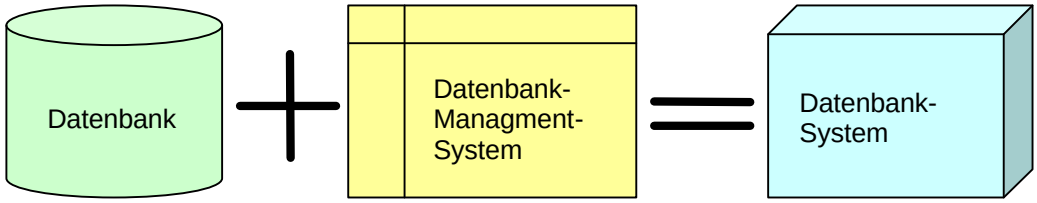
DBMS - Grundkonzept

1. Sprachkonzept zur Kommunikation und Abstraktion
2. Übersetzer, welcher Übergang zwischen Benutzerebene und interner Ebene darstellt (Integritätskontrollen, Sicherungsmechanismen etc.)
3. Mehrbenutzersystem mit Zugriffsschutz



Datenunabhängigkeit

als zentrales Ziel von DBMS bedeutet, eine (weitgehende) Unabhängigkeit von Daten mit ihren arbeitenden Programmen und Nutzern zu erreichen. Dies bedeutet, daß die physische Organisation der Daten für Dialoge und Programme transparent sein soll. Das heißt ein Programm muß sich nicht um die Datenstruktur und den Zugriffspfad kümmern. So ist eine Datenbank anpassungsfähig und flexibel. Werden neue Programme oder Views integriert, haben diese keinen Einfluß auf bereits bestehende Anwendungen und Schichten der DB. Die Fähigkeit zur eigenständigen Aufbereitung von Daten bezeichnet man auch als Datenunabhängigkeit.

Physische Datenunabhängigkeit

Bezieht sich darauf, dass Anwenderprogramme sich nicht um die interne Speicherung und Verwaltung der Daten kümmern müssen. So sollten Datenbanktuning oder Erweiterung von Speicherstrukturen keine Auswirkungen auf Anwendungsprogramme haben.

Logische Datenunabhängigkeit

–
besagt, dass ein Datenbestand und seine Zusammenhänge für verschiedene Anwendungen aus unterschiedlichen Perspektiven interpretiert werden können.

Effiziente Organisation

Anfragen an das DBMS müssen in akzeptabler Zeit beantwortbar sein. Aus diesem Grund müssen geeignete Organisationsformen zur Verfügung gestellt werden, nach denen die Daten effizient von externen Speichern gelesen und geschrieben werden können.

Datensicherheit

Im Falle eines Betriebsmittelsausfalles kann der interne Zustand der Datenbank in einen nicht definierten Zustand geraten. Das DBMS muss deshalb Funktionen bereitstellen, welche dies überwachen und gegebenenfalls reparieren – das so genannte Recovery.

Datenschutz

Um sensible Daten zu schützen sind gewisse Vorkehrungen notwendig. Es muss möglich sein verschiedenen Nutzern auch verschiedene Rechte zuzuweisen zu können. Insbesondere Datensicherheit und Datenintegrität sind Gegenstand der Sparc Drei-Sichten-Architektur.

Datenmodell

Ist das zentrale Hilfsmittel zur Herstellung einer Abstraktion eines gegebenen Realweltausschnitts und den Einzelheiten der physischen Speicherung. Es umfaßt eine Menge von Konzepten mit denen sich die Struktur einer DB beschreiben läßt.
Struktur: Datentypen, Beziehungen, Bedingungen, welche von Daten erfüllt werden sollen (DDL)
Operationen: zur Beschreibung von Anfragen und Updates an eine Datenbank (DML)

Die drei Datenmodelle

Hierarchisches Datenbanksystem

Ein Record ist hier ein Knoten einer Baumstruktur, welche die Beziehungen zwischen dieses Records darstellt. Ein Knoten kann mehrere Nachfolger. Während einem Knoten mehrere Söhne zugeordnet werden können, kann er aber höchstens einen Vorgänger haben. Falls aus logischer Sicht zwei verschiedene Knoten den gleichen Record als Nachfolger haben, muss dieser mehrfach, also redundant, aufgeführt werden.

Technisch werden die Beziehungen zwischen den Records über Pointer realisiert. Der Zugriff erfolgt über eine geeignete Traversierung der Struktur.

Netzwerksysteme

Netzwerksysteme sind den hierarchischen Systemen ähnlich. Sie müssen sich aber nicht der Baumstruktur beugen. Redundante Knoten sind hier auch nicht notwendig, da Link-Records eingeführt werden, welche die Mehrfachzuweisungen realisieren. Es werden ebenso Pointer zur Umsetzung der Beziehungen verwendet.

Relationale Systeme

Dieses Modell verzichtet im Gegensatz zu den anderen auf Pointer zur Darstellung von Beziehungen. Daten werden ausschließlich über inhaltliche Angaben aufeinander bezogen. Die Entities einer Klasse werden in Form von Tabellen (Relationen) dargestellt, in deren Zeilen die Beschreibungen stehen. Die Struktur der Tabelle wird über die Spalten-Überschriften festgelegt. Eine wichtige Eigenschaft für Attribute ist der elementare Charakter, d.h. es sind keine zusammengesetzten Attribute oder ähnliches erlaubt. Der Zugriff erfolgt über drei Operationen: Selektion (Zeilenauswahl), Projektion (Spaltenausblendung) und Join (Zusammenführen / Durchschnitt).

Probleme bei Filesystemen

1. Hohe Redundanz als Folge von Mehrfachspeicherung
2. Gefahr der Inkonsistenz
3. Inflexibilität gegenüber Veränderungen in der Anwendung wegen fehlender Datenunabhängigkeit
4. daraus resultierende eingeschränkte Produktivität
5. Datenschutz kaum gewährleistet

Ansi / Sparc Schemaebenen

Externe Ebene	Umfasst verschiedene Sichten der Nutzer (z.B. ein SQL Statement) Diese Subschemata werden in einem externen Schema beschrieben, welches nur diesen Ausschnitt der konzeptuellen Schicht enthält und gegebenenfalls Teile verwehrt.
Konzeptuelle Ebene	Repräsentiert die logische Gesamtsicht der Daten und ihrer Beziehungen untereinander. Sie beschreibt also das, was in der Datenbank abgelegt werden soll. Ferner werden auch Integritätsbedingungen und Zugriffsrechte dokumentiert.
Interne Ebene	Verwaltung der Daten als „interne Records“ (Datensätze). Informationen über Art und Aufbau verwendeter DS und spezieller Zugriffsmechanismen. Sie lokalisiert die Daten auf den Sekundärspeichern. Ist extrem eng mit BS verknüpft (z.B. Wandlung logischer Adressen in physikalische)
	Transformationsvorschriften beschreiben, wie man aus einem konzeptuellen Modell ein internes bzw. externes Modell herleitet. Datenmodelle auf der 3-Ebenen Architektur basierend, gewährleisten ein hohes Maß an Datenunabhängigkeit. **Physische Datenunabhängigkeit:** Bei Änderungen des internen Schemas sind nur die Transformationsvorschriften zu ändern. Externe Sichten bleiben unbeeinflusst. **Logische Datenunabhängigkeit:** In Abhängigkeit von der Flexibilität der Transformationen können ggf. unterschiedliche Interpretationen der Daten in den externen Schemata niederschlagen.

Data Dictonary

Metadaten bzw. Beschreibungen der Daten und somit der Schemata werden im Data Dictonary gespeichert.
Datenbank selbst und das interne Logbuch der Datenbank sind zwei weitere zentrale Systemkomponenten.

Schritte des DBMS nach einer (deskriptiven) Anfrage

1. Syntax prüfen (Integrität)
2. Tabellen in Schema vorhanden?
3. Zugriffsrechte prüfen
4. Festellen welche Operationen intern auszuführen sind und wie der Anfrage Operand intern gespeichert ist
5. Erstellung eines effizienten Codestückes zur Antwortberechnung
6. Operanden aus Datenbank lesen
7. Aufbereiten der Operanden (Optimieren)
8. Sicherstellen, daß Operanden nicht während Ausführung durch andere Anfragen geändert werden (Kritischer Abschnitt – Zwei Phasen Protokoll)

Bestandteile des DBMS

Ebene 1 der Benutzersprache	**I/O-Prozessor** nimmt Anfragen entgegen und gibt Ergebnisse und Fehler aus **Parser** syntaktische Analyse der Anfrage bzw. Auftrages **Precompiler** für eingebettete Kommandos **Autorisierungskontrolle** in jedem Falle bevor es zu Ebene 2 geht *Weitergabe des Codes an Ebene 2 (Update – oder den Query-Prozessor) durch die Autorisierungskontrolle* (bei relationalen DB als Baum mit Operanden als Blätter und Operatoren als Knoten)
Ebene 2 der Anfrageverarbeitung	**Update-Prozessor** Änderungen in der Datenbank vornehmen **Integritätsprüfung** semantische Korrektheit der DB = Konsistenz, deren Regeln beim Erstellen des Schemas definiert werden, wie z.B. höhe<>0 **Query-Prozessor** nur Anfragen an Datenbank und somit keine Integritätsprüfung notwendig, dafür aber Optimierung **Optimierer** versucht unnötigerweise komplizierte Benutzeranfragen zu vereinfachen *Weitergabe des Codes an Ebene 3 durch den Optimierer*
Ebene 3 der Zugriffsstrukturen	**Zugriffsplanerstellung** Bestimmung eines Zugriffspfades aus den vorhandenen Zugriffsstrukturen (z.B. Indexe) **Code-Erzeugung** Erzeugung des Zugriffs- und Ausführungscodes, was immer nur eine Folge von *Lese und Schreiboperationen (=Transaktionen)* sind *Weitergabe der Transaktionen an die Transaktionsmanager*
Ebene 4 der Synchronisation paralleler Zugriffe	**Transaktionsverwaltung** Synchronisation parallel ablaufender Transaktionen durch Verzahnung, da Nutzer einer DB nie exklusiven Zugriff hat. Kein sequentieller Ablauf, da sonst kurze Transaktionen gegebenenfalls lange warten müssen, obwohl diese in disjunkten Teilen arbeiten **Schedule** Synchronisation der zeitlich überlappten Transaktionen (Concurrency Control) **Recovery-Manager** Transaktionen werden nach „Alles oder Nichts“ Prinzip bearbeitet. Wird eine Transaktion nicht beendet, stellt der Recovery Manager den Urzustand der DB mit Hilfe des Log-Buches wieder her. (auch nach Hardwarefehlern o.ä.)
Ebene 5 der Speicherverwaltung	**Buffermanager** Hauptspeicherverwaltung für jede Transaktion **Data-Manager** mit Geräte- und Sekundärspeichermanager führt die physischen Zugriffe aus *Nutzung von Betriebssystem Funktionen*
	Alle Ebenen sind mit dem Data-Dictonary verbunden!

Datenbankentwurf

Qualitätsmerkmale

Vollständigkeit in Bezug auf den nachgebildeten Umweltausschnitt

Korrektheit des Datenbankschemas ist gegeben, wenn die Konzepte des betreffenden Datenmodells korrekt verwendet werden und das Schema syntaktisch und semantisch Korrekt ist.

Minimalität eines Datenbankschemas in Bezug auf einmaliges vorkommen verschiedener Anforderungen. Gleichzustellen mit Redundanzvermeidung, welche aber manchmal z.B. durch Normalisierung, sogar erwünscht ist.

Lesbarkeit des Schemas durch z.B. ER-Diagramme (symmetrisch etc) und selbsterklärende Entity-Bezeichnungen.

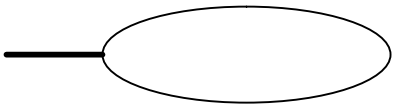
Modifizierbarkeit eines Datenbank Schemas durch gute Modularisierung und Dokumentation des Schemas.

Schritte des Datenbankentwurfs

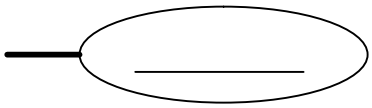
1. **Anforderungsanalyse und –spezifikation**
    a. Informationsanforderung (statische Informationen und Integritätsbedingungen
    b. Bearbeitungsanforderungen (Analyse der Häufigkeit verschiedener Anfragen)
    c. Benutzergruppenanalyse
    d. Einsicht in vohandene Dokumentation
    e. Fragebögen und Besprechungen mit Betroffenen des modellierten Umweltausschnitts
2. **Konzeptueller Entwurf**
    a. Erstellung eines Zielsystemunabhängigen Datenbankschemas
    b. z.B. als Entity-Relationship Modell
    c. Zusammensetzen von Einzelschichten (der jeweiligen Nutzergruppen) zu einem global view
3. **Logischer Entwurf**
    a. Umsetzung eines Konzeptuellen Schemas auf ein gewähltes DBMS
    b. Einhaltung von Transformationsregeln wie z.B. Normalisierung von Relationen
4. **Physischer Entwurf**
    a. Definition des internen Schemas
    b. Wahl geeigneter Speicherstrukturen
    c. Festlegung von Zugriffsmechanismen mit Ziel, Suchpfade zu minimieren
    d. Später Tuning um Parametereinstellungen zu korrigieren
        i. Speicherformat (Records,Baum,Hash)
        ii. Block- Seitenzuweisung auf Platte
        iii. Clustering
        iv. Indexauswahl
        v. Denormalisierung eines Schemas, falls häufig zwei Schemas in Anfragen wieder zusammengesetzt werden müssen, nachdem diese durch Normalisierung zerlegt wurden

Das ER-Modell

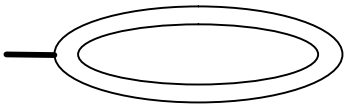
Entity



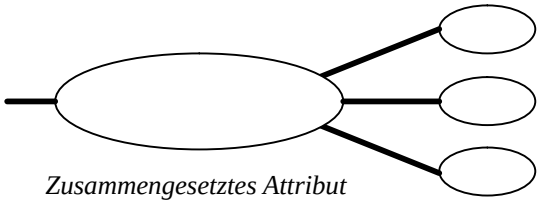
Attribut



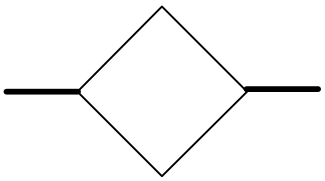
Schlüsselattribut



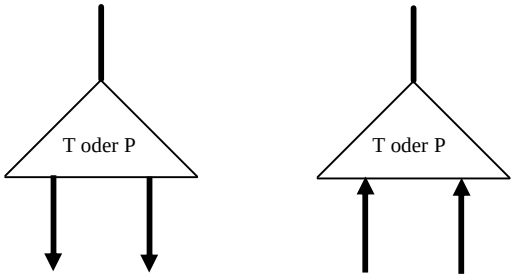
Mehrwertiges Attribut



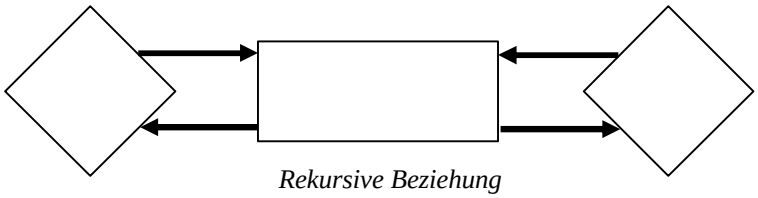
Zusammengesetztes Attribut



Relationship



*IS-A Beziehung **disjunkt** und **nichtdisjunkt** (**T**otal und **P**artiell)*



Rekursive Beziehung

- „Eine Relation entspricht einem Entity-Set“ Vossen Seite 115 unten
- Eine Relation ist partiell, wenn sie partielle Tupel enthält, d.h. nicht jedes Attribut einen Wert haben muß.
- Datenabhängigkeiten sind semantischer Natur und müssen zusätzlich betrachtet werden.
- Entity Integrität ist die Forderung, daß Nullwerte auf Attribute ausgeschlossen werden.
- Intraregionale Abhängigkeiten beziehen sich nur auf eine Attributmenge.
- 1NF bedeutet, daß alle Attribute elementar sind und nicht aus Sets oder Mengen zusammengesetzt sind,

Transformation des ERD ins Relationenmodell

Das „flache“ Relationenmodell erlaubt keine zusammengesetzten Attribute und Relationen in 1NF.

Integrität von relationalen Datenbanken

Gespeicherte Informationen in Datenbanken sind normalerweise bestimmten Randbedingungen unterworfen. Sinn und Zweck ist, die gültigen Zustände aus der Menge aller möglichen Zustände einer Datenbank herauszufiltern. Sind die semantischen Bedingungen der jeweiligen Anwendung und werden zur Entwurfszeit der Datenbank erstellt.

Arten von Integritätsbedingungen

Integritätskontrolle soll semantische Fehler und sinnlose Zustände der Datenbank verhindern. Sie wird in der Regel dem DBMS überlassen, welches über einen speziellen Monitor die Integrität der DB überwacht. Dies hat den Vorteil, dass im Grunde alle Transaktionen problemlos gestartet werden können und selbst ad-hoc Benutzer sich nicht um die Integrität der DB kümmern brauchen.

Statische Bedingungen

Durch Festlegung des jeweiligen Schemas definierte Regeln. Bsp: Jahr beetween 1900 and 2030

1. Schlüsselbedingungen (primary key (idx))
2. referentielle Integrität und Fremdschlüssel
3. Attribut-Bedingungen (not null etc..)
4. Tupel Bedinungen (Check-Klausel)

Transitionale Bedingungen

Sind halbdynamische Bedingungen und schränken die möglichen Übergangszustände ein.

Dynamische Bedingungen

Sind temporale Bedingungen und schränken als Verallgemeinerung der transitionalen Bedingungen die möglichen Zustandsfolgen ein.

Funktionale Abhängigkeiten

Man beachte den logischen Zusammenhang, welcher Attribute und ihre Attributwerte untereinander haben. Zum Beispiel lässt sich durch einen fortlaufenden CD-Index in einer CD-öDatenbank eindeutig eine CD bestimmen, da die dazugehörigen Attribute über diesen Schlüssel eindeutig definiert sind. Man sagt nun auch, dass alle restlichen Attribute vom CD-Index voll funktional abhängig sind. Dabei ist das Auffinden von funktionalen Abhängigkeiten nicht immer trivial.

Voll funktional abhängig heißt, wenn B von keiner echten Teilmenge von A abhängig ist. (Minimalitätsforderung)

Ein Attribut B ist von einem Attribut A funktional abhängig, wenn es zu jedem A genau ein B gibt.

A $\longrightarrow$ B

Wenn zwei Tupel einer Relation gleiche Werte für alle Attribute in A haben und auch ihre B-Werte übereinstimmen, dann ist B von A funktional abhängig. Anders formuliert, bestimmen die A-Werte **eindeutig** die B-Werte.

Für funktionale Abhängigkeiten gelten folgende Eigenschaften:

- *Projektivität*
- *Distributivität*
- *Additivität*
- *Transitivität*
- *Pseudotransitivität*
- *Akkumulation*
- *Erweiterung*

Kontextunabhändigkeit funktionaler Abhängigkeiten heißt, das man eine gegebene Relation durch ergänzende Attribute erweitern kann, ohne die funktionalen Abhängigkeiten zu zerstören.

Gibt es mehrere funktionale Abhängigkeiten können diese in mehrere Relationen geteilt werden und durch ein en verlustfreien Join auch wieder hergestellt werden.

Assertions und Trigger

Assertions werden durch check geprüft. Eine Assertion bildet meist eine Mengendifferenz und prüft, ob ein gültige Teilmenge vorliegt.

Bsp: create assertion GueltigesFachgebiet
Check (not exists (select distinct Fachgebiet from Lektor
Except select distinct Fachgebiet from Buch)

Behanldung kann entweder immediate oder deferred (verzögert) erfolgen.

Integritätsüberwachung:

1. transaktionsorientiert (Überprüfung erfolgt im Kontext eines Programmes)
2. ereignisorientiert (durch Trigger definierte Bedingungen)

Trigger können BEFORE oder AFTER gesteuert sein. Before-Trigger werden oft genutzt, um Berechnungen vorzunehmen, um diese in die Integritätsprüfung aufzunehmen. After Trigger eignen sich besonders gut, um Änderungen in Tabellen zu loggen, um z.B. Change Notes abzuspeichern.

Trigger z.B. werden gestartet, wenn eine Integritätsverletzung aufgetreten ist, um diese zu beseitigen. Aktive DBMS verfolgen diese Strategie viel intensiver.

Ereignis -> Bedingung -> Aktion

Normalisierung

Projektion: Auswahl der Spalten einer Tabelle ohne dabei doppelte Zeilen zuzulassen.

Selektion: Auswahl von Entitäten (Zeilen) aus einer Relation (Tabelle) unter einer bestimmten Bedingung

Natürlicher Verbund:

Verbindet über die Vereinigung zweier oder mehrerer Attributmengen Relationen zur einer neuen Relation

- ist kommutativ und assoziativ
- wenn die Attributmengen disjunkt sind wird ist die Operation ein kartesisches Produkt
- Projektion und Verbund sind nicht invers zueinander, da bei Projektion mehrfache Tupel herausfallen

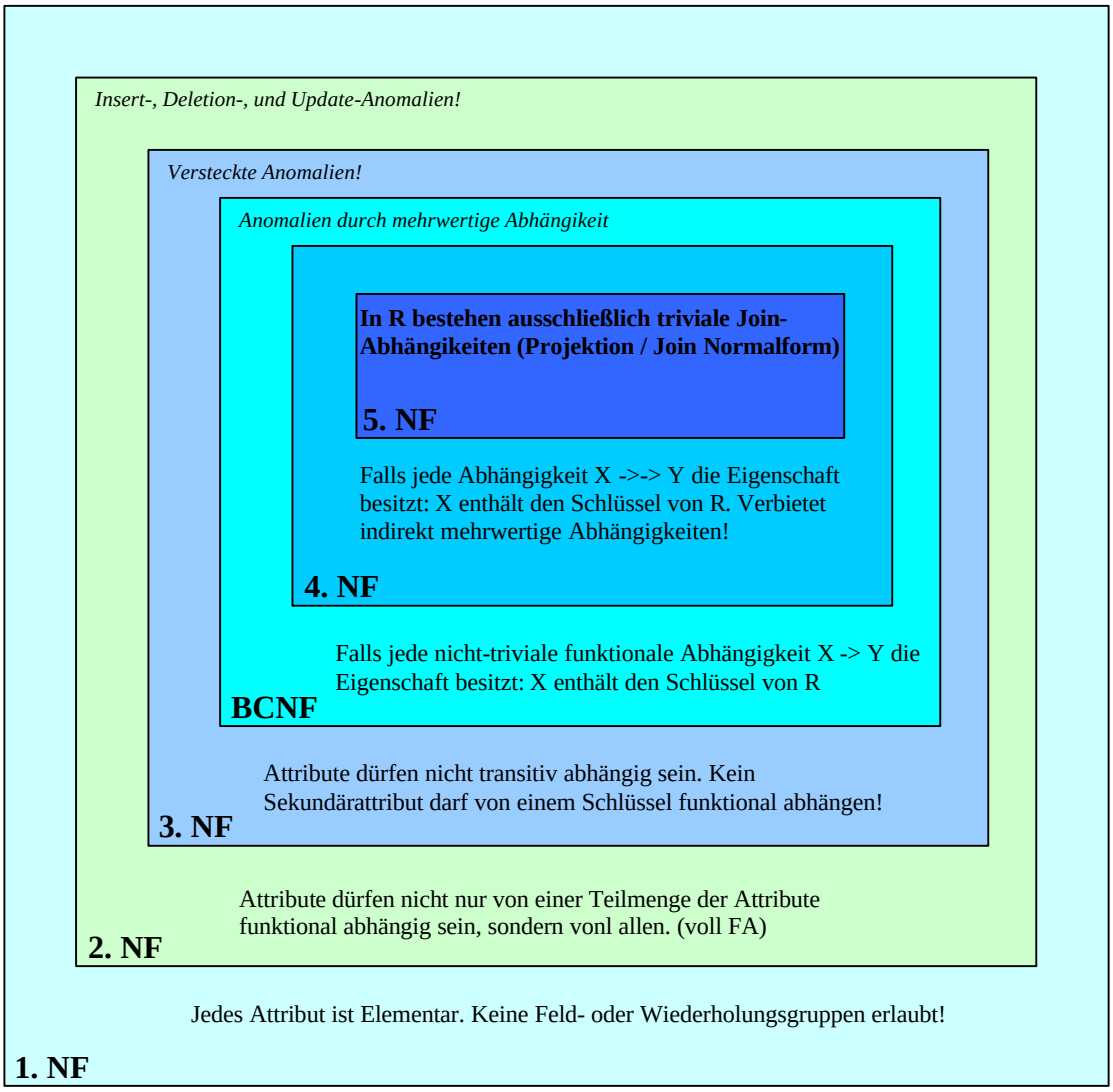
Update Anomalien

Einfüge Anomalie: Ein neues Tupel kann erst eingetragen werden, wenn alle X Informationen vorliegen. Falls Nullwerte erlaubt sind, gibt es dann Probleme falls die fehlende Info zum Schlüssel gehört.

Lösch Anomalie: Wird eine Information aus der Relation entfernt, so gehen mehr Informationen verloren, als gewollt war.

Änderungs-Anomalie: Falls eine Änderung der Daten vorgenommen werden muss, muss dies an mehreren Stellen in der Relation geschehen, da sonst die Konsistenz bedroht ist.

Normalformen



1NF

*Eine Relation ist in der Ersten Normalform, wenn jeder **Attributwert atomar** ist, d.h. es gibt keine zusammengesetzten Attribute in Form von Arrays, Sets oder Aufzählungstypen.*

2 NF

*Eine Relation ist in der Zweiten Normalform, wenn sie in der Ersten Normalform ist und jedes Nicht-Schlüsselattribut von jedem Schlüsselkandidaten **vollständig funktional abhängig** ist.*

Ein Attribut Y ist von einem Attribut X funktional abhängig, wenn es zu jedem X genau ein Y gibt.
Vollständig funktional abhängig bedeutet, daß das Nicht-Schlüsselattribut nicht nur von einem Teil der Attribute eines zusammengesetzten Schlüsselkandidaten funktional abhängig ist, sondern von allen Teilen.

Datenfelder, die von einem Schlüsselkandidaten nicht vollständig funktional abhängig sind, werden in weiteren Tabellen untergebracht. Besteht der Primärschlüssel nur aus einem einzigen, so ist eine Relation in Erster Normalform automatisch in Zweiter Normalform.

3 NF

*Eine Relation ist in der Dritten Normalform, wenn Sie in der Zweiten Normalform ist und jedes Nicht-Schlüssel-Attribut von keinem Schlüsselkandidaten **transitiv abhängig** ist.*

Boyce-Codd Normalform (BCNF)

*Eine Relation ist in Boyce-Codd Normalform, wenn jeder **Determinant** ein Schlüsselkandidat ist.*

Ein **Determinant** ist eine Attributmenge, von der ein anderes Attribut vollständig funktional abhängig ist.
Die Boyce-Codd-Normalform ist eine Weiterentwicklung der 3 NF. In der Dritten Normalform kann es vorkommen, daß ein Teil eines (zusammengesetzten) Schlüsselkandidaten funktional abhängig ist von einem Teil eines anderen Schlüsselkandidaten. Die Boyce-Codd-Normalform verhindert dies.

4 NF

Eine Relation ist in Vierter Normalform, wenn sie in Boyce-Codd Normalform ist und für jede mehrwertige Abhängigkeit einer Attributmenge Y von einer Attributmenge X gilt:
- Die mehrwertige Abhängigkeit ist trivial ist oder
- X ist ein Schlüsselkandidat der Relation

Mehrwertige Abhängigkeit (Multivalued Dependency, MVD)

Ein Schlüssel bestimmt nicht nur ein Attribut X sondern eine ganze Liste einer Attributmenge X. Dann liegt eine mehrwertige Abhängigkeit vor.
Die mehrwertige Abhängigkeit ist trivial, wenn sich die Tabelle nicht weiter zerlegen läßt.

5 NF

Eine Relation R ist in Fünfter Normalform (oder Project-Join-Normalform), wenn sie in Vierter Normalform ist und für jede Join-Abhängigkeit (R1, R2, ..., Rn) gilt:
- *Die Join-Abhängigkeit ist trivial oder*
- *Jedes Ri aus (R1, R2, ..., Rn) ist Schlüsselkandidat der Relation*

Join-Abhängigkeit (Join Dependency)

Eine Relation R genügt der Join-Abhängigkeit (R1, R2, ..., Rn) genau dann, wenn R gleich dem Join von R1, R2, ..., Rn ist. Eine Join-Abhängigkeit ist trivial, wenn ein Ri aus (R1, R2, ..., Rn) gleich R ist.

Die goldenen Regeln der Relationalität

Quelle: http://v.hdm-stuttgart.de/~riekert/lehre/db-kelz/ (Kelz)

Regel 1: Darstellung von Information

Alle Information in relationalen Datenbanken müssen logisch in Tabellen dargestellt sein, insbesondere

- Daten
- Definitionen von Tabellen und Attributen
- Integritätsbedingungen und Beschreibung der Aktion bei Verletzung (siehe Regel 10)
- Sicherheitsinformationen (z. B. Zugriffsrechte etc.)

Regel 2: Zugriff auf Daten

Jeder Wert einer relationalen Datenbank muß logisch durch eine Kombination von Tabellenname, Primärschlüssel und Attributname (Spaltenname) auffindbar sein. Dies bedeutet, daß in einer Tabelle an jedem Schnittpunkt einer Zeile mit einer Spalte nur ein Wert stehen darf.

Beispiel:
Tabellenname: Angestellte ; Primärschlüssel: Angestellter = Meier; Attributname: Gehalt = 5300

Regel 3: Systematische Behandlung von Nullwerten

Nullwerte stellen in Attributen, die nicht Teil eines Primärschlüssels sind, fehlende Information dar und werden durchgängig gleich, insbesondere unabhängig vom Datentyp des Attributes, behandelt.

Beispiel:
Man kann also nicht in numerischen Feldern bei fehlenden Daten das Feld leerlassen, und bei Textfeldern das Zeichen -- einfügen. Fehlende Informationen werden heute meist mit NULL bezeichnet.

Regel 4: Struktur einer Datenbank

Die Datenbankstruktur wird in derselben logischen Struktur wie die Daten gespeichert, also in Tabellen. Dazu muß die Struktur aller Tabellen, die zu einer Datenbank gehören, in einer Tabelle (dem Katalog) zugänglich sein. Diese Forderung bedingt, daß sich eine Änderung im Katalog automatisch in einer geänderten Datenbankstruktur auswirkt!

Regel 5: Die Abfragesprache

Ein relationales System enthält mindestens eine befehlsgesteuerte Abfragesprache, die mindestens die folgenden Funktionen unterstützt:

- Datendefinition
- Definition von Views
  (logische Sichten der Datenbank, die der Benutzer aus den Attributen der Basistabellen erstellt und mit den gewohnten Operatoren manipulieren kann)
- Definition von Integritätsbedingungen
- Definition von Transaktionen
  Eine Transaktion ist eine Folge von Befehlen, die eine Datenbank von einem konsistenten Zustand in einen anderen überführen. Eine Transaktion muß entweder vollständig durchgeführt oder, bei einem Abbruch, vollständig zurückgesetzt werden
- Definition von Berechtigungen

Regel 6: Aktualisieren von Views

Alle Views, die theoretisch aktualisiert werden können, können auch vom System aktualisiert werden.

Beispiel:
Hat man zwei Spalten A und B mit Zahlen, so kann man sich eine weitere Spalte definieren, die A*B enthält. Ändert man einen Wert in dieser Spalte, so kann daraus nicht der Wert der Spalten A und B bestimmt werden, da im allgemeinen aus dem Produkt zweier Zahlen diese zwei Zahlen nicht bestimmt werden können. Dieses View kann also theoretisch nicht aktualisiert werden.

Problem:
Im allgemeinen kann nicht entschieden werden, ob ein View theoretisch aktualisiert werden kann

Regel 7: Abfragen und Editieren ganzer Tabellen

Abfrage- und Editieroperationen müssen als Operanden ganze Tabellen und nicht nur einzelne Sätze erlauben

Regel 8: Physikalische Unabhängigkeit

Der Zugriff auf die Daten durch den Benutzer muß unabhängig davon sein, wie die Daten gespeichert werden oder wie physikalisch auf sie zugegriffen wird. Dies bedeutet, daß Anwendungen nur auf die logische Struktur des Systems zugreifen dürfen.

Beispiel:
Die Daten dürfen auf einem Datenträger durchaus hierarchisch gespeichert sein. Nur die logische Struktur der Datenbank muß relational sein. Ändert der Datenbankverwalter die physikalische Struktur, darf der Anwender davon nichts mitbekommen.

Regel 9: Logische Unabhängigkeit der Daten

Anwendungen und Zugriffe dürfen sich logisch nicht ändern, wenn Tabellen so geändert werden, daß alle Information erhalten bleibt (z. B. beim Aufspalten einer Tabelle in zwei Tabellen)

Regel 10: Unabhängigkeit der Integrität

Alle Integritätsbedingungen müssen in der Abfragesprache definierbar sein und in Tabellen dargestellt werden. Das System muß mindestens die folgenden Integritätsbedingungen prüfen:

- Vollständigkeitsintegrität (*Entity Integrity, Existential Integrity*)
  Ein Primärschlüssel muß eindeutig sein und darf insbesondere keinen Nullwert enthalten.
- Beziehungsintegrität (*Referentielle Integrität, Referential Integrity*)
  Zu jedem Fremdschlüsselwert existiert ein Primärschlüsselwert.

Beispiel: Zu jeder Personalnummer muß es auch einen Namen eines Angestellten geben

Regel 11: Verteilung der Daten

Anwendungen für eine nicht-verteilte Datenbank dürfen sich beim Übergang zu einer verteilten Datenbank logisch nicht ändern.

Beispiel:
Wenn die oben beschriebenen Tabellen in einem Netzwerk auf zwei verschiedenen Rechnern gespeichert sind, darf sich bei der Anwendung nichts ändern, wenn die Tabellen irgendwann auf denselben Rechnern gespeichert werden

Regel 12: Unterlaufen der Abfragesprache

Unterstützt ein relationales Datenbanksystem neben der High-Level-Abfragesprache eine Low-Level-Abfragesprache, so darf diese die Integritätsbedingungen der High-Level-Sprache nicht unterlaufen.

Beispiel:
Die Low-Level Abfragesprache darf z. B. nicht direkt auf die physikalischen Eigenschaften der gespeicherten Daten zugreifen

Codd Vollständige relationale Sprachen

- Prozedurale Algebra
- Deskriptive Kalküle

Relationenalgebra (prozedural)

Ein Ausdruck einer prozeduralen Sprache gibt indirekt immer mit an, wie d.h. durch Ausführung welcher Operationen das Ergebnis berechnet werden kann.

Mengenorientiert. Aus einer oder mehreren Tupelmengen werden neue erzeugt. Anfragen werden in Form formaler Ausdrücke (der Relationenalgebra bzw. des Relationenkalküls) übermittelt.
Die Relationenalgebra ist niedrig polynomiell.

Optimieren kann man die Relationenalgebra indem man die Anfragebäume optimiert. Selektion und Projektion werden in Richtung Blätter verschoben, um schon anfangs kleine Zwischenergebnisse zu erhalten.
Relationenalgebra ist stets in Maßstab für die Ausdrucksstärke einer Sprache. Falls eine Sprache so Ausdrucksstark (d.h. es gibt zu jedem Ausdruck in der einen Sprache einen äquivalenten in der anderen) wie die RA ist, so ist diese Codd-Vollständig.

SQL ist bspw. Nicht prozedural, d.h. der Fragesteller stellt eine Frage, gibt aber keinen Algorithmus zur Lösung vor. Dieser ist in der Sprache selbst enthalten.

Relationenkalküle

Geben im Gegensatz zu prozeduralen Sprachen keine Berechnungsprozedur mit auf dem Weg, sondern nur eine Beschreibung der betroffenen Tupel der Ergebnisrelation.

Sie sind unter bestimmten Umständen Codd-Vollständig!

SQL - Structured Query Language

SQL ist eine nicht-prozedurale, mengenorientierte Sprache für Relationale Datenbanksysteme. SQL kann Relationen als Einheiten verarbeiten, verfügt aber z.Z. noch nicht über Steuerstrukturen wie Schleifen oder Case-Anweisungen. SQL kann interaktiv eingesetzt werden oder in einer Programmiersprache eingebettet sein. (embedded SQL)

SQL-Anweisungen teilen sich in zwei Sprachen:

Datendefinitionsprache, zum Vereinbaren von Datenstrukturen, wie Relationen, Zugriffspfade etc..
Datenmanipulationssprache, zum Verarbeiten der Daten durch Einfügen, Löschen oder Updates.

Wichtige SQL-Anweisungen:

Select-Anweisung

SELECT * FROM relation [WHERE Bedingung];

Bedinung ist durch Vergleichsoperatoren und den Anweisungen BETWEEN, IN, ANY und ALL definierbar.

Join

SELECT name.attribut {,…} FROM relation [Korrelation], relation [Korrelation] WHERE bedingung

Durch die Korrelation ist es möglich Joins mit ein und der selben Tabelle durchzuführen. Korrelation heißt nix anderes als zwei Variablen einzuführen, um die Tabellen unterscheiden zu können.

Um doppelte Tabellenzeilen zu Vermeiden verwende man SELECT DISTINCT!

Gruppenverarbeitung

Um Tupel herauszufiltern, welche auf gemeinsamen Eigenschaften basieren, bietet SQL einige Möglichkeiten:

SELECT attribut, {…} FROM relation GROUP BY gruppenattribut {,…} [HAVING bedingung]

Gruppenauswertungsfunktionen:

AVG(A), COUNG(A),COUNT(*),MAX(A),MIN(A),SUM(A)

Sortieren

ORDER BY [ASC|DESC]

Interne Ebene

Entities werden als Record (in gewissen Sinne das File) gespeichert. Dabei werden Betriebssytemmethoden wie LRU und Pages verwendet.

Physische Datenorganisation

Festplattengeräte

Um Daten dauerhaft speichern zu können, wird ein Externspeicher benötigt. Diese sind um ein Vielfaches langsamer als Arbeitspeicher und Caches. Genau aus diesem Grund sind ausgeklügelte Strukturen und Zugriffsmöglichkeiten notwendig.

Eine Festplatte besteht aus mehreren übereinander liegenden Magnetscheiben. Diese Oberflächen sind in eine feste Anzahl von gleichgroßen Spuren eingeteilt. Übereinander liegende Spuren bilden Zylinder. Die Festplattenkapazität setzt sich dann in erster Linie aus Zylindern zusammen, welche genauso viele Spuren haben, wie es Oberflächen gibt. Jede Spur ist wiederum in eine feste Anzahl von Sektoren eingeteilt, welche die kleinsten adressierbaren Einheiten in einer Festplatte darstellen. So sind Informationen über Angabe von Zylinder, Oberfläche und Sektornummer abrufbar.

Records und Dateien

DBMS speichern ihre Tupel in Form von Records ab. Solche Records werden in Dateien abgelegt. Eine Datei ist eine vom Betriebssystem verwaltete logische Einheit. Relationale DBMS fassen aber oft ihre Daten in einer eigenen allumfassenden katalogisierten Datei zusammen.

Datenblöcke

Um die Effizienz zu erhöhen werden mehrere Sektoren zu Blöcken zusammengefasst. (oder auch Pages etc.) Der Vorteil dieser Herangehensweise ist, das DBMS durch die Nutzung von Blöcken unabhängiger von den Geräten werden.

Adressierung

Adressiert kann ein Block über die Adresse des ersten Sektors werden. Wie so oft werden auch hier meist logische Adressen statt der physikalischen Adressen verwendet. Die logischen Adressen werden dann einfach durchnummeriert, um einen einfachen Zugriff zu gewährleisten. Der Vorteil an der logischen Adressierung ist, dass bei Umspeicherung nur die Umsetzungsfunktion zwischen logischer und physischer Adresse angepasst wird. So müssen die vielfach auftretenden Adressverweise der Verwaltungsdaten im Gegensatz zur physikalischen Adressierung nicht aktualisiert werden.

Indexierung

ISAM Indexed-Sequential Access Method

Für ein sortiert gespeichertes File gibt es einen Index, welcher für die Werte dieses Schlüssels die Adressen der entsprechenden Tupel referenziert.
Ein dünner Index hat zur Folge dass nach der Referenzierung eine Sequentielle Suche nach dem gewünschten Tupel folgt. Ein dichter Index ist eine eindeutige Abbildung der Schlüsselmenge auf die Tupel, was natürlich viel Speicher in Anspruch nimmt.

ISAM kann 1-D-linear aber auch mehrstufig sein. Dabei liegen die Daten an den „Blättern“ dieses mehrstufigen Indexes. Nach Löschoperationen kann es notwendig werden, die Verwaltungsdaten aus Performancegründen neu zu organisieren. Durch Einfügen können so genannte Überlaufketten entstehen.

So eignet sich ISAM ideal für große, statische Datenbestände. Bestände mit starken Schwankungen werden besser als B*-Baum repräsentiert, da dort der Index dynamisch an das Bestandsmaß angepasst wird.

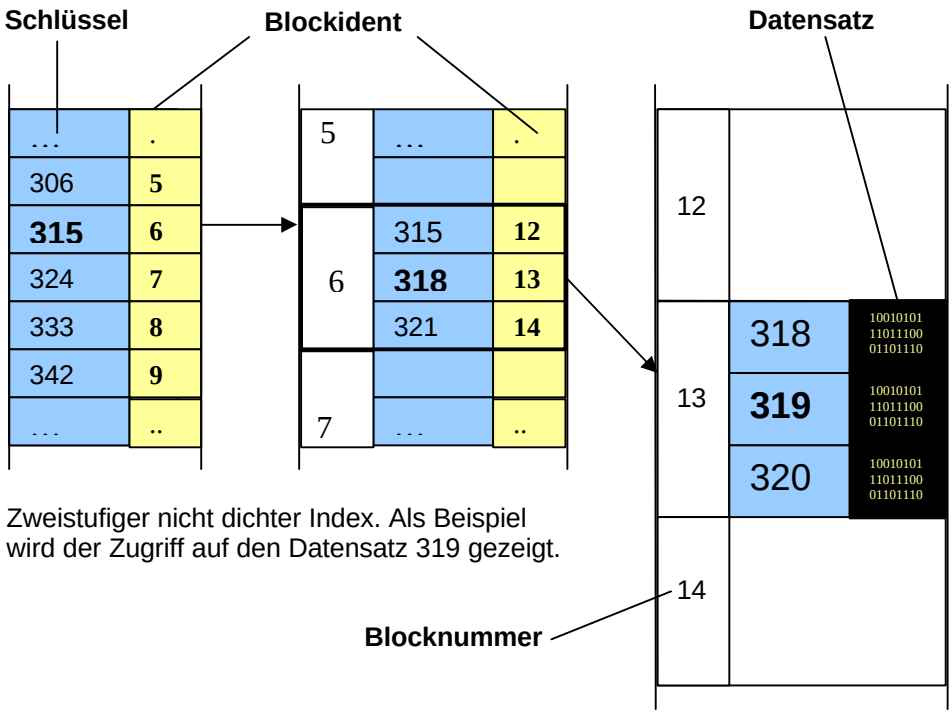
Suchanfragen welche nicht unmittelbar durch den physikalischen Schlüssel erschlagen werden können, machen in großen Datenbanken die Einrichtung von so genannten Sekundärindexen erforderlich. Ein Sekundärindex verwendet normalerweise individuelle Record-Adressen, weshalb nach Bestandsänderungen dieser Index auch angepasst werden muss.

Baumstrukturen als mehrstufige Indexe

ISAM hat den Nachteil, dass mit zunehmender Anfrage/Update Aktivität die Performance immer schneller abnimmt. Bäume haben logarithmische Komplexität und eignen sich deshalb ebenfalls. Im besonderen Maße werden B-Bäume verwendet (jedes Blatt gleich Tief ist, wächst von unten nach oben, bestimmter Füllgrad für Knoten). B*-Bäume sind B-Bäume, welche nur an den Blättern einen bestimmten Füllgrad der Knoten verlangen, um den Baum im Speicher halten zu können.
Sie vereinen die Vorteile von ISAM und B-Bäumen, da sie erstens strikt zwischen Verwaltungsdaten und Bestand (an den Blättern) trennen und gleichzeitig dynamisch ihre innere Struktur nach Operationen anpassen bzw. reorganisieren.
Weiterte Organisationsformen sind mehrdimensionale Bäume und Hash-Tables.

Gird Files

Grid Files verzichten auf die lineare Ordnung der Records und auf die Auszeichnung eines physischen Schlüssels. Sie ordnen Records in einen mehrdimensionalen Record-Raum ein, dessen Koordinatenachsen von den linear geordneten Wertebereichen der gewählten Attribute bestimmt werden.



Zweistufiger nicht dichter Index. Als Beispiel wird der Zugriff auf den Datensatz 319 gezeigt.

Anfrage-Optimierung

Durch algebraische Optimierung, kann vor Ausführung des Codes schon optimiert werden, in dem Umformungen auf Basis der Relationenalgebra vorgenommen werden.

Anfrage > Parsen > Validierung > View-Resolution > Optimierung > QEP > Code > Ausführung > Ausgabe

High Level Optimierung:Auf Grund von Rechenregeln wird die Anfrage optimiert. Ist somit unabhängig vom DBMS.

- Ausnutzen von Kommutativität von Selektion, Projektion und natural Join.
- Ausnutzen von Distributivität der Selektion bzgl. Des Verbundes
- Ausnutzen von Distributivität der Projektion bzgl. Des Verbundes
- Assoziativität des Verbundes, Vereinigung und Durchschnitt…
- …

Low Level Optimierung:QEP Erstellung und Optimierung der Datenorganisation und Implementierung.

- Suche einer suboptimalen QEP mit Hilfe von Heuristiken oder Greedy Verfahren
- Vollständige Strategien zu aufwendig
- Falls Unterschiedliche QEPs für eine Anfrage möglich wird eine Kostenfunktion einbezogen (z.B. Anzahl notwendiger Plattenzugriffe)

Verbundimplementierung

Nested-Loop-Join

sucht nach gleichen Werten der gemeinsamen Attribute in den zu verbindenden Relationen. Dabei muss für jedes Tupel der Relation A jedes Tupel der Relation B verglichen werden. Optimiert wird hier durch blockweißes Lesen der Daten.
Somit entsteht ein quadratischer Aufwand.

Sort-Merge-Join

Sortiert erst die Operanden anhand der Verbund-Attribute aufsteigend und mischt anschliessend die gleichwertigen Attribute zur Ergebnisrelation.

Falls die Operanden-Relation schon sortiert ist, weil z.B. für Attribut A ein Index existiert, ist das Verfahren dem Nested-Loop Join in jedem Fall vorzuziehen, da dann der Aufwand nur noch linear ist…

Hash-Join

Einteilung der Relationen in Partitionen über eine Hashfunktion. Danach kann jede Partition einzeln betrachtet werden.

Query Prozessor

Stellt einen Pool von Ausführungsoperatoren sowie Mechanismen für deren Kommunikation und Synchronisation (zB. Pipes) zur Verfügung. Im Grunde ist der Anfrageprozessor dafür verantwortlich die algebraischen Ausdrücke auf physischer Ebene zu verarbeiten. Dabei wird optimiert durch Parallelverarbeitung mehrerer Prozesse. Ein Prozess schreibt ein Zwischenergebnis in ein File und eine andere Prozess nimmt diese Zwischenergebnisse heraus zur Fortsetzung der Operation.

Datenbank-Tuning

Es gibt vier grundlegende Prinzipien.

1. Lokale Optimierung zur Erreichung globaler Effekte (Hardwareausnutzung überwachen)
2. Partitionierung zur Vermeidung von Bottlenecks (Transaktionen in Lange und Kurze)
3. Intitiale Kosten > Laufende kosten (Leseoperationen sind beim Starten teurer – dann billig)
4. Angemessene Aufgabenverteilung zwischen Systemkomponenten

    a. Concurrency Controll
    b. Recovery- und Logging-System
    c. Umgebendes OS (Puffer, Disk-Scheduling etc.)
    d. Zugrunde liegende Hardware (RAM, Anzahl Prozessoren)

Tuning kann auch durch **Denormalisierung** geschehen, wenn z.B. extrem viele Verbund-Operationen auf die Normalisierte Relation angewendet werden müssen.

Datenintegrität

Durch einzelne wenige fehlerhafte Daten kann der gesamte Datenbestand einer Datenbank zerstört werden. DBMS müssen deshalb einige Fehlersituationen selbstständig erkennen und entsprechend darauf reagieren.

- fehlerbehaftete Bestandsänderungen durch Anwender und fehlerhafte Programmfragmente
- Kollisionen im Mehrbenutzerbetrieb bei zeitgleichem Zugriff auf die Daten
- Systemausfall aufgrund von Hardware- oder anderer Fehler innerhalb eines Abarbeitungsprozesses

Überprüfen von Randbedingungen für Attribut-Ausprägungen und das Überwachen von Beziehungen zwischen Tupeln sind zwei der wenigen Möglichkeiten des DBMS für den ersten Fall. Diese werden im Allgemeinen unter logischer Datenintegrität als Integritätsbedingungen bezeichnet.

Die wichtigste Komponente ist mit Sicherheit der Transaktionsmanager, welcher den Mehrbenutzerbetrieb abgleicht, um Kollisionen zu vermeiden.

Transaktionen und korrekte Verarbeitung

Es muss eine zeitlich verzahnte Verarbeitung verschiedener Aufträge möglich sein. Dabei muss ein gewisser Grad an Fehlertoleranz garantiert werden können.

- ein DB Benutzer darf nicht merken, dass er nicht alleine am System arbeitet
- das DBMS muss selbstständig auftretende Konflikte lösen können

Transaktionsverarbeitung = Concurrency-Control + Recovery-Manager

Wichtigstes Prinzip bei der Transaktionsverarbeitung ist das Read-Write-Transaktionen Modell, welches eine Abstrahierung erlaubt und dadurch Fehlerverarbeitung und Umgehung erleichtert. Read-Write-Transaktionen führen wiederrum zu **Schedules**.

Transaktionskonzept

Was geschieht bei einer SQL-Query an ein DBMS?

1. Prüfung der Syntax und Zugriffsrechte
2. Wahl einer geeigneten Methode zur Ausführung der Operation
3. Feststellen, wie die Operanden gespeichert sind und wie effektiv auf diese Datenstrukturen zugegriffen werden kann (Zugriffspfade)
4. Generieren von Ausführbaren Code, welcher
    a. die beteiligten Daten aus der DB in den Datenbank-Puffer transferiert
    b. die oben gewählte Methode integriert
    c. das Ergebnis ausgibt
5. Ausführen des Erstellten Codes

Um nun mehreren Benutzern ein gleichzeitiges Arbeiten zu garantieren, sind alle Schritte bis 5 reentrant. D.h. sie können parallel ablaufen. Jeder Auftrag bekommt einen Platz im Buffer und wird abgearbeitet, wenn es an der Reihe ist. Hier kann es zur Synchronisationsproblemen kommen. Man kann den Buffer als Cache sehen. Schreibt Prozess geänderte Daten in die DB zurück, muss womöglich ein Update eines anderen Pufferbereiches ausgeführt werden, da dieser sonst noch das falsche Datum trägt. Dies wird meist, wie aus Rechnerarchitektur bekannt, über Dirty-Bits in den einzelnen Seiten des Buffers geregelt. Wird eine Seite benötigt, wird das Dirty-Bit interpretiert und gegebenenfalls die Seite neu von der Platte in den Puffer geladen.

Parallelität

Ein DBMS könnte Konsistenz bewahren, wenn es alle ankommenden Transaktionen in einen Buffer legen würde, um sie dort sequentiell abzuarbeiten. Dies ist aber auf Grund extremer Wartezeiten nicht akzeptabel. Deshalb müssen Transaktionen so weit wie möglich Parallel verarbeitet werden. Dies kann ohne Bedenken geschehen, wenn folgende beiden Bedingungen zutreffen:

- alle Transaktionen sind nur lesend
- alle Transaktionen greifen auf keine gemeinsamen Elemente zu

Sind diese Bedingungen nicht gegeben kommt es zu Konflikten.

Serialisierbarkeit

Zwei Schedules sind äquivalent, wenn sie die gleichen Transaktionen beherbergen und die gleiche Abhängigkeitsrelation besitzen.

Ein Schedule heißt Konflikt-serialisierbar, wenn es für den Schedule einen äquivalenten seriellen Schedule gibt. D.h. wenn ein zeitlich verzahnter Schedule die gleichen Konflikte wie irgend eine serielle Abarbeitung aufweist, so kann dieser als Konsistenz bewahrend eingestuft werden.
Prüfen lässt sich dies mit einem Präzedenzgraph, welcher als Knoten die Transaktionen und die Kanten nach der zeitlichen Abarbeitung beinhaltet. Tritt in diesem Graph ein Zyklus auf, so ist der Schedule nicht Konflikt-serialisierbar. Die topologische Sortierung eines DAG's würde einen korrekten Schedule ergeben.

Synchronisationsprobleme

Lost Update

read1(x) > read2(x) > update1(x) > update2(x)

Das Update von Prozess 1 geht somit verloren.

Dirty Read

Read1(x) > Update1(x) > Write1(x) > Read2(x) > Update2(x) > ABORT1 > Write2(x)

Inconsistent Read

Prozess eins ließt ein Datum und speichert es zwischen. Prozess 2 dagegen, ändert das Datum erst danach und somit hat Prozess 1 den falschen alten Wert des Datums gelesen.

ACID-Prinzip

Atomicity (Atomar)

Aus Sicht des Nutzers wird seine Anfrage (bzw. Programm) komplett oder gar nicht ausgeführt. Falls ein Fehler während der Ausführung auftritt, erscheint die Datenbank in dem Zustand, als wäre der Code nie ausgeführt worden. Der Recovery Manager biegt alles wieder gerade ☺

Consistency (Konsistenz)

Alle Integritätsbedingungen der DB werden eingehalten und die DB wird in einem konsistenten „Zustand“ gehalten, falls sie vor Ausführung der Transaktion in einer solchen war.

Isolation

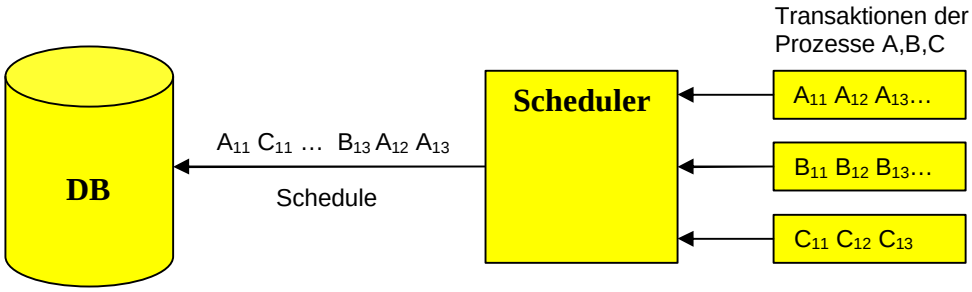
Das Programm läuft völlig unabhängig von eventuell parallel laufenden Prozessen in einem gesicherten Pufferbereich ab. Zur Verarbeitung bekommt des Programm nur sichere konsistente Daten aus der DB. Die Isolation verhindert somit Konflikte im Mehrbenutzerbetrieb.

Durability (Persistenz)

Falls eine „Befehl erfolgreich ausgeführt“ Meldung angezeigt wird, muss sichergestellt sein, dass die geänderten Daten auch nach jedem Hard- oder Softwarefehler permanent in der Datenbank erhalten bleiben. Sogar wenn die Daten sich zu diesem Zeitpunkt noch im Puffer und nicht auf dem Sekundärspeicher befinden.

Read-Write Modell der Transaktionen

Eine Transaktion ist eine endliche Folge von Schritten oder Aktionen der Form read(x) oder write(x). Somit ist eine Transaktion ein Straight-line Programm von Lese und Schreiboperationen. Es wird die Annahme gemacht, das eine Transaktion keine internen Redundanzen aufweisst. D.h. jedes Datenobjekt wird nur einmal gelesen bzw. geschrieben.



Schwerpunkte:

1. Zeitlich verzahnte Ausführungsreihenfolgen von Schritten (read oder write) werden durch Schedules modelliert. (dynamischer Aspekt)
2. Unter welchen Bedingungen ist ein Schedule korrekt?
3. Entwurf und Verifizierung von Scheduling-Verfahren oder kurz Schedulern.

Schedules

Durch Hinzunahme von zwei Pseudoschritten zu den Schritten read und write, lassen sich Schedules modellieren:

Commit - Transaktion erfolgreich beendet. Ergebnisse können für andere Prozesse freigegeben werden.

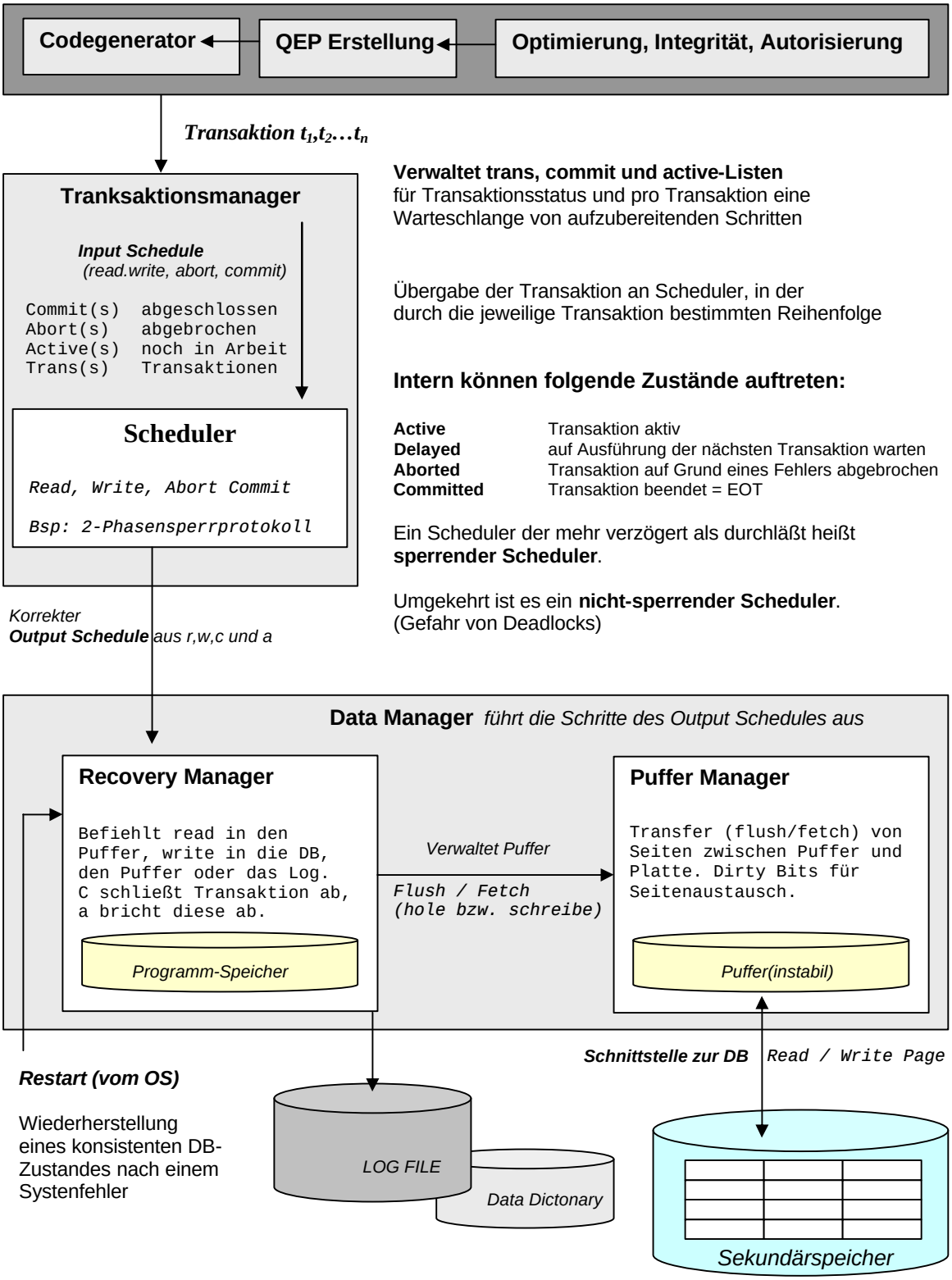
Abort - Transaktion aufgrund eines Fehlers vorzeitig abgebrochen.

Vollständige Schedules sind Schedules, bei denen alle geöffneten Schritte terminiert werden und auch die Schritte a und c enthalten sind. Normale Schedules beinhalten dies normalerweise nicht, da aus Effizienzgründen bei Begin einer Transaktionsverarbeitung noch gar nicht alle Transaktionen vorhanden sind. Diese Treffen beim Scheduler ja eh Schrittweise ein…

Zwei Daten-Operationen stehen im Konflikt, falls sie auf demselben Objekt operieren und eine von ihnen schreibt.

Concurrency Control

Dynamische Erzeugung serialisierbarer Schedules, um Mehrbenutzerbetrieb zu überwachen und zu regulieren.



Sendet der Scheduler ein **Commit** an den Recovery-Manager,
so ist die Transaktion erfolgreich beendet worden. Dies bedeutet auch,
daß alle Transaktionen **im Scheduler initiiert** werden. Wird ein Commit an den Recovery-Manager
übermittelt, so weiß dieser, daß der Effekt nicht in der Datenbank sichtbar werden soll.

Sperrende Scheduler

Bei Sperrenden Schedulern werden so genannte **Locks** verwendet, um den Zugriff auf gemeinsam genutzte Daten zu **synchronisieren**. Eine gesetzte Sperre bedeutet, daß das Objekt für andere nicht verfügbar ist.

Ein sperrender Scheduler unterscheidet zwischen:

Read lock, Write lock, Read unlock,Write unlock

Folgende Regeln gelten

1. falls im Schedule irgendwo ein Read (oder Write) auftritt, so gibt es vor diesem im Schedule ein Lock und irgendwo nach diesem ein Unlock
2. Für jedes Read (oder Write) gibt es genau ein Lock und ein Unlock und nicht mehr
3. read unlocks und write unlocks sind nicht redundant

2-Phasen-Sperren

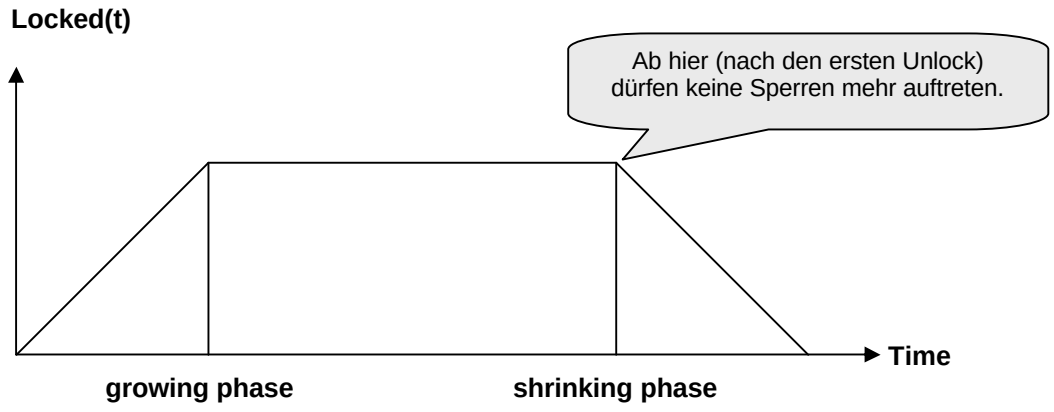
Ein Scheduler ist ein 2PL Scheduler, wenn

1. er die oben erwähnten drei Regeln einhält
2. ist x durch zwei verschiedene Transaktionen gesperrt, so stehen diese nicht in Konflikt
3. ein Sperrprotokoll hat zwei Phasen, wenn nach dem ersten Unlock-Schritt, keine weiteren Locks folgen können

   Somit ist genau nach den ersten Unlock Phase 1 (das Sperren) beendet und Phase 2 begonnen.

Falls ein Objekt mit einer **Lese-Sperre** versehen wurde, so ist es nicht exklusiv gesperrt, sondern im shared mode. Somit können andere Objekte auch Lese-Sperren auf das Objekt referenzieren.

Falls das Objekt mit einer **Schreib-Sperre** versehen wurde, so ist es im exclusive mode, d.h. exklusiv gesperrt. Somit können keine anderen Transaktionen Lese- oder Schreib-Sperren auf das Ojekt referenzieren.



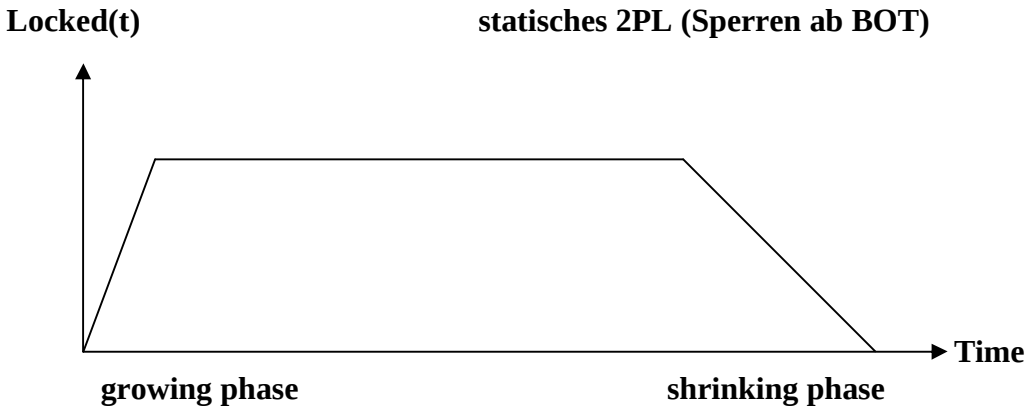
Deadlocks

Das 2-PL ist nicht Deadlock frei. Erklärung dafür ist, daß Locks sich gegenseitig blockieren können, aufgrund der Vorbedingung, daß ein exklusiv gesperrtes Objekt, nicht mehrfach reserviert werden kann. Einmal im Deadlock, kann man ihn nicht einfach durch ein Unlock auflösen, da dann ja die nicht aufgerufenen Locks nicht mehr in Kraft treten könnten.
Eine weitere Situation, wo Deadlocks entstehen können, sind Lock-Konvertions. Updaten 2 Transaktionen gleichzeitig ihr Lesesperren auf ein Objekt X zu Schreib-Sperren, kann dadurch auch ein Deadlock provoziert werden.
Deadlocks können mit Hilfe von Timeouts oder Wartegraphen (Zyklus bedeutet Deadlock) erkannt werden.

Varianten des 2PL

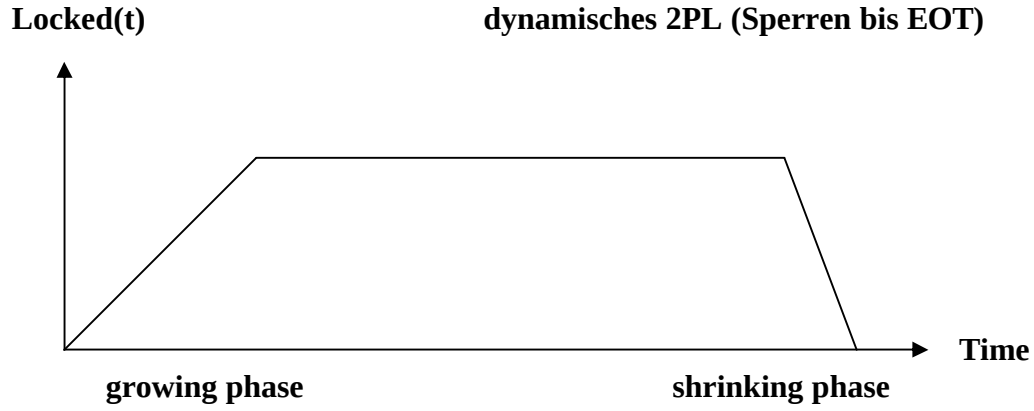
Conservative two-phase-locking C2PL (statisches 2PL)

Jede Transaktion setzt zu Begin ihrer Ausführung (BOT) alle Sprerren gleichzeitig. Nachteil dieses Verfahren ist es, daß alle Read- und Write-Sets vorher bekannt sein müssen. Dazukommt, daß falls ein Konflikt mit einer Sprerre Auftritt, die gesamte Transaktion warten muß, bis diese durch das Unlock der anderen Transaktion aufgelöst wird. Da dies selten der Fall ist, wird es wenig benutzt. DerVorteil ist hier aber, daß Deadlocks umgangen werden, da diese nicht auftreten können.



Strict two-phase-locking S2PL (dynamisches 2PL)

Alle gehaltenen Sperren einer Transaktion werden erst nach ihrem letzten Schritt aufgelöst.



Tuning von Sperrenden Schedulern

1. Entfernen unnötiger Sperren
2. Zerlegen langer Transaktionen in mehrere kurze
3. spezielle Techniken für lange Reads (Read Only keine Sperren weil Backup dieser)
4. sinnvolle Granularität (Record-Level, Page oder Table Locking)
5. Schema Updates bei geringer Aktivität durchführen (System Katalog wird zum Flaschenhals)
6. Deadlock Intervall bei 2PL (Zylisch Wartegraphen auf Zyklen untersuchen)

Zeitstempel-Verfahren

Vor allem bei verteilten Datenbanken wird das Zeitstempel-Verfahren angewendet. Dieses Verfahren soll von vornherein verhindern, dass Deadlocks entstehen können. Anstelle von Sperren, wird eine Transaktion mit einer drohenden Verletzung der Serialisierbarkeitskriterien abgebrochen und neu gestartet.

Recovery

Das Recovery System eines DBMS ist für die Fehlerverarbeitung zuständig. Dabei werden die Fragen geklärt, wie es Transaktionen bei Fehlererkennung abbrechen kann, ohne andere aktive Transaktionen zu beeinflussen oder wie sich das System nach einem Totalausfall wieder in einen stabilen konsistenten Zustand zurückversetzen kann.

Folgende Arten von Situationen muß es erfassen:

1. Transaktionsfehler, wenn eine Transaktion den Commit-Status nicht erreicht (abort). Der Zustand der DB vor Ausführung der Transaktion muß wieder hergestellt werden. Somit müssen eventuell ausgeführte Transaktionen wieder rückgängig gemacht werden.
2. Systemfehler, wie Fehler im DBMS Code, BS oder in der Hardware selbst. Des weiteren zählen Stromausfälle mit dem Verlust des instabilen Speichers (Puffers) zu Systemfehlern.
3. Mediafehler, wie Head Crashes von Platten, wo Teile des stabilen Speichers verlorengehen.

Das Recovery Protokoll

Es gibt zwei Klassen von Transaktionen, welche beim Recovery unterschieden werden müssen.

1. Transaktionen, welche vor dem Crash commited waren.
2. Transaktionen, welche zum Zeitpunkt des Fehlers noch aktiv waren.

Der Rücksetzmechanismus muß nun sicherstellen, daß alle Änderungen der DB einer not-commited transactions mit einer **UNDO**-Operation rückgängig gemacht werden und alle freigegebenen Operationen mit einer **REDO**-Operation erneut durchlaufen und permanent auf die DB übertragen werden.

Es gibt zwei Arten von Datamanagern. **In-Place-Updateing** bedeutet, daß immer eine Kopie der aktuellen Daten sich im Puffer befindet. Ein zweites Konzept hält immer eine weitere Sicherheitskopie im Buffer – die Schattenkopie. Das Konzept heißt Schattenkopiekonzept oder **Defeered Updating**.

Der Puffer enthält je Seite ein Dirty-Bit, falls die Seite mit der Originalseite auf dem Sekundärspeicher inkonsistent ist.

Recovery-Manager

Er stellt sicher,daß alle Operationen atomar ausgeführt werden und konkurrierende Zugriffe synchronisiert werden. Aber der Scheduler ist es, welcher die Ausführungsreihe bestimmt.

Um nach einem Systemfehler erfolgreich ein restart durchführen zu können, ist ein Logbuch notwendig. Es gibt zwei grundlegende Log-Konzepte.

Physischer Log	**Logischer log**
Enthält alle Informationen über die von Transaktionen geschriebenen Werte. (Transaktion, Before- und Afterimage) Die Einträge werden sortiert Log gespeichert.	Enthält eine Beschreibung der Operationen in deklarativer Art und Weise. Macht das wiederherstellen aufwendiger.

Zusätzlich zur vollen wiederherstellung (Rollbacks), ist es notwendig, alle Listen (active, commit, abort) und auch die Zeiten BOT(t) und EOT(t) im Log zu sichern. So kann aufgeklärt werden, welche Operationen mit Undo- und welche mit Redo-Operationen zu bearbeiten sind.

Quellen:

Gottfried Vossen
Datenbankmanagementsysteme

Heuer und Saake
Konzepte und Sprachen

Prof. Benn
Skript und Vorlesung

Relationale Datenbanken
Andreas Kelz

Word Wide Web
Verschiedenste Seiten