# ETL: Overview

- Two steps
  - From the sources to staging area
    - Extraction of data from the sources
    - Creation / detection of differential updates
    - Creating LOAD Files
  - From the staging area to the base database
    - Data Cleaning and Tagging
    - Preparation of integrated data sets
  - Continuous data provision for the DWH
  - Assurance of consistency regarding DWH data sources
- Efficient methods essential → minimize offline time
- Rigorous tests essential → ensure data quality

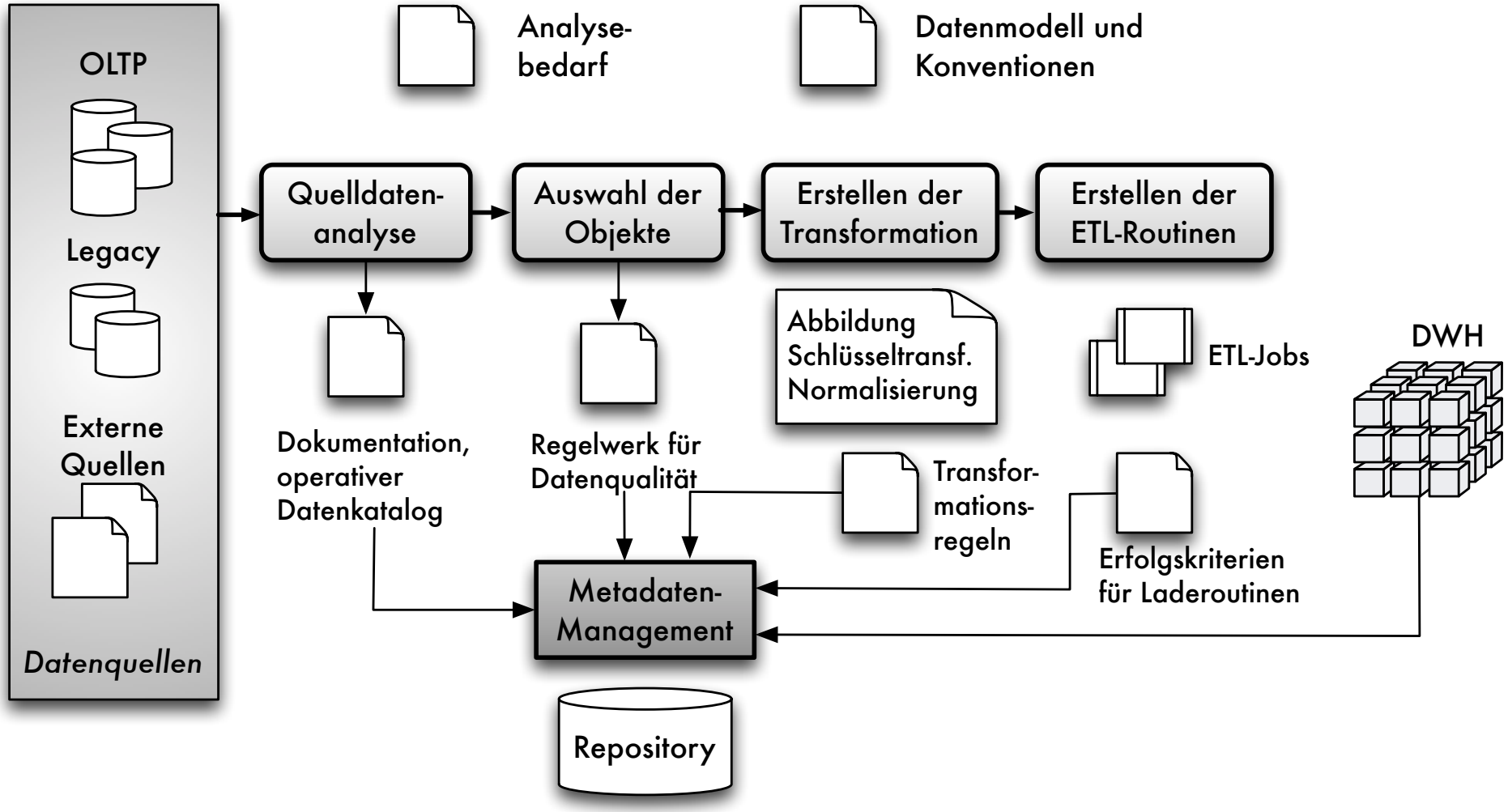
# ETL Process

- Frequently most elaborate part of the Data Warehousing
  - Variety of sources
  - Heterogeneity
  - Data volume
  - Complexity of the transformation
    - Schema and instance integration
    - Data cleansing
  - Hardly consistent methods and system support, but variety of tools available

# ETL Process

- Extraction: selecting a section of the data from the sources and providing it for transformation
- Transformation: fitting the data to predefined schema and quality requirements
- Load: physical insertion of the data from the staging area into the data warehouse (including necessary aggregations)

# Definition Phase of the ETL Process



# Extraction

- Task
  - Regular extraction of change data from sources
  - Data provision for the DWH
- Distinction
  - Time of extraction
  - Type of extracted data

# Point in Time

- Synchronous notification
  - Source propagates each change
- Asynchronous notification
  - Periodically
    - Sources produce extracts regularly
    - DWH regularly scans dataset
  - Event-driven
    - DWH requests changes before each annual reporting
    - Source informs after each X changes
  - Query-controlled
    - DWH queries for changes before any actual access

# Type of Data

- Flow: integrate all changes in DWH
  - Short positions, trade
  - accomodate for changes
- Stock: point in time is essential, must be set
  - Number of employees at end of the month in a store
  - Stock at the end of the year
- Value per Unit: Depending on unit and other dimensions
  - Exchange rate at a point in time
  - Gold price on a stock exchange

# Type of Data

- Snapshots: Source always provides complete data set
  - New suppliers directory, new price list, etc.
  - Detect changes
  - Depict history correctly
- Logs: Source provides any change
  - Transaction logs, application-controlled logging
  - Import changes efficiently
- Net Logs: Source provides net changes
  - Catalog updates, snapshot deltas
  - No complete history possible
  - Changes efficiently importable

# Point in Time of Data Provision

| **Source ...** | | **Method** | **Timeliness DWH** | **Workload on DWH** | **Worload on Sources** |
|---|---|---|---|---|---|
| creates files periodically | | Batch runs, Snapshots | depending on frequency | low | low |
| propagates each change | | Trigger, Replication | maximum | high | very high |
| creates extracts on request | before use | very hard | maximum | medium | medium |
| | application-driven | application-driven | depending on frequency | depending on frequency | depending on frequency |

# Point in Time of Data Provision

Comments for three previous options:

- Many systens (Mainframe) not accessible online
- Contradicts idea of DWH: More workload on sources
- Technically not efficiently implementable

# Extraction from Legacy Systems

- Very dependent on the application
- Access to host systems without online access
  - Access via BATCH, Report Writer, scheduling
- Data in non-standard databases without APIs
  - Programming in PL-1, COBOL, Natural, IMS ...
- Unclear semantics, double occupancy of fields, speaking keys, missing documentation, domain knowledge only held by few people
- But: Commercial tools available

# Differential Snapshot Problem

- Many sources provide only the full dataset
  - Molecular biological data bases
  - Customer lists, employee lists
  - Product catalogues
- Problem
  - Repeated import of all data is inefficient
  - Duplikates need to be detected
- Algorithms to compute Delta-Files
- Hard for very large files

[Labio Garcia-Molina 1996]

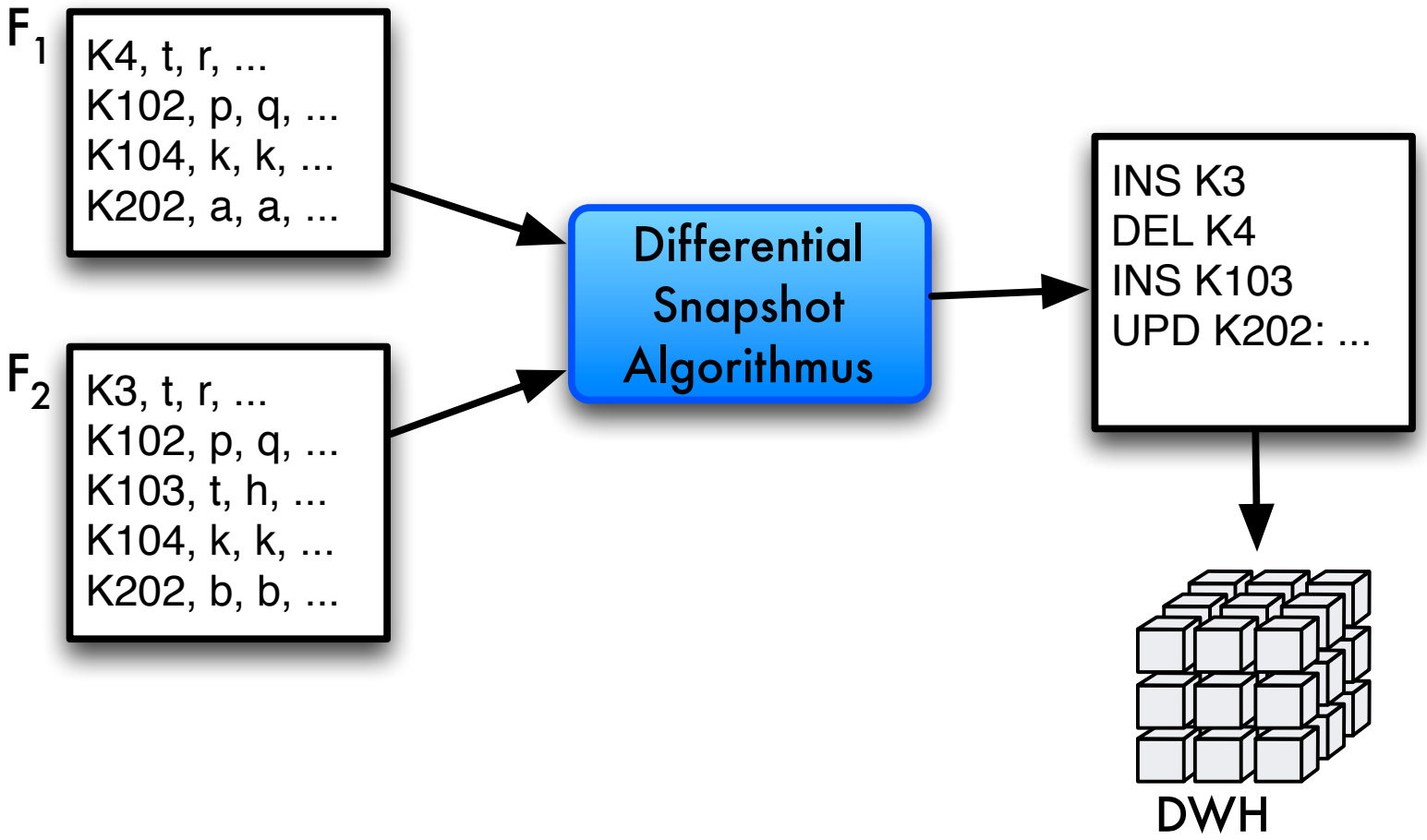
# Scenario

- Sources provide Snapshots as file $F$
  - Unordered set of records $(K, A_1, \ldots, A_n)$
- Given: $F_1$, $F_2$, mit $f_1 = |F_1|, f_2 = |F_2|$
- Calculate smallest set $O = \{\texttt{INS}, \texttt{DEL}, \texttt{UPD}\}^*$ with $O(F_1) = F_2$
- $O$ not unique!

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

**Differential Snapshot Problem**

# Scenario


F1
K4, t, r, ...
K102, p, q, ...
K104, k, k, ...
K202, a, a, ...
F2
K3, t, r, ...
K102, p, q, ...
K103, t, h, ...
K104, k, k, ...
K202, b, b, ...
Differential Snapshot Algorithmus
INS K3
DEL K4
INS K103
UPD K202: ...
DWH

# Assumptions

- Computing a consecutive order of DS
  - Files from 1.1.2010, 1.2.2010, 1.3.2010, ...
- Cost Model
  - All operations in the main memory are for free
  - IO counts the number of records: sequential read
  - No consideration of block sizes
- Size of main memory: $M$ (Records)
- File size $|F_x| = f_x$ (Records)
- Files generally larger than main memory

# $DS_{naive}$ – Nested Loop

- Computing $O$
  - Read record $R$ from $F_1$
  - Read $F_2$ sequentially and compare to $R$
    - $R$ not in $F_2 \rightarrow O := O \cup (\texttt{DEL}(R))$
    - $R$ in $F_2 \rightarrow O := O \cup (\texttt{UPD}(R))$ / ignore
- Problem: **INS** is not found
  - Auxiliary structure necessary
  - Array with IDs from $F_2$ (generated on the fly)
  - Mark $R$ respectively, final run for **INS**
- Number of IO operations: $f_1 \cdot f_2 + \delta$
- Improvements?
  - Cancel search in $F_2$ if $R$ has been found
  - Load partitions of size $M$ from $F_1$: $\frac{f_1}{M} \cdot f_2$

# $DS_{small}$ – small files

- Assumption: Main memory $M > f_1$ (or $f_2$)
- Computing $O$
  - Read $F_1$ completely
  - Read $F_2$ sequentially ($S$)
    - $S \in F_1$: $O := O \cup (\texttt{UPD}(S))$ / ignore
    - $S \notin F_1$: $O := O \cup (\texttt{INS}(S))$
    - Mark $S$ in $F_1$ (Bitarray)
  - Finally: Records $R \in F_1$ without marks: $O := O \cup (\texttt{DEL}(R))$
- Number of IO operations: $f_1 + f_2 + \delta$
- Improvements
  - Sort $F_1$ in the main memory $\rightsquigarrow$ faster lookup

# $DS_{sort}$ – Sort-Merge

- General case: $M \ll f_1$ und $M \ll f_2$
- Assumption: $F_1$ is sorted
- Sort $F_2$ in secondary storage
  - read $F_2$ in partitions $P_i$ with $|P_i| = M$
  - Sort $P_i$ in main memory and write in $F^i$ ("Runs")
  - Mix all $F^i$
  - Assumption: $M > \sqrt{|F_2|} \quad \rightarrow$ IO: $4 \cdot f_2$
- Keep sorted $F_2$ for next DS (becomes $F_1$ there)
  - Per DS only $F_2$ needs to be sorted
- Computing $O$
  - Open sorted $F_1$ and $F_2$
  - Mix (parallel reads with skipping)
- Number of IO operations: $f_1 + 5 \cdot f_2 + \delta$

# $DS_{sort2}$ – Interleaved

- Sorted $F_1$ given
- Computing $O$
  - Read $F_2$ in partitions $P_i$ with $|P_i| = M$
  - Sort $P_i$ in main memory and write in $F_2^i$
  - Mix all $F_2^i$ and simultaneously compare to $F_1$
- Number of IO operations: $f_1 + 4 \cdot f_2 + \delta$

# $DS_{hash}$ – Partitioned Hash

- Calculating $O$
  - Hash $F_2$ in partitions $P_i$ with $|P_i| = M/2$
  - Hash funktion has to guarantee:

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

  - Partitions are "equivalence classes" w.r.t. the hash function
  - $F_1$ is still partitioned
  - $F_1$ and $F_2$ have been partitioned by the same hash function
  - Read and mix $P_{1,i}$ and $P_{2,i}$ in parallel
- Number of IO operations: $f_1 + 3 \cdot f_2 + \delta$

# Why not simply ...

- UNIX diff?
  - diff requires / considers surroundings of records
  - Here: records are not ordered
- in the database with SQL?
  - Requires to read each relation three times

```
INSERT INTO delta
   SELECT 'UPD', ...FROM F1, F2
   WHERE F1.K = F2.K AND F1.W <> F2.W
UNION
   SELECT 'INS', ...FROM F2
   WHERE NOT EXISTS (...)
UNION
   SELECT 'DEL', ...FROM F1
   WHERE NOT EXISTS (...)
```

# Comparison – Features

| | IO | Bemerkungen |
|---|---|---|
| $DS_{naive}$ | $f_1 \cdot f_2$ | out of concurrence, auxiliary data structure required |
| $DS_{small}$ | $f_1 + f_2$ | only for smaller files |
| $DS_{sort2}$ | $f_1 + 4 \cdot f_2$ | |
| $DS_{hash}$ | $f_1 + 3 \cdot f_2$ | non-overlapping hash function, hard to estimate partition size, assumptions about distribution (Sampling) |

- Extensions of $DS_{hash}$ for "worse" hash functions known

# Further DS Approaches

- Number of partitions / runs larger than file descriptors in OS
  - Hierarchical external sorting methods
- Compression: Compress Files
  - Larger partitions / runs
  - Better chance of performing comparisons within the main memory
  - In reality faster (assumptions of the cost model)
- "Windows" Algorithm
  - Assumption: Files have a "fuzzy" order
  - Mixing with Sliding Window over both files
  - Returns many redundant **INS-DEL** pairs
  - Number of IO operations: $f_1 + f_2$

# DS with Timestamp

- Assumption: Records are $(K, A_1, \ldots, A_n, T)$
- $T$: Timestamp of the last change
- Creating $O$
  - Adherence of $T_{alt}$: Last update ($max\{T\}$ of $F_1$)
  - Read $F_2$ sequentially
  - Entries with $T > T_{alt}$ interesting
  - But: **`INS`** or **`UPD`**?
- Another problem: **`DEL`** is not found
- Timestamp spares only attribute comparison

# Load

- Aufgabe
  - Efficient incorporation of external data in DWH
- Critical Point
  - Loading operations may block the entire DWH (Write lock on fact table)
- Aspects:
  - Triggers
  - Integrity constraints
  - Index update
  - Update or Insert?

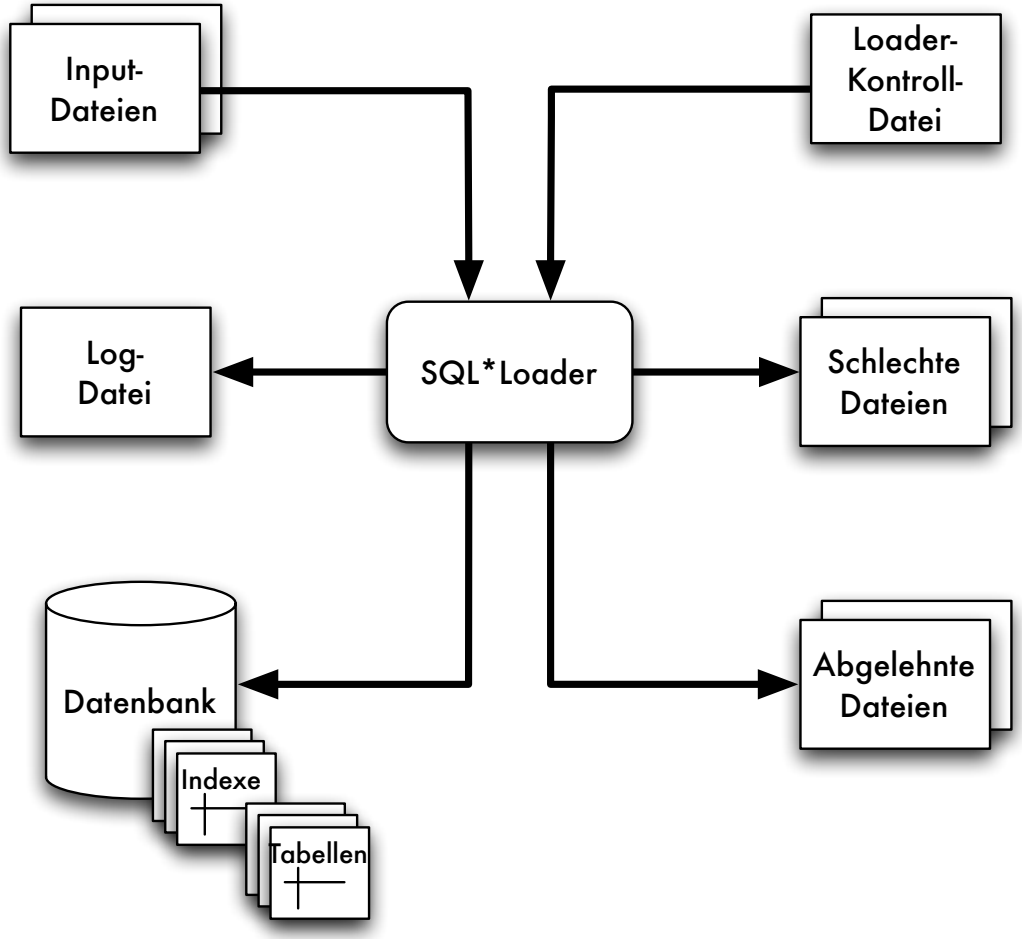
# Set based

- Use of standard interfaces:
  PRO*SQL, JDBC, ODBC, ...
- Works in the normal transaction context
- Triggers, indexes and constraints remain active
  - Manual deactivation possible
- No large-scale locks
- Locks can be reduced by **`COMMIT`**
  - Not in Oracle: Read operations are never locked (MVCC)
- Using prepared statements
- Partial proprietary extensions (arrays) available

# BULK Load

- DB-specific extensions for loading large amounts of data
- Running (usually) in a special context
  - Oracle: DIRECTPATH option in the loader
  - Complete table lock
  - No consideration of triggers or constraints
  - Indexes are not updated until after
  - No transactional context
  - No logging
  - Checkpoints for recovery
- Practice: BULK Uploads

# Example: ORACLE sqlldr




[Oracle 11g Documentation]

# Example: 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"
)
```

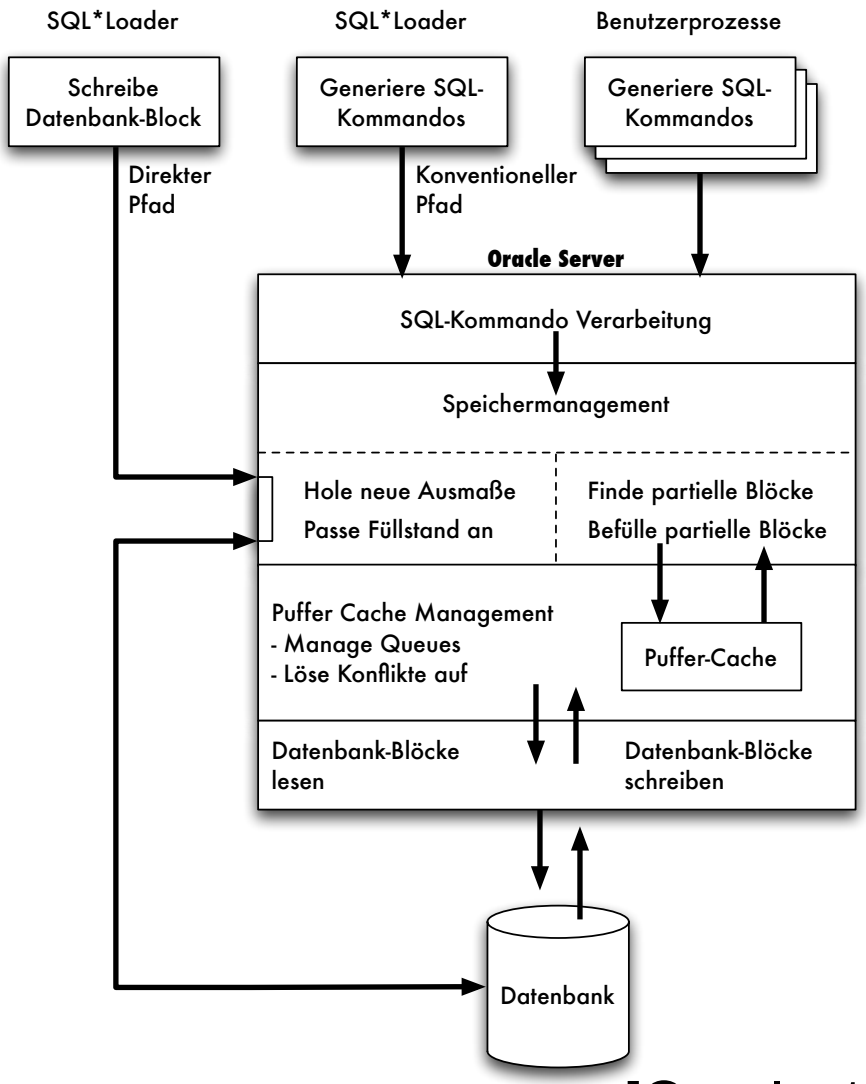
- Data file: *bier.dat*

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

# BULK Load Example

- Many options
  - Treatment of exceptions (Badfile)
  - Data transformations
  - Checkpoints
  - Optional fields
  - Conditional loading into multiple tables
  - Conditional loading of records
  - REPLACE or APPEND
  - Parallel load
  - ...

# Direct Path Load




[Oracle 11g Documentation]

# Multi-Table-Insert in Oracle

- Insert in to multiple tables or multiple times (e.g., for 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)

- Conditional insert

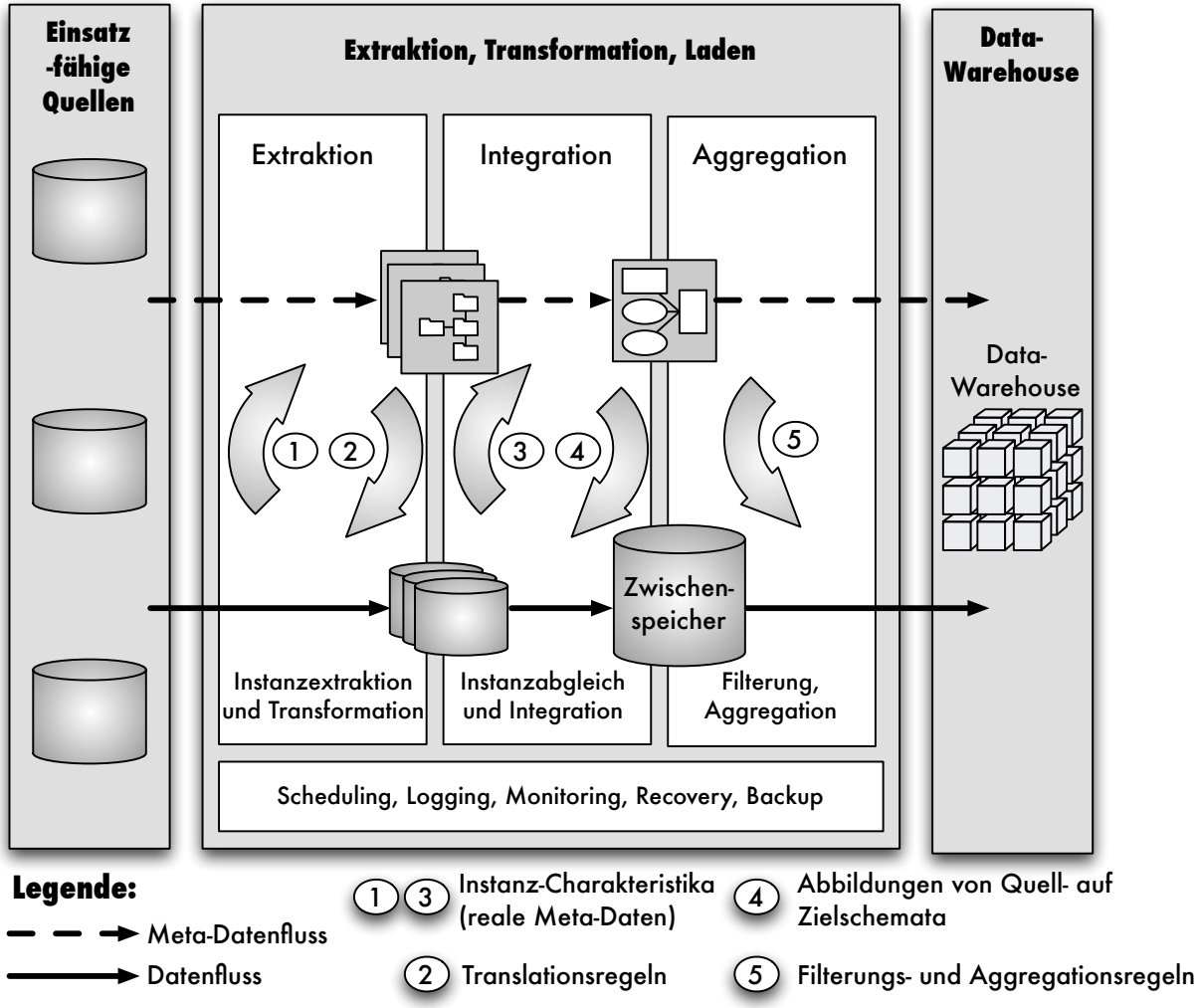
```
INSERT ALL
WHEN ProdNr IN
    (SELECT ProdNr FROM Werbe_Aktionen)
INTO Aktions_Verkauf
   VALUES (ProdNr, Quartal, Umsatz)
WHEN Umsatz > 1000
   INTO Top_Produkte VALUES (ProdNr)
SELECT ... FROM ...
```

# Merge in Oracle

- Merge: attempt an insert in error (by breach of a key condition) → Update

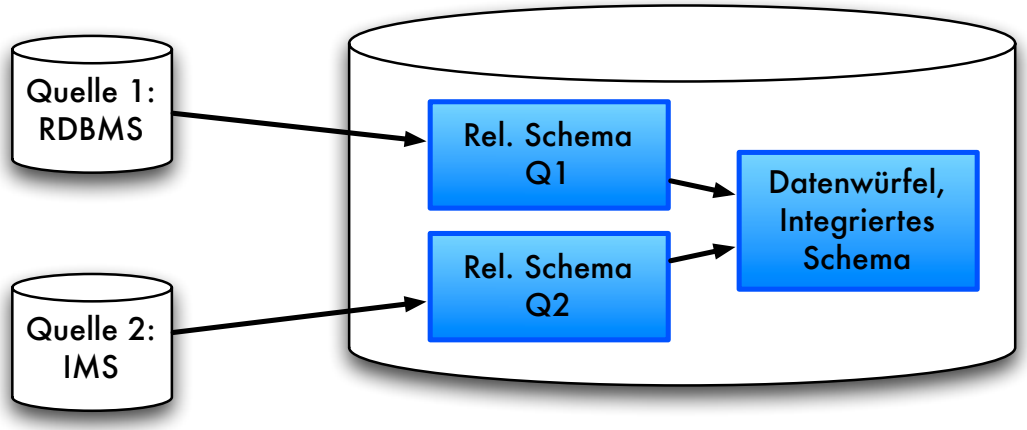
```
MERGE INTO Kunden K USING Neukunden N
ON (N.Name = K.Name AND N.GebDatum = K.GebDatum)
WHEN MATCHED THEN
UPDATE SET K.Name = N.Name, K.Vorname=N.Vorname,
   K.GebDatum=N.GebDatum
WHEN NOT MATCHED THEN
INSERT VALUES (MySeq.NextVal, N.Name,
   N.Vorname, N.GebDatum)
```

# The ETL Process: Transformation Tasks




[Rahm Do 2000]

# Method: Source – Staging Area – BaseDB




- BULK Load only the first step
- Next loads
  - `INSERT INTO ...SELECT ...`
  - Logging can be switched off
  - Parallelizable

# Transformation Tasks

- When loading
  - Simple conversions (for LOAD - File)
  - Record orientation (tuples)
  - Preparation for BULK Loader –> mostly scripts or 3GL
- In the data staging area
  - set-oriented calculations
  - Inter-and intra-relation comparison
  - Comparison with base database → Duplicates
  - Tagging of records
  - SQL
- Loading in the BaseDB
  - Bulk-Load
  - set-oriented inserts without logging

# Task: Source – Staging Area – BaseDB

- What to do, where and when?
  - No defined task assignment

| | **Extraction**<br><br>**Source → Staging Area** | **Load**<br><br>**Staging Area → BaseDB** |
|---|---|---|
| Access type | record-oriented | set-oriented |
| Available databases | one source (Updatefile) | many sources |
| Available datasets | Depending in source: sll, all changes, deltas | BaseDB additionally available |
| Programming language | Skripts: Perl, AWK, ... or 3GL | SQL, PL/SQL |

# Transformation

- Problem
  - Data in non-working area not in the format of the basic database
  - Structure of the data varies
    - Staging Area: Schema close to source
    - BaseDB: Multidimensional schema
    - Structural heterogeneity
- Aspects
  - Data transformation
  - Schema transformation

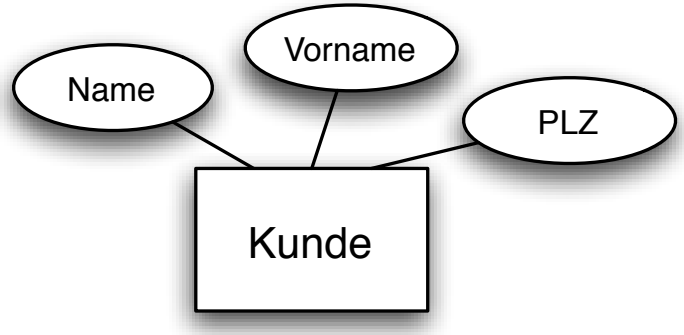
# Data and schema heterogeneity

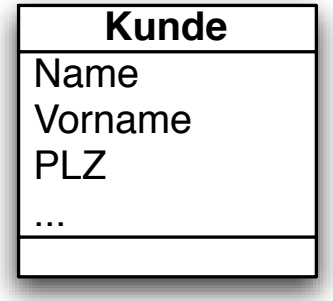
- Main data source: OLTP systems
- Secondary sources:
  - Documents from in-house old archives
  - Documents from the Internet via WWW, FTP
    - Unstructured: access via search engines, ...
    - Semi Structured: access via search engines, mediators, wrappers etc. as XML documents or similar

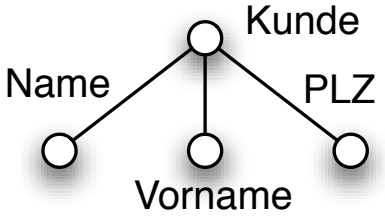
Basic problem: heterogeneity of sources

# Aspects of Heterogeneity

- Various data models
  - Due to autonomous decisions on acquisition of systems in the divisions,
  - Various and different powerful modeling constructs,
  - Application semantics are specifiable in varying degrees Mapping ambiguous between data models
- Example: Relational Model vs. object-oriented modeling vs. XML

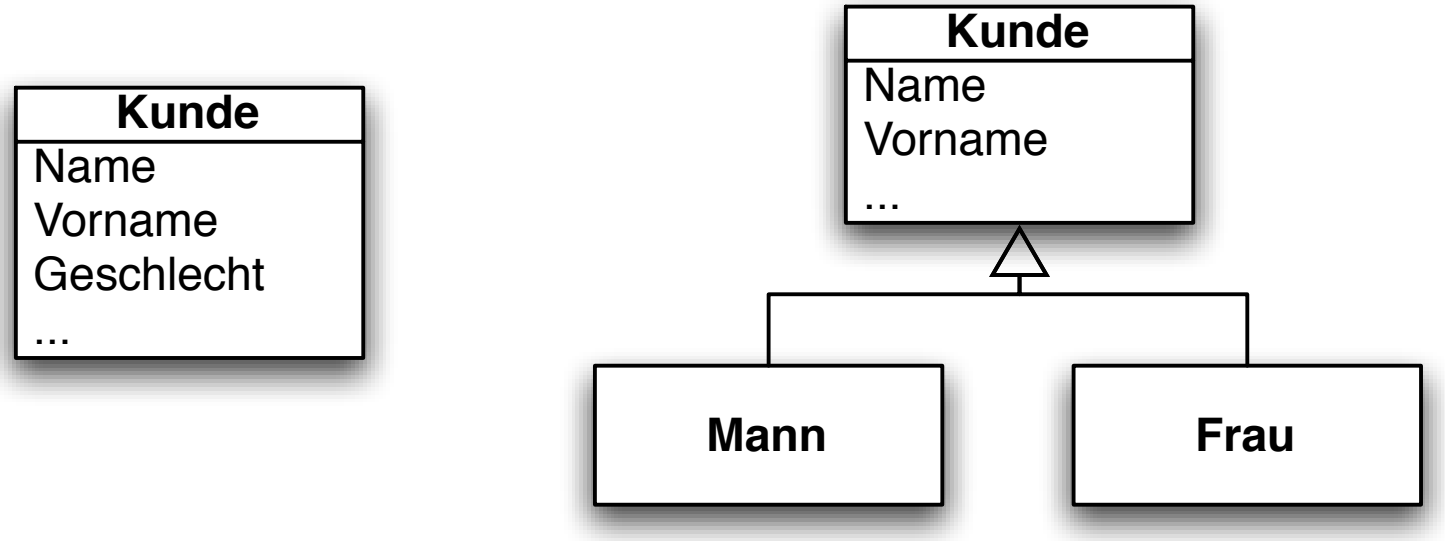


# Aspects of Heterogeneity (2)

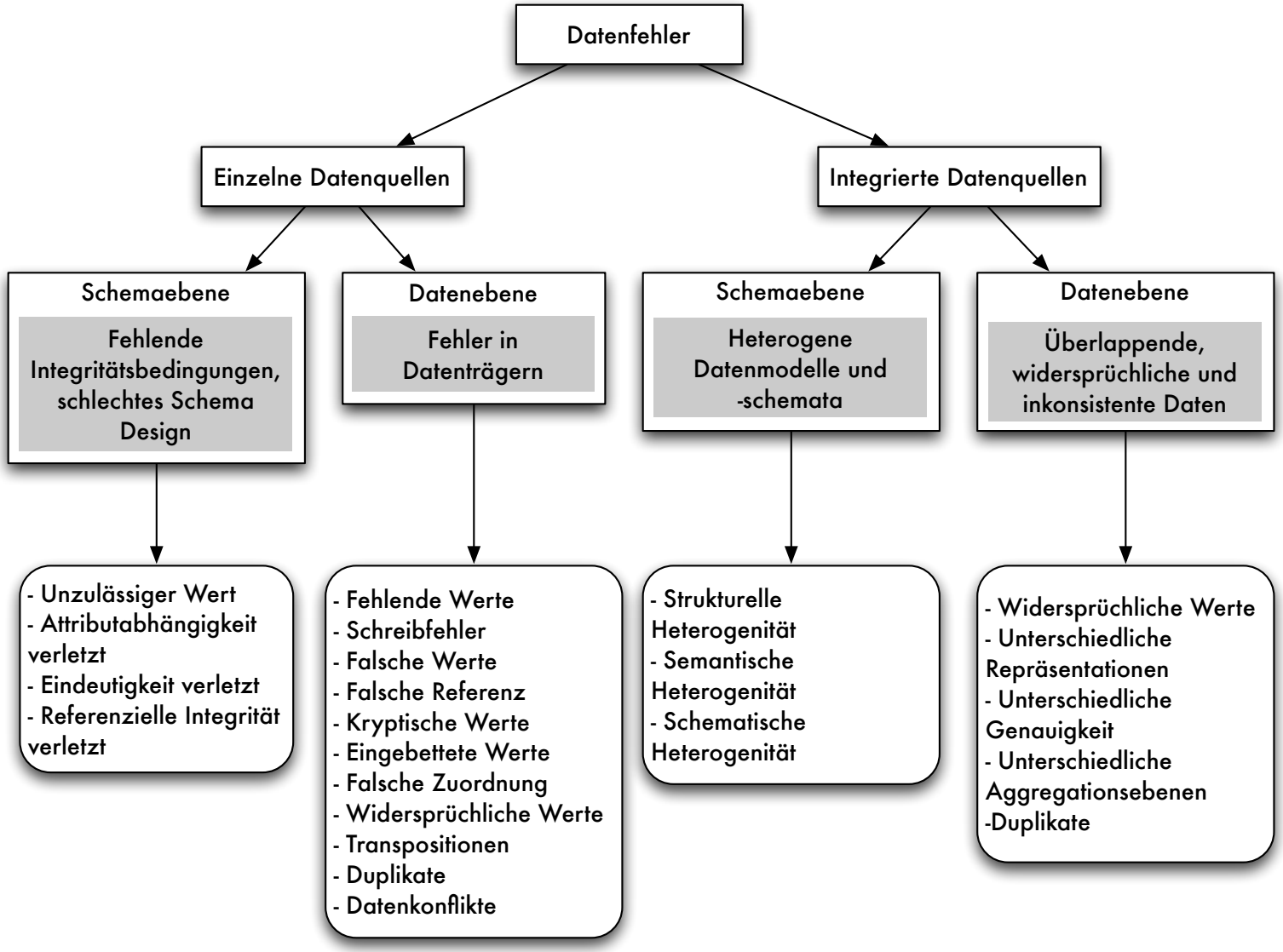
- Different models for the same real-world facts
  - Due to design autonomy
  - Even in the same data model different modeling possible, e.g., by different modeling perspectives of DB Designer



# Aspects of Heterogeneity (3)

- Different representations of the data
  - Different data types possible
  - Different scopes of the supported data types
  - Different internal representations of the data
  - Also, different "values" of a data type to represent the same information

# Data Error Classification




[Rahm Do 2000, Leser Naumann 2007]

# Schema Heterogeneity

- Cause: design autonomy $\rightsquigarrow$ different models
  - Different normalization
  - What is a relation, what is an attribute, what is a value?
  - Distribution of data in tables
  - Redundancies from source systems
  - Keys
- In SQL is not well supported
  - INSERT has only one target table
  - SQL accesses data, not schema elements
  - Usually requires programming

# Schema Mapping

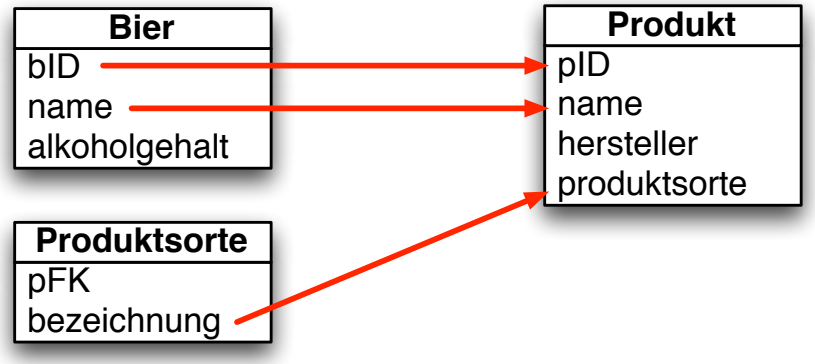
- Data transformation between heterogeneous schemas
  - Old but recurrent problem
  - Usually, experts write complex queries or programs
  - Time intensive
    - Expert for the domain, for schemas and for queries
    - XML makes it even more difficult: XML Schema, XQuery
- Idea: Automation
  - Given: Two schemas and a high-level mapping between them
  - Wanted: query for data transformation

# Why is schema mapping difficult?

- Generation of the "right" request, taking into account
  - the source and target schema
  - the mapping
  - and the user intention: semantics!
- Guarantee that the transformed data correspond to the target schema
  - Flat or nested
  - Integrity constraints
- Efficient data transformation

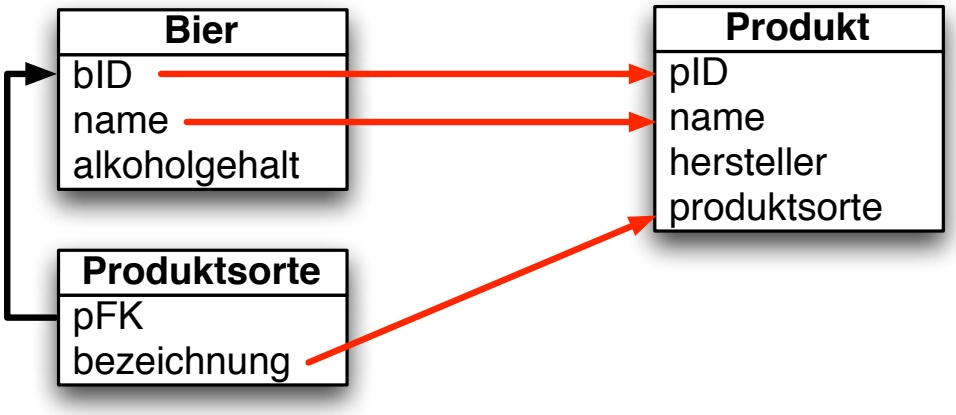
# Schema Mapping:Normalized vs. Denormalized

- 1:1 associations are represented differently
  - By occurrence in the same tuple
  - Due to foreign key relationship




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

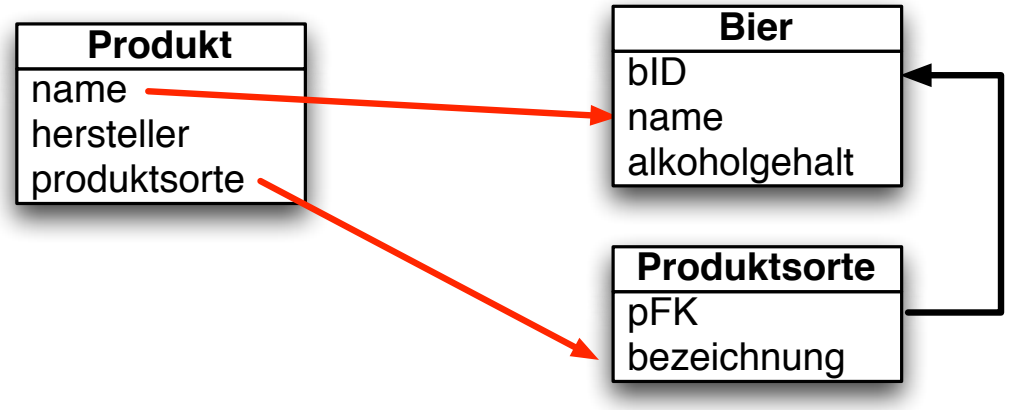
# Schema Mapping:Normalized vs. Denormalized (2)




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

Only one of four possible interpretations!

# Schema Mapping:Normalized vs. Denormalized (3)

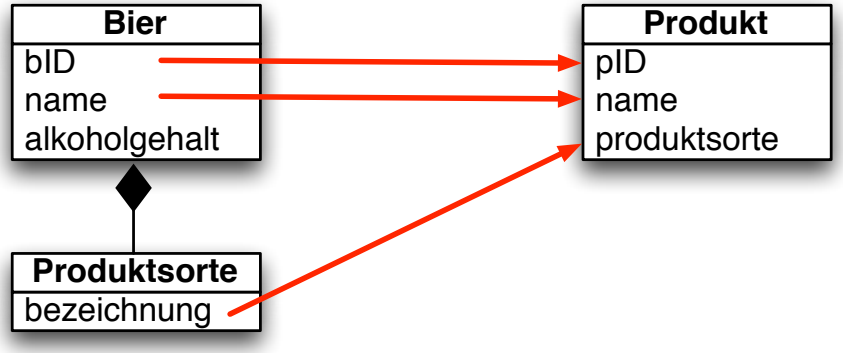


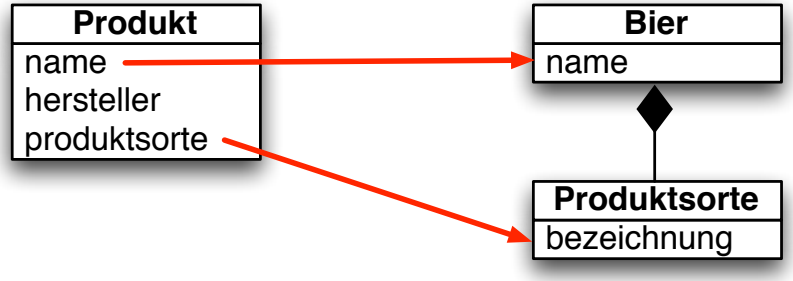

- Requires key generation: Skolem funktion $SK$, supplying a unique value with respect to the input (e.g., concatenation of all values)

```
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: Nested vs. Flat

- 1:1 associations are represented differently
  - I.e., nested elements
  - Due to foreign key relationship



# Difficulties

- Example: `Source(ID, Name, Street, ZIP-Code, Revenue)`
- Target schema #1

  `Customer(ID, Name, Revenue)`

  `Address(ID, Street, ZIP-Code)`
  - Requires 2 scans of the source table

```
INSERT INTO Customer ... SELECT ...
INSERT INTO Address ... SELECT ...
```

- Target schema #2

  `PremCustomer(ID, Name, Revenue)`

  `NormCustomer(ID, Name, Revenue)`
  - Requires 2 scans of the source table

```
INSERT INTO PremCustomer ... SELECT ... WHERE Revenue>
INSERT INTO NormCustomer ... SELECT ... WHERE Revenue<
```

# Difficulties (2)

- Schema

  `P1(Id, Name, Gender)`
  `P2(Id, Name, M, W)`
  `P31(Id, Name),P32(Id, Name)`
- P1 → P2

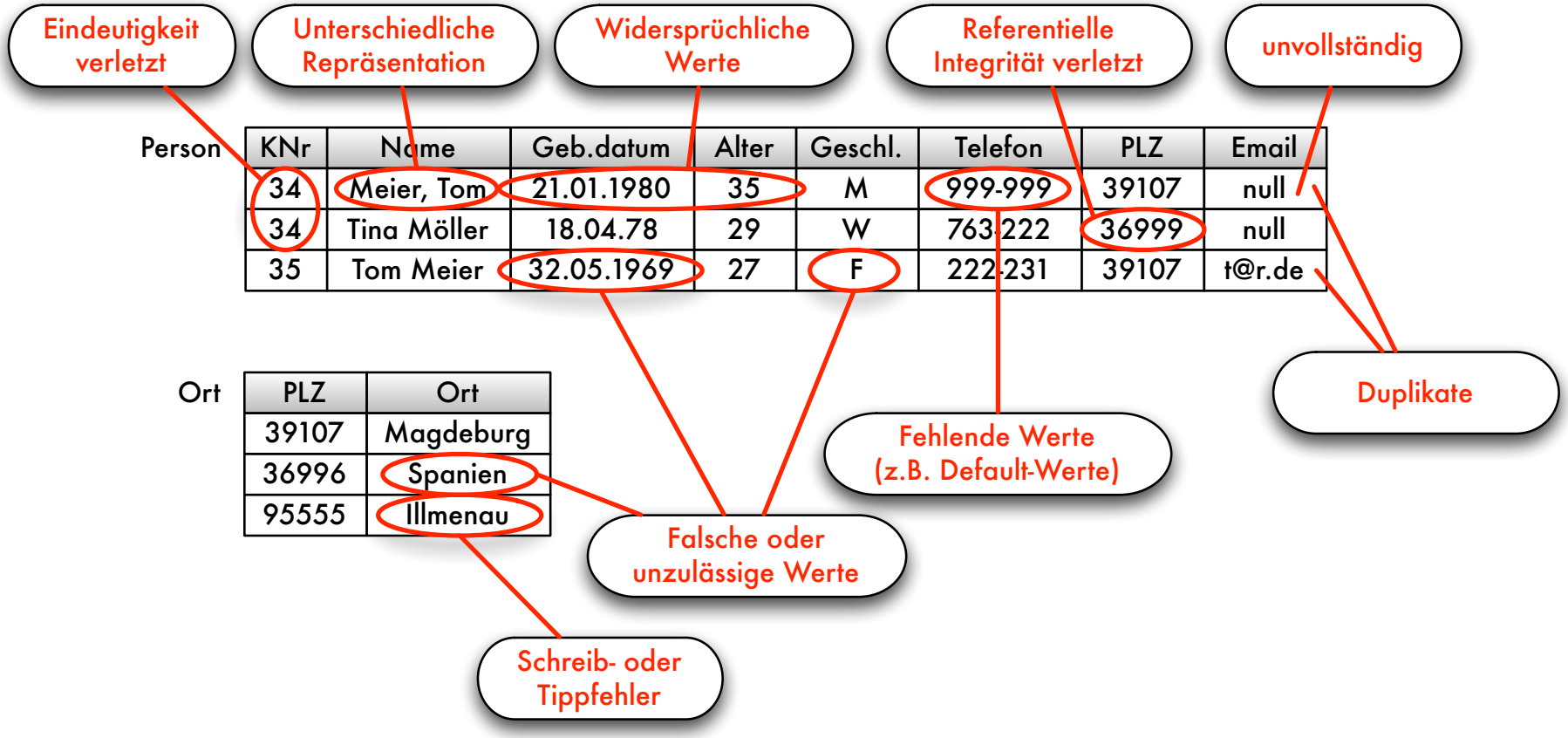
```
INSERT INTO P2 (id, name, 'T', 'F') ... SELECT ...
INSERT INTO P2 (id, name, 'F', 'T') ... SELECT ...
```

- P3 → P1

```
INSERT INTO P1(id, name, 'female') ...
   SELECT ... FROM P31
INSERT INTO P1(id, name, 'male') ...
   SELECT ... FROM P32
```

- Number of values must be fixed; new gender – Change all queries

# Data Errors

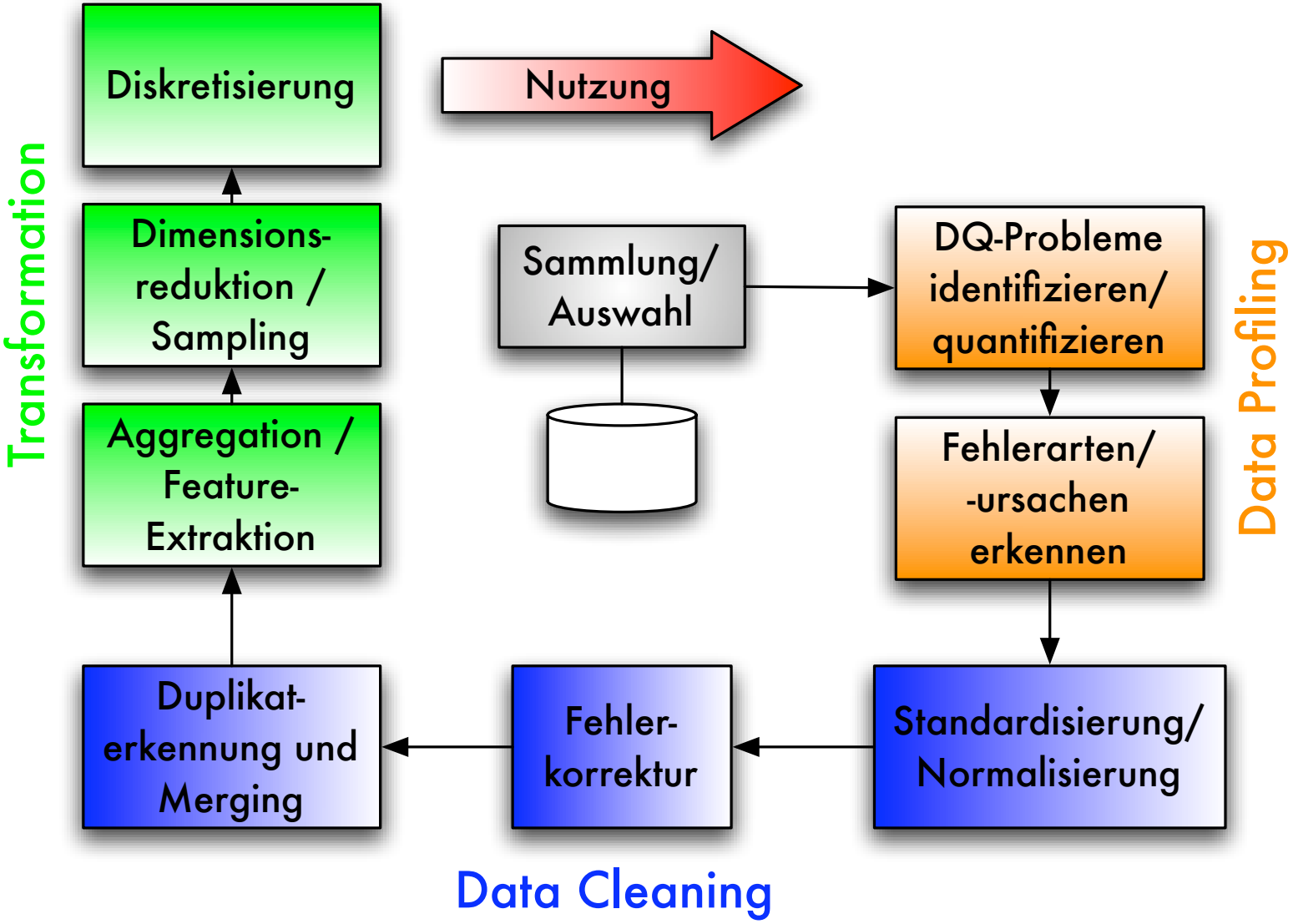

Eindeutigkeit verletzt
Unterschiedliche Repräsentation
Widersprüchliche Werte
Referentielle Integrität verletzt
unvollständig
Person
KNr | Name | Geb.datum | Alter | Geschl. | Telefon | PLZ | Email
34 | Meier, Tom | 21.01.1980 | 35 | M | 999-999 | 39107 | null
34 | Tina Möller | 18.04.78 | 29 | W | 763-222 | 36999 | null
35 | Tom Meier | 32.05.1969 | 27 | F | 222-231 | 39107 | t@r.de
Ort
PLZ | Ort
39107 | Magdeburg
36996 | Spanien
95555 | Illmenau
Duplikate
Fehlende Werte (z.B. Default-Werte)
Falsche oder unzulässige Werte
Schreib- oder Tippfehler

# Avoiding Data Errors

| **Avoiding of** | **by** |
|---|---|
| wrong data types | Data type definition, **`domain`**-constraints |
| wrong values | **`check`** |
| missing values | **`not null`** |
| invalid foreign key references | **`foreign key`** |
| Duplikates | **`unique`**, **`primary key`** |
| Inkonsistencies | transactions |
| outdated data | replikation, materialized views |

- However, in practice:
  - Lack of metadata and integrity constraints, ...
  - Input errors, ignorance, ...
  - Heterogeneity
  - ...

# Phases of Data Processing



# Data Profiling

- Analysis of the content and structure of individual attributes
  - Data type, range, distribution and variance, occurrence of null values, uniqueness, pattern (e.g., dd / mm / yyyy)
- Analysis of dependencies between attributes of a relation
  - "fuzzy" keys
  - Functional dependencies, potential primary key, "fuzzy" dependencies
  - Need:
    - No explicit constraints specified
    - However, in most data satisfied
- Analysis of overlaps between attributes of different relations
  - Redundancies, foreign key relationships

# Data Profiling (2)

- Missing or incorrect values
  - Calculated vs. Expected cardinality (e.g. number of branches, gender of clients)
  - ANumber of null values, minimum / maximum, variance
- Data or input errors
  - Sorting and manual testing
  - Similarity tests
- Duplicates
  - Number of tuples vs. attribute cardinality

# Data Profiling with SQL

- SQL queries for simple profiling tasks
  - Schema, data types: requests to schema catalog
  - Range of values

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

  - Data errors, default values

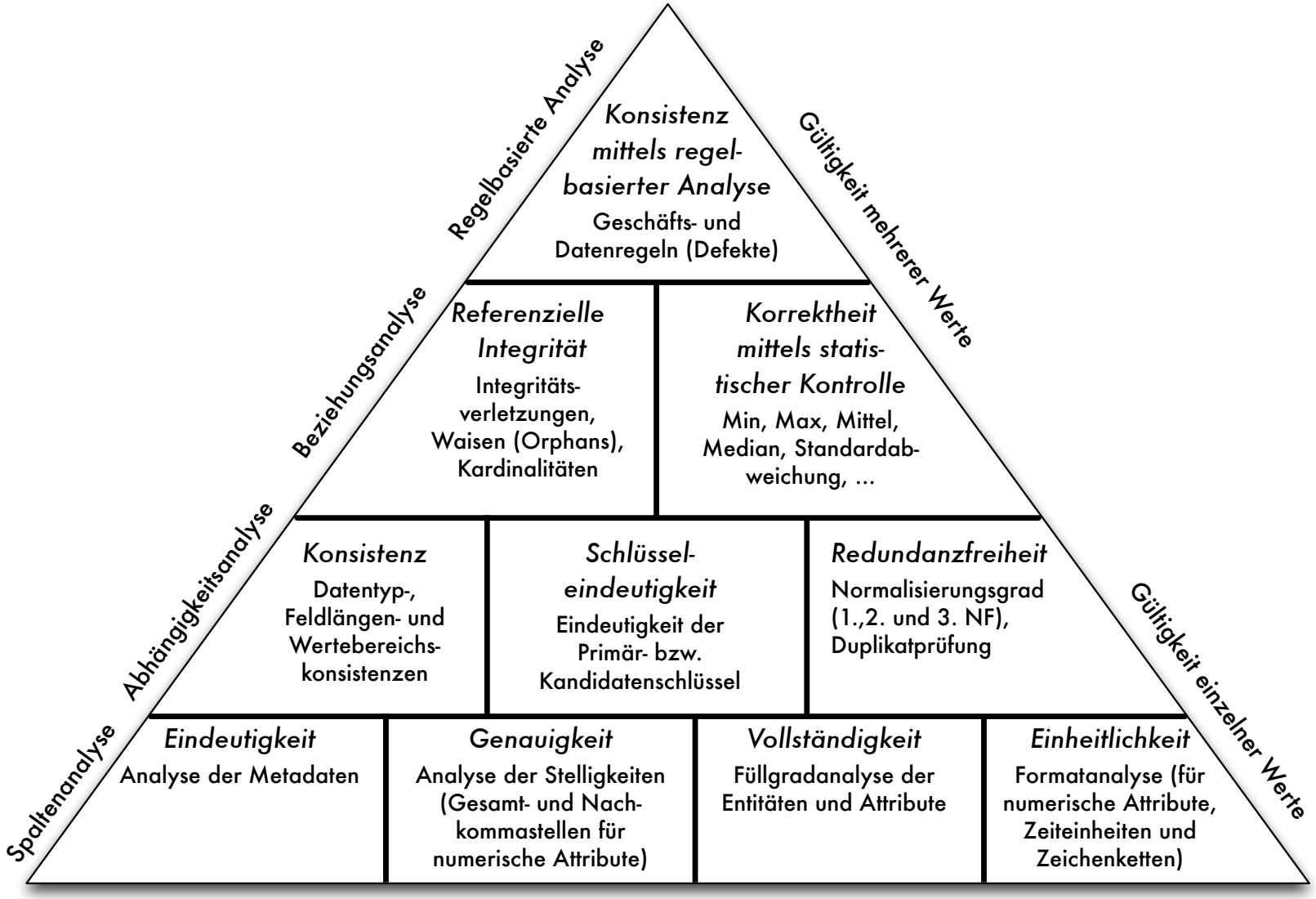
```
select City, count(*) as Numb
from Customer group by City order by Numb
```

    - ★ Ascending: Input errors, e.g., Illmenau: 1, Ilmenau: 50
    - ★ Descending: undocumented default values, z.B. AAA: 80

# Data Cleaning

- Detect & eliminate inconsistencies, contradictions, and errors in data with the aim of improving the quality.
- Also Cleansing or Scrubbing
- Up to 80% of the expense in DW projects
- Cleaning in DW: part of the ETL process

# Data Quality and Data Cleaning



# Normalization and Standardization

- Data type conversion: varchar → int
- Encodings: 1: address unknown, 2: old address, 3: current address, 4: addresse of spouse, ...
- Normalization: mapping in unified format
  - Date: 03/01/11 → 01. März 2011
  - Currency: $ → €
  - Strings to uppercase
- Tokenization: "Saake, Gunter" → "Saake", "Gunter"
- Discretization of numeric values
- Domain-specific transformations
  - Codd, Edgar Frank → Edgar Frank Codd
  - Str. → Street
  - Addresses from address databases
  - Industry-specific product names

# Data Transformation

- In SQL well supported
  - Multiple functions in the language standard
  - SString functions, decoding, conversion date, formulas, system variable, ...
  - Create functions in PL/SQL - use in SQL

- Daten

| | | |
|---|---|---|
| "Pause, Lilo" | $\Rightarrow$ | "Pause", "Lilo" |
| "Prehn, Leo" | $\Rightarrow$ | "Prehn", "Leo" |

- SQL

```
INSERT INTO customers (last_name, first_name)
SELECT SubStr(name, 0, inStr(name,',')-1),
   SubStr(name, inStr(name,',')+1)
FROM rawdata;
```

# Duplicate Detection

- Identify semantically equivalent data sets, i.e., they represent the same real world object
- See also: Record Linkage, Object Identification, Duplicate Elimination, Merge / Purge
  - Merge: Detect duplicates
  - Purge: selection / calculation of the "best" representative per class.

| CustomerNr | Name | Address |
|---|---|---|
| 3346 | Just Vorfan | Hafenstrasse 12 |
| 3346 | Justin Forfun | Hafenstr. 12 |
| 5252 | Lilo Pause | Kuhweg 42 |
| 5268 | Lisa Pause | Kuhweg 42 |
| $\perp$ | Ann Joy | Domplatz 2a |
| $\perp$ | Anne Scheu | Domplatz 28 |

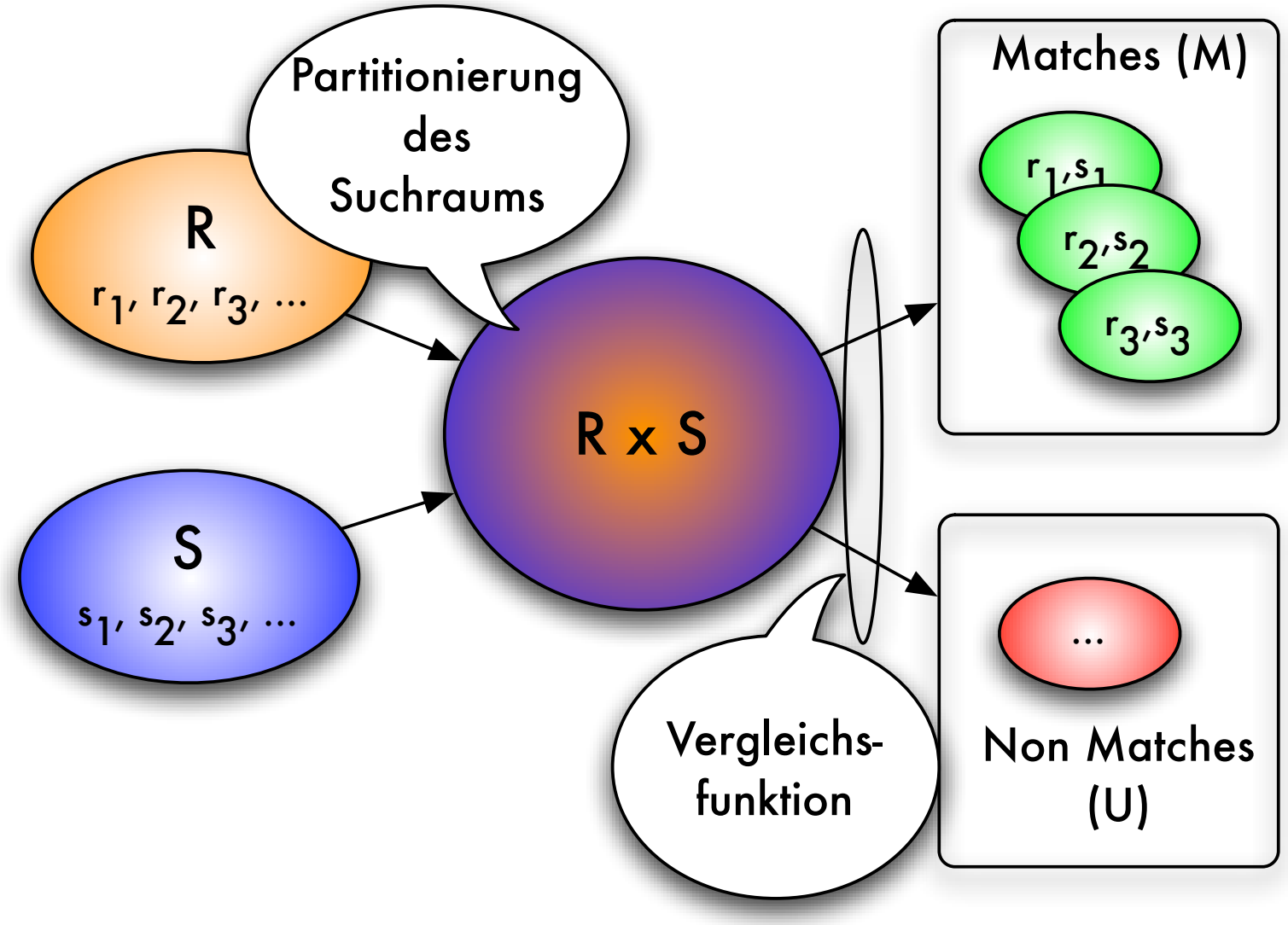
# Duplicate Detection: Comparisons

- Typical comparison rules

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

- Naive approach: "all-vs-all"
  - $O(n^2)$ comparisons
  - Maximum accuracy (depending on rules)
  - Far too expensive

# Duplicate Detection: Principle

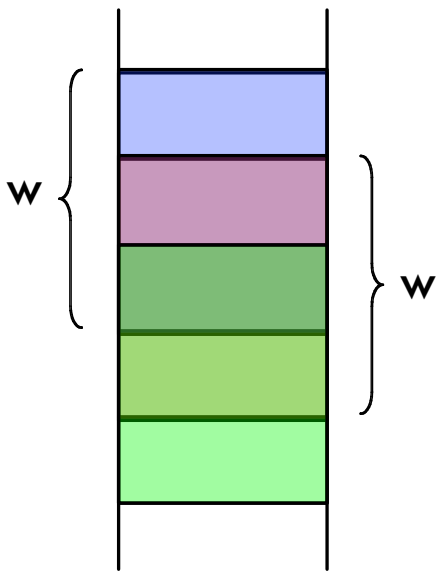


# Partitioning

- Blocking
  - Division of the search space into disjoint blocks
  - Duplicates only within a block
- Sorted neighborhood [Hernandez Stolfo 1998]
  - Sorting the data based on a selected key
  - Compare in a sliding window
- Multi-pass technique
  - Transitive closure over different collations

# Sorted Neighborhood

1. Compute a key for each record
   - ex: SSN + "first 3 characters of Name" + ...
   - Observance of typical errors: 0-O, Soundex, neighboring keys, ...
2. Sort by key
3. Scan list sequentially
4. Comparisons within a window $W$, $|W| = w$
   - Which tuples really need to be compared?



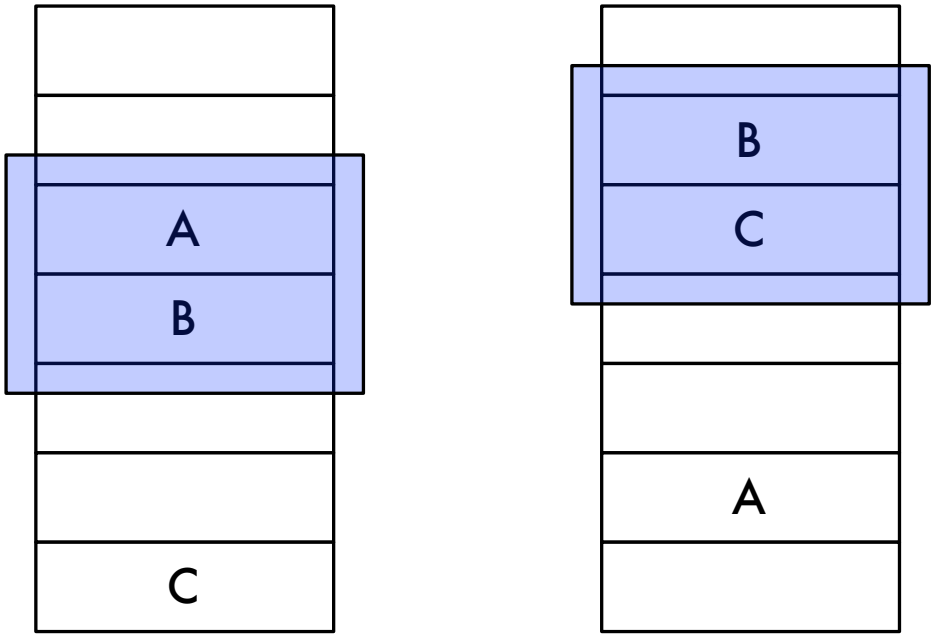

- Complexity
  - Key generation: $O(n)$, sorting: $O(n \cdot \log(n))$; comparing: $O((n/w) \cdot (w^2)) = O(n \cdot w)$;
  - Total: $O(n \cdot \log(n))$ or $O(n \cdot w)$

# Sorted Neighborhood: Problems

- Poor Accuracy
  - Sorting criterion always prefers certain attributes
  - Are the first letters more important for identity than the last ones?
  - Is Surname more important than the house number?
- Increase window size?
  - Not helpful
  - Dominance of an attribute remains the same, but runtime deteriorates rapidly

# Multi-pass technique

- Sort by multiple criteria and identification of duplicates
- Formation of the transitive closure of the duplicates up to a given length




- 1. Run: "A matches B"
- 2. Run: "B matches C"
- Transitivity: "A matches C"

# Comparison functions

- Comparison functions for fields (String $A$ und $B$), including:
  - EEdit distance: number of edit operations (insert, delete, Change) for change from $A$ to $B$
  - q-Grams: Comparison of the amounts of all substrings of $A$ and $B$ of length $q$
  - Jaro distance and Jaro-Winkler distance: Consideration of common characters (within the half string length) and transposed characters (at another position)

# Edit Distance

- Levensthein Distance:
  - Number of edit operations (insert, delete, modify) for change from A to B
  - Example:

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

  - Application:

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

# q-Grams

- Set of all substrings of length $q$
  Qualität$_3$ := { __Q, _Qu, Qua, ual, ali, lit, itä, tät, ät_, t__ }
- Observation: strings with small edit distance have many common q-grams, i.e., for edit distance $k$ min.

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

  common q-grams
- Positional q-grams: extension with position in a string
  Qualität := { (-1, __Q), (0, _Qu), (1, Qua), ... }
  - Filtering for efficient comparison:
    - COUNT: number of common q-grams
    - POSITION:Position difference between corresponding q-grams $\leq$ k
    - LENGTH: The difference in string lengths $\leq$ k

# Data Conflicts

- Data conflict: Two duplicates have different attribute values for a semantically same attribute
  - In contrast to conflicts with integrity constraints
- Data conflicts arise
  - Within an information system (intra-source) and
  - With the integration of multiple information systems (inter-source)
- Prerequisite: Duplicate, already established that identity
- Requires: Conflict Resolution (Purging, Reconciliation)

# Data Conflicts: Origins

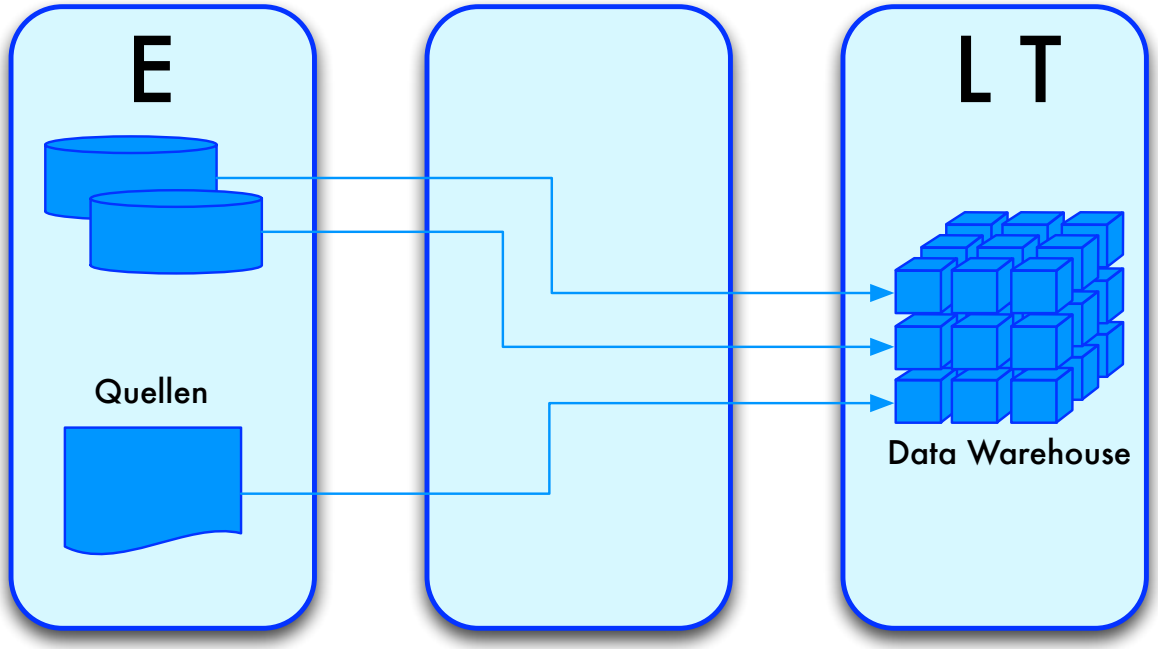
- Lack of integrity constraints or consistency checks
- In case of redundant schemas
- By partial information
- With emergence of duplicates
- Incorrect entries
  - Typing errors, transmission errors
  - Incorrect calculation results
- Obsolete entries
  - Different update times
    - Adequate timeliness of a source
    - Delayed update
  - Forgotten update

# Data Conflicts: Remedies

- Reference tables for exact value mapping
  - For example, cities, countries, product names, codes...
- Similarity measures
  - With typos, language variants (Meier, Mayer, ...)
- Standardizing and transforming
- Use of background knowledge (metadata)
  - For example, conventions (typical spellings)
  - Ontologies, thesauri, dictionaries for the treatment of homonyms, synonyms,. . .
- At integration
  - Preference ordering over data sources according to relevance, trust, opening times, etc.
  - Conflict resolution functions

# ETL vs. ELT

- ELT = Extract-Load-Transform
  - Variant of the ETL process, in which the data is transformed after the load
  - Objective: transformation with SQL statements in the target database
  - Waiving special ETL engines



# ELT

- Extraktion
  - For Database optimized queries (e.g. SQL)
  - Extraction also monitored with monitors
  - Automatic extraction difficult (e.g. data structure changes)
- Laden
  - Parallel processing of SQL statements
  - Bulk Load (assumption: no write access to the target system)
  - No record-based logging
- Transformation
  - Utilization of set operations of the DW-transformation component
  - Complex transformations by means of procedural languages (e.g., PL/SQL)
  - Specific statements (e.g., CTAS von Oracle)

# Summary

- ETL as a process of transferring data from source systems in the DWH
- Topics of ETL and data quality typically make up 80% of efforts in DWH projects!
  - Slow queries are annoying
  - Incorrect results make the DWH useless
- Part of the transformation step
  - Schema level: Schema mapping and schema transformation
  - Instance level: data cleaning