4–1 Data Warehouse – ETL Processing

Data Warehousing
Spring Semester 2011
The ETL Process in the DWh Reference Architecture

ETL = Extract, Transform and Load
Overview of the ETL Process

- **ETL** =
  - **Extract** (from the various sources)
  - **Transform** (data representation of sources to that of the warehouse)
  - **Load** (into the single warehouse ... or to several data marts)

- **Pro memoria** *(cf introduction):* characterization of DWh
  - integrated
  - subject-oriented (≈ uses star-schemas)
  - non-volatile (= keeps history)

- Operationally, ETL processes ensure integration and historization (and, when loading data marts, subject-orientation)
Note: None of the “data pots” shown below *have* to exist physically, although they often do …
Main criteria for explicitly storing “intermediate results”:
- independence from other autonomous teams/systems, especially the sources
- ability to restart a failed process step from an intermediate result
- necessity of other manual intervention

Another note: “Extract > Clean > Conform > Deliver" = “Extract > Transform > Load"
- Transform = Clean + Conform
- Load = Deliver
Development of ETL Processes

• Before developing ETL Processes, a rough overview of major applications, their databases, and the data flows between them should be available.

• For each data item, the **best source** should be determined.

• The best data source is typically the **system of record**, meaning the (OLTP) system where the data **originates**, i.e., is captured for the first time.

• However, there may be cases where there is more than system of record, depending on geography (e.g., European employees vs US employees) or product lines. If this data is already propagated into a single integrated data store, consider using this as a source.

  Note: The further away from the origin, the lower the quality of data usually is.

• **Q:** Who does how much in the data transformation from source to DWh?
  **A:** Usually, most if not all work is done by the ETL / DWh crew ...
Implementation Decisions for ETL

• „Buy vs Build“
  - „Buy“ = buying an ETL Tool, offering many services (=„creature comforts“),
    . connectors to large variety of sources, including abilities like log sniffing
    . session control
    . error handling
    . efficient generation of surrogate keys
    . high-level (graphical) language to specify transformations
  - „Build“ = develop your own solution

• If your choice is „build“, then what technology should you use?
  - plain old „flat files“ or XML files + „procedural“ language (Java, Perl, ...)
  - DBMS technology: SQL and procedural extensions (+ new SQL features)
    for reading data, DBMS technology tends to be competitive w.r.t. performance
    for writing data, file technology tends to be more efficient
    (but consider using DBMS loader utilities if write performance is an issue)
Auxiliary Data Structures in ETL Area

- Mapping Tables
  - keys / codes in sources to corresponding warehouse keys / codes
  - „successor“ or „survivor“ information for duplicate entities

- Data Quality Assessments:
  how trustworthy is the information in the warehouse

- Data Lineage information:
  which transformations were applied to source data items

- Data Volume information:
  e.g. row counts of previous loads
Data Profiling – Prerequisite to ETL Development

• **Data Profiling** of chosen source(s) -- see e.g. [Olson 2003]:
A systematic examination of the quality, scope, and context of a data source, especially with respect to

  - data **structures**, e.g.,
    relational schemas, COBOL copybooks, spreadsheet conventions, XML DTDs / XML Schema / ...

  - data **content**, e.g.,
    conventions / patterns, ranges / enumerations of values, dependencies

  - data characteristics vs data requirements, e.g.
    - rate of change in data vs required „refresh“ frequencies
    - data volumes
    - special „time windows“ for data extractions (if any 😊)
    - accuracy requirements
More on Data Profiling – Data Structures

• Required structural information
  - tables, their columns, and data types
  - primary keys, foreign keys, uniqueness and not null constraints
  - helpful: additional explanations

• Many database design tools can reverse engineer an entity relationship diagram from the DBMS dictionary / catalog tables (e.g., in Oracle, USER_TABLES, USER_COLUMNS, USER_CONSTRAINTS etc.)

• But note: dictionary / catalog tables only contain metadata what was explicitly declared to the DBMS ...
  Often missing:
  - explanations (in Oracle: USER_TAB_COMMENTS, USER_COL_COMMENTS)
  - constraints, in particular foreign key constraints
    (usual excuse: „it’s more efficient to check this in application code“)
More on Data Profiling – Data Content

• Presence of null values (or similar, e.g., „missing“, „unknown“, ... )
• Numeric information in text fields
• Date information in text fields, including which format
• For „discrete“ data domains:
  Distribution of data values, can be done using SQL, e.g.,
  select STATE, count(*) as OCCURENCES
  from CUSTOMER
  group by STATE
  order by count(*)
• For ranges: min and max values, maybe average or median, standard dev
• For strings: min and max lengths, maybe average or median
• Check documentation, e.g., for comments on measurement units etc
• Check for dependencies between data fields (often called „Business Rules“)
Extracting Data from Sources

- Access to hardware / OS / DBMS platform of source
  - mainframe legacy systems (EBCDIC character set, decimal data types, ... )
  - ODBC / JDBC vs native drivers for relational (or nearly relational) DBMSs
  - use of an ETL Tool (e.g., Informatica PowerCenter) offering required „connectors“

- (Usually for dimensional data only):
  How to get the only the data that changed since the last extract?
  - use explicit „audit columns“ in source databases (if there are any ... )
    and select only rows where timestamp > last extract time
    Does every change procedure in the source set this correctly?
  - database log sniffing
    What if log entries get cleared before the next extract?
  - Keep previous source database extracts in the landing/staging area
    and do an explicit „diff“ before the warehouse upload.
    Unfortunately, this is not very efficient ...
Extracting from non-relational sources

- Comma- (or tab-) Separated Value (CSV) files (text)
- Excel Spreadsheets (pre 2007 internal format: xls)

Issues with csv / xls formats:
  - Convention to describe expected data structure needed
  - Format flexible and hence error-prone (unless machine-generated)

- XML files (text) – hopefully with DTD or XML Schema
- Mainframe Legacy (Database) Systems
  - ISAM/VSAM etc files containing fixed- or variable-length records
    (the structure of which is described in a program, e.g. COBOL copybooks)
  - hierarchical (IMS) and network DBMSs
    these sources are difficult to access from UNIX/LINUX/Win environments
      ⇒ best bet: ask mainframe staff to provide text files

Issues with ISAM/VSAM and XML files:
  - may contain „repeating fields“ etc ⇒ convert to first normal form (1NF)
Example of a COBOL Copybook

01 EMP-RECORD.
   05 FIRST-NAME       PIC X(10).
   05 MIDDLE-INITIAL   PIC X.
   05 LAST-NAME        PIC X(15).
   05 SSN              PIC X(9).
   05 EMP-DOB.
      10 DOB-YYYY       PIC 9(4).
      10 DOB-MM         PIC 9(2).
      10 DOB-DD         PIC 9(2).
   05 EMP-ID           PIC X(9).
   05 HIRE-DATE.
      10 HIRE-YYYY      PIC 9(4).
      10 HIRE-MM        PIC 9(2).
      10 HIRE-DD        PIC 9(2).
   05 TERM-DATE.
      10 TERM-YYYY      PIC 9(4).
      10 TERM-MM        PIC 9(2).
      10 TERM-DD        PIC 9(2).
   05 TERM-REASON_CODE PIC X(2).
### Example of a COBOL Copybook with Redefine

```cobol
05 EMPLOYEE-TYPE PIC X.
   88 EXEMPT VALUE 'E'.
   88 HOURLY VALUE 'H'.
05 WAGES.
   10 EXEMPT-WAGES.
      15 PAY-GRADE PIC X(2).
      15 SALARY PIC 9(6)V99 COMP3.
      15 PAY-PERIOD PIC X.
      88 BI-WEEKLY VALUE '1'.
      88 MONTHLY VALUE '2'.
   10 NON-EXEMPT-WAGES
      REDEFINES EXEMPT-WAGES.
      15 PAY-RATE PIC 9(4)V99.
      15 JOB-CLASS PIC X(1).
   15 FILLER PIC X.
```

<table>
<thead>
<tr>
<th>Len</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>71</td>
</tr>
<tr>
<td>8</td>
<td>72</td>
</tr>
<tr>
<td>8</td>
<td>72</td>
</tr>
<tr>
<td>2</td>
<td>72</td>
</tr>
<tr>
<td>5</td>
<td>74</td>
</tr>
<tr>
<td>1</td>
<td>79</td>
</tr>
<tr>
<td>8</td>
<td>72</td>
</tr>
<tr>
<td>6</td>
<td>72</td>
</tr>
<tr>
<td>1</td>
<td>78</td>
</tr>
<tr>
<td>1</td>
<td>79</td>
</tr>
</tbody>
</table>

---

R. Marti  4-1 DWh 2011: Data Warehouse – ETL Processing  Page 14
Cleaning (sometimes also “Cleansing”) Data

• The quality of the data in many organizations has ample room for improvement ... and Data Warehousing / ETL brings this to the light ... (Data Warehousing = GIGO: Garbage In, Garbage Out)

• Ultimately, „dirty“ data should be avoided at the source, namely, in the system / database, where the data is entered for the first time

• However, fixing this sustainably often involves much more than debugging, refactoring, and extending a single IT application

⇒ Business Process Reengineering:
  - which tasks does a process entail, in which order
  - which tasks should be performed where (e.g., minimizing handovers)

⇒ Data Quality Initiatives
Data Quality Dimensions

Following [Lehner 2003], we mention only

• Accuracy
  Is the data correct (and precise enough)?

• Completeness
  Has all the data required been captured?

• Consistency
  Is there is a single convention for data values that occurs in several places?

• Actuality
  Does the data describe the current state of the real world, or is it already out of date?
Detecting and Reacting to Data Quality Issues

Typical tradeoffs in the cleaning phase

- Completeness vs speed of data checks
  more checks result in higher quality data but take longer

- (Automatic) data corrections vs transparency
  Corrections (hopefully!) improve the data quality ... but the data provider may not recognize his/her data anymore ...

- What if a record fails to pass a check (or even several checks)?
  (1) pass it on anyway (but why then perform the check(s) in the first place)
  (2) pass it on with a „flag“ (or quality score) that marks it as defective
  (3) reject it (but then the data in the Warehouse will not be complete)
  (4) stop the complete ETL process (but „the show must go on“ – usually)
  Typical policy for most issues: Option (2)

- Who should ultimately be responsible to fix (at least the most glaring) data quality issues?
  - the source
  - the warehouse
Responsibilities for Data Quality Issues

Data Quality Issues Policy

Category A
MUST be addressed at the SOURCE

Category B
BEST addressed at the SOURCE

Category C
BEST Addressed in ETL

Category D
MUST be Addressed in ETL

ETL Focus is here

Political DMZ

Universe of Known Data Quality Issues
Hard vs Soft Constraints

• Given that
  - you receive an entire batch of records at once
  - most DBMSs enforce integrity constraints (PK, FK, check) immediately, i.e.
    raise an exception after the first record violates a constraint
    using DBMS constraints is not really recommended
    (unless disabling and later re-enabling them is supported in a useful fashion)

• Besides, multi-row / multi-table constraints are not supported in most DBMSs
  anyway ...

• Typical strategy: transform constraint as a query so that offenders are returned
  and define a view and/or table to capture this information
  e.g.:
  - select PK from TAB where COL is null
  - select PK, COL from TAB where not COL between low and high
  - select PK, count(*) as OCCURENCES from TAB
    group by PK having count(*) > 1
  - select PK, FK from FK_TAB f where not exists
    ( select * from PK_TAB p where p.PK = f.FK )
Special DBMS Support for Error Logging (Oracle)

Inserting Into a Table with Error Logging: Example

The following statements create a raises table in the sample schema hr, create an error logging table using the DBMS_ERRLOG package, and populate the raises table with data from the employees table. One of the inserts violates the check constraint on raises, and that row can be seen in errlog. If more than ten errors had occurred, then the statement would have aborted, rolling back any insertions made:

CREATE TABLE raises (emp_id NUMBER, sal NUMBER
    CONSTRAINT check_sal CHECK(sal > 8000));

EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('raises', 'errlog');

INSERT INTO raises
    SELECT employee_id, salary*1.1 FROM employees
    WHERE commission_pct > .2
    LOG ERRORS INTO errlog ('my_bad') REJECT LIMIT 10;

SELECT ORA_ERR_MSG$, ORA_ERR_TAG$, emp_id, sal FROM errlog;

ORA_ERR_MSG$ ORA_ERR_TAG$  EMP_ID  SAL
----------------- --------------  ------  ----
ORA-02290: check constraint my_bad 161  7700
(HR.SYS_C004266) violated
Typical Checks Performed in Cleaning Phase

- Column property enforcement
  - Not null checks
  - Range checks
  - Membership in a fixed list of valid values (enumerations)
  - Adherence to patterns (e.g. dates, phone numbers, ISBNs, ...)
  - „Non-membership“ in a fixed list of known (frequently occurring) incorrect values
  - „Unusual“ string lengths (!)
  - Possibly spell checks in „narrative text“
  - Possibly check for outliers in numbers, e.g. \( \mu \pm n \cdot \sigma \), where \( \mu \) = hist average (or median), \( \sigma \) = hist standard deviation, and \( n > 3 \)
- Structure enforcement
  - Primary keys (identifiers)
  - Alternate keys („natural“ keys, uniqueness)
  - Foreign keys
- Business Rules (several columns / rows / tables)

Detection of Duplicates covered in separate section
Dealing with Possible Errors: Error Event Tables

- [Kimball & Caserta 2004] propose the use of **error event tables** which capture data quality scores per ETL check, table (or even column), and row. Such a solution can be designed in the form of a ... star schema:

![Diagram of error event tables]

Note: Usually, rows are a “degenerate” dimension. Instead of a user-defined primary key, physical row IDs are used.
Conforming Data

- **Conforming** is the step where not only the data structures but also the data values of a source database are mapped to the common data representation used in the Data Warehouse.

- Conforming involves the following subtasks:
  - Standardizing representations of data values
  - Matching (the same or similar) data values of different rows (usually from different sources) describing the same real-world entity [Duplicate Detection → see separate section]
  - Determining the surviving data values describing rows matched in the previous step
Conforming Dimensional Data

Remember: Dimensions supply the context in which the measures in the fact tables are viewed and compared using roll-up, drill-down, and slice ops

⇒ Conforming primarily involves:

• Standardization of formats
  - dates, phone numbers, ...
  - capitalization of strings, trimming leading & trailing blanks etc.
  - standardization of street and company names
    (e.g., suffixes like 'Street' vs 'St', 'Incorporated' vs 'Inc', ...)

• Mapping of external codes (currency, country, industry, ... codes) and internal codes (profit centers, accounting categories) to standard codes

• Mapping of source system identifiers of entities to common warehouse identifiers (based on exact matches)

• + Fuzzy matching where needed [⇒ Duplicate Detection]

Note: Also applicable to normalized representations
Conforming Facts

Conforming facts primarily involves:

- Conforming the foreign keys referring to dimensional entities, i.e., mapping source system identifiers of entities to common warehouse identifiers – using the mapping established when conforming dimensions

- Standardizing units and scales of observed measures, e.g., imperial to metric units (or vice versa), foreign exchange, etc.

- Maybe aggregation of measures to a common „higher“ dimension level

- Standardizing the computation of derived measures – if needed (*), e.g., ratios etc.

*) Recommendation:
- ask for the formulas to compute derived measures and document this computation
- procure required observed measures (inputs) and derived measures (output)
- check whether you can reproduce the procured output from its inputs ....
- ... and, if so, keep the checks but don’t recompute the outputs in your code
Typical Transformations in Conforming Step

- Relational select (sometimes called „filter transformation“), project, and join operations, unions

- Mapping source codes and identifiers to common target codes and identifiers (sometimes called „lookup transformation“)

- Some ETL tools support such transformations in a graphical „language“ (see some Informatica PowerCenter example screens on the next slides)

- Lookup transformation is essentially:

```
select ... , coalesce(m.TARGET_CODE, '*ERROR*'), ... 
from    SOURCE_TABLE s 
    left outer join MAPPING m 
        on m.SOURCE_CODE = s.GENDER
```
Filter Transformation in Informatica PowerCenter (1)

Figure 6-1. Sample Mapping With a Filter Transformation
Figure 6-2. Specifying a Filter Condition in a Filter Transformation
Joiner Transformation in PowerCenter

Figure 7-1. Sample Mapping with a Joiner Transformation
Delivering Data

- Starting point is a staging area containing a cleaned (or at least „quality scored“) and conformed package of data ready for an upload to the data warehouse.

- This data package should be logically consistent, meaning it represents a snapshot as of a specific valid time in which structural integrity constraints such as entity integrity and referential integrity hold.

- Logically, the upload establishes the most recent snapshot of the history (unless a bi-temporal model allowing for corrections of existing data is used ... ).

- Regardless of whether the dimensional design is rigidly followed, we can distinguish between an upload of
  - dimensional data, which may lead to both inserts and updates
  - fact data, which should only lead to inserts
Delivering Dimensions

The following steps are required when uploading dimensional data:

• Generation of new surrogates for dimensional objects (rows) where the (stable!) source system identifier is not known in the mapping table or in the Data Warehouse. Recommendation: For efficiency, use special DBMS or ETL Tool features.

• If no history is kept for properties of dimensional objects (rows) – SCD\(^*\) Type 1 – update existing rows and insert new ones.
  Recommendation: For efficiency, use a DBMS „Bulk Load“ facility, and not an SQL merge statement:
  ```sql
  merge into TARGET_TABLE t using SOURCE_TABLE s on ( s.PK = t.PK )
  when matched then update set ... where ...
  when not matched then insert ( ... ) values ( ... )
  ```

• If history is kept, insert (and possibly update) rows according to the strategy used:
  - **SCD Type 2**: insert a new row with a new surrogate and update the table mapping the source system ID to the Data Warehouse surrogate
  - **general temporal database concepts**: insert a new row with the same surrogate and the \texttt{VALID\_FROM} time, and set the \texttt{VALID\_TO} time of the existing row

• Remember: Do not delete dimensional objects (rows) that were deleted in the source.

\(^*\) SCD = \textit{Slowly Changing Dimension}
Delivering Facts

Uploading facts is generally easier, and involves the following steps:

• Map all foreign keys to the appropriate surrogates.

• Insert fact tables with the appropriate valid time (which maybe a foreign key to a date/time dimension table).
  If performance matters, consider
  - switching foreign key checks off temporarily (or permanently, but with periodic „background“ checks by running queries)
  - dropping indexes and re-building them (especially for bitmap indexes)

• Recompute dependent materialized tables (typically aggregates) or MOLAP (Multi-dimensional OLAP) cubes.
Literature

General Data Warehousing

[Lehner 2003]

General Data Quality / Data Profiling

[Olson 2003]

ETL Processing

[Kimball & Caserta 2004]