DAB2 Zusammenfassung AS

3-Ebenen-Architektur: ① Externes Schema (Benutzer/Anwendungsschichten) ↔ ② Konzeptionelles Schema (Logische Gesamtsicht) ↔ ③ Interner Sch. (Interne Ebene) (Batch Processing)
Transformationen

Stored **P**rocedure: explizit → Durch Benutzer/Anwender | Transaktionskontrolle
vs → Aufruf ←
Trigger: implizit → Durch DBMS in Abhängigkeit von Datenänderungen

- Kapselung von "business rules"
- Optimierung von Abfragen, Reduktion des Netzwerkverkehrs
- erhöhte Sicherheit benötigt

→ Einsatzbereiche ←

- Konsistenzsicherung
- Logging
- Nachführen von Tabellen
- Referentielle Integrität

→ Probleme ← SP: Fehlerbehandlung Trigger: • Komplexität zu testen • Aufrufreihenfolge nicht deterministisch

DDL: CREATE / ALTER / DROP {TRIGGER | PROCEDURE | FUNCTION} Test...

②	③	Dateisystem
Relationen	Log. Dateien	Phys. Dateien
Tupel	Datensätze	Seiten/Blöcke
Attributwerte	Felder	Bytes

SQL-Datentypen für sehr große, unstrukturierte Datensätze:
- Binary Large Objects (BLOBs): Byte-Folgen wie Bilder, Audio- und Videos
- Character Large Objects (CLOBs): Folgen von Zeichen

Ausführungsplan: Varianten → OPTIMIERUNG ←

Beim Aufbau des Ausführungsplans bestehen verschiedene Freiheitsgrade:

- Reihenfolge und Gruppierungen von assoziativen und kommutativen Operatoren:
  - Joins, Vereinigung, Schnittmenge ...
- Wahl eines Algorithmus für jeden Operator
- Zusätzliche Operatoren einbauen, die im logischen Plan nicht auftauchen:
  - Scan
  - Sort
- Modus des "Datentransports" zwischen Operatoren festlegen:
  - Temporäre Tabelle, Pipeline
- Dafür nötig: Aufwand-/Kostenabschätzungen

DAB 2 Zusammenfassung:

Trigger:
- Code der vom DBMS ausgeführt wird
- Ausführung auf Grund von sich ändernden Datenbankzuständen
- geeignet für
  - Realisierung von Integritätsbedingungen
  - Berechnungen (z.B. nachführen von Summen)
  - Protokollieren von Datenänderungen

Prinzip: ECA (Event – Condition – Action)

\+ Entlastung der Anwenderprogramme ⊖ Fehlerbehandlung
\+ effiziente Ausführung möglich ⊖ Testen, Debuggen
\+ einfacher Rollout ⊖ Ergebnis event. abhängig von Aufruf-Reihenfolge ⊖ Terminierung von geschachtelten Triggern

SQL: CREATE / DROP / ALTER Trigger...
↳ Auslösung: BEFORE / AFTER / INSTEAD OF

SQL Example:

```
CREATE TRIGGER NewPODetail
  ON Purchasing.PurchaseOrderDetail      -- Table to watch
  AFTER INSERT                           -- condition
  AS
  IF @@ROWCOUNT = 1
  BEGIN
    UPDATE Purchasing.PurchaseOrderHeader
    SET SubTotal = SubTotal + LineTotal
    FROM inserted
    WHERE PurchaseOrderHeader.PurchaseOrderID =
          inserted.PurchaseOrderID
  END;
```

INSTEAD OF:
\+ Verhindern unerlaubte DML-Operationen
\+ SQL-constraints nicht ausreichend
\+ Nicht änderbare Sichten "änderbar" machen

AFTER:
\+ weitere Operationen auslösen
\+ Historisierung

	Operationen	(Objekte)
MOS:	SELECT... FROM... WHERE...	(Relationen, Sichten)

Deklarative DML/DQL auf Tabellen, Sichten, Tupel

SOS: FINDNEXT satz STORE satz (Externe Sätze, Indexstrukturen, Scans)
Externe Sätze, logische Dateien, logische Zugriffspfade
Navisierender Zugriff

ISS: LOOKUP im B-Baum INSERT in B-Baum (Interne Sätze, Bäume, Hashtabellen)
Interne Sätze, Zugriffspfade, Manipulation von Sätzen und Zugriffspfaden

SPS: Bereitstellen Seite i, Freigeben Seite i (Segmente, Seiten)
Seiten, Seitenadressen, Freigeben und Bereitstellen.

DS: Lies Block k, Schreibe Block k (Dateien, Blöcke)
Hole Seite, Schreibe Seite

GS: – (Zylinder, Spuren)
Spuren, Zylinder, Armbewegungen.

5-schichten Architektur:

Schnittstelle	Schicht	Komponenten
MOS · Mengenorientierte Schnittstelle →	Datensystem	Übersetzung, Zugriffspfadwahl, Zugriffskontrolle, Integritätsk.
SOS · satzorientierte Schnittstelle →	Zugriffssystem	DataDictionary, Currency, Pointer, Sortierung, Transaktionsverwalt.
ISS · interne Satz-Schnittstelle →	Speichersystem	RecordManager, Zugriffspfadverw., Sperrverw., Logging, Recovery
SPS · Systempuffer Schnittstelle →	Pufferverwalt.	Systempufferverw., Seitenersetzung, Seitenzuordnung
DS · Dateischnittstelle →	Betriebssystem	Externspeicherverwaltung, Speicherzuordnung
GS · Geräteschnittstelle →		

Externe Speicher

Speicherhierarchie

Tertiärspeicher in Zusammenhang mit Datenbanken:
⊖ Speichern der aktuellen Datenbankinhalte für laufenden Betrieb (anstatt Sekundärspeicher)
⊕ Datensicherung
⊕ Archivierung

1. Primärspeicher + sehr schnell ⊖ sehr teuer ⊖ Volatil (Inhalt nur zur Systemlaufzeit verfügbar)
2. Sekundärspeicher (HDD, SSD) ⊖ Faktor 10^5 - 10^6 langsamer (SSD 10^3) ⊕ Günstig ⊕ Persistent
3. Tertiärspeicher ⊖ Langsam ⊕ Billig ⊕ Hohe Kapazität ⊖ nicht Permanent zugreifbar ⊕ Persistent

Eigenschaften Ebene X

Speicher- und Zugriffszeit	:	$X < X+1$
Kosten pro Speicherplatz	:	$X > X+1$
Kapazität	:	$X < X+1$
Lebensdauer	:	$X < X+1$

Problem: Zugriffslücke zwischen Primärspeicher und Sekundärspeicher ca. 10^5
Caching kann helfen wenn 1) Zeitliche Lokalität (in kurzer Zeit auf gleiche Daten zugreifen) und Räumliche Lokalität (Data Cluster: Zusammen angefragte Daten sind auf dem Sekundärspeicher eng zusammen)

Data-Mart

- Langfristig gehaltener Datenbestand innerhalb eins DWH (Kopie vom Teil)
  ↳ Geschaffen für bestimmte Anwendung/Organisationsbereich
- Viele Architekturvarianten, übergänge oft nicht scharf.
- "Analysedatenbanken"
  ↳ können auch Excel-File sein

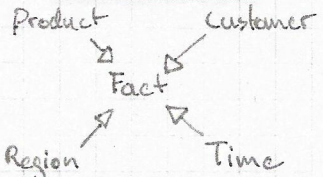
Drill-Down

- Hierachie von oben nach unten traversieren
- "Verfeinerung" entlang einer Dimension

Roll-Up

- Hierachie von unten nach oben traversieren
- "Aggregation" entlang einer Dimension

Sternschema



Faktentabelle: Enthält Fakten und Fremdschlüssel auf Dimensionstabellen

Dimensionstabelle: Enthält Dimension und Primärschlüssel, die mit Faktentabelle "verknüpft" ist

No-SQL (Not only)

- nicht relational
- schemenfrei
- einfache API
- Auf horizontale skalierung ausgerichtet
- einfache Datenreplikation
- Konsistenzmodell: BASE und Eventual Consistency

Core NoSQL:

- Key/Value Stores
- Wide Column Stores
- Document Stores
- Graphdatenbank

Soft NoSQL:

- XML-DB
- Obj. DB
- Grid/Cloud DBAnw.

Multidimensionalität

- Unterscheidung von Fakten (abgemessene Werte) und Dimensionen (Raum, Zeit, Organisation...)
- Hierarische Struktur von Daten.

Fakten (Measures):

- Eigentliche Obj. der Analyse
- Ansammlung von Einzelmessungen/Kennzahlen, verknüpft mit Kontext (Dimension)
- Jedes Faktum stellt einen Businessvorfall, eine Transaktion oder ein Ereignis dar.
- i.d.R. numerische Werte (Verkaufszahlen, Lagerbestand, Umsatz, Gewinn etc.)
- Analyse der Fakten soll es dem Agent ermöglichen die Unternehmensleistung zu überwachen/steuern
- Numerische Werte lassen sich auf verschiedene arten verdichten/aggregieren.

Dimensionen:

- Kontext der Daten / Fakten
- Eindeutige Strukturierung des Datenraums
- Hoffentlich orthogonal (unabhängig)
- Verfügen über Attr. welche geordnet werden können
- Anzahl nicht beschränkt
- 'Zeit' ist häufig

"Non-Standard DB-Systeme" (alle die nicht Stand.)

- Quasi Echtzeit
- Riesiges Datenvolumen (Peta+)
- Lange Transa. (z.B. Konstruktionsbereich)
- Abbildung obj.-orientierter Konstrukte (z.B Vererbung, Polymorphie) und speicherung von Objekten

No-SQL: CAP

Consistency: Alle Knoten sehen zu jeder Zeit die selben Daten. ACID hat nur innere Konsistenz

Availability: * Ausgefallene Knoten beeinflussen die Verfügbarkeit anderer nicht

Partition Tolerance: System arbeitet trotz Verlust von Nachrichten, einzelner Knoten/Partitionen von Netz weiter

Anwendung:

- Klassische vert. Systeme ist 'C' am wichtigsten → C.O.A eingeschränkt
- Web Anwendungen können Fokus auf A + P, dabei Einschränkung von 'C'

Bsp. DUS → AP, RDBMS → CA
Geldautomat → CP

Eventual Consistency (BASE):

- Konsistenz erst nach definiertem Zeitraum erreicht, nicht nach jeder Transa.

Big Data (5-V)

Grosses Datenvolumen

Volume: Tera- bis Petabyte

Velocity: Hohe Stream/Verarbeitungsgeschwindigkeit • Viele Datensätze pro Tag/Stunde/Minute

Variety: Unterschiedliche Datenquellen (Text, Bilder, videos, Blogs etc.)

Veracity: Schwankende Datenqualität

Value: Wert der Daten

Herausforderung: Erfassung, Speicherung, Auswertung und Visualisierung von Daten

"Standard-DB-Systeme" Anforderungen

- DBanwendungen kommen aus dem Betriebswirtschaftlichen Umfeld.
- Transaktionssicherheit
- Hohe Performance bei Search/Update
- Verwaltung 'einfacher', stark strukturierter Daten
- i.d.R. zentraler DB-Server

→ **Relationelle DB-Systeme**
↳ Grenzen: Fokus auf strukturierte Daten • Begrenzte Skalierbarkeit • Schemaänderungen sind komplex

Skalierung

Vertikal: Leistungssteigerung durch (Scale up) hinzufügen von Recourcen (z.B: + CPU, + RAM, + HDD, etc.)
- Unabhängig von der Implementierung möglich. Begrenzt durch Stand der Technik und Kosten
- ⊕ Transparent für DBMS
- ⊕ Admin. aufwand ist konstant.
- ⊖ Hardwarekosten
- ⊖ Skalierung nur in grösseren Stufen möglich
  ↳ höhere Kosten für ungenutzte Leistung

Horizontal: Hinzufügen von (Scale out) Rechnern/Knoten.
- System muss für parallelisierbarkeit (Anwendung) ausgerichtet sein
- ⊕ Kostengünstig
- ⊕ Skalierung in kleinen Stufen
- ⊖ last/Datenverteilung notwendig
- ⊖ ggf. verteilte Protokolle
- ⊖ Höhere Fehlerrate
- ⊖ Höherer Administrationsaufwand

* Alle Anfragen an das System, werden stets beantwortet