DBI Zusammenfassung

Leon Schlömmer

7. April 2020

Inhaltsverzeichnis

Kapitel 1

Modellierung

1.1 Entwurfsmethodik

Modellieren ist der Weg von der realen Welt bis zum Datenmodell. Der Entwurf eines Datenmodells geschieht in mehreren Schritten:

1. Anforderungsanalyse
2. Konzeptionelles Modell erstellen
3. Implementierung überlegen
4. Normalisieren

1.1.1 Abstraktionsebenen

Konzeptionelle Ebene

In der konzeptionellen Ebene erstellt man ein konzeptionelles Modell. Meist wird ein Entity-Relationship Modell verwendet. Hierbei werden Gegenstände zu Gegenstandsmengen abstrahiert, Beziehungen zu Beziehungstypen abstrahiert, und den Gegenstandsmengen und Beziehungstypen werden Attribute zugeordnet.

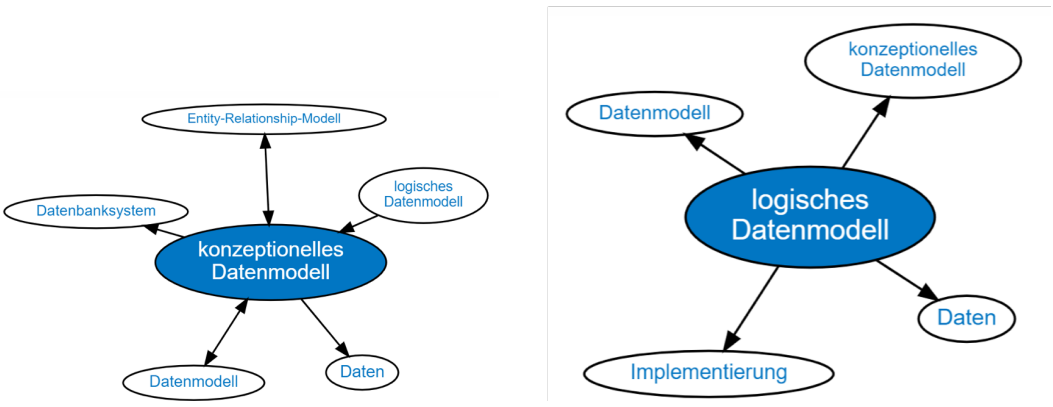
Das konzeptionelle Modell ist Implementierungs-unabhängig.

Implementationsebene

Die Datenbankanwendung wird im DBMS nach dem konzeptionellen Modell modelliert. Hier entsteht ein relationales Modell.

Physische Ebene

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



(a) Konzeptuelles Datenmodell (b) Logisches Datenmodell

Abbildung 1.1: Grafiken zu den Datenmodellen

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 Anforderungsanalyse

Die Anforderungsanalyse ist der erste Schritt von der zu modellierenden Welt (= Miniwelt) zum Datenmodell. Nötige Informationen der Miniwelt werden betrachtet, Bestandteile der Informationsstrukturanforderungen sind:

- Objekte werden zu Objekttypen
- Attribute werden Objekttypen zugeordnet
- Beziehungen zwischen Objekten werden zu Beziehungstypen

1.2.2 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. 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.*

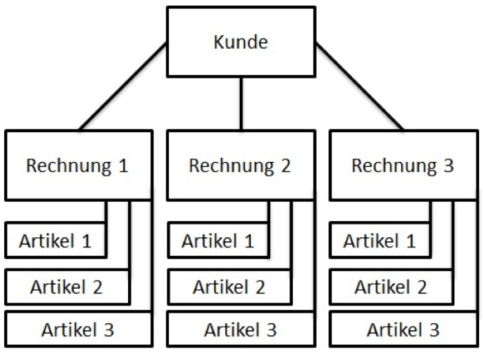


Abbildung 1.2: Stablinienorganisation

1.2.3 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.

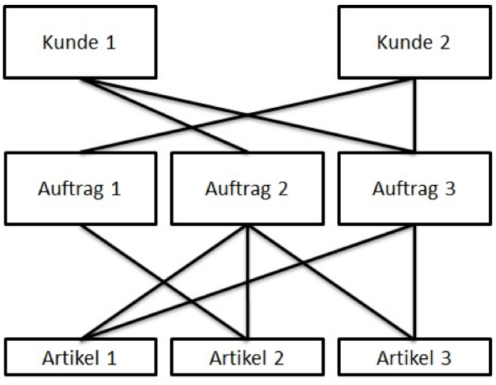


Abbildung 1.3: Netzwerk Datenmodell

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

1.2.4 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. Mithilfe des Entity-Relationship-Modells soll eine Typisierung von Objekten, ihrer relationalen Beziehungen untereinander und der zu überführenden Attribute, stattfinden.

Bestandteile des ERM

- **Entities (Gegenstände)**: Unterscheidbare Konzepte der Miniwelt
- **Beziehung zwischen Entities**: Eine Verknüpfung / Zusammenhang zwischen Entitäten. Die Abstrahierung erfolgt analog zu Entities.
- **Attribut:** Eine Eigenschaft, die im Kontext zu einer Entität steht.

Abbildung 1.4: UML Darstellung

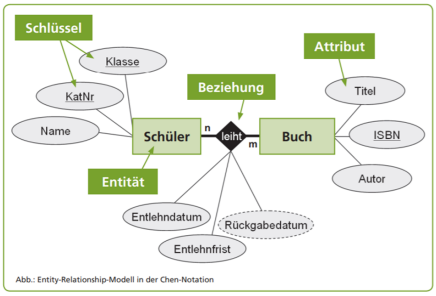


Abbildung 1.5: Was ist was

Schlüssel ERM haben weiters Schlüssel. Schlüssel sind eine minimale Menge von Attributen, deren Werte das zugeordnete Entity innerhalb aller Entities seines Typs eindeutig identifizieren.

Surrogatschlüssel / künstlicher Schlüssel Der Surrogatschlüssel ist ein Attribut welches künstlich als Schlüssel eingebaut wird.

Beispiel

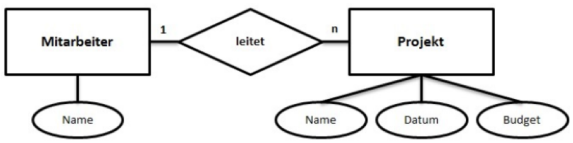


Abbildung 1.6: Beispiel Darstellung

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

1.2.5 Relationales Datenmodell

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

Im relationalen Modell werden:

- Relationen als flache Tabellen abgebildet
- Datenobjekte als Zeilen abgebildet
- gespeicherte Daten durch Operationen Mengenorientiert verknüpft und verarbeitet

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

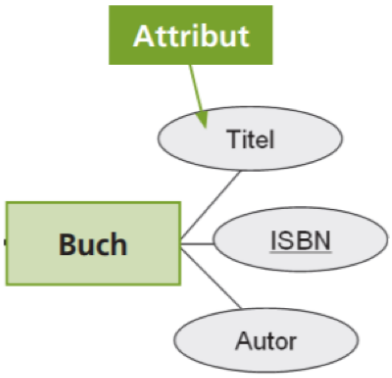


Abbildung 1.7: Attribute einer Entity

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.

1.2.6 Beziehungen im Relationalen Modell

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)

1.2.7 Kardinalität - Funktionalitäten der Beziehungen

Die Kardinalität zwischen dem Entitätstyp E1 und dem Entitätstyp E2 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

Jedem Entity aus einer Entitymenge E1 ist maximal ein Entity aus E2 zugewiesen

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.

1:n Beziehung

Jedem Entity e1 aus einer Entitymenge E1 können beliebig viele Entities aus einer anderen Entitymenge E2 zugeordnet werden.

Jedem Entity aus der Entitymenge E2 ist maximal 1 Entity aus E1 zugeordnet. Es kann auch Entities ohne Partner geben.

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.

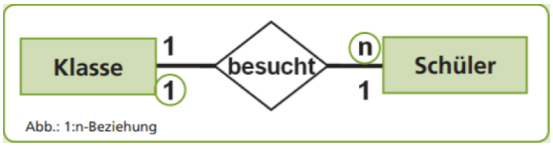
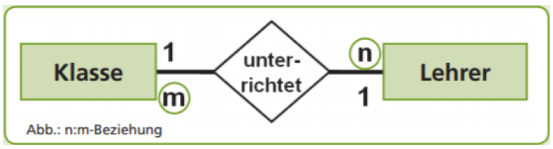


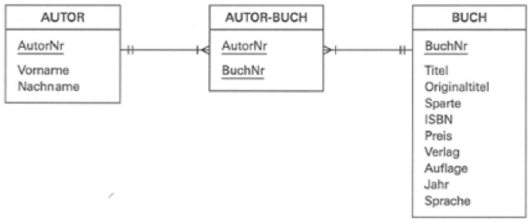
Abbildung 1.8: Eine Klasse hat mehrere Schüler. Ein Schüler besucht 1 Klasse

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.



(a) Eine Klasse wird von mehreren Lehrern unterrichtet. Ein Lehrer unterrichtet mehrere Klassen



(b) Verbindungstabelle zwischen Autor und Buch

Abbildung 1.9: Grafiken zu den Datenmodellen

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). 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.

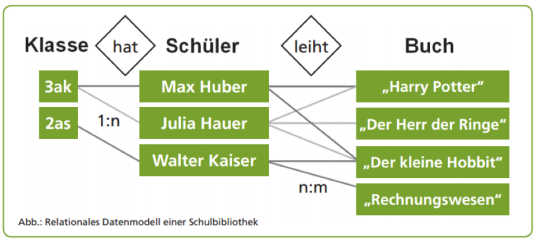


Abbildung 1.10

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

Fremdschlüssel = Schlüsselattribute fremder Entitytypen in einer Relation

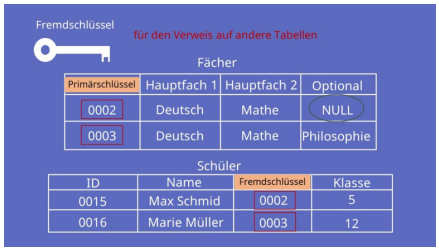


Abbildung 1.11: Foreign Key

1.2.8 Überleitung eines ERM's in ein relationales Modell

Ausgangspunkt ist ein vollständiges ER Modell. 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.

Grundsätzlich werden einfach alle Entitytypen und Beziehungstypen in eigenen Relationen abgebildet.

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

Darstellung von Beziehungen

Im Initialentwurf wird für jeden Beziehungstyp eine eigene Relation definiert. 1:1 Relationen und 1:n Relationen können in der Verfeinerung zusammengefasst werden.

Im Initialentwurf enthält eine Relation alle Schlüsselattribute seiner Entitytypen und zusätzliche Attribute die der Relation zugeordnet wurden

Schemaverfeinerung

Relationen die 1:1 und 1:N Beziehungstypen repräsentieren können eliminiert werden, indem sie zusammengefasst werden. Nach der Verfeinerung enthält die Relation R1 den Schlüssel einer anderen Relation um den Beziehungstyp abzubilden.

1.2.9 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.

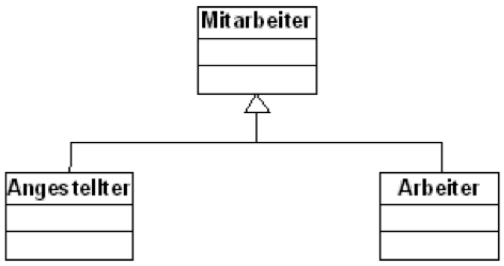


Abbildung 1.12: Vererbung bei Objekten

1.3 Aggregation

1.3.1 „is-part-of“-Beziehung (ERM)

Bei der Aggregation werden Unterschiedliche Entitytypen, die in ihrer Gesamtheit einen Objekttypen bilden, einander zugeordnet.

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.

1.4 Generalisierung / Spezialisierung

Eigenschaften von Entitytypen mit ähnlichen Attributen und Beziehungen werden herausfaktorisiert und einem gemeinsamen Obertyp zugeordnet.

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

Die Generalisierung ist Teil des konzeptionellen Entwurfs. Weiters ist sie eine Abstraktion auf Typebene. Die Untertypen sind eine Spezialisierung des Obertyps. Die **Vererbung** ist ein Schlüsselkonzept der Generalisierung, da ein Untertyp alle Eigenschaften des Obertyps erbt.

Diese Beziehungstypen werden durch „is-a“ — „can-be“ beschrieben.

1.4.1 Im Relationalen Modell

Sowohl der Obertyp als auch die Untertypen haben eine eigene Relation. Die Informationen einer Entity werden also in unterschiedlichen Relationen gespeichert. Streng genommen ist die Vererbung im relationalen Modell nicht realisiert, da in einer Spezialisierung nicht die vollständige Information verfügbar ist.

1.4.2 Beispiel:

Flugreise is-a Reise
und in anderer Leserichtung:
Reise can-be Flugreise

1.4.3 Verdeutlichung des Unterschieds Generalisierung / Spezialisierung

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 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).

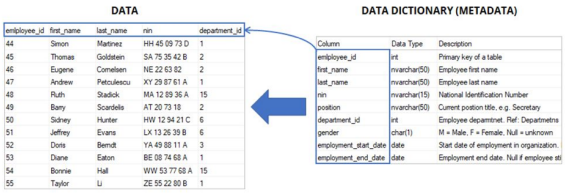


Abbildung 1.13

1.5.1 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:**

```
100 SELECT object\_name, object\_type FROM USER\_OBJECTS;
```

Alle Objekte auf die der Benutzer Zugriff hat:

```
100 SELECT owner, object\_name, object\_type FROM ALL\_OBJECTS;
```

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

```
100 SELECT owner, object\_name, object\_type FROM SYS.DBA\_OBJECTS;
```

1.6 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.

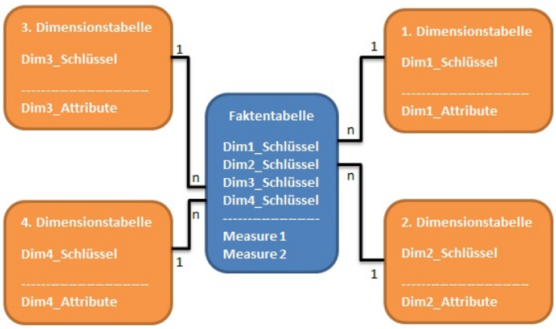


Abbildung 1.14: Aufbau

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

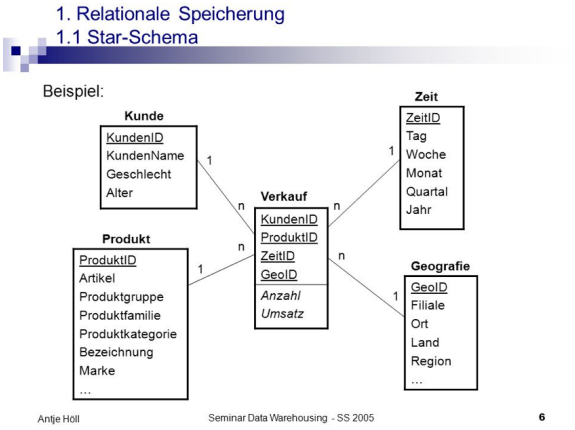


Abbildung 1.15: Beispiel

1.6.1 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.

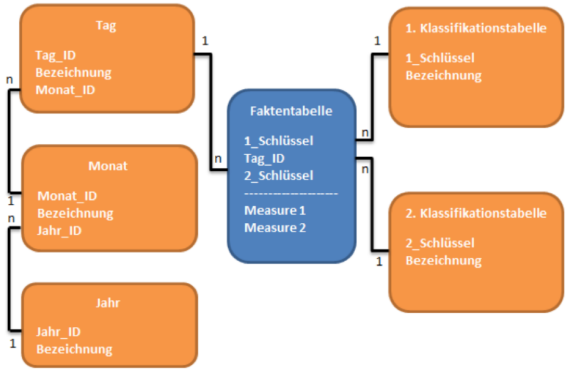


Abbildung 1.16: Aufbau Snowflake Schema

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

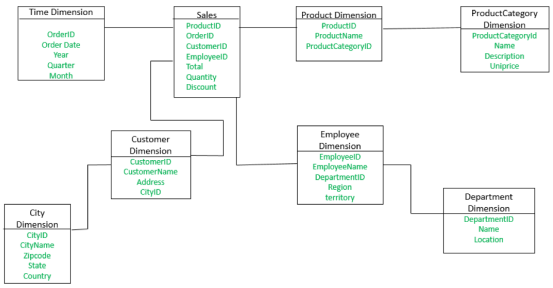


Abbildung 1.17: Beispiel Snowflake

1.7 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.

Kapitel 2

Normalisierung

Bei der Normalisierung werden Relationsschemata verfeinert. Durch die Normalisierung sollen Redundanzen verhindert werden, und es werden Leistungs- und Speichereffizienz angestrebt.

2.1 Anomalien

Zu Beginn muss geklärt werden welche Folgen redundant gespeicherte Daten haben und wieso normalisiert werden soll. Schlecht entworfene Relationsschemata führen zu Anomalien. Redundanzen entstehen, wenn verschiedene Entitytypen in einer Relation vermischt sind. Anomalien sind darauf zurückzuführen, dass nicht zusammenpassende Informationen in einer Relation gebündelt wurden.

2.1.1 Update-Anomalien

Wird eine redundant gespeicherte Information verändert, kann es passieren dass Einträge übersehen werden, und so manche Einträge unverändert bleiben.

Selbst wenn man garantieren kann, dass alle Einträge gleichzeitig ändern kann hat man folgende Nachteile:

- erhöhter Speicherbedarf
- Leistungseinbußen bei Änderungen

2.1.2 Einfüge-Anomalien

Sind mehrere Entitytypen in einer Relation vermischt, treten Probleme auf, wenn man neue Informationen, die nur zu einem Entitytyp gehören, einfügt.

Man müsste die Attribute der anderen Entitytypen mit NULL-Werten besetzen.

2.1.3 Löschanomalien

Sind mehrere Entitytypen in einer Relation vermischt, treten Probleme auf, wenn man Informationen, die nur zu einem Entitytypen gehören, löschen will.

Es kann zu einem unbeabsichtigten Verlust der anderen Entitytypen kommen.

Dies wäre nur zu vermeiden, wenn man die Werte der zu löschenden Attribute mit NULL-Werten besetzt. Sind allerdings die nicht zu löschenden Informationen redundant gespeichert, wäre das nicht nötig.

2.2 Funktionale Abhängigkeiten

Eine funktionale Abhängigkeit ist, wenn die Werte einer Menge von Attributen die Werte einer anderen Menge von Attributen bestimmt.

So ist zum Beispiel das Attribut Sozialversicherungsnummer ein Attribut, welches die Attribute Name und Alter in der Tabelle Person bestimmt.

Funktionale Abhängigkeiten sind Konsistenzbedingungen, die in jedem gültigen Datenbankzustand eingehalten werden müssen.

2.2.1 Schlüssel

Funktionale Abhängigkeiten sind eine Verallgemeinerung des Schlüsselbegriffs.

Superschlüssel

Ein Schlüssel ist ein Superschlüssel, wenn die Werte aller anderen Attribute der Relation von ihm abhängen. Die Menge der Schlüsselattribute muss nicht minimal klein sein.

Volle Funktionale Abhängigkeit

Ein Schlüssel ist eine volle funktionale Abhängigkeit, wenn kein Attribut mehr aus der Schlüsselmenge entfernt werden kann, ohne den Schlüssel zu zerstören.

2.3 Zerlegung von Relationen

Um einen schlechten Entwurf zu verbessern, werden bei der Normalisierung Relationen aufgespalten.

2.3.1 Korrektheitskriterien

Bei der Zerlegung von Relation sollten folgende Kriterien eingehalten werden:

- Verlustlosigkeit: Die Daten der ursprünglichen Relationen müssen aus den Daten der neuen Relationen rekonstruierbar sein
- Abhängigkeitserhaltung: Die auf die ursprüngliche Relation geltenden funktionalen Abhängigkeiten müssen auf die neuen Schemata übertragbar sein

Verlustlosigkeit

Eine Zerlegung ist verlustlos, wenn alle Information durch eine Join wieder rekonstruierbar sind.

Eine Zerlegung ist nicht verlustlos, falls durch den Join der neuen Relationen Tupel entstehen, die in der ursprünglichen Relation nicht vorhanden waren.

Abhängigkeitserhaltung

Die Überprüfung aller Abhängigkeiten muss erfolgen können, ohne dass Joins notwendig sind.

2.4 Normalformen

2.4.1 Erste Normalform

Die erste Normalform verlangt, dass alle Attribute einen atomaren Wertebereich haben.

Es sind:

- zusammengesetzte
- mengenwertige
- relationenwertige

Attributdomänen (= Wertebereich) sind nicht zulässig.

2.4.2 Zweite Normalform

Attribute, die in keinem Schlüssel vorkommen, müssen von *jedem* Kandidatenschlüssel **voll funktional** abhängen.

Eine Relation in zweite Normalform bringen

Relation in mehrere Teilrelationen zerlegen, die beide die zweite Normalform erfüllen

2.4.3 Dritte Normalform

Die dritte Normalform verlangt, dass für jede funktionale Abhängigkeit einer Relation mindestens eine von 3 Bedingungen gilt:

- das Abhängige Attribut ist im Schlüssel enthalten
- das Abhängige Attribut ist Teil eines anderen Schlüssels
- die funktionale Abhängigkeit ist ein Superschlüssel der Relation, das heisst, er bestimmt alle Attributwerte der Relation

Die dritte Normalform ist verletzt, wenn ein Attribut nicht von jedem Kandidatenschlüssel abhängt und die Relation keinen Superschlüssel hat.

Eine Relation in dritte Normalform bringen

Relationen aufteilen, sodass eine der drei oberen Bedingungen gilt.

2.4.4 Boyce-Codd Normalform

die Schwierigkeit beim Finden einer Zerlegung die in BCNF ist, liegt dabei eine Zerlegung zu finden welche Abhängigkeitserhaltend ist.

Eine Relation ist in BCNF, wenn jede Abhängigkeit entweder trivial ist, oder die Abhängigkeit ein Superschlüssel der Relation.

2.4.5 Vierte Normalform

Die 4. Normalform ist nicht mehr Abhängigkeitserhaltend.

2.4.6 Arten der Funktionalen Abhängigkeit

- Triviale FA: Die abhängigen Attribute sind im Schlüssel enthalten
- Volle FA: Die Menge an Attributen im Schlüssel ist minimal
- Superschlüssel: Der Schlüssel bestimmt alle Attribute einer Relation
- Transitive Abhängigkeit: Eine transitive Abhängigkeit liegt dann vor, wenn Y von X funktional abhängig und Z von Y, so ist Z von X funktional abhängig. Das heißt, dass Nichtschlüsselattribute funktional abhängig zu anderen Nichtschlüsselattributen sind.

2.5 Nachteile der Normalisierung

Obwohl die Normalisierung Vorteile wie die Konsistenz der Daten, die effiziente Organisation der Daten usw., kann sie jedoch auch einige Nachteile mit sich bringen.

So speichert eine vollständig normalisierte Datenbank zusammengehörige Daten nicht in einer, sondern in mehreren verschiedenen, miteinander verknüpften Tabellen.

Für Anwender ist das Datenbankschema auf den ersten Blick unübersichtlicher. Um die zusammengehörenden Informationen zu erhalten, ist die Zusammenführung der Daten aus den verschiedenen Tabellen notwendig. Die Datenabfragen arbeiten mit Joins und sammeln die gewünschten Informationen aus den einzelnen Tabellen. Komplexe und umfangreiche Joins führen zu einer starken Belastung des Datenbanksystems.

Ein weiterer Nachteil ist, dass die Normalisierung einen erheblichen Aufwand darstellen kann. Er steigt zusätzlich, wenn die Normalisierung nicht schon beim Entwurf des relationalen Datenbankschemas stattfindet, sondern bei einer bereits mit Daten gefüllten Datenbank.

Kapitel 3

SQL - DDL

Kapitel 4

SQL DCL

4.1 Was ist DCL?

Die *Data Control Language* (DCL), auch Datenüberwachungssprache genannt, ist eine Teilmenge der Structured Query Language (SQL) und ermöglicht es Datenbankadministratoren, den Sicherheitszugriff auf relationale Datenbanken zu konfigurieren. Eine „vollwertige“ SQL-Datenbank enthält umfassende Regelungen über die Vergabe von Rechten für den Zugriff auf Objekte (Tabellen, einzelne Felder, interne Funktionen usw.).

DCL ist die einfachste der SQL-Teilmengen, da sie nur aus drei Befehlen besteht: GRANT, REVOKE und DENY. Zusammen bieten diese drei Befehle Administratoren die Flexibilität, Datenbankberechtigungen äußerst detailliert festzulegen und zu entfernen. Es gibt auch einige Software-Hersteller die den Begriff DCL nicht verwenden, stattdessen werden da die Berechtigungsbefehle zur DDL gezählt.

4.2 Wer kann den Benutzern diese Rechte gewähren oder von ihnen entfernen?

Nur Datenbankadministratoren (DBAs) oder Datenbankeigentümer sind berechtigt, den Benutzern Berechtigungen zu erteilen oder diese zu entfernen. Andere Benutzer müssen ausdrücklich zu einzelnen Handlungen ermächtigt werden.

4.3 Privilegien

Wenn mehrere Benutzer auf Datenbankobjekte zugreifen können, kann die Berechtigung für diese Objekte mit Berechtigungen gesteuert werden. Jedes Objekt hat einen Besitzer. Berechtigungen steuern, ob ein Benutzer ein Objekt ändern kann, dessen Eigentümer ein anderer Benutzer ist. Berechtigungen werden entweder vom Instanz Administrator, einem Benutzer mit der Berechtigung ADMIN oder für Berechtigungen für ein bestimmtes Objekt vom Eigentümer des Objekts erteilt oder entzogen.

4.3.1 System Privilegien

Systemberechtigungen sind Berechtigungen, mit denen Benutzer bestimmte Funktionen ausführen können, die sich mit der Verwaltung der Datenbank und des Servers befassen. Die meisten verschiedenen Arten von Berechtigungen, die von den Datenbankanbietern unterstützt werden, fallen unter die Kategorie der Systemberechtigungen. Nur der Instanz Administrator oder ein Benutzer mit ADMIN - Berechtigungen kann Systemberechtigungen erteilen oder entziehen.

Beispiele für System Privilegien:

- *CREATE USER* - Wenn die CREATE USER-Berechtigung einem Datenbankbenutzer erteilt wird, kann dieser Datenbankbenutzer neue Benutzer in der Datenbank erstellen.
- *CREATE TABLE* - Mit der CREATE TABLE-Berechtigung, die einem Datenbankbenutzer erteilt wird, kann dieser Datenbankbenutzer Tabellen in seinem eigenen Schema erstellen. Diese Art von Berechtigungen steht auch für andere Objekttypen zur Verfügung, z. B. gespeicherte Prozeduren und Indizes.

- *CREATE SESSION* - Wenn die CREATE SESSION-Berechtigung einem Datenbankbenutzer erteilt wird, kann dieser Datenbankbenutzer eine Verbindung zur Datenbank herstellen.

4.3.2 Objekt Privilegien

Ein Objektprivileg ist das Recht, eine bestimmte Aktion an einem Datenbankobjekt auszuführen oder auf das Datenbankobjekt eines anderen Benutzers zuzugreifen - wobei es sich bei Datenbankobjekten um Tabellen, gespeicherte Prozeduren, Indizes usw. handelt. Der Eigentümer eines Objekts verfügt über alle Objektberechtigungen für dieses Objekt, und diese Berechtigungen können nicht widerrufen werden. Der Eigentümer des Objekts kann anderen Datenbankbenutzern Objektberechtigungen für dieses Objekt erteilen. Ein Benutzer mit ADMIN - Berechtigungen kann Objektberechtigungen von Benutzern erteilen und entziehen, die nicht Eigentümer der Objekte sind, für die die Berechtigungen erteilt wurden.

Beispiele für Objekt Privilegien:

- *SELECT* - Ermöglicht einem Benutzer die Auswahl aus einer Tabelle, einer Sequenz, einer Ansicht usw.
- *UPDATE* - Ermöglicht einem Benutzer das Aktualisieren einer Tabelle.
- *DELETE* - Ermöglicht einem Benutzer das Löschen aus einer Tabelle.

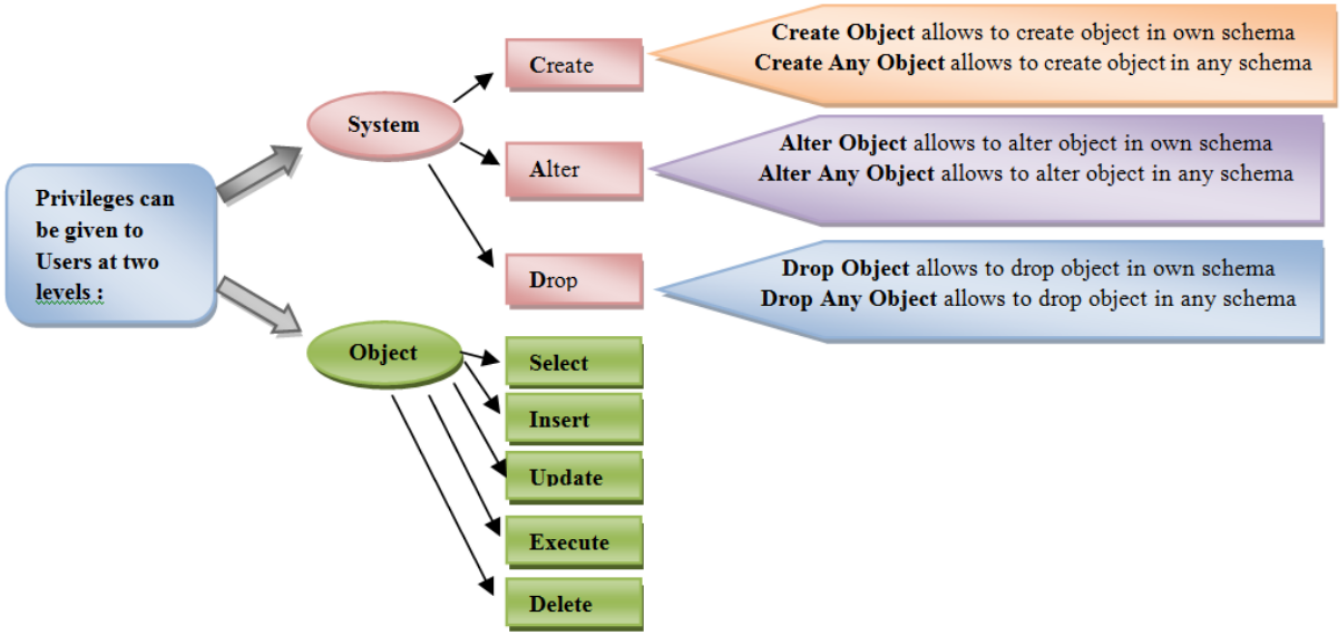


Abbildung 4.1: Privilegien

4.4 Hinzufügen von Berechtigungen mit GRANT

Der Befehl GRANT wird von Administratoren verwendet, um einem Datenbankbenutzer neue Berechtigungen hinzuzufügen. Es hat eine sehr einfache Syntax, die wie folgt definiert ist:

4.4.1 Syntax

Hier ist der Überblick über die einzelnen Parameter, die Sie mit diesem Befehl bereitstellen können:

- *Privilege*: Dies kann entweder das Schlüsselwort ALL (um eine Vielzahl von Berechtigungen zu erteilen) oder eine bestimmte Datenbankberechtigung oder eine Reihe von Berechtigungen sein. Beispiele sind CREATE DATABASE, SELECT, INSERT, UPDATE, DELETE, EXECUTE, CREATE VIEW usw.
- *Object*: Kann ein beliebiges Datenbankobjekt sein. Die gültigen Berechtigungsoptionen hängen vom Typ des Datenbankobjekts ab, das Sie in diese Klausel aufnehmen. In der Regel handelt es sich bei dem Objekt entweder um eine Datenbank, eine Funktion, eine gespeicherte Prozedur, eine Tabelle oder eine Ansicht.

```
GRANT [privilege]
ON [object]
TO [user]
[WITH GRANT OPTION]
```

Abbildung 4.2: GRANT Syntax

- *User*: Kann ein beliebiger Datenbankbenutzer sein. Sie können in dieser Klausel auch eine Rolle für den Benutzer ersetzen, wenn Sie die rollen basierte Datenbanksicherheit nutzen möchten.Verwenden Sie das Schlüsselwort PUBLIC, um alle Benutzer anzugeben.

4.4.2 With grant option and With admin option

Es gibt einen sehr deutlichen Unterschied, und es ist wichtig, die Auswirkungen der Verwendung von entweder "with grant optionöder "with admin optionßorgfältig zu prüfen. Diese Option gibt die Kontrolle über die Datenbankberechtigungen von einem einzelnen Datenbankbesitzer an mehrere Benutzer weiter. Dies muss sorgfältig kontrolliert werden, um die richtige Sicherheit zu gewährleisten.

Die Verwendung der Optionen "with adminünd "with grant"wird als Oracle-gefährlich eingestuft, da es bei unsachgemäßer Verwaltung zu unbeabsichtigten Nebenwirkungen kommen kann, die zu einer Sicherheitslücke führen. Sowohl die Optionen 'with grant' als auch 'with admin' dienen dazu, die zentrale Sicherheitskontrolle aufzugeben, gelten jedoch für verschiedene Arten von Berechtigungen.

With Grant Option

- Nur für Objektprivilegien, keine Systemprivilegien.
- Nur die Person, die das Privileg erteilt hat, kann das Privileg widerrufen.
- Aufgehobene Berechtigungen können "kaskadieren", sodass der erste Berechtigte viele nachfolgende Berechtigungen widerrufen kann.

With admin option

- Nur für Systemprivilegien, keine Objektprivilegien.

4.5 Datenbankzugriff widerrufen mit REVOKE

Der Befehl REVOKE wird verwendet, um den Datenbankzugriff von einem Benutzer zu entfernen, dem zuvor ein solcher Zugriff gewährt wurde. Die Syntax für diesen Befehl ist wie folgt definiert:

4.5.1 Syntax

```
REVOKE [GRANT OPTION FOR] [permission]
ON [object]
FROM [user]
[CASCADE]
```

Abbildung 4.3: REVOKE Syntax

- *Permission*: Gibt die Datenbankberechtigungen an, die vom identifizierten Benutzer entfernt werden sollen. Der Befehl widerruft sowohl GRANT- als auch DENY - Zusicherungen, die zuvor für die identifizierte Berechtigung gemacht wurden.
- *Object, User*: Das gleiche wie beim GRANT Kommando.

- *Grant option for*: Die Klausel entfernt die Fähigkeit des angegebenen Benutzers, anderen Benutzern die angegebene Berechtigung zu erteilen. Anmerkung: Wenn Sie die Klausel GRANT OPTION FOR in eine REVOKE - Anweisung aufnehmen, wird die primäre Berechtigung nicht widerrufen. Diese Klausel widerruft nur die GRANT Fähigkeit.
- *Cascade*: Entzieht allen Benutzern, denen der angegebene Benutzer die Berechtigung erteilt hat, die angegebene Berechtigung.

4.6 Verweigerung des Datenbankzugriffs mit DENY

Mit dem Befehl DENY wird explizit verhindert, dass ein Benutzer eine bestimmte Berechtigung erhält. Dies ist hilfreich, wenn ein Benutzer Mitglied einer Rolle oder Gruppe ist, der eine Berechtigung erteilt wurde, und Sie möchten verhindern, dass dieser einzelne Benutzer die Berechtigung erbt, indem Sie eine Ausnahme erstellen. Die Syntax für diesen Befehl lautet wie folgt:

```
DENY [permission]
ON [object]
TO [user]
```

Abbildung 4.4: DENY Syntax

Die Parameter für den Befehl DENY sind identisch mit denen für den Befehl GRANT.

4.7 Rollen

Rollen können als Bündel oder Paket von Berechtigungen definiert werden. Wenn dem Benutzer automatisch eine Rolle zugewiesen wird, werden alle untergeordneten Berechtigungen auch dem Benutzer zugewiesen. Ähnliches gilt für den Widerruf. Der Hauptvorteil des Erstellens von Rollen besteht darin, dass in einer Mehrbenutzerumgebung, in der mehrere Berechtigungen auf einmal erteilt oder widerrufen werden können, das Erteilen und Widerrufen von Rollen vereinfacht wird. Andernfalls würde dies viel Zeit in Anspruch nehmen, wenn dies separat erfolgt. Es ist auch möglich, einer Rolle eine weitere Rolle hinzuzufügen.

4.7.1 Systemrollen

- *Connect*: Create table, Create view, Create synonym, Create sequence, Create session etc.
- *Resource*: Create Procedure, Create Sequence, Create Table, Create Trigger etc. Die Hauptverwendung der Rolle Ressource"besteht darin, den Zugriff auf Datenbankobjekte zu beschränken
- *DBA*: Alle System Berechtigungen.

4.7.2 Rolle anlegen und zuordnen

Zunächst muss der (Datenbankadministrator) DBA die Rolle erstellen. Dann kann der DBA der Rolle Berechtigungen und der Rolle Benutzer zuweisen.

Syntax

```
100 CREATE ROLE manager;
```

- Nachdem die Rolle erstellt wurde, kann der DBA die GRANT - Anweisung verwenden, um der Rolle Berechtigungen zuzuweisen und der Rolle danach Benutzer zuzuweisen.
- Es ist einfacher, Berechtigungen für die Benutzer über eine Rolle zu erteilen oder zu widerrufen, als jedem Benutzer direkt eine Berechtigung zuzuweisen.

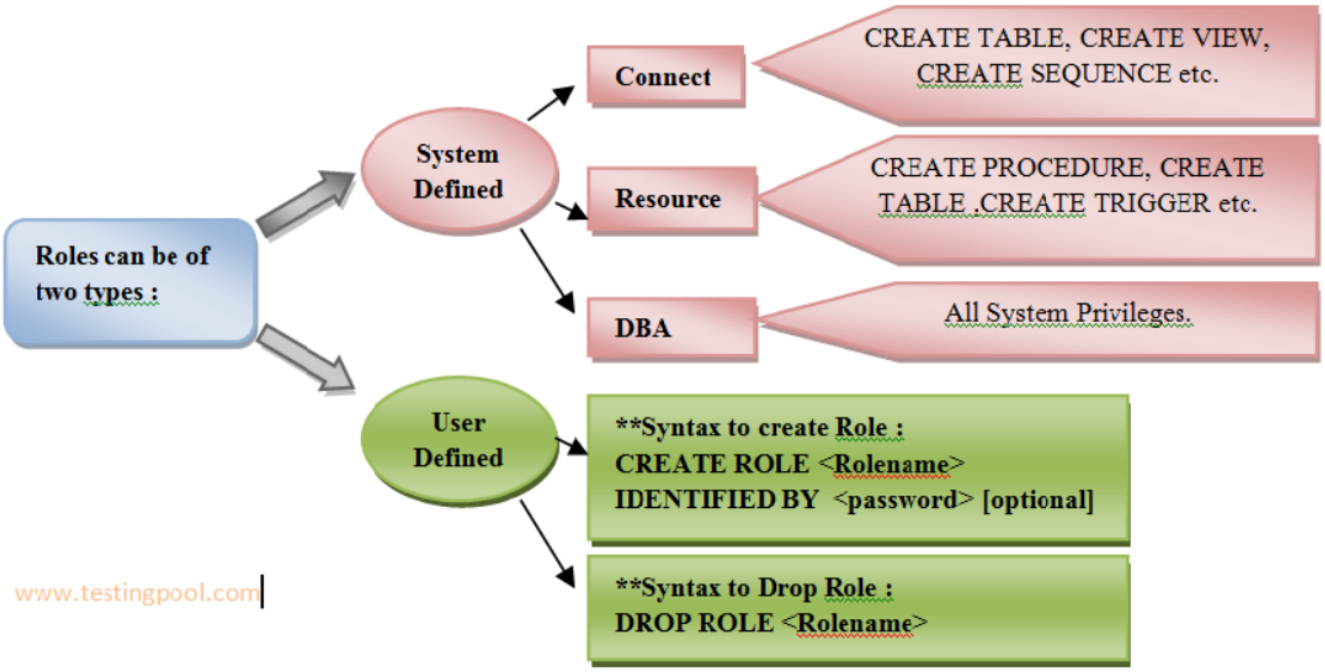


Abbildung 4.5: Rollen

Gewähren Sie einer Rolle Berechtigungen

```
100 GRANT create table, create view
101 TO manager;
```

Gewähren Sie Benutzern eine Rolle

```
100 GRANT manager to SAM, STARK;
```

Berechtigung von Rolle entziehen

```
100 REVOKE create table from manager;
```

Rolle löschen

```
100 DROP ROLE manager;
```

Zunächst wird eine Manager-Rolle erstellt, und anschließend können Manager Tabellen und Ansichten erstellen. Es gewährt dann Sam und Stark die Rolle von Managern. Jetzt können Sam und Stark Tabellen und Ansichten erstellen. Wenn Benutzern mehrere Rollen zugewiesen wurden, erhalten sie alle Berechtigungen, die mit allen Rollen verknüpft sind. Anschließend wird die Berechtigung zum Erstellen einer Tabelle mit Revoke aus der Rolle Managerëntfernt. Die Rolle wird mit drop aus der Datenbank entfernt.

4.8 User erstellen in Oracle

Um einen User zu erstellen muss mindestens folgendes Statement ausgeführt werden.

4.8.1 Syntax

```
100 CREATE USER max_mustermann
101 IDENTIFIED BY 'passme';
```

Sie müssen über das Systemprivileg CREATE USER verfügen. Wenn Sie einen Benutzer mit der Anweisung CREATE USER erstellen, ist die Berechtigungsdomäne des Benutzers leer.

4.8.2 Optionale Parameter

Default Tablespace clause

Geben Sie den default Tablespace für Objekte , die der Benutzer erstellt, immer an, wenn es möglich ist. Wenn Sie diese Klausel weglassen, werden die Objekte des Benutzers im default Tablespace der Datenbank gespeichert.

Wenn für die Datenbank kein default Tablespace angegeben wurde, werden die Objekte des Benutzers im SYSTEM-Tabellenbereich gespeichert, was nicht gut ist.

Temporary Tablespace clause

Geben Sie den Tablespace oder die Tablespace - Gruppe für die temporären Segmente des Benutzers an. Wenn Sie diese Klausel weglassen, werden die temporären Segmente des Benutzers im temporären Standardtabellenbereich der Datenbank oder, falls keiner angegeben wurde, im SYSTEM-Tabellenbereich gespeichert.

Temporäre Tablespaces werden für spezielle Vorgänge verwendet, insbesondere zum Sortieren von Datenergebnissen auf der Festplatte und für Hash-Joins in SQL. Bei SQL mit Millionen von zurückgegebenen Zeilen ist der Sortiervorgang für den RAM-Bereich zu groß und muss auf der Festplatte erfolgen. In dem temporären Tablespace findet dies statt.

Quota clause

Verwenden Sie die QUOTA-Klausel, um den maximalen Speicherplatz anzugeben, den der Benutzer im Tablespace bzw. default Tablespace zuweisen kann. Mit UNLIMITED kann der Benutzer Speicherplatz im Tablespace ohne Einschränkung zuweisen.

4.8.3 Erstellen eines Benutzer mit optionalen Parametern

```
100 CREATE USER max_mustermann
101 IDENTIFIED BY 'passme'
102 DEFAULT TABLESPACE example
103 QUOTA 10M ON example
104 TEMPORARY TABLESPACE temp
105 QUOTA 5M on temp;
```

4.8.4 Einem Benutzer erlauben, eine Sitzung zu erstellen

Wenn wir einen Benutzer in SQL erstellen, ist dieser nicht einmal erlaubt, sich anzumelden um eine Session zu erstellen, bis dem Benutzer die richtigen Berechtigungen / Privilegien erteilt wurden. Der folgende Befehl kann verwendet werden, um eine Session zu erstellen.

```
100 GRANT CREATE SESSION TO username;
```

Einem Benutzer erlauben, eine Tabelle zu erstellen

Damit ein Benutzer Tabellen in der Datenbank erstellen kann, können Sie den folgenden Befehl verwenden:

```
100 GRANT CREATE TABLE TO username;
```

Dem Benutzer Platz auf dem Tablespace geben, um die Tabelle zu speichern

Wenn beim erstellen des Benutzers kein Tablespace angeben wurde, reicht es nicht aus, einem Benutzer das Erstellen einer Tabelle zu erlauben, um Daten in dieser Tabelle zu speichern. Wir müssen dem Benutzer auch das Recht gewähren, den verfügbaren Tablespace für seine Tabelle und Daten zu verwenden. Der obige Befehl ändert die Benutzerdetails und ermöglicht den Zugriff auf unbegrenzten Tablespace auf dem System. (Generally unlimited quota is provided to Admin users)

```
100 ALTER USER username QUOTA UNLIMITED ON SYSTEM;
```

Gewähren Sie einem Benutzer alle Berechtigungen

sysdba ist eine Reihe von Berechtigungen, die alle Berechtigungen enthalten. Wenn wir also einem Benutzer alle Berechtigungen gewähren möchten, können wir ihm einfach die Berechtigung sysdba erteilen.

```
100 GRANT sysdba TO username;
```

Es gibt noch unzählige System Privilegien, die einem User zugeteilt werden können z.B.:

- CREATE PROCEDURE
- CREATE SEQUENCE
- CREATE VIEW
- DROP ANY TABLE
- DROP ANY INDEX
- SELECT ANY TABLE
- DELETE FROM ANY TABLE

Nochmals zur Erinnerung, System Privilegien können nur vom Datenbankadministrator oder einem Benutzer mit der Admin Berechtigung erteilt werden.

Der User kann Objekt Privilegien für seine eigens erstellten Tabellen, Prozeduren usw. an andere User vergeben. Natürlich kann dies der Admin auch. Beispiele dafür wären:

- DELETE
- INSERT
- SELECT
- UPDATE

Berechtigungen zurücknehmen

Wenn Sie die Berechtigungen von einem beliebigen Benutzer zurücknehmen möchten, verwenden Sie den Befehl REVOKE.

```
100     REVOKE CREATE TABLE FROM username;
```

Kapitel 5

SQL - DML

Kapitel 6

PL/SQL

Kapitel 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
```

Abbildung 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	...	...	...	...	...

Abbildung 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	...	...	...	...	...

Abbildung 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	...	...	...	...	...

Abbildung 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

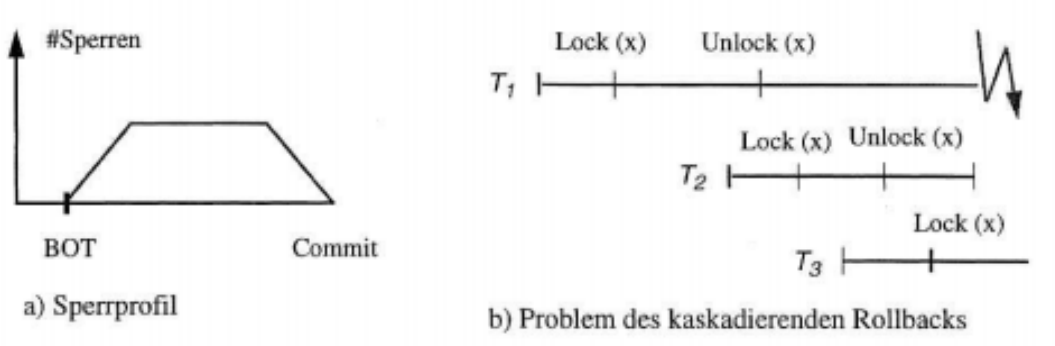
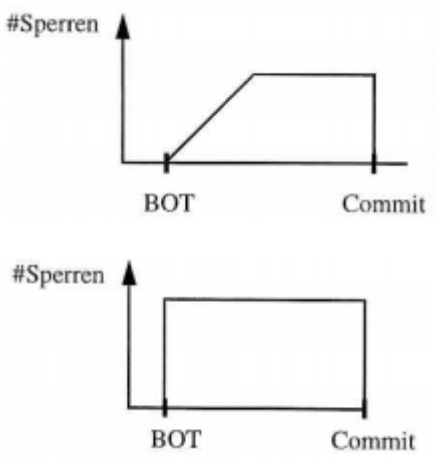


Abbildung 7.5: Inconsistent Analysis

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.

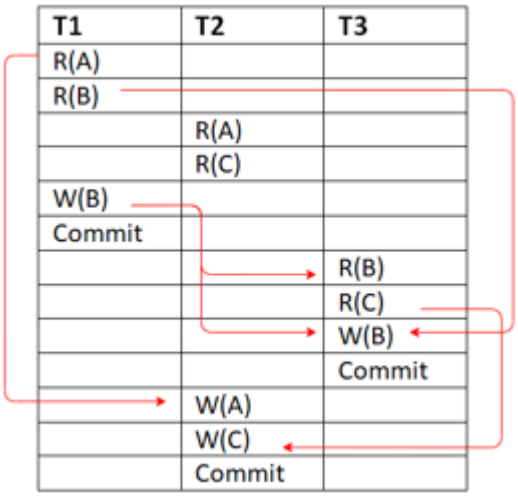
7.5.1 Precedence Graph

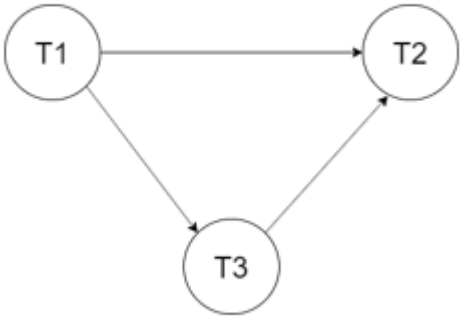
Eine Menge von Datenbank-Transaktionen lässt sich genau dann serialisieren, wenn der zugehörige Serialisierbarkeitsgraph zyklenfrei ist.

Relegn für das Zeichnen

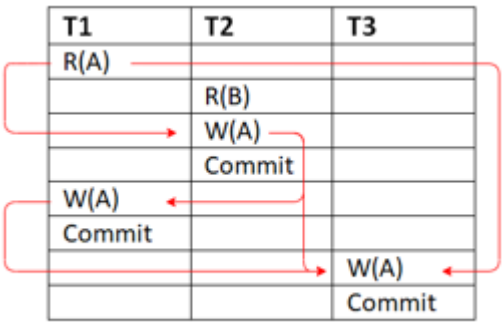
- Transaktionen werden durch Knoten dargestellt
- Pfeile zwischen Knoten für:
  - R1(A) $\longrightarrow$ W2(A)
  - W1(A) $\longrightarrow$ W2(A)
  - W1(A) $\longrightarrow$ R2(A)

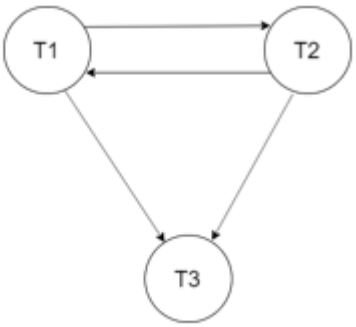
Beispiel 1 - serialisierbar





Beispiel 2 - nicht serialisierbar





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:

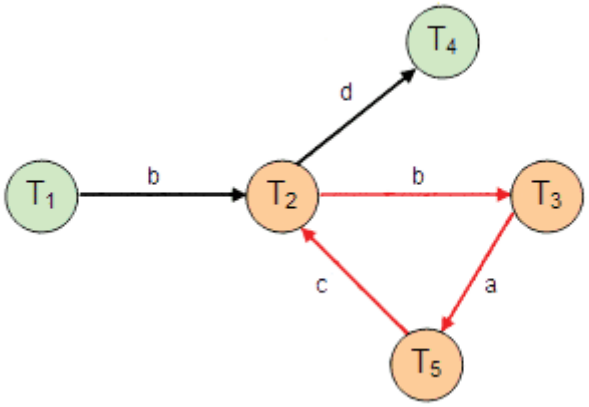
S[r1(b),r2(b),w2(b),r2(c),r2(d),r3(b),r4(d),w3(a),w4(d),r5(a),r5(c),w2(c)]

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.

Kapitel 8

Indizes

8.1 Allgemeines

8.1.1 Abbildung von Relationen im Hauptspeicher

Eine Relation wird in einer Datei abgebildet, dabei werden zusätzlich zu den Tupeln **Datensatztabellen**, die Verweise zu den Tupeln verwalten, gespeichert. Um die Tupel zu referenzieren verwendet man einen *Tupel-Identifikator*, **TID**, auch **ROWID**.

8.1.2 Indexstrukturen

Indizes sind Teil des physischen Entwurfs, sie dienen dazu Speicherzugriffe zu beschleunigen.

Meist werden bei Datenbankabfragen nur einige, nicht alle, Tupel einer Relation benötigt, man muss die Datensätze also suchen.

Sind die Daten ohne Zusatzinformation gespeichert, so muss man ganze Dateien durchsuchen bis man die Tupel, die die Suchkritierien erfüllen, findet.

Suchkriterien der Indexe werden Schlüssel genannt.

Indexstrukturen geben die passenden Datensätze zu einem Suchkriterium an.

8.1.3 Nachteile von Indizes

- Extra Wartungsaufwand
- erhöhter Speicherbedarf
- `INSERT`, `UPDATE`, `DELETE` Operationen können erheblich langsamer werden

8.1.4 Verwendung

- Primary Keys und Unique Keys sind automatisch indiziert
- Foreign-Key Columns sollten indiziert werden (Join-Columns)
- Columns die in `WHERE` / `GROUP BY` / `ORDER BY` häufig gebraucht werden sollten indiziert werden
- Beschleunigung häufig erst ab mehreren 100 Datensätzen
- Traditioneller Index mit B*-Baum nicht gut für Spalten mit wenig verschiedenen Werten (schlecht: Geschlecht, gut: Telefonnummer)
- Spalte sollte indiziert werden, wenn die Queries weniger als 5% der Datensätze zurückliefern
- Tabellen, auf die gleich oft eine Abfrage und `INSERT`, `UPDATE`, `DELETE` abgesetzt wird, sollte nicht indiziert werden

8.2 Unterscheidungen

8.2.1 Primärindex

Ein Primärindex legt die physische Anordnung der Daten fest. Es kann daher nur einen Primärindex geben. Meist wird der Primärschlüssel vom Primärindex indiziert.

8.2.2 Sekundärindex

Die Indexstruktur des Sekundärindex verweist mittels der TID / ROWID auf die physisch gespeicherten Datensätze. Ein Sekundärindex kann ohne Einfluss auf die physischen Daten erstellt und gelöscht werden.

8.2.3 Concatenated Index

Ein Index wird für 2 oder mehrere Spalten angelegt.

Sollte verwendet werden, wenn die indizierten Spalten in der (beim Statement angegebenen) Reihenfolge häufig mit AND und OR in WHERE Klausel vorkommen. Reihenfolge beim Index anlegen in diesem Fall äußerst wichtig, ansonsten wird der Index nicht verwendet!

- Unique Index: Jede Wertkombination in den indizierten Spalten kommt max. 1x vor
- Non-unique Index: Wertkombinationen können öfter vorkommen

8.2.4 Ascending / Descending Index

Einzelne Indexspalten können auf- und absteigend sortiert angelegt werden. Besonders nützlich falls sortierte Ausgabe erforderlich ist kein zusätzliches Sortieren mehr notwendig.

8.2.5 Arten von Indizes in Oracle

- B-Tree Index
- Bitmap Index
- Bitmap-Join Index
- Funktionsbasierter Index

8.2.6 B-Tree Index

Der B-Tree Index kann weiter unterteilt werden in folgende Arten:

Reverse-Key-Index

Der Schlüssel wird reversed gespeichert.

Index-Organized Table

Oracle-Way of saying dass Index ein Primärindex ist.

Descending Index

Index wird absteigend gehalten.

B-Tree-Cluster Index

Index zeigt nicht auf TID sondern auf Cluster der Datensätze enthält.

8.2.7 Bitmap Index

Mit einer Hashfunktion wird ein Schlüssel auf einen Behälter (Bucket) abgebildet, der den zugehörigen Datensatz (oder TID) aufnehmen soll. Die Bitmap wird aus dem Schlüssel errechnet. Ein Bucket hat platz für eine bestimmte Anzahl von Datensätzen, es können also mehrere Schlüssel dem gleichen Bucket zugewiesen werden.

Die Verwendung eines Bitmap Indexes macht Sinn bei wenig verschiedenen Werten (Geschlecht, Noten).

8.2.8 Bitmap-Join Index

Ist ein Bitmap Index um mehrere Tabellen zu joinen.

8.2.9 Funktionsbasierter Index

Ist auf B-Tree und Bitmap Index anwendbar. Der Funktionsbasierte Index enthält Columns die durch Funktionen, wie z.B. `UPPER` modifiziert gesucht werden.

8.3 Zugriffsarten

8.3.1 Index Scan

Zugriff über indizierte Tabellenspalte (Anzahl I/O Operationen = Höhe des B-Baums) Im Blattknoten ROWID

8.3.2 Full Index Scan

gesamter Index wird verarbeitet – verfügbar, wenn in WHERE auf Indexspalte referenziert wird. Erfordert keine Sortierung, Daten werden über Index ausgegeben. Key-Value Paare des Indexes sind aber sortiert.

8.3.3 Fast Full Index Scan

Zugriff auf Indexdaten, nicht Tabellendaten. Möglich, wenn alle Query-Spalten im Index enthalten sind.

8.3.4 Index Range Scan

nur die Blätter des B-Baums betroffen, werden vor- oder rückwärts gelesen. (z.B. alle Abteilungen mit einer Nr. zwischen 20 und 40)

8.3.5 Index Unique Scan

Wenn in WHERE ein Gleichheitsoperator auf Spalten referenziert, wo alle unique- indiziert sind.

8.3.6 Index Skip Scan

Bei Concatenated Indices, wenn die führende (= 1.) Spalte wenig verschiedene Werte hat und die nicht führenden (!= 1) Spalten viele verschiedene Werte. Verwendet wird Index Skip Scan dann, wenn führende Spalte nicht in WHERE enthalten.

8.3.7 Wann ein Index wirklich verwendet wird

Die endgültige Entscheidung, ob der Index verwendet wird, liegt beim Optimizer. Es kann aber im Query angegeben werden, dass ein Index nicht verwendet werden soll bzw. eine andere Zugriffsart definieren.

Verwendung des Index überprüfen

```
100 SELECT table_name, index_name, monitoring, used FROM v$object_usage
```

`Used` sagt hierbei mit YES/NO aus, ob der Index verwendet wurde.

8.4 Indexstufen

8.4.1 Einstufiger Index

In EINER Indexdate besteht das Key-Value-Pair aus dem Primary Key und dem Index.

8.4.2 Mehrstufiger Index

Mehr als eine Indexdatei. Die Indexdatei der ersten Stufe ist gleich dem einstufigem Index. Bei allen anderen Dateien der Stufen bestehen die Key-Value-Pairs aus dem Primary Key und dem Index der Indexdatei eine Stufe niedriger.

8.5 SQL Syntax

```
100 CREATE [ UNIQUE ] INDEX indexname ON tablename (columns,...)
```

8.5.1 Clustered Index

```
100 CREATE [ UNIQUE ] CLUSTERED INDEX index_name ON table_name;
```

8.5.2 Drop Index

```
100 DROP INDEX index_name;
```

Kapitel 9

Portieren von Projekten

Kapitel 10

Datenbank Tuning

Kapitel 11

Data Warehousing

Kapitel 12

Struktur und Aufbau einer Datenbank anhand von Oracle

Kapitel 13

Datenintegrität

Kapitel 14

Trigger

Kapitel 15

JPA

Kapitel 16

Apex

Kapitel 17

Verteilte Datenbanken

17.1 Grundlagen

Motivation: Geographisch verteilte Verwaltung von Firmen.

17.1.1 Beispiel

Bank mit mehreren Filialen:

- jede Filiale hat eine lokale Kundendatenbank
- andere Filialen oder die Zentrale sollten Zugriff auf die lokale DB haben

17.2 Terminologie

Verteilte Datenbanken bestehen aus einer Sammlung von Informationseinheiten. Diese auf mehreren Rechnern verteilt. Die Rechner sind über ein Kommunikationsnetz verbunden.

17.3 Entwurf

17.3.1 Globales Schema

Das Globale Schema ist der Ausgangspunkt des Entwurfs für ein verteiltes System. Es ist ein **relationales Schema** eines zentralen Entwurfs.

17.3.2 Die Aufgaben eines Verteilten DBMS

- Die Daten fragmentieren = ein Fragmentierungsschema erstellen
- Ein Zuordnungsschema für die Fragmente erstellen

17.4 Fragmentierungsschema

Das zentrale Schema wird in Fragmente zerlegt. Im zentralen Schema gibt es zusammengehörende Information, (= Relationen, Tabellen), und diese werden in Untereinheiten (= **Fragmente**) aufgeteilt, wobei jedes Fragment Informationen enthält, welche in keinem anderen Fragment enthalten sind. Man spricht von **Disjunktheit**.

Die Zerlegung in Fragmente erfolgt anhand vom Zugriffsverhalten. Daten die oft gemeinsam abgefragt werden, werden in einem Fragment zusammengefasst.

17.5 Zuordnungsschema

Sobald das Fragmentierungsschema fertig ist, werden die Fragmente den Stationen des VDBMS zugeteilt. Man spricht von der **Allokation.**

17.5.1 Arten der Zuordnung

Man kann Fragmente auch mehreren Stationen zuweisen, dann sind sie mehrfach, redundant gespeichert.

- Redundanzfrei
- Redundant = Allokation mit Replikation

17.6 Fragmentierungs-Arten

Es gibt 3 Arten wie man die Daten Fragmentieren kann. Vorerst muss geklärt werden welche Anforderungen eine Fragmentierung hat, damit die Daten wieder vollständig hergestellt werden können.

17.6.1 Korrektheitsanforderungen an die Fragmentierung

1. Rekonstruierbarkeit: Die ursprünglichen Daten lassen sich vollständig wiederherstellen
2. Vollständigkeit: Jeder Datensatz (= Datum) wird einem Fragment zugeordnet
3. Disjunktheit: Die Fragmente überlappen sich nicht. Ein Datum ist nur an einer Station gespeichert

17.6.2 Horizontale Fragmentierung

Eine Tabelle wird nach Zeilen aufgeteilt (= eine Relation wird nach Tupel aufgeteilt). Um die Zuordnung zu bestimmen, braucht man eine Bedingung, welche für die Daten entweder gilt oder nicht.

17.6.3 Vertikale Fragmentierung

Eine Tabelle wird nach Spalten aufgeteilt (= Eine Relation wird nach Attributen aufgeteilt). Die Attribute weisen ähnliche Zugriffsmuster auf. Es ist bei beliebiger Fragmentierung allerdings die Rekonstruierbarkeit nicht gewährleistet. Man nehme als Beispiel die Tabelle Person mit den Attributen SVN (= Schlüssel), Name und Alter. Teilt man diese Tabelle vertikal auf in ein Fragment SVN, ein Fragment Name, und ein Fragment Alter, so weiss man nicht mehr welcher Person eine Zeile im Fragment Name oder Alter gehört.

Rekonstruierbarkeit gewährleisten

1. Jedem Fragment den Primärschlüssel zuweisen
2. Jedem Fragment ein Surrogat zuweisen

Bei der Zuweisung des PK ist allerdings die Disjunktheit verletzt, denn der PK ist mehrmals gespeichert.

17.6.4 Kombinierte Fragmentierung

Man kann die Fragmentierungen auch kombinieren.

- Zuerst horizontal dann vertikal
- Zuerst vertikal dann horizontal

17.7 Transparenz

Die Transparenz ist der Grad von Unabhängigkeit den das DBMS dem Benutzer beim Zugriff auf die verteilten Daten vermittelt. Idealerweise muss der Nutzer nicht wissen, dass die Daten verteilt gespeichert werden.

17.7.1 Fragmentierungstransparenz

Der Nutzer merkt nicht, dass die Daten verteilt ist. Seine Interaktionen sind wie bei einem zentralen Schema.

17.7.2 Allokationstransparenz

Die Nutzer müssen nur wissen, wie die Daten fragmentiert sind, allerdings nicht wo die Daten gespeichert sind.

17.7.3 Lokale-Schema-Transparenz

Die Nutzer brauchen Kenntnis über die Fragmentierung und den Aufenthaltsort der Daten.

17.8 Schwierigkeiten im VDBMS

17.8.1 Anfrageübersetzung und Optimierung

Die Aufgabe des Anfrageübersetzers ist es, einen Anfrageauswertungsplan auf den Fragmenten zu generieren, und das möglichst effizient.

17.8.2 Transaktionskontrolle

Transaktionen können sich über mehrere Stationen verteilen.

EOT

So wie lokale Transaktion müssen auch globale Transaktionen atomar behandelt werden. Entweder alle Stationen committen, oder keine.

Dadurch ist das EOT eine besondere Schwierigkeit.

17.8.3 Mehrbenutzerfähigkeit

Es reicht nicht das Transaktionen lokal serialisierbar sind. Es können Trotzdem global Deadlocks auftreten.

Deadlocks

Die Deadlockerkennung mit dezentralisierter Sperrverwaltung gestaltet sich als schwierig.

17.9 SQL Database Link

```
100 CREATE DATABASE LINK link_name
101 CONNECT TO username IDENTIFIED BY 'password'
102 USING '(DESCRIPTION =
103                     (ADDRESS = (PROTOCOL = TCP)(HOST = ip-Adresse)(PORT = 1522))
104                     (CONNECT_DATA = (SERVER = DEDICATED)(SERVICE_NAME = z.b.:xe))
105         )';
```

Kapitel 18

OracleNet

Kapitel 19

XML

Kapitel 20

Objektorientierte und Objektrelationale Datenbanken

Kapitel 21

Backup und Recovery

Es können viele Dinge passieren, die die Daten in einer Datenbank gefährden - Ausfälle, Fehlbenutzungen, etc. Gegenüber diesen Vorkommnissen muss man sich absichern.

21.1 Arten von Backups

21.1.1 Physisches Backup

Ein Physisches Backup kann hot/cold und Full/Incremental sein. Dabei werden ganze Datenbankfiles auf Betriebssystemebene kopiert. Physische Backups und Recoveries können mit dem RMAN-Tool durchgeführt werden. RMAN unterstützt Inkrementelle Backups, speichert also nur geänderte Datenblöcke und hat einen speziellen Kompressionsalgorithmus, welcher für Oracle-DBs entwickelt wurde. Man kann die Backups mit ihm verschlüsseln. RMAN hat viele Optimierungen, wie z.B dass beim Recovery nicht betroffene Blöcke übersprungen werden können.

21.1.2 Physisches Hot Backup

Während des Backups können Benutzer das System verwenden. Die Changes die während des Backups passieren werden separat mitgelogged, und dann auf das Backup vorwärtsgerollt. (Rolling Forward = REDO-Log-Files anwenden - im vergleich zu Rolling Back = REDO-Logs rückwärts abspielen). Hierbei ist die Gefahr, dass Backup und DB nicht zur Gänze synchronisiert sind.

21.1.3 Physisches Cold backup

Kopieren passiert während die DB gerade in Downtime - also nicht für User zugänglich, bzw. heruntergefahren - ist.

21.1.4 Physisches Inkrementelles Backup

Hierbei werden nur Änderungen (REDO-Log-Files) kopiert, die seit dem letzten Backup passiert sind. Ein Vorteil ist, dass ein Inkrementelles Backup Hot gemacht werden kann, ohne eine große Auswirkung auf die Nutzer zu haben (DB wird nur etwas langsamer).

21.1.5 Physisches Full Backup

Hierbei wird die ganze Datenbank kopiert. Dies sollte Cold gemacht werden, und die Datenbank muss im "ARCHIVELOG" modus sein. Ein Richtwert hierfür ist alle zwei Wochen. Hierbei wird kopiert:

- Control Files
- REDO-Log (Auch genannt: Transaction-Files)
- Archive Files
- Data Files

21.1.6 Logisches Backup

Ein Logisches Backup kopiert keine Files, sondern nur die Daten. Wird meistens verwendet wenn man eine Datenbank verschieben oder archivieren will. Bei einem logischen Backup werden SQL-Statements generiert, die gebraucht werden würden, um die Datenbank (beziehungsweise die Objekte die man backup-en will) wiederherzustellen.

21.2 Recovery - Auslöser und Fehler

21.2.1 User Error

Der Benutzer macht einen Bedienfehler, welcher für die Datenbank wie ein ganz normaler Befehl aussieht und deswegen protokollgerecht abgehandelt wird. (Das DBMS weiß nicht, dass der Benutzer eigentlich nicht gerade * FROM CUSTOMERS droppen wollte. Für es sieht es aus als wäre es ein ganz normaler und gewollter Befehl)

21.2.2 Media Failure

Fehler des Speichermediums. Kann passieren, wenn die Festplatte kaputt wird (hierfür würde sich z.B ein RAID-System empfehlen, um diese Fehler zu minimieren).

21.3 Datenbankstrukturen für Recovery

21.3.1 Datafiles und Data Blocks

Ein Datafile hat anfangs bereits eine fixe Größe, enthält aber noch keine Daten, wenn noch keine vorhanden sind. Der Speicherplatz ist sozusagen "reserviert". Innerhalb der Datafiles sind Segmente, mit Extends welche aus Datablocks (aka. Logical Blocks, Oracle Blocks, Pages) bestehen. Diese Datenfiles bilden die Grundlage für Full-Backups und den Ausgangspunkt für das Recovery über REDO-Logs.

21.3.2 Online REDO-Logs

Bevor eine Änderung in einem Datafile gemacht wird, wird diese Änderung im REDO-Log protokolliert. Ein REDO-Log besteht aus Log-Gruppen und Log-Member welche Files darstellen. Diese können manuell erstellt und verwaltet werden.

21.3.3 Archived REDO-Logs

Genau wie Daten-Files können REDO-Logs offline genommen (archiviert) werden. Nur im Falle der REDO-Logs ist dies wichtiger als bei Datenfiles, da nur archivierte REDO-Logs für Recoveries genutzt werden können. Archivierte REDO-Logs werden in speziellen Archived-Locations gespeichert.

21.3.4 UNDO-Segmente

UNDO-Segmente speichern Daten, während einer Transaktion vor dem Commit. Diese werden automatisch gehandled und müssen nicht beim Planen der Backup-Strategie berücksichtigt werden.

21.4 Recovery

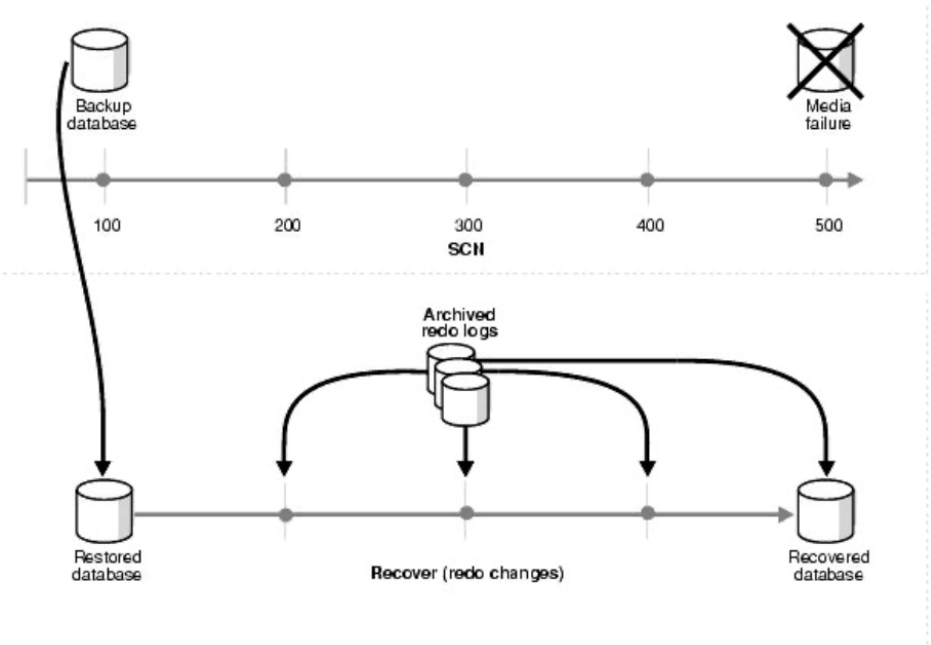


Abbildung 21.1: Für ein Recovery nimmt man einen Full-DB-Backup stand, und spielt auf diesem alle REDO-LOGS wieder ab, die seit diesem archiviert wurden.

21.4.1 Media-Recovery

Media Recovery hilft gegen Media-Failure oder User-Errors. Dafür braucht man die gebackupten Control-Files, Data-Files, Archivierte Redo-Logs und die Redo-Logs der aktuell kaputten Datenbank. Dabei werden alle Redo-Logs, die nicht in der aktuell kaputten Datenbank abgebildet sind abgespielt um den kompletten Stand der Datenbank wiederherzustellen. Media-Recoveries müssen manuell durchgeführt werden.

21.4.2 Complete Recovery

Die gesamten Daten vor dem Error werden wiederhergestellt. Es wird ein komplettes Backup genommen, und die seitdem aufgenommenen Inkrementellen-Backups werden darauf angewandt.

21.4.3 Point-In-Time Recovery (Incomplete Recovery)

Die Datenbank wird auf den Stand eines speziellen Punktes in der Vergangenheit gesetzt. Man kann auch einen einzelnen Tablespace Point-In-Time-Recoveren.

21.4.4 Instance- und Crash-Recovery

Diese Recovery-Arten werden von Oracle automatisch durchgeführt. Hierbei werden die - möglicherweise korrupten - Datenfiles wieder auf den Transaktionskonsistenten Stand, welcher vor dem Absturz vorhanden war gebracht.

Instance-Recovery

Dies ist der Fall, wenn ein Oracle-Cluster vorliegt. Bei diesem kann eine einzelne kaputte Instanz von den anderen - die noch funktionieren - recovered werden (Ähnlich wie bei einem Raid-FS)

Crash-Recovery

Dies ist der Fall, wenn alle Instanzen eines Clusters ausfallen.

Kapitel 22

Datenbank-Entwurfstools

Kapitel 23

Data Analytics

Kapitel 24

Docker

Kapitel 25

Bäume

Kapitel 26

NoSQL