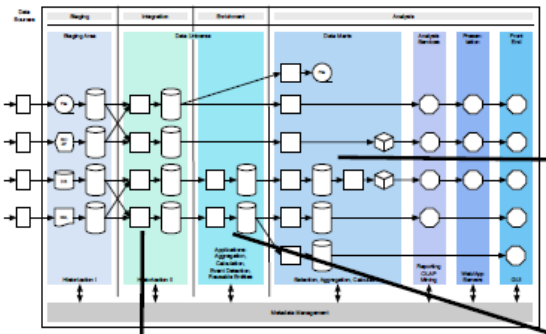
IE2

DWH 01

- DWH Architektur: staging -> integration -> enrichment -> analysis
- staging = zusammenführen unterschiedlicher Datenquellen mit verschiedenen Formaten, Qualitätscheck (Format, Vollständigkeit)
- integration = einheitliches Datenmodell aus staging erstellen
- analysis = data marts, denormalisiert, star-schema, optimiert für Abfragen
- Auswertungen basieren auf den Data Marts



Analytische Schicht:
- End User Interface für Fachseite
- Aggregationen nach Business-Sicht (performance-optimierte Strukturen) in Data Marts
- Modellierung: relational normalisiert, Sternschemata, flache Strukturen
- Zugriffsschicht für Reporting Applikationen (Standard und ad hoc)

Integrationsschicht:
- Enterprise Data Warehouse Modell
- Zentrale Plattform für Informationsversorgung (single version of truth)
- Themen- und Businessorientierte Interpretation der Daten für analytische Nutzung
- Normalisierte Modellierung
- Basis für die analytische Schicht

Anreicherungsschicht:
- Zentrale Erzeugung wiederverwendbarer Daten. Beispiele:
  - Kunden, Verträge, ...
  - Berechnete KPIs (Key Performance Indicators)
  - Klassifikationen
  - Segmente
  - Simulationsdaten
  - Conformed Dimensions

- OLTP = Online Transaction Processing; viele kleine Transaktionen, updates + inserts, normalisiertes Schema, konsistente Zugriff auf aktuelle Daten; z.B. Buchungssysteme, ATMs, ERP; Ziel: so viele Transaktionen wie möglich verarbeiten

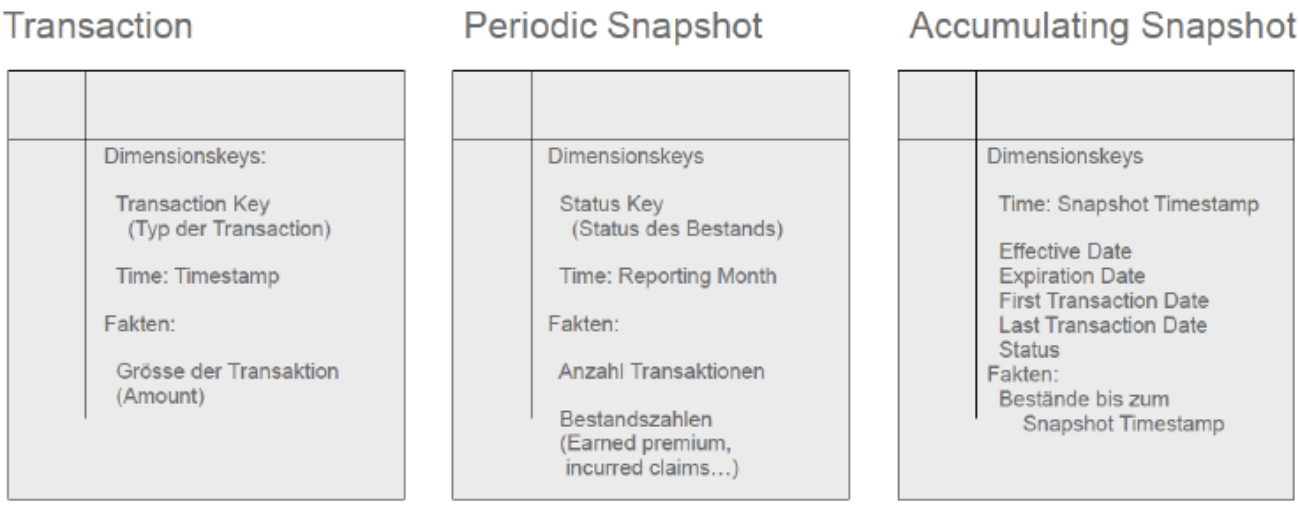
- OLAP = Online Analytical Processing; grosse Queries mit vielen Joins, keine Updates; Redundanz in Daten (materialized views, spezielle Indizes, denormalisiert), periodisches Neuladen; z.B. MIS, Wetter, wissenschaftliche DBs; Ziel: Queries mit sehr niedrigen Antwortzeiten

DWH 02

- Sternschema: Faktentabelle = Fremdschlüssel der Dimensionstabellen; Dimensionstabellen = Masterdaten, Primärschlüssel zur Verknüpfung mit Faktentabelle
- Denormalisierung (non-3NF) durch Duplikate; dadurch höhere Abfragegeschwindigkeit
- Fakten: Einzelmessungen/Kennzahlen verknüpft mit Kontextdaten (-> Dimensionen); typischerweise wenige, meist numerisch Attribute, aber viele Datensätze; Partitionierung und Auslagerung alter Records nötig; sollten sich auf einen fachlichen Prozess beschränken; 1 Data Mart kann n Faktentabellen haben
- Kennzahlen: Aggregate über bestimmten Zeitraum
- Dimensionen: Ansammlung von Daten, die die Fakten von einer bestimmten Sicht aus beschreiben; beschreiben kontextuellen Hintegrund von Fakten und dienen der Klassifizierung von Fakten; nicht so gross wie Fakten, aber viele Attribute
- Drill-Down: mehr Details anzeigen, Hierarchie von oben nach unten (all->cat->subcat->single); Roll-Up: Gegenteil
- Slicing: ausschneiden von Scheiben aus einem Würfel; Dicing: Sub-Würfel ausschneiden
- Schneeflockenschema: Dimensionen sind weiter aufgespalten (normalisiert); verwendet normalisierte Dimensionen entlang der Hierarchien (langsam wegen Join); Zeit-/Platzbedarf wichtiger als Einfachheit von Schema/Abfrage; gut wenn Entitäten unterschiedliche Granularität haben
- Zeitdimension: Modellierung erlaubt Drill-Down und weitere Merkmale (als DATE) wie Feiertag, Firmenkalender, Fiskaljahr, Tag in Monat/Jahr etc.; Zeitraum oft 10-20 Jahre, ein Datensatz pro Tag; Roll-Up inkompatibel, da Y->Q->M->D und Y->W->D d.h. 2 Dimensionen
- Conformed Dimensions: "A conformed dimension is a dimension that has exactly the same meaning and content when
  being referred from different fact tables." Manche Dimensionen können von mehreren Fakten referenziert werden; Voraussetzung: Dimensionstabellen haben die notwendige Granularität
- Galaxy Schema: mehrere Subjekte abbildbar; Grundlage unternehmensweiter Anwendungen können «Galaxien» sein; Es existieren mehrere Faktentabellen und zugehörige Stern- bzw. Schneeflockenschemata (Inhaltliche Zusammenhänge, Unterschiedliche Aggregierungsstufen in Aggregattabellen); Dimensionen werden häufig von mehreren Faktentabellen verwendet
- Fakt = WHAT, Dimension = BY

DWH 03

- Fakten sind quantifiziert (BWL-Grössen) und aggregierbar (additiv: entlang aller Dimensionen / Produktionsdaten, Verkaufszahlen; semi-additiv: nur entlang ausgewählter Dimensionen / Kontostand, Anzahl Versicherungsverträge; nicht-additiv: Durschnittswerte, Prozentanteile, Extrema)
- Charakteristika von Fakten: oft nur eine Kennzahl, "Anzahl", Kontext durch unabhängige Dimensionen gegeben; Faktentabelle beschreibt eine atomare Aktion zu einem gegebenen Zeitpunkt
- 3 Modellierungstypen (oft 1/2 genutzt): Transaction (z.B. detailierte Verlaufsanalysen; unvorhersehbares, repetitives Verhalten); Periodic Snapshot (ein Record pro Kombination der relevanten Dimensionskeys; auch wenn keine Veränderungen; Kennzahlen, die schwer auf Basis von Transaktionen zu berechnen wären; z.B. Geschäftsbericht); Accumulating Snapshot (Bestand mit Verlauf, nur neuer Record, wenn es Änderungen gibt; oft keine feste Zeitperiode; eine Zeile pro Ereignis, evtl. mit NULL; z.B. Produkte mit endlichem Lebenszyklus)



- Dimensionen enthalten Referenzdaten in Sternschemata; • Referenzdaten werden in Hierarchien gruppiert; Gruppierungen über mehrere Hierarchieebenen gestatten (-> drill-down); oftmals Bäume; kann 1:1, 1:n (Mitarbeiter / Chef; da zirkulär, ist Abbruchkriterium notwendig), n:m (Produkte & Teile / bill of materials) sein
- Fixierung als Flachstruktur (1 Tabelle, für jede Ebene ein Attribut) oder Schneeflocke (1 Tabelle pro Ebene) kann sinnvoll sein, sodass keine Rekursion mehr nötig ist
- unbalancierte/unvsollständige Hierachien: ragged (NULL Werte für untere Ebene(n), pro: each parent has a direct child, con: keine korrekte Konzepthierarche) und skip-level (kann balanciert sein, skipped = NULL, korrekte Konzepthierarchien)
- Historisierung (Datenänderungen müssen nachvollziehbar, audit-fähig und erklärbar sein)
    - SCD Type 1 (overwrite): der ursprüngliche Attributwert wird überschrieben; keine Versionierung, as-is-reporting
    - SCD Type 2 (new record): bei Änderungen wird ein neuer Eintrag in der Dimensionstabelle generiert; volle Versionierung, as-was-reporting
        - Monotemporale Historisierung: gibt an, wann Ereignisse dem System bekannt sind (system time) ODER wann sie in der realen Zeit stattgefunden haben (business time, validity)
        - Bitemporale Historisierung: gibt an, wann Ereignisse dem System bekannt sind (system time) UND wann sie in der realen Zeit stattgefunden haben (business time, validity)
    - SCD Type 3 (current value field): der neue Wert verdrängt den alten; der ursprüngliche Wert wird in einer speziellen Spalte gespeichert; aufbewahren des Vorgängers (selten verwendet)

DWH 04

- DWH ist ein Hub mit Data Marts als Spokes; ETL sind pipe-and-filter; pub/sub macht Aktualisierungen für alle verfgübar ohne Polling (auch mit producer/consumer/broker verwendbar)
- Virtual DWH: mehrere Views auf operative DBs, nicht alle Aggregationen sind materialisiert; pro: quick win für erste Auswertung und tiefe Initialkosten; con: beschränkte Adressierung Datenqualität, keine Trenning OLAP/OLTP, keine/wenige historische Daten
- Data Marts: Teilmenge aller Informationen; sind independent (satellite; direkt aus operationellen Systemen gefüttert) oder dependent (integrated; Teilmenge des DWH oder eines anderen Data Marts); oft: Teilaspekt und schnell & einfach implementiert; direct feeding kann praktisch sein für Exploration oder einmalige Validierung
- Operational Data Store (ODS): DWH für OLTP; flüchtig, wenig historische Daten, gute Antwortzeiten; Complex Event Processing, Aktualisierung der Daten zwischen Applikationssilos
- Inmon: Enterprise DWH (3NF, Integriertes Datenmodell, Beinhaltet alle Unternehmensdaten), Data Marts
- Kimball: Kein integriertes Datenmodell (es werden nur Teile der Unternehmensdaten benötigt), Conformed Facts + Dimensions, Data Marts (Dimensionales Modell)

- Data Warehouse: beinhaltet alle Daten eines Unternehmens, sehr grosses Datenvolumen, ganze Datenhistorie, ganzheitliche Sicht, applikationsunabhängig, Modellierung in 3NF (klassische Datenbankmodellierung)
- Data Mart: beinhaltet nur fachspezifische Information, geringeres Datenvolumen, nur Teile der Datenhistorie, applikationsabhängig (z.B. Marketing, Finanzen, Entwicklung, ...), dimensionale Modellierung

Characteristic	Favors Kimball	Favors Inmon
nature of the organization's decision support requirements	tactical	strategic
data integration requirements	individual business areas	enterprise-wide integration
structure of data	business metrics, performance measures, and scorecards	non-metric data and for data that will be applied to meet multiple and varied information needs
scalability	need to adopt to highly volatile needs within a a limited scope	growing scope and changing requirement are critical
persistency of data	source system are relatively stable	high rate of change from source systems
staffing and skills requirements	small teams of generalists	larger team(s) of specialists
time to delivery	need for the first DWH application is urgent	organization's requirements allow for longer start-up time
cost to deploy	lower start-up costs, with each subsequent project costing about the same	higher start-up costs, with lower subsequent project development costs

- Linstedt: hub-and-spoke; Integration von (Sub-)Dimensionen via Business Keys; Hybrid aus dimensionaler Modellierung und 3NF für DWH-Persistenz; pros: schnelles und paralleles Laden möglich, robust gegenüber Beziehungsänderungen, einfach erweiterbar
- Hub-Link-Sat Modell: Hub = list of unique business keys, link = list of relationships/associations, satellites = descriptive data

Big Data

most parts omitted

- 5 Vs: Volume, Velocity, Variety (text, image, video etc.), Veracity (different data quality), Value
- NoSQL: key-value store, focus on horizontal scalability (shared-nothing storage)
- MapReduce is cool!
- shared storage: SAN, does not scale well for big data; shared-nothing: every server has local disk array, scales well with HDFS
- architecture goals of Hadoop:

    - resilient against hardware failures: data distribution and replication across nodes, automatic error detection and correction
    - management of large data: file sizes of Gigabytes to Terabytes, millions of files per instances
    - simple coherence model: write-once-read many access, data can't be changed anymore
    - portability: across hardware and software
    - linear scalability: more nodes can perform more work within the same time, linear in data size and resources
    - computation near data: minimize expensive data transfer across network, big data / small programs, "bring the code to the data" / "processing at the edge"
    - streamed data access: avoid random access, read large data blocks
- Hadoop is good for large, streamable data on commodity hardware; bad for many small files, low-latency access, and many inserts w/ random access

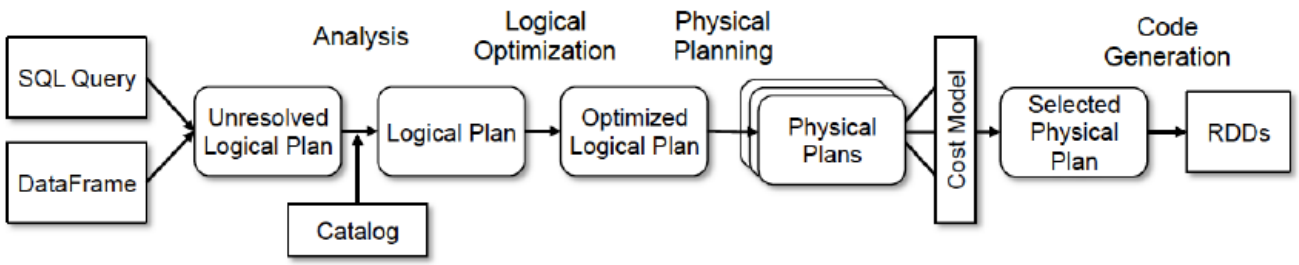
Spark Intro

- Apache Spark provides uses main-memory processing and provides: Map/Reduce, SQL, real-time stream processing, ML, graph processing
- typical throughput disk 100 MB/s, RAM 50 GB/s (net, assuming 10 Gbps, 1.25 GB/s)
- driver: written by dev, connects to worker clusters; workers: reads data blocks from distributed file syste, uses RAM for storage
- DataFrames are like tables; immutable, lineage information (to recompute lost data), parallelized; can be created from lists, pandas, or files (HDFS or other); each row is a `Row` with attributes
- Spark abstractions: RDD (resilient distributed data set, low-level), DataFrame, Dataset (JVM object); DFs are high-level, typed, performance-optimized
- some methods: `show, regiserTempTable, createDataFrame, sql, read.{format}, write.save, printSchema, select/where/sort/orderBy/distinct/filter, show/take/collect/count/describe`
- Spark can access different data sources and vice-versa (JDBC/ODBC)

HDFS Parquet

- Hadoop Distributed File System HDFS: parallel fs for large amounts of data, on top of Linux, data distributed onto n machines, 128M block size
- Mapper: reads data line-by-line, one mapper task per input block, logical separation if data is larger than input block size (no separation of single fields)
- HDFS commands `hadoop fs: --copyFromLocal, --ls, mkdir, --cat, --copyToLocal`
- Data Lake: local storage should be fast for queries, easy to backup, minimal learning curve, easy tooling integration, resource efficient
- Parquet: column storage for Hadoop, binary, encoded, compressed, machine-friendly; flat/nested schema: encoding schemes: incremental, dictionary, plan, RLE (run-length)
- Parquet writing is slower than CSV due to encoding, yet reading is faster (only specific columns need to read, columns are compressed); often: write-once, read-often

Spark Query Optimization



- from trees, create logical plan (describes computation w/o defining on how to conduct it), then physical plan (which describes specifics of how it's conducted)
- JOIN types: nested-loop (two/four for-each loops), sort-merge (sort, then merge; typically faster than nested-loop on larger tables, faster than hash if tables are sorted), hash (scan and hash, then iterate by tuples and join using hash; no sorting, thus often faster than sort-merge, can't be used for inequality operators)
- DataFram = execution plan + result type schema + RDD (= lineage + partition information + instructions); DF is NOT data
- DataFrame is declarative and operates on typed, structured columns being executed somewhere; RDD is functional and operates on homogeneous rows being executed on n machines; JVM code is imperative operating on typed variables being executed on 1 machine with n CPUs
- query optimization: prune unnecessary data as early as possible and minimize per-operator cost; e.g. predicate pushdown (in tree, execute predicate in a lower/earlier level), column pruning (only select / project what's actually needed in the end)
- distributed JOIN types: shuffle hash (map by join condition, shuffle by output key, join in reduce phase; best when even distribution of key and adequate number of keys for parallelism; problems due to uneven sharding and limited parallelism (e.g. 26 cantons)), broadcast hash (good when data frame is small enough to fit into main memory; no shuffling and no net communication required; often better than shuffle hash; doesn't work on text input w/ Spark)

Spark Machine Learning

- Learning method (algorithm) depends on:
    - type of performance element (classify? regress? control?)
    - available feedback (labels)
    - type of component to be improved (representation? utility function? action?)
    - data representation (numerical or categorical data, logical clauses, raw pixels, ...)
- Gradient descent is a general-purpose optimizer; implementation details (simultaneous updates) and hyper parameters are practically very relevant
- Learning performance = prediction accuracy measured on separate test set
    - Development using 5-fold cross validation (without ever looking at test set!)
    - Systematic and repeatable experiments are paramount (e.g. using UNIX-style scripts)
- Algos in Spark:
    - • Classification and regression: Linear models (SVM, linear regression, logistic regression), Naïve Bayes, Decision trees, Ensemble trees
    - Collaborative filtering: Alternating least squares
    - Clustering: K-means, Gaussian mixtures
    - Dimensionality reduction: Singular value decomposition (SVD), Principle component analysis (PCA)
- Transformer: abstraction that includes feature transformers and learning models, appends columns to DataFrame; e.g. maps texts to integer (feature vector) or predicts label for feature vector
- Estimator: abstraction for learning algorithm that fits or trains data

Spark Scalable Machine Learning

- supervised = labeled (SVM, neural nets etc.), unsupervised = unlabeled data (k-means)
- clustering usage: partitioning, image segmentation, anomaly detection, text categorization, grouping
- k-means: initial random centroids, repeat until converged; k can be derived from data, choose the one with smallest error; can't handle non-spherical clusters

- Many machine learning problems can be formulated as a convex optimization problem of the form

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

  where
  - x_i … training examples (features)
  - y_i … corresponding labels that need to be predicted
  - w … weights for features (vector of dimension d)
  - g … gradient of the loss
  - n … #training points (can be large!)

- Summation of losses
- Goal: find the best w with optimization, i.e. minimize error on training data
- Method is linear if the function can be expressed as $w^T x$ and y
- In summary, many machine learning problems rely on efficient linear algebra libraries

- using and exploiting sparsity is beneficial

Datenqualität

- Fehlertypen: Duplikate, out-of-range, missing values, Typos, Abkürzungen, vertauschte Werte, abweichende Bezeichnungen, unvollständige Angaben
- Ursachen: downtime, manuelle Eingabe, fehlerhafte Konfigurationen, unvollständiger / asynchroner Informationsfluss, fehlerhafte Konvertierung
- Methoden zur Duplikatserkennung
  - Masken / Wildcards: pro: simpel; con: Suchmuster muss genau bekannt sein
  - String-Distanzen: Editierdistanz/Levenshtein, Jaro-Winkler; pro: generell einsetzbar, gut für kleine Tippfehler, Jaro-Winkler ist ein Ähnlichkeitsmass; con: funktioniert nur bei kurzen Strings
  - phonetische Suche: Soundex, Kölner Phonetik; pro: gut für Namensabgleich; con: sprachabhängig, kleine Tippfehler können zu grossen Änderungen führen, kein richtiges Ähnlichkeitsverfahren
- Datenfehler: Typenfehler (try-catch), Bereichsfehler (Alter 0<x<120), ungültige Domänenwerte: pro: gilt für alle Prozesse; con: bedingt Anpassung an der DB-Definition
- Datenfehlererkennung: outlier detection, decision trees, box & whisker plots
- Prozess für Fehlereleminationskette: Fehlerkorrektur ist in der Verantwortung des Business (Data-Owner); Fehler werden nicht a priori beseitigt, sondern markiert; Fehler sind nicht immer Fehler, sondern Anomalien von Geschäftsdaten
- Anomalien als Anwendungsfall: (Preventative-)Maintenance, fraud detection, Finanzdaten (Börse, Steuern, Risiko, Verbrechen), erkennen dieser Anomalien als Adressprüfer, SLA Überprüfung
- Projektmanagement: DWH ist viel komplexer als typische IT-Projekte; 1 Personenjahr = 200 Tage, 1 Personentag = CHF 1000, Overhead = 25%