenschema) entstehen.

Verteilungsentwurf

Dieser Schritt ist relevant, falls die Datenbank später auf mehreren Knoten (Servern) verteilt sein soll. (Wird hier nicht behandelt).

Logischer Entwurf

Transformation des konzeptionellen Entwurfs in die passende technische Beschreibung (Datenbankschema).

Datendefinition

Der noch systemunabhängige logische Entwurf wird in eine konkrete Deklaration mithilfe einer Datendefinitionssprache (DDL) überführt, z.B. mit SQL.

Hier werden auch Integritätsbedingungen (wie Schlüssel, Fremdschlüssel, etc.) und Benutzersichten angelegt.

Physischer Entwurf

Entwurf der physischen Speicherstrukturen der Datenbank. Die Optimierung (z.B. Indizierung) erfolgt meist automatisch über das DBMS.

Implementierung und Wartung

Realisierung auf der Datenbank sowie die kontinuierliche Anpassung an neue Anforderungen und Updates.

3.2 Konzeptioneller Entwurf - Das Entity-Relationship-Modell (ERM)

Ein ERM soll die Daten der Datenbank auf einem hohem Abstraktions-Level beschreiben. Die Notation muss hierbei unabhängig von einem spezifischen DBMS sein. Ziele des ERM:

- Kunden sollen einfach Anforderungen kommunizieren können.
- Das Modell soll sich einfach in ein konkretes DBMS übertragen lassen.

3.2.1 Darstellung

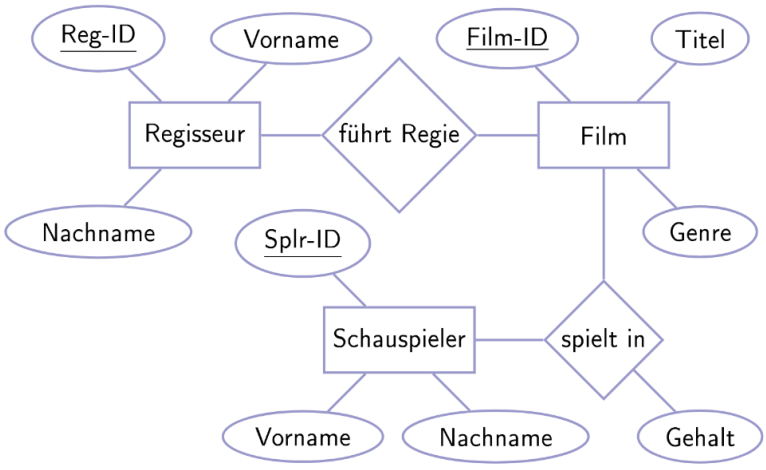


Abbildung 3.1: Grundlegende Notation eines Entity-Relationship-Modells (ERM)

Eine **Entität** ist ein Objekt in der realen Welt. Eine Menge gleichartiger Objekte wird zu einer **Entitätsmenge** zusammengefasst. Ein **Entitätstyp** wird in einem Rechteck dargestellt.

Eine Entität kann ein oder mehrere **Attribute** haben. Ein Attribut wird durch eine Ellipse dargestellt und ist mit dem Entitätstypen verbunden. Zu jedem Attribut gehört eine Domäne, die die erlaubten Werte des Attributes beschreibt.

Beziehungen zwischen zwei oder mehr Entitätstypen werden durch eine Raute dargestellt. Auch diese können Attribute besitzen.

Eine Beziehung kann zwischen Entitäten des gleichen Entitätstyps bestehen (rekursive Beziehung). Genauso können mehrere unterschiedliche Beziehungen zwischen denselben zwei Entitätstypen existieren.

Außerdem kann es **n-äre Beziehungen** geben, d.h. eine Beziehung kann zwischen beliebig vielen Entitätstypen existieren.

3.3 Schlüsselbeziehungen

Um Informationen zu verarbeiten, müssen einzelne Objekte (Entitäten) in diesen Mengen eindeutig identifizierbar sein.

Entitäten können durch einzelne Attribute als auch durch eine Kombination mehrerer Attribute eindeutig identifiziert werden. Eine solche Teilmenge wird auch **Schlüssel** genannt.

Eine **minimale** Menge an Attributen, über die eine Entität identifiziert werden kann, wird **Schlüsselkandidat** genannt. Handelt es sich nicht um eine minimale Menge, ist es ein **Superschlüssel**. Im ERM wird der Primärschlüssel meist durch Unterstreichen gekennzeichnet.

3.4 Kardinalitäten, Schwache Entitäten, IST-Beziehung

3.4.1 Beziehungen

Beziehungen benutzen keine eigenen eindeutigen Schlüssel, stattdessen werden die Schlüssel der verknüpften Entitäten verwendet.

In einem ERM werden Kardinalitäten (Multiplizitäten) oft durch ein `[min, max]`-Intervall an der Beziehung notiert.

Beispiel

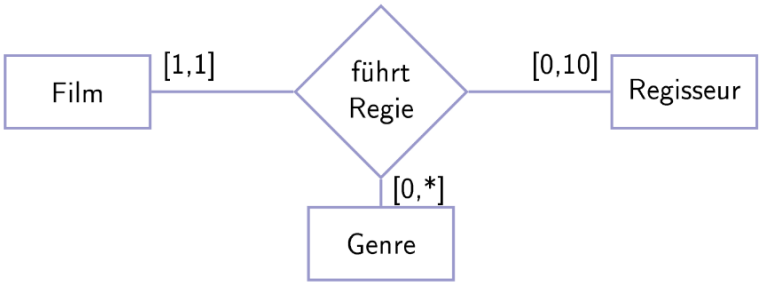


Abbildung 3.2: Beispiel für Kardinalitäten in einem ERM

Ein Regisseur arbeitet an 0 bis 10 Filmen (`[0, 10]`) und ein Film wird von mindestens einem und maximal 3 Regisseuren produziert (`[1, 3]`). Einem Genre sind beliebig viele Filme zugeordnet (`[0, n]`).

Schwache Entitäten

Schwache Entitäten (Weak Entities) besitzen keinen eigenen vollständigen Primärschlüssel. Stattdessen wird ihr identifizierendes Attribut (Partial Key) um den Primärschlüssel der ßtarkenËntität, von der sie abhängig sind, ergänzt.

3.4.2 IST-Beziehung (Generalisierung/Spezialisierung)

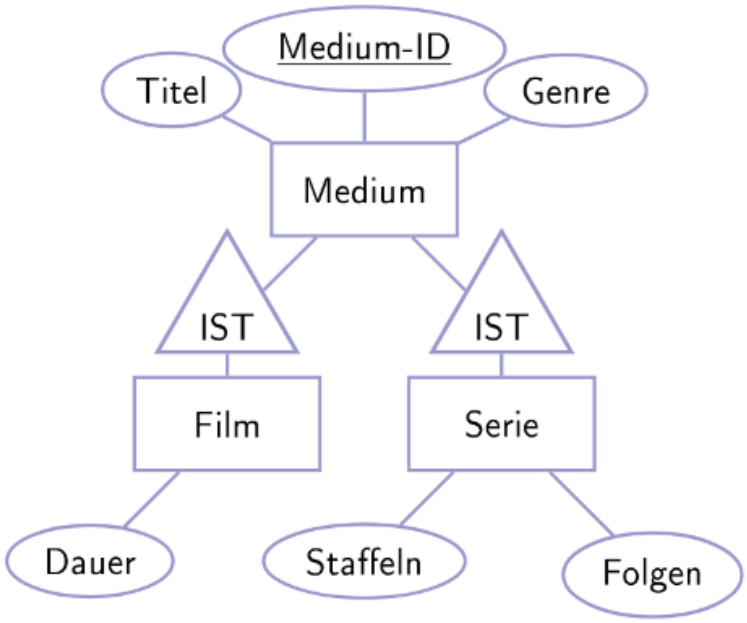


Abbildung 3.3: Beispiel einer IST-Beziehung (Generalisierung)

Die IST-Beziehung (Generalisierung) ist keine eigene festgelegte Notation im Standard-ERM (nach Chen), wird aber oft so oder ähnlich (z.B. als Dreieck) dargestellt. Alle Filme und Serien ßind ein"Medium und besitzen einen Datensatz der Entität Medium. Dies kann später durch Fremdschlüssel modelliert werden.

3.5 Logischer Entwurf - Vom ERM zum Datenbankschema

In dieser Phase wird das konzeptionelle ERM systematisch in ein relationales Datenbankschema (eine Sammlung von Tabellendefinitionen) überführt.

3.6 Relationen und Tabellen

Eine Relation (im relationalen Modell) bzw. eine Tabelle (in SQL) besteht aus zwei Teilen:

3.6.1 Schema

Der Schemateil besteht aus der Beschreibung der Struktur:
sch(Film) = (Film-ID, Titel, Genre)

Jedes Attribut besitzt außerdem eine Domäne: *dom(Film-ID) = INTEGER*

In der Datenbank gehören außerdem Schlüsselbeziehungen (Primary Key, Foreign Key) zum Schema.

3.6.2 Wert / Instanz

Als Wert oder Instanz *val(Film)* der Relation bezeichnen wir die aktuelle Tupelmenge (Zeilen), die in der Tabelle enthalten ist.

Eigenschaften einer Relation:

- Die Reihenfolge der Tupel (Zeilen) ist nicht definiert.
- Eine Relation kann nicht das gleiche Tupel mehrfach enthalten (Mengenlehre).
- Die Attribute der Tupel können nur aus atomaren (unteilbaren) Werten bestehen (1. Normalform).

3.7 Fremdschlüssel, Null-Werte, Informationskapazität

3.7.1 Fremdschlüssel

Ein Fremdschlüssel verweist eindeutig auf einen Schlüsselkandidaten (meist den Primärschlüssel) einer anderen (oder derselben) Tabelle.

Sie werden wie folgt definiert:

Listing 3.1: Definition von Fremdschlüsseln

```
1 CREATE TABLE spielt_in (
2     Film_ID INTEGER NOT NULL,
3     Splr_ID INTEGER NOT NULL,
4     Gehalt CHAR(15),
5     PRIMARY KEY (Film_ID, Splr_ID),
6     FOREIGN KEY (Film_ID) REFERENCES Film(Film_ID),
7     FOREIGN KEY (Splr_ID) REFERENCES Schauspieler( ↩
          Splr_ID)
8 );
```

3.7.2 Null-Werte

Werte können auch `NULL` sein (unbekannt oder nicht vorhanden). Durch die Angabe von `NOT NULL` wird dies verboten. Ein `NULL`-Wert kennzeichnet, dass ein Wert nicht bekannt ist. Die Behandlung von `NULL`-Werten ist anwendungs- und systemspezifisch.

Bedeutungen von NULL

- **Wert existiert, aber ist nicht bekannt:** Das Gehalt eines Schauspielers.
- **Kein Wert existiert (nicht zutreffend):** Ein Film ist auf einer Streaming-Plattform, oder eben nicht.
- **Attribut ist nicht anwendbar:** In der Tabelle "Medium"(siehe IST-Beziehung) besitzt nur eine Serie Staffeln, ein Film jedoch nicht.

Verhalten von NULL in SQL

- In arithmetischen Operationen ist das Ergebnis (fast) immer `NULL`. (z.B. `0 + NULL = NULL`)
- In Vergleichen ist das Ergebnis (fast) immer `NULL` (unbekannt), nicht `TRUE` or `FALSE`. (z.B. `(Dauer < NULL) = NULL`)
- In der Logik (mit `AND`, `OR`, `NOT`) ist das Ergebnis nicht immer `NULL`. (z.B. `TRUE OR NULL = TRUE`)

3.7.3 Informationskapazität

Welche Möglichkeiten für Primärschlüssel gibt es und wie wirken sie sich aus?

Beispiel 1:

Film-ID	Reg-ID
4	1
3	3

1. Nur FilmID: Filme können nur noch einen Regisseur haben. ⇒ **Kapazitätsvermindernd**
2. FilmID, RegID: **Kapazitätserhaltend**

Beispiel 2:

Film-ID	Titel	Genre
1	Wonka	Family
2	Dune:Part Two	Action
3	Barbie	Comedy
4	Oppenheimer	Thriller
5	John Wick 4	Thriller

1. Nur FilmID: **Kapazitätserhaltend**
2. FilmID und Genre: **Kapazitätserhöhend** (Ein Film könnte nun mit mehreren Genres existieren, z.B. (2, Dune, Action) und (2, Dune, Sci-Fi))

Zwei Tabellen mit dem selben Schlüssel können zusammengelegt werden:

Tabelle: Schauspieler

Splr-ID	Vorname	Nachname
1	Zendaya	Coleman
2	Cillian	Murphy
3	Margot	Robbie
4	Timothee	Chalamet
5	Keanu	Reeves

Tabelle: Geburtsdaten

Splr-ID	Geburtsdatum
1	1996-09-01
2	1976-05-25
3	1990-07-02
4	1995-12-27
5	1964-09-02

Zusammengelegt zu **Tabelle: Schauspielerstammdaten**:

Splr-ID	Vorname	Nachname	Gebu
1	Zendaya	Coleman	1996-
2	Cillian	Murphy	1976-
3	Margot	Robbie	1990-
4	Timothee	Chalamet	1995-
5	Keanu	Reeves	1964-

3.8 Abbilden von ERMs auf Tabellen

Bezogen auf die IST-Beziehung (Generalisierung) aus Abbildung 3.3 gibt es mehrere Abbildungsstrategien:

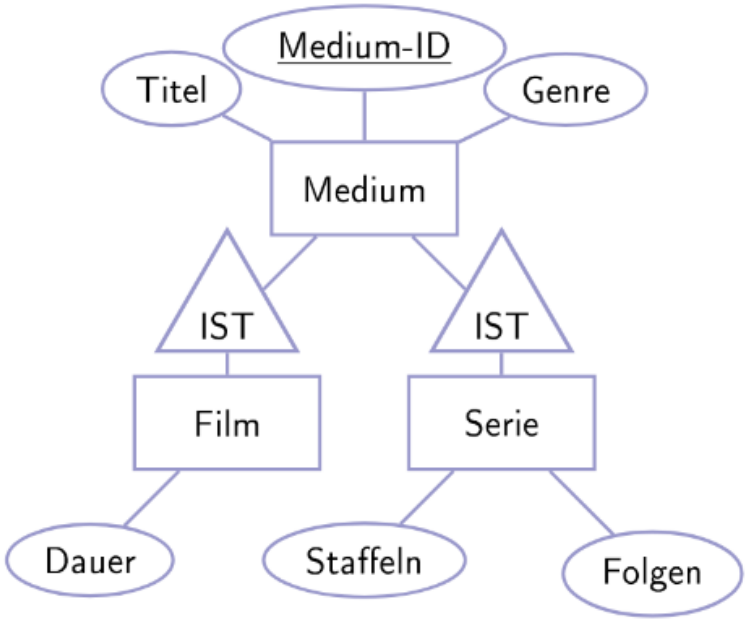


Abbildung 3.4: Abbildungsstrategien für eine IST-Beziehung

Horizontale Strategie (Table per Subclass)

```
Medium(MID, Titel, Genre)
Film(MID, Titel, Genre, Dauer)
Serie(MID, Titel, Genre, Staffeln, Folgen)
```

Vertikale Strategie (Table per Subclass, nur eigene Attribute)

```
Medium(MID, Titel, Genre)
Film(MID, Dauer)    (mit FK (MID) auf Medium)
Serie(MID, Staffeln, Folgen)    (mit FK (MID) auf Medium)
```

Nullwerte-Strategie (Table per Hierarchy)

`Medium(MID, Titel, Genre, Dauer, Staffeln, Folgen, Typ)`

(Hier wären `Dauer` für Serien und `Staffeln/Folgen` für Filme `NULL`)

4. Die Relationale Algebra

4.1 Die Relationale Algebra

Eine Algebra wird über einer Menge von Operanden und einer Operatorenmenge definiert.

Die Operationen sind dabei abgeschlossen über der gegebenen Menge, d.h. die Anwendung eines Operators auf einem Element der Menge resultiert wieder in einem Element der Menge.

4.2 Operationen in der Relationalen Algebra

Die relationale Algebra basiert auf der Mengenlehre. Beispiele für Vereinigung ($\cup$), Schnitt ($\cap$) und Mengendifferenz ($\setminus$): Seien $A = \{(1,1),(1,3),(2,5)\}$ und $B = \{(1,1),(1,2),(3,1)\}$.

- $A \cup B = \{(1,1),(1,2),(1,3),(2,5),(3,1)\}$
- $A \cap B = \{(1,1)\}$
- $A \setminus B = \{(1,3),(2,5)\}$

Für die Anwendung in der Relationalen Algebra müssen die Relationen (Tabellen) dasselbe Schema (Spaltennamen und -typen) besitzen.

Nehmen wir an:

Tabelle A:

Film-ID	Titel
1	Wonka
2	Dune 2
4	Oppenheimer

Tabelle B:

Film-ID	Titel
1	Wonka
3	Barbie
5	John Wick 4

Tabelle C:

Serien-ID	Titel
1	Euphoria
2	Sherlock

Da die Schemata von A & B identisch sind, können die Mengenoperationen auf diese Mengen angewendet werden. Das Schema von C unterscheidet sich von den anderen beiden, daher sind Mengenoperationen wie bspw. $A \cup C$ **undefiniert** (und NICHT die leere Menge).

Beispiel $A \cup B$: (Für $A \cup B$ müssen A und B nicht disjunkt sein.)

Tabelle $A \cup B$:

Film-ID	Titel
1	Wonka
2	Dune 2
3	Barbie
4	Oppenheimer
5	John Wick 4

Beispiel $A \setminus B$: (Für $A \setminus B$ muss B keine Teilmenge von A sein.)

Tabelle $A \setminus B$:

Film-ID	Titel
2	Dune 2
4	Oppenheimer

Beispiel $A \cap B$:

Tabelle $A \cap B$:

Film-ID	Titel
1	Wonka

4.2.1 Selektion (σ)

Sei R eine Relation und ϕ (phi) eine beliebige Bedingung. Die Selektion $\sigma_\phi(R)$ bildet R auf eine Teilmenge R' ab, in der alle Tupel die Bedingung ϕ erfüllen.

Beispiel: Sei $R = \{(1,1),(1,2),(2,1),(2,2),(2,3)\}$ und $sch(R) = (A, B)$. Dann ist $\sigma_{A>1}(R) = \{(2,1),(2,2),(2,3)\}$.

$\sigma_{\text{Dauer}<120}$

Titel	Dauer
Wonka	117
Dune 2	166
Barbie	114
Oppenheimer	180
John Wick 4	169

$=$

Titel	Dauer
Wonka	117
Barbie	114

Die Bedingung ϕ ist ein boolescher Ausdruck aus folgenden Operanden und Operatoren:

- Konstanten
- Attribute
- arithmetische Operatoren ($+$, $-$, $*$, ...)
- Vergleiche ($=$, $>$, $<$, ...)
- Boolesche Operatoren ($\vee$ (oder), $\wedge$ (und), $\neg$ (nicht))

ϕ wird für jedes Tupel einzeln ausgewertet. Es sind keine Quantoren ($\exists$, $\forall$) oder verschachtelte Algebraausdrücke erlaubt.

4.2.2 Projektion (π)

Sei R eine Relation und L eine Attributsliste. Die Projektion $\pi_L(R)$ bildet R auf eine Relation R' ab, in der alle Tupel aus R enthalten sind, bei denen die Attribute, die nicht in L sind, jeweils entfernt wurden.

Beispiel: Sei $R = \{(1,1,1),(1,2,3),(2,2,2),(3,3,3)\}$ und $sch(R) = (A,B,C)$. Dann ist $\pi_{A,C}(R) = \{(1,1),(1,3),(2,2),(3,3)\}$.

- π kann auch dazu genutzt werden, die Spalten einer Tabelle neu zu sortieren.
- Intuitiv: σ entfernt Zeilen, π entfernt Spalten.

Durch die Projektion kann sich die Anzahl der Zeilen in der Tabelle ändern (Duplikate werden entfernt).

4.2.3 Das kartesische Produkt ($\times$)

Das kartesische Produkt $R \times S$ kombiniert jedes Tupel aus R mit jedem Tupel aus S.

Beispiel:

Relation R:

A	B
1	1

Relation S:

C	D
4	4
4	5

Relation T:

E
8
9

Relation $R \times S$:

A	B	C	D
1	1	4	4
1	1	4	5

Relation $(R \times S) \times T$:

A	B	C	D	E
1	1	4	4	8
1	1	4	5	8
1	1	4	4	9
1	1	4	5	9

4.2.4 Umbenennung (β)

Seien A und B Attribute und R eine Relation. Die Umbenennung $\beta_{B \leftarrow A}(R)$ (beta) benennt die Spalte A in R in B um.

$\beta_{\text{ID} \leftarrow \text{Film-ID}}$

Film-ID	Titel
1	Wonka
2	Dune 2
3	Barbie
4	Oppenheimer
5	John Wick 4

$=$

ID	Titel
1	Wonka
2	Dune 2
3	Barbie
4	Oppenheimer
5	John Wick 4

4.3 Rechengesetze und Eigenschaften

Das kartesische Produkt ist assoziativ:

$$R \times (S \times T) \equiv (R \times S) \times T$$

Das kartesische Produkt ist nicht kommutativ. (Außer in Kombination mit einer Projektion, die die Spalten neu sortiert):

$$\pi_L(R \times S) \equiv \pi_L(S \times R)$$

Außerdem gilt: Gegeben σ_ϕ enthält nur Attribute von R, dann:

$$\sigma_\phi(R \times S) \equiv \sigma_\phi(R) \times S$$

4.4 Join-Operationen

4.5 Anfragebäume und Anfrageoptimierung (kurz)