Datenbanken II

Transaktionsmanagement des DBMS hat 2 Aufgaben:

1. Synchronisation: Kontrolle der Parallelarbeit
2. Recovery: Wiederherstellung konsistenter Zustände nach Fehlern.

4. Synchronisation

1. Synchronisationsproblem

- lost update (T1 und T2 lesen aus, T1 schreibt, T2 schreibt,hätte jedoch Wert vorher nochmals lesen müssen. T1 daher unwirksam).
- Inkonsistente Sicht (a:=a-100;b:=b+100; wird nach 1.Anweisung gelesen: neuer Wert für a, alter für b!)
- Inkonsistenz der Datenbank (T1 erhöht a und b um 10, T2 a und b um 10%- Reihenfolge kann zu Inkonsistenzen führen!)
- Phantome (Filterung im ersten Durchlauf, weitere Filterung der Auswahl im zweiten- wird dazwischen neuer Datensatz eingefügt, wird dieser im 2. Lauf nicht angezeigt)

2. Transaktionen:

Eigenschaften einer Transaktion: ACID

- Atomicity (ganz oder gar nicht ausgeführt)
- Consistency (von konsistetem zu konsistentem Zustand)
- Isolation (Ablauf, als sei Transaktion die einzige)
- Durability (erfolgreich beendete Transaktion dauerhaft)

3. Serialisierbarkeit

- Ein paralleles System von Transaktionen ist dann korrekt synchronisiert, wenn es serialisierbar ist, dh. Wenn es mindestens eine (gedachte) serielle Ausführung derselben Transaktion gibt, die a) denselben Datenbankzustand b) dieselben Ausgabedaten der Transaktion liefert.
- Transaktionsmanager:
  Schedule=Folge von Lese und Schreiboperationen
  Aufgabe des Transaktionsmanagers: Strom von Operationen so verändern, dass das Transaktionssystem insgesamt serialisierbar wird.
- Kriterium für Serialisierbarkeit: Eine Schedule ist dann seriell, wenn für jede Transaktion in der Schedule alle ihre Operationen unmittelbar nacheinander ausgeführt werden.
  2 Operationen stehen in Konflikt, wenn sie auf dasselbe Objekt zugreifen und mindestens eine von ihnen ein write ist.
  Serialisierbarkeitskriterium: Ein Transaktionssystem ist dann serialisierbar, wenn der zugehörige Abhängigkeitsgraph zyklenfrei ist (kante = Abhängigkeit= Konflikt zweier Operationen.)

4. Klasse von Synchronisationsverfahren:

- Verifizierende Verfahren = optimistisch: Bei Nichtserialisierbarkeit Abbruch.
- Präventive Verfahren = pessimistisch (Sperrverfahren, Zeitstempel)

5. Sperrverfahren

- Exclusive Sperren: Auf das gesperrte Objekt kann keine andere Transaktion zugreifen.Transaktionsmanager prüft, ob eine Transaktion die erforderlichen Sperren besitzt, Verwaltung der Sperren durch den Lock-Manager.
  Probleme:
  1. Sperrprotokoll-wie setzten und freigeben
  - zB 2-Phasen-Sperrprotokoll: In Wachstumsphase setzen der Sperren und in Schrumpfungsphase freigeben (beim Schrumpfen kein weiteres Setzen!)
  - Strikte Zweiphasigkeit: wie vor, nur dass Sperren erst zum Transaktionsende freigegeben werden dürfen, ansonsten fortgepflanzter Rollback möglich (T1 gibt Objekt frei, T2 sperrt dieses, T1 stürzt ab (abort)->Wert von T2-read ungültig-ZB. Flugscheinverkauf, Schein schon in Hand des Kunden)
  - Preclaiming: Alle Objekte werden schon zu Beginn von T gesperrt. Problematisch: Schwierig, Objektmenge im Voraus zu bestimmen, hohe Kosten, Einbußen an Parallelität.
  2. Sperrmodi -welche Arten von Sperren
  - Lesesperre (S-Lock):Erlaubt besitzender Transaktion nur lesenden Zugriff und anderen

Transaktionen das Setzen weiterer Lesesperren (keine Schreibsperren!)

- Schreibsperre (X-Lock):Erlaubt lesenden und schreibenden Zugriff, andere Transaktionen können keine gleichzeitigen Sperren erhalten.
- Intensionssperren
- hot spots: Sind Objekte, auf denen periodisch viele Transaktionen gleichzeitig schreiben müssen (zB Summenfelder, auf die bei jeder Ein- und Auszahlung gebucht werden muss). Verbesserung durch atomare Operationen:Add(x,Wert), Sub(x,Wert). Beide kommutativ. Dadurch können sie schwächere Sperren setzen als normale Schreiboperationen mit Add-Lock und Sub-Lock: Ist Add-Lock gesetzt, so kann dennoch einer anderen Transaktion ein Add-oder Sublock gewährt werden. Nicht kompatibel zu S- und X-Locks!

3. Sperreinheit – welches sind diese in der Datenbank

Sperrhierarchie: Tupel(Satz)-Relation (Datei)- Area(größere disjunkte Bereiche einer Datenbank)- Datenbank.

Übergeordnete Ebenen sperren untergeordnete. Wird eine Sperre auf ein Objekt O gesetzt, wird eine Intensionssperre auf alle übergeordneten Objekte gesetzt (IS und IX):

- Mit IS erhält T das Recht, auf einer tiefergelegenen Ebene Objekte zu lesen und darf dort IS und S-Sperren setzen.
- Mit IX erhält T das Recht, auf tieferen Ebenen alle Sperren zu setzen.

	IS	IX	S	X	<- gesetzte Sperre
IS	ja	ja	ja	nein	
IX	ja	ja	nein	nein	
S	ja	nein	ja	nein	
X	nein	nein	nein	nein	

^
angeforderte Sperre

- Lösung des Phantomproblems:T muss ganze Relation sperren oder Index-Locking: Sperren auf Index-Ebene (zB Indexeintrag auf den gefiltert wurde, etwa Berlin).

4. Deadlock

2 Transaktionen warten jeweils auf die andere, um ein Objekt sperren zu können. Lösung durch

1. Time-out: Wahl der Zeitschranke jedoch kritisch
2. Deadlockerkennung und Auflösung durch Wartegraph:Für jede Transaktion ein Knoten, und eine gerichtete Kante, wenn Ti auf Tj wartet. Deadlock, wenn Zyklen vorhanden. Abbruch der Transaktion mit den wenigsten Kosten (bisher verbrauchte Rechnerzeit, Kosten für Rücksetzen etc.).
   Dadurch aber auch zyklischer Neustart möglich->Transaktion wird beendet und neu gestartet, gerät in Deadlock, wird wieder als Opfer ausgewählt.

5. Livelock: Transaktion verhungert in Warteschlange, zB aufgrund niedriger Priorität und wird immer wieder überholt. Verhinderung durch first-come-first-served oder durch erhöhen der Priorität bei lange wartenden Transaktionen.
6. Der Lockmanager:
   - Wird aufgerufen durch LOCK und UNLOCK
   - implementiert Sperren als Einträge in eine Sperrtabelle, üblicherweise Hash-Tabelle mit Schlüssel Objekt-ID.

6. SQL-Transaktionssyntax:

Transaktionen werden gestartet durch:
ALTER, CREATE,DROP, GRANT,REVOKE (Schema-Statements)
OPEN, CLOSE, FETCH, INSERT, UPDATE, DELETE, FREE LOCATOR, HOLD LOCATOR (Datenstatements)
START TRANSAKTION; COMMIT AND CHAIN , ROLLBACK AND CHAIN (SQL 99).
RETURN (Kontrollstatement).

Beenden durch: COMMIT ,ROLLBACK.

- COMMIT: Beendet T, veranlasst DBMS alle Änderungen dauerhaft zu machen.Anschliessen einer neuen Transaktion mit AND CHAIN, sichern und keinen neue Transaktion anschliessen mit NO CHAIN.
- ROLLBACK: Beendet T, veranlasst DBMS alle Änderungen zurückzunehmen. Keine neue Transaktion mit ROLLBACK, ROLLBACK AND WORK, ROLLBACK AND NO CHAIN, neue Transaktion anschliessen mit ROLLBACK AND CHAIN, ROLLBACK WORK AND CHAIN.
  Möglichkeit nur bis SAVEPOINTS zurückzugehen.
  INSERT INTO ANG...
  SAVEPOINT PROJEKT_GESICHERT
  .....

  ROLLBACK TO SAVEPOINT PROJEKT_GESICHERT
- START TRANSAKTION/SET TRANSAKTION:
  neu in SQL99.
  SET TRANSAKTION mit Modi:
  - Zugangsmodus (READ,READ WRITE)
  - Isolationslevel:
    1. READ UNCOMITTED (niedrigster Level, darf lesen, worauf Transaktionen geschrieben haben, die noch nicht mit COMMIT bestätigt wurden, evtl falscher Wert!),
    2. READ COMITTED (darf lesen, worauf Transaktionen geschrieben haben, die mit COMMIT bestätigt wurden),
    3. REPEATABLE READ (Verhindert Non Repeatable Reads, diese entstehen, wenn ein gelesener Wert von einer anderen Transaktion geschrieben wurde und dann dieser nochmals gelesen wird, schwierig zu entscheiden welches nun der wahre Wert ist)
    4. SERIALIZABLE(Standardlevel).

  Beispiel:
  SET LOCAL TRANSAKTION
  READ
  ISOLATION LEVEL REPEATABLE READ

1. Zeitstempelverfahren: Jedes Objekt besitzt TSR(x) und TWR(x)-Zeitstempel. Ist der Zeitstempel der Transaktion kleiner als TSW(beim Lesen) oder kleiner als max(TSR,TSW) (beim Schreiben), so wird T abgebrochen. Kein Deadlock möglich!Serialisierbarkeit gewährleistet
2. Optimistische Verfahren: Erst bei COMMIT wird geprüft, ob T an nicht-serialisierbarer Schedule beteiligt ist. 2 Möglichkeiten: Entweder Abhängigkeitsgraph mitführen und vor COMMIT auswerten oder Zeitstempelverfahren: Jede Transaktion untergliedert in 3 Phasen: Arbeitsphase, Validationsphase und Schreibphase. In Arbeitsphase nur lesend auf Datenbank, in Validation Prüfung für COMMIT (feststellen, ob gelesene Objekte noch aktuell sind).
3. Erweiterungen des Transaktionskonzepts:
   1. Konzept langer Transaktionen: Bei langen Arbeitsvorgängen kein Alles-oder Nichts-Prinzip, da bei Recovery Verlust der gesamten Arbeit. Daher Checkout/Checkin-Mechanismus: Checkout in private Datenbank, Checkin wieder in die öffentliche.Objekt bleibt in dieser Zeit gesperrt.
   2. Geschachtelte Transaktionen: Top-Level-Transaktion gegliedert in mehrere Subtransaktionen. Rückgängigmachen durch Kompensation, da einzelne Subtransaktionen u.U schon abgeschlossen sind.
   3. SAGA=lange Transaktion, die aus einer Folge eigenständiger Subtransaktionen mit zugehörigen Kompensationstransaktionen besteht.

5. Recovery

Datenbanken II

1. Das Recoveryproblem: Übernimmt der Recovery-Manager des DBMS. Nötig wegen möglichem undefinierten Zustand, dh. physische (Zeiger auf falsche Objekte) oder logische(widersprüchliche Daten) Inkonsistenz.
Fehlerfälle:
   1. Transaktionsfehler: Fehler im Programm, gewollter Abbruch, Deadlock. Rücksetzen von T=Rollback
   2. Systemfehler: Fehler im DBMS, Crash des BS.. Neustart nötig, alle nicht beendeten Transaktionen müssen rückgängig gemacht werden.
   3. Speicherfehler: zB disk head crash. Kopien der DB nötig=Rekonstruktion.

2. Rücksetzbarkeit:
   - COMMIT und ABORT

Aktive Transaktion kann abgebrochen werden. Bei ABORT wird ROLLBACK ausgeführt.
   - Fortgepflanzter Rollback: Liest eine andere Transaktion TB ein von TA verändertes Objekt und wird TA abgebrochen, so muss TB auch abgebrochen werden->pflanzt sich fort.
   - Rücksetzbedingung fürs Lesen: Eine Transaktion T kann erst vom DBMS abgeschlossen werden, wenn alle Transaktionen, von denen T Daten gelesen hat, abgeschlossen sind. (COMMIT muss also evtl künstlich hinausgezögert werden. In der Praxis: Transaktion darf nur Daten von abgeschlossenen Transaktionen lesen)
   - Rücksetzbedingung fürs Schreiben: Eine Transaktion T darf ein Objekt x erst schreiben, wenn alle Transaktionen, die x zuvor geschrieben haben, abgeschlossen oder abgebrochen sind.
   - Strikte Ausführung: Das DBMS gewährleistet, dass eine Transaktion T ein Objekt erst dann lesen oder schreiben kann, wenn alle Transaktionen, die x geschrieben haben, abgeschlossen, oder abgebrochen sind..

   2.

   3. Der Recovery-Manager: = die Komponente des DBMS, die für sämtliche Recovery-Aufgaben zuständig ist. Er muss 2 Prinzipien sicherstellen:
   - UNDO: Muss alle bereits durchgeführten Änderungen einer aktiven Transaktion rückgängig machen können
   - REDO: Muss die von einer abgeschlossenen Transaktion durchgeführten Änderungen wiederherstellen können.

4. Recovery-Mechanismen: Ziel: Jüngsten konsistenten Zustand der DB nach dem Auftreten von Fehlern wiederherstellen.
Techniken zum Führen der nötigen Informationen:
1. Log
   - Physisches Log: Enthält Informationen über Werte von Datenbankobjekten vor und nach den Änderungen durch Transaktionen. Muss mindestens enthalten: Marke für Beginn der Transaktion(BOT), Before-Image für jedes veränderte Objekt (Identifikation des Objektes, Wert des Objektes vor der Veränderung, Indentifikation der ändernden Transaktion), After-Image für jedes veränderte Objekt, Marke für das Ende einer Transaktion (cOMMIT/ABORT).
   - Logisches Log: Enthält statt der Objektwerte Infos über die durchgeführten Operationen (Tupel x wurde in Relation R eingefügt...)

   Log-Protokolle:
   - UNDO-Protokoll (write ahead prot.): Vor Schreiben in DB muss Before-Image im Log gespeichert sein.
   - REDO-Protokoll: Vor COMMIT müssen alle Objektveränderungen in der Datenbank oder im Log gespeichert sein.
   - UNDO=Rückwärts-Durchlauf im Log beim Einspielen des Before-Images
   - REDO=Vorwärtsdurchlauf beim Einspielen des After-Images

5. Recovery-Strategien:

UNDO notwendig, wenn nicht-beendete Transaktionen Daten in die Datenbank schreiben

können->bei Systemfehler können dann nach Abbruch Daten auf der DB sein, die von nicht beendeten Transaktionen stammen, diese müssen ungeschehen gemacht werden.
REDO notwendig, wenn Transaktion beendet werden kann, bevor alle ihre Schreibdaten in die DB übertragen sind-> bei Abbruch enthält die DB jetzt nicht alle Daten bereits beendeter Transaktionen.
4 Recoverystrategien:

1. UNDO/REDO:Neustart-Algo: Setze alle zum Zeitpunkt des Abbruchs nicht beendeten Transaktionen zurück und schreibe alle Änderungen beendeter Transaktionen, die noch nicht in der DB sind, aus dem Log in die DB. Damit nicht das ganze Log nach zum Zeitpunkt des Abbruchs aktiven Transaktionen durchsucht werden muss, werden in Abständen Checkpoints in das Log geschrieben (enthalten Liste der zu diesem Zeitpunkt aktiven Transaktionen).Dazu werden geschrieben: After-Images der abgeschlossenen und Before-Images der abgebrochenen Transaktionen.
2. UNDO/NO-REDO: REDO dann nicht notwendig, wenn vor COMMIT alle Änderungen in die DB geschrieben sind->vor COMMIT-Meldung DB-Cache in DB zwingen. Neustart: Alle nicht beendeten Transaktionen werden mit Before-Images zurückgesetzt.
3. NO-UNDO/REDO:UNDO-Vermeidung durch a) Cache-Management verdrängt nur abgeschlossene Objekte b) Transaktionen schreiben Änderungen nicht in Cache, sondern in privaten Speicherbereich.->Übertragung in Cache erst nach Commit->in Cache sind damit nur abgeschlossene Objekte. Neustart: Nur After-Images nachfahren, weil nicht sicher, ob Veränderungen abgeschlossener Transaktionen wirklich in DB geschrieben wurden.
4. NO-UNDO/NO-REDO: Es dürfen keine Änderungen vor COMMIT in DB geschrieben werden und bei COMMIT müssen alle Änderungen in DB sein->einzige atomare Operation. Realisierungsprinzip: **Shadowing**! ->In Hauptkatalog wird Speicherplatz festgehalten, auf dem der letzte committed Wert eines Objekts steht, bei Änderungen wird Kopie des Objekts angelegt und die Position in Hilfskatalog festgehalten. Bei COMMIT: Rollentausch von Haupt- und Hilfskatalog. Nachteile: Zugriff auf DB indirekt und Clustering schwer zu realisieren.

6. Logisches Log
Aufwand geringer als physisches Log: Insert record r into file x.
Recovery-Manager muss erheblich erweitert werden->muss für jede Update-Operation o Einträge im Log erzeugen, mit deren Hilfe er später sowohl REDO als auch UNDO von o erreichen kann- inverse Operationen müssen verfügbar sein.
Wesentlicher Unterschied zw. Logisch und Physisch: Beim Schreiben eines Before-Images ist es unerheblich, ob das Objekt tatsächlich in DB verändert wurde, es wird nur ein Wert zurückgeschrieben.
Bei Logischem Logging hängt die Wirkung einer UNDO-Operation vom aktuellen Zustand der DB ab!
UNDO-Operation kann nur ausgeführt werden, wenn die ursprüngliche Operation tatsächlich in der DB realisiert wurde zB wenn eine Op schon im Logfile steht, aber noch nicht in DB geschrieben wurde, etwa insert->delete führt dann zu Fehler.
Rücksetzen im Falle kurzer Sperren auf hot spots: Auch für die speziellen, vertauschbaren Operationen muss jeweils die inverse Operation ausgeführt werden.
7. Speicherfehler:

- dumps (Kopien) der DB mitsamt After-Images-Logs speichern
- mirroring

dumps teuer, Speicherfehler häufig nur auf kleinen Teilen der DB->Schreiben von dumps in kleinen Schritten und/oder Rekonstruktion von kleinen Teilen der DB

6. Integrität der DB

- Semantische Integrität: Benutzerfehler
- Operationale Integrität: Synchronisation paralleler Transaktionen
- Recovery: Fehler in DB erkennen und korrekten DB-Zustand herstellen
- Datenschutz: Verhindern unberechtigter Zugriffe
- Konsistenz: Widerspruchsfreiheit und Vollständigkeit der Daten

Semantische Integrität

Integritätsbedingungen nötig

- durch Datendefinitionen (DDL)
- in relationalen Systemen durch:
  - Schlüssel: Duplikateleminierung durch Primar-, referentielle Integrität durch Fremdschlüssel
  - Existenz von Werten (NULL, NOT NULL)
  - Eindeutigkeit von Werten (UNIQUE)
  - Bedingungen von Werten (CHECK)
  - aktive Komponenten (TRIGGER)

zB. Löschverhalten festlegen durch a) SET DEFAULT/NULL b) CASCADE c) RESTRICT beschreiben, ob Tupel/Werte von ANGEST in ANG_PRO a) auf speziellen Wert gesetzt, b) kaskadierend gelöscht oder c) Löschung des ANGEST verbieten, solange noch Tupel in ANG_PRO existieren, die sich auf den ANGEST beziehen.
Mit CHECK kann man zulässige Werte weiter eingrenzen (CREATE TABLE ANGEST...CHECK(GEHALT BETWEEN 1000 AND 2000)...), mit TRIGGER wird festgelegt, dass beim Eintreten eines bestimmten Ereignisses eine Folge von Anweisungen ausgeführt wird.

- Klasse von Integritätsbedingungen:

Integritätsbedingungen unterscheiden für Zustand (zB Vorgabe eines Wertevorrats) und Veränderung (daher müssen viele Integritätsbedingungen nicht an Operationen, sondern an Transaktionen gebunden werden zB ist die Kombination der Steuerklasse bei Ehepartnern noch korrekt?).

7. Physische Datenorganisation

Zugriffszeiten: Arbeitsspeicher 10^-6 bis 10^-7 sec, Platte 8msec.
Systempuffer=Speicherbereich im Hauptspeicher, der die von der Platte übertragenen Seiten aufnimmt. Größe abhängig von Art des Anwendungsprogrammes.

- Das Entwurfsproblem

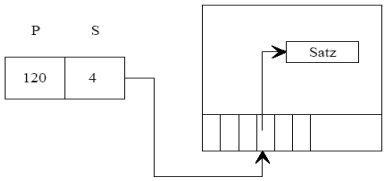
Entwurfsprozess: Zuerst konzeptuelles Modell (meist Entity-Relationsship-Modell), davon abgeleitet dann das logische Modell, darin dargestellt die Informationen des konzeptuellen Modells mit Hilfe der Konstrukte des ausgewählten DB-Modells (Netzwerk, Relational).
Relational: Anwendungsprogramme berücksichtigen nicht die interne Organisation der Daten, Realisierung auf Basis der logischen Tabellenstrukturen- entkoppelt von physischer Datenspeicherung.
Netzwerk: Zugriff auf Daten durch Navigation.
Allerdings: Physische Speicherung wichtig für Performance!
Datenbank-Admin muss physische Unterstützung organisieren:= Beschleunigen des Auffindens der Entities durch Art der Abspeicherung der Daten (Primärorganisation) oder zusätzliche Daten (Sekundärorganisation, zB Indexe).
Trade-off: Gewinn beim Retrieval (Lesen) bedeutet Verteuerung des Update.
Entwurfsschritte der physischen Datenorganisation:

1. Abbildung der Entities auf interne Sätze
2. Festlegung der Primärorganisation der Daten.
3. Einrichtung von sekundären Zugriffspfaden zu internen Sätzen

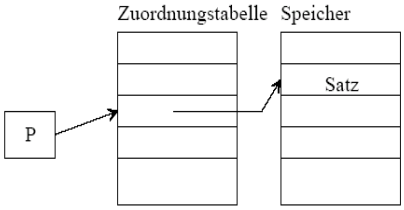
1. Interne Sätze
   1. Abbildung der Entities auf interne Sätze (=physische Sätze)
      Gleichartige Sätze werden zu Satztypen zusammengefasst. Festlegung durch Admin, welche Attribute der konzeptuellen Entity-Typen zu Satztypen des internen Modells zusammengefasst werden und wie die Sätze dieser Typen realisiert werden.
      a) Zusammenfassen von Attributen verschiedener Entities in einem internen Satz bei 1:n-Beziehungen zB. Name u. Geburtsdatum der Kinder eines Entities ANGESTELLTER.
      b) Aufspaltung eines Entity-Typs in mehrere Satztypen ANGEST1 und ANGEST2, wobei ANGEST1 die häufig gebrauchten Attribute beinhaltet.
   2. Realisierung interner Sätze
      1. Speicherung variabel langer Sätze als Sätze fester Länge
         a) Wenn Obergrenze für Satzlänge bekannt ist: Nullwerte für die nicht belegten Werte
         b) Logischen Satz in Folge von Sätzen fester Länge zerlegen und Verkettung dieser physischen Sätze über Zeiger. ANG_NAME,...^PT ->ANG_KIND, GEBTAG

2. Direkte Implementierung variabel langer Sätze
   - Längenangaben: In speziellem Feld wird angegeben, wieviele Werte das Attribut umfasst: 3|Wert1|Wert2|Wert3
   - Zeiger: Am Satzanfang wird für jedes Attribut ein Zeigerfeld eingerichtet, dieser weist auf das erste Feld des zugehörigen Attributwerts.

2. Addressierung von internen Sätzen
   Alle Seiten erhalten Seitennummer, hardwaremäßige Ansteuerung durch das BS geschieht durch physische Adresse zB (Zylinder#, Spur#,Sektor#).
   Aufgabe des DBMS: Alle Sätze eines Typs zu einer internen Datei zusammenfassen. Interne Datei selbst wird auf einer wohldefinierten Menge von Seiten abgespeichert. Organisation der Seitenreihenfolge durch physische Nachbarschaft oder Verkettung.
   Einfügen eines neuen Satzes: Unter Berücksichtigung des Ordnungskriterium der internen Datei (zB. nach Primärattribut).
   Auffinden: durch weitere sekundäre Zugriffspfade (Zeiger).
   Addressierungsmöglichkeiten:
   - Physische Adressen: Schnell, aber Satz dann nicht mehr verschiebbar
   - Seitenbezogene Adressen: Zeiger=Paar(Seite, Positionszeiger auf Seitenindex):

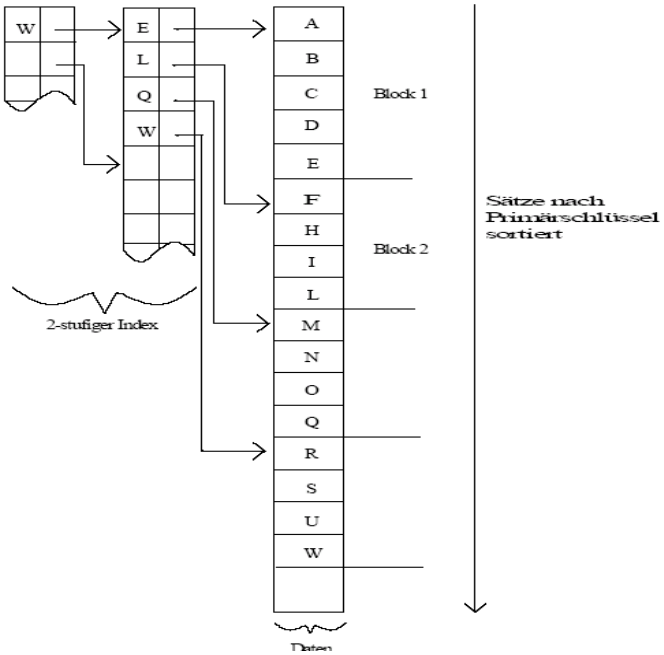


   Satz kann auch weitere seitenbezogene Adressen beinhalten.
   - Logische Adressen: durch Zuordnungstabelle:



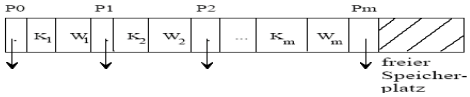
3. Festlegung von sekundären Zugriffspfaden (zu internen Sätzen)
   - durch Primärorganisation (Physische Abspeicherung der Sätze): zB durch Berechnung des Speicherplatzes aufgrund des Attributwerts und/oder Clusterung=Sätze, die häufig zusammen gelesen werden, nahe beieinander abspeichern.
   - durch sekundäre Zugriffspfade (zusätzliche Strukturen)

   a) Primärorganisation von Dateien:
   - Sequentielle Organisation: Aufeinanderfolgende Abspeicherung nach Primärschlüssel, Suchen gut (binär), Löschen und Einfügen aufwändig (Sätze verschieben)
   - Index-sequentielle Organisation: Sortierung nach Primärschlüssel, zusätzlich Index über Primärschlüssel: Dieser ist nicht dicht=enthält nicht alle Schlüssel s der Schlüsselmenge S der Ursprungsdatei. Kann zB aus Paaren (s_b, b) bestehen, wobei s_b der höchste Schlüssel in Seite b ist. Bei Suche dann Eintrag (s_b, b) suchen, für den gerade noch gilt: s<=s_b
     Für große Indexe zusätzliche Index-Stufe:



Einfügen: Um nicht großen Teil der Sätze bewegen zu müssen, Speicherreserve freihalten- bei Überlauf Zeiger auf Überlaufbereich: Entweder für jede übergelaufene Seite der Datei eigene Überlaufseite (schnell, schlechte Speicherplatzausnutzung) oder gemeinsamer Überlaufbereich für alle (umgekehrt).

- Hash-Verfahren: Idee: Speicheradresse für einen Satz aus dessen Primärschlüssel berechnen, häufig durch Divisions-Rest-Verfahren: h(s)=s mod n.
  Bei Kollosionen: Abspeicherung in separatem Überlaufbereich, im ersten freien Speicherplatz nach dem Platz h(s), erster freier Speicherplatz nach Zufallsberechnung oder zweite Hashfunktion verwenden.
- B-Bäume:
  Knoten des B-Baumes sind Speicherblöcke gleicher Länge, jeder Block kann bis zu 2k Sätze aufnehmen, jeder Satz besteht aus einem Schlüssel K und einem Nicht-Schlüsselteil W. Hat ein Knoten (nicht Blatt) m Schlüssel, so hat er m+1 Söhne:



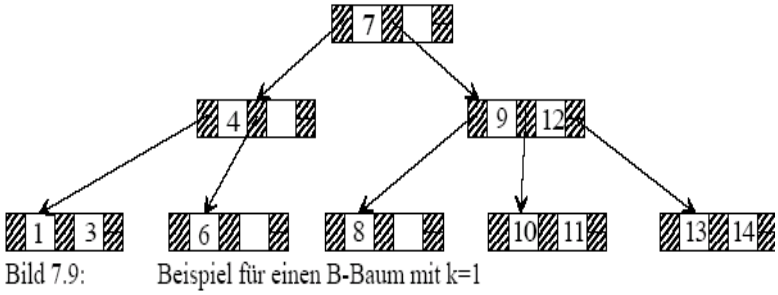
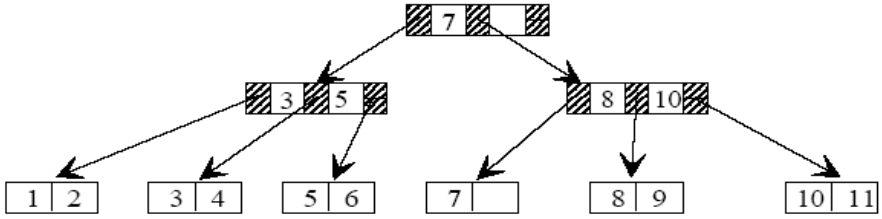


Bild 7.9: Beispiel für einen B-Baum mit k=1

Einfügen und Löschen-Invarianten:

- Alle Blätter haben selbe Höhe h
- Jeder Knoten ausser Wurzel hat mind. K und höchstens 2k Schlüssel.
- Wurzel hat entweder keinen Sohn oder mindestens 2 Söhne
- Jeder Knoten mit m Schlüsselnausser Blatt hat m+1 Söhne

- B*-Bäume: Unterscheidung B-Baum-Datei (eigentliche sortierte Sätze) und B*-Baum-Index.



- Verbindungen und Clusterung
  Miteinander in Beziehung stehende Entities clustern durch:
  Physikalischen Satz: 1:1oder 1:n physikalische zusammenhängend abspeichern.
  Allgemeinere Formen der Clusterung:
  Regal (Regal-Typ, <weitere Felder zur Beschreibung des Regals>)
  Seitenwand (Regal-Typ, Seitenwand-Typ, Höhe, Tiefe)
  Regalbrett (Regal-Typ, Regalbrett-Typ, Breite, Tiefe, Anzahl)
  ->Sätze aller betroffenen Dateien nicht mehr getrennt in verschiedenen Dateien abspeichern, sondern in einer Menge von Seiten sortiert nach Kriterium Regal-Typ!

b) Sekundäre Zugriffspfade:

Problem: Abfragen häufig nicht über Primärschlüssel.

2 Lösungen:

- Trennung der Zugriffspfade von den Sätzen (Invertierung)
  Sekundärindex=Datei mit variabler Satzlänge: WERT (SATZID)*
  Existiert für eine Attributkombination A ein Index, so heißt die Datei invertiert bezüglich A:

INDEX "Abteilung"

ABT-NR	PrimärSchlüssel		
ABT1	ANG4		
ABT2	ANG1		
ABT3	ANG2	ANG5	ANG7

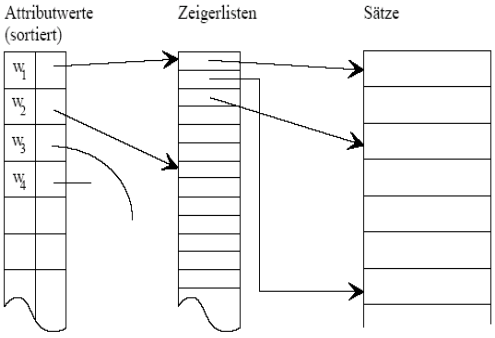
Angestellten-Sätze

ANG-NR	ABT-NR	ORT	#KINDER
ANG7	ABT3	KA	2
ANG4	ABT1	M	1
ANG5	ABT3	B	2
ANG1	ABT2	KA	0
ANG2	ABT3	M	2

INDEX "Ort"

ORT	Primärschlüssel	
B	ANG5	
M	ANG2	ANG4
KA	ANG1	ANG7

Wesentliche Eigenschafte: Zugriffspfad, den ein Index realisiert, ist völlig getrennt von den Sätzen selbst!
Hauptnachteil: Großer Speicherbedarf und Aufwand für Verwaltung der Indexee.
Realisierung zB als Indextabellen=Einträge der Form (Attributwert, Zeigerliste).
Besser: Zeigerlisten und Attributwerte getrennt führen:

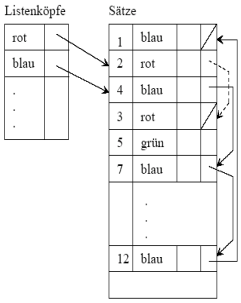


oder Organisation des Index als B-Baum: Nichtschlüsselteil enthält eine Zeigerliste auf die Sätze der Datei, in denen der zugehörige Schlüsselwert als Attributwert vorkommt. Geeignet für große

Indexe, die nicht vollständig im Arbeitsspeicher gehalten werden können.
Auch Indexe als Bitlisten möglich, nur für Gleichheitsabfragen und wenn Zahl der möglichen Werte eines Attributs relativ klein ist:

	Bit 1	2	3		#S
w_1	1	0	1	...	0
w_2	0	0	0	...	1
w_3	...				

- Einbettung der Zugriffspfade in die Sätze (Multilist-Strukturen)
  Sätze, die Auswahlkriterium erfüllen (zB. alle Blauen) werden durch Zeiger zu Liste zusammengefasst (Einlagerung in Primärdaten!). Einfach zu programmieren, aber Suchzeit für lange Listen hoch und ungeschickt über Speicherblöcke verteilt sind. Aufwändig: Komplexe Anfragen, da Sätze selbst berührt werden müssen (bei Invertierung Aufgabe häufig durch Auswertung der Indexe lösbar). Verbesserungen: Rückwärtsverkettung, Ring.



- Verbindung zwischen Sätzen:
  Um Joins zu unterstützen, sind am besten Invertierungen geeignet- ist zB die Gleichheit in Attribut A bei Satz 1 mit Attribut B in Satz 2 gefordert, so findet man die Schnittmenge schnell über die Invertierungen bzgl A und B.

c) Mechanismen in relationalen Systemen
- Primärorganisation der Daten.
  Grundsätzlich:
  - unsortierte Abspeicherung von einzelnen Tupeln (Heap)
  - sortierte Abspeicherung im Zusammenhang mit einem Primärzugriffspfad (B-Baum)
    In Oracle bzw durch ORGANIZATION INDEX nach CREATE TABLE statement.
    Cluster in Oracle enthält Menge von Tabellen, die ein Attribut gemeinsam haben.
    Organisation auf 2 Arten:
    - Bei Indexcluster wird jeder Wert des Clusterattributs nur einmal pro Seite gespeichert, alle Tupel mit demselben Clusterattributwert benachbart.
    - Bei Hash-Cluster werden in einer Seite Tupel abgelegt, bei denen die Anwendung der Hash-Funktion auf denselben Wert führt (Buckets).
- Sekundär-Indexe
  Durch CREATE INDEX .. ON ...
- Entwurf des internen Modells der DB
  Je mehr Zugriffspfade, desto teurer Ändern, Einfügen und Löschen.
  Daher verherige Analysen:
  1. Analyse der Datenmenge und ihrer Änderungstendenzen (zB Fluktuation etwa Aufträge)
  2. Analyse der Benutzung der Daten: Häufigkeit der Anfragen, min. Antwortzeiten

  Nach Möglichkeit Join durch Sekundärzugriffspfade unterstützen. Häufig Clusterung durch Denormalisierung: Vorwegnahme der Joins durch redundante Daten.
  Vorteilhaft auch Splittung der Tabelle eines Entitytyps horizontal oder vertikal und speichern auf unterschiedliche physische Datenträger. Viele DBMS bieten aber Möglichkeit der Partitionierung auf interner Ebene->Anwendungsprogramme nicht betroffen zB verteilte Datenbank.
  Generell: Invertierung einer Tabelle nach Attribut A sinnvoll, wenn Selektivität von A hoch ist=

Zahl der zu lesenden Tupel von A ist sehr viel kleiner als die Anzahl der Tupel der Tabelle. Invertierung lohnt dann, wenn durchschnittliche Zahl der qualifizierten Tupel in der DB kleiner als 10% der Gesamtzahl der Tupel ist!

8. Datenschutz

8.1 Datenschutzproblem

Zwei Ebenen:

- Juristische Ebene-Legislative legt fest, welche Daten schutzbedürftig sind.
- Technische Ebene=Datensicherheit

Von außen nach innen: Legislative Maßnahmen-Organisatorische Maßnahmen-Identitätskontrolle-Zugriffskontrolle-Kryptographie-DB

8.2 Identitätskontrolle

Identifikation durch: Informationsabfrage (Passwort), Prüfen von Erkennungsmerkmalen (Chipkarten..), physischer Merkmale (Fingerabdruck..).

8.3 Zugriffskontrolle

Zugriffskontrollen regeln das Lesen, Verändern, Einfügen, Löschen von Daten aus der DB.
Voraussetzung: BS muss Schutzmaßnahmen enthalten zB strenge Trennung der Benutzerprozesse, Schutz der DB vor Benutzern, die versuchen, das DB zu umgehen und direkt auf die DB zuzugreifen.
Berechtigungsmatrix: Für jeden Benutzer und jedes Objekt individuelle Zugriffsberechtigungen (R RW N). Auch wertabhängige Zugriffsbedingungen möglich: zB R: GEHALT < 3000 – kostspieliger.
Realisierung durch Sichten (CREATE VIEW Sicht WHERE GEHALT < 3000) und Abfragemodifikation (für Benutzer B zB DENY (NAME,GEHALT) WHERE GEHALT > 2500).
Sichten sind wirkungsvolles Mittel für den Datenschutz.

8.4 Kryptographische Methoden

Da das Datenbanksystem umgangen werden kann (Diebstahl), sollten die Daten zusätzlich verschlüsselt werden. Der Verschlüsselungsalgorithmus muss nicht geheim sein, wohl aber der Schlüssel!
Schlüsselverwaltung im DBS:

- Schlüssel unzugänglich halten
- Schlüssel nicht im System halten (jedesmal Eingabe)

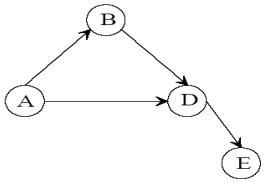
Kryptographische Methoden auch nötig zum Schutz der Datenübertragung (Lauschangriff).
Verfahren:

- Symmetrisch (Gleicher Schlüssel für Ver- und Entschlüsselung)
- Assymetrisch (Öffentlicher Schlüssel (=“Telefonnummer“) zum Verschlüsseln einer Nachricht AN den Benutzer, dort entschlüsselt dieser mit privatem Schlüssel). Es gilt: (T^O)^P=T. Bekannte Verfahren: RSA, PGP,SSL

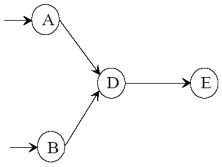
8.5 Zugriffskontrolle in relationalen Systemen

Verschiedene Zugriffskontrollen in den kommerziellen Systemen:

- DB2: Jeder Benutzer ist Eigentümer von privaten Relationen mit voller Verfügungsgewalt über diese (keine zentrale Instanz wie Administrator). Dieser kann an andere Benutzer explizit Zugriffsrechte einräumen:
  GRANT {ALL/privileges} ON relation TO {user[{ ,user}..]/PUBLIC WITH GRANT OPTION
  privileges: SELECT (Recht auf R), INSERT , DELETE, UPDATE (Recht auf W).
  Rücknahme der Rechte mit REVOKE.
  Da einem Benutzer für dieselbe Relation auch noch von anderen Benutzern Rechte eingeräumt werden können, ist es diffizil, welche Rechte denn nun bei REVOKE zurückgenommen werden.
  Lösung über Graph:
  Nimmt A seine Rechte für D zurück, so gilt auch das Recht über B nicht mehr, da B nicht unabhängiger Verleiher dieser Rechte ist:



Ist B aber unabhängig, so werden nur Rechte von A zurückgenommen, die B an D nicht gewährt hat:



Allgemeine Regel:
Besitzt D dasselbe Recht R von mehreren Benutzern B1, ..., Bn und wird R von Benutzer B1 zurückgezogen, so bleibt R für nur erhalten, wenn nicht alle Wege, auf denen R vom Besitzer der Sicht zu D gelangt sein kann, über B1 führen.

Noch komplizierter ist die Rechte-Rücknahme an E. Da im Bild A an D Rechte zum Zeitpunkt 50 gewährt hat, D an E jedoch zum Zeitpunkt 40, wird das Recht an E zurückgenommen, wenn B seine Rechte revokt (denn dann muss das Recht an E von B kommen). Ist das GRANT von D an E aber zum Zeitpunkt 60 gewährt worden, wird es nicht zurückgenommen, denn dann könnte es auch von A kommen!

9. Weitere Entwicklungen

9.1 Objektorientierte Datenbanken

Erlauben natürliche Darstellung von komplexen Objektstrukturen.

Definition: *Ein objektorientiertes Datenbanksystem ist ein Datenbanksystem, das die Kernkonzepte objektorientierter Programmiersprachen integriert.*

Ein OODBS muss neben einem objektorientierten Datenmodell alle wesentlichen Grundfunktionen eines DBS beinhalten:

- Datenintegration und -unabhängigkeit
- Transaktionsmanagement und Synchronisation
- Recovery
- Integritätskontrolle
- ad-hoc-Abfragesprachen
- Datenschutz

Datendefinition

Muß Spezifikation des Schemas durch Definition der Klassen ermöglichen:

```
make-class Classname
    :superclasses  ListofSuperclasses
    :attributes    ListofAttributes
    :methods       ListOfMethodSpecs
```

Datenmanipulation

DML (Datenmanipulationssprache) muss die Erzeugung, das Verändern, Löschen und Suche nach Objekten erlauben.

Erzeugen:

```
make Classname
        :Attribut1 Wert1
        :...
        Attribut2 Wert2
```

Abfragesprache: Schwierig, deskriptive Sprachen zu entwickeln, die der Mächtigkeit des Datenmodells gerecht werden. Grundform meist:

```
select Class QueryExpression
```

Pfad von Objekt zu Objekt mittels Punknotation=JOIN

```
zB. Select Fahrzeug (Hersteller.Name="Ford").
```

Datenbanken II

Vorteile objektorientierter DBS

- Modellierung: Erlauben sehr viel mehr von der Semantik der Anwendungswelt im DBS zu erfassen (komplexe Objekte, Klassenhierarchie, Vererbung...)
- Kontrolle über das Verhalten von Objekten: durch vordefinierte Methoden
- Erweiterbarkeit: durch Verkapselung von Objekten können neue Datentypen definiert werden, die wiederum zur Definition neuer Typen benutzt werden können.
- Abbau des impendance mismatch: Großer Abstand zwischen Programmiersprachen der Anwendungswelt und den Datenmanipulationssprachen der DBS (SQL) hinsichtlich der Datenstrukturen, zB. müssen C-Strukturen auf die sehr primitiven Strukturen des DBMS (Tupel) abgebildet werden. Da OODB die Kernkonzepte des objektorientierten Paradigmas unterstützen, wird der abstand zwischen einer objektorientierten Anwendungswelt und der Datenbank sehr viel kleiner.

9.2 **Erweiterungen des Relationenmodells**

Zum einen durch mächtigere Relationenmodelle und zum anderen durch Konzepte aus der Objektorientierung in die relationalen Modelle.

1. NF²-Relationen:
   Durch 1.NF (atomare Attribute) unnatürliche Relationenschemata.
   Idee: Relationen schachteln (Attributwerte nun auch Mengen von Tupeln, also Relationen).

LIEFERANT

LNR	LNAME	LIEF_LAGER	
		STADT	LIEF_ARTIKEL
			ANR | EK_PREIS

FILIALE

FNR	STADT	SORTIMENT			BESTELLER
		ANR	VPREIS	BESTAND	ANGNR

Beispiel einer Ausprägung von FILIALE

1	Hamburg	1 60 1000 2 65 2000 3 40 400 4 70 0 5 90 0 6 80 900	123 156 567
2	Hagen	1 55 1000 4 65 1000 5 85 1000 6 75 900 7 40 0	234 345
3	Stuttgart	2 65 2000 3 50 300 6 85 3000 7 45 400	456 274 891

Selektionen

Auch **geschachtelte** Selektionen zB Suche Filiale die ihr gesamtes Sortiment vorrätig haben:
FILIALE select((SORTIMENT select (Bestand=0)) = {}))

Projektion

Erweiterung nötig, da Ergebnisrelation nur eine Teilmenge ihrer Attribute besitzt:

- Projektionen auf inneren Relationen
  Schachtelung: FILIALE projekt_to (FNR, SORTIMENT project_to (ANR))
  =Gib zu jeder Filiale die Artikel im Sortiment aus.

FNR	SORTIMENT
	ANR

- Selektion auf inneren Relationen
  Zur gezielten Eliminierung von inneren Tupeln ist Selektion nötig:
  FILIALE project_to (FNR, SORTIMENT select (Bestand=0) project_to (ANR))
  Die unterhalb der Projektion angeordnete Selektion hat keinen Einfluß, welche FILIALE-Tupel ausgegeben werden, nur welche SORTIMENT-Tupel!

1	4 5
2	7
3	

Nest,Unnest

Durch Nest werden 1-NF Relationen in genestete Relationen umgewandelt:

```
R1 nest (X = [C,D])
```

R1:

A	B	C	D
x	2	5	a
x	2	6	b
y	7	11	c
y	7	12	d

R2

A	B	X	
		C	D
x	2	5 6	a b
y	7	11 12	c d

2. Objektrelationale Datenbanken

   Sind Erweiterungen, um objektorientierte Konzepte in relationale Systeme einzuführen. Sie basieren auf Überlegungen zu nicht-normalisierten Relationen- erlauben Definition von komplexeren Attributen und haben Spracherweiterungen und Objekttypen, die selbst definiert werden können:

   create type SORTIMENTS_EINTRAG
   (ANR: integer ....)
   create type FILIALE
   (FNR: integer
   SORTIMENT: **setOf(SORTIMENTS_EINTRAG)**
   member function LAGERWERT return integer ->selbst definiert zB. SQL/ C

   Benutzung durch: create table FILIALE_TABELLE of type FILIALE

   Beziehungen zwischen Objekten durch Referenzierung:

   create type FILIALE1
   (F_LEITER: ref ANGESTELLTER,...)

   Abfrage durch Pfadausdrücke: select F.F_LEITER.NAME from FILIALE F;

   Vererbung: Entweder Definition des Objekttyps als Subtyp oder auch Vererbung von Tabellen:

   create table PERSON_TABELLE (...)
   create table ANGESTELLTEN_TABELLE
   inherits-from PERSON_TABELLE
   (ANR: .....)

   Objektrelationale Systeme sind deutlich sicherer als objektorientierte, da sie weitgehend mit deskriptiven Mechanismen zur Datenmanipulation arbeiten – keine benutzerdefinierten Programmstücke zur Manipulation.

9.3 **Multimedia-Datenbanken**

Derzeit Speicherung von Multimedia-Daten in separaten Dateien außerhalb der DB, DB verwaltet Zeiger und Zusatzattribute wie Ursprung und textuelle Inhaltsbeschreibung.

Speicherung in DB nötig, wenn große Mengen zu verwalten sind wegen Transaktionen, Abfragen, Indizierung und Gefahr von Inkonsistenzen.

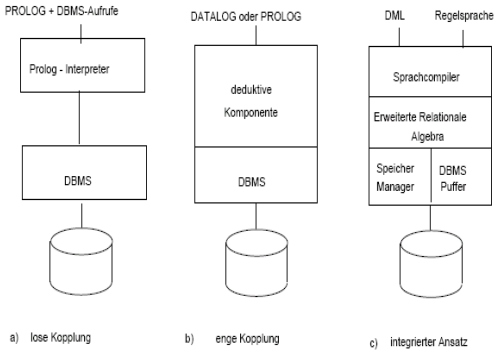
Deshalb folgende Anforderungen an die DB:

- BLOBS (Binary Large Objects) beliebiger Größe müssen speicherbar sein.
- Abfragen nach Ähnlichkeiten zwischen Inhalten von Multimedia-Objekten müssen möglich sein, zB Datenbank mit Fingerabdrücken
- Video und Audio müssen als Datenströme mit stabiler Datenrate ausgelesen werden, oft auch Audio und Video-abspielung synchron.

Abspeicherung von Multimedia-Daten meist in komprimierter Form.
9.4 Deduktive Datenbanken
Wissen wird in Form von Regeln ausgedrückt: if..then..
Die Tupel der zugrundeliegenden konventionellen Datenbank stellen die Fakten dar (extensionale DB).
Die Regeln definieren die intensionale DB.

if DIREKT_VORGESETZT(X,Y) then vorgesetzter (X,Y)
if DIREKT_VORGESETZT(X,Z) and vorgesetzter (Z,Y) then vorgesetzter (X,Y).

Zwei Schwerpunkte der Implementierung deduktiver DB:
Entwicklung einer adäquaten Regelsprache und effiziente Berechnung von Abfragen.
Verwendung einer logischen Programmiersprache, bekannt: DATALOG (Variante von Prolog).
Abfragen eingeteilt in 2 Klassen: bottom-up und top-down (rückwärtsverkettet) -Auswertung.
Abfrageoptimierung: Unterscheidung zwischen interpretativen (werden zur Laufzeit angewendet) und compilierten Methoden.
Architektur: Unterscheidung lose Kopplung (Erweiterung der logischen Programmiersprache um Prädikate, die das DBMS aufrufen), enge Kopplung (Regelinterpreter setzt auf relationaler DB auf und lässt jede DB-Relation als normales vordefiniertes Prädikat innerhalb eines Regelprogramms erscheinen) und integrierter Ansatz (weist innerhalb des DBMS eine Regelsprache und ein Inferenzsystem auf, um Anfrage zu beantworten und Updates zu unterstützen).



9.5 Verteilte Datenbanken

Eine verteilte Datenbank ist eine DB, deren Datenbestand nicht zentral an einem Ort physikalisch gespeichert ist. Lokale Datenbestände und Transaktionen kontrolliert das autonome DBMS. Nur bei globalen Transaktionen kooperieren die lokalen DBMS.
Orts- und Replikationstransparenz: Geographische Verteilung und redundante Datenhaltung bleiben dem Benutzer verborgen.
Unterscheidung strikt homogenes (überall DBMS eines Herstellers), homogenes (Hersteller verschieden, jedoch gleiches Datenmodell) und heterogenes VDBS (verschiedene Datenmodelle).
Client-Server-Systeme: Zentrale Daten liegen auf einem oder mehreren Servern, benutzerorientierte Prozesse laufen auf Clients.

9.6 Web-Datenbanken

CGI-Programme (Schnittstelle zwischen Informationsdienst und dem Web- schickt http-Anfragen an Informationsdienst und umgekehrt) binden Datenbanken an das www an.
Beispiele für DB im Netz:

- Zentrale Speicherung von Web-Dokumenten, dadurch keine Update-Probleme.
- Dynamischer Zuschnitt von Web-Dokumenten auf Interessen der Benutzer.
- Internetshops: Koppelung von Online-Kundendaten und -Bestellungen direkt an die Datenhaltung.
- Suchmaschinen
- Kombination von Wissen mit multimedialen Inhalten.

9.7 Entscheidungsunterstützende Systeme

Unmengen von Daten müssen ausgewertet werden, zB für Zielgruppen von Werbemaßnahmen.
Analyse der gesammelten Daten:
Abfrageerweiterung der DB um Aggregation und Statistik.
Für Analyse von Daten werden häufig Histogramme verwendet. Ein Histogramm teilt den Wertebereich

eines Attributes in bestimmte Bereiche auf und berechnet eine gewünschte Aggregierung, zB Summe, für die Datensätze jedes Bereichs einzeln. Histogramm=Gruppierung:

	Rot	Grün	Metallic	**Summe**
Benzin	4	12	7	**23**
Diesel	7	10	13	**30**
Summe	**11**	**22**	**20**	**53**

Kreuztabellenerzeugung nicht durch einzelne SQL-Abfragen möglich->Einführung eines speziellen Wertsymbols **alle** zur Repräsentation von Teilsummen. **Alle** kann als Platzhalter für die enge aller Werte des Attributs verstanden werden.
Die Zusammenfassung von Daten durch Aggregation nennt man auch Rollup, in SQL:
SELECT ... FROM ... GROUP BY ... WITH CUBE
bewirkt WITH CUBE, dass ein Rollup für jede Kombination der beiden Gruppierungsattribute durchgeführt wird.

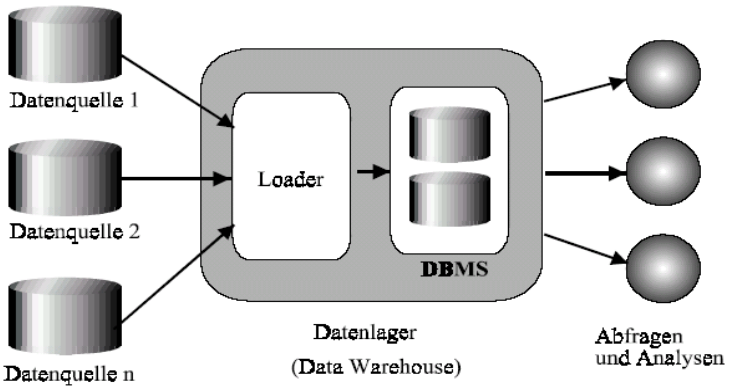
Motor	Farbe	Anzahl
Benzin	Rot	4
Benzin	Grün	12
Benzin	Metallic	7
Benzin	**Alle**	23
Diesel	Rot	7
Diesel	Grün	10
Diesel	Metallic	13
Diesel	**Alle**	30
Alle	Rot	11
Alle	Grün	22
Alle	Metallic	20
Alle	**Alle**	53

Data Mining
=Schürfen nach Daten. Herausholen von Daten unter einer bestimmten Fragestellung.
Unterschied zu deduktiven Datenbanken: Regeln erzeugen keine Daten, sondern aus Daten werden Regeln abgeleitet.
Meist nur eine Variable pro Regel. Jede Regel ist mit zwei Gewichtungen versehen, die deren Bedeutung und Zuverlässigkeit bewerten:

- **Unterstützung** einer Regel=Anteil der Population, der sowohl die Bedingung als auch die Konsequenz der Regel erfüllt. Je kleiner, desto unbedeutender die Regel. >50% bedeutsam.
- **Zuverlässigkeit**=Anteil der Population mit zutreffender Konsequenz an der Population mit zutreffender Bedingung. >80%-> Regel trifft in 80% der Fälle zu. Ca. 50%=Zufall.

Klassifikationsregeln unterteilen einen Datenbestand in disjunkte Teilmengen (etwa Kreditwürdigkeit), **Assoziationsregeln** repräsentieren Wissen über Zusammenhänge (etwa Kaufverhalten von Kunden).

Data Warehousing
'Größere Unternehmen/historisch gewachsene EDV: Filialen haben meist eigene Datenbestände, Verteilung der Daten auf verschiedene Schemata und Plattformen, Archivierung alter Daten auf Band.
Data Warehouse speichert die gesammelten Informationen typischerweise für einen langen Zeitraum (Archiv). Normale Datenbanken werden durch Data-Mining-Abfragen nicht belastet!



Wann werden Daten gesammelt? Quellensteuerung (kontuinierliche Weiterleitung an DWH) oder Sammeln und paketweise Übermittlung.
Welches Schema? Integration aller Quellenschemata und entsprechende Konvertierung durch Loader.

Datenbanken II

Daten im DWH also nicht einfache Kopien, sondern gespeicherte Sicht über alle Quelldaten.
Wie wirken sich Änderungen der Quellschemata auf das DWH aus? Änderungen müssen auch im DWH in geeigneter Form nachvollzogen werden. Evtl Neubildung der Sichten.
Welche Daten sollen in welcher Form aggregiert werden? Rohdaten zu umfangreich. Einzeldaten evtl uninteressant. Durch Sichten und Abfrageoptimierung können Relationen transparent durch aggregierte Versionen ersetzt werden.