DBI Zusammenfassung

Leon Schlömmer

April 4, 2020

Contents

Chapter 1

Modellierung

1.1 Datenmodellierung

Modellieren ist der Weg von der realen Welt bis zum Datenmodell. Ein Datenmodell verdeutlicht die Beziehungen zwischen Daten. Ergebnis des Modellierungsprozesses ist ein sogenanntes Datenschema, das zumeist grafisch visualisiert wird. Im Hinblick auf die Datenmodellierung eignen sich folgende Ansätze:

Konzeptuelles Datenmodell Beschreibt die globale logische Struktur aller Daten Implementierungs-unabhängig und stellt diese in einer systematischen Form dar.

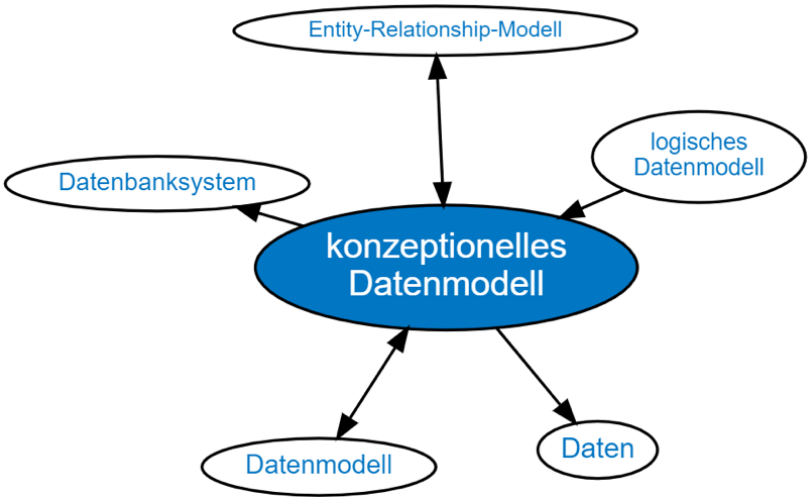


Figure 1.1: Konzeptuelles Datenmodell

Logisches Datenmodell Auf die spätere Implementierung ausgerichtetes Datenmodell, das die Daten für den späteren Einsatz bereits vorstrukturiert.

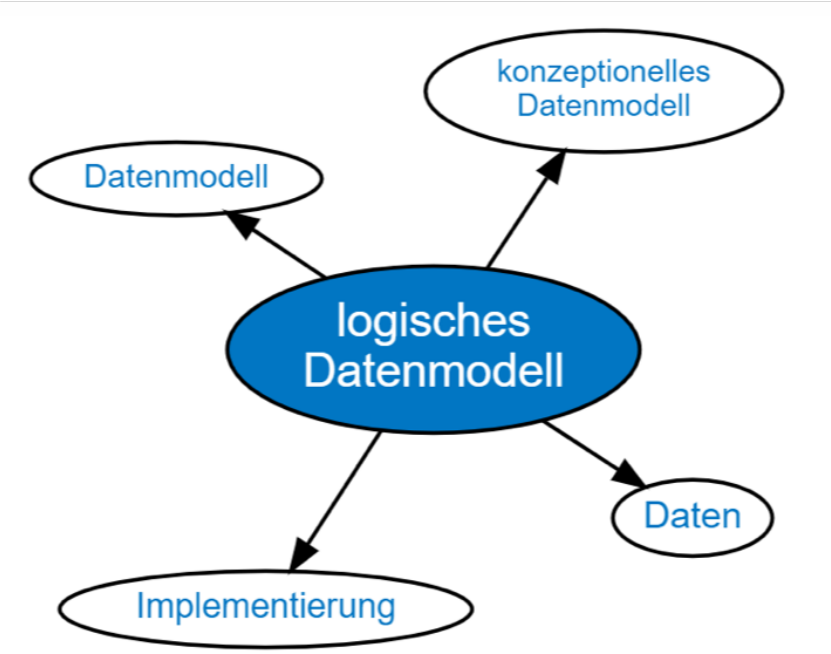


Figure 1.2: Logisches Datenmodell

Physisches Datenmodell Basiert auf der Grundlage des logischen Datenmodells. Im physischen Datenmodell wird besonders auf Effizienz der SQL-Abfragen geachtet, beispielsweise durch den Entwurf von Indexstrukturen.

Aufgabe der Datenmodellierung ist es, bei der Konzeption eines Informationssystems dessen *Objekte mittels Attribute und Beziehungen gemäß den Anforderungen zu strukturieren* und diese Struktur formal zu dokumentieren.

Ziele: Redundanzfreie Datenspeicherung und hohe Datenkonsistenz. Eine redundanzfreie Datenspeicherung liegt dann vor, wenn jede Information in einer Datenbank genau einmal vorkommt. Des Weiteren muss eine hohe Datenkonsistenz verfolgt werden, so dass Daten eindeutige Informationen darstellen.

1.2 Datenmodelle

Ein Modell ist ein *vereinfachtes Abbild der Wirklichkeit.* Zur Beschreibung der Art und Weise, wie Daten in einer Datenbank gespeichert werden, gibt es verschiedene Datenmodelle. Einige dieser Modelle findet man auch in der betriebswirtschaftlichen Organisationslehre wieder.

1.2.1 Hierarchisches Datenmodell

Das hierarchische Datenmodell kennen wir vom Dateisystem unserer Festplatte (Laufwerksbuchstabe, Ordner, Dateien). Dieses Modell wird auch „Baumstruktur“ genannt.

Ganz links (oder oben) befindet sich die Wurzel (Root). Von ihr sind alle Objekte abhängig, die weitere abhängige Objekte haben können. In der betriebswirtschaftlichen Lehre ist es mit einer Stablinienorganisation vergleichbar. Das hierarchische Datenmodell war früher ein sehr gebräuchliches Modell, deshalb bildet es die Grundlage älterer Datenbanksysteme.

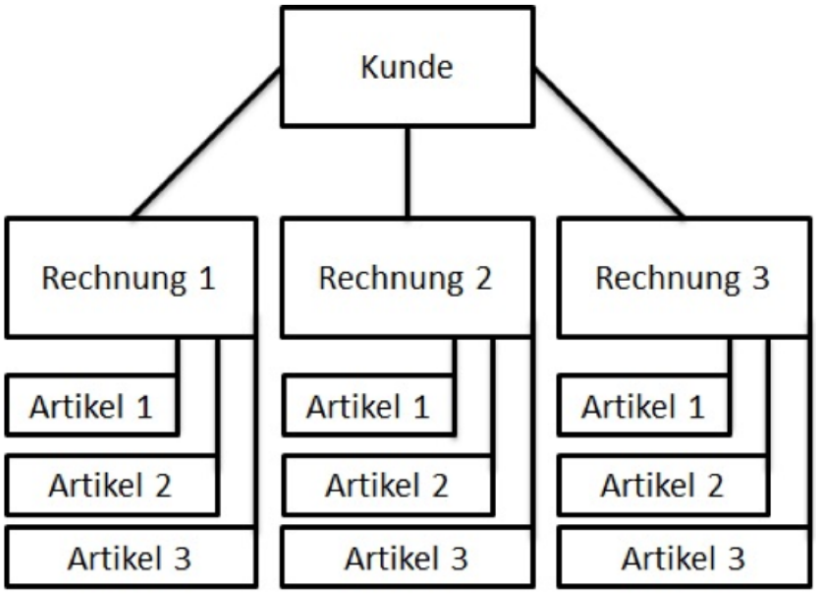


Figure 1.3: Stablinienorganisation

Durch die *hierarchische Baumstruktur ist der lesende Zugriff extrem schnell.* Der Nachteil der baumstrukturierten Verweise liegt bei der Speicherung der Daten und deren Verknüpfungen, da die Verweise untereinander vorab ermittelt werden müssen. Die Verknüpfungen werden über Eltern-Kind-Beziehungen realisiert und in der Baumstruktur abgebildet. Der große Nachteil bei diesem Modell ist es, dass man nur eine Baumstruktur verwenden kann: Es ist also nicht möglich, zwei Baumstrukturen miteinander zu verknüpfen. *Das hat zur Folge, dass dieses Modell sehr starr ist und wenig Freiheit für den Entwickler bietet.*

1.2.2 Netzwerk Datenmodell

Definition: Datenmodell, mit dem Netzwerkstrukturen zwischen Datensätzen beschrieben werden können. Es dient als Grundlage vieler Datenbanksysteme. Die drei Datenbanksprachen DML, DDL und DCL werden angewandt. Durch das netzwerkartige Modell existieren meist unterschiedliche Suchwege, um einen bestimmten Datensatz zu ermitteln. Es ähnelt dem hierarchischen Datenbankmodell und kann einer Matrixorganisation gegenübergestellt werden. Das Netzwerk Datenbankmodell besitzt keine strenge Hierarchie. Ein Datenfeld besteht aus einem Namen und einem Wert. Es sind m:n Beziehungen möglich, d.h. ein Datensatz kann mehrere Vorgänger haben. Des Weiteren

können auch mehrere Datensätze an erster Stelle stehen. Der Vorteil dieses Modells ist, das es unterschiedliche Suchwege als Lösungsweg angibt, was natürlich auch zu Problemen führen kann, wenn der Entwickler genau einen Lösungsweg benutzen will. Auch die Übersichtlichkeit verringert sich, wenn das Modell ständig weiter wächst.

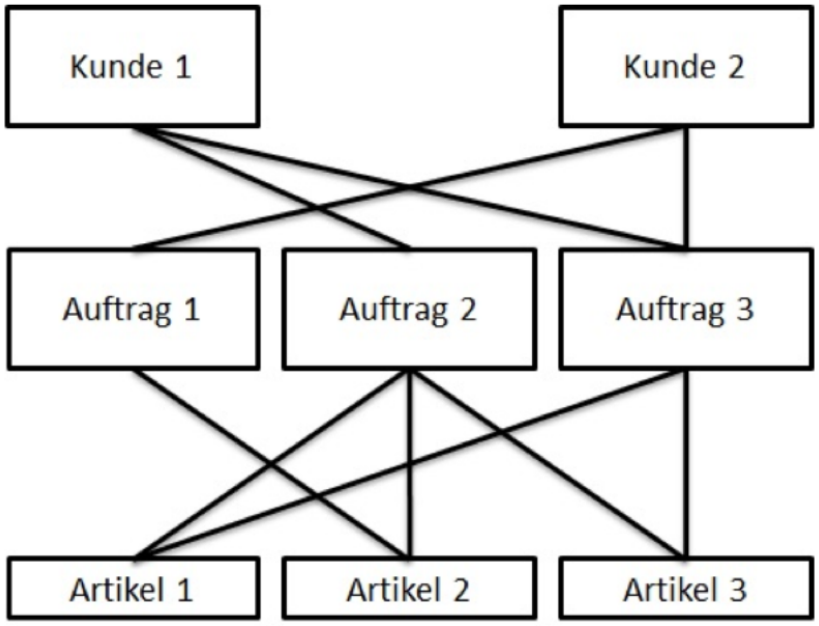


Figure 1.4: Netzwerk Datenmodell

Heute wird das Netzwerkdatenbankmodell als eine Art Verallgemeinerung des hierarchischen Datenbankmodells gesehen.

1.2.3 Relationales Datenmodell

= Auf den Arbeiten von Edgar F. Codd von 1970 basierendes Datenmodell, mit dem Beziehungen zwischen Daten in Form von Relationen beschrieben werden.

In einfachen Worten: Ein relationales Datenmodell ist eine Ansammlung von Tabellen, die miteinander verknüpft sind. Das relationale Datenmodell ist das am weitesten verbreitete Datenmodell, welches in der Datenbankentwicklung als Standard genutzt wird. Das Fundament des Datenbankmodells besteht aus drei Elementen: Tabellen, Attributen und Beziehungen.

Vorteile: sehr einfach und flexibel zu erstellen, hohe Flexibilität, leichte Handhabung

Nachteil: Effizienzprobleme bei großem Datenvolumen

Die wichtigsten Operationen mit Relationen (relationale Algebra), die ein Datenbankmanagementsystem zur Verfügung stellen muss, sind folgende:

- Auswahl von Zeilen
- Auswahl von Spalten
- Aneinanderfügen von Tabellen
- Verbund von Tabellen

Entität

Als Entität wird in der Datenmodellierung ein eindeutig zu bestimmendes Objekt bezeichnet, über das Informationen gespeichert oder verarbeitet werden sollen. Ein Entitätstyp beschreibt die Ausprägungen eines Objektes durch die Angabe von Attributen. Durch Typisierung (Erkennen gleicher Attribute von Entitäten) können Entitätstypen abgeleitet werden – aus mehreren Personen werden z.B. Kunden. Eine Entität kann materiell oder immateriell, konkret oder abstrakt sein. Beispiele für Entitäten: Fahrzeug, Konto, Person.

Attribute in einer Entität

Jede Entität besitzt eine bestimmbare Anzahl an Attributen (Ausprägungen bzw. Eigenschaften), die sich eindeutig von anderen Entitäten des gleichen Entitätstyps abgrenzen. Eine Eigenschaft ist ein konkreter Attributwert, den ein zuvor definiertes Attribut annehmen kann. Die Attribute stellen einen „Bauplan“ dar, der eine abstrakte Abbildung der Wirklichkeit ist.

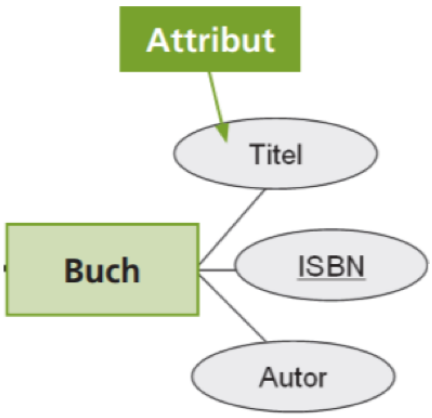


Figure 1.5: Attribute einer Entity

Die Attribute in einer Entität können unterschiedlich aufgebaut sein. Man unterscheidet zwischen *zusammengesetzte, mehrwertige und abgeleitete* Attribute.

Zusammengesetzte Attribute bestehen aus der Kombination mehrerer Attribute, die inhaltlich zusammengehören. Anhand einer Firmenadresse wird dies deutlich. Die Firmenadresse selbst ist ein Attribut der Firma, sie enthält aber die Attribute Straße, Hausnummer, Postleitzahl und Ort.

Mehrwertige Attribute können einen oder mehrere Attributwerte aufnehmen. So könnte ein Student gleichzeitig in zwei Studiengänge eingeschrieben sein.

Abgeleitete Attribute werden aus anderen Attributen oder Entitäten berechnet. Bezogen auf eine Datenbank wäre das z.B. die Bildung einer Summe aus mehreren Spalten einer Tabelle.

1.2.4 Beziehungen in Datenbanken

Zwischen Relationen (Tabellen/Entitäten) können Beziehungen in einer Datenbank bestehen. Angenommen man hat eine Relation „Mutter“ und eine Relation „Kind“ – denkbar wären nun vier Möglichkeiten von Assoziationen/Beziehungen zwischen den Tabellen.

In einem Datenbankmodell können folgende Beziehungen auftreten:

- Jede Mutter hat exakt ein Kind (1)
- Jede Mutter hat ein oder kein Kind (c)
- Jede Mutter hat mindestens ein Kind (m)
- Jede Mutter hat eine kein, ein, oder eine beliebige Anzahl von Kindern (mc)

1.2.5 Kardinalität

Die Kardinalität zwischen dem Entitätstyp 1 und dem Entitätstyp 2 gibt an, wie viele Entitäten des Entitätstyps 2 höchstens mit einer Entität des Entitätstyps 1 in Beziehung stehen. Die Kardinalität von Beziehungen ist in relationalen Datenbanken in folgenden Formen vorhanden: 1:1 Beziehung, 1:n Beziehung und m:n Beziehung. Des Weiteren gibt es noch die sogenannte *Modifizierte Chen-Notation* (c).

1:1 Beziehung

In einer „eins zu eins“ Beziehung in relationalen Datenbanken ist jeder Datensatz in Tabelle A genau einem Datensatz in Tabelle B zugeordnet und umgekehrt. Diese Art von Beziehung sollte in der Modellierung vermieden werden, weil die meisten Informationen, die auf diese Weise in Beziehung stehen, sich in einer Tabelle befinden können. Eine 1:1 Beziehung verwendet man nur, um eine Tabelle aufgrund ihrer Komplexität zu teilen oder um einen Teil der Tabelle aus Gründen der Zugriffsrechte zu isolieren.

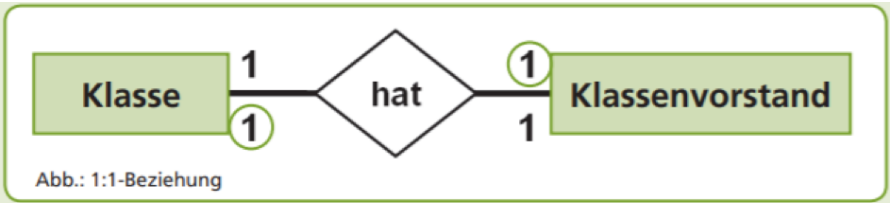


Figure 1.6: Eine Klasse hat einen Klassenvorstand. Ein Klassenvorstand hat eine Klasse

1:n Beziehung

Eine „eins zu viele“ Beziehung in relationalen Datenbanken ist der häufigste Beziehungstyp in einer Datenbank. In einer 1:n Beziehung können einem Datensatz in Tabelle A mehrere passende Datensätze in Tabelle B zugeordnet sein, aber einem Datensatz in Tabelle B ist nie mehr als ein Datensatz in Tabelle A zugeordnet. Eine 1:n Beziehung wird mit einem Fremdschlüssel aufgelöst. Als Fremdschlüssel in der abhängigen Relation (n-Seite) wird der Primärschlüssel der unabhängigen Relation (1-Seite) eingefügt. Er verweist daher auf eine Zeile in dieser Relation.

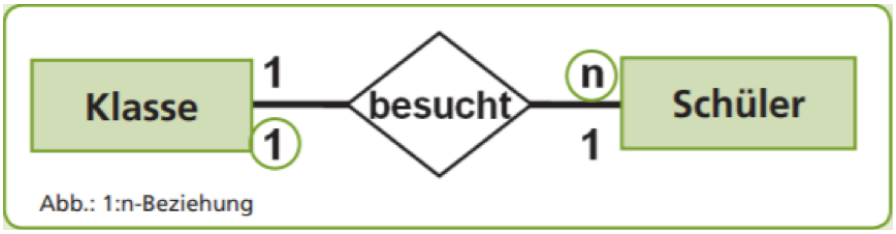


Figure 1.7: Eine Klasse hat mehrere Schüler. Ein Schüler besucht 1 Klasse

m:n Beziehung

Bei „viele zu viele“ Beziehung in relationalen Datenbanken können jedem Datensatz in Tabelle A mehrere passende Datensätze in Tabelle B zugeordnet sein und umgekehrt. Diese Beziehungen können nur über eine dritte Tabelle, eine Verbindungstabelle C, realisiert werden. Die Verbindungstabelle C enthält die beiden Primärschlüssel der Tabellen A und B als Fremdschlüssel. Der Primärschlüssel der Verbindungstabelle wird aus diesen beiden Fremdschlüsseln gebildet. Daraus folgt, dass eine m:n Beziehung in Wirklichkeit zwei 1:n Beziehungen sind.

Conditional c

Die obigen 3 Beziehungen sind die Standardbeziehungen. Diese wird Chen Notation genannt. Bei der modifizierten Chen Notation gibt es zusätzlich noch das conditional c. Dieser Buchstabe repräsentiert „eine oder keine“ (1:c) Beziehungen und wird beispielsweise bei der Generalisierung angewandt (siehe Kapitel Generalisierung/Spezialisierung).

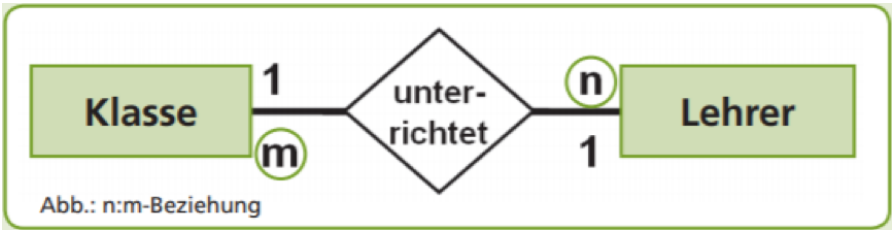


Figure 1.8: Eine Klasse wird von mehreren Lehrern unterrichtet. Ein Lehrer unterrichtet mehrere Klassen.

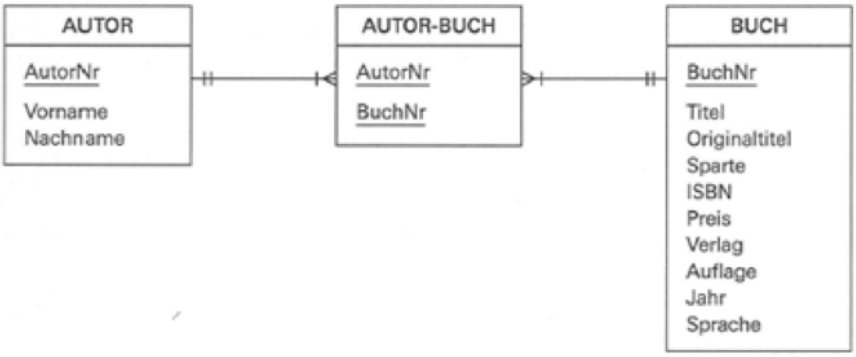


Figure 1.9: Verbindungstabelle zwischen Autor und Buch

Daraus ergeben sich nun folgende zusätzliche Beziehungen:

- 1:c Beziehung: Schüler (1) besitzt Buchausweis (c)
- 1:mc Beziehung: Schüler (1) startet Druckaufträge (mc)
- c:c Beziehung: Schüler (c) mietet Spind (c)
- c:m Beziehung: Religionslehrer (c) unterrichtet Schüler (m)
- c:mc Beziehung: Schüler (c) recherchiert im Lexikon (mc)
- m:mc Beziehung: Schüler (m) hat Robotik Unterricht (mc)
- mc:mc Beziehung: Schüler (mc) benutzt Onlinelexika (mc)

Jede Zeile (auch Tupel genannt) in einer Tabelle ist ein Datensatz. Jedes Tupel besteht aus einer großen Reihe von Eigenschaften (Attributen), den Spalten der Tabelle. Ein Relationsschema legt dabei die Anzahl und den Typ der Attribute für eine Tabelle fest.

Das relationale Datenmodell erhält man, indem man das hierarchische Datenmodell mit dem Netzwerk-Datenmodell kombiniert. Ein Datenobjekt ist von einem oder

mehreren Datenobjekten abhängig. Daraus ergeben sich mehrere Arten von Abhängigkeiten, die wir als Beziehungen bezeichnen.

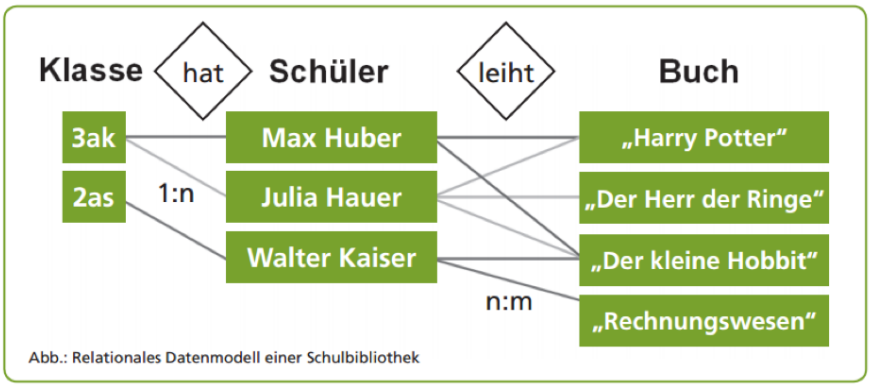


Figure 1.10

Des Weiteren können Verknüpfungen (Beziehungen) über sogenannte Primärschlüssel hergestellt werden, um bestimme Attribute, die den gleichen Primärschlüssel oder in einer Detailtabelle als Fremdschlüssel besitzen, abzufragen.

Primary Keys

Der Primärschlüssel kommt in relationalen Datenbanken zum Einsatz und wird zur eindeutigen Identifizierung eines Datensatzes verwendet. Neben der Eindeutigkeit soll ein Primärschlüssel kurz sein und leicht geschrieben werden können. Des Weiteren sollen aus dem Schlüssel gewisse Eigenschaften der Entität ersichtlich sein. Diese Eigenschaft kann im Widerspruch zu der Eindeutigkeit und der Kürze stehen. Beispiele:

- Sozialversicherungsnummer (nnnn dd mm yy) besteht aus einer 4-stelligen Nummer und den Geburtsdaten - Eindeutigkeit: ja, Kürze: teilweise, Sprechender Schlüssel: kaum
- Matrikelnummer eines Studenten (yysssnnn) besteht aus dem Immatrikulationsjahr, einer Studienkennzahl und einer laufenden Nummer - Eindeutigkeit: ja, Kürze: ja, Sprechender Schlüssel: teilweise

In einer normalisierten Datenbank besitzen alle Tabellen einen Primärschlüssel. Der Wert eines Primärschlüssels muss in einer Tabelle einmalig sein, da er jeden Datensatz eindeutig kennzeichnet. Des Weiteren wird er häufig als Datenbank-Index verwendet, um die Daten auf der Festplatte abzulegen. Der Primärschlüssel einer Relation kann unterschiedlich aufgebaut sein. Man unterscheidet zwischen eindeutigen, zusammengesetzten und künstlichen Primärschlüsseln.

Eindeutiger Primary Key Hierbei handelt es sich um einen eindeutigen Schlüssel, der in einer Spalte der Tabelle gespeichert wird. Als Spalte kann ein Attribut des Datensatzes verwendet werden, das für jeden Eintrag in der Tabelle einen einmaligen Wert annimmt. Als eindeutiges Primärschlüssel-Attribut könnte beispielsweise die Sozialversicherungsnummer in einer Mitarbeitertabelle verwendet werden.

Zusammengesetzter Primary Key Ist ein Datensatz anhand eines Attributes nicht eindeutig identifizierbar, so kann der Primärschlüssel auch aus einer Kombination mehrerer Attribute bestehen. Dabei muss sichergestellt werden, dass jede dieser Kombinationen nur einmalig auftritt. Ein zusammengesetzter Primärschlüssel kann z.B. der Vor- und Nachname in Kombination mit dem Geburtsdatum sein.

Künstlicher Primary Key Gibt es in einer Tabelle keine eindeutigen Spalten bzw. Kombinationen aus Spalten, so kann auch auf einen künstlichen Schlüssel zurückgegriffen werden. Dieser ist auch als Surrogate Key bekannt und wird als zusätzliche Spalte in einer Tabelle eingefügt. In der Praxis wird häufig eine fortlaufende Ganzzahlen-Folge verwendet, um einen Datensatz eindeutig identifizieren zu können.

Foreign Key

Der Fremdschlüssel kann Bestandteil einer Tabelle in einer relationalen Datenbank sein. Dabei handelt es sich um eine Schlüsselspalte, die auf einen Primärschlüssel einer anderen oder aber derselben Tabelle verweist. In einfachen Worten: Als Fremdschlüssel wird eine Menge von Attributen bezeichnet, die in einer anderen Tabelle den Primärschlüssel bildet. 1:n Beziehungen werden beispielsweise mit einem Fremdschlüssel aufgelöst.

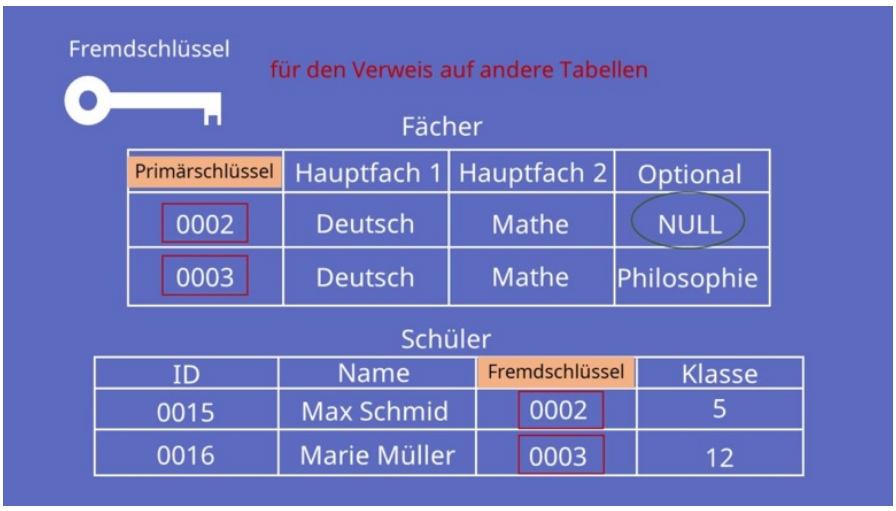


Figure 1.11: Foreign Key

1.2.6 Objekt-Datenmodell

Objekte sind modellhafte Abbilder der Wirklichkeit. Das Objekt-Datenmodell wird vor allem im Bereich der Softwareentwicklung eingesetzt. Ein wichtiges Prinzip beim Objektmodell ist die Vererbung, durch die ein effizienteres Programmieren möglich wird.

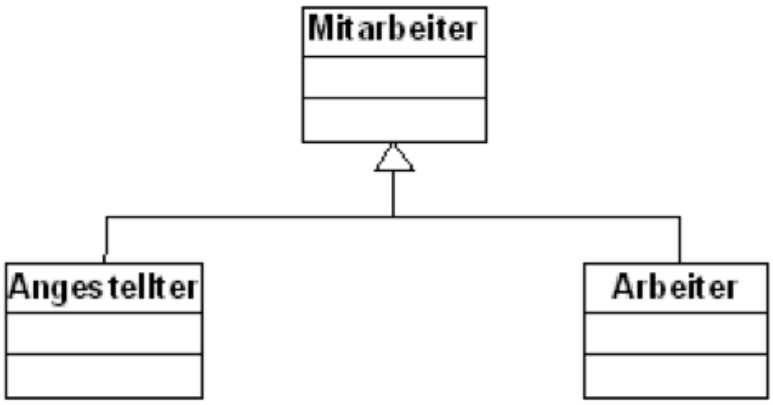


Figure 1.12: Vererbung bei Objekten

1.3 Entity-Relationship-Modell

Die Modellierung relationaler Datenbanken erfolgt mit dem von Peter Chen entwickelten Entity-Relationship-Modell. Bevor mittels SQL angefangen wird, Tabellen und Beziehungen anzulegen, wird zuerst geplant, wie die Datenbankstruktur funktionieren und aufgebaut werden soll. Der Einsatz von ER-Modellen ist in der Praxis ein gängiger Standard für die Datenmodellierung, auch wenn es unterschiedliche grafische Darstellungsformen von Datenbankmodellen gibt. Mithilfe des Entity-Relationship-Modells soll eine Typisierung von Objekten, ihrer relationalen Beziehungen untereinander und der zu überführenden Attribute, stattfinden.

Bestandteile des ERD

- **Entität:** Ein individuell identifizierbares Objekt der Wirklichkeit.
- **Beziehung**: Eine Verknüpfung / Zusammenhang zwischen zwei oder mehreren Entitäten.
- **Attribut:** Eine Eigenschaft, die im Kontext zu einer Entität steht.

Beispiel
Erklärung: Ein Mitarbeiter hat einen Namen. Ein Projekt hat einen Namen, ein Datum und ein Budget. Ein Mitarbeiter kann mehrere Projekte leiten, aber ein Projekt kann nur von einem Mitarbeiter geleitet werden. Zur Wiederholung: Diese Notation

Figure 1.13: UML Darstellung

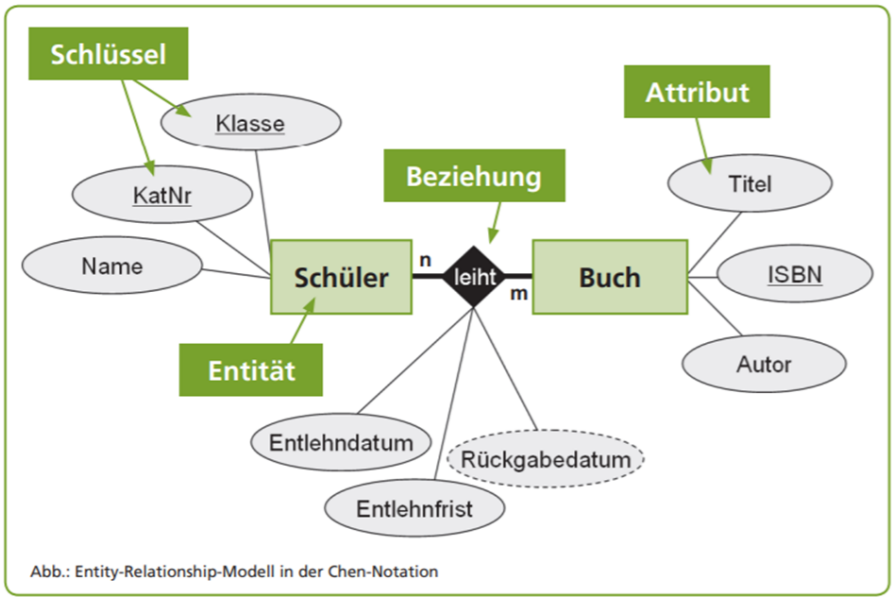


Figure 1.14: Was ist was

nennt man Chen-Notation und ist ein gängiger Standard in der Praxis der Datenmodellierung.

1.3.1 Überleitung eines ERD's in ein relationales Modell

Ausgangspunkt ist ein vollständiges ER-Diagramm. Ziel ist die Darstellung des Sachverhaltes in Form von Relationen (Tabellen). Eine Tabelle ist eine Menge von Tupeln; ein Tupel ist eine Liste von Werten und entspricht einer Tabellenzeile.

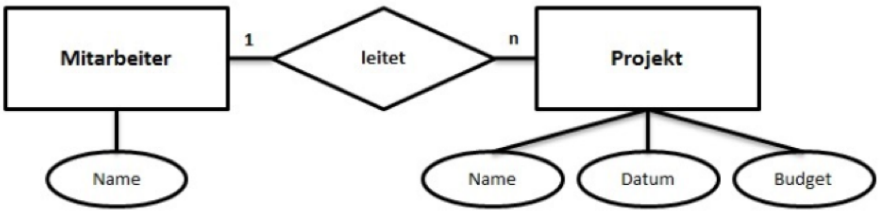


Figure 1.15: Beispiel Darstellung

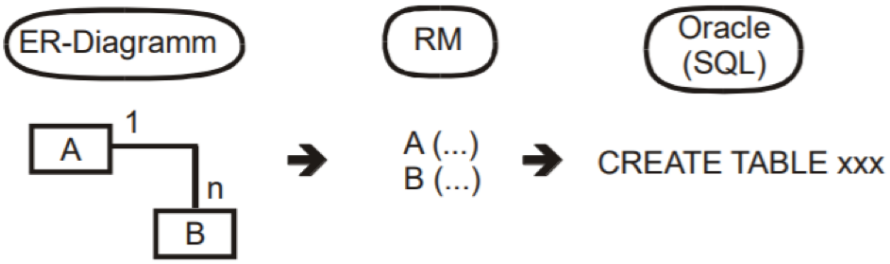


Figure 1.16: ERD zu Relational

Merkmale einer Tabelle	
Name	eine Tabelle hat einen eindeutigen Namen z.B. ARTIKEL
Zeilen	sie hat mehrere Tupel (Tabellenzeilen). Die Ordnung der Tupel ist bedeutungslos, weil eine Zeile nicht aufgrund ihrer Position, sondern aufgrund von Werten angesprochen wird. z.B.: (300, Müller, 4060, Leonding)
Spalten	sie hat mehrere Spalten (Eigenschaften). Die Ordnung der Spalten ist bedeutungslos, weil eine Eigenschaft nicht aufgrund der Position, sondern aufgrund ihres Namens angesprochen wird; dieser Name ist eindeutig z.B. Ort
Werte	Eine bestimmte Spalte enthält Attributwerte als Daten z.B. Leonding

1. Jede Entitätsmenge wird als Tabelle mit einem Primärschlüssel dargestellt	Strukturregel 1 (SR1): Bei der Darstellung von Entitätsmengen durch Relationen muss für jede Relation ein Primärschlüssel existieren
2. 1:n Beziehungen werden Fremdschlüssel	Als Fremdschlüssel in der abhängigen Relation (n-Seite) wird der Primärschlüssel der unabhängigen Relation (1-Seite) eingefügt. Er verweist daher auf eine Zeile in dieser Relation. (ähnlich einem Pointer oder einem Link)
3. m:n Beziehungen werden assoziative Tabellen	Die neu gebildete assoziative Relation enthält die Primärschlüssel der beiden in Beziehung stehenden Relationen. Sie erhält ihren Primärschlüssel entweder direkt als Kombination der beiden Fremdschlüssel oder es wird („künstlich“) ein neuer Primärschlüssel zugeteilt; die beiden Fremdschlüssel werden dann als Nichtschlüsselattribute verwendet.
4. Eigenschaften werden Spalten	Alle Eigenschaften werden als Attribute abgebildet. Die Eigenschaftswerte finden sich dann in der Spalte „darunter“.
5. Jede Beziehung wird zu einem Fremdschlüssel	Auch 1:1 – Beziehungen und insbesondere rekursive Beziehungen werden als Fremdschlüssel abgebildet.
6. Normalisierung	Prof. Kainerstorfer um Skript bitten (1.4)
7. Aggregation	Prof. Kainerstorfer um Skript bitten (1.4)

1.3.2 Kochrezept - Auflösung m:n Beziehungen

Im ERD können m:n - Beziehungen vorkommen, diese müssen jedoch spätestens bei der Überleitung in das Relationale Modell aufgelöst werden. Grund: Die Abbildung durch eine Menge von Fremdschlüsseln widerspricht der Forderung nach skalaren Attributen. Diese Auflösung erfolgt durch die Einführung einer neuen, sogenannten assoziativen Tabelle, die die Fremdschlüssel der beiden in Beziehung stehenden Entitäten enthält. (siehe 1.17)

Würde man nach der Methode der Überleitung für 1:n - Beziehungen vorgehen, so würde man eine der beiden Lösungen erhalten: (siehe 1.18)

Das neue „Attribut“ Mitarbeiter.Projekte müsste alle jene ProjektId's aufnehmen, an denen der Mitarbeiter arbeitet – und umgekehrt für Projekt.Mitarbeiter. Weil das aber jeweils ganze Mengen sind, und solche Konstrukte in Relationalen Modell nicht definiert sind, führt man eine sogenannte assoziative Relation ein – also eine Relation, die eine Assoziation darstellt. (siehe 1.19)

Die Relation Arbeit enthält dann die Fremdschlüssel von Mitarbeiter und Projekt – diese können auch gleichzeitig die Primärschlüssel der neu gebildeten Relation sein (muss aber nicht): (siehe 1.20)

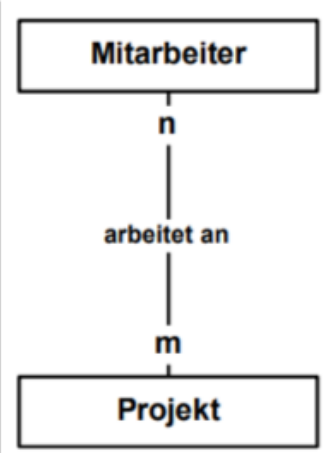


Figure 1.17: Beispiel

```
Mitarbeiter     (..., Projekte, ...)
Projekt         (..., Mitarbeiter, ...)
```

Figure 1.18: Beispiel

1.4 Aggregation

Eine Aggregatfunktion ist eine Funktion, die gewisse Eigenschaften von Daten zusammenfasst. Man unterscheidet:

- Distributive Funktionen
  - Summe (SUM)
  - Anzahl (COUNT)
  - Maximum (MAX)
  - Minimum (MIN)
- Algebraische Funktionen
  - Mittelwert (AVG)
- weitere Funktionen
  - Standardabweichung (StDev)
  - Standardabweichung für die Grundgesamtheit (StDevP)
  - Median

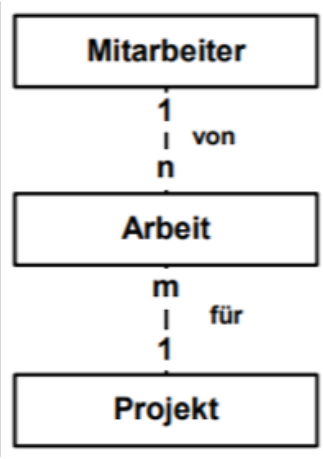


Figure 1.19: Beispiel

```
Arbeit          (..., ProjektId, PersNr, ...)
```

Figure 1.20: Beispiel

1.4.1 „is-part-of“-Beziehung (ERM)

Eine andere Verwendung des Begriffs Aggregation ist auch beim Entity-Relationship-Modell zu finden. So können hier mehrere Einzelobjekte logisch zu einem Gesamtobjekt zusammengefasst werden. Dies geschieht mit der „is-part-of“-Beziehung.

Werden mehrere Einzelobjekte (z.B. Person und Hotel) zu einem eigenständigen Einzelobjekt (z.B. Reservierung) zusammengefasst, dann spricht man von Aggregation. Dabei wird das übergeordnet eigenständige Ganze Aggregat genannt. Die Teile, aus denen es sich zusammensetzt, heißen Komponenten. Aggregat und Komponenten werden als Entitätstyp deklariert.

Bei einer Aggregation wird zwischen Rollen- und Mengenaggregation unterschieden: Eine Rollenaggregation liegt vor, wenn es mehrere rollenspezifische Komponenten gibt und diese zu einem Aggregat zusammengefasst werden.

Beispiel: Fußballmannschaft is-part-of Fußballspiel und Spielort is-part-of Fußballspiel und in anderer Leserichtung: Fußballspiel besteht-aus Fußballmannschaft und Spielort. Eine Mengenaggregation liegt vor, wenn das Aggregat durch Zusammenfassung von Einzelobjekten aus genau einer Komponente entsteht.

Beispiel: *Fußballspieler is-part-of Fußballmannschaft*
und in anderer Leserichtung:
Fußballmannschaft besteht-aus (mehreren, N) Fußballspielern.

Generalisierung / Spezialisierung

Zur weiteren Strukturierung der Entitätstypen wird die Generalisierung eingesetzt. Hierbei werden Eigenschaften von ähnlichen Entitätstypen einem gemeinsamen Obertyp, auch „Superentität“ genannt, zugeordnet.

Bei diesem Modellierungs-Konstrukt werden gemeinsame Eigenschaften von Entitäten nur einmal modelliert.

- Der Generalisierungstyp enthält die Attribute, die alle Entitäten gemeinsam haben.
- Die Spezialisierungstypen enthalten die speziell für sie zutreffenden Attribute, wobei
  - die Spezialisierung disjunkt (=getrennt) oder überlappend sein kann
  - die Eigenschaften der Generalisierung erbt

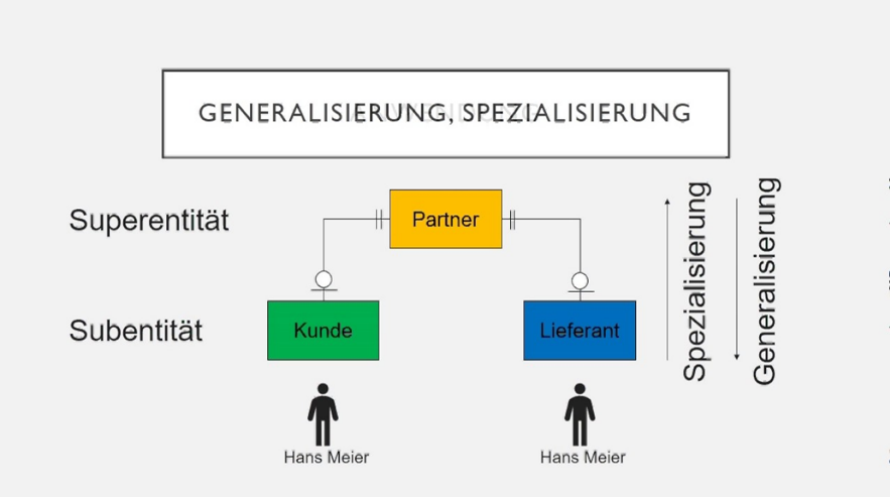


Figure 1.21: Generalisierung und Spezialisierung

Diese Beziehungstypen werden durch „is-a“ — „can-be“ beschrieben.
Beispiel: *Flugreise is-a Reise*
und in anderer Leserichtung:
Reise can-be Flugreise

Während Spezialisierungen durch Bildung von Teil-Entitätsmengen aus gegebenen Entitäten entstehen, werden bei der Generalisierung gemeinsame Eigenschaften und Beziehungen, die in verschiedenen Entitätstypen vorkommen, zu einem neuen Entitätstyp zusammengefasst. So können z.B. Kunden und Lieferanten zusätzlich zu Geschäftspartnern zusammengeführt werden, da Name, Anschrift, Bankverbindung etc. sowohl bei den Kunden als auch bei den Lieferanten vorkommen.

1.5 Unterschied Aggregation und Generalisierung/Spezialisierung

Generalisierung: bottom-up.
Spezialisierung: top-down.

Bei einer Aggregation werden die Beziehungen zwischen 2 Entitäten (von außen) als eine einzige Entität gesehen, es können jedoch beide Entitäten eigenständig ohne der anderen existieren. Beispiel:

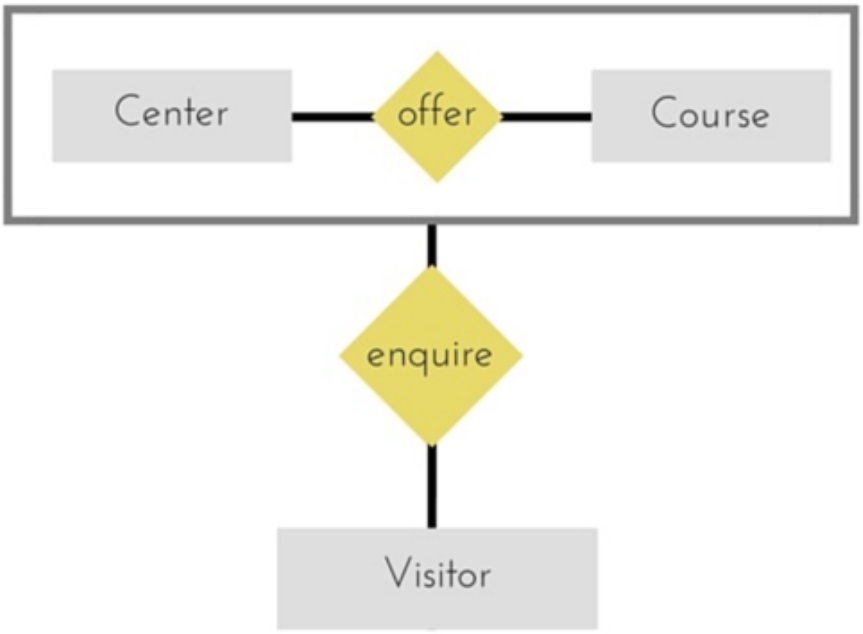


Figure 1.22: Generalisierung und Spezialisierung

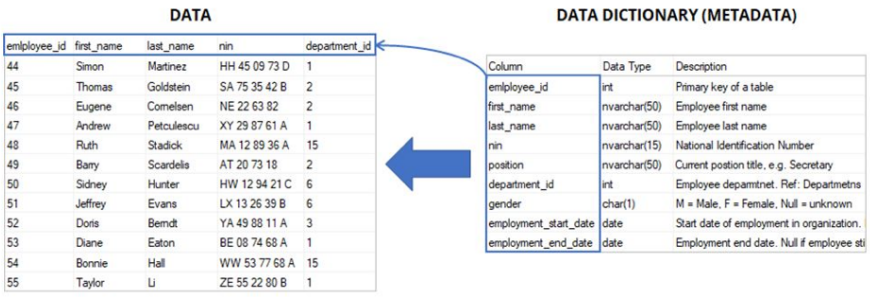
Bei einer Aggregation gibt es KEINE VERERBUNG, bei der Generalisierung/Spezialisierung schon.

1.6 Data Dictionary

Ein Data-Dictionary – deutsch Datenwörterbuch, Datenkatalog – ist ein Katalog von Metadaten, der die Definitionen und Darstellungsregeln für alle Anwendungsdaten eines Unternehmens und die Beziehungen zwischen den verschiedenen Datenobjekten enthält, damit der Datenbestand redundanzfrei und einheitlich strukturiert wird.

Bei einer relationalen Datenbank ist ein Datenwörterbuch eine Menge von Tabellen und Views, die bei Abfragen nur gelesen werden (read-only). Datenbankobjekte, die dort nicht eingetragen sind, gibt es auch nicht. Die Informationen in einem Data Dictionary beinhalten die Namen der Tabellen und deren Spalten inklusive Datentypen und Längen. Es werden alle Datenbankobjekte gelistet - alle Tabellen, Sichten, Indices, Sequenzen, Constraints, Synonyme und mehr. Das Data-Dictionary ist wie eine Datenbank aufgebaut, enthält aber nicht Anwendungsdaten, sondern Metadaten, das heißt Daten, welche die Struktur der Anwendungsdaten beschreiben (und nicht den Inhalt selbst). Aufbau und Pflege eines solchen Datenkatalogs erfolgen üblicherweise über einen interaktiven

Dialog oder mit Hilfe einer Datendefinitionssprache (DDL).



emlployee_id	first_name	last_name	nin	department_id
44	Simon	Martinez	HH 45 09 73 D	1
45	Thomas	Goldstein	SA 75 35 42 B	2
46	Eugene	Cornelsen	NE 22 63 82	2
47	Andrew	Petculescu	XY 29 87 61 A	1
48	Ruth	Stadick	MA 12 89 36 A	15
49	Barry	Scardelis	AT 20 73 18	2
50	Sidney	Hunter	HW 12 94 21 C	6
51	Jeffrey	Evans	LX 13 26 39 B	6
52	Doris	Berndt	YA 49 88 11 A	3
53	Diane	Eaton	BE 08 74 68 A	1
54	Bonnie	Hall	WW 53 77 68 A	15
55	Taylor	Li	ZE 55 22 80 B	1

Column	Data Type	Description
emlployee_id	int	Primary key of a table
first_name	nvarchar(50)	Employee first name
last_name	nvarchar(50)	Employee last name
nin	nvarchar(15)	National Identification Number
position	nvarchar(50)	Current postion title, e.g. Secretary
department_id	int	Employee deparmtnet. Ref: Departmetns
gender	char(1)	M = Male, F = Female, Null = unknown
employment_start_date	date	Start date of employment in organization.
employment_end_date	date	Employment end date. Null if employee sti

Figure 1.23

1.6.1 Oracle Data-Dictionary

Der Datenbankbenutzer SYS besitzt alle Basistabellen und vom Benutzer zugreifbaren Views des Data Dictionary. Kein Oracle-Datenbankbenutzer sollte Zeilen oder Schemaobjekte im SYS-Schema ändern (UPDATE, DELETE oder INSERT), da diese Aktivitäten die Datenintegrität beeinträchtigen können.

Die Views des Data Dictionary dienen als Referenz für alle Datenbankbenutzer. Einige Ansichten sind für alle Oracle-Datenbankbenutzer zugänglich, andere nur für Datenbankadministratoren. Das Data Dictionary ist immer verfügbar und befindet sich im SYSTEM Tablespace.

Das Data Dictionary besteht aus Views. In vielen Fällen besteht eine Gruppe aus drei Views, die ähnliche Informationen enthalten und sich durch ihre Präfixe voneinander unterscheiden:

- USER: Die Datenbankobjekte, die der Benutzer in seinem Schema angelegt hat
- ALL: Die Datenbankobjekte, auf die der Benutzer Zugriffsrechte erhalten hat
- DBA: Alle angelegten Datenbankobjekte

Des Weiteren gibt es noch einen vierten Präfix (V$), welcher für statistische Informationen, die für Laufzeitverbesserungen (Tuning) oder für optimierte Speicherformen ausgewertet werden können, verwendet wird.

Beispiele: **Alle Objekte im eigenen Schema:**

```
SELECT object_name, object_type FROM USER_OBJECTS;
```

Alle Objekte auf die der Benutzer Zugriff hat:

```
SELECT owner, object_name, object_type FROM ALL_OBJECTS;
```

Globale Ansicht auf die Datenbank (wird vom Administrator ausgeführt):

```
SELECT owner, object_name, object_type FROM SYS.DBA_OBJECTS;
```

1.7 Sternschema (Star-Schema)

Das Star-Schema ist, wie der Name schon beschreibt, sternförmig aufgebaut und für analytische Anwendungen im Data Warehouse-Umfeld geeignet. Die Fakten (engl. Measures) in einem Star-Schema sind in diesem Modell das zentrale Element der Datenanalyse. Sie haben die Aufgabe wichtige Zusammenhänge in quantitativ messbarer und verdichteter Form wiederzugeben. Die Dimensionen (engl. Dimensions) in einem Star-Schema ermöglichen unterschiedliche Sichten auf die Fakten. Sie liefern einen fachlichen Bezug auf die quantitativen Werte in einem Sternschema.

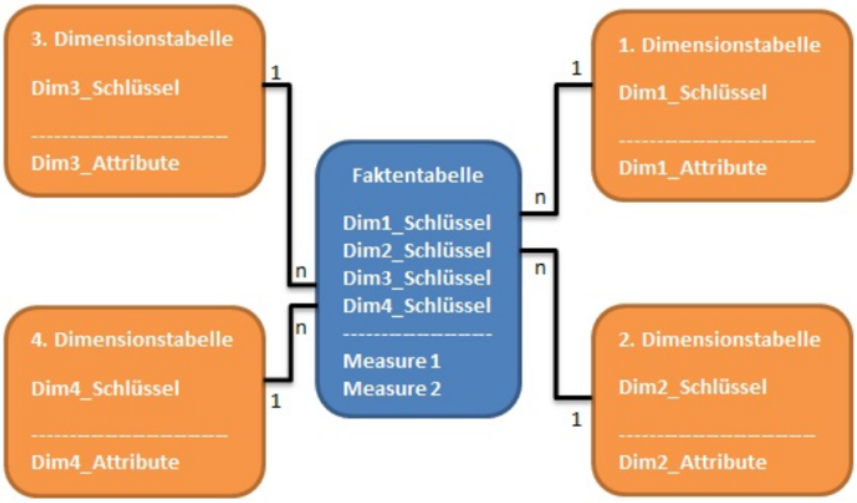


Figure 1.24: Aufbau

Die Fakten können in einem Star-Schema gruppiert und analysiert werden. Die Betrachtung von Verdichtungsstufen ermöglicht die sogenannten Hierarchisierungen. Da im Star-Schema keine direkte Hierarchisierung von Dimensionen möglich ist, wird über die „kontrollierte Redundanz“ in den Dimensionen eine Hierarchisierung ermöglicht. Das Star-Schema setzt sich aus Fakten- und Dimensionstabellen zusammen. Die Faktentabelle enthält einerseits die wichtigen Kennzahlen und andererseits speichert sie die Fremdschlüssel der Dimensionstabellen.

Vorteile

- einfaches, intuitives Datenmodell (Join-Tiefe nicht größer als 1)
- schnelle Anfrageverarbeitung
- Verständlichkeit und Nachvollziehbarkeit

Nachteile

- verschlechterte Antwortzeit bei häufigen Abfragen sehr großer Dimensionstabellen
- Redundanz innerhalb einer Dimensionstabelle
- Aggregationsbildung ist schwierig

Ziel

- Benutzerfreundliche Abfrage

Ergebnis

- Einfaches, lokales und standardisiertes Datenmodell
- Eine Faktentabelle und wenige Dimensionstabellen

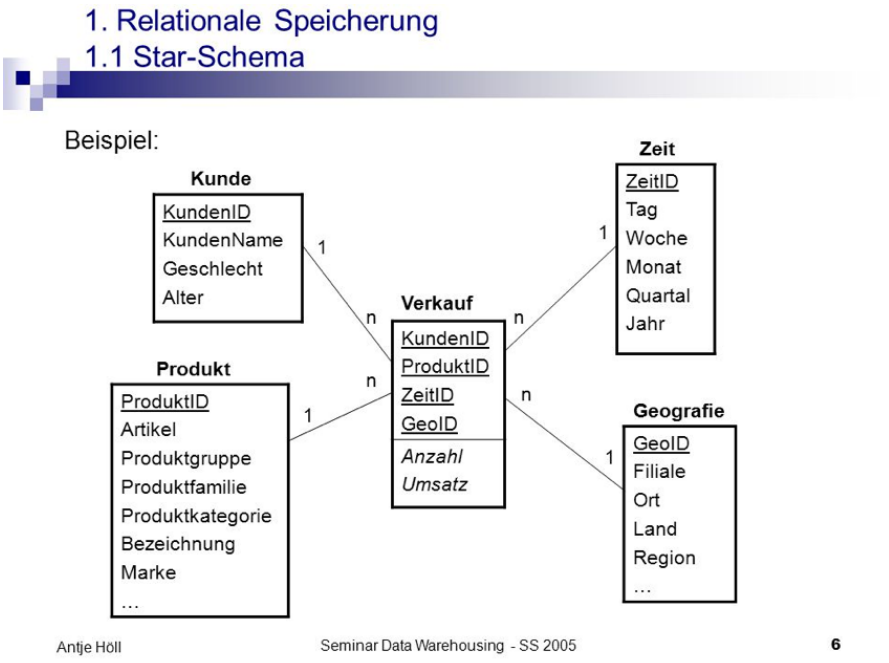


Figure 1.25: Beispiel

1.7.1 Schneeflockenschema (Snowflake-Schema)

Das Snowflake-Schema ist eine weitere Variante Informationen in mehrdimensionalen Datenräumen zu speichern. Dabei ähnelt die Struktur des Datenmodells eines Datenmodells, das sich in der 3. Normalform befindet. Das Snowflake-Schema (Schneeflockenschema) ist eine Weiterführung des Sternschemas. Im Snowflake-Schema wird jede weitere Hierarchiestufe durch eine weitere Datenbanktabelle realisiert. Dadurch steigt

die Anzahl von SQL Joins beim Schneeflockenschema im Gegensatz zum Sternschema linear mit der Anzahl der Aggregationspfade an.

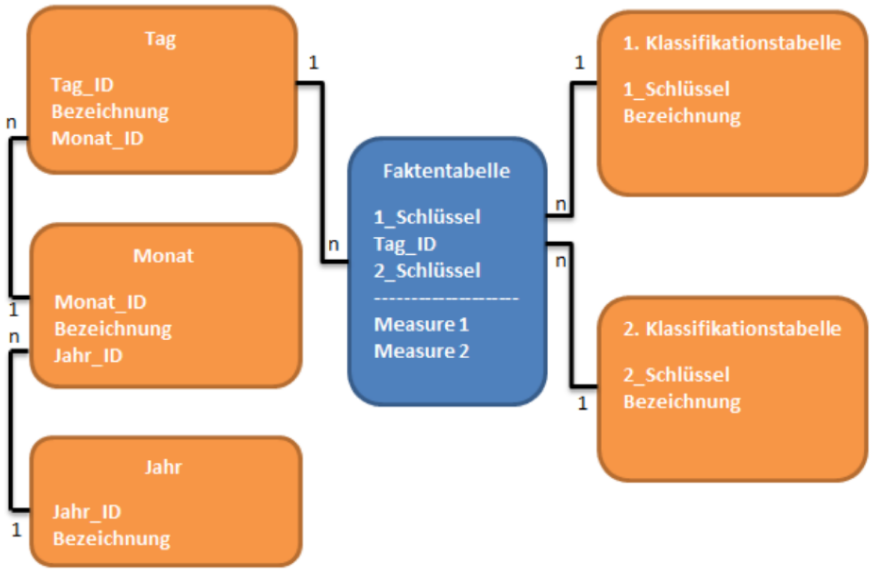


Figure 1.26: Aufbau Snowflake Schema

Die Kennzahlen werden innerhalb einer Faktentabelle gespeichert. In der Faktentabelle sind numerische Fremdschlüsselbeziehungen zu den jeweils niedrigsten Hierarchiestufen der verschiedenen Dimensionen enthalten, gemäß der Granularität des Datenwürfels. Jede weitere Hierarchiestufe wird in einer eigenen Tabelle angelegt und mittels 1:n Beziehungen an die vorherige Hierarchiestufe miteinander verbunden. Durch den Einsatz des Snowflake-Schemas entsteht ein sehr komplexes Datenmodell, welches die Ausprägung einer Schneeflocke hat.

Vorteile

- geringerer Speicherplatzverbrauch (keine redundanten Daten)
- n:m - Beziehungen zwischen Aggregationsstufen können über Relationstabellen aufgelöst werden
- optimale Unterstützung der Aggregationsbildung
- häufige Abfragen über sehr große Dimensionstabellen erbringen Zeitersparnis und Geschwindigkeitsvorteil (Browsing-Funktionalität)

Nachteile

- Komplexere Strukturierung: die Daten sind zwar weniger redundant, die Zusammenhänge sind jedoch komplexer als in einem Star-Schema

- größere Tabellenanzahl
- Reorganisationsproblem: Änderungen im semantischen Modell führen zu umfangreicher Reorganisation der Tabellen und somit zu einem höheren Wartungsaufwand

Ziele

- Redundanzminimierung durch Normalisierung
- Effiziente Transaktionsverarbeitung

Ergebnis

- Komplexes und spezifisches Schema
- Viele Entitäten und Beziehungen bei großen Datenmodellen

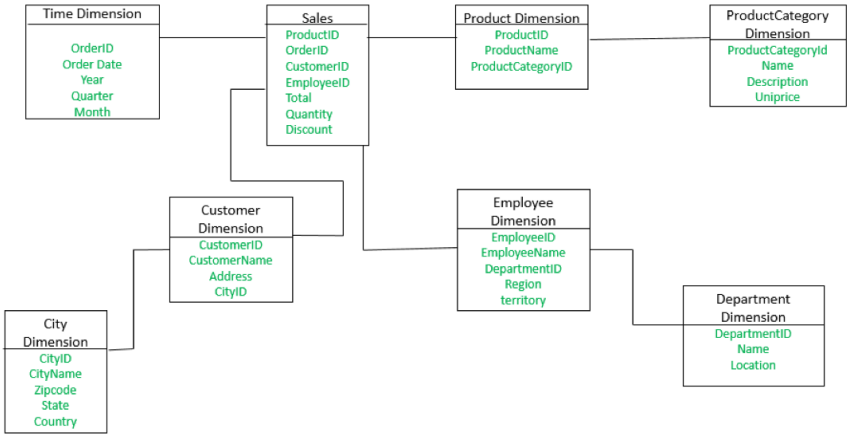


Figure 1.27: Beispiel Snowflake

1.8 Problem der Zeit

Die Zeit kann man im relationalen Datenmodell nicht abbilden, außer man fügt zusätzliche Spalten ein.

Es wird zwischen einer Zeitdauer mit Lücken und einer Zeitdauer ohne Lücken unterschieden.

Beispiel: Tabelle mit Artikel und Preis

Der Kunde Max Meier kauft am 1. Februar 2020 einen Artikel. Drei Wochen später schickt er diesen zurück und möchte sein Geld zurückbekommen. Der Artikel kostet derzeit €89. Man kann jedoch nicht genau sagen, wie viel der Artikel am 1. Februar gekostet hat.

Lösung: In der Datenbank muss ein Feld sein, wo der Preis mit der Zeit eingetragen ist (€89 bis 12.02.2020).

Bei einem zusätzlichen „bis“ Feld muss die Zeit ohne Lücken dargestellt werden. Eine Zeitdauer mit Lücken geht nur, wenn ein „von – bis“ Feld vorhanden ist.

Ähnlich ist es bei einer Materialverfolgung. Hier möchte man beispielsweise wissen, wann sich der Rohstoff X wo befindet. Deshalb muss die Zeit in Kombination mit dem Ort in einem Feld gespeichert werden. In Echtzeit kann das Material aber trotzdem nicht verfolgt werden, da mit dem zusätzlichen Zeit – Ort Feld lediglich feststellen kann, zwischen welchen Zeitpunkten sich das Material an welchem Ort befunden hat.

Chapter 2

Normalisierung

Chapter 3

SQL - DDL

Chapter 4

SQL DCL

Chapter 5

SQL - DML

Chapter 6

PL/SQL

Chapter 7

Gleichzeitigkeit

7.1 Einleitung

Probleme der Gleichzeitigkeit entstehen immer dann, wenn mehrere Benutzer oder Prozesse zeitgleich auf eine Datenbank zugreifen können. Das DBMS muss hierbei einen Kontrollmechanismus bereitstellen, der gewährleistet, dass keine inkonsistenten Zustände auftreten.

7.2 Transaktionen

```
update Konto
set Saldo = Saldo - 500
where KontoNr = 4711
```

```
update Konto
set Saldo = Saldo + 500
where KontoNr = 0815
```

Figure 7.1: Beispiel Banktransaktion

Bei einer Banküberweisung muss sichergestellt werden, dass sowohl die Abbuchung, als auch die Gegenbuchung stattfinden. Wird nur eines der Statements durchgeführt, so gerät die Datenbank in einen inkonsistenten Zustand. Die Buchungen werden daher in einer Transaktion zusammengefasst. Eine Transaktion ist eine logisch zusammenhängende Sequenz von Operationen, die eine Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt. Sie darf nur in ihrer Gesamtheit durchgeführt werden. Schlägt eine der Teiltransaktionen in der Gruppe fehl, werden alle anderen Teilaktionen ebenfalls rückgängig gemacht (Rollback). Transaktionen verhindern, dass die Datenbank in einen inkonsistenten Zustand gerät, wenn ein Problem bei der Durchführung einer der Aktionen auftritt, aus denen sich die Transaktion zusammensetzt.

7.2.1 Die ACID-Eigenschaften von Transaktionen

Bei einer Transaktion muss das Transaktionssystem die ACID-Eigenschaften garantieren und beherrschen:

Atomarität

Eine Transaktion wird vollständig oder gar nicht ausgeführt. Die Transaktion ist unteilbar und wenn Sie abgebrochen wird, hat das keine Auswirkungen auf das System.

Konsistenz

Nach einer Ausübung einer Transaktion muss der Datenbestand wieder konsistent sein und darf keine Anomalien aufweisen.

Isolation

Jede Transaktion arbeitet unabhängig von anderen Transaktionen. Gleichzeitige Transaktionen dürfen sich gegenseitig nicht beeinflussen.

Dauerhaftigkeit

Wenn eine Transaktion die Datenbasis verändert, so gilt diese Änderung anschließend auf Dauer (persistent, permanent).

7.3 Probleme der Gleichzeitigkeit

7.3.1 Lost Update

Zeit	Transaktion A		Preis	Transaktion B	
1	Transaktion liest einen bestimmten Wert „Preis“ (100)...	select	100,-	-	-
2	... um ihn zu analysieren und dann ...	-	100,-	select	In der Transaktion wird genau das gleiche Anwendungsprogramm durchgeführt: also Preis lesen (100)...
3	... abhängig von vielen Faktoren zu verändern. In diesem Fall soll es eine Erhöhung um 10% sein.	update	110,-	-	... analysieren ...
4	...	...	120,-	update	... und um 20% erhöhen.
5	...	...	...	...	...

Figure 7.2: Lost Update

Transaktion A liest einen bestimmten Datensatz. Transaktion B liest denselben Datensatz noch bevor Transaktion A ihn verändert. Sobald Transaktion B den Datensatz ebenfalls verändert, geht die Veränderung von Transaktion A verloren. Sie ist ein Lost Update.

7.3.2 Umcommitted Dependency

Zeit	Transaktion A		Preis	Transaktion B	
1	Transaktion liest einen bestimmten Wert „Preis“ (100)...	select	100,-	-	-
2	... um ihn zu analysieren und dann ...	-	100,-	select	In der Transaktion wird genau das gleiche Anwendungsprogramm durchgeführt: also Preis lesen (100)...
3	... abhängig von vielen Faktoren zu verändern. In diesem Fall soll es eine Erhöhung um 10% sein.	update	110,-	-	... analysieren ...
4	...	...	120,-	update	... und um 20% erhöhen.
5	...	...	...	...	...

Figure 7.3: Umcommitted Dependency

Transaktion A verändert einen Wert, der im Anschluss wieder zurück gerollt wird. Transaktion B liest der Wert noch vor dem Rollback und rechnet mit dem veränderten Wert. Das Problem der Uncommitted Dependency tritt dann auf, wenn eine Transaktion mit einem Wert rechnet der verändert, aber noch nicht committed wurde. Man spricht auf von einem „dirty read“, also vom Lesen, vor der Änderungsbestätigung.

7.3.3 Inconsistent Analysis

Zeit	Transaktion A		Preis	Transaktion B	
1	Transaktion liest einen bestimmten Wert „Preis“ (100)...	select	100,-	-	-
2	... um ihn zu analysieren und dann ...	-	100,-	select	In der Transaktion wird genau das gleiche Anwendungsprogramm durchgeführt: also Preis lesen (100)...
3	... abhängig von vielen Faktoren zu verändern. In diesem Fall soll es eine Erhöhung um 10% sein.	update	110,-	-	... analysieren ...
4	...	...	120,-	update	... und um 20% erhöhen.
5	...	...	...	...	...

Figure 7.4: Inconsistent Analysis

Eine inconsistent Analysis tritt dann auf, wenn eine Transaktion Werte liest und analysiert, die von einer anderen Transaktion verändert werden. Dabei kann es auch zum Phantom-Problem kommen: Die Menge der zu lesenden Objekte werden durch INSERTs / DELETs verändert.

7.4 Locking

Um die Probleme der Gleichzeitigkeit zu verhindern, werden Objekte gesperrt (gelockt). Eine solche Sperre ermöglicht den exklusiven Zugriff eines Prozesses auf eine Ressource, d. h. mit der Garantie, dass kein anderer Prozess diese Ressource liest oder verändert, solange die Sperre besteht.

7.4.1 Locking-Arten

Exclusive Lock (X-Lock, Write-Lock)

Eine Ressource mit einem Exclusive Lock verhindert, dass die Ressource von anderen Prozessen gelesen oder geschrieben wird, da der Prozess, der den Lock gesetzt hat, die Ressource verändern möchte. Das Verfahren wird auch als pessimistisches Locking bezeichnet, da es von der Annahme ausgeht, dass in der Regel eine Aktualisierung der Daten erfolgen wird. Beim optimistischen Locking wird davon ausgegangen, dass in der Regel keine Aktualisierung erfolgt oder eine gleichzeitige Aktualisierung durch zwei Nutzer nicht wahrscheinlich ist. Es wird erst beim Aktualisieren geprüft, ob der Wert verändert wurde.

Shared Lock (S-Lock, Read-Lock)

Besitzt eine Ressource einen Shared Lock, so möchte der Prozess, der diese Sperre gesetzt hat, von der Ressource nur lesen. Somit können auch andere Prozesse auf diese Ressource lesend zugreifen, dürfen diese aber nicht verändern.

Mithilfe von Locking werden die Probleme der Gleichzeitigkeit gelöst:

- Lost Update: Transaktion B bekommt das Schreibrecht erst, wenn Transaktion A die Veränderung vorgenommen hat.
- Uncommitted Dependency: Transaktion B bekommt das Leserecht erst, nachdem Transaktion A zurück gerollt wird.
- Inconsistent Analysis: Transaktion B bekommt das Schreibrecht erst, wenn Transaktion A mit der Analyse fertig ist.

7.4.2 Deadlocks & Strategien zur Vermeidung

Das Setzen einer Sperre kann Deadlocks verursachen, nämlich dann, wenn zwei Prozesse gegenseitig auf die Freigabe der vom jeweils anderen gesperrten Ressource warten.

Two Phase Locking

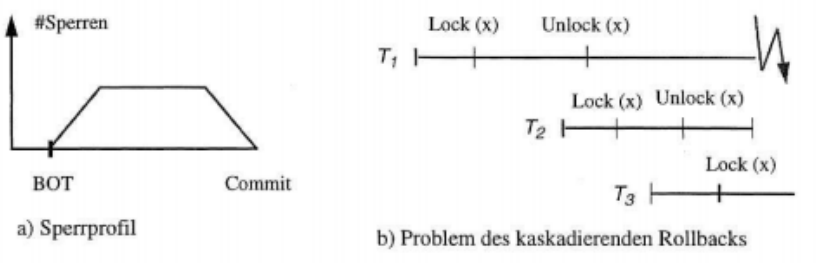
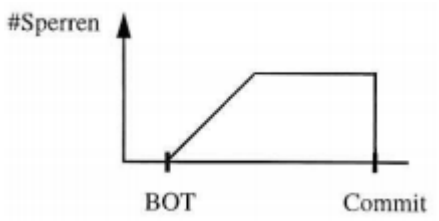


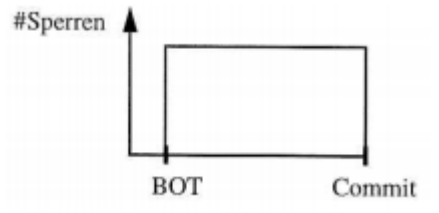
Figure 7.5: Inconsistent Analysis

Das 2-Phasen-Sperrprotokoll ist das gängigste Sperrverfahren und wird in zwei Varianten unterschieden. Die Gemeinsamkeit besteht darin, dass eine Transaktion nach bestimmten Regeln, sogenannten Protokollen abgearbeitet wird. Die zwei Phasen des Protokolls bestehen aus einer Sperrphase, in der alle benötigten Objekte für die Transaktion gesperrt werden. In der zweiten Phase werden die Sperren wieder freigegeben, sodass die Objekte von anderen Transaktionen genutzt werden können.

Wird nun aber eine Transaktion (aus irgendwelchen Gründen) abgebrochen und zurückgerollt, so müssen auch die später durchgeführten Transaktionen kaskadierend zurückgerollt werden.



Strikte Zweiphasigkeit Durch eine strikte Zweiphasigkeit werden alle Sperren erst am Ende einer Transaktion freigegeben. Dadurch kann eine Transaktion ohne Auswirkungen auf andere Transaktionen abgebrochen werden. Ein fortgepflanzter Rollback kann nicht eintreten.



Preclaiming Mit dem Preclaiming werden zu Beginn einer Transaktion alle Objekte gesperrt. Dadurch werden Deadlocks oder Abbrüche durch andere Transaktionen verhindert. Die Schwierigkeit besteht jedoch darin, dass es sehr aufwändig ist, bereits vor einer Transaktion alle benötigten Objekte zu kennen und zu sperren. Preclaiming ist daher für die Praxis nicht relevant.

Transaktions-Scheduling

Dieses Verfahren beschränkt die parallele Durchführung: nur solche Transaktionen dürfen sich zeitlich überlappen, die keine gemeinsamen Datenobjekte behandeln. Damit behindern sich die Transaktionen gegenseitig niemals.
Voraussetzung ist, dass bereits vor dem Ausführen der Operationen bekannt ist, auf welche Datenobjekte zugegriffen wird. Da man dies i.a. nicht „sagen“ bzw. automatisch feststellen kann, sind solche Verfahren pessimistisch: es werden daher oft unnötigerweise Transaktionen von der Verarbeitung ausgeschlossen.

Zeitmarken-Verfahren

Die Grundidee dabei ist, dass jede Transaktion einen Zeitstempel (timestamp) zur eindeutigen Identifikation erhält. Diese Zeitmarke (= Transaktions-Startzeitpunkt) ordnet der Transaktion gleichzeitig eine bestimmte Priorität zu: je älter oder jünger eine Transaktion ist, desto eher kommt sie zum Zuge. Konflikte werden nach einem genau definierten Schema durch eine Kombination aus Warten (Wait) und Restart (Rollback and Retry) jeweils einer der im Konflikt befindlichen Transaktion gelöst.

Wenn die Transaktion A einen von B bereits gesperrten Datensatz sperren will, dann passiert je nach gewähltem Verfahren folgendes:

- **Wait-Die:** Falls A älter ist als B, wartet A sonst (falls A jünger als B) wird A zurückgenommen
- **Wound-Wait:** Falls A älter ist als B, wird B zurückgenommen sonst (falls A jünger als B) wartet A

Timeout

Eine weitere recht gebräuchliche Methode ist die Angabe eines sogenannten Time-Outs: auch hier wird für jede Transaktion vermerkt, wann sie gestartet wurde. Ist sie nach einer bestimmten, als Systemparameter einstellbarer Dauer nicht beendet, so wird angenommen, dass sie in einem Deadlock steckt – sie wird zurückgenommen.

7.5 Serialisierbarkeit

In Transaktionssystemen existiert ein Ausführungsplan für die parallele Ausführung mehrerer Transaktionen. Der Plan wird auch Historie genannt und gibt an, in welcher Reihenfolge die einzelnen Operationen der Transaktion ausgeführt werden. Als serialisierbar wird eine Historie bezeichnet, wenn sie zum selben Ergebnis führt wie eine nacheinander (seriell) ausgeführte Historie über dieselben Transaktionen. Überprüft werden kann die Serialisierbarkeit mithilfe eines Precedence Graphs.

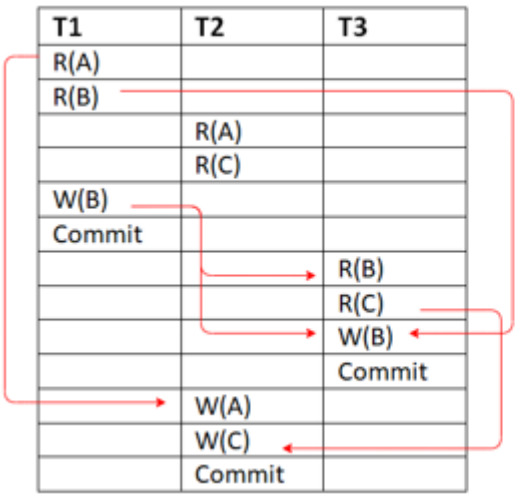
7.5.1 Precedence Graph

Eine Menge von Datenbank-Transaktionen lässt sich genau dann serialisieren, wenn der zugehörige Serialisierbarkeitsgraph zyklenfrei ist.

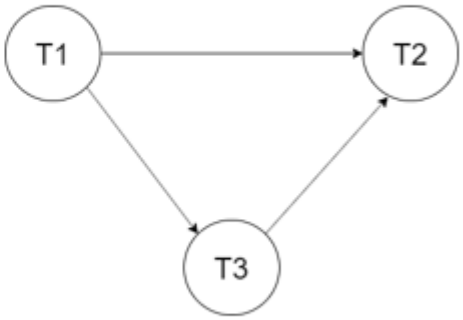
Relegn für das Zeichnen

- Transaktionen werden durch Knoten dargestellt
- Pfeile zwischen Knoten für:
  - R1(A) $\longrightarrow$ W2(A)
  - W1(A) $\longrightarrow$ W2(A)
  - W1(A) $\longrightarrow$ R2(A)

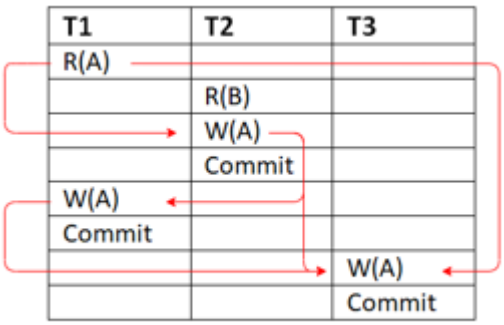
Beispiel 1 - serialisierbar



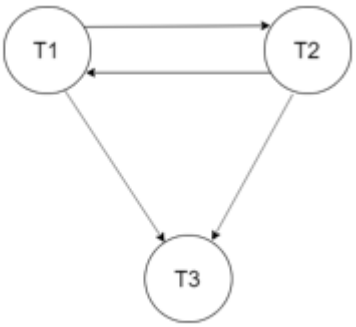
T1	T2	T3
R(A)		
R(B)		
	R(A)	
	R(C)	
W(B)		
Commit		
		R(B)
		R(C)
		W(B)
		Commit
	W(A)	
	W(C)	
	Commit	



Beispiel 2 - nicht serialisierbar



T1	T2	T3
R(A)		
	R(B)	
	W(A)	
	Commit	
W(A)		
Commit		
		W(A)
		Commit



7.6 Anhang Prof. Kainerstorfer

Wie prüft man einen Schedule auf Deadlocks?

Man stellt die einzelnen Logs auf und prüft den Wartegraphen. Ein Log ist eine Folge von Transaktionen auf ein bestimmtes Objekt. Folgendes Beispiel soll das Vorgehen erläutern.

Folgender Schedule ist gegeben:

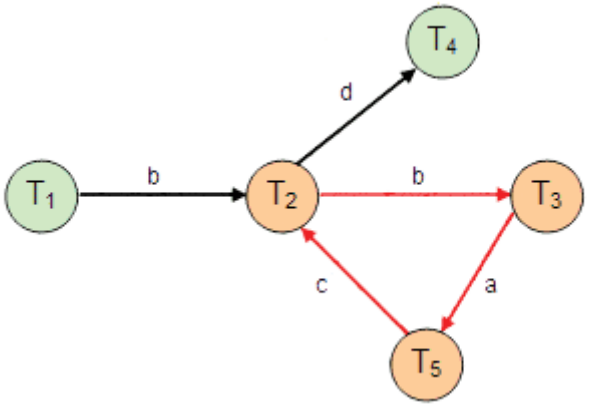
S[r1(b),r2(b),w2(b),r2(c),r2(d),r3(b),r4(d),w3(a),w4(d),r5(a),r5(c),w2(c)]

Aufstellen der Logs für die beteiligten Objekte a,b,c und d

log(a)	log(b)	log(c)	log(d)
w3	r1	r2	r2
r5	r2	r5	r4
	w2	w2	w4
	r3		

Den Wartegraphen nach dem extrahierten Log aufstellen

- Zeichne für jede Transaktion einen Knoten
- Wiederhole für jeden Log in welchem mindestens eine Transaktion schreibt und zwei lesend zugreifen
  - Verbinde jede Leseoperation r vor dem Write mit dem Knoten der Schreibertransaktion
  - Verbinde jede Schreibertransaktion mit jedem danach folgenden Readbefehl
  - Schlingen sind zwecklos und können dabei ignoriert werden



Wartegraph auf Zyklus prüfen, denn mit ist der Schedule nicht serialisierbar

Wie wir sehen enthält der Graph einen Zyklus. Dies bedeutet, daß die genannten Bedingungen in den Logs nicht gelten. Und zwar das die Reihenfolgen der Zugriffe in allen Logs gleich ist. Dies ist hier nicht der Fall. Denn was nicht sein darf, ist
T2 vor T3 kommt, T3 vor T5 komt und T5 vor T2 kommt
Deshalb erzeugen T2, T3 und T5 einen Deadlock.

Chapter 8

Indizes

Chapter 9

Portieren von Projekten

Chapter 10

Datenbank Tuning

Chapter 11

Data Warehousing

Chapter 12

Struktur und Aufbau einer Datenbank anhand von Oracle

Chapter 13

Datenintegrität

Chapter 14

Trigger

Chapter 15

JPA

Chapter 16

Apex

Chapter 17

Verteilte Datenbanken

Chapter 18

OracleNet

Chapter 19

XML

Chapter 20

Objektorientierte und Objektrelationale Datenbanken

Chapter 21

Datenbank-Entwurfstools

Chapter 22

Data Analytics

Chapter 23

Docker

Chapter 24

Bäume

Chapter 25

NoSQL