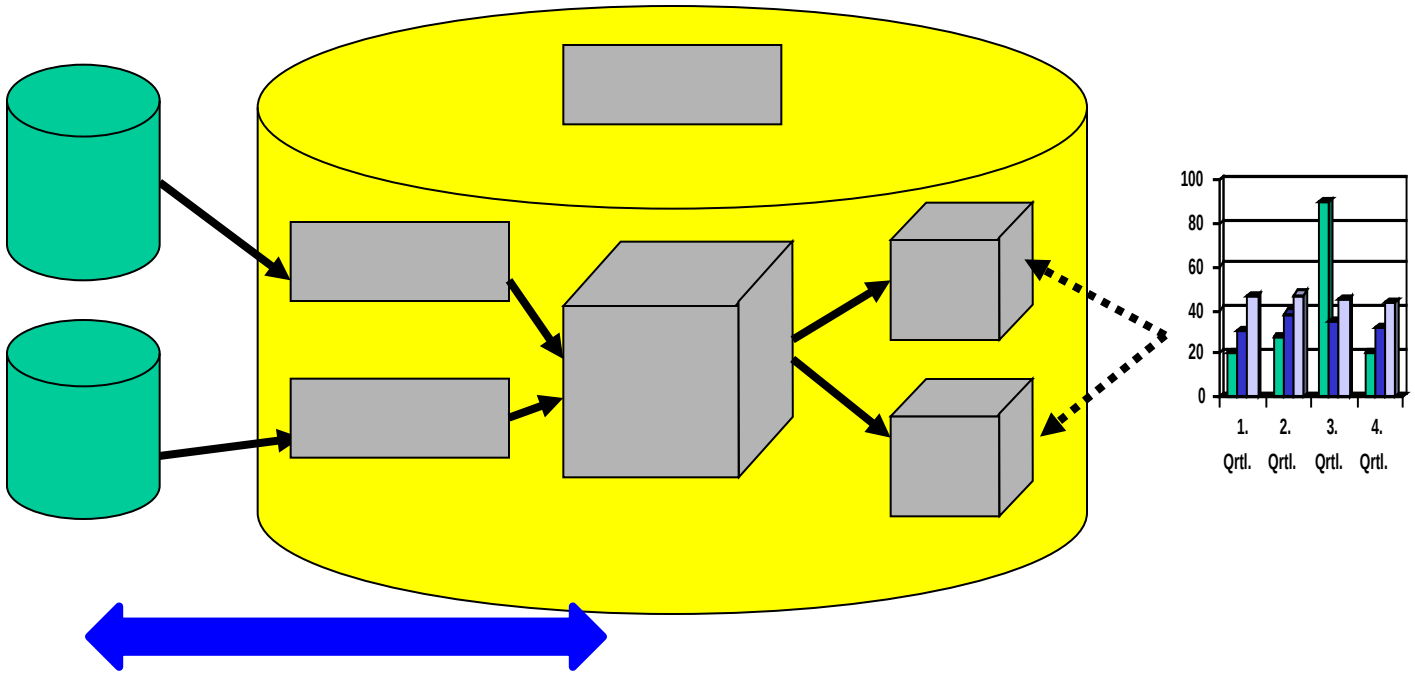
Data Warehousing und Data Mining

ETL: Extraction, Transformation, Load

Ulf Leser

Wissensmanagement in der Bioinformatik

Der ETL Prozess



- Extraction
- Transformation
- Load

Inhalt dieser Vorlesung

- ETL: Extraction, Transformation, Load
  - Extraktion von Daten aus Quellen
  - Das Differential Snapshot Problem
  - Laden von Daten in das DWH
- Schemaheterogenität
- Transformation
- Datenqualität

ETL - Übersicht

- Daten werden physikalisch bewegt
  - Von den Quellen zur Staging Area
    - Extraktion von Daten aus den Quellen
    - Erstellen / Erkennen von differentiellen Updates
    - Erstellen von LOAD Files
  - Von der Staging Area zur Basisdatenbank
    - Data Cleaning und Filtering
    - Erstellung integrierter Datenbestände
- Ziele
  - Höchstmögliche Aktualität des Datenbestands im DWH
  - Sicherung der DWH Konsistenz bzgl. Datenquellen
  - Bei maximaler Datenqualität
  - Bei minimaler Belastung des DWH und der Quellsysteme

1. Extraktion

- Extraktion von Änderungsdaten aus Quellen
- Überwindet Systemgrenzen
  - Zugriff auf HOST Systeme
    - BATCH Jobs, Reportwriter, ...
  - Daten in non-standard Formaten
    - „DB-nahe“ Programmierung wie PL-1, COBOL, Natural, IMS, ...
    - Beliebig strukturierte Dateien, XML, Reports, etc.
  - Verteiltes (Herrschafts-)Wissen
    - Doppelbelegung von Feldern, sprechende Schlüssel, ...
    - Fehlende Dokumentation
  - Teilweise kommerzielle Tools vorhanden
    - Insb. zum Zugriff auf Standardsoftware
    - Adaptoren – Riesenmarkt, siehe EAI

Extraktion

- Zwei wichtige Eigenschaften
  - Zeitpunkt
  - Art der extrahierten Daten
- Unterscheidung Fakten / Dimensionen
  - Fakten werden immer nur eingefügt
  - Änderungen in Klassifikationsknoten müssen versioniert und eventuell erst erkannt werden

Zeitpunkt der Extraktion

- Synchrone Extraktion: Quelle propagiert jede Änderung
- Asynchrone Extraktion
  - Periodisch
    - Push: Quellen erzeugen regelmäßig Extrakte (Reports)
    - Pull: DWH fragt regelmäßig Datenbestand ab
  - Ereignisgesteuert
    - Push: Quelle informiert z.B. alle X Änderungen
    - Pull: DWH erfragt Änderungen bei bestimmten Ereignissen
      - Jahresabschluss, Quartalsabschluss, ...
  - Anfragegesteuert
    - Push: -
    - Pull: DWH erfragt Änderungen bei jedem Zugriff auf die (noch nicht vorhandenen oder potentiell veralteten) Daten

Art der Daten

- Snapshot: Quelle liefert Bestand zu festem Zeitpunkt
  - Typisch für Klassifikationsknoten: Produktkatalog, Kundenliste
  - Import erfordert Erkennen von Änderungen
    - Differential Snapshot-Problem
    - Alternative: Komplettes Überschreiben (aber IC!)
  - Historie aller Änderungen kann nicht exakt abgebildet werden
- Log: Quelle liefert jede Änderung seit letztem Export
  - Typisch für Fakten: Verkäufe etc.
  - Kann relativ leicht importiert werden
    - DWH speichert ggf. Ereignis und seine Stornierung
- Nettolog: Quelle liefert das Delta zum letzten Export
  - Kann relativ leicht importiert werden
  - Keine komplette Historie möglich

Datenversorgung

Quelle ...		Technik	Aktualität DWH	Belastung DWH	Belastung Quellen
... erstellt periodisch Exportfiles		Reports, Snapshots	Je nach Frequenz	Eher niedrig	Eher niedrig
... pushed jede Änderung (Real-Time)		Trigger, Changelogs, Replikation	Hoch	Hoch	Hoch
... erstellt Extrakte auf Anfrage (Pull)	**Vor Benutzung**	Schwierig	Hoch	Hoch	Je nach Anfragen
	Anwendungs-gesteuert	Je nachdem	Je nach Frequenz	Je nach Frequenz	Je nach Frequenz

Inhalt dieser Vorlesung

- ETL: Extraction, Transformation, Load
  - Extraktion von Daten aus Quellen
  - Das Differential Snapshot Problem
  - Laden von Daten in das DWH
- Schemaheterogenität
- Transformation
- Datenqualität

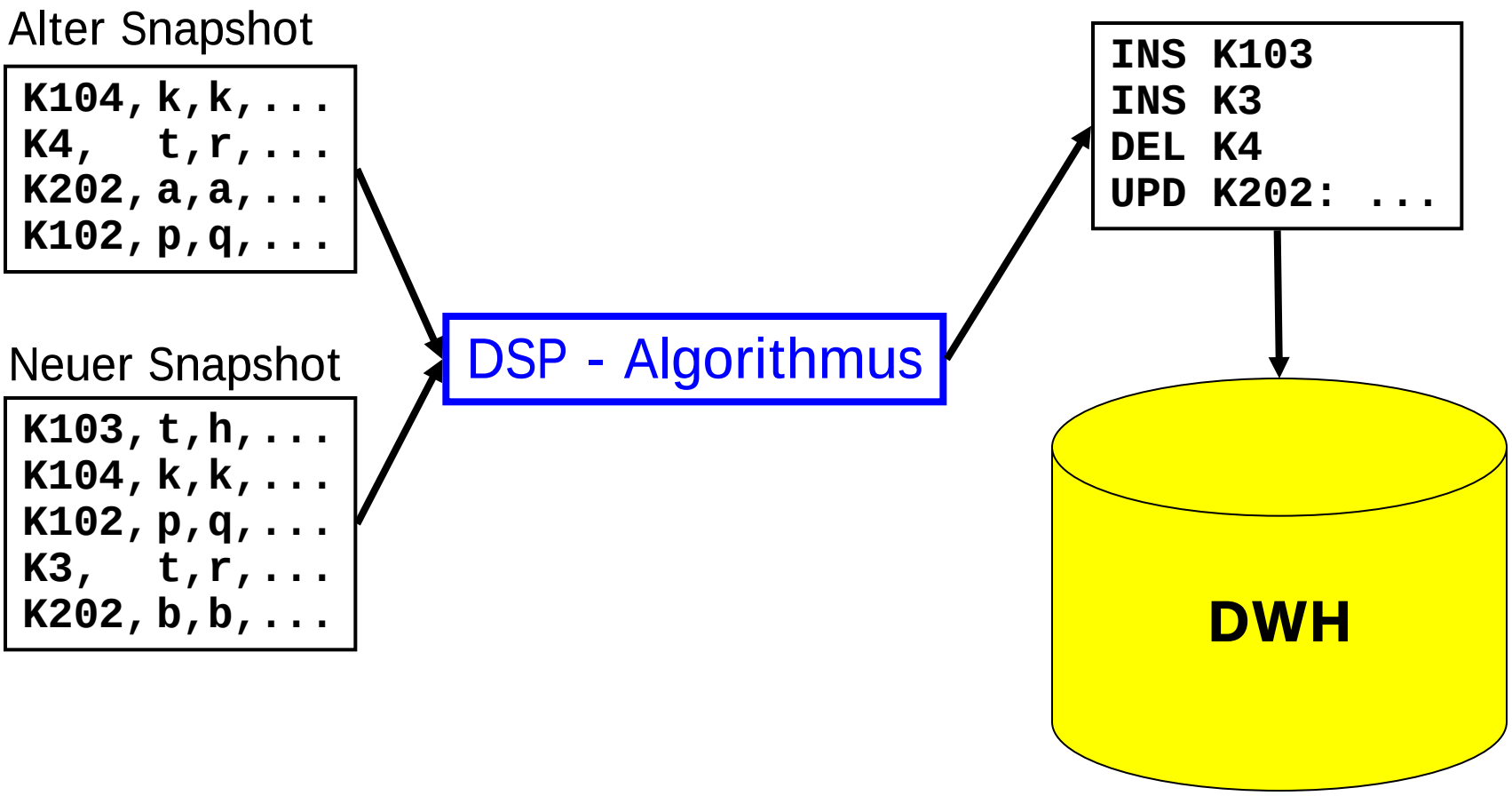
Differential Snapshot Problem [LGM96]

- Viele Quellen liefern immer den vollen Datenbestand
  - Kundenlisten, Angestelltenlisten, Produktkataloge
- Problem: Die meisten Tupel ändern sich selten
  - Ständiges (Neu-)Einspielen aller Daten ineffizient
  - Führt zu vielen unnötigen Versionen
  - Viele Probleme mit Foreign-Keys etc.
- Gesucht: Algorithmen zur Berechnung des „Delta“
- Annahmen: Quelle liefert Snapshots als Files F_i
  - Snapshot einer Tabelle
  - Zeilen (Tupel) haben identische Struktur mit Attributen A_1, ... A_n
  - Jedes Tupel hat einen Schlüssel k
  - File = ungeordnete Menge von Tupeln (K, A_1, ... A_n)

Differential Snapshot Problem (DSP)

- Definition
  Gegeben zwei Dateien F_1, F_2 gleicher Struktur mit $f_1=|F_1|$, $f_2=|F_2|$. Das Differential Snapshot Problem (DSP) besteht darin, die kleinste Menge $O=\{INS, DEL, UPD\}^*$ zu finden für die gilt: $O(F_1) = F_2$
- Bemerkung
  - INS, DEL, UPD operieren jeweils auf genau einem Tupel identifiziert über Schlüssel k
    - Sei $t_1 \in F_1$, $t_2 \in F_2$. Dann ist $t_1=t_2$ gdw. $T_1.k=t_2.k$
    - Siehe auch Vorlesung zu Duplikaterkennung
  - O im Allgemeinen nicht eindeutig
    - $O_1=\{ (ins(X)),(del(X)) \} \equiv \{\}$
  - Ähnlich zu Joins, aber Tupel ohne Match erzeugen INS / DEL
    - Full outer Join

Szenario


Alter Snapshot
K104,k,k,...
K4, t,r,...
K202,a,a,...
K102,p,q,...
Neuer Snapshot
K103,t,h,...
K104,k,k,...
K102,p,q,...
K3, t,r,...
K202,b,b,...
DSP - Algorithmus
INS K103
INS K3
DEL K4
UPD K202: ...
DWH

Annahmen

- Kostenmodell
  - Wir zählen nur IO
    - Alle Operationen im Hauptspeicher sind umsonst
  - Das Lesen/Schreiben eines Tupels kostet 1
    - Keine Beachtung von Blockgrößen, sequentiellen Reads etc.
  - Das Schreiben des DS ignorieren wir
    - Immer gleich teuer (wenn wir das kleinste finden ...)
- Hauptspeichergröße: m Tupel
- Files in der Regel wesentlich größer als Hauptspeicher

1. Versuch: DS_{small} - kleine Files

- Annahme: $m>f_1$ oder $m>f_2$
- Algorithmus (sei $m>f_1$) DS_{small}:
  - F_1 komplett in den Hauptspeicher lesen
  - Array A mit $|A|=f_1$ initialisieren
  - F_2 sequentiell lesen. $\forall r \in F_2$
    - $r \in F_1$: $O=O \cup$(UPD r) (oder ignorieren, wenn keine Änderung)
    - $r \notin F_1$: $O=O \cup$(INS r)
    - A[r]=true
  - $\forall r \in F_1$ mit A[r]=false: $O=O \cup$(DEL r)
- Anzahl IO: f_1+f_2
- Verbesserungen
  - F_1 im Speicher sortieren – schnellerer Lookup
  - Hat keinen Effekt auf IO

2. Versuch: DS_{naive} – Nested Loop

- Ab jetzt: $m << f_1$ und $m << f_2$
- Algorithmus DS_{naive}
  - $\forall r \in F_1$
    - r in F_2 suchen, dazu F_2 sequentiell lesen
    - r nicht in F_2: $O = O \cup (DEL\ r)$
    - r verändert in F_2 : $O = O \cup (UPD\ r)$
    - r unverändert in F_2 : ignorieren
- Aber: INS wird nicht gefunden
- Lösung
  - Array A verwalten: $\forall r \in F_1 \cap F_2 : A[r] = true$
  - $\forall r$: $A[r] = false$: (INS r)
  - Geht nur, wenn A in den Hauptspeicher passt
  - Sonst: Zweiten Nested Loop mit F_1, F_2 vertauscht

DS_{naive} – Analyse

- Kosten
  - Worst-Case: $f_1 * f_2$ + INS-Erzeugung
  - Average-Case: $f_1 * f_2 / 2$ + INS-Erzeugung
- Verbesserungen
  - F_2 sortieren und mit bin-search suchen
    - Zusätzliche Sortierphase
    - Vergleichsphase kostet dann $f_1 * \log(f_2)$
    - Aber: Random Access IO, deutlich langsamer als sequentielles Lesen
  - Jeweils Partition mit Größe m von F_1 laden
    - Verbesserung der Kosten auf $f_1/m * f_2$
    - Ähnlich zu Blocked-Nested-Loop

Einschub: Sortieren auf Platte

- Wie sortiert man ein File F, das nicht in den Hauptspeicher passt?
  - F in Partitionen P_i mit $|P_i|=m$ lesen
  - Alle P_i im Hauptspeicher sortieren und als Runs F^i schreiben
  - Alle F^i gleichzeitig öffnen und ...
    - Dazu müssen wir vielleicht sehr viele Dateien öffnen
    - Kann man umgehen (hierarchisches externes Sortieren)
  - ... aus allen Files in Sort-Merge-Reihenfolge lesen und aktuelles Tupel jeweils in F^{sort} schreiben
    - Implementierung über Priority-Queues
- Kosten: f + f + f + f = 4*f
  - Ist das Sortieren in linearer Zeit?

3. Versuch: DS_{sort} – Sort-Merge

- F_1 und F_2 sortieren
- Naiv
  - Beide sortieren: $4*f_1 + 4*f_2$
  - Beide Files parallel öffnen und in Sort-Merge-Art lesen
    - r in F_1 und F_2: UPD
    - r nur in F_1: DEL
    - r nur in F_2: INS
  - Zusammen: $5*f_1 + 5*f_2$
- Arbeit sparen
  - Sortiertes F_2 aufheben (wird sortiertes F1 im nächsten Update-Zyklus)
  - Kosten: f_1+5*f_2

DS_{sort2} – Verbesserung

- Idee: F_2 nicht erst komplett sondern nur „halb“ sortieren
- F_2 in Partitionen P_i mit $|P_i|=m$ lesen
- P_i im Hauptspeicher sortieren und als F_2^i schreiben
- Alle F_2^i mergen und gleichzeitig mit F_1 vergleichen
  - Dabei sortierte F_2^i für das nächste Mal schreiben
  - F_1 ist noch vom letzten Mal sortiert
- Kosten: f_1+4*f_2
- Noch besser: Immer nur auf Runs arbeiten
  - Kosten: f_1+3*f_2
  - Braucht doppelt so viele File Handle

4. Versuch: DS_{hash} – Partitioned Hash

- Sortierung der P_i ist nicht wirklich notwendig
  - Einzige Bedingung: Aufteilung in P_i muss reproduzierbar sein
- Hash-basierter Algorithmus
  - F_2 in Partitionen P_i mit $|P_i|=m/2$ hashen
    - Schwierig: Sicherstellen der Platzobergrenze der Hash-Buckets
  - F_1 liegt noch partitioniert vom letzten Mal vor
    - Mit derselben Hashfunktion
    - Damit hat man pro Hashkey zwei maximal m/2 große Partitionen
  - Jeweils $P_{1,i}$ und $P_{2,i}$ parallel lesen und DS berechnen
- Anzahl IO: $f_1 + 3*f_2$
  - Ev. billiger als DS_{sort2}, weil Sortierung im Speicher gespart wird
  - Braucht aber mehr Partitionen und eine sehr gute Hashfunktion

Warum nicht einfach …

- UNIX diff verwenden?
  - diff erwartet / beachtet Umgebung der Records
  - Hier: Records sind ungeordnet
- DSP in der Datenbank lösen?
  - Wahrscheinlich dreimaliges Lesen jeder Relation notwendig

```
INSERT INTO delta
       SELECT ‚UPD‘, ...
       FROM F1, F2
       WHERE F1.K=F2.K AND K1.W≠F2.W
UNION
       SELECT ‚INS‘, ...
       FROM F2
       WHERE NOT EXISTS ( ...)
UNION
       SELECT ‚DEL‘, ...
       FROM F1
       WHERE NOT EXISTS ( ...)
```

Vergleich - Eigenschaften

	IO	Bemerkungen
DS_{naiv}	$f_1 * f_2$	• Außerdem INS-Datenstruktur notwendig
Ds_{small}	$f_1 + f_2$	• Nur für kleine Dateien
Ds_{sort2}	$f_1 + 3*f_2$	• Viel sortieren im Hauptspeicher • Muss sehr viele Files gleichzeitig öffnen

Ds_{hash}	$f_1 + 3*f_2$	• Braucht gleichverteilende Hashfunktion
		  • Kein Bucket darf größer m/2 sein
		  • Defensiv vorgehen - Platzverschwendung

Weitere DS Verfahren

- Mehr Runs als mögliche Filehandles
  - Hierarchische externe Sortierverfahren
- Files komprimieren
  - Größere Partitionen / Runs
  - Größere Chance, Vergleich in-memory durchzuführen
  - In Realität schneller (bei unserem Kostenmodell nicht)
- „Window“ Algorithmus
  - Annahme: Files haben eine „unscharfe“ Ordnung
  - Merging mit sliding window über beide Files
  - Liefert u.U. falsche INS-DEL Paare
  - Deutlich schneller on average

Inhalt dieser Vorlesung

- ETL: Extraction, Transformation, Load
  - Extraktion von Daten aus Quellen
  - Das Differential Snapshot Problem
  - Laden von Daten in das DWH
- Schemaheterogenität
- Transformation
- Datenqualität

Load

- Effizientes Einbringen von externen Daten in DWH
- Kritischer Punkt: Load-Vorgänge blockieren ggf. das komplette DWH
  - Schreibsperre auf der Faktentabelle und ihren Indexen
- Möglichkeiten
  - Tupel für Tupel (SQL)
  - BULK-Load
  - Anwendungsspezifische Schnittstellen
- Aspekte
  - Trigger, Integritätsconstraints, Indexaktualisierung
  - Update oder Insert?

Satzbasiert: Tupel für Tupel

- Standard-Schnittstellen: PRO*SQL, JDBC, ODBC, ...
- Arbeiten im normalen Transaktionskontext
- Trigger, Indexe und Constraints bleiben aktiv
  - Manuelle Deaktivierung möglich
- Nur Satzsperren
  - Können durch Zwischen-COMMIT verringert werden
    - In Oracle ist das weniger performant – shadow memory
    - Aber andere Prozesse erhalten wieder eine Chance
- Benutzung von Prepared Statements wichtig
  - Sonst wird sehr oft die gleiche Anfrage optimiert
- Teilweise proprietäre Erweiterungen (Arrays)

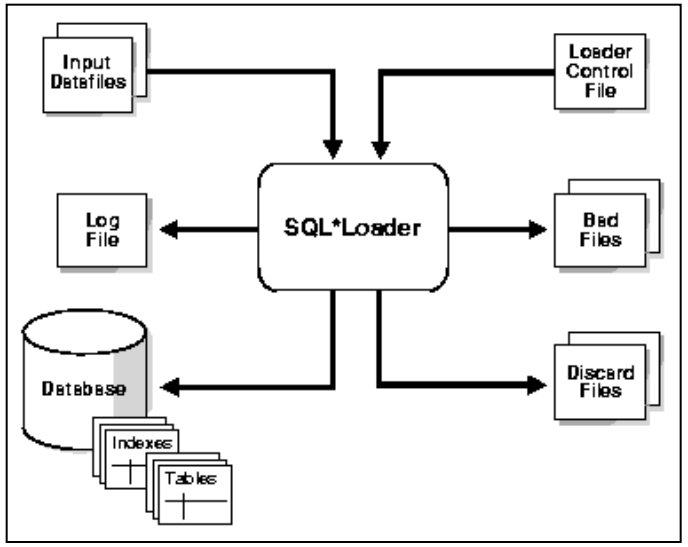
BULK Load

- Proprietäre Verfahren
- Viel schneller als satzbasiertes Einfügen
- Laufen in speziellem Kontext, nutzen eine spezielle, proprietäre, nicht offene API zum DBMS
  - Oracle: sqlldr mit DIRECTPATH Option
  - Komplette Tabellensperre
  - Keine Beachtung von Triggern oder Constraints
  - Indexe werden erst nach Abschluss aktualisiert
  - Kein transaktionaler Kontext
  - Kein Logging (d.h. kein Recovery etc.)
  - Checkpoints zum Wiederaufsetzen
  - Rasend schnell

Beispiel: ORACLE sqlldr

Controlfile

```
LOAD DATA
   INFILE 'book.dat'
   REPLACE INTO TABLE book
   (
   book_title POSITION(1) CHAR(35),
   book_price POSITION(37) ZONED(4,2),
   book_pages POSITION(42) INTEGER,
   book_id "book_seq.nextval"
   )
```

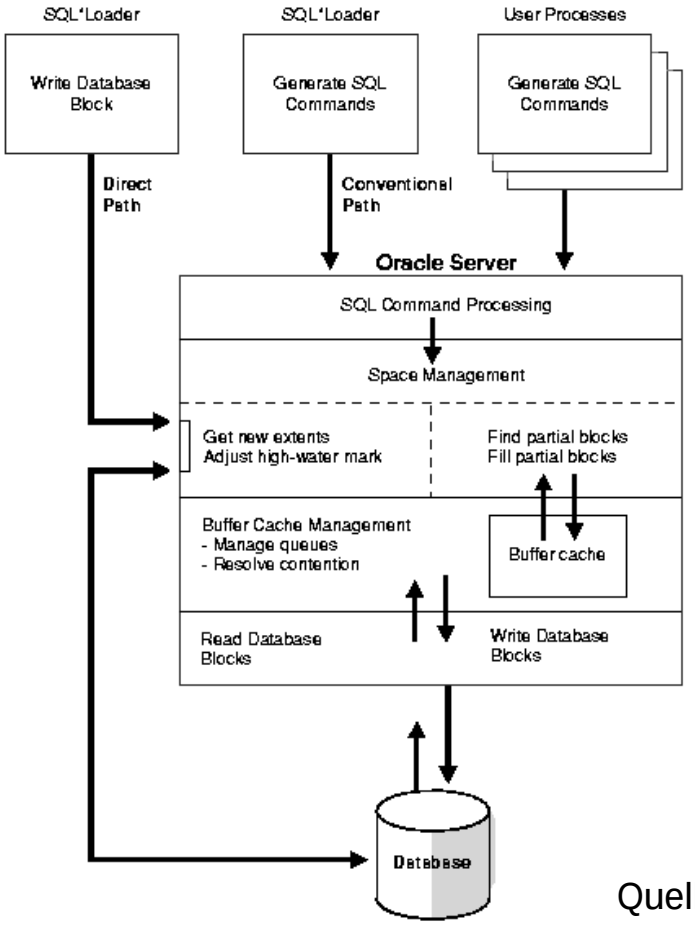


Quelle: Oracle 9.2, Dokumentation

Optionen

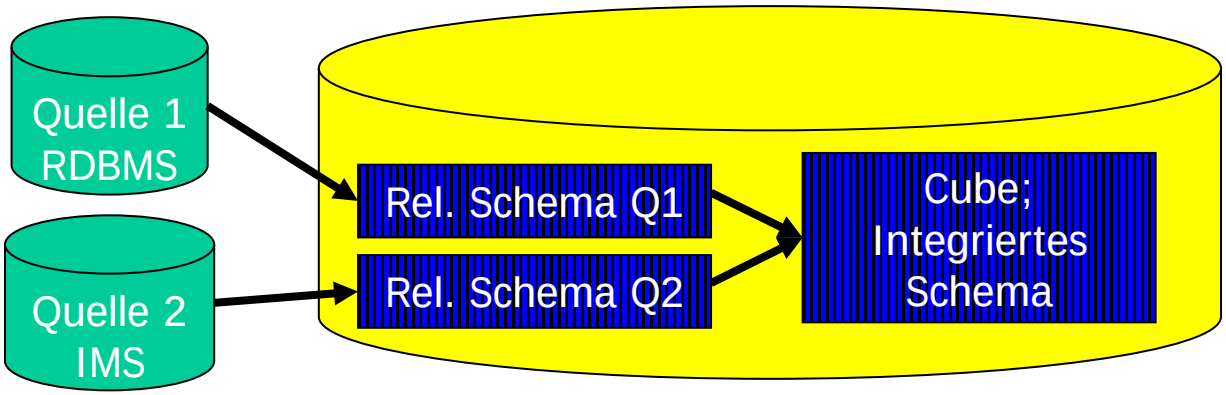
- Behandlung von Ausnahmen (Badfile)
- Einfache Datentransformationen (im Konfig-File)
- Checkpoints (Wiederaufsetzen nach Abbruch)
- Optionale Felder
- Konditionales Laden in mehrere Tabellen
- Konditionales Laden von Records
- REPLACE oder APPEND
- Paralleles Laden (in parallele Datenbank)
- RT*M

Direct Path oder nicht



Quelle: Oracle 9.2, Dokumentation

Quelle – Arbeitsbereich - Basisdatenbank



- BULK Load für ersten Schritt – in die Datenbank bringen
- Zweiter Schritt: Aus Staging Area in Cube
  - INSERT INTO ... SELECT ....
  - Logging ausschalten
  - Parallelisierbar

Transformationen beim Laden der Daten

- Extraktion der Daten aus Quelle mit einfachen Filtern
  - Spaltenauswahl, keine Reklamationen, ...
- Anwendung einfacher Konvertierungen
  - Zahl – String, Integer – Real, ...
  - Transformation sind zeilenorientiert – keine Buffer
- File für den BULK-LOADER erstellen

```
while (read_line)
  parse_line
  if (f[10]=2 & f[12]>0)
    write(file, f[1], string(f[4]), f[6]+f[7],...
  ...
bulk_upload( file)
```

Transformationen in Staging Area

- Benutzt SQL (mengenorientiert)
- Vergleiche über Zeilen hinaus möglich
  - Schlüsseleigenschaft, Duplikaterkennung, ...
- Vergleiche mit Daten in Basisdatenbank möglich
  - Duplikate, Plausibilität (Ausreißer), Konsistenz (Artikel-Ids)
- Typisch: Tagging von Datensätzen durch Prüf-Regeln

```
UPDATE newsales SET price=price/MWST;
UPDATE newsales SET cust_name=
  (SELECT cust_name FROM customer WHERE id=cust_id);
...
UPDATE newsales SET flag1=FALSE WHERE cust_name IS NULL;
...
INSERT INTO DWH
  SELECT * FROM newsales WHERE f1=TRUE & f2=TRUE & ...
```

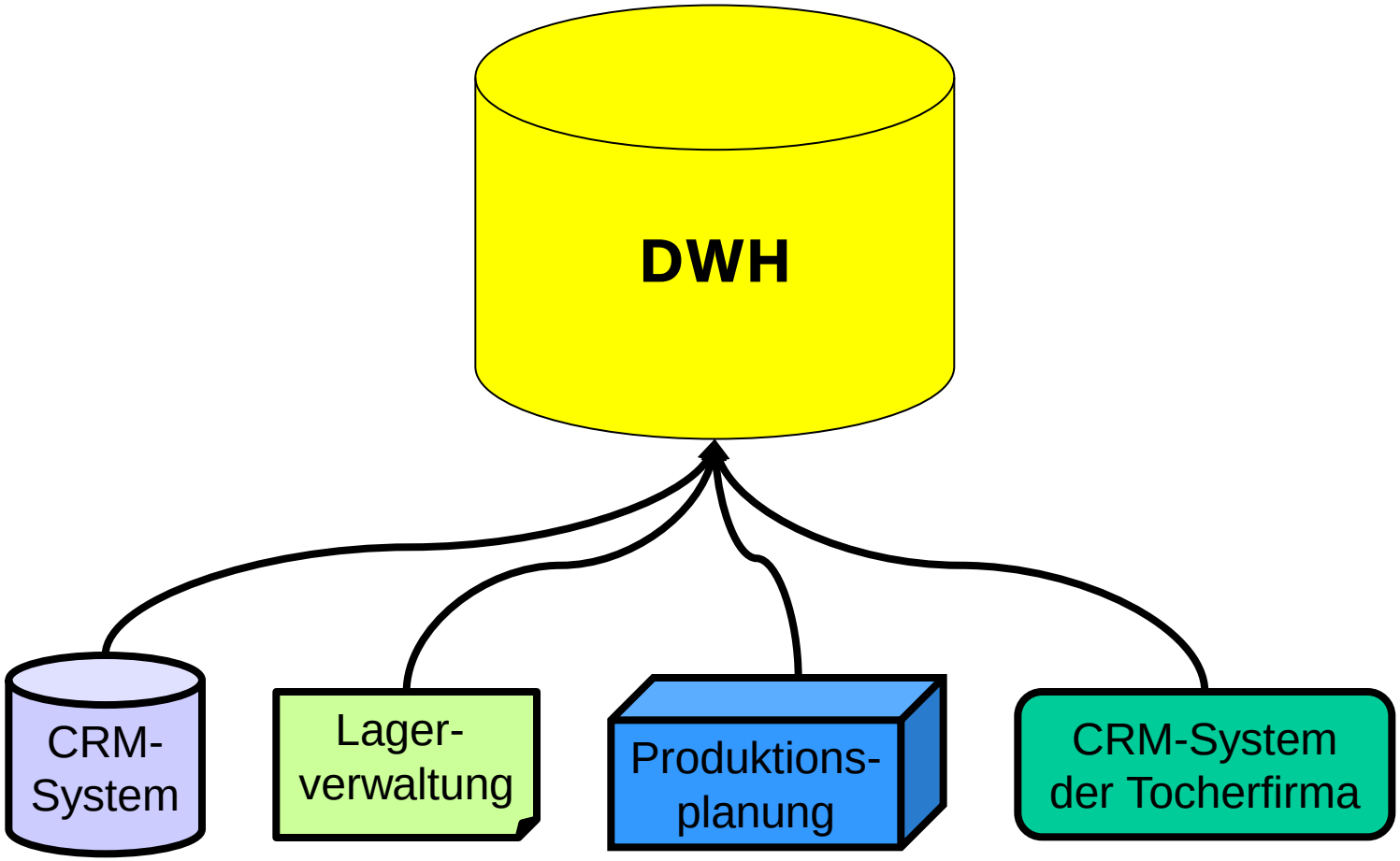
Quelle – Arbeitsbereich – Cube

	Quelle -> Arbeitsbereich	Arbeitsbereich -> Cube
Art des Zugriffs	Satzorientiert (file)	Mengenorientiert (SQL)
Zugriff auf	Eine Quelle	Viele Quellen sowie Cube
Verfügbare Datensätze	Quellabhängig: Alle, alle Änderungen, Deltas	Alle Änderungen
Typische Sprache für Transformation	Skripte: Perl, AWK, ...	SQL, PL/SQL
Typische Technik für Datenbewegung	Bulk-Loader	SQL, PL/SQL

Inhalt dieser Vorlesung

- ETL: Extraction, Transformation, Load
- Schemaheterogenität
- Transformation
- Datenqualität

Heterogenität



Heterogenität

- Zwei Informationssysteme sind heterogen, wenn sie sich in Darstellung oder Modellierung der verwalteten Daten unterscheiden
  - Schema, Modell, Format, ...
- Informationsintegration = Überbrückung von Heterogenität
  - Erstellung eines homogenen Systems
    - Materialisierte Integration
  - Erweckung des Anscheins eines homogenen Systems
    - Virtuelle Integration

Übersicht

- Syntaktische Heterogenität
- Datenmodellheterogenität
- Strukturelle Heterogenität
- Semantische Heterogenität

- Sind keine disjunkten Konzepte
- Starke Überlappungen, keine klaren Abgrenzungen

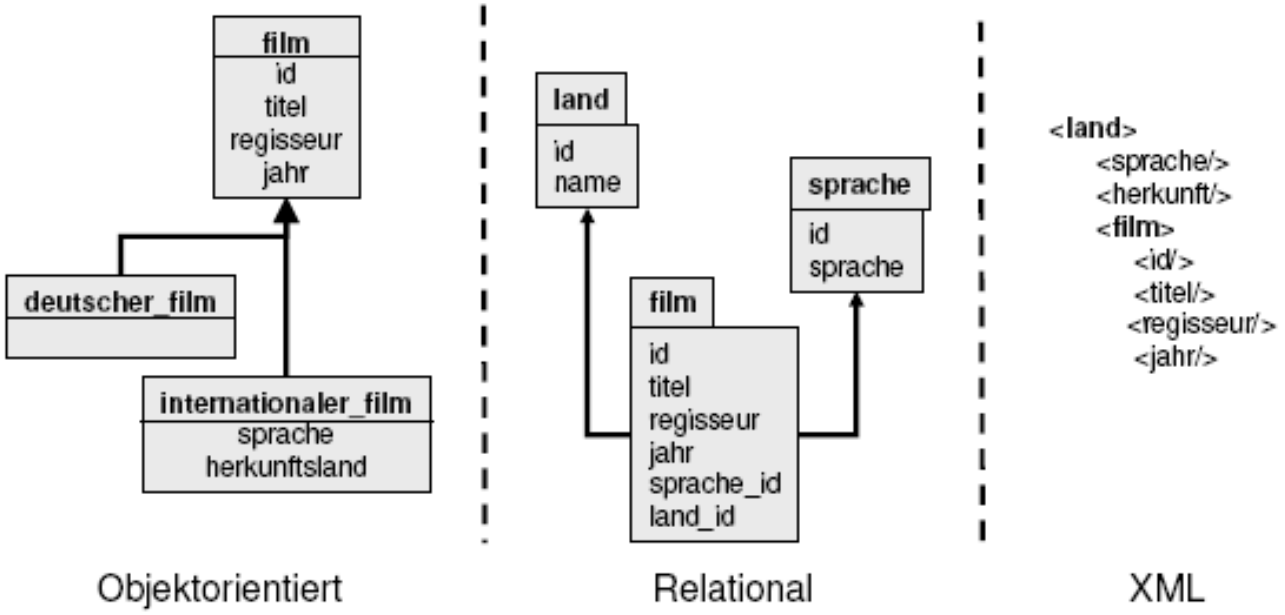
Syntaktische Heterogenität

- Unterschiedliche Darstellung derselben Fakten
  - Dezimalpunkt oder –komma
  - Euro oder €
  - Comma-separated oder tab-separated
  - HTML oder ASCII oder Unicode
  - Notenskala 1-6 oder „sehr gut“, „gut“, ...
  - Binärcodierung oder Zeichen
  - Datumsformate (12. September 2006, 12.9.2006, 9/12/2006, ...)
- Überwindung in der Regel nicht problematisch
  - Umrechnung, Übersetzungstabellen, ...

Datenmodellheterogenität

- Typische Datenmodelle
  - CSV, relational, XML, JSON
  - Exotisch (ASN.1)
  - Domänenspezifisch (ACeDB, EXPRESS, OPEN-GIS, ...)
- Zweck: Datenaustausch oder Datenspeicherung
  - XML als Speicherformat?
  - Black-Box-Sicht – was zählt, ist was die Quelle liefert
- Erfordert Konvertierung beim Load
  - Spezielle Semantik muss relational modelliert werden
  - XML-Schachtelung im relationalen Modell?

Beispiel

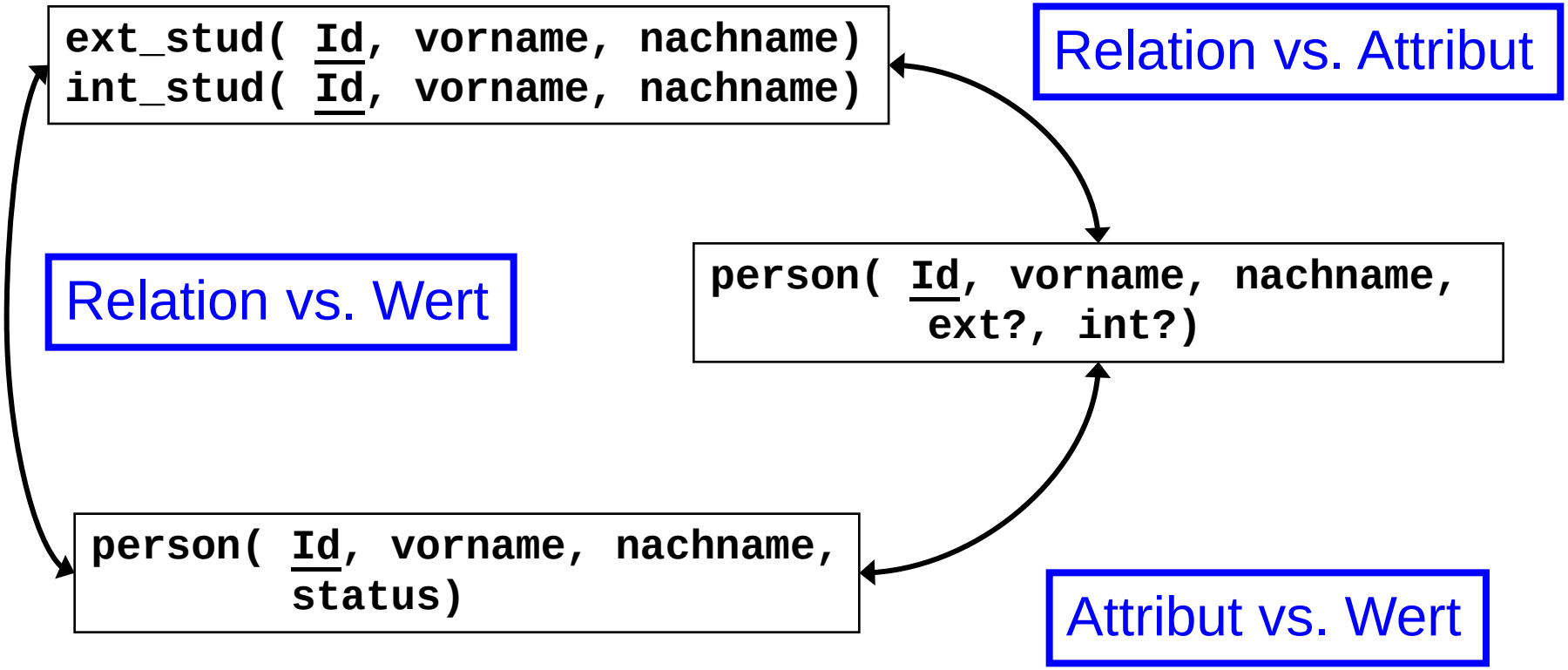


„Fast“ dasselbe in verschiedenen Datenmodellen

Strukturelle Heterogenität

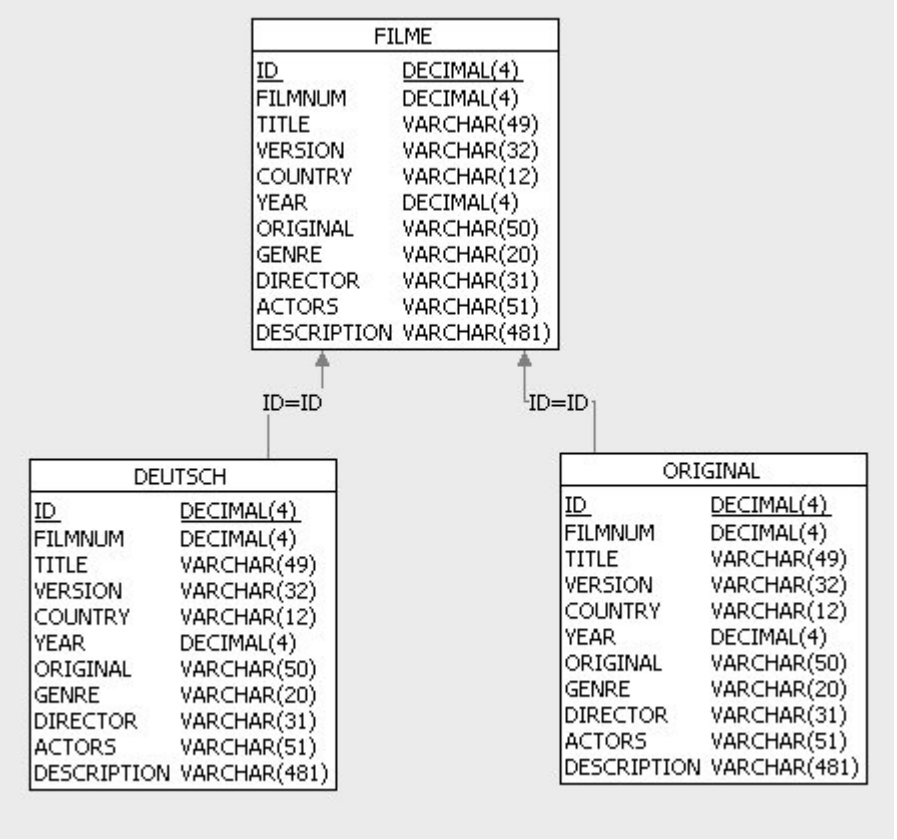
- Gleiche Sachverhalte mit unterschiedlichen Schemata ausdrücken
  - Andere Aufteilung von Attributen auf Tabellen
  - Fehlende / neue Attribute
- Fast immer mit semantischer Heterogenität verbunden
- Spezialfall: Schematische Heterogenität
  - Verwendung anderer Elemente eines Datenmodells
  - Kann nur schwer durch SQL überwunden werden

Spezialfall: Schematische Heterogenität

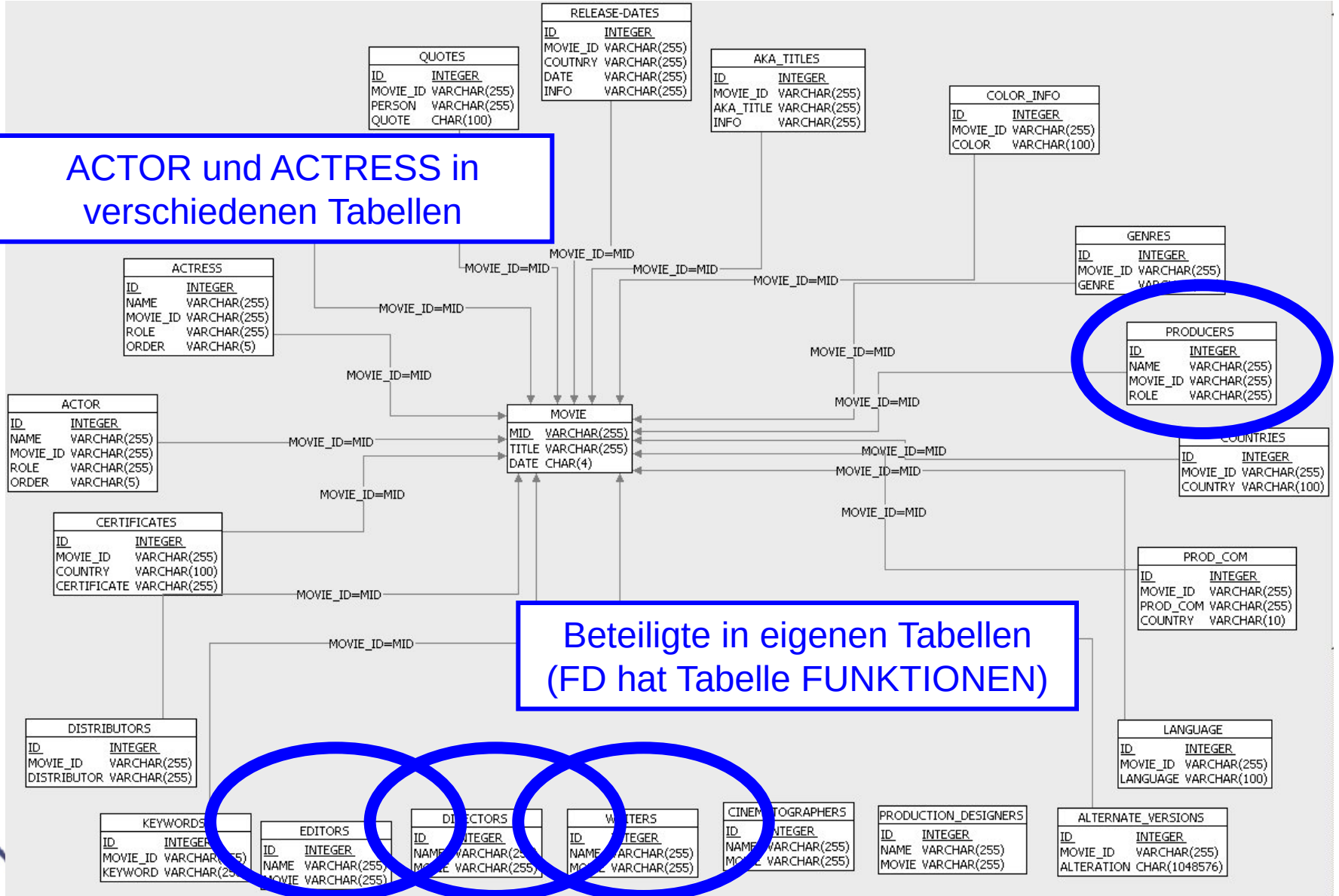


Exotische Probleme?

- Verleiher „XYZ“
  - ACTORS als VARCHAR
  - ORIGINAL?
  - TITLE, YEAR, ... an drei Stellen
  - ID-Räume DEUTSCH und ORIGINAL getrennt?



Schema der IMDB

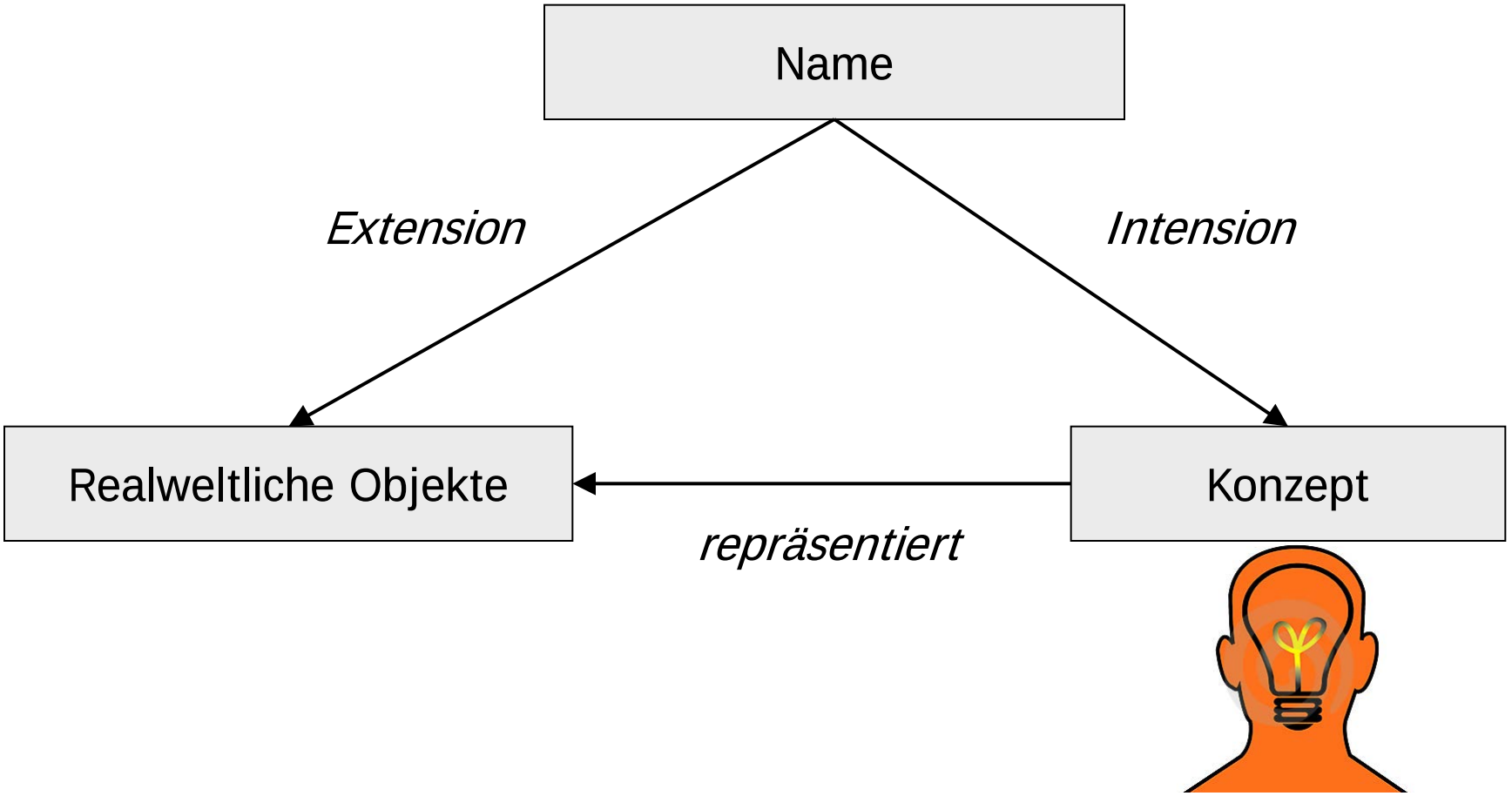


Semantik

- Fremdwörterduden zu “Semantik”
  - „Teilgebiet der Linguistik, das sich mit den Bedeutungen sprachlicher Zeichen und Zeichenfolgen befasst“
  - „Bedeutung, Inhalt eines Wortes, Satzes oder Textes“
- Programmiersprachen
  - Syntax: EBNF, Grammatiken
  - Semantik: Wirkungen der Ausführung
    - Operationale Semantik, Fixpunktsemantik, ...

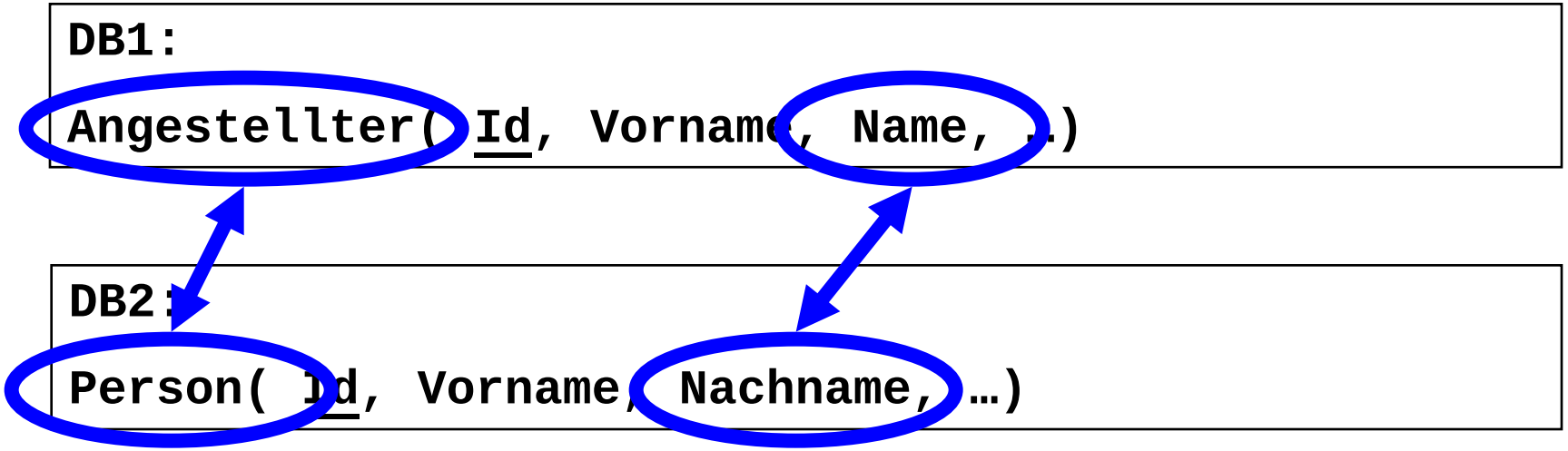
Syntaktisch falsch	Semantisch falsch
„Ich esse Butterbrot ein“	„Ich esse einen Schrank“
„IF (a==5) {...} els {...};“	„A=B/0“;

Semantik von was?



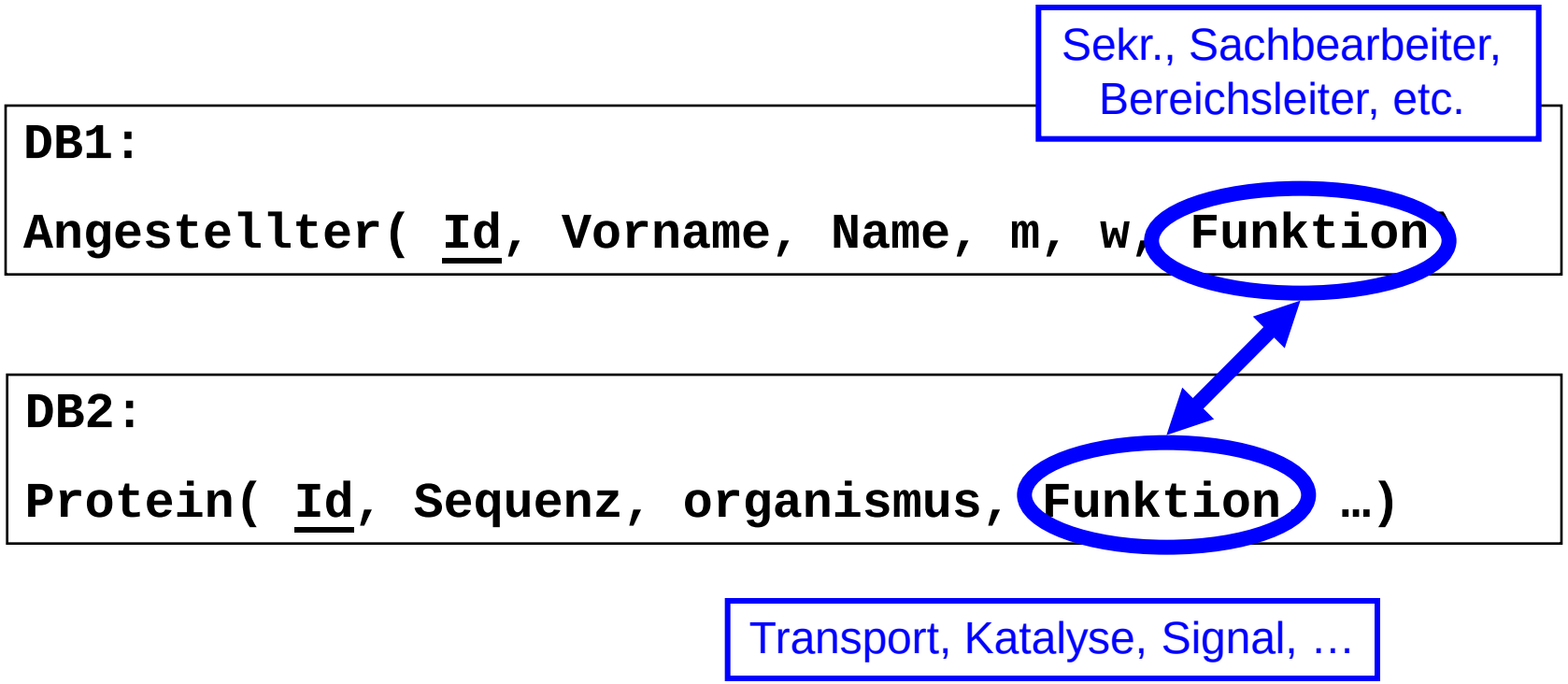
Synonyme

- Verschiedene Namen für dasselbe Konzept
  - Und die selbe „Menge“ von Objekten



Homonyme

- Gleiche Namen für verschiedene Konzepte
  - Treten oft bei Überschreitung von Domänengrenzen auf



Inhalt dieser Vorlesung

- ETL: Extraction, Transformation, Load
- Schemaheterogenität
- Transformation
- Datenqualität

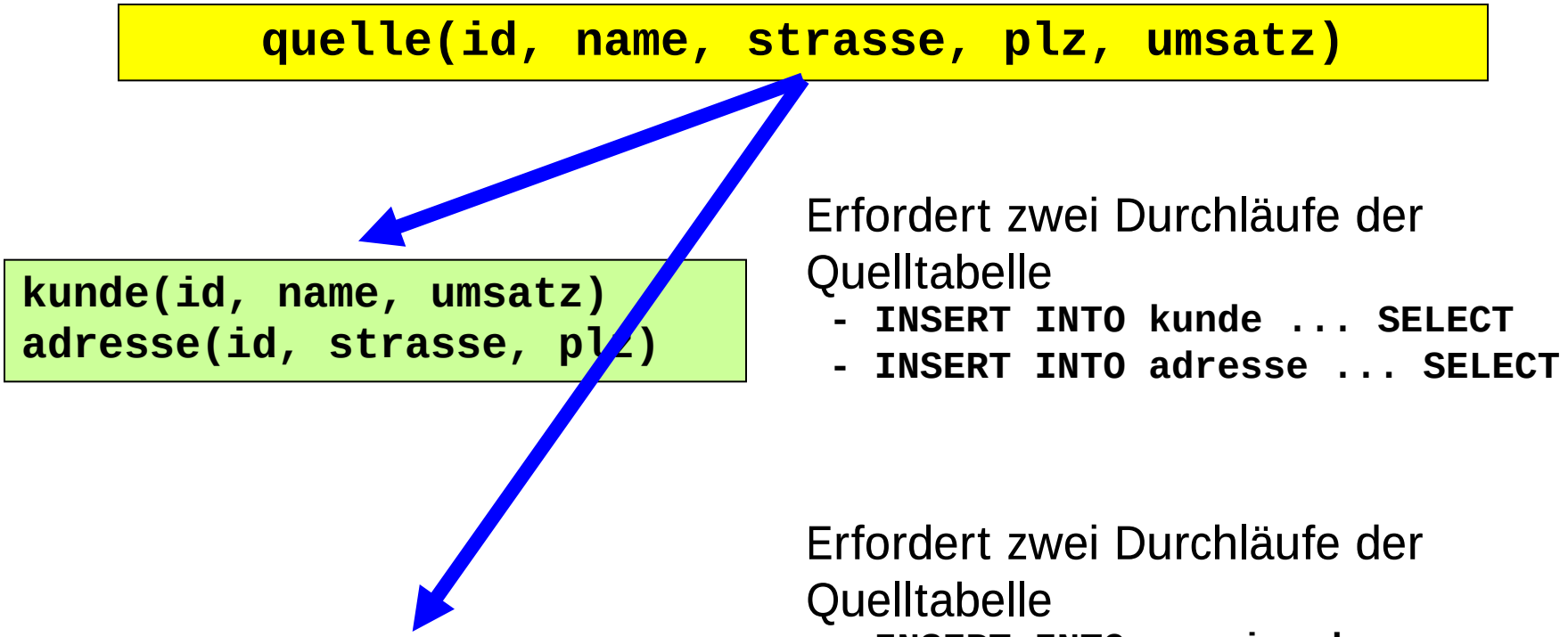
Transformation

- Der Transformationsschritt muss Heterogenität zwischen Quellen und DWH Schema überbrücken
- Arten von Transformationen
  - Schematransformation
  - Datentransformation
- Transformationen möglich an zwei Stellen
  - Transformation der Quell-Extrakte zu Load-Files
  - Transformation von der Staging-Area in die Basis-DB

Schematransformationen

- Transformation von Daten eines Schemas in ein anderes Schema
- Notwendig bei struktureller Heterogenität
- In SQL nur teilweise unterstützt
  - INSERT hat(te) nur eine Zieltabelle
  - SQL greift nur auf Daten zu
    - Keine Überbrückung schematischer Heterogenität
- Transformation erfordert meistens Programmierung

Beispiele



Erfordert zwei Durchläufe der Quelltabelle

- `INSERT INTO kunde ... SELECT`
- `INSERT INTO adresse ... SELECT`

Erfordert zwei Durchläufe der Quelltabelle

- `INSERT INTO premium_k ...`
  `SELECT ...`
  `WHERE umsatz>=X`
- `INSERT INTO normal_k ...`
  `SELECT ...`
  `WHERE umsatz<X`

Vermeidung redundanter Durchläufe

- Multi-Table INSERT

```
INSERT ALL
       WHEN umsatz < X
         THEN INTO normal_k
       WHEN umsatz >= X
         THEN INTO premium_k
       ELSE
         INTO normal_k
  SELECT id, name, umsatz
  FROM quelle;
```

- Alternative: PL/SQL (JDBC) Programmierung
  - Durchlauf der Quelltabelle mit Cursor
  - Conditionales Insert in Zieltabelle
  - Cursor meistens langsamer als SQL (aber flexibler)

Datentransformationen

- Umwandlung von Datenwerten
- Notwendig bei syntaktischer oder semantischer Heterogenität
- In SQL recht gut unterstützt
  - Vielfältige Funktionen im Sprachstandard
  - Stringfunktionen, Decodierung, Datumsumwandlung, Formeln, Systemvariable, ...
  - User Defined Functions

Beispiele

```
„Leser, Ulf“                    „Leser“,      „Ulf“
„Naumann, Felix“                „Naumann“,    „Felix“

INSERT INTO kunden( nachname, vorname)
SELECT  SubStr(name, 0, inStr(name,',')-1),
        SubStr(name, inStr(name,',')+1)
FROM rawdata;
```

```
„Leser“, 12/20/1954             „Leser“,      20.12.1954
„Naumann“, 5/18/1954            „Naumann“,    18.05.1954

INSERT INTO kunden( name, geburtsdatum)
SELECT  name,
        to_char(to_date(birthday, „MM/DD/YYYY“) „DD.MM.YYYY“)
FROM rawdata;
```

```
„Müller“, ‚m‘, „sehr gut“       „Müller“      0       1
„Meier“, ‚w‘, „gut“             „Meier“       1       2

INSERT INTO schueler(name, geschlecht, klasse)
SELECT  name,
        decode( upper(sex), ‚M‘, 0, 1),
        decode( grade, ‚sehr gut‘, 1, ‚gut‘, 2, ...)
FROM rawdata;
```

Inhalt dieser Vorlesung

- ETL: Extraction, Transformation, Load
- Schemaheterogenität
- Transformation
- Datenqualität
  - Data Cleansing
  - Der Merge/Purge Algorithmus

Datenqualität [Spiegel Online, 30.5.2007]

- **Bundesagentur meldet zu geringe Arbeitslosenzahlen**
- Die offiziellen Arbeitsmarktdaten der vergangenen Monate waren falsch. Knapp 40.000 Arbeitslose sind nicht in die Statistik eingeflossen, gab die Bundesagentur für Arbeit heute zu. **Ein ganzer Datensatz war einfach verloren gegangen**.
- Nürnberg - Zwischen der amtlichen Statistik und der tatsächlichen Arbeitslosenzahl gab es eine Differenz, sagte der Vorstandsvorsitzende der Bundesagentur für Arbeit, Frank-Jürgen Weise, heute in Nürnberg bei der Vorstellung der aktuellen Zahlen. (mehr...) Wegen einer Panne bei der Datenübertragung habe die Bundesagentur monatelang zu niedrige Arbeitslosenzahlen gemeldet.
- Angefangen hat alles mit einem Fehler im Dezember 2006. Nach Angaben Weises waren damals knapp 40.000 erfasste Arbeitslose nicht in die Arbeitsmarktstatistik eingeflossen. "Ein ganzer Datensatz ist nicht verarbeitet worden", sagte er. Besonders pikant: Das in solchen Fällen automatisch erstellte **Fehlerprotokoll blieb von den Mitarbeitern der Bundesagentur unbeachtet** liegen.
- Die Statistik vom Januar weist dadurch beispielsweise 37.500 weniger Erwerbslose aus, als es tatsächlich gab. Die Differenz zwischen der amtlichen Statistik und der tatsächlichen Arbeitslosenzahl habe sich allerdings in den darauf folgenden Monaten nach und nach verringert und betrage im Mai nur noch 6000, sagte Weise. Der Grund sei, dass viele der nicht erfassten Jobsucher inzwischen eine Stelle gefunden hätten.
- "Die Differenz in der Arbeitslosenstatistik ändert aber nichts an unserer generellen Arbeitsmarkteinschätzung und dem Trend", sagte Weise. Die amtliche Statistik werde nun nachträglich korrigiert. Außerdem werde künftig bei auftauchenden Fehlermeldungen die Verarbeitung von Arbeitslosendaten sofort gestoppt.
- Weise zufolge deckte erst eine **Plausibilitätskontrolle** im April die Statistik-Lücke auf. Es sei aufgefallen, dass sich erfolgreiche Jobsucher bei den Arbeitsagenturen abgemeldet hätten, die statistisch gar nicht erfasst waren.

Datenqualität

- Datenqualität: „Fitness for use“
- Folgen geringer Datenqualität können dramatisch sein
  - Falsche Prognosen, verpasstes Geschäft, Schadensersatz, ...
- DWH besonders anfällig für Qualitätsprobleme
  - Probleme akkumulieren
  - Qualität der Ursprungsdaten (Eingabe, Fremdfirmen, ...)
  - Qualität der Quellsysteme (Konsistenz, Constraints, Fehler, ...)
  - Qualität des ETL-Prozess (Parsen, Transformieren, ...)
- Viele Probleme treten erst bei Konsolidierung zu Tage
- DWH unterstützt strategische Entscheidungen: Hohe Folgekosten bei Fehlentscheidungen

Beispiel: Kampagnenmanagement

- Probleme im CRM eines Multi-Channel Vertriebs
  - Kunden doppelt geführt
  - Kunden falsch bewertet
  - Falsche Adressen
- Folgen
  - „False positives“: Verärgerte Kunden durch mehrere / unpassende Mailings
  - „False negatives“: Verpasste Gelegenheiten durch fehlende / falsche Zuordnung (Cross-Selling)
  - Sinnlose Portokosten bei falschen Adressen
  - ...

Aspekte von Datenqualität

- Verletze Integrität: Schlüssel, Fremdschlüssel, NOT NULL, Wertebereiche, ...
- Fehlende Daten: Fehlende Werte, fehlende Tupel
- Falsche Daten
  - Objektiv falsch: Negative Preise, 32.13.2002, ...
  - Unplausible Daten: Ausreißer, unerwartete Werte, ...
- NULL Werte
  - Wert möglich, aber unbekannt: Ist er Professor?
  - Wert möglich, existiert aber nicht: Er ist kein Professor!
  - Wert unmöglich: Kinder unter 18 haben keinen Titel
- Duplikate
- Schlechte / fehlende Dokumentation
- ...

Data Cleansing Operationen

- Ziel: Verbesserung der Datenqualität
  - Ergänzung fehlender Werte
  - Korrektur durch Lookup, Neuberechnen, Runden, ...
  - Erkennen und Löschen „unrettbarer“ Daten
  - Kosten/Nutzen: 80/20 Regel
- Datenqualität muss (und kann) gemessen werden
  - Datenqualitäts-Metriken
  - Verbesserungen quantifizieren
- Domänenabhängiger Prozess
- Produkte gibt es vor allem für Adressdaten

Nachvollziehbarkeit

- Daten im DWH müssen erklärbar sein
  - Anwender: „Da fehlt ein Produkt im Report“, „Die Umsatzzahl kann nicht stimmen“, „Da habe ich ganz andere Zahlen“
  - Analysewerkzeug fehlerhaft? Report falsch? Data Mart falsch? Basisdatenbank unvollständig? ETL Prozeduren fehlerhaft?
- ETL und DC müssen nachvollziehbar sein
  - Protokollieren aller Verarbeitungsschritte
  - Wiederholbarkeit aller Aktionen bei neuen Daten / Anfragen
    - Programmieren, keine manuellen Änderungen
  - Keine Ad-Hoc Verbesserungen
  - Mühsam, aber notwendig zur Unterstützung kontinuierlicher Prozesse

Inhalt dieser Vorlesung

- ETL: Extraction, Transformation, Load
- Schemaheterogenität
- Transformation
- Datenqualität
  - Data Cleansing
  - Duplikaterkennung und der Merge/Purge Algorithmus

Duplikate

- Relationale Welt: Zwei Tupel einer Relation die identische Werte in allen Attributen besitzen
- Allgemein: Paar von Tupeln, die demselben Realweltobjekt entsprechen („synonyme“ Objekte)
- Manchmal leicht, manchmal schwer
  - Mit (globaler? lokaler?) ID: ISBN, EAN, Handelsregister
  - Ohne ID: Personen, Orte, Produkte, ...
- Viele kommerzielle Tools (für Personen und Adressen)
- Duplikate von „Duplikaterkennung“
  - Record Linkage, Object Identification, Deduplication, Entity Resolution, object reconcilation, ...

Einspielen neuer Objekte

- Duplikatvermeidung: Vor dem Einfügen eines neuen Tuples prüfen, ob das Objekt schon vorhanden ist
- Typischer Ablauf
  -
```
FOR $new IN new_cust
  SELECT count(*) INTO c FROM cust WHERE name=$new;
  if (c == 0)
    INSERT INTO cust VALUES( $new, …);
  else
    // Daten ändern
    UPDATE cust SET … WHERE name=$new;
END FOR;
```
- Effizient?
  - Für jedes **UPDATE** zwei Suchen nach **$new** in **cust**

MERGE

- Besser: MERGE („Upsert“)
  - ```
    MERGE INTO cust C
    USING (
      SELECT *
      FROM new_cust) N
    ON (C.name = N.name)
    WHEN MATCHED THEN
      UPDATE SET …
    WHEN NOT MATCHED THEN
      INSERT VALUES (…);
    ```
- Benötigt nur einen Zugriff
  - Cursor ist implizit definiert
  - Effizient, wenn Duplikaterkennung einfach ist (Schlüssel)
    ```

Duplikate: Das formale Problem

- Gegeben: Tupel $T=\{t_1,..,t_m\}$ mit Attributen $A_1,...,A_k$
  - Kein identifizierenden Schlüssel
  - Komplexere Modelle (Bäume, Graphen) möglich
- Tupel entsprechen Objekten $O=\{o_1,...,o_n\}$ $(n<m)$
- Gesucht: eine Funktion dup: $TxT \rightarrow bool$
  - dup gibt true zurück, wenn Tupel t_1 und t_2 demselben Objekt o entsprechen
- Eine Möglichkeit: Hashing, Abbildung auf Schlüssel
  - Definiere dedup: $T \rightarrow N$ und $dup(t_1,t_2)= (dedup(t_1)=dedup(t_2))$
  - Zwei Tupel bekommen dann und nur dann durch dedup() dieselbe Zahl zugeordnet, wenn sie Duplikate sind
  - dedup() kann z.B. auf einen identifizierenden Schlüssel abbilden
    - Nachname+Vorname+Alter+Augenfarbe+Geschlecht+...

Duplikaterkennung

- Vorgehen: Ähnlichkeit zweier Tupel aus der Ähnlichkeit ihrer Attributwerte berechnen
  - Z.B. (gewichtetes) Mittel
- In der Praxis verwendet man oft Domänenwissen, das man in Regeln kodiert
  - Binär

```
IF t.name=s.name AND
    phonsim(t.vorname,s.vorname)>0.5 AND
    edit(t.adresse, s.adresse)>0.9
THEN s~t;
```

  - Ähnlichkeitsfunktion

```
sim(s,t) := 0.6*edit(t.name=s.name) +
            0,2*phonsim(t.vorname,s.vorname)+
            0.7*edit(t.adresse, s.adresse);
if sim(s,t)>2.6 then s~t;
```

Transitivität

- Im strengen Sinne ist die Duplikatrelation transitiv
  - $(dup(t_1,t_2)=true \wedge dup(t_2,t_3)=true) \rightarrow dup(t_1,t_3)=true$
- Im realen Leben muss man damit vorsichtig sein
  - Man nimmt an, dass Duplikate sehr ähnlich sind
  - Ähnlichkeit misst man durch eine Funktion sim: TxT $\rightarrow$ [0,1]
  - Typisch ist ein Schwellwert: $dup_{sim}(t_1,t_2)=true$ gdw. $sim(t_1,t_2)>t$
  - Aber: Ähnlichkeit ist nicht transitiv
    - Meier, Meyer, Mayer, Bayer, Bayes, Bades, ...
- Gut überlegen, wie man mit Transitivität umgeht
  - Hängt ab vom Vertrauen in sim und in Schwellwert t

Problem: Alle Duplikate Finden

- Input
  - Tabelle mit n Tupeln
  - Ähnlichkeitsmaß sim()
- Output: Cluster äquivalenter Tupel (= Duplikate)
- Problem: Es gibt $O(n^2)$ viele Tupelpaare
  - Das dauert zu lange
  - Außerdem: Nur sehr wenige sind tatsächlich Duplikate
- Idee
  - Partitioniere Tabelle geschickt
  - Tupel werden nur noch innerhalb einer Partition verglichen
  - Sprich: dedup berechnet eine Art „Superschlüssel“

Sorted Neighborhood (Merge-Purge) [HS98]

- Algorithmus
  - Sortiere Tupel so, dass Duplikate (hoffentlich) nahe beieinander liegen
  - Merge-Phase: Fenster der Größe w über sortierte Liste schieben
    - Nur Tupel innerhalb eines Fensters vergleichen
    - Man hat also überlappende Partitionen
  - Purge-Phase: Duplikate werden verschmolzen, gelöscht, ...
  - In anderen Worten: dedup() bildet auf einen Bereich um den Rang des Objekts ab
- Komplexität (n Tupel)
  - Sortieren ist O(n*log(n))
  - Anzahl Vergleiche ist O(n*w)
    - Warum?

Schlüsselerzeugung

- Sortierschlüssel zentral für Effektivität des Verfahrens
  - Rang der Tupel bestimmt die Partitionen
- Leider ist überhaupt nicht klar, was ein guter Schlüssel zur Duplikaterkennung ist
  - Implizite Priorisierung der Attribute durch Aufbau des Schlüssels
  - Wieder: Domänenwissen

Vorname	Nachname	Adresse	ID	Schlüssel
Sal	Stolpho	123 First St.	456780	STOSAL123FRST456
Mauricio	Hernandez	321 Second Ave	123456	HERMAU321SCND123
Felix	Naumann	Hauptstr. 11	987654	NAUFEL11HPTSTR987
Sal	Stolfo	123 First Street	456789	STOSAL123FRST456

Merge

Vorname	Nachname	Adresse	ID	Schlüssel
Mauricio	Hernandez	321 Second Ave	123456	HERMAU321SCND123
Felix	Naumann	Hauptstr. 11	987654	NAUFEL11HPTSTR987
Sal	Stolpho	123 First St.	456780	STOSAL123FRST456
Sal	Stolfo	123 First Street	456789	STOSAL123FRST456

Problem

- Genauigkeit ist schlecht
  - Sortierkriterium bevorzugt immer irgendein Attribut
  - Sind erste Buchstaben wichtiger für Identität als letzte?
  - Ist Nachname wichtiger als Hausnummer?
- Lösung 1: Fenster vergrößern
  - Dominanz eines Attributes wird weniger entscheidend
  - Laufzeit verschlechtert sich linear mit Fenstergröße
  - Folge: Keine nennenswerte Verbesserung bei kleiner Vergrößerung und hohe Kosten bei großer Vergrößerung
- Anderer Vorschlag?

Multipass Verfahren

- Merge-Purge mehrmals mit verschiedenen Schlüsseln laufen lassen
- Pro Lauf können kleine w und einfache Schlüssel verwendet werden
- Duplikatbeziehungen aller Läufe sind gleichwertig
- Weniger effizientes, aber deutliches effektiveres Verfahren als Single-Pass

Literatur

- Rahm, Do (200). „Data Cleaning: Problems and Current Approaches“, IEEE Data Engineering Builletin
- Labio, W. and Garcia-Molina, H. (1996). "Efficient Snapshot Differential Algorithms for Data Warehousing". 22nd VLDB, Bombay, India. pp. 63-74
- Leser, U. and Naumann, F. (2006). „Informations-integration“, dpunkt.verlag Heidelberg
- M. Hernandez and S. Stolfo (1998). „Realworld data is dirty: Data cleansing and the merge/purge problem“. Data Mining and Knowledge Discovery, 2(1): 9-37

Selbsttest

- Was macht der SQL MERGE Befehl?
- Erklären Sie das Sorted-Neighborhood Verfahren. Was hat es für eine Komplexität? In was messen Sie Komplexität?
- Wie funktioniert externes Sortieren?
- Welche Varianten der Datenversorgung eines DWH gibt es?
- Warum kann das linear sein, obwohl doch n*log(n) die beweisbare untere Schranke für Sortieren ist?
- Warum sind BULK-Loader so viel schneller? Gibt es sowas bei MySQL?