

Teil IV

Extraktion, Transformation, Laden

Extraktion, Transformation und Laden

- 1 ETL-Prozess
- 2 Extraktion von Daten aus Quellen
- 3 Differentielle Snapshots
- 4 Laden von Daten
- 5 Transformationsaufgaben
- 6 Schematische Heterogenität
- 7 Datenfehler

ETL: Überblick

- Zwei Schritte
 - ▶ 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 Tagging
 - ★ Erstellung integrierter Datenbestände
 - ▶ Kontinuierliche Datenversorgung des DWH
 - ▶ Sicherung der DWH Konsistenz bzgl. Datenquellen
- Effiziente Methoden essentiell → Sperrzeiten minimieren
- Rigorose Prüfungen essentiell → Datenqualität sichern

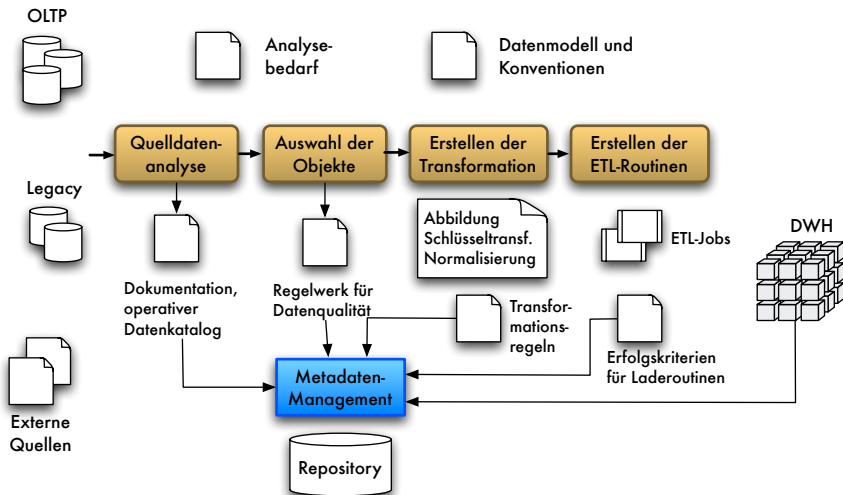
ETL-Prozess

- Häufig aufwendigster Teil des Data Warehousing
 - ▶ Vielzahl von Quellen
 - ▶ Heterogenität
 - ▶ Datenvolumen
 - ▶ Komplexität der Transformation
 - ★ Schema- und Instanzintegration
 - ★ Datenbereinigung
 - ▶ Kaum durchgängige Methoden- und Systemunterstützung, jedoch Vielzahl von Werkzeugen vorhanden

ETL-Prozess

- **Extraktion:** Selektion eines Ausschnitts der Daten aus den Quellen und Bereitstellung für Transformation
- **Transformation:** Anpassung der Daten an vorgegebene Schema- und Qualitätsanforderungen
- **Laden:** physisches Einbringen der Daten aus dem Datenbeschaffungsbereich in das Data Warehouse (einschl. eventuell notwendiger Aggregationen)

Definitionsphase des ETL-Prozesses



Extraktion

- **Aufgabe**
 - ▶ Regelmäßige Extraktion von Änderungsdaten aus Quellen
 - ▶ Datenversorgung des DWH
- **Unterscheidung**
 - ▶ Zeitpunkt der Extraktion
 - ▶ Art der extrahierten Daten

Zeitpunkt

- **Synchrone** Benachrichtigung
 - ▶ Quelle propagiert jede Änderung
- **Asynchrone** Benachrichtigung
 - ▶ Periodisch
 - ★ Quellen erzeugen regelmäßig Extrakte
 - ★ DWH fragt regelmäßig Datenbestand ab
 - ▶ Ereignisgesteuert
 - ★ DWH erfragt Änderungen vor jedem Jahresabschluss
 - ★ Quelle informiert alle X Änderungen
 - ▶ Anfragegesteuert
 - ★ DWH erfragt Änderungen vor jedem tatsächlichen Zugriff

Art der Daten

- **Flow**: alle Änderungen im DWH integrieren
 - ▶ Verkaufspositionen, Lieferungen
 - ▶ Änderungen mit aufnehmen
- **Stock**: Zeitpunkt ist essentiell muss festgelegt werden
 - ▶ Mitarbeiteranzahl zum Monatsende einer Filiale
 - ▶ Lagerbestand zum Jahresende
- **Value per Unit**: Abhängig von Unit und anderen Dimensionen
 - ▶ Währungskurs zu einem Zeitpunkt
 - ▶ Goldpreis an einem Börsenplatz

Art der Daten

- **Snapshots**: Quelle liefert immer kompletten Datenbestand
 - ▶ Neuer Lieferantenkatalog, neue Preisliste, etc.
 - ▶ Änderungen erkennen
 - ▶ Historie korrekt abbilden
- **Logs**: Quelle liefert jede Änderung
 - ▶ Transaktionslogs, Anwendungsgesteuertes Logging
 - ▶ Änderungen effizient einspielen
- **Netto-Logs**: Quelle liefert Netto-Änderungen
 - ▶ Katalogupdates, Snapshot-Deltas
 - ▶ Keine vollständige Historie möglich
 - ▶ Änderungen effizient einspielbar

Zeitpunkt der Datenversorgung

Quelle ...		Technik	Aktualität DWH	Belastung DWH	Belastung Quellen
erstellt periodisch Files		Batchläufe, Snapshots	Je nach Frequenz	Niedrig	Niedrig
propagiert jede Änderung		Trigger, Replikation	Maximal	Hoch	Sehr hoch
erstellt Extrakte auf Anfrage	vor Benutzung	Sehr schwierig	Maximal	Medium	Medium
	Anwendungs-gesteuert	Anwendungs-gesteuert	Je nach Frequenz	Je nach Frequenz	Je nach Frequenz

Zeitpunkt der Datenversorgung

Quelle ...		Technik	Aktualität DWH	Belastung DWH	Belastung Quellen
erstellt periodisch Files		Viele Systeme (Mainframe) nicht online zugreifbar			
propagiert jede Änderung		Widerspricht DWH-Idee: Mehrbelastung der Quellen			
erstellt Extrakte auf Anfrage	vor Benutzung	Sehr schwierig	Maximal	Medium	Medium
	Anwendungs-gesteuert	gesteuert	Frequenz	Frequenz	Frequenz

Extraktion aus Legacy-Systemen

- Sehr anwendungsabhängig
- Zugriff auf Host-Systeme ohne Online-Zugriff
 - ▶ Zugriff über BATCH, Reportwriter, Scheduling
- Daten in Non-Standard-Datenbanken ohne APIs
 - ▶ Programmierung in PL-1, COBOL, Natural, IMS, ...
- Unklare Semantik, Doppelbelegung von Feldern, sprechende Schlüssel, fehlende Dokumentation, Herrschaftswissen bei wenigen
- Aber: Kommerzielle Tools vorhanden

Differential Snapshot Problem

- Viele Quellen liefern immer den vollen Datenbestand
 - ▶ Molekularbiologische Datenbanken
 - ▶ Kundenlisten, Angestelltenlisten
 - ▶ Produktkataloge
- Problem
 - ▶ Ständiges Einspielen aller Daten ineffizient
 - ▶ Duplikate müssen erkannt werden
- Algorithmen um Delta-Files zu berechnen
- Schwierig bei sehr großen Files

[Labio Garcia-Molina 1996]

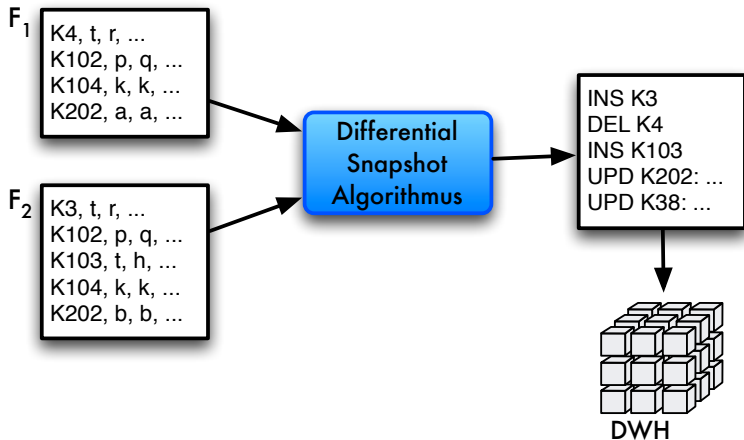
Szenario

- Quellen liefern Snapshots als File F
 - ▶ Ungeordnete Menge von Records (K, A_1, \dots, A_n)
- Gegeben: F_1, F_2 , mit $f_1 = |F_1|, f_2 = |F_2|$
- Berechne kleinste Menge $O = \{\mathbf{INS}, \mathbf{DEL}, \mathbf{UPD}\}^*$ mit $O(F_1) = F_2$
- O nicht eindeutig!

$$O_1 = \{(\mathbf{INS}(X)), \emptyset, (\mathbf{DEL}(X))\} \equiv O_2 = \{\emptyset, \emptyset, \emptyset\}$$

Differential Snapshot Problem

Szenario



Annahmen

- Berechnung einer konsekutiven Folge von DS
 - ▶ Files am 1.1.2006, 1.2.2006, 1.3.2006, ...
- Kostenmodell
 - ▶ Alle Operationen im Hauptspeicher sind umsonst
 - ▶ IO zählt mit Anzahl Records: sequenzielles Lesen
 - ▶ Keine Beachtung von Blockgrößen
- Hauptspeichergröße: M (Records)
- Filegrößen $|F_x| = f_x$ (Records)
- Files i.d.R. **größer** als Hauptspeicher

DS_{naive} – Nested Loop

- Berechnung von O
 - ▶ Record R aus F_1 lesen
 - ▶ F_2 sequenziell lesen und mit R vergleichen
 - ★ R nicht in $F_2 \rightarrow O := O \cup (\mathbf{DEL}(R))$
 - ★ R in $F_2 \rightarrow O := O \cup (\mathbf{UPD}(R))$ / ignorieren
- Problem: **INS** wird nicht gefunden
 - ▶ Hilfsstruktur notwendig
 - ▶ Array mit IDs aus F_2 (on-the-fly generieren)
 - ▶ R jeweils markieren, abschließender Lauf für **INS**
- Anzahl IO: $f_1 \cdot f_2 + \delta$
- Verbesserungen?
 - ▶ Suche in F_2 abbrechen, wenn R gefunden
 - ▶ jeweils Partition mit Größe M von F_1 laden: $\frac{f_1}{M} \cdot f_2$

DS_{small} – kleine Files

- Annahme: Hauptspeicher $M > f_1$ (oder f_2)
- Berechnung von O
 - ▶ F_1 komplett lesen
 - ▶ F_2 sequenziell lesen (S)
 - ★ $S \in F_1$: $O := O \cup (\mathbf{UPD}(S))$ / ignorieren
 - ★ $S \notin F_1$: $O := O \cup (\mathbf{INS}(S))$
 - ★ S in F_1 markieren (Bitarray)
 - ▶ Abschließend: Records $R \in F_1$ ohne Markierung: $O := O \cup (\mathbf{DEL}(R))$
- Anzahl IO: $f_1 + f_2 + \delta$
- Verbesserungen
 - ▶ F_1 im Hauptspeicher sortieren \rightsquigarrow schnellerer Lookup

DS_{sort} – Sort-Merge

- Allgemeiner Fall: $M \ll f_1$ und $M \ll f_2$
- Annahme: F_1 ist sortiert
- Sortieren auf Sekundärspeicher von F_2
 - ▶ F_2 in Partitionen P_i mit $|P_i| = M$ lesen
 - ▶ P_i im Hauptspeicher sortieren und schreiben in F^i („Runs“)
 - ▶ Alle F^i mischen
 - ▶ Annahme: $M > \sqrt{|F_2|}$ → IO: $4 \cdot f_2$
- Sortiertes F_2 aufheben für nächstes DS (wird dort F_1)
 - ▶ Pro DS muss nur F_2 sortiert werden
- Berechnung von O
 - ▶ Sortierte F_1 und F_2 öffnen
 - ▶ Mischen (paralleles Lesen mit Skipping)
- Anzahl IO: $f_1 + 5 \cdot f_2 + \delta$

DS_{sort2} – Verschränkung

- Sortiertes F_1 vorhanden
- Berechnung von O
 - ▶ F_2 in Partitionen P_i mit $|P_i| = M$ lesen
 - ▶ P_i im Hauptspeicher sortieren und schreiben in F_2^i
 - ▶ Alle F_2^i mischen und gleichzeitig mit F_1 vergleichen
- Anzahl IO: $f_1 + 4 \cdot f_2 + \delta$

DS_{hash} – Partitioned Hash

- Berechnung von O

- ▶ F_2 in Partitionen P_i mit $|P_i| = M/2$ hashen
- ▶ Hashfunktion muss garantieren:

$$P_i \cap P_j = \emptyset, \quad \forall i \neq j$$

- ▶ Partitionen sind „Äquivalenzklassen“ bzgl. der Hashfunktion
- ▶ F_1 liegt noch partitioniert vor
- ▶ F_1 und F_2 wurden mit derselben Hashfunktion partitioniert
- ▶ Jeweils $P_{1,i}$ und $P_{2,i}$ parallel lesen und mischen

- Anzahl IO: $f_1 + 3 \cdot f_2 + \delta$

Warum nicht einfach ...

- UNIX diff?
 - ▶ diff erwartet / beachtet Umgebung der Records
 - ▶ Hier: Records sind völlig ungeordnet
- in der Datenbank mit SQL?
 - ▶ 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}	$f_1 \cdot f_2$	außer Konkurrenz, extra Datenstruktur notwendig
DS_{small}	$f_1 + f_2$	nur für kleine Dateien
DS_{sort2}	$f_1 + 4 \cdot f_2$	
DS_{hash}	$f_1 + 3 \cdot f_2$	überlappungsfreie Hashfunktion, Partitionsgröße schwierig zu schätzen, Verteilungsannahmen (Sampling)

- Erweiterung von DS_{hash} für „schlechtere“ Hashfunktionen bekannt

Weitere DS Verfahren

- Anzahl Partitionen / Runs größer als File-Handles des OS
 - ▶ Hierarchische externe Sortierverfahren
- Kompression: Files komprimieren
 - ▶ Größere Partitionen / Runs
 - ▶ Größere Chance, Vergleich im Hauptspeicher durchzuführen
 - ▶ In Realität schneller (Annahmen des Kostenmodells)
- „Windows“ Algorithmus
 - ▶ Annahme: Files haben eine „unscharfe“ Ordnung
 - ▶ Mischen mit Sliding Window über beide Files
 - ▶ Liefert u.U. redundante **INS-DEL** Paare
 - ▶ Anzahl IO: $f_1 + f_2$

DS mit Zeitstempel

- Annahme: Records sind (K, A_1, \dots, A_n, T)
- T : Zeitstempel der letzten Änderung
- Erstellen von O
 - ▶ Festhalten von T_{alt} : Letztes Update ($\max\{T\}$ von F_1)
 - ▶ F_2 sequenziell lesen
 - ▶ Entries mit $T > T_{alt}$ interessant
 - ▶ Aber: **INS** oder **UPD**?
- Weiteres Problem: **DEL** wird nicht gefunden
- **Zeitstempel erspart nur Attributvergleich**

Laden

- Aufgabe
 - ▶ Effizientes Einbringen von externen Daten in DWH
- Kritischer Punkt
 - ▶ Ladevorgänge blockieren unter Umständen das komplette DWH
(Schreibsperre auf Faktentabelle)
- Aspekte
 - ▶ Trigger
 - ▶ Integritätsbedingungen
 - ▶ Indexaktualisierung
 - ▶ Update oder Insert?

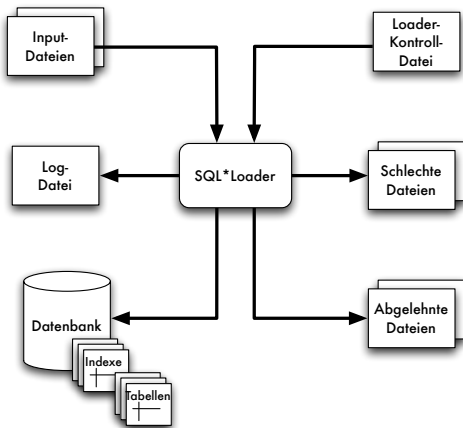
Satzbasiert

- Benutzung von Standard-Schnittstellen:
PRO*SQL, JDBC, ODBC, ...
- Arbeitet im normalen Transaktionskontext
- Trigger, Indexe und Constraints bleiben aktiv
 - ▶ Manuelle Deaktivierung möglich
- Keine großräumigen Sperren
- Sperren können durch **COMMIT** verringert werden
 - ▶ Nicht bei Oracle: Leseoperationen werden nie gesperrt (MVCC)
- Benutzung von Prepared Statements
- Teilweise proprietäre Erweiterungen (Arrays) verfügbar

BULK Load

- DB-spezifische Erweiterungen zum Laden großer Datenmengen
- Läuft (meist) in speziellem Context
 - ▶ Oracle: DIRECTPATH option im Loader
 - ▶ Komplette Tabellensperre
 - ▶ Keine Beachtung von Triggern oder Constraints
 - ▶ Indexe werden erst nach Abschluss aktualisiert
 - ▶ Kein transaktionaler Kontext
 - ▶ Kein Logging
 - ▶ Checkpoints zum Wiederaufsetzen
- Praxis: BULK Uploads

Beispiel: ORACLE sqlldr



[Quelle: Oracle 11g Dokumentation]

Beispiel: ORACLE sqlldr (2)

- Control-File

```
LOAD DATA
INFILE 'bier.dat'
REPLACE INTO TABLE getraenke (
bier_name POSITION(1) CHAR(35),
bier_preis POSITION(37) ZONED(4,2),
bier_bestellgroesse POSITION(42) INTEGER,
getraenk_id "getraenke_seq.nextval"
)
```

- Datenfile: *bier.dat*

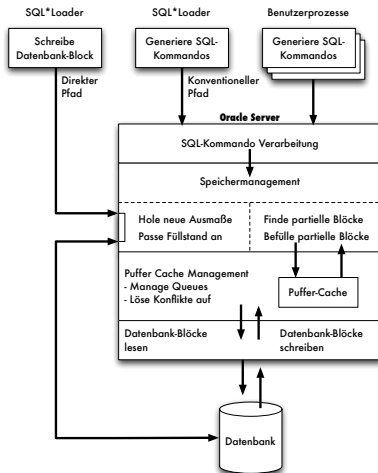
Ilmenauer Pils	4490	100
Erfurter Bock	6400	80
Magdeburger Weisse	1290	20
Anhaltinisch Flüssig	8800	200

BULK Load Beispiel

- Vielfältige Optionen

- ▶ Behandlung von Ausnahmen (Badfile)
- ▶ Datentransformationen
- ▶ Checkpoints
- ▶ Optionale Felder
- ▶ Konditionales Laden in mehrere Tabellen
- ▶ Konditionales Laden von Records
- ▶ REPLACE oder APPEND
- ▶ Paralleles Laden
- ▶ ...

Direct Path Load



[Quelle: Oracle 11g Dokumentation]

Multi-Table-Insert in Oracle

- Einfügen in mehrere Tabellen bzw. mehrfach (z.B. für Pivoting)

```
INSERT ALL  
INTO Quartal_Verkauf  
    VALUES (Produkt_Nr, Jahr || '/Q1', Umsatz_Q1)  
INTO Quartal_Verkauf  
    VALUES (Produkt_Nr, Jahr || '/Q2', Umsatz_Q2)  
INTO Quartal_Verkauf  
    VALUES (Produkt_Nr, Jahr || '/Q3', Umsatz_Q3)  
INTO Quartal_Verkauf  
    VALUES (Produkt_Nr, Jahr || '/Q4', Umsatz_Q4)  
SELECT ... FROM ...
```

Multi-Table-Insert in Oracle (2)

- Bedingtes Einfügen

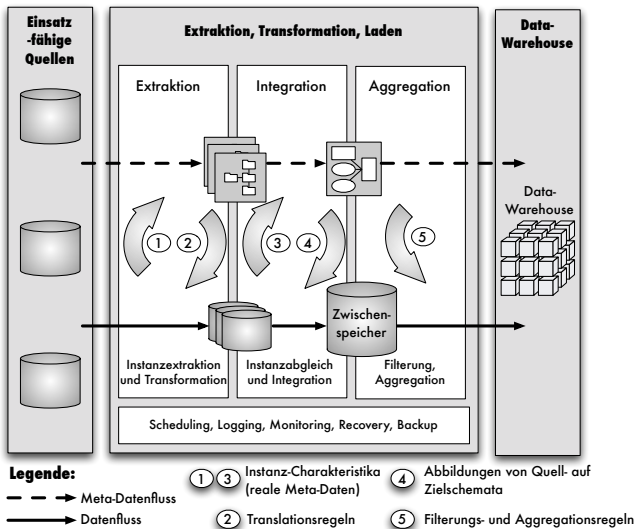
```
INSERT ALL  
WHEN ProdNr IN  
    (SELECT ProdNr FROM Werbe_Aktionen)  
INTO Aktions_Verkauf  
    VALUES (ProdNr, Quartal, Umsatz)  
WHEN Umsatz > 1000  
    INTO Top_Produnkte VALUES (ProdNr)  
SELECT ... FROM ...
```

Merge in Oracle

- Merge: Versuch eines Inserts, bei Fehler (durch Verletzung einer Schlüsselbedingung) → Update

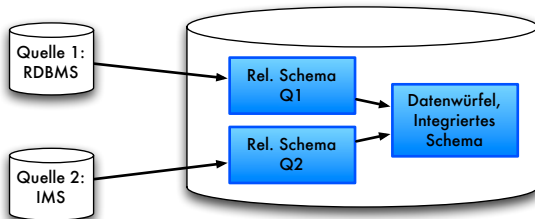
```
MERGE INTO Kunden K USING Neukunden N
ON (N.Name = K.Name AND N.GebDatum = K.GebDatum)
WHEN MATCHED THEN
UPDATE SET K.Name = N.Name, K.Vorname=N.Vorname,
    K.GebDatum=N.GebDatum
WHEN NOT MATCHED THEN
INSERT VALUES (MySeq.NextValue, N.Name,
    N.Vorname, N.GebDatum)
```

Der ETL-Prozess: Transformationsaufgaben



[Rahm Do 2000]

Technik: Quelle – Arbeitsbereich – BasisDB



- BULK Load nur für ersten Schritt
- Zweiter Schritt
 - ▶ **INSERT INTO ...SELECT ...**
 - ▶ Logging ausschaltbar
 - ▶ Parallelisierbar

Transformationsaufgaben

- Beim Laden
 - ▶ Einfache Konvertierungen (für LOAD - File)
 - ▶ Satzorientierung (Tupel)
 - ▶ Vorbereitung für BULK Loader → meist Scripte oder 3GL
- Im Datenbereitstellungsbereich
 - ▶ Mengenorientierte Berechnungen
 - ▶ Inter- und Intra-Relationenvergleich
 - ▶ Vergleich mit Basisdatenbank → Duplikate
 - ▶ Tagging der Datensätze
 - ▶ SQL

Aufgabe: Quelle – Arbeitsbereich – BasisDB

- Was macht man wo und wann?
 - ▶ Keine definierte Aufgabenteilung vorhanden

	Quelle → Arbeitsbereich	Arbeitsbereich → Basis-DB
Art des Zugriffs	Satzorientiert	Mengenorientiert
Verfügbare Datenbasen	Eine Quelle (Updatefile)	Viele Quellen
Verfügbare Datensätze	Quellabhängig: Alle, alle Änderungen, Deltas	Zusätzlich Basis-DB verfügbar
Programmiersprache	Skripte: Perl, AWK, ... oder 3GL	SQL, PL/SQL

Transformation

● Problem

- ▶ Daten im Arbeitsbereich nicht im Format der Basisdatenbank
- ▶ Struktur der Daten unterschiedlich
 - ★ Arbeitsbereich: Quellnahes Schema
 - ★ Basis-DB: Multidimensionales Schema
 - ★ Strukturelle Heterogenität

● Aspekte

- ▶ Datentransformation
- ▶ Schematransformation

Daten- und Schemaheterogenität

- Hauptdatenquelle: OLTP-Systeme
- Sekundärquellen:
 - ▶ Dokumente in firmeninternen Altarchiven
 - ▶ Dokumente im Internet via WWW, FTP
 - ★ Unstrukturiert: Zugriff über Suchmaschinen, ...
 - ★ Semistrukturiert: Zugriff über Suchmaschinen, Mediatoren, Wrapper als XML-Dokumente o.ä.

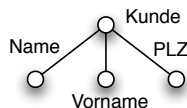
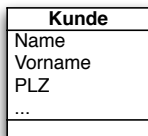
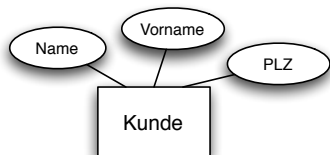
Grundproblem: Heterogenität der Quellen

Aspekte der Heterogenität

- **Verschiedene Datenmodelle**

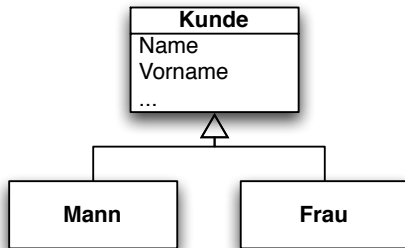
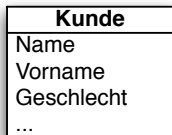
- ▶ Bedingt durch autonome Entscheidung über Anschaffung von Systemen in den Unternehmensbereichen
- ▶ Verschiedene und verschieden mächtige Modellierungskonstrukte,
- ▶ D.h. Anwendungssemantik in unterschiedlichem Ausmaß erfassbar
Abbildung zwischen Datenmodellen nicht eindeutig

- Beispiel: Relationenmodell vs. objektorientierte Modellierung vs. XML



Aspekte der Heterogenität (2)

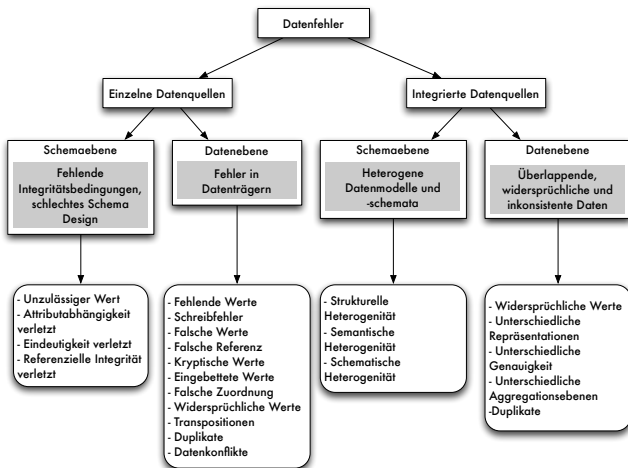
- **Unterschiedliche Modellierungen** für gleiche Sachverhalte der Realwelt
 - ▶ Bedingt durch Entwurfautonomie
 - ▶ Selbst im gleichen Datenmodell verschiedene Modellierungen möglich, z.B. durch unterschiedliche Modellierungsperspektiven der DB-Designer



Aspekte der Heterogenität (3)

- **Unterschiedliche Repräsentation der Daten**
 - ▶ Unterschiedliche Datentypen möglich
 - ▶ Unterschiedliche Umfang der unterstützten Datentypen
 - ▶ Unterschiedliche interne Darstellung der Daten
 - ▶ Auch unterschiedliche „Werte“ eines Datentyps zur Repräsentation derselben Information

Datenfehler-Klassifikation



[Rahm Do 2000, Leser Naumann 2007]

Schematische Heterogenität

- Ursache: Entwurfsautonomie \rightsquigarrow unterschiedliche Modellierung
 - ▶ Unterschiedliche Normalisierung
 - ▶ Was ist Relation, was Attribut, was Wert?
 - ▶ Aufteilung von Daten in Tabellen
 - ▶ Redundanzen aus Quellsystemen
 - ▶ Schlüssel
- In SQL nicht gut unterstützt
 - ▶ INSERT hat nur eine Zieltabelle
 - ▶ SQL greift auf Daten zu, nicht auf Schemaelemente
 - ▶ Erfordert meist Programmierung

Schema Mapping

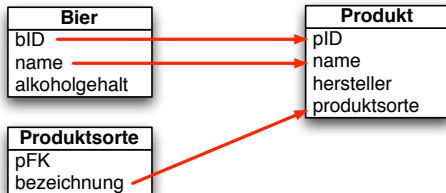
- **Datentransformation zwischen heterogenen Schemata**
 - ▶ Altes aber immer wiederkehrendes Problem
 - ▶ Üblicherweise schreiben Experten komplexe Anfragen oder Programme
 - ▶ Zeitintensiv
 - ★ Experte für die Domäne, für Schemata und für Anfrage
 - ★ XML macht alles noch schwieriger: XML Schema, XQuery
- **Idee: Automatisierung**
 - ▶ Gegeben: Zwei Schemata und ein high-level Mapping dazwischen
 - ▶ Gesucht: Anfrage zur Datentransformation

Warum ist Schema Mapping schwierig?

- Generierung der „richtigen“ Anfrage unter Berücksichtigung
 - ▶ des Quell und Ziel-Schemas,
 - ▶ des Mappings
 - ▶ und der Nutzer-Intention: **Semantik!**
- Garantie, dass die transformierten Daten dem Zielschema entsprechen
 - ▶ Flach oder geschachtelt
 - ▶ Integritätsbedingungen
- Effiziente Datentransformation

Schema Mapping: Normalisiert vs. Denormalisiert

- 1:1-Assoziationen werden unterschiedlich dargestellt
 - ▶ Durch Vorkommen im gleichen Tupel
 - ▶ Durch Fremdschlüsselbeziehung

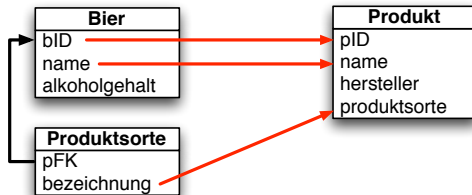


```
SELECT bID AS pID, name, NULL AS hersteller,  
       NULL AS produktsorte FROM Bier
```

```
UNION
```

```
SELECT NULL AS pID, NULL AS name, NULL AS herstelle  
       bezeichnung AS produktsorte FROM Produktsorte
```

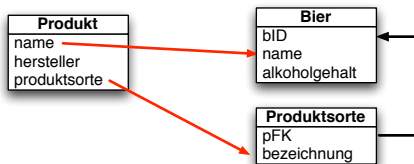
Schema Mapping: Normalisiert vs. Denormalisiert (2)



```
SELECT bID AS pID, name, NULL AS hersteller,  
        bezeichnung AS produktsorte  
FROM Bier, Produktsorte  
WHERE bID = pFK
```

Nur eine von vier möglichen Interpretationen!

Schema Mapping: Normalisiert vs. Denormalisiert (3)



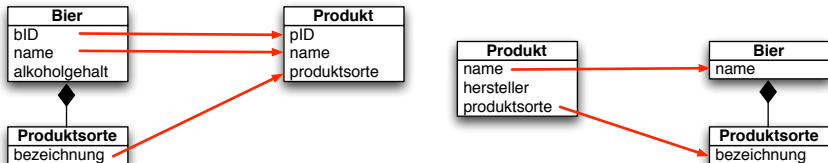
- Erfordert Generierung von Schlüsseln: Skolemfunktion SK , die einen bzgl. der Eingabe eindeutigen Wert liefert (z.B. Konkatination aller Werte)

```

Bier      := SELECT SK(name) AS bID, name,
            NULL AS alkoholgehalt FROM Produkt
Produktsorte := SELECT SK(name) AS pFK,
                produktsorte AS bezeichnung FROM Produkt
  
```

Schema Mapping: Geschachtelt vs. Flach

- 1:1-Assoziationen werden unterschiedlich dargestellt
 - ▶ D.h. geschachtelte Elemente
 - ▶ Durch Fremdschlüsselbeziehung



Schwierigkeiten

- Beispiel: Quelle (ID, Name, Strasse, PLZ, Umsatz)

- Zielschema #1

Kunde (ID, Name, Umsatz)

Adresse (ID, Strasse, PLZ)

- ▶ Erfordert 2 Durchläufe der Quelltablelle

```
INSERT INTO Kunde ... SELECT ...  
INSERT INTO Adresse ... SELECT ...
```

- Zielschema #2

PremKunde (ID, Name, Umsatz)

NormKunde (ID, Name, Umsatz)

- ▶ Erfordert 2 Durchläufe der Quelltablelle

```
INSERT INTO PremKunde ... SELECT ... WHERE Umsatz >= X  
INSERT INTO NormKunde ... SELECT ... WHERE Umsatz < X
```

Schwierigkeiten (2)

- Schema

P1 (Id, Name, Geschlecht)

P2 (Id, Name, M, W)

P31 (Id, Name), P32 (Id, Name)

- P1 → P2

```
INSERT INTO P2 (id, name, 'T', 'F') ... SELECT ...
```

```
INSERT INTO P2 (id, name, 'F', 'T') ... SELECT ...
```

- P3 → P1

```
INSERT INTO P1 (id, name, 'weiblich') ...
```

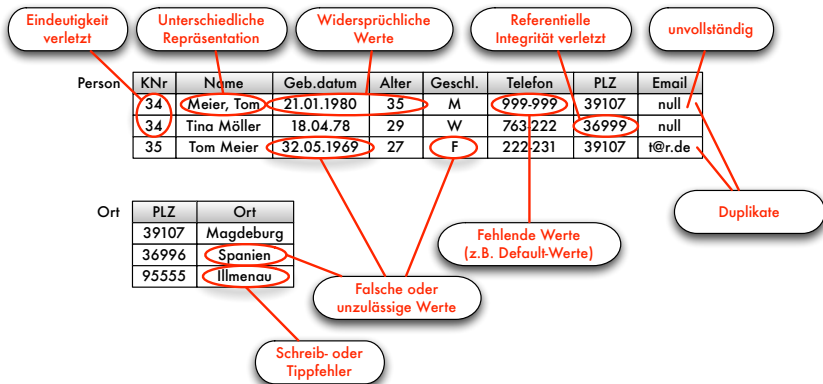
```
  SELECT ... FROM P31
```

```
INSERT INTO P1 (id, name, 'männlich') ...
```

```
  SELECT ... FROM P32
```

- Anzahl Werte muss feststehen; Neues Geschlecht – Alle Anfragen ändern

Datenfehler

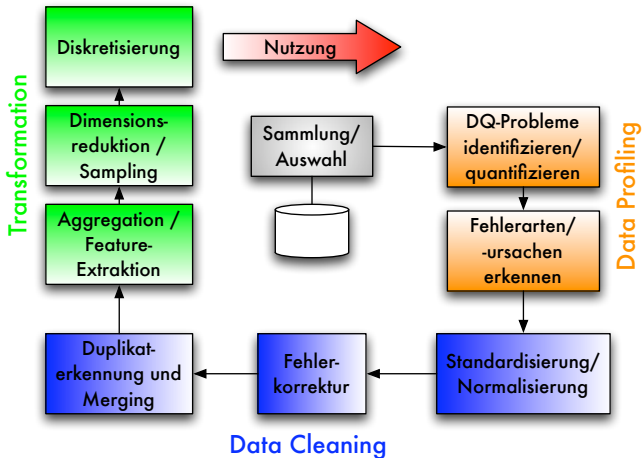


Vermeidung von Datenfehlern

Vermeidung von	durch
falschen Datentypen	Datentypdefinition, domain-Constraints
falschen Werte	check
fehlenden Werten	not null
ungültigen Referenzen	foreign key
Duplikaten	unique, primary key
Inkonsistenzen	Transaktionen
veralteten Daten	Replikation, materialisierte Sichten

- Dennoch in der Praxis:
 - ▶ Fehlen von Metadaten, Integritätsbedingungen, ...
 - ▶ Eingabefehler, Unkenntnis, ...
 - ▶ Heterogenität
 - ▶ ...

Phasen der Datenaufbereitung



Data Profiling

- Analyse von Inhalt und Struktur einzelner Attribute
 - ▶ Datentyp, Wertebereich, Verteilung und Varianz, Vorkommen von Nullwerten, Eindeutigkeit, Muster (z.B. dd/mm/yyyy)
- Analyse von Abhängigkeiten zwischen Attributen einer Relation
 - ▶ „unscharfe“ Schlüssel
 - ▶ Funktionale Abhängigkeiten, potenzielle Primärschlüssel, „unscharfe“ Abhängigkeiten
 - ▶ Notwendigkeit:
 - ★ Keine expliziten Integritätsbedingungen spezifiziert
 - ★ Jedoch in Daten in den meisten Fällen erfüllt
- Analyse von Überlappungen zwischen Attributen verschiedener Relationen
 - ▶ Redundanzen, Fremdschlüsselbeziehungen

Data Profiling (2)

- Fehlende bzw. falsche Werte
 - ▶ Ermittelte vs. Erwartete Kardinalität (z.B. Anzahl von Filialen, Geschlecht von Kunden)
 - ▶ Anzahl der Nullwerte, Minimum / Maximum, Varianz
- Daten- bzw. Eingabefehler
 - ▶ Sortierung und manuelle Prüfung
 - ▶ Ähnlichkeitstests
- Duplikate
 - ▶ Tupelanzahl vs. Attributkardinalität

Data Profiling mit SQL

- SQL-Anfragen für einfache Profiling-Aufgaben
 - ▶ Schema, Datentypen: Anfragen an Schemakatalog
 - ▶ Wertebereich

```
select min(A), max(A), count(distinct A)  
from Tabelle
```

- ▶ Datenfehler, Defaultwerte

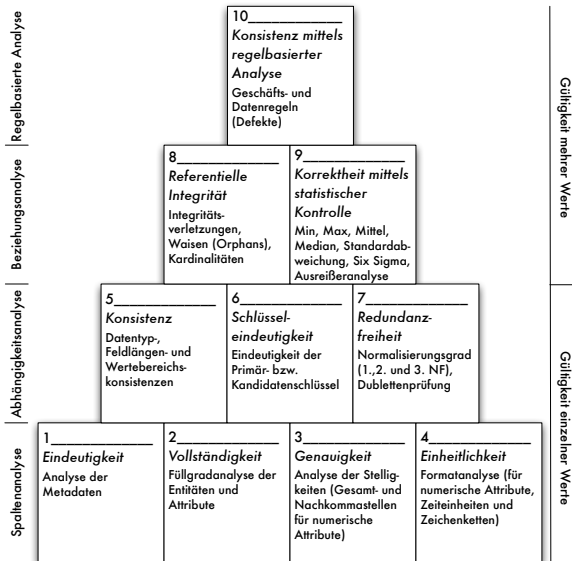
```
select Ort, count(*) as Anz  
from Kunden group by Ort order by Anz
```

- ★ Aufsteigend: Eingabefehler, z.B. Illmenau: 1, Ilmenau: 50
- ★ Absteigend: undokumentierte Default-Werte, z.B. AAA: 80

Data Cleaning

- Erkennen & Beseitigen von Inkonsistenzen, Widersprüchen und Fehlern in Daten mit dem Ziel der Qualitätsverbesserung
- Auch Cleansing oder Scrubbing
- Bis zu 80% des Aufwandes in DW-Projekten
- Cleaning im DW: Teil des ETL-Prozesses

Datenqualität und Datenbereinigung



Normalisierung und Standardisierung

- Datentypkonvertierung: varchar → int
- Kodierungen: 1: Adresse unbekannt, 2: alte Adresse, 3: gültige Adresse, 4: Adresse bei Ehepartner, ...
- Normalisierung: Abbildung in einheitliches Format
 - ▶ Datum: 03/01/11 → 01. März 2011
 - ▶ Währung: \$ → €
 - ▶ Zeichenketten in Großbuchstaben
- Zerlegung in Token: "Saake, Gunter" → "Saake", "Gunter"
- Diskretisierung numerischer Werte
- Domänenspezifische Transformationen
 - ▶ Codd, Edgar Frank → Edgar Frank Codd
 - ▶ Str. → Straße
 - ▶ Adressen über Adressdatenbanken
 - ▶ Branchenspezifische Produktbezeichnungen

Datentransformation

- In SQL gut unterstützt
 - ▶ Vielfältige Funktionen im Sprachstandard
 - ▶ Stringfunktionen, Decodierung, Datumsumwandlung, Formeln, Systemvariable, ...
 - ▶ Funktionen in PL/SQL erstellen - in SQL verwenden

- Daten

"Pause, Lilo" ⇒ "Pause", "Lilo"

"Prehn, Leo" ⇒ "Prehn", "Leo"

- SQL

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

Duplikaterkennung

- Identifikation von semantisch äquivalenten Datensätzen, d.h. die das gleiche Realwelt-Objekt repräsentieren
- Auch: Record Linkage, Object Identification, Duplicate Elimination, Merge/Purge
 - ▶ Merge: Erkennen von Duplikaten
 - ▶ Purge: Auswahl /Berechnung des „besten“ Vertreters pro Klasse

KundenNr	Name	Adresse
3346	Just Vorfan	Hafenstraße 12
3346	Justin Forfun	Hafenstr. 12
5252	Lilo Pause	Kuhweg 42
5268	Lisa Pause	Kuhweg 42
⊥	Ann Joy	Domplatz 2a
⊥	Anne Scheu	Domplatz 28

Duplikaterkennung: Vergleiche

- Typische Vergleichsregeln

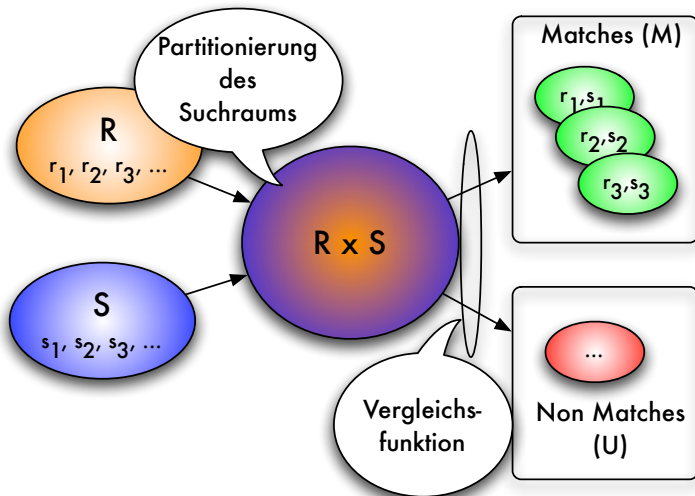
```
if ssn1 = ssn2 then match
else if name1=name2 then
    if firstname1=firstname2 then
        if adr1=adr2 then match
        else unmatched
    else if adr1=adr2 then match_household
else if adr1=adr2 then
...

```

- Naiver Ansatz: „Jeder-gegen-jeden“

- ▶ $O(n^2)$ Vergleiche
- ▶ Maximale Genauigkeit (je nach Regeln)
- ▶ Viel zu teuer

Duplikaterkennung: Prinzip

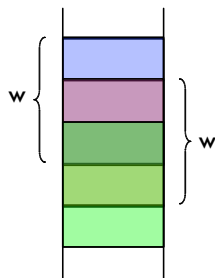


Partitionierung

- Blocking
 - ▶ Aufteilung des Suchraums in disjunkte Blöcke
 - ▶ Duplikate nur innerhalb eines Blockes
- Sortierte Nachbarschaft [Hernandez Stolfo 1998]
 - ▶ Sortierung der Daten anhand eines gewählten Schlüssels
 - ▶ Vergleiche in einem gleitenden Fenster
- Multi-Pass-Technik
 - ▶ Transitive Hülle über verschiedene Sortierungen

Sortierte Nachbarschaft

- 1 Berechne einen Schlüssel pro Datensatz
 - ▶ Bsp: SSN + „ersten 3 Zeichen von Name“ + ...
 - ▶ Beachtung typischer Fehler: 0-O, Soundex, Nachbartasten, ...
- 2 Sortiere nach Schlüssel
- 3 Laufe Liste sequenziell ab
- 4 Vergleiche innerhalb eines Fensters W , $|W| = w$
 - ▶ Mit welchen Tupeln muss wirklich verglichen werden?



• Komplexität

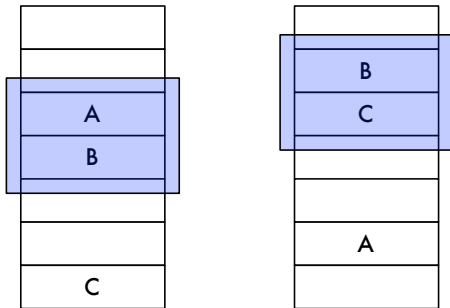
- ▶ Schlüsselerzeugung: $O(n)$, Sortieren: $O(n \cdot \log(n))$; Vergleichen: $O((n/w) \cdot (w^2)) = O(n \cdot w)$;
- ▶ Gesamt: $O(n \cdot \log(n))$ oder $O(n \cdot w)$

Sortierte Nachbarschaft: Probleme

- Genauigkeit schlecht
 - ▶ Sortierkriterium bevorzugt immer Attribute
 - ▶ Sind erste Buchstaben wichtiger für Identität als letzte?
 - ▶ Ist Nachname wichtiger als Hausnummer ?
- Window vergrößern?
 - ▶ Keine Hilfe
 - ▶ Dominanz eines Attributes bleibt gleich, aber Laufzeit verschlechtert sich schnell

Multi-Pass-Technik

- Sortieren nach mehreren Kriterien und Identifikation von Duplikaten
- Bildung der transitiven Hülle der Duplikate bis zu gegebener Länge



- 1. Lauf: „A matches B“
- 2. Lauf: „B matches C“
- Transitivität: „A matches C“

Vergleichsfunktionen

- Vergleichsfunktionen für Felder (String A und B), u.a.:
 - ▶ Editierdistanz: Anzahl der Editieroperationen (Einfügen, Löschen, Ändern) für Änderung von A in B
 - ▶ q -Grams: Vergleich der Mengen aller Teilstrings von A und B der Länge q
 - ▶ Jaro-Distanz und Jaro-Winkler-Distanz: Berücksichtigung von gemeinsamen Zeichen (innerhalb der halben Stringlänge) und transponierten Zeichen (an anderer Position)

Edit-Distanz

- Levensthein-Distanz:

- ▶ Anzahl der Editieroperationen (Einfügen, Löschen, Ändern) für Änderung von A in B
- ▶ Beispiel:

```
edit_distance("Qualität", "Quantität") = 2  
  ⇒ update(3, 'n')  
  ⇒ insert(4, 't')
```

- ▶ Anwendung:

```
select P1.Name, P2.Name  
from Produkt P1, Produkt P2  
where edit_distance(P1.Name, P2.Name) <= 2
```

q-Grams

- Menge aller Substrings der Länge q
 $Qualitt_3 := \{ _Q, _Qu, Qua, ual, ali, lit, itä, tät, ät_, t_ \}$
- Beobachtung: Strings mit kleiner Edit-Distanz haben viele gemeinsame q-Grams, d.h. für Edit-Distanz = k mind.

$$\max(|A|, |B|) - 1 - (k - 1) \cdot q$$

gemeinsame q-Grams

- Positionale q-Grams: Ergänzung um Position im String
 Qualität := $\{ (-1, _Q), (0, _Qu), (1, Qua), \dots \}$
 - ▶ Filterung für effizienten Vergleich:
 - ★ COUNT: Anzahl der gemeinsamen q-Grams
 - ★ POSITION: Positionsunterschied zwischen korrespondierenden q-Grams $\leq k$
 - ★ LENGTH: Differenz der Stringlängen $\leq k$

Datenkonflikte

- Datenkonflikt: Zwei Duplikate haben unterschiedliche Attributwerte für semantisch gleiches Attribut
 - ▶ Im Gegensatz zu Konflikten mit Integritätsbedingungen
- Datenkonflikte entstehen
 - ▶ Innerhalb eines Informationssystems (intra-source) und
 - ▶ Bei der Integration mehrerer Informationssysteme (inter-source)
- Voraussetzung: Duplikat, d.h. Identität schon festgestellt
- Erfordert: Konfliktauflösung (Purging, Reconciliation)

Datenkonflikte: Entstehung

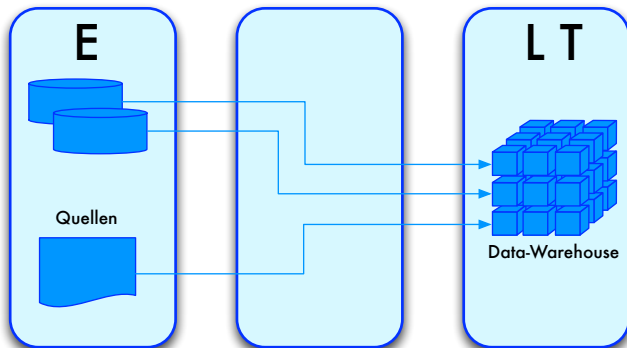
- Mangels Integritätsbedingungen oder Konsistenz-Checks
- Bei redundanten Schemata
- Durch partielle Informationen
- Bei Entstehung von Duplikaten
- Nicht korrekte Einträge
 - ▶ Tippfehler, Übertragungsfehler
 - ▶ Falsche Rechenergebnisse
- Obsolete Einträge
 - ▶ Unterschiedliche Aktualisierungszeitpunkte
 - ★ Ausreichende Aktualität einer Quelle
 - ★ Verzögerte Aktualisierung
 - ▶ Vergessene Aktualisierung

Datenkonflikte: Behebung

- Referenztabelle für exakte Wertabbildung
 - ▶ Z.B. Städte, Länder, Produktnamen, Codes...
- Ähnlichkeitsmaße
 - ▶ Bei Tippfehlern, Sprachvarianten (Meier, Mayer,...)
- Standardisieren und Transformieren
- Nutzung von Hintergrundwissen (Metadaten)
 - ▶ Z.B. Konventionen (landestypische Schreibweisen)
 - ▶ Ontologien, Thesauri, Wörterbücher zur Behandlung von Homonymen, Synonymen, ...
- Bei der Integration
 - ▶ Präferenzordnung über Datenquellen nach Aktualität, Trust (Vertrauen), Öffnungszeiten usw.
 - ▶ Konfliktlösungsfunktionen

ETL vs. ELT

- **ELT = Extract-Load-Transform**
 - ▶ Variante des ETL-Prozesses, bei dem die Daten erst nach dem Laden transformiert werden
 - ▶ Ziel: Transformation mit SQL-Anweisungen in der Zieldatenbank
 - ▶ Verzicht auf spezielle ETL-Engines



ELT

● Extraktion

- ▶ Für Quellsysteme optimierte Abfragen (z.B. SQL)
- ▶ Extraktion ebenfalls mit Monitoren überwacht
- ▶ Automatische Extraktion schwieriger (z.B. bei Datenstrukturänderungen)

● Laden

- ▶ Parallele Verarbeitung der SQL-Statements
- ▶ Bulk Load (Annahme: keine Schreibzugriffe im Zielsystem)
- ▶ Keine satzbasierte Protokollierung

● Transformation

- ▶ Ausnutzung von Mengenoperationen der DW-Transformationskomponente
- ▶ Komplexe Transformationen mittels prozeduraler Sprachen (z.B. PL/SQL)
- ▶ Spezifische Statements (z.B. CTAS von Oracle)

Zusammenfassung

- ETL als Prozess der Überführung von Daten aus Quellsystemen in das DWH
- Themen von ETL und Datenqualität machen typischerweise 80% des Aufwands von DWH-Projekten aus!
 - ▶ Langsame Anfragen sind ärgerlich
 - ▶ Falsche Ergebnisse machen das DWH nutzlos
- Teil des Transformationsschrittes
 - ▶ Schemaebene: Schema Mapping bzw. Schematransformation
 - ▶ Instanzebene: Datenbereinigung