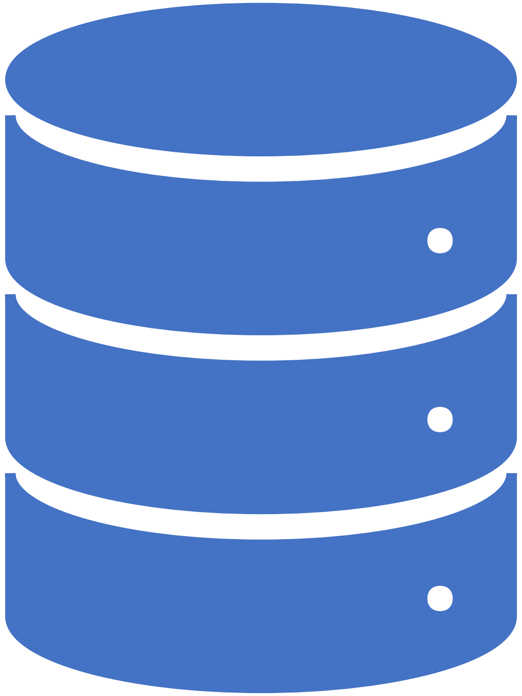
MUNI
FACULTY
OF INFORMATICS



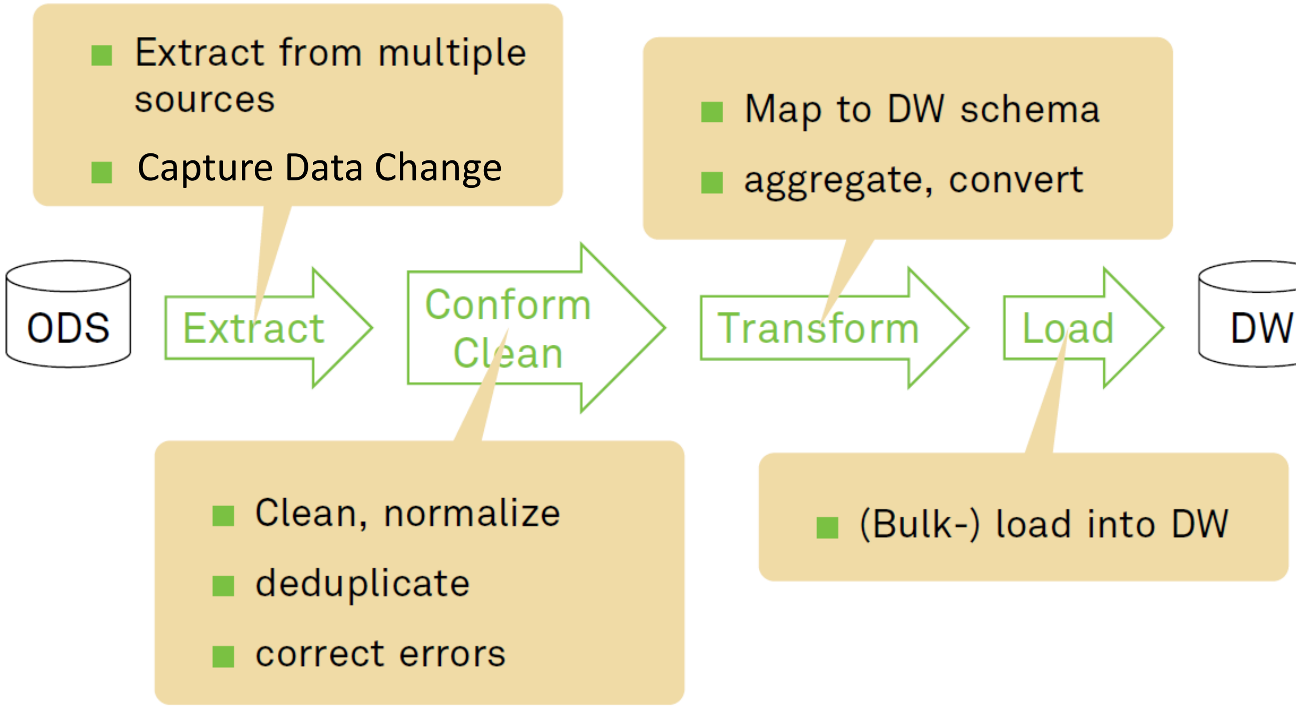
PA220: Database systems for data analytics

ETL Process

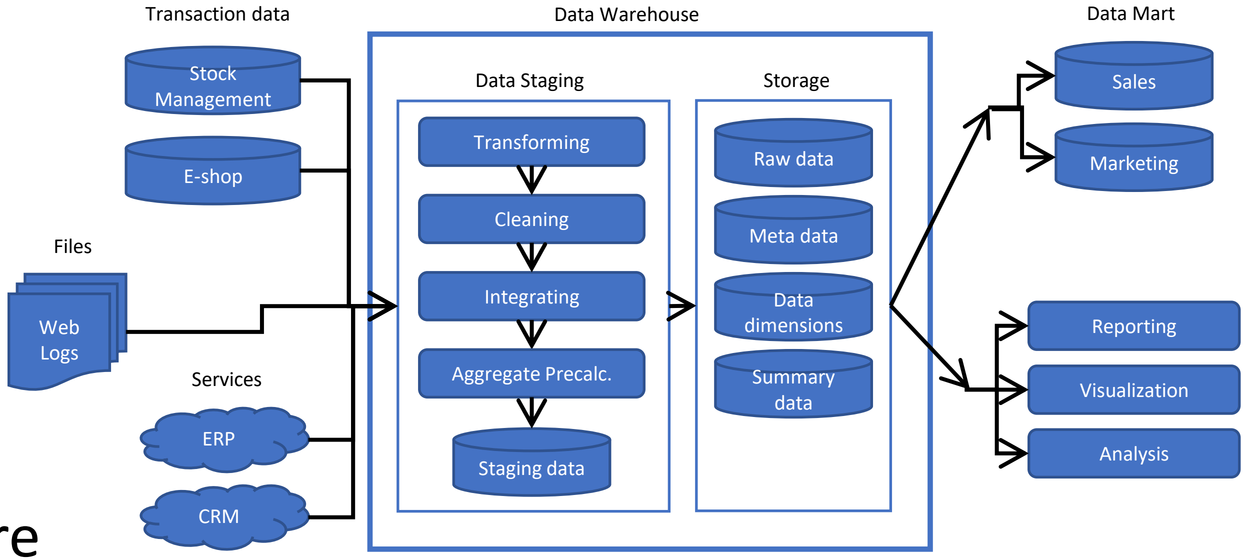
Contents

ETL Process Overview

- Data is periodically brought from the ODS to the data warehouse.
- In most DW systems, the ETL process is the most complex part.
  - and the most underestimated and time-consuming part.
    - Often, 80% of development time is spent on ETL



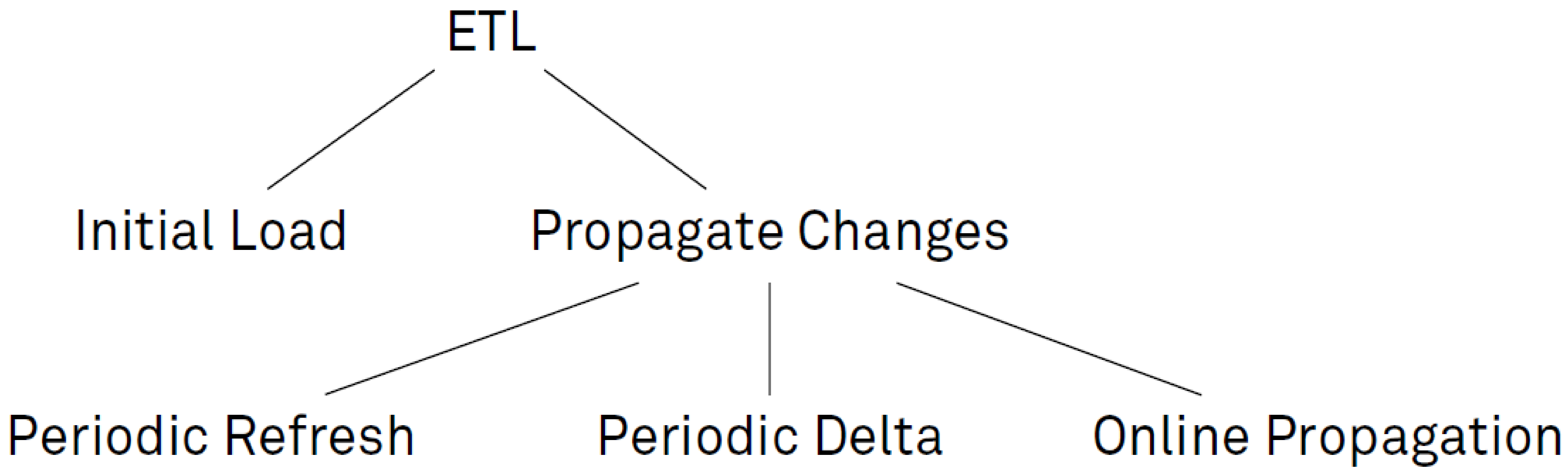
Data Staging Area



- Transit storage for data underway in the ETL process
  - Transformations/cleansing done here
- No user queries (some do it)
- Sequential operations (few) on large data volumes
  - Performed by central ETL logic
  - Easily restarted
  - No need for locking, logging, etc.
  - RDBMS or flat files? (DBMS have become better at this)
- Finished dimensions copied from the staging area to relevant marts

ETL Process Types

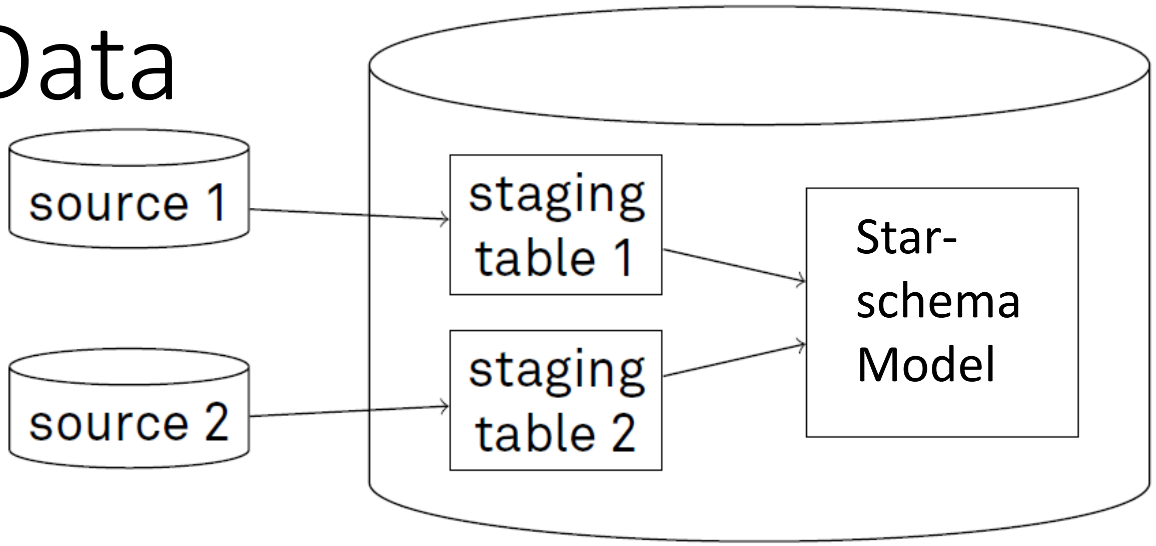
- When do we run the ETL process?



ETL Process Types

- Considerations:
  - **Overhead** on data warehouse and source sides.
    - E.g., online propagation puts a permanent burden on both sides; cannot benefit from bulk loading mechanisms
  - **Data Staleness**
    - Frequent updates reduce staleness but increase overhead.
  - **Debugging, Failure Handling**
    - With online/stream-based mechanisms, it may be more difficult to track down problems.
  - Different process for different flavors of data?
    - E.g., periodic refresh may work well for small (dimension) tables.

Data Extraction: Getting Data



- **Source → Staging Table**:
  - Tool selection depends on data source
    - database, XML, flat files, etc.
  - Use SQL, XQuery, Perl, awk, etc. to query the source system
  - Often:
    - Extract source data to flat file (e.g., CSV)
    - Then bulk-load into staging table

  - Data compression for large data transfers
  - Data encryption if transfer over public networks

Data Extraction: Capturing Data Changes

- Detecting changes is a challenge:
  - **Audit Columns**
    - E.g., “last modified” timestamp
    - Set timestamps or “new” flags on every row update. How?
    - Unset “new” flags on every load into the DW. Why?
  - **Full Diff**
    - Keep old snapshot and diff it with the current version.
    - Thorough, will detect any change
    - Resource-intensive: need to move and scan large volumes
    - Optimization: Hashes/checksums to speed up comparison
  - **Database Log Scraping**
    - The database’s write-ahead log contains all change information
    - Scraping the log may get messy, though.
    - Variant: create a message stream ODS → DW
  - **Message Queue Monitoring**
    - The source system must use a messaging framework; then low overhead

Data Cleansing

- After extraction, data must be **normalized** and **cleaned**.

	Name	Street	CIty	Phone
r_1	Sweetlegal Investments Inc	202 North	Redmond	425-444-5555
r_2	ABC Groceries Corp	Amphitheatre Pkwy	Mountain View	4081112222
r_3	Cable television services	One Oxford Dr	Cambridge	617-123-4567

	Name	Street	CIty	Phone
s_1	Sweet legal Invesments Inc.	202 N	Redmond	
s_2	ABC Groceries Corpn.	Amphitheetre Parkway	Mountain View	
s_3	Cable Services	One Oxford Dr	Cambridge	6171234567

Data Quality (Revision)

- Data in DW must be:
  - Precise
    - DW data must match known numbers - or explanation needed
  - Complete
    - DW has all relevant data, and the users know
  - Consistent
    - No contradictory data: aggregates fit with detail data
  - Unique
    - The same thing is called the same and has the same key (customers)
  - Timely
    - Data is updated ”frequently enough” and the users know when
- Data almost never has decent quality

Data Cleansing

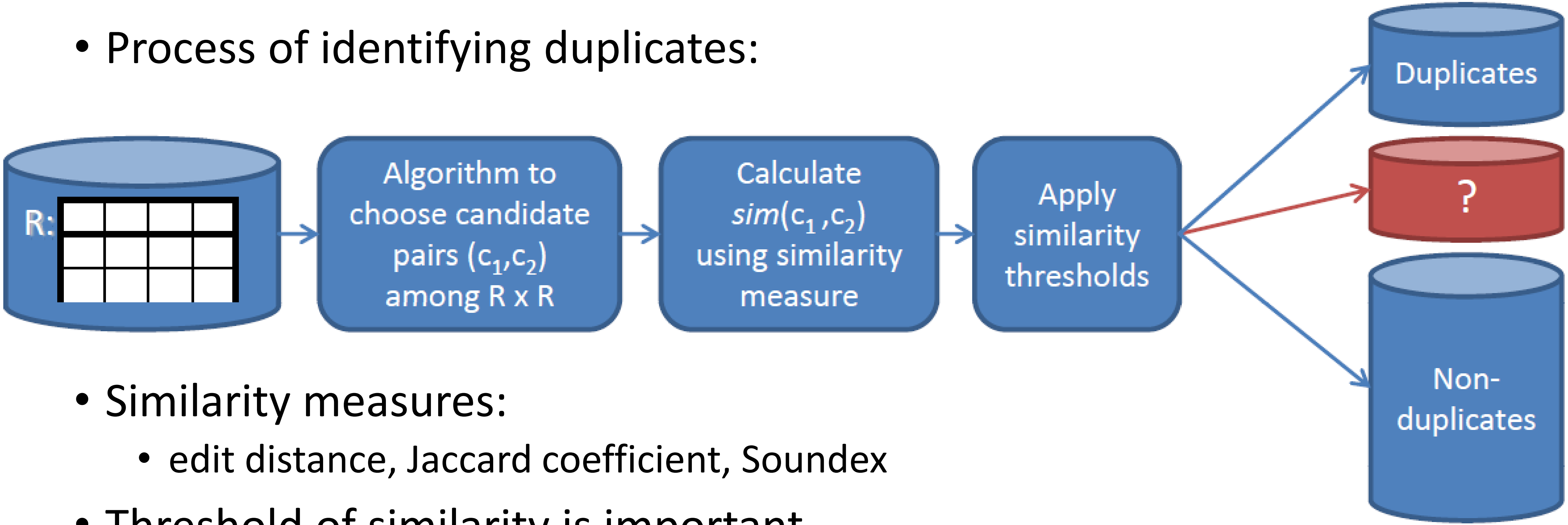
- Problem:
  - Real-world data is messy.
  - Consistency rules in the OLTP system?
    - A lot of data is still entered by people.
    - Data warehouses serve as an integration platform.
- Typical cleaning and normalization tasks:
  - Correct spelling errors / data type conversion.
  - Handle missing / null values.
  - Identify record matches and duplicates.
  - Resolve conflicts and inconsistencies.
  - Normalize (“conform”) data.

Data Cleansing: Primitives

- Parsing
  - E.g., source table has an ‘address’ column; whereas target table has ‘street’, ‘zip’, and ‘city’ columns; pieces of a string to normalize (e.g., “Road” → “Rd”)
- Similarity Join – bring together similar data
  - For record matching (same entity recognition) and deduplication
- Clustering – put items into groups, based on “similarity”
  - E.g., pre-processing for deduplication
- Outlier detection – values not matching the pattern
  - E.g., failure of a sensor; detection of a new “class” / entity

Data Cleansing: Similarity Join

- Process of identifying duplicates:



- Similarity measures:
  - edit distance, Jaccard coefficient, Soundex
- Threshold of similarity is important
  - Limits the number of candidates for duplicates!

Similarity Join – Edit distance

- What is the “similarity” between strings s_1 and s_2?
- $d_{edit}(s_1,s_2)$ = min. number of operations to transform s_1 into s_2
  - E.g., s_1 = “Sweet” and s_2 = “Sweat”

insert	$\mathtt{ab} \rightarrow \mathtt{axb}$
delete	$\mathtt{axb} \rightarrow \mathtt{ab}$
replace	$\mathtt{axb} \rightarrow \mathtt{ayb}$
transpose	$\mathtt{axyb} \rightarrow \mathtt{ayxb}$

- Levenshtein distance – ins, del, replace only
- Longest common subsequence (LCS) – ins, del only

Source: Jens Teubner Data Warehousing

Similarity Join – Jaccard coefficient

- Similarity of two sets S_1 and S_2
  - by comparing cardinalities of intersection and union
- Example
  - q-grams: converting a string to a set
    - i.e. a set of all substrings of length q
  - 2grams("Sweet") = { Sw, we, ee, et }

$$\frac{|\{\mathtt{Sw}, \mathtt{we}, \mathtt{ee}, \mathtt{et}\} \cap \{\mathtt{Sw}, \mathtt{we}, \mathtt{ea}, \mathtt{at}\}|}{|\{\mathtt{Sw}, \mathtt{we}, \mathtt{ee}, \mathtt{et}\} \cup \{\mathtt{Sw}, \mathtt{we}, \mathtt{ea}, \mathtt{at}\}|} = \frac{|\{\mathtt{Sw}, \mathtt{we}\}|}{|\{\mathtt{Sw}, \mathtt{we}, \mathtt{ee}, \mathtt{et}, \mathtt{ea}, \mathtt{at}\}|} = \frac{2}{6} = \frac{1}{3}$$

Source: Jens Teubner Data Warehousing

Similarity Join – SoundEx

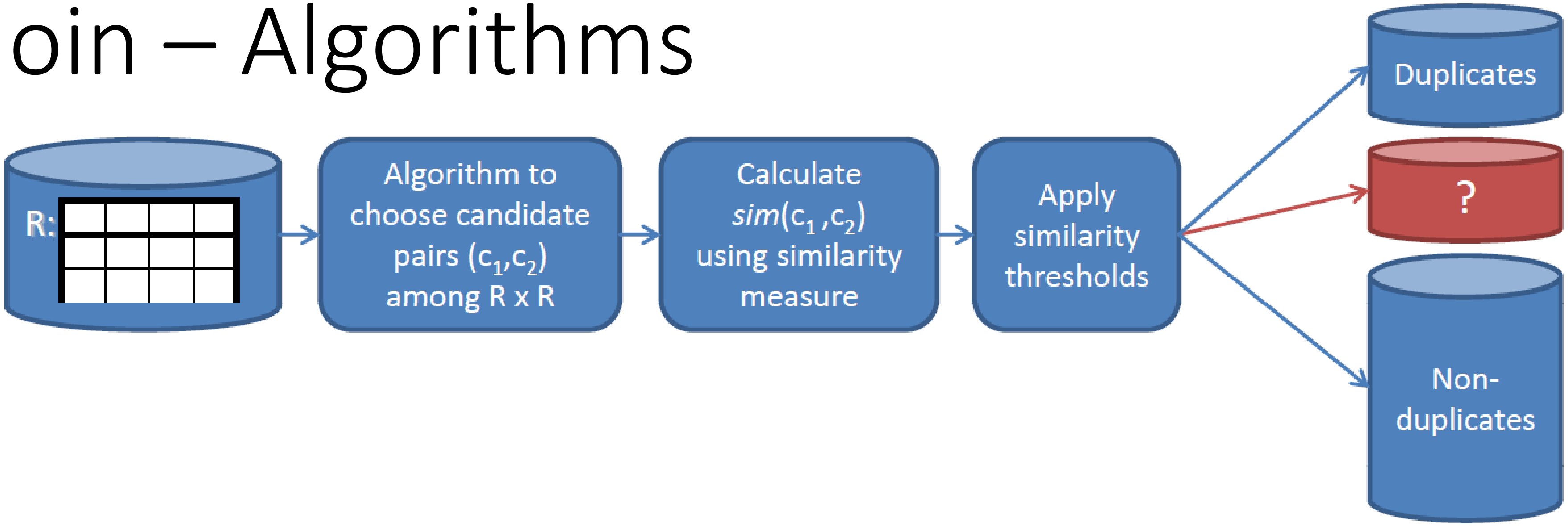
- Phonetic algorithm to index words by sound
  1. Retain the first letter
  2. Replace letters with numbers
     - Mapping of alike sounds to the same number
     - If no mapping, drop the letter
  3. Drop letters where the preceding letter yielded the same number.
  4. Collect three numbers, fill with zero's if necessary.

$$
\begin{aligned}
b, f, p, v &\rightarrow 1 \\
c, g, j, k, q, s, x, z &\rightarrow 2 \\
d, t &\rightarrow 3 \\
l &\rightarrow 4 \\
m, n &\rightarrow 5 \\
r &\rightarrow 6
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{soundex}(\texttt{"Sweet"}) = \texttt{S300}; \\
&\mathit{soundex}(\texttt{"Robert"}) = \mathit{soundex}(\texttt{"Rupert"}) = \texttt{R163}.
\end{aligned}
$$

Source: Jens Teubner Data Warehousing

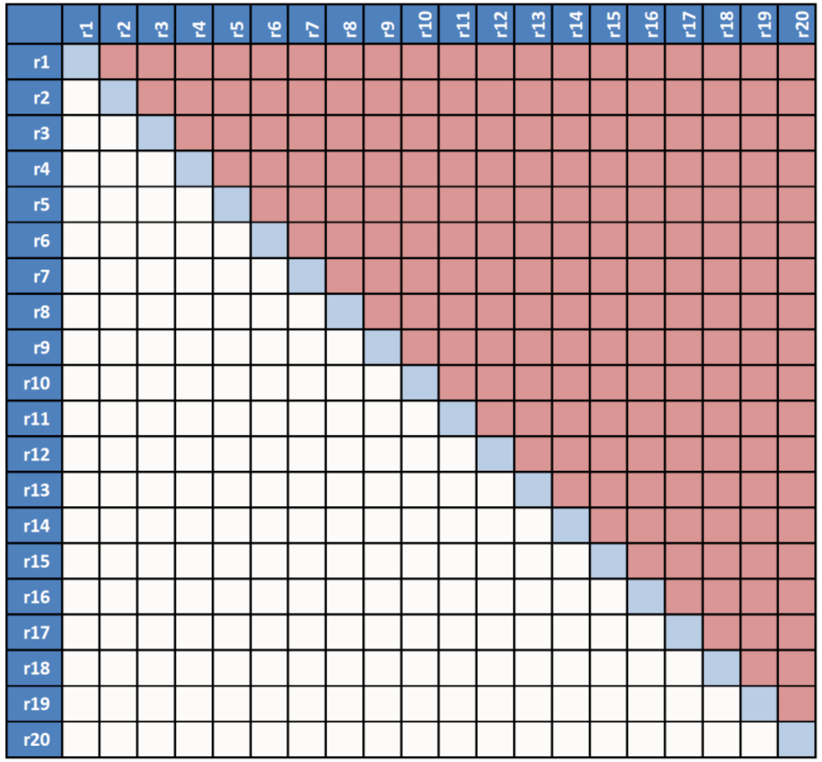
Similarity Join – Algorithms



- Naïve strategy
  - $O(n^2)$
- Blocking naïve strategy
  - Assume b blocks and compare only within
  - $O(1/2\ (n^2/b - n))$
- Sorted neighborhood
  - Sort the inputs and scan with sliding windows of w $(w-1)\cdot\left(n-\frac{w}{2}\right)$
- Index to accelerate similarity range query R(q,threshold)

Similarity Join – Naïve strategy

- Compare every record with each other
  - Assume symmetry of distance

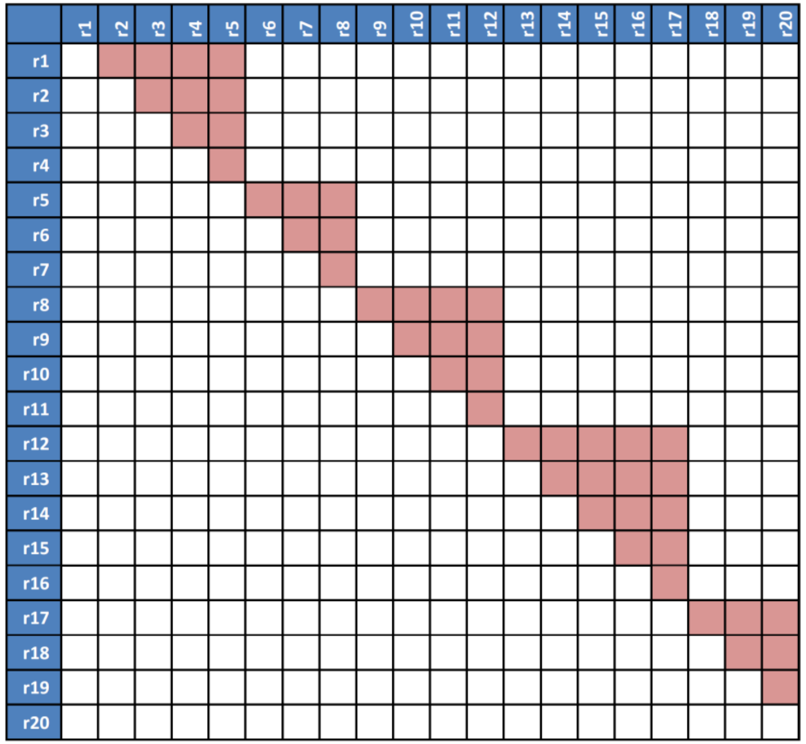


Source: Jens Teubner Data Warehousing

Similarity Join – Blocking naïve strategy

- Partition input data into blocks
  - Duplicates must end in the same block!
  - Disadv: typically, uneven block sizes…
- Compare all pairs within blocks only

$$b \cdot \frac{\frac{n}{b}\left(\frac{n}{b}-1\right)}{2} = \frac{n\left(\frac{n}{b}-1\right)}{2} = \frac{1}{2}\left(\frac{n^2}{b}-n\right)$$



- E.g. for matching customers
  - Use their ZIP code (assuming ZIP has not changed, customer has not moved)
  - Use first character of last name

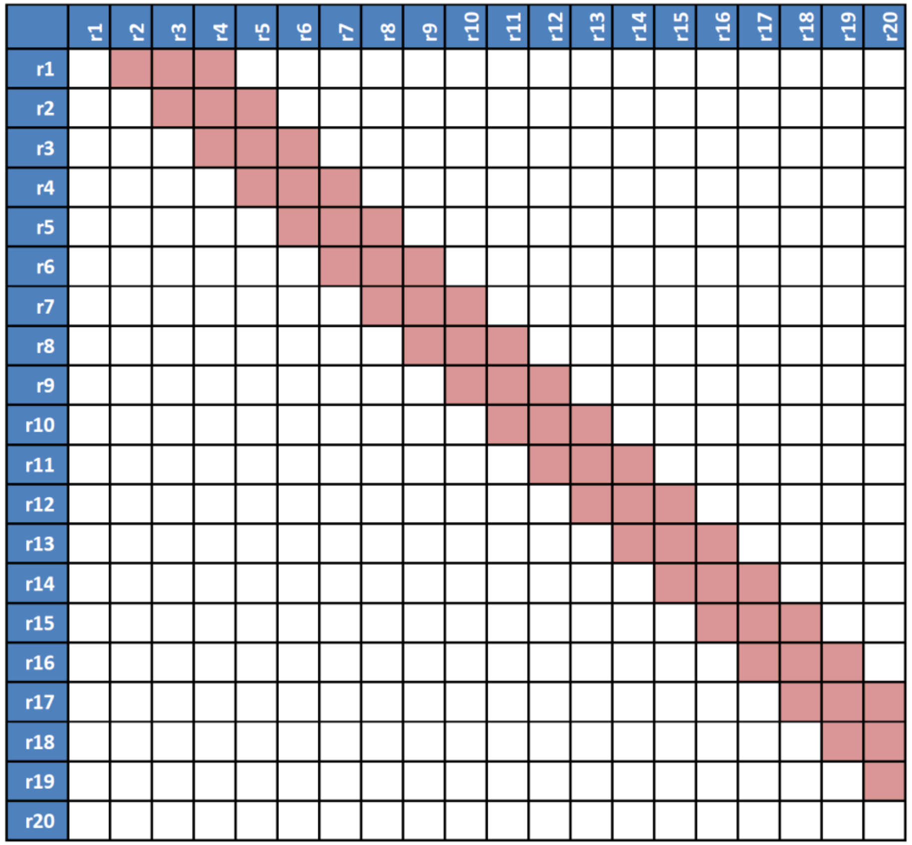
Source: Jens Teubner Data Warehousing

Similarity Join – Sorted neighborhood

- Assign a sort key to each record
- Sort the records
- Apply a sliding window of size w across the sorted list and join within.

$$(w-1) \cdot \left(n - \frac{w}{2}\right)$$

- E.g., sort customers by
  - First 3 consonants of last name
  - First letter of last name and first 2 digits of ZIP code



Source: Jens Teubner Data Warehousing

Data Cleansing: Detecting Inconsistencies

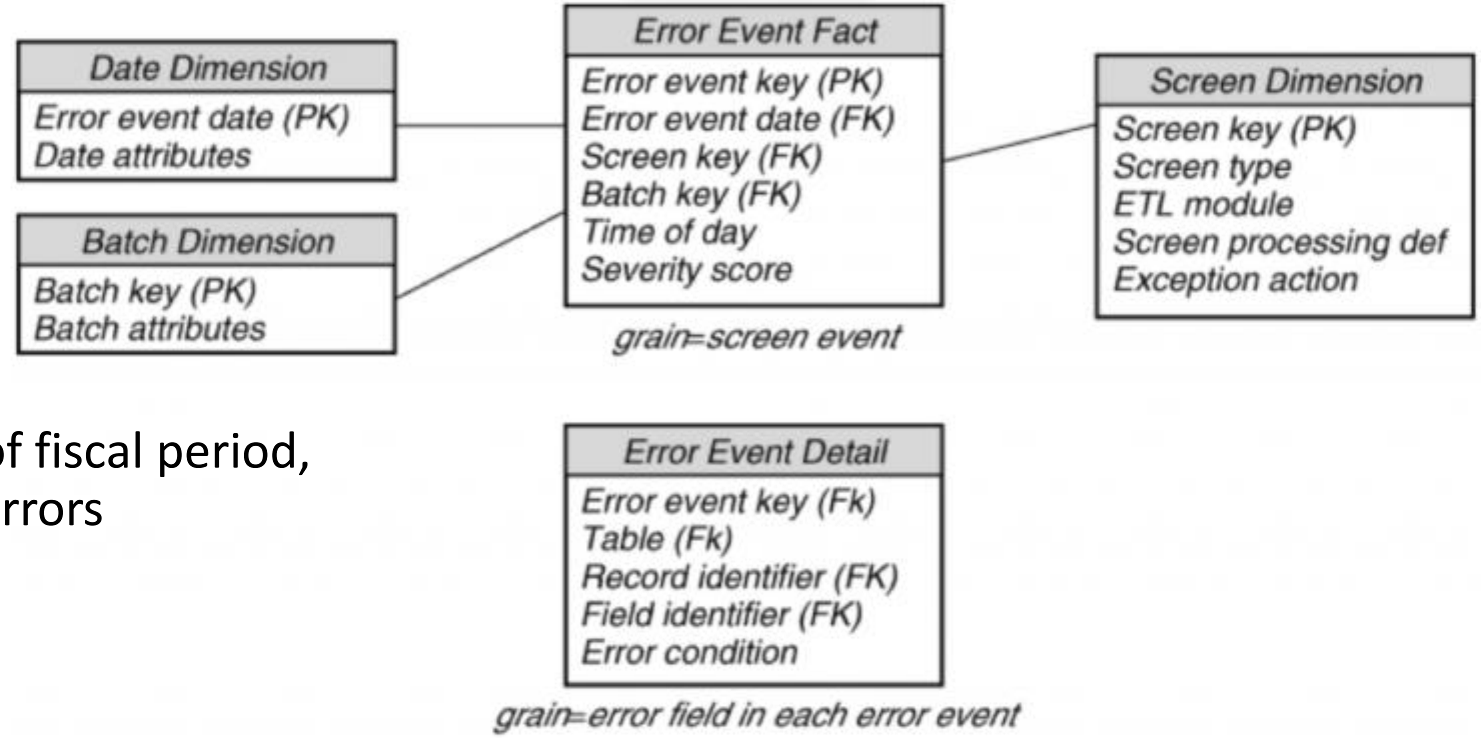
- Data (quality) screening system:
  - Column screens: Test data within a column
    - Correct value ranges, value formatting, null values?
    - Detect random/noise values
  - Structure screens: Relationship across columns
    - Two or more columns implement a hierarchy (e.g., a series of m:n relationships)
    - Foreign key relationships between tables
    - Combination of columns is a valid item, e.g., an existing postal address
  - Business rule screens: Data plausible according to business rules?
    - E.g., customer status X requires Y years of loyalty, Z EUR total revenue in previous period

Data Cleansing: Error Handling

- Halting the process on error
  - requires manual intervention – diagnose, restart/resume the job or abort it
- Create a suspense file (logging)
  - Log the errors in a side channel for later processing
  - Not clear when to handle its contents – fix the records and re-introduce to the job?
    - until these data items are restored, the overall DB integrity is questionable
- Tag the data and continue
  - Bad fact records – create an audit dimension
  - Bad dimension data – use unique error values
  - Best solution whenever possible

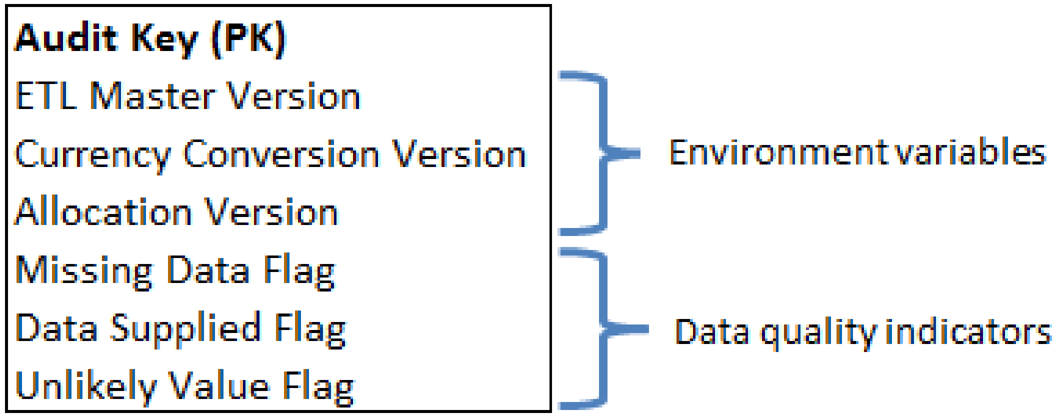
Data Cleansing: Error Handling by Logging

- A special error event schema can be created
  - as a result of “Tag the data and continue“
- Grain corresponds to the error appearance
  - Batch dim – info of the job
  - Date dim – not a minute and sec of the error
    - rather a weekday, last day of fiscal period, to constraint / summarize errors
  - Time of day – timestamp when the error occurred



Data Cleansing: Error Handling by Tagging

- Audit dimension
  - attached to the resulting fact table
  - created in data cleansing
  - stores audit conditions
- Example
  - an ETL job finished with no error
    - a new audit rec. describing it is created
    - all new fact records are associated with it
  - if an error occurred (e.g., out of bounds)
    - another audit rec is created, and the failing fact records get attached



Improving Data Quality

- Appoint ”data stewards” - responsible for data quality
  - A given steward has the responsibility for certain tables
  - Includes manual inspections and corrections!
- DW-controlled improvement
  - Default values
  - ”Not yet assigned 157” note to data steward
- Source-controlled improvements
  - The optimal?
- Construct programs that check data quality
  - Are totals as expected?
  - Do results agree with alternative source?
- Do not fix **all** problems with data quality
  - Allow management to see ”weird” data in their reports?

Data Transformation: Schema Integration

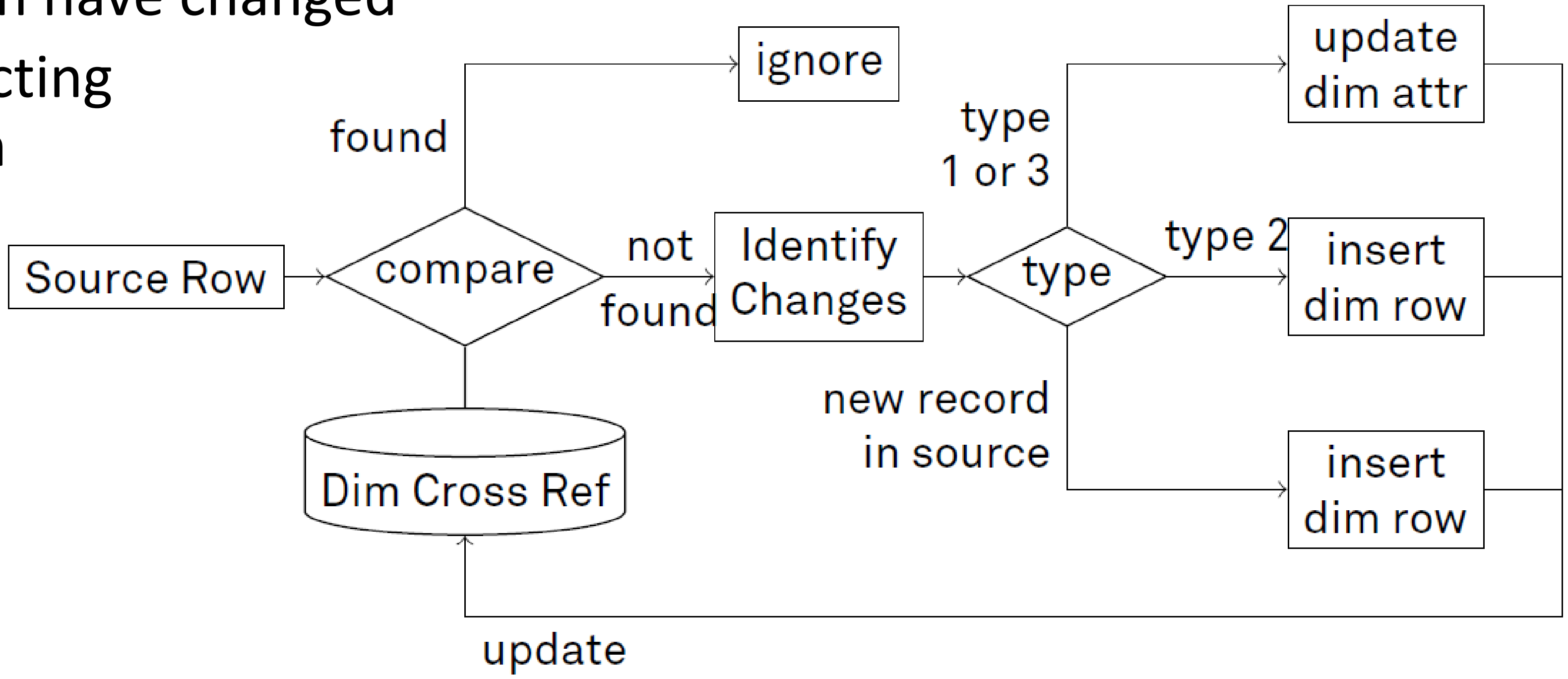
- Different source systems, types, and schemas must be integrated.
- Infer mapping between schemas (automatically)?

- Tools:
  - Compare table and attribute names; consider synonyms and homonyms
  - Infer data types/formats and mapping rules
  - Techniques like similarity joins and deduplication.

- Still:
  - Often a lot of manual work needed.

Data Loading: Prepare Dimension Tables

- For each dimension do the following checks:
  - Dimension row is new
    - Generate the surrogate keys
  - Attributes in dimension have changed
  - Handle updates respecting SCD type of dimension



Data Loading: Prepare Dim Tables - Problems

- “upsert” – update if exists, else insert (aka SQL-based update)
  - often a real performance killer
  - better: separate updates and bulk-load inserts
- Generate and find dimension surrogate keys
  - e.g., use key generator of back-end DB
  - Maintain “Dim Cross Ref” table in memory or in back-end DB
- Dimensions must be updated before facts
  - The relevant dimension rows for new facts must be in place
  - Special key considerations if initial load must be performed again
- May re-compute aggregates (Type 1 updates)
  - again, bulk-loading/changing is a good choice

Data Loading: Performance Tips

1. Turn off logging
   - Databases maintain a write-ahead log to implement failure tolerance mechanisms.
   - Row-by-row logging causes huge overhead.
2. Disable indexes and reindex after updates
3. Pre-sort data
   - Depending on system, may speed up index construction.
   - Additional benefit: may result in better physical layout
4. Truncate table
   - When loading from scratch

Loading Data – Performance Tips

5. Enable “fast mode”
   - If data is prepared properly, database may use faster parsing mechanisms
   - e.g., “copy from” command
6. Make sure data is correct
   - Transformation, field truncation, error reporting may slow down bulk-loading significantly
7. Temporarily disable integrity control
   - Avoid checking during load, but do it in bulk, too.
   - e.g., foreign keys in the fact table

Loading Data – Performance Tips

8. Parallelization
    - Dimensions can be loaded concurrently
    - Fact tables can be loaded concurrently
        - Partitions of one fact table can be loaded concurrently
            - when horizontal partitioning is used

Hints on ETL Design

- Do **not** try to implement all transformations in one step!

- Do **one** (or just a few) thing(s) at a time
  - Copy source data one-one to staging area
  - Compute deltas
    - Only if doing incremental load
  - Handle versions and generate DW keys
    - Versions only if handling slowly changing dimensions
  - Implement complex transformations
  - Load dimensions
  - Load facts

General Issues / Decisions

- Files versus streams/pipes
  - Streams/pipes: no disk overhead, fast throughput
  - Files: easier restart, often the only possibility
- ETL tool or self-coding
  - Code: easy start, co-existence with IT infrastructure
  - Tool: better productivity on subsequent projects
- Load frequency
  - ETL time depends on processed data volumes.
    - Daily load is much faster than monthly.
  - Applies to all steps in the ETL process
- Should DW be on-line 24/7?
  - Use partitions or several sets of tables

Summary

- ETL is very time consuming (80% of entire project)
  - Needs to be implemented as a sequence of many small steps
  - Types of ETL processes
- **Extraction** of data from source systems might be very time consuming
  - Incremental approach is suggested
- **Transformation** into DW format includes many steps, such as
  - building key, cleansing the data, handle inconsistent/duplicate data, etc.
- **Load** includes the loading of the data in the DW, updating indexes, computing pre-aggregates, etc.
- Performance issues and tips