Extract – Transform - Load

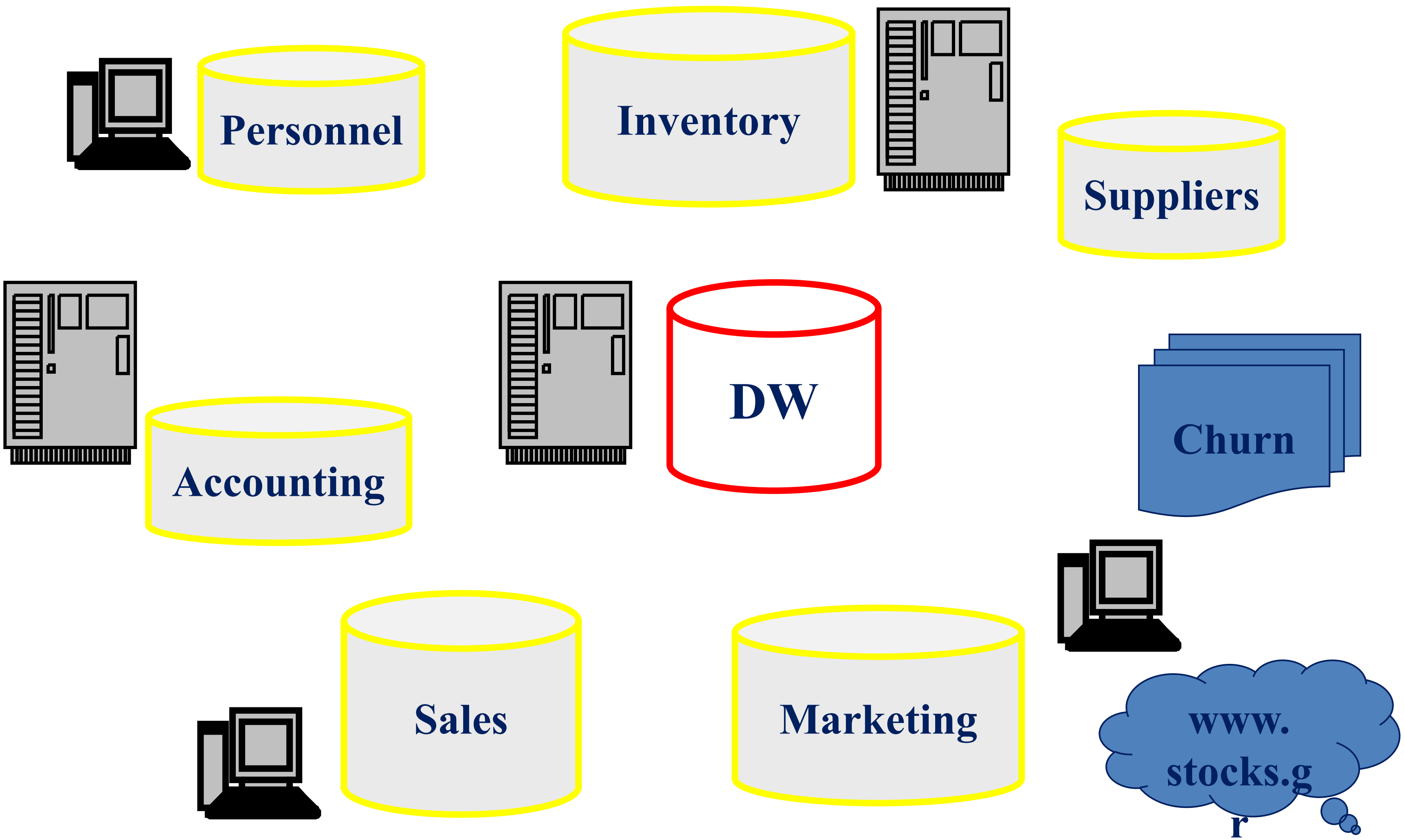
Dr. George Papastefanatos
Researcher, Athena Research Center
gpapas@athenarc.gr

MSc in Business Analytics
25/06/2020

What is a Data warehouse

- The data warehouse is a huge repository of enterprise data that will be used for decision making
- Data is collected from multiple data sources, cleansed and organized in data warehouses
- After data is loaded in the data warehouse, (OnLine Analytical Processing) OLAP cubes are often pre-summarized across dimensions of interest to drastically improve query time

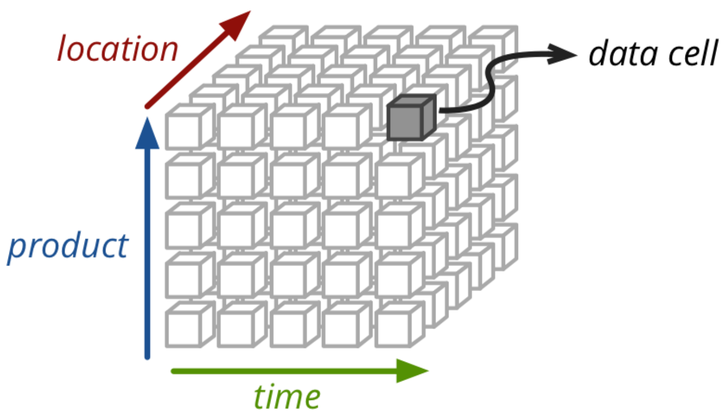
DW Example – Telecom Co.



Multidimensional model

- Analysis of a set of quantitative observations (measures)
  - Sales, cost, stock, population, etc.
- Over a set of context parameters that identify each observation (dimensions)
  - Date, product, location, sales person
  - Each having different levels (hierarchies) of details, e.g., date refers to day, month, year; a product belongs to a hierarchy of categories, etc.
- Cubes: combination of dimensions that defines a set of measures
  - E.g, Sales ($$$) per product, date and location

Multidimensional model



Dimensions: Product, Location, Time

Hierarchies

Industry
Category
Product

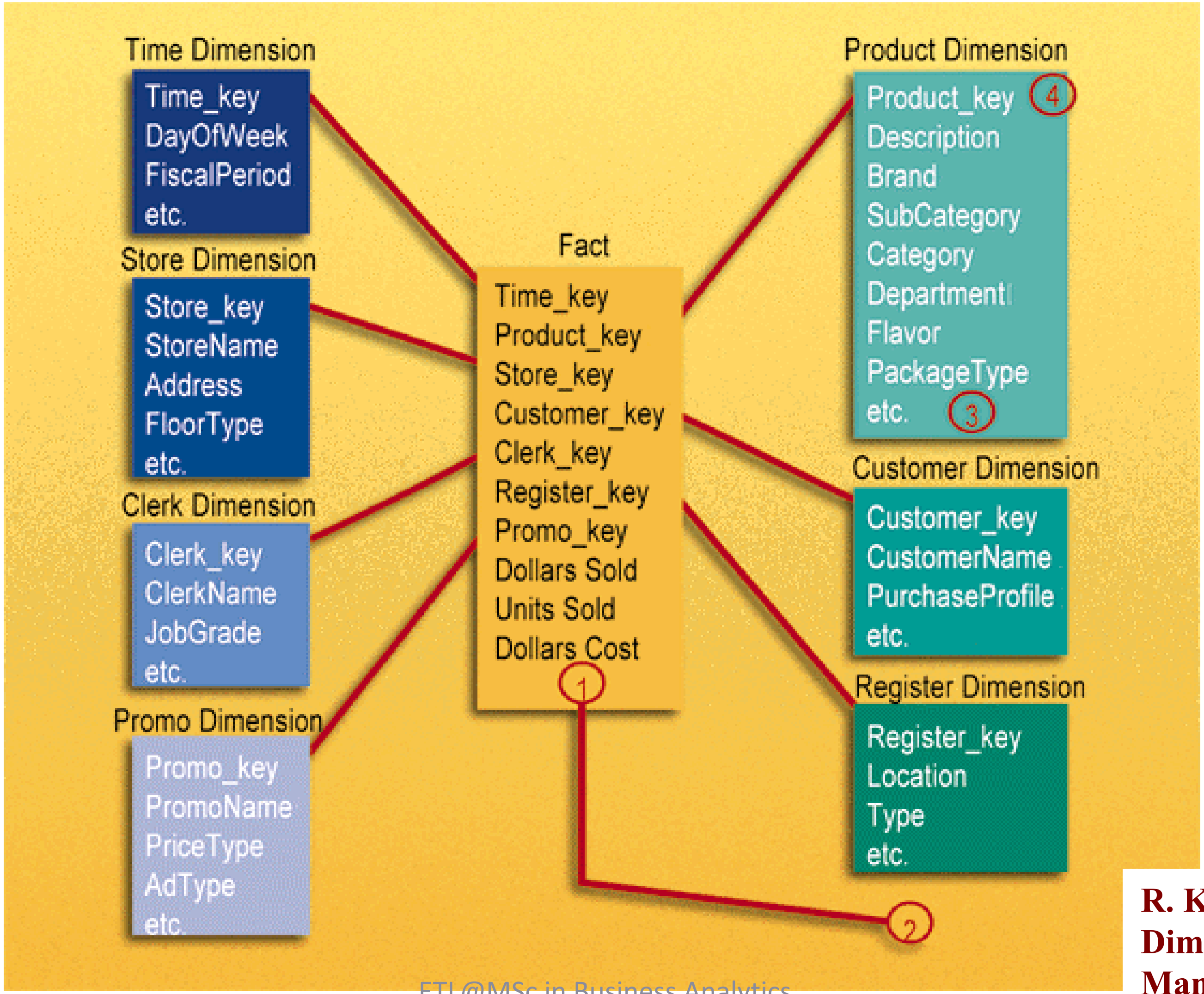
Country
Region
City
Store

Year
Quarter
Month
Week
Day

Multidimensional model

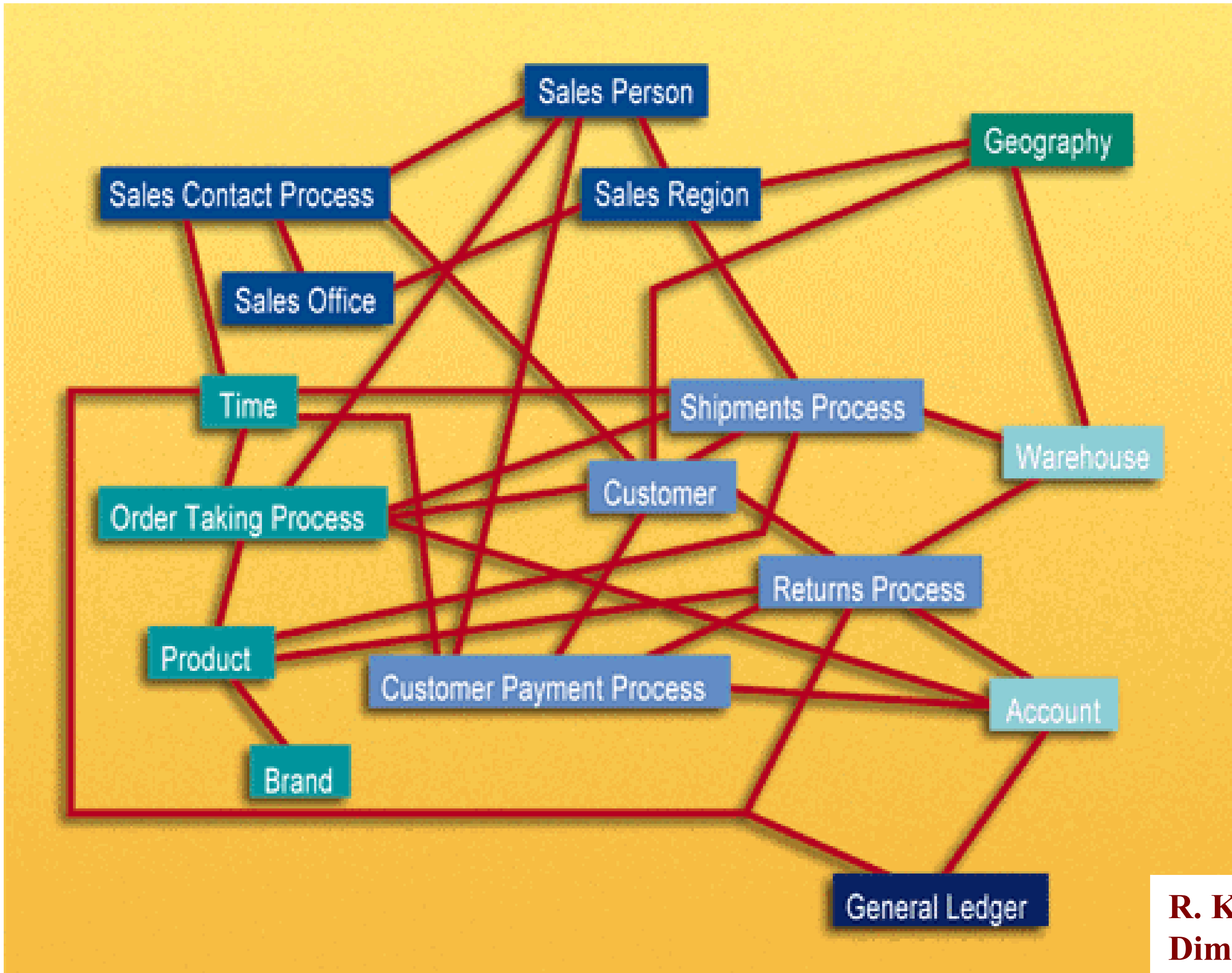
- Dimension tables : Contains info about a dimension. It identifies a dimension value through a unique key as well as (if dimension is hierarchical) with the dimension level.
- Fact table : The table that implements the cube
  - Each record corresponds to a data cell in the cube
  - For each dimension value, there is a key to the dimension table
  - For each measure there is a single column
  - Primary key of the Fact table is the combination of the dimension keys. (Cell coordinates)

Multidimensional Modeling



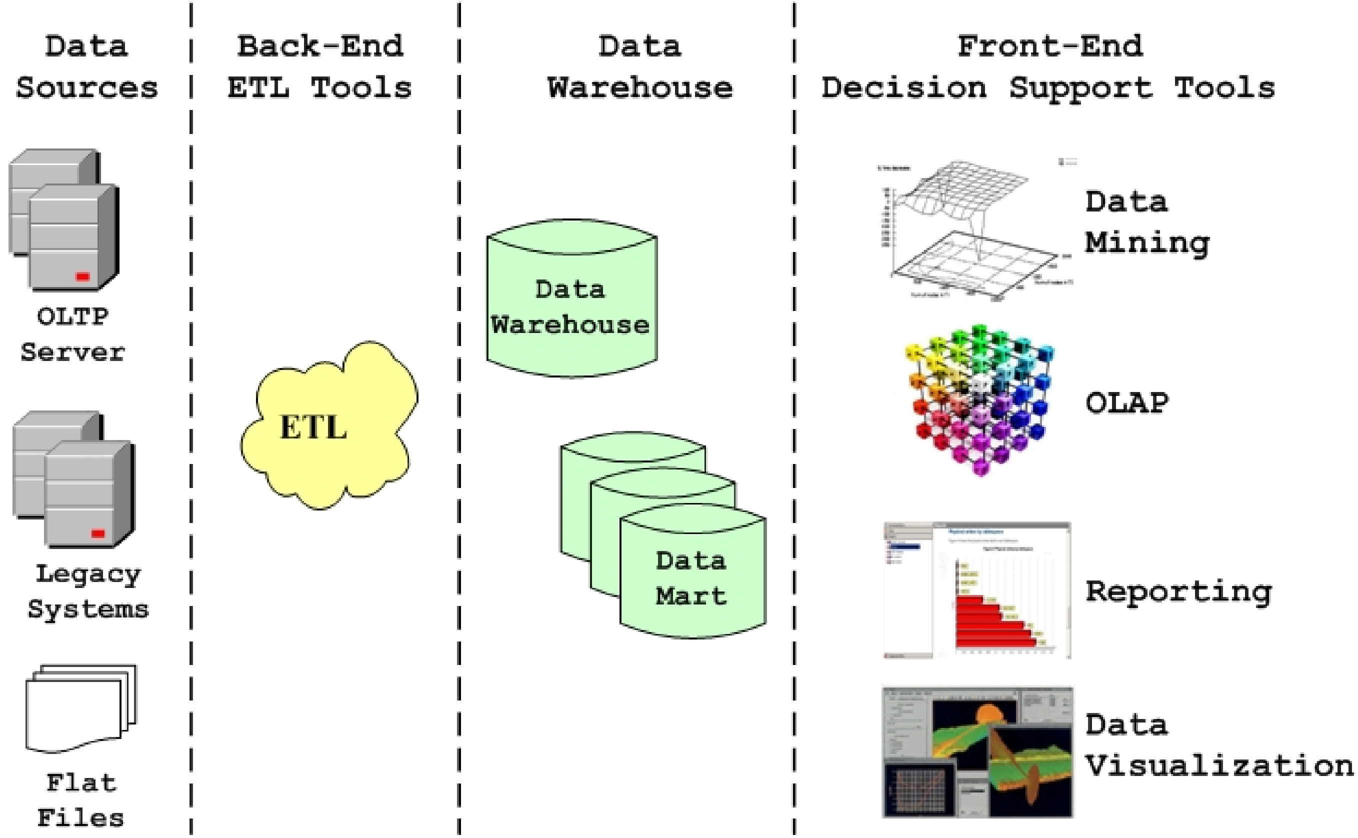
R. Kimball, A Dimensional Modeling Manifesto, DBMS Magazine, Aug. 1997

Traditional Relational Model of the previous example

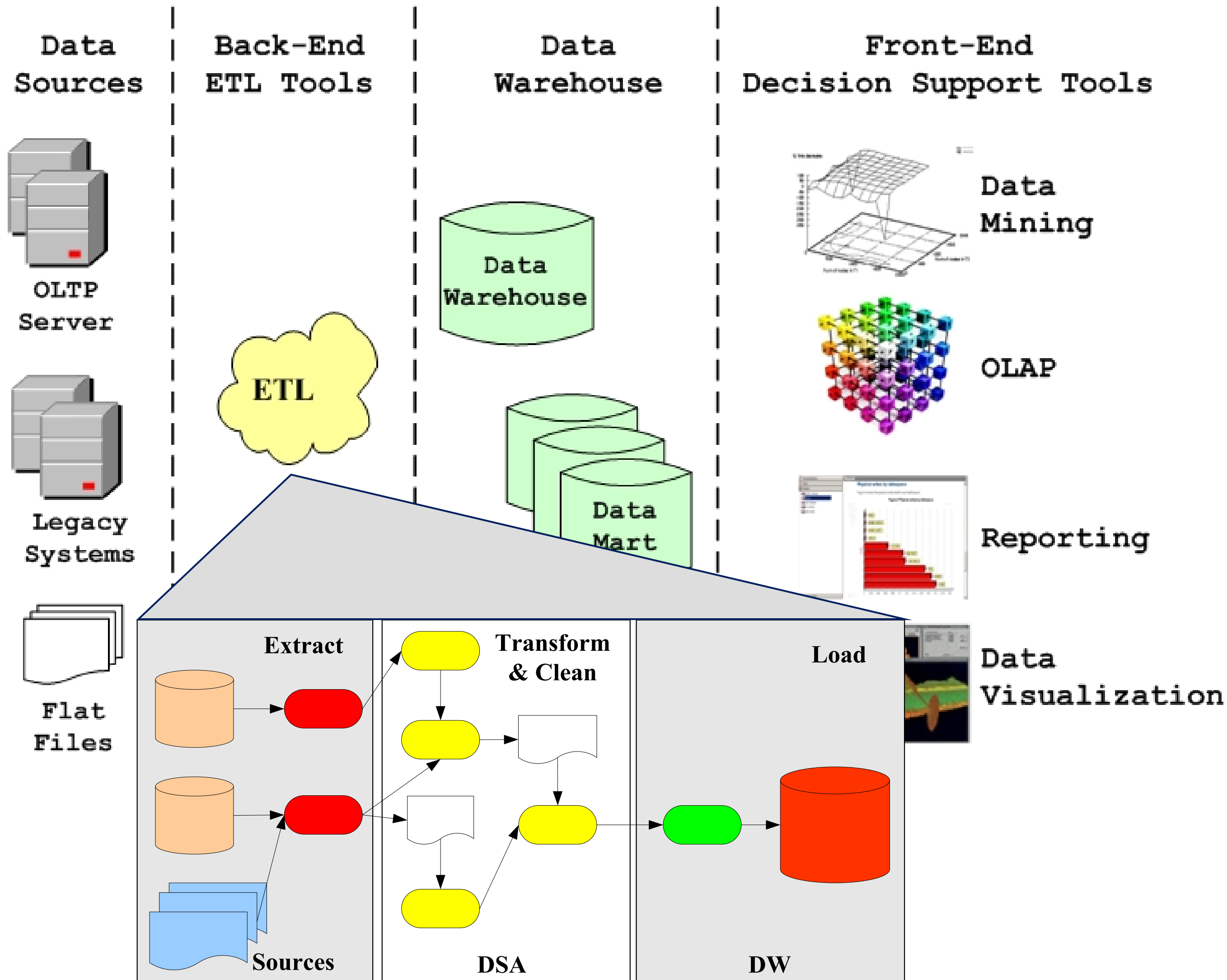


R. Kimball, A Dimensional Modeling Manifesto, DBMS Magazine, Aug. 1997

Data Warehouse Environment



Extract-Transform-Load (ETL)



What is an ETL process?

- Initial loading and updating of a data warehouse with data from the sources is done with a multi-level ETL workflow (extract, transform & load)
  - **E: export** + transfer +
  - **T: transformation** + cleaning +
  - **L**: data **loading** in the Data Warehouse.
- The standard execution is in the form of a **workflow**.
- Any intermediate data storage to serve the ETL process takes place in a storage area called **Data Staging Area (DSA)**.
  - also serves for temporary (for a short time) storage of source data for reasons of debugging, provenance,…

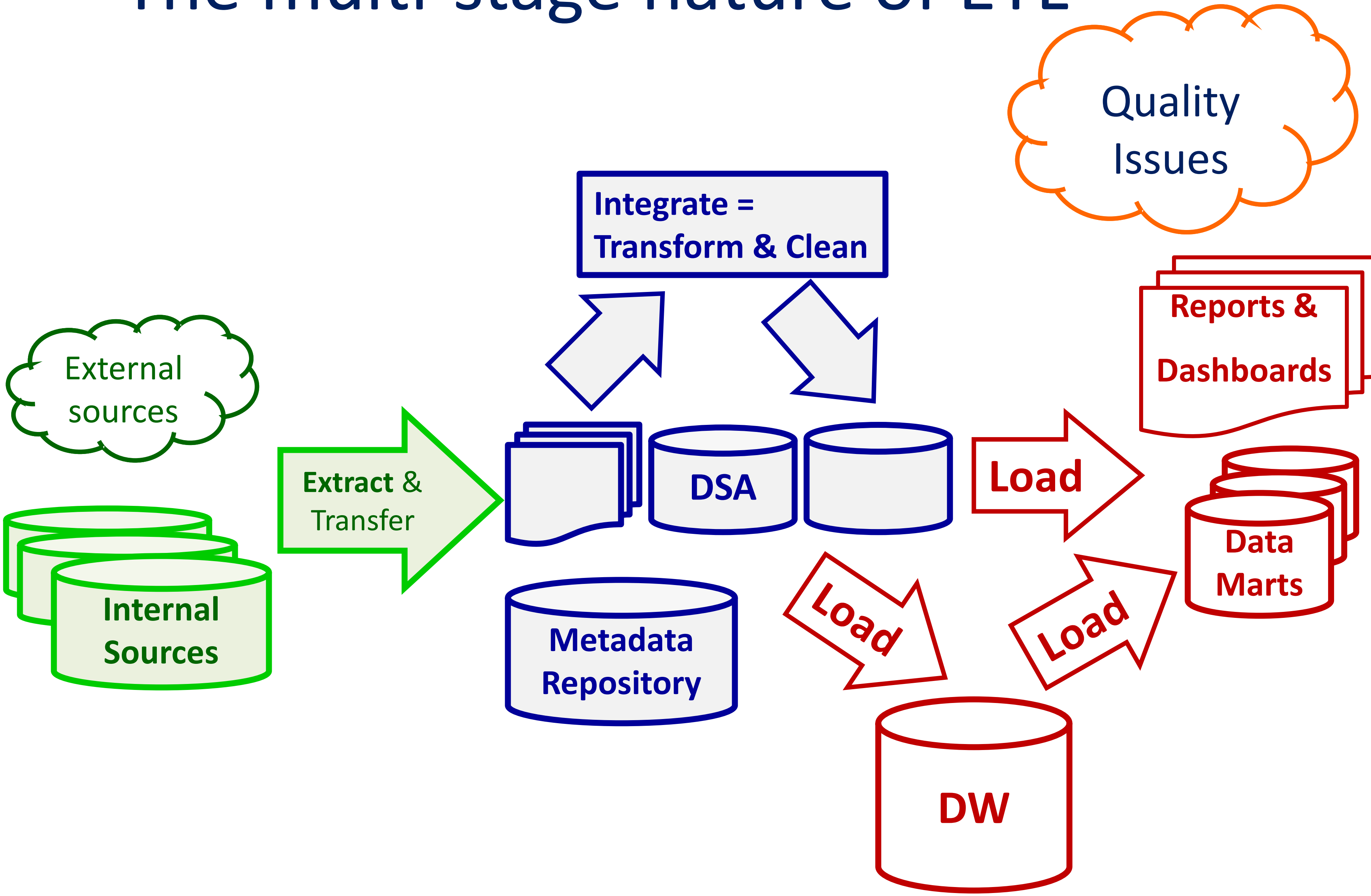
Why we need ETL processes?

- Data warehouses contribute to an organization's data architecture by **integrating data** from various internal information systems, but also from external sources.
- The goal is to OLAP, reporting & dashboard applications to be implemented over a **single**, **consistent** and **complete** data set with various quality guarantees.
- The basic guarantee concerns the elimination of inconsistencies (different values in the data for the same thing in the physical world) and errors in the data and is summarized with the term **single version of the truth**
  - No errors
  - Data consistency (between different data measuring / representing the same real world entity)
- The second guarantee concerns the availability of all data and mainly concerns their **completeness** and **freshness**.
  - Completeness (no critical data are missing)
  - Freshness (as fresh as possible)

70% of cost / time / …
@build : design and test
@maintenance: redesign, change requests

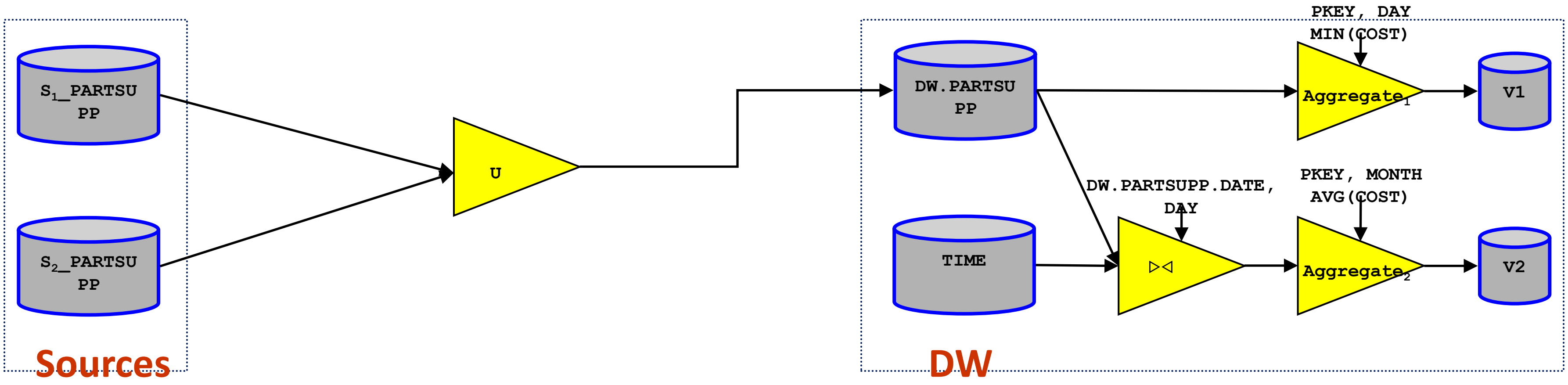
30% @execution, monitoring and data debugging

The multi-stage nature of ETL

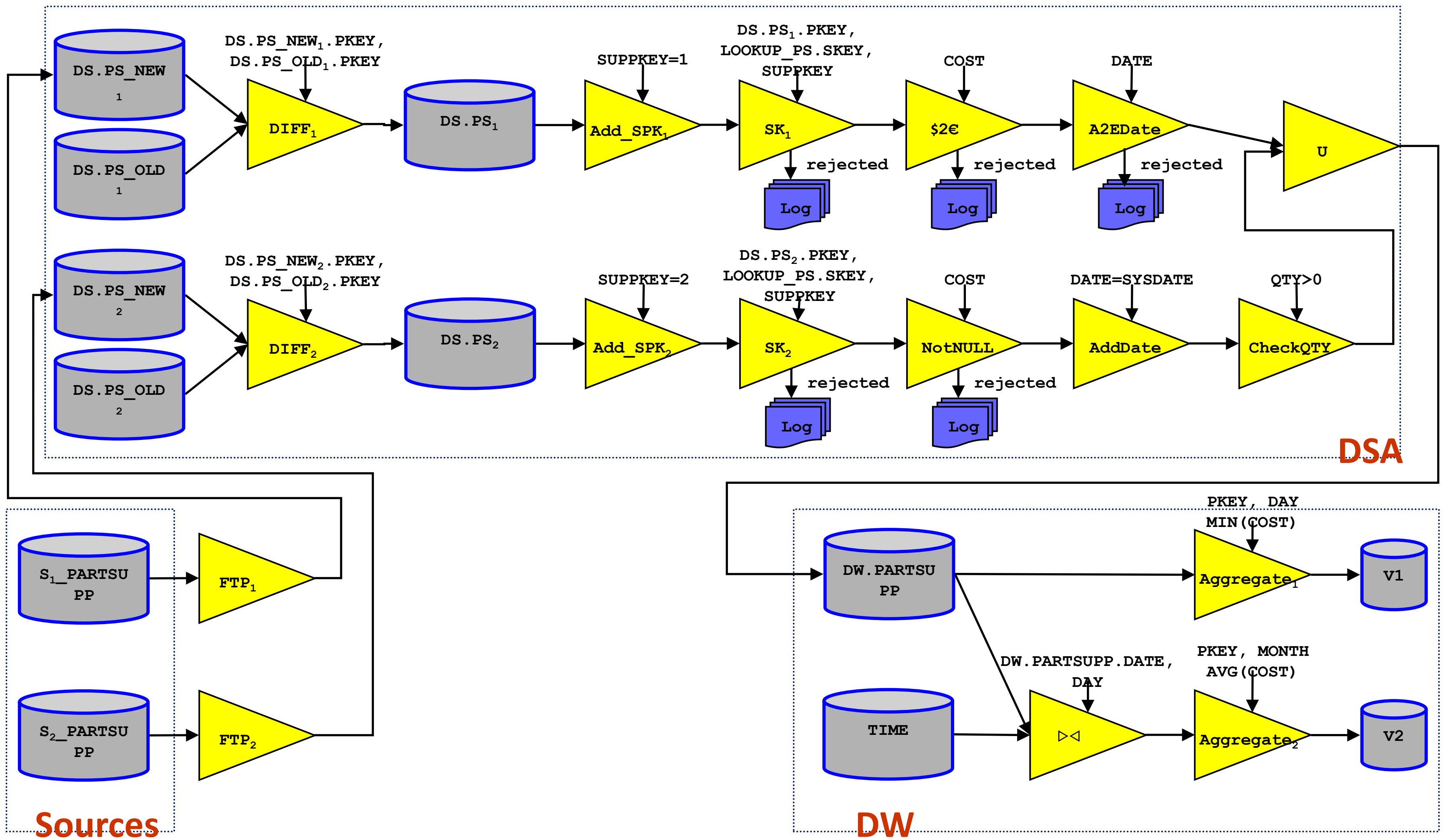


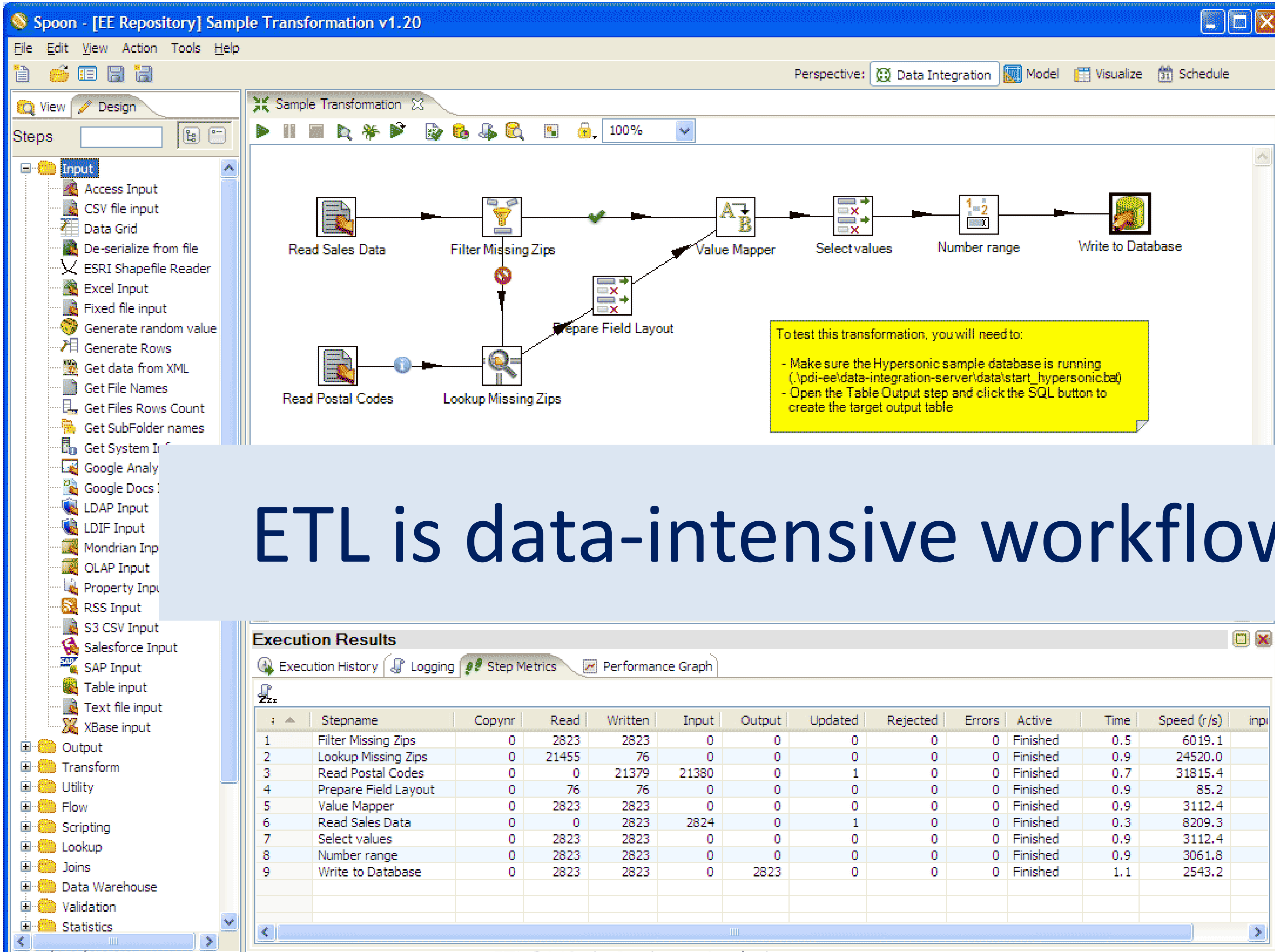
DW = Materialized views?

Notation from: P. Vassiliadis, A. Simitsis, S. Skiadopoulos. Modeling ETL Activities as Graphs. DMDW 02



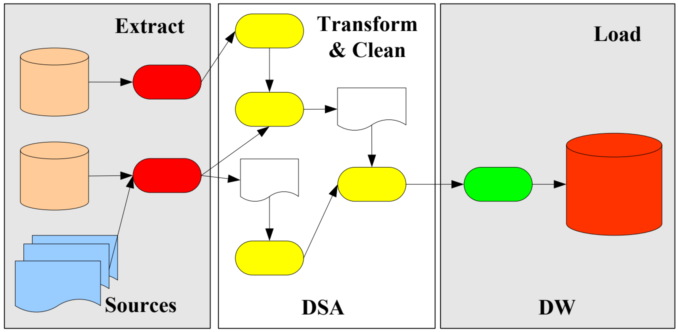
DW ≠ Materialized views!





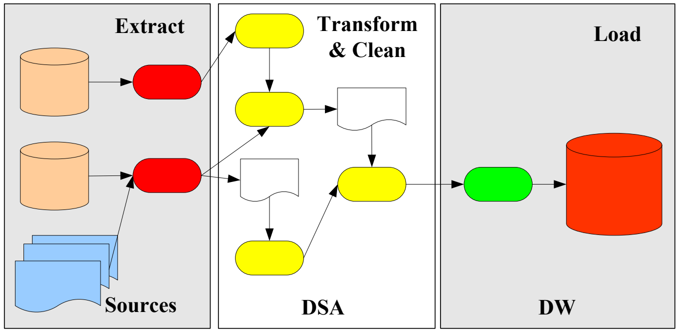
ETL is data-intensive workflow

ETL ingredients



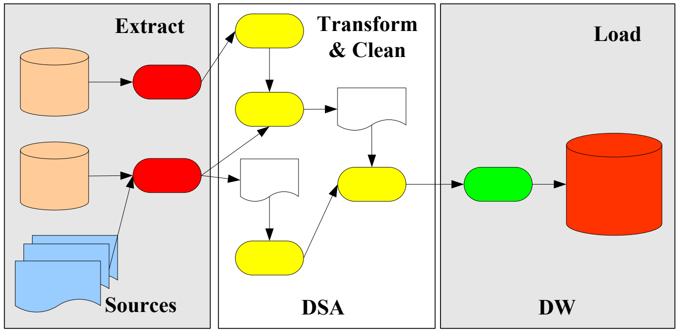
- **Extract** (data from their sources)
  - find only the data that you need (e.g., only the increments wrt previous refresh)
  - with minimal overhead for the source systems
  - as quickly as possible

ETL ingredients



- **Extract** (data from their sources)
  - find only the data that you need (e.g., only the increments wrt previous refresh)
  - with minimal overhead for the source systems
  - as quickly as possible
- **Transform** (the data to a consistent, DW-compliant format, wrt both schema+values)
  - Surrogate Keys !!!
  - compute any functions, value transformations, KPIs, …
  - schema restructuring (from source to target schema)
  - clean!

ETL ingredients



- **Extract** (data from their sources)
  - find only the data that you need (e.g., only the increments wrt previous refresh)
  - with minimal overhead for the source systems
  - as quickly as possible
- **Transform** (the data to a consistent, DW-compliant format, wrt both schema+values)
  - Surrogate Keys !!!
  - compute any functions, value transformations, KPIs, …
  - schema restructuring (from source to target schema)
  - clean!
- **Load** (the data)
  - to the tables of both the DW and the data marts
  - update indexes and refresh any materialized views
  - refresh reports, spreadsheets, …

DW refreshment & Requirements

- Operational challenges
  - Data Quality: users have requirements on the
    - **completeness**,
    - **correctness** (remember: single version of the truth) and
    - **freshness** of DW data
  - (hard) time constraints: complete refreshment within a time window (e.g., within a couple of hours, such that daily reports are prepared)
  - The sources must not be overloaded or significantly reconfigured
  - Resilience & Recovery from failures

Initial build vs DW Refreshment

- **Initial Build**: refers to the bulk loading of data to initiate the contents of a (DW) table.
  - Happens once
  - Serves as an initial feed + initial testing of the ETL flow
  - ATTN: MUST **test ETL adequately** before going to production!!
  - Special care for dimensions (see the “Transformation” part)

- **DW refreshment**: as data change at the sources, the DW needs to be updated. When and how? Old dilemma:
  - On update (whenever a change occurs at the sources)
  - On demand (whenever a query requests new data)
  - **Periodic** (the only viable solution)

Incremental, periodic DW Refreshment

- Typically **nightly**
  - as users are more happy, frequency can increase
- The goal is to add the new data + sync the DW on any updates & deletions(rare) that took place at the sources
  - Sometimes, the inverse is also part of the goals: as data cleaning happens at the DW, push back clean data at the sources to replace erroneous one
- Order of execution:
  - Highly depends on the prioritization of freshness & completeness by the upper management (not all tables are equal)
  - For every "sub-schema", though:
    - **Dimensions first (ATTN: handling of keys SUPER IMPORTANT)**
    - Facts later

E FOR EXTRACT

Extract

- Goal
  - find changes in data sources; i.e., new/deleted/updated tuples
  - fast extract of relevant data
    - extract from source systems can take a **long** time
- Techniques
  - use full or differential snapshots of source data
    - **too** time consuming to ETL all data at each load
      - can take days/weeks
    - drain on the operational systems and DW systems
    - extract/ETL only changes since last load (delta)
- Constraints
  - limited time window
  - minimized overhead on operational (OLTP) systems
  - minimize changes on the software configuration of the OLTP systems

Transfer

- Compression
  - network bandwidth, stability

- Encryption
  - security

Extract

- Where and How we compare the full snapshots?
  - sources? Data Staging Area (DSA)?
  - partial file comparisons, hashing?

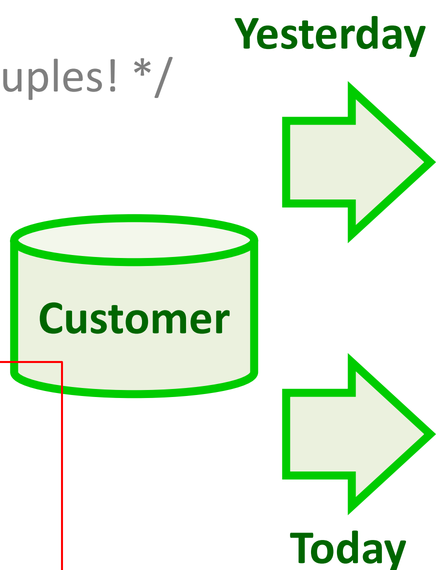
- Differential techniques
  - Inherent change data capture provided by vendors (recommended)
    - Based on log sniffing – fast
  - change data entry programs – risky and costly
  - use triggers – not very often any more

- Heavy use of files with ETL tools (expensive, but more functionality) or house made scripts (cheap, but we have to implement all operations)

Differentials via snapshot comparison

- Assume you keep the snapshot of the source of the previous load S_{prev}+ you have the current snapshot S_{curr}
- The minus operation gives the changes since the last load:
  - $D^{+} = S_{curr} - S_{prev}$
  - $D^{-} = S_{prev} - S_{curr}$

  /*no updates unless you compare entire tuples! */



Yesterday:

CID	Name	Age
10	Mary	18
20	Jack	4
30	Joe	20

Today:

CID	Name	Age
10	Mary	18
20	Jack	**5**
40	John	7

Morale (not only for DW):
+ updates can be treated as sequences of DEL;INS
… but …
- it's not always equivalent (unless you take care)
- it's impossible if you have foreign keys ☹

Differentials via snapshot comparison

- A **simple** but **slow** algorithm can extract INS, DEL, UPD by checking the two snapshots
  - Think of it as a nested-loops variant – any other join works
  - Update: the key is the same and we check changes at important attributes
  - Faster variants by sorting and using window comparisons
- Typically, **heavy to perform** @ source => requires transfer of the source's snapshot to the DSA
  - workable for small sources
  - impossible for large sources
  - **Remember: you need to do it to ALL the tables you load!**

Overall: the minus technique, although simple, is potentially slow and impractical

CHECKLIST
☺ correct deltas
😐 limited time window
😐 minimize source overhead
☺ minimize configuration changes

XOR

Old tricks (that occasionally worked)

What if we compromise the "avoid interfering with source configuration" constraint?

- Trick #1: Timestamping the rows of the sources
  - Put a timestamp column in each source table
  - Remember at which timestamp the extraction stopped the last time
  - Only additions considered- Lose deletions and updates ☹
- Trick #2: Flag the rows of the sources
  - Put a "flag" column in each source table: "I-have-changed"
  - Modify source applications / add triggers to populate flag ☹ ☹ ☹ ☹ ☹
  - Can complement timestamps for DEL/UPD

CHECKLIST

☹ correct deltas
☺ limited time window
☺ minimize source overhead
☹ minimize configuration changes

XOR

Old tricks (that occasionally worked)

What if we compromise the "avoid interfering with source configuration" constraint?

- Trick #3: Triggers (or "why good, sexy ideas often fail")
  - Add trigger per monitored source table: on INS/DEL/UPD, trigger copies delta to a dedicated table
  - No modification of source applications, no errors, fast ETL
  - Extremely painful for source overhead ☹ ☹ ☹ ☹ ☹
- Trick #4: Message queues
  - Applications do not modify the source db/files; instead, they use message queues, which in turn, perform the update at the source AND populate the delta-dedicated tables
  - Modifies apps ☹ ☹ ☹; Painful for source overhead ☹

<u>TRIGGERS</u>
☺ correct deltas
☺ limited time window
☹ ☹ minimize source overhead
☺ minimize configuration changes

<u>MSG QUEUES</u>
☺ correct deltas
☺ limited time window
☹ minimize source overhead
☹ ☹ minimize configuration changes

Change Data Capture: Use the Log!

- What practically works nowadays
  - provided that the source is a relational database with a log file
  - not always possible: if not, you ‘re back at the minus method
- Sniff the log of the source for changes!
  - Does not affect applications
  - Turns out to be both fast and lightweight
  - No data loss

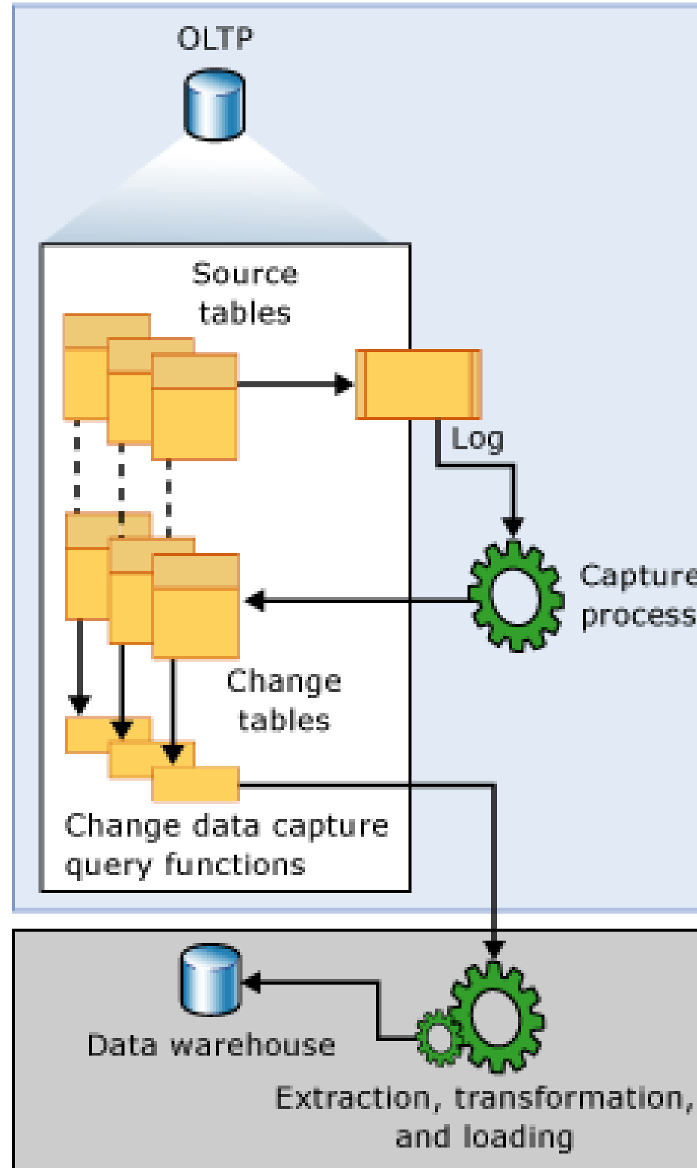
- The big vendors will give you the tools to do it for you; you just have to register the source tables that are monitored

Change Data Capture

https://msdn.microsoft.com/en-us/library/cc645937.aspx

"The source of change data for change data capture is the SQL Server transaction log. As inserts, updates, and deletes are applied to tracked source tables, entries that describe those changes are added to the log. ***The log serves as input to the capture process. This reads the log and adds information about changes to the tracked table's associated change table****. Functions are provided to enumerate the changes that appear in the change tables over a specified range, returning the information in the form of a filtered result set. The filtered result set is typically used by an application process to update a representation of the source in some external environment. "*

"The capture job is started immediately. It runs continuously, processing a maximum of 1000 transactions per scan cycle with a wait of 5 seconds between cycles. "



CHECKLIST

☺ correct deltas
☺ limited time window
😐 minimize source overhead
☺ minimize configuration changes

… and C for Cleaning …

T FOR TRANSFORM

Transformations overview

- Basic transformations:
  - **Cleaning:** Mapping NULL to 0 or "Male" to "M" and "Female" to "F," date format consistency, etc.
  - **Deduplication:** Identifying and removing duplicate records
  - **Format revision:** Character set conversion, unit of measurement conversion, date/time conversion, etc.
  - **Surrogate keys :** Establishing key relationships across tables

Transformations overview

- Advanced transformations:
  - **Derivation** Applying business rules to your data that derive new calculated values from existing data – for example, creating a revenue metric that subtracts taxes
  - **Filtering:** Selecting only certain rows and/or columns
  - **Joining:** Linking data from multiple sources
  - **Splitting \ Forking:** Splitting a single column into multiple columns
  - **Merging:** Merging multiple columns data into a single one
  - **Data validation:** Simple or complex data validation – for example, if the first three columns in a row are empty then reject the row from processing
  - **Summarization:** Values are summarized (sum, avg, min, max, etc) to obtain total figures

Surrogate Keys

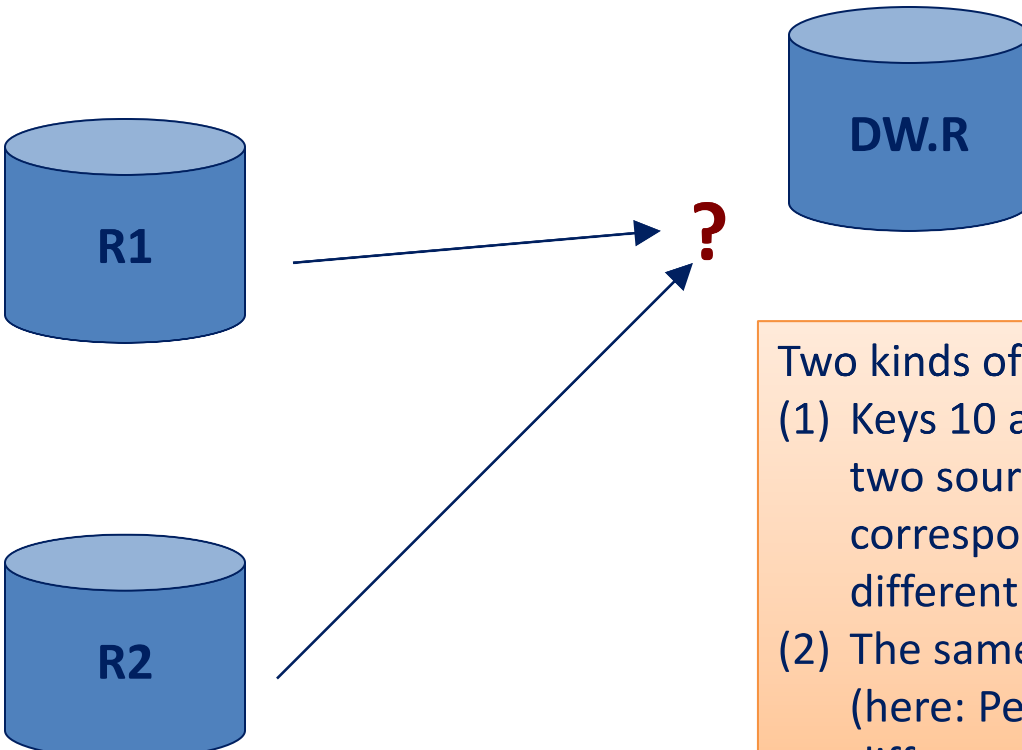
- Source keys are usually called **production** keys or **natural** keys.
- The new, homogenized keys, are called **surrogate** keys

- There are special techniques for how to change the keys in the Data Warehouse, if you change a key in a source ...

Surrogate Keys

ID	Descr
10	Coca
20	Pepsi

ID	Descr
10	Pepsi
20	HBH



ID	Descr
??	??
??	??

Two kinds of conflicts:

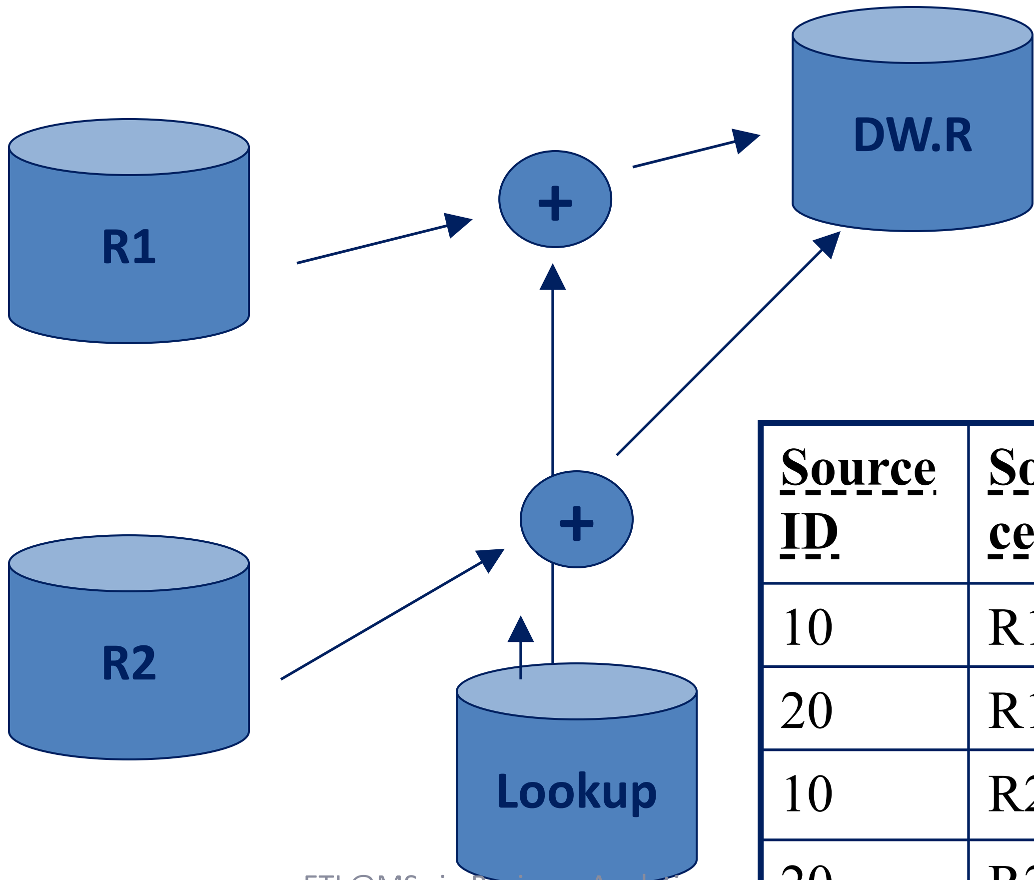
(1) Keys 10 and 20 in the two sources correspond to different products

(2) The same product (here: Pepsi) has different keys in the two sources

Surrogate Keys

ID	Descr
10	Coca
20	Pepsi

ID	Descr
10	Pepsi
20	HBH



ID	Descr
100	Coca
110	Pepsi
120	HBH

Source ID	Source	Surrogate Key
10	R1	100
20	R1	110
10	R2	110
20	R2	120

ALWAYS

USE

SURROGATE KEYS

More transformations

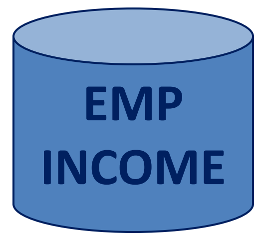
- Schema modification
  - DW schemas are typically different than sources schemas
    - e.g., source data may be unstructured
- Value change/computation
  - source tuples may have different format, type, value
    - integer → real
    - euro → dollar
  - new values may need to be created
    - date of birth → age

More transformations

- Data type conversions
  - EBCDIC → ASCII/Unicode
  - String manipulations , e.g., Name → lastname , firstname
  - Date/time format conversions
    - E.g., Unix time 1201928400 = what time?
- Normalization/denormalization
  - To the desired DW format
  - Depending on source format
- Building keys
  - Table matches production keys to surrogate DW keys
  - Correct handling of history - especially for total reload

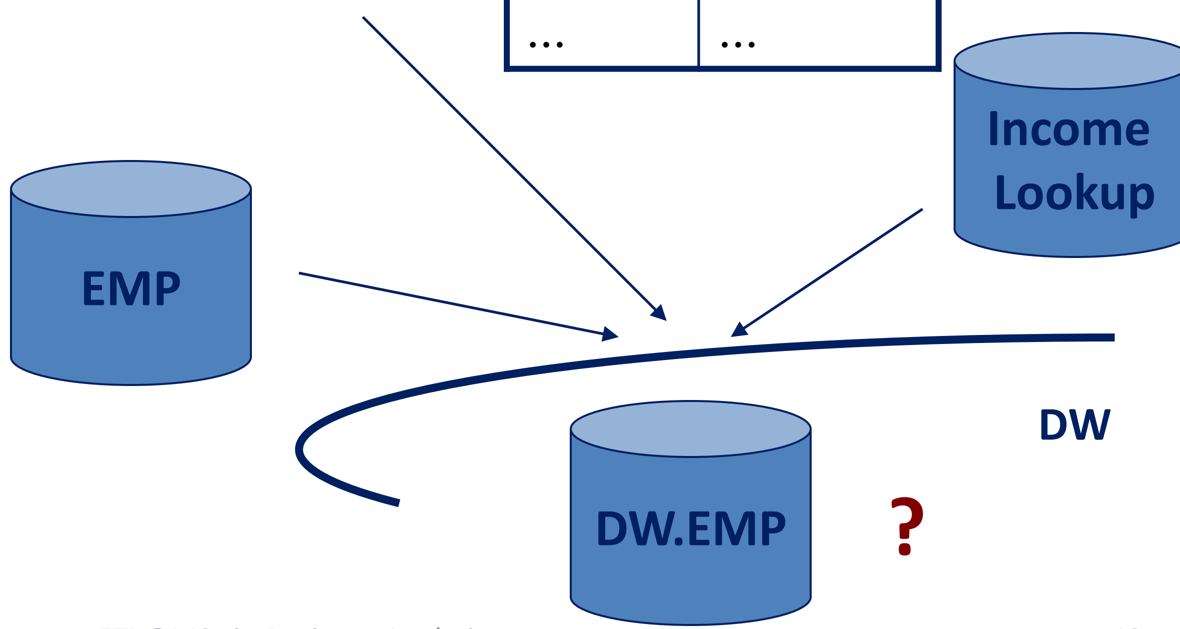
Denormalization

EMP ID	IL_ID	Amount
110	10	1500
110	30	300

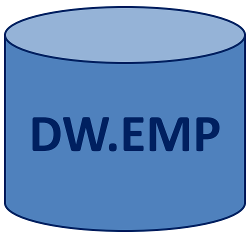


IL_ID	Descr
10	Salary
20	Bonus
30	Tax
…	…

EMP ID	Name	Age
110	Bob	30
120	Rob	48
130	Ron	29



Denormalization



EMP ID	**Name**	**Age**	**Salary**	**Tax**	**Bonus**
110	Bob	30	1500	300	NULL
…	…				…

- Flat tables not so easily evolvable
  - add new income category => new column
- Fast to answer queries
  - for a single question we save 2 joins
- The opposite (normalization) may be used too

Data Validation

- Data violating DB rules
  - duplicates, primary/foreign key violations, out-of-range values, …
  - logical rule violations

```
IF (SEX='F' AND ILLNESS='PROSTATE')
THEN (ALERT ERROR MESSAGE)
```

- Homonyms and conflicts
- Missing data
- Renicing
  - e.g., strings like addresses

Data cleansing

Source Value	DW value
HP	HP
H.P.	HP
H-P	HP
Hewlett-Packard	HP
Hioulet-Pakard	HP
DEC	DEC
Digital Co.	DEC
…	…

- Synonym table
  - addresses
    - ave, st, blvd, …
  - names
    - Mr John Doe / Dr John Doe / J. Doe
- Regular expressions
  - e.g., perl

Data cleansing

- Do not use "special" values (e.g., 0, -1, 999) in your DW
  - They are hard to understand in query/analysis operations
- Annotate facts with Data Status dimension
  - Normal, abnormal, outside bounds, impossible,…
  - Facts can be taken in/out of analyses
- Uniform treatment of NULL
  - Use NULLs only for measure values (estimates instead?)
  - Use special dimension key (i.e., surrogate key value) for NULL dimension values
    - E.g., for the time dimension, instead of NULL, use special key values to represent "Date not known", "Soon to happen"
    - **Avoids problems in joins, since NULL is not equal to NULL**

Data Status Dimension

SID	Status
1	Normal
2	Abnormal
3	Out of bounds
…	…

Sales fact table

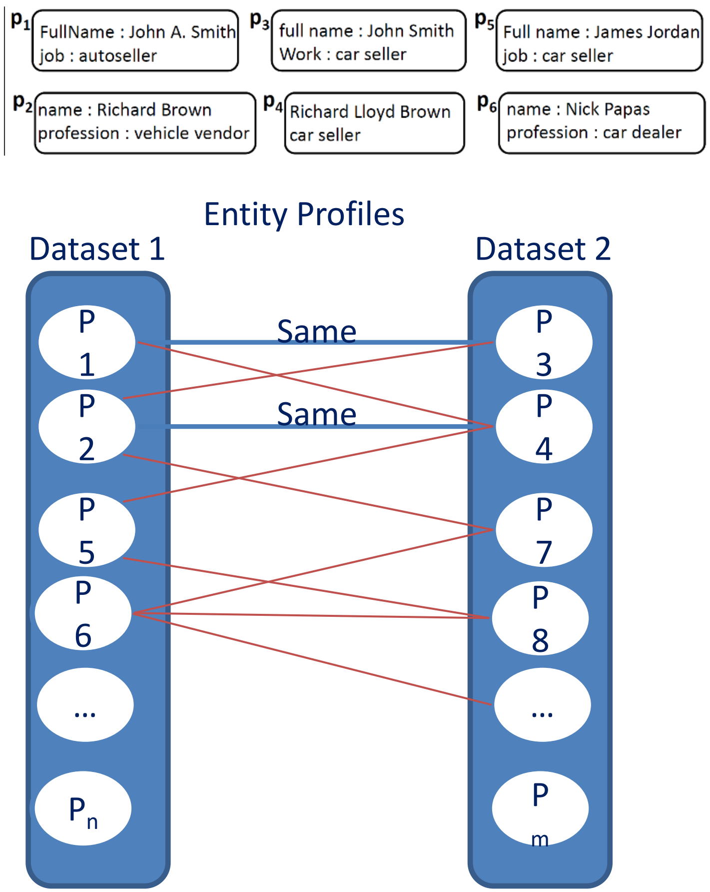
Sales	SID	…
10	1	…
20	1	…
10000	2	…
-1	3	…

For highly heterogenous sources

Entity resolution = Data deduplication = Data interlinking

is needed

Entity Resolution (ER)



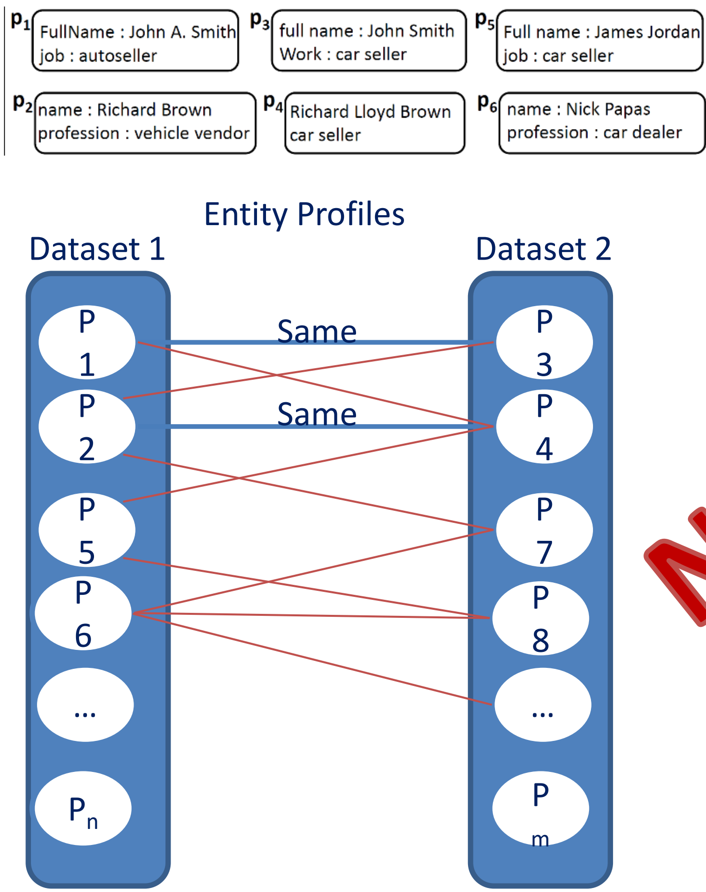
- Identifies and aggregates the different entity profiles that actually describe the same real-world object.
- Applications:
  - Duplicate detection – Dirty ER
  - Record linkage – Clean Clean ER

Time Complexity → quadratic

$O(n^2)$

Every entity is compared with all others.

Entity Resolution (ER)



- Identifies and aggregates the different entity profiles that actually describe the same real-world object.
- Applications:
  - Duplicate detection – Dirty ER
  - Record linkage – Clean Clean ER

Next Lecture

Time Complexity → quadratic

$O(n^2)$

Every entity is compared with all others.

Mostly: Slowly
but also: Rapidly

... and **why Surrogate Keys are SUPER IMPORTANT** …

Several figures taken from
https://en.wikipedia.org/wiki/Slowly_changing_dimension

CHANGING DIMENSIONS

The problem

- Dimensions are used to form a **dimension bus**, over which fact tables are "glued" (via FK's)

- At the initial DW built/loading, both the dimension data and the factual data are snapshot.
- But then, data change. Let's start with the dimensions first:
  - new dimension values are added, some of them are updated and maybe, some are deleted.

- How do we handle change of the dimensions?
- MUST read: R. Kimball et al., the DW toolkit

Assume a simple update

@ Source

Supplier_Code	Supplier_Name	Supplier_State
ABC	Acme Supply Co	CA

Supplier_Code	Supplier_Name	Supplier_State
ABC	Acme Supply Co	IL

… well it’s not so simple…

Dim Table: Supplier

SuppK	Code	Name	State
121	AAA	X	CA
122	BBB	Y	NY
123	ABC	Z	CA
…			

Fact Table: Supplies

SuppK	ProdK	Date	Amt	…
123	100	160303	100	
123	200	160303	20	

16-03-04 comes and the supplier changes State
+
New data arrive for this supplier

123	Abc	Z	IL

SuppK	ProdK	Date	Amt	…
123	100	160304	30	
123	200	160304	10	

… and assume this state…

Dim Table: Supplier

SuppK	Code	Name	State
121	AAA	X	CA
122	BBB	Y	NY
123	ABC	Z	IL
…			

Fact Table: Supplies

SuppK	ProdK	Date	Amt	…
123	100	160303	100	
123	200	160303	20	
123	100	160304	30	
123	200	160304	10	

We successfully updated the dimension table's row …

… retained the same Surrogate Key …

… and appended the new data…

… and assume this state…

Dim Table: Supplier

SuppK	Code	Name	State
121	AAA	X	CA
122	BBB	Y	NY
123	ABC	Z	IL
…			

Fact Table: Supplies

SuppK	ProdK	Date	Amt	…
123	100	160303	100	
123	200	160303	20	
123	100	160304	30	
123	200	160304	10	

… and someone asks: "how many shipments did we have from CA this month?"

Remember: this IL used to be CA!

The correct answer should be **120!**

Now, we are going to answer: 0!

What do we do?

- Slowly Changing Dimensions:
  - The dimension values, change…
  - … but not too often – certainly, much more rare than factual data do
  - … and we need to handle change in dimensions, in the presence of factual foreign keys

- Several solutions, all known as **SCD Type X**, with X ranging from 0 to (hmm, at least) 7
- **Basically, SCD type 1,2,3 are the most important**

SCD Type 1

- **Type 1**: Simply replace the old value with the new one!
- Issues:
  - Keep the same surrogate key!
  - How to detect change?
    - If the Production Key exists, assign the same SK, … and…
    - overwrite!
- Easy
- Fast
- Does not augment the dimension table
- Misses the previous value => historical queries to the facts & dimensions are incorrect!

SCD Type 3

@ Source

Supplier_Code	Supplier_Name	Supplier_State
ABC	Acme Supply Co	CA

Supplier_Code	Supplier_Name	Supplier_State
ABC	Acme Supply Co	IL

@ DW

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	CA

Supplier_Key	Supplier_Code	Supplier_Name	Original_Supplier_State	Effective_Date	Current_Supplier_State
123	ABC	Acme Supply Co	CA	22-Dec-2004	IL

- **SCD Type 3**: extra attribute in the dimension row with the previous value
- Almost Type 1 with little more info
- Same SK, fact data refer to the correct SK

- Must take special care for historical queries (doable but hard ☹)

SCD Type 2

@ Source

Supplier_Code	Supplier_Name	Supplier_State
ABC	Acme Supply Co	CA

Supplier_Code	Supplier_Name	Supplier_State
ABC	Acme Supply Co	IL

@ DW

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	CA

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Version
123	ABC	Acme Supply Co	CA	0
124	ABC	Acme Supply Co	IL	1

(a)

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date
123	ABC	Acme Supply Co	CA	01-Jan-2000	21-Dec-2004
124	ABC	Acme Supply Co	IL	22-Dec-2004	

(b)

Alternatives

SCD Type 2

- **SCD Type 2**: new variant of the dimension row
  - New record at the dimension table;
  - **SAME Production Key, NEW Surrogate Key**
  - Variants: (a) version number / (b) valid time timestamps
  - Can have status columns to indicate which row is current (see **SCD Type 6**)
- Fact data refer to the correct SK

@ DW

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	CA

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Version
123	ABC	Acme Supply Co	CA	0
124	ABC	Acme Supply Co	IL	1

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State	Start_Date	End_Date
123	ABC	Acme Supply Co	CA	01-Jan-2000	21-Dec-2004
124	ABC	Acme Supply Co	IL	22-Dec-2004	

- NOT Easy
- NOT (so) Fast
- Augments the dimension table
- Correctly answers historical queries

Type 2 answers historical queries, …but… mind your groupings!

Dim Table: Supplier

SuppK	Code	Name	State	Vid
121	AAA	X	CA	0
122	BBB	Y	NY	0
123	ABC	Z	CA	0
124	ABC	Z	IL	1

Fact Table: Supplies

SuppK	ProdK	Date	Amt	…
123	100	160303	100	
123	200	160303	20	
124	100	160304	30	
124	200	160304	10	

If someone asks: **"how many shipments did we have from CA this month?"**
We can join fact and dim and group by State

If someone asks: **"how many shipments did we have from ABC?"**
We MUST join fact and dim and group by Code!

i.e., the SK CANNOT help with counting unique dim objects!
(and we must pay the price of joining too)

SCD Type 6

- **SCD Type 6: Type 2 +**
  - **status columns** (Flag + dates) to indicate which row is current
- Fact data refer to the correct SK
- Hard to implement
- All version info is available; most complete solution

@ Source

Supplier_Code	Supplier_Name	Supplier_State
ABC	Acme Supply Co	CA

Supplier_Code	Supplier_Name	Supplier_State
ABC	Acme Supply Co	IL

@ DW

Supplier_Key	Supplier_Code	Supplier_Name	Current_State	Historical_State	Start_Date	End_Date	Current_Flag
123	ABC	Acme Supply Co	CA	CA	01-Jan-2000	31-Dec-9999	Y

Supplier_Key	Supplier_Code	Supplier_Name	Current_State	Historical_State	Start_Date	End_Date	Current_Flag
123	ABC	Acme Supply Co	IL	CA	01-Jan-2000	21-Dec-2004	N
124	ABC	Acme Supply Co	IL	IL	22-Dec-2004	31-Dec-9999	Y

Supplier_Key	Supplier_Code	Supplier_Name	Current_State	Historical_State	Start_Date	End_Date	Current_Flag
123	ABC	Acme Supply Co	NY	CA	01-Jan-2000	21-Dec-2004	N
124	ABC	Acme Supply Co	NY	IL	22-Dec-2004	03-Feb-2008	N
125	ABC	Acme Supply Co	NY	NY	04-Feb-2008	31-Dec-9999	Y

Here: a 3rd change too: IL->NY

SCD – Type 6 (1+2+3)

Type 6

Supplier_Key	Supplier_Code	Supplier_Name	Current_State	Historical_State	Start_Date	End_Date	Current_Flag
123	ABC	Acme Supply Co	CA	CA	01-Jan-2000	31-Dec-9999	Y

Supplier_Key	Supplier_Code	Supplier_Name	Current_State	Historical_State	Start_Date	End_Date	Current_Flag
123	ABC	Acme Supply Co	IL	CA	01-Jan-2000	21-Dec-2004	N
124	ABC	Acme Supply Co	IL	IL	22-Dec-2004	31-Dec-9999	Y

Supplier_Key	Supplier_Code	Supplier_Name	Current_State	Historical_State	Start_Date	End_Date	Current_Flag
123	ABC	Acme Supply Co	NY	CA	01-Jan-2000	21-Dec-2004	N
124	ABC	Acme Supply Co	NY	IL	22-Dec-2004	03-Feb-2008	N
125	ABC	Acme Supply Co	NY	NY	04-Feb-2008	31-Dec-9999	Y

for each fact record, find the current supplier state and the state the supplier was located in at the time of the delivery

```
SELECT
  delivery.delivery_cost,
  supplier.supplier_name,
  supplier.historical_state,
  supplier.current_state
FROM delivery
INNER JOIN supplier
  ON delivery.supplier_key = supplier.supplier_key
```

ensure a single supplier record is retrieved for each transaction

```
  AND delivery.delivery_date >= supplier.start_date
  AND delivery.delivery_date <= supplier.end_date
```

OR

how a specific date can be used

```
  AND delivery.delivery_date >= '2012-01-01 00:00:00'
  AND delivery.delivery_date <= '2012-01-01 00:00:00'
```

Dimension Table Growth

- The problem at hand: if Type 2 or 6, the dim-table can grow too large -- esp., if
  - too many attributes are monitored
  - some attributes change fast (e.g., age)
- So occasionally, Types 2 & 6 are not really feasible

- Remember: some dimension table are too big on their own (e.g., Customer)
- Big dimension tables means that they do not fit in main memory and joins with them become slow

- So, we need to battle dimension table scale up!

Slowly Changing Dimensions

- Type 4: definitions vary
  - History Table: split type-2 table in two tables, subsets of the data set: the historical one and the current one (single row)
  - Kimball's Mini-dimensions (see next): if some attributes of the dimension change frequently,
    - export a new table (called "mini-dimension") just for them;
    - facts have two FK's for the dimension, one for the dim table and another for the profile table

SCD Type 4

@ Source

Supplier_Code	Supplier_Name	Supplier_State
ABC	Acme Supply Co	CA

Supplier_Code	Supplier_Name	Supplier_State
ABC	Acme Supply Co	IL

@ DW

Supplier_Key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	CA

Supplier

Supplier_key	Supplier_Code	Supplier_Name	Supplier_State
123	ABC	Acme Supply Co	IL

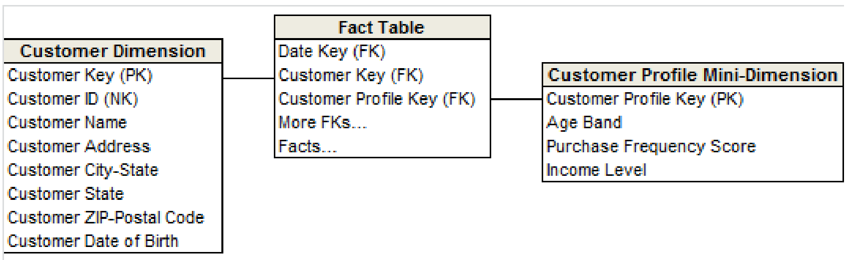
Supplier_History

Supplier_key	Supplier_Code	Supplier_Name	Supplier_State	Create_Date
123	ABC	Acme Supply Co	CA	22-Dec-2004

- **SCD Type 4**: two tables
  - A current + a historical one
  - **SAME Production Key, SAME Surrogate Key**
  - Time timestamps at history
- Fact data refer to the same SK
- One has to do complicated queries to handle historical values (yet, doable)

Rapidly Changing Dimensions

- **(Kimball's) SCD Type 4**:
  - split off frequently changing attributes of a dimension into a separate **mini-dimension**.
  - assign SK to the new mini-dim table too



- If the dim table is too large, one can use mini-dimensions as a trick to have a smaller dimension table for frequently used attributes too

Several other tricks to combat dimension-table augmentation over time:

- Group attribute values in group (e.g., instead of age, age band)
- further split the mini-dimension too, if necessary

L FOR LOAD

Load

- Goal: fast loading into DW
  - loading deltas is much faster than total load
- Issues
  - lots of data & short time window
  - freshness
  - table updates, but also indices, views, etc.
  - preserve data integrity after a failure
- Sometimes extra care is needed
  - sort / aggregation

Load

- Load techniques

  – SQL is not a good choice
    - slow: tuple-by-tuple
    - slow: random disk i/o
    - overflow of rollback segment / log file (might create zombie processes)

  – batch loading tools
    - DB load tools are much faster

Load

- Load techniques (cntd)
  - index on tables **slows** load a lot
    - drop index and rebuild after load
    - can be done per index partition
  - disable logging & locking
    - risky for load failures
  - sort tuple on a clustering key (esp. if the table is clustered too)
  - prefer sequential i/o than random i/o

Load

- Load techniques (cntd)
  - parallelize, parallelize, parallelize…
    - dimensions can be loaded concurrently
    - fact tables can be loaded concurrently
    - partitions can be loaded concurrently
  - aggregates
    - can be built and loaded at the same time as the detail data

- Design
- Optimization
- ...

DESIGN & OPTIMIZATION

Issues

- Use ETL tool or write ETL code?
  - Code: easy start, co-existence with IT infrastructure, maybe the only possibility
  - Tool: better productivity on subsequent projects, “self-documenting”
- Load frequency
  - ETL time dependent of data volumes
  - Frequency & Prioritization of flows is dictated by user req’s, data volumes, strength of source servers, …
  - Daily load is much faster than monthly
  - Applies to all steps in the ETL process
- Files versus streams/pipes
  - Streams/pipes: no disk overhead, fast throughput
  - Files: easier restart, often only possibility

Pipes: Redirect output from one process to input of another process

```
cat payments.dat | grep 'payment' | sort -r | uniq -u
```

Test Test Test

- **For each flow, esp., at initial build:**
  - per step: **ensure** that **result is as expected**
  - overall: **ensure** that **result is as expected**
- Often, this requires intermediate storage and debugging
  - Unknown data errors often hidden in the sources
  - SQL/… statements of the ETL process buggy
- Do not skip this part; DW is expected to hold the single version of the truth for its data; otherwise it serves no purpose

How to design?

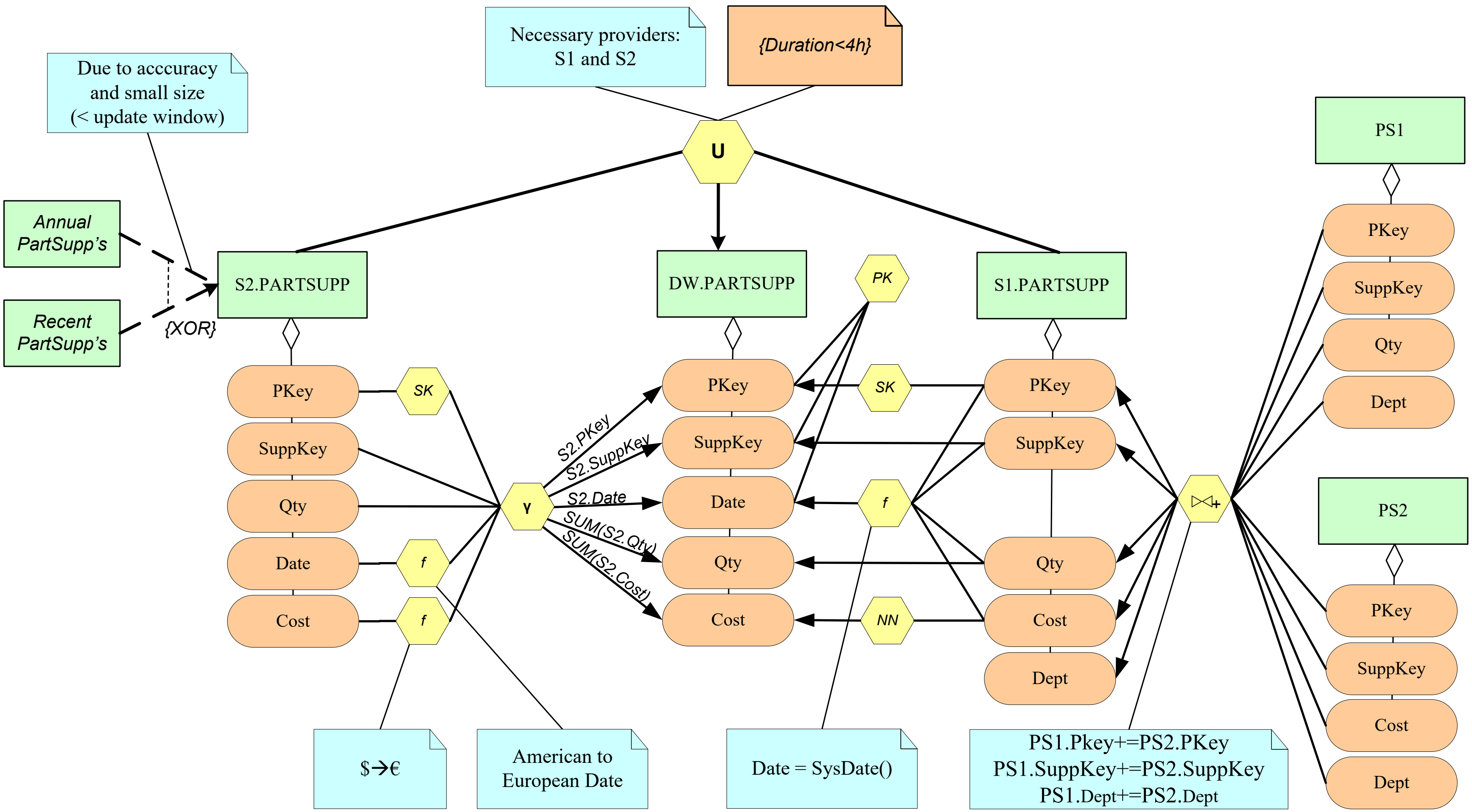
- Early stage requirements: the hardest part
  - Source data must be mapped to the DW format
  - DON'T FORGET THE DIMENSIONS!
  - They must be profiled for errors and cleaning actions have to be taken
  - Target aggregates/reports/cubes to be delivered must be identified and prioritized

- ETL design research efforts mainly focused here

Conceptual Modeling for (early stage) ETL Processes

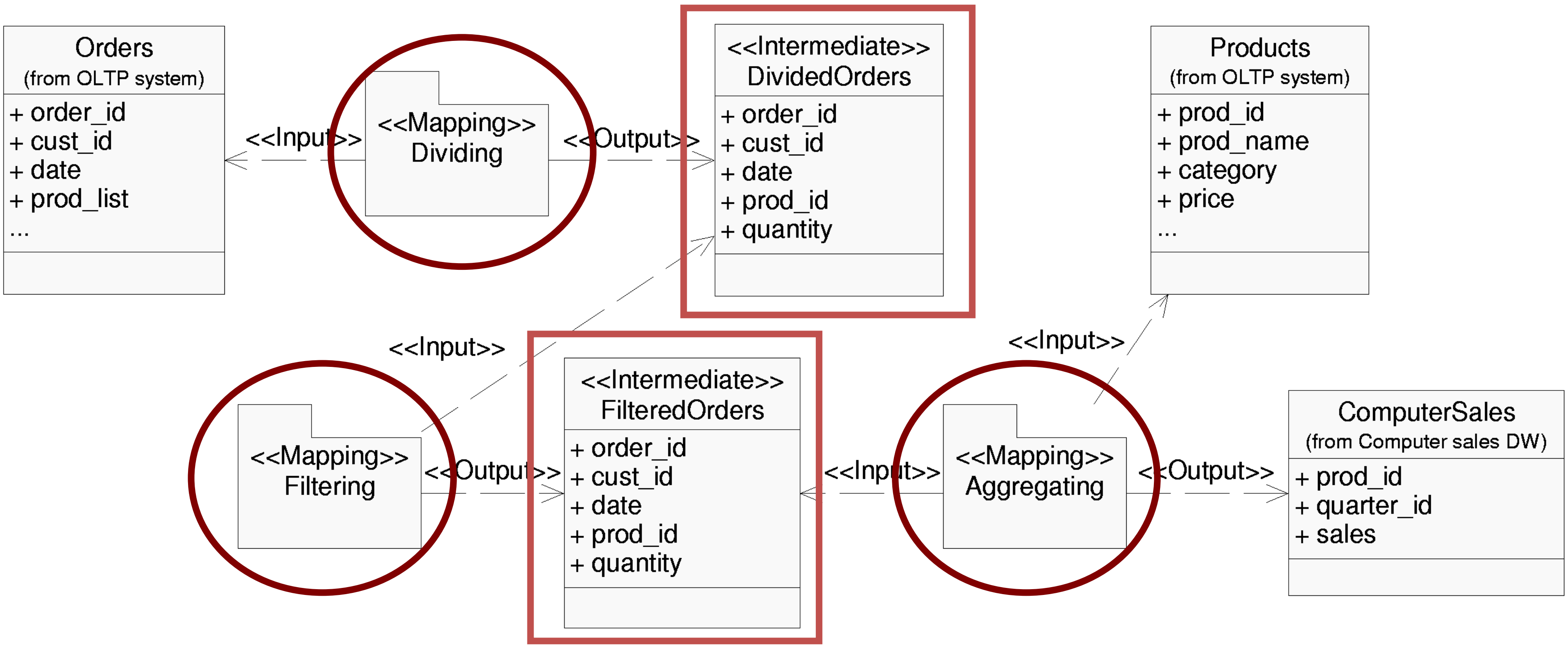
P. Vassiliadis, A. Simitsis, S. Skiadopoulos DOLAP'02

A. Simitsis & P. Vassiliadis DSS'08



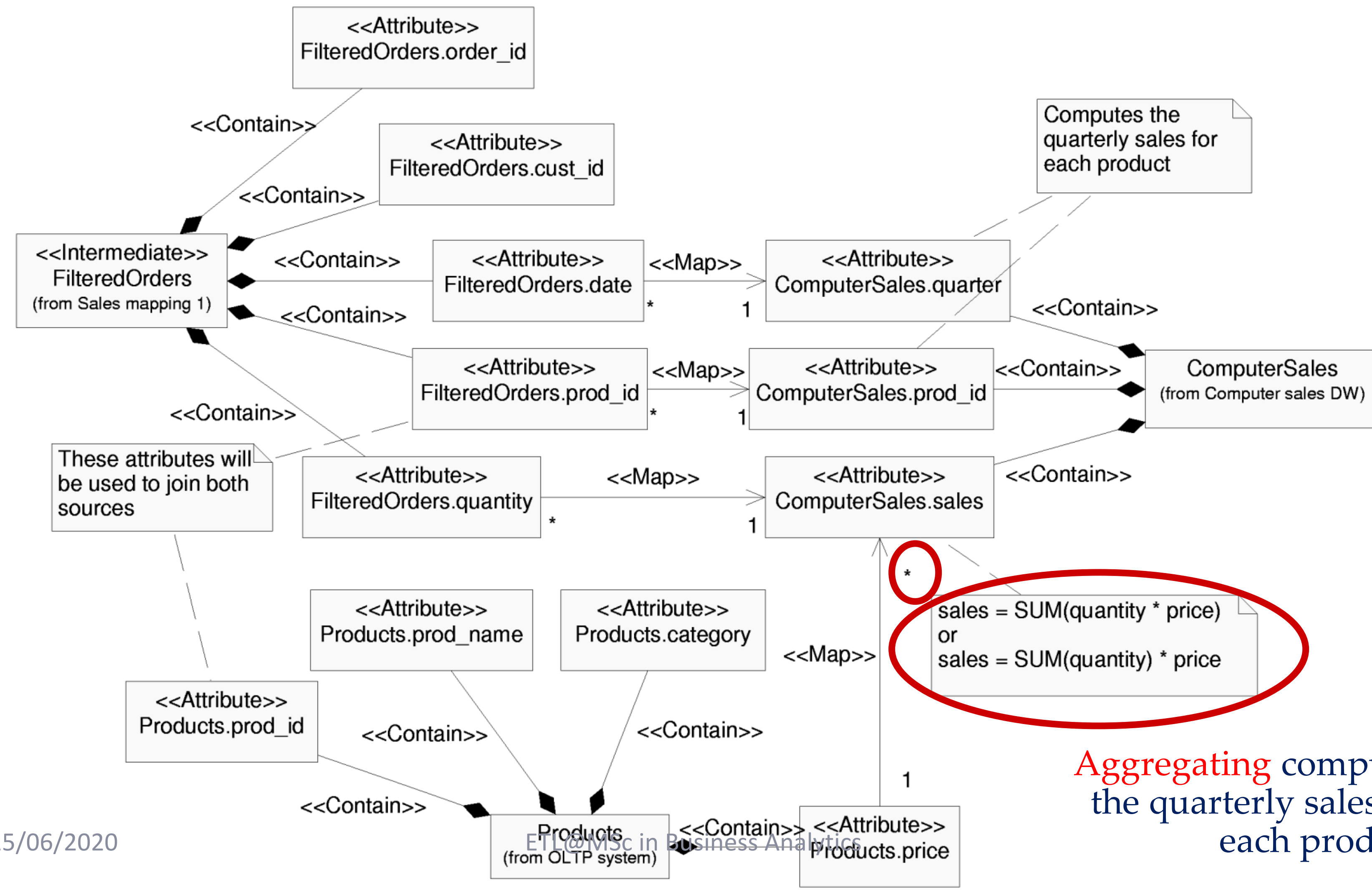
The UML Data Mapping Diagram

S. Lujan-Mora, P. Vassiliadis, J. Trujillo, ER 2004



The UML Data Mapping Diagram

S. Lujan-Mora, P. Vassiliadis, J. Trujillo, ER 2004



Aggregating computes the quarterly sales for each product.

How to design?

The tools help a lot!

- Later stage: a flow must be
  - designed & built
  - tested & debugged,
  - documented,
  - executed =>initial load of the DW,
- Moreover, incremental updates must be
  - designed & built
  - tested tested tested
  - documented,
  - deployed at production

You need to do data profiling (tools can help)

ADVANCED TRENDS

Data Lakes / Data Vaults / ETLT-ELT

- Typically, ETL is a principled process, with a lot of a priori design both of the schemata and the data flows
- Data Lakes (aka Data vaults): keep any data you find in a DSA, typically in an unstructured or loosely structured format, and if you need it later, you do your best to exploit it
- ELT & ETLT: load first & then we see what we do with the acquired data
- Many issues:
  - Provenance & metadata
  - Data quality
  - Timeliness of data & purging of expired data sets
  - Integration
  - Scale of the number of sources and acquired data sets

ETL on the cloud

- Both for traditional & novel architectures
- HDFS for the files of the DSA
- any HDFS-based data management system for loosely structuring DSA data
  - Graph DBMS for graph data
  - Text DBMS for textual /json data
  - Column-family DBMS for loosely structured tabular data
  - …
- Spark, Hadoop based ETL tool / set of scripts to gracefully scale-out

(Near) Real Time ETL

- What timeliness/latency is acceptable?
  - Yesterday's data?
  - 1 hour-old data?
  - 15'?
  - 15"?
  - …
- Latency of what?
  - End to end? (from data production to the OLAP server)
  - Data Load?
  - …
- How to tune and schedule the simultaneous loading and querying?
  - Remember: one of the original motivations for separating OLTP & BI was exactly the impossibility of handling both tasks at the same server

Super scale ETL

- Internet of Things with thousands/millions of sources
- Sometimes sensor sources have unique requirements/capabilities
- Issues:
  - Data transfer
  - Energy consumption
  - Handling a huge number of sources (occasionally antagonizing for the same time slot)
  - No time for transformations++, if completeness is a goal
  - Error handling & data correction /interpolation for collected/missing data

Super scale ETL

- Internet of Things with thousands/millions of sources
- Sometimes sensor sources have unique requirements/capabilities
- Issues:
  - Data transfer
  - Energy consumption
  - Handling a huge number of sources (occasionally antagonizing for the same time slot)
  - No time for transformations++, if completeness is a goal
  - Error handling & data correction /interpolation for collected/missing data

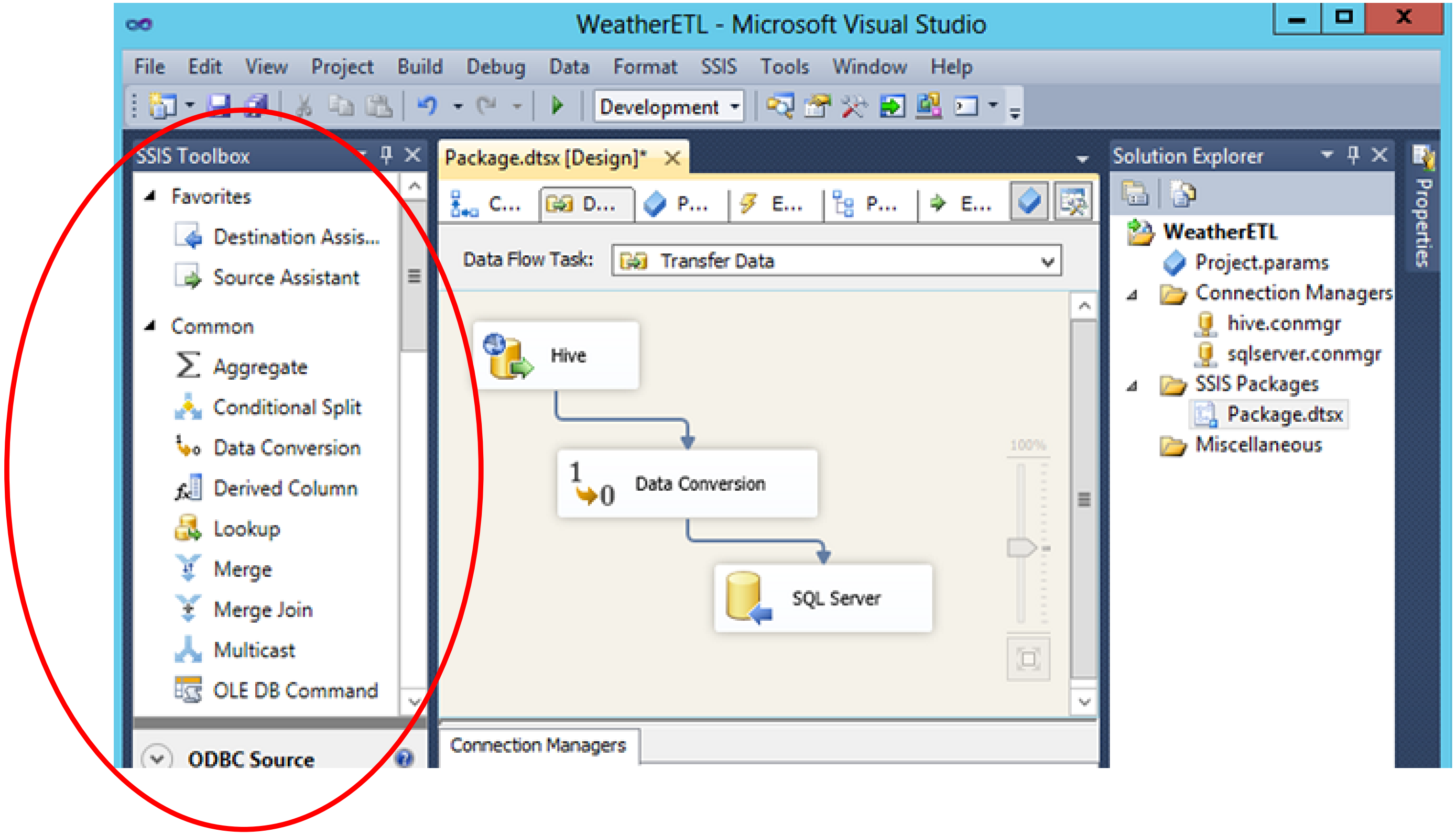
Big Data Technologies

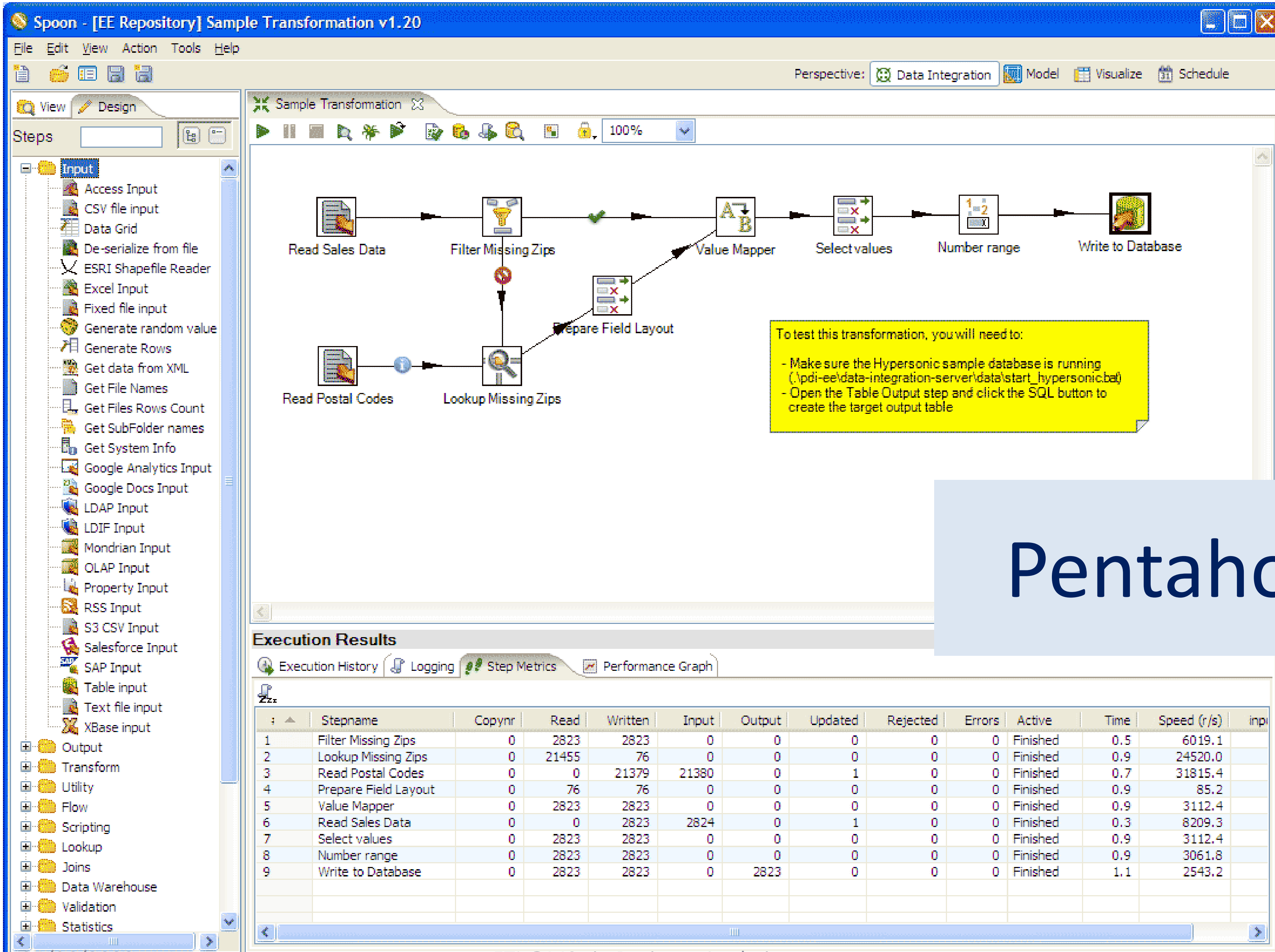
ETL COMMERCIAL TOOLS

Actian DataConnect	StreamSets Data Collector
Adeptia Integration Suite	Syncsort DMX
Alteryx	Talend
ApatarForge	Etlworks
Astera Centerprise	Singer
Attunity Compose	Alooma
Bryte Systems BryteFlow	Blendo
Bubbles	Built.io Flow
CloverETL	DataVirtuality
Elixir Repertoire Data ETL	Dell Boomi
FlyData	Eight Wire Conductor
IBI iWay Data Migrator	Etleap
IBM InfoSphere DataStage	Fivetran
Microsoft (SQL Server Integration)	Improvado
OpenText Integration Center	Informatica
Oracle Data Integrator	Matillion
Pentaho Data Integration (Kettle)	OpenBridge
Pervasive Data Integrator	Paxata
Petl	Rivery
pygrametl	Segment
Relational Junction ETL Manager	SnapLogic Elastic Integration Platform
Sagent Data Flow	Stitch
SAP BusinessObjects Data Services	Textur
SAS Data Management	Treasure Data
Scriptella	Xplenty

Actian DataConnect	StreamSets Data Collector
Adeptia Integration Suite	Syncsort DMX
Alteryx	**Talend**
ApatarForge	Etlworks
Astera Centerprise	Singer
Attunity Compose	Alooma
Bryte Systems BryteFlow	Blendo
Bubbles	Built.io Flow
CloverETL	DataVirtuality
Elixir Repertoire Data ETL	Dell Boomi
FlyData	Eight Wire Conductor
IBI iWay Data Migrator	Etleap
IBM InfoSphere DataStage	Fivetran
Microsoft (SQL Server Integration)	Improvado
OpenText Integration Center	**Informatica**
Oracle Data Integrator	Matillion
Pentaho Data Integration (Kettle)	OpenBridge
Pervasive Data Integrator	Paxata
Petl	Rivery
pygrametl	Segment
Relational Junction ETL Manager	SnapLogic Elastic Integration Platform
Sagent Data Flow	Stitch
SAP BusinessObjects Data Services	Textur
SAS Data Management	Treasure Data
Scriptella	Xplenty

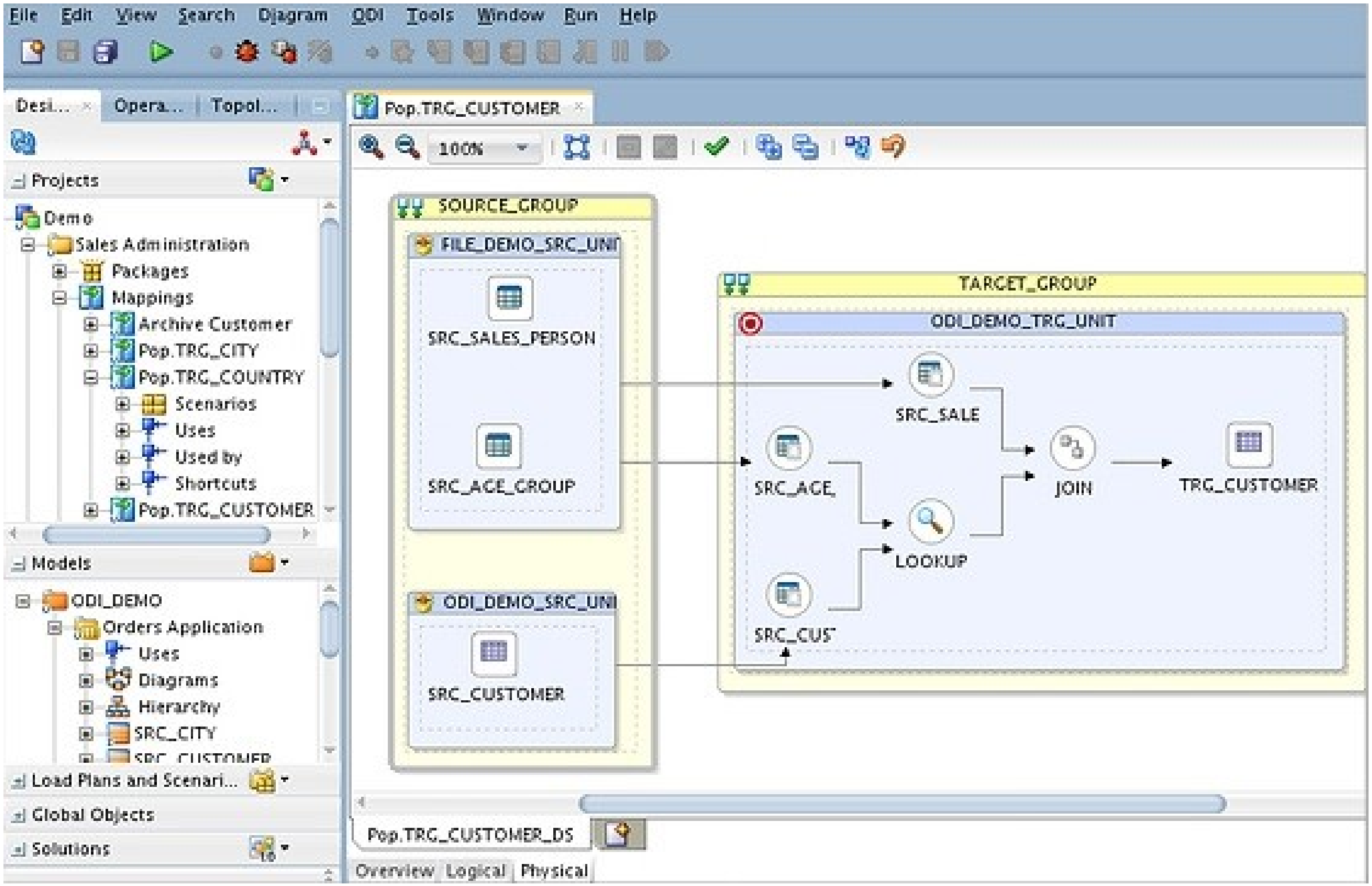
SQL Server Integration Services




Spoon - [EE Repository] Sample Transformation v1.20
File Edit View Action Tools Help
Perspective: Data Integration Model Visualize Schedule
View Design
Steps
Input
Access Input
CSV file input
Data Grid
De-serialize from file
ESRI Shapefile Reader
Excel Input
Fixed file input
Generate random value
Generate Rows
Get data from XML
Get File Names
Get Files Rows Count
Get SubFolder names
Get System Info
Google Analytics Input
Google Docs Input
LDAP Input
LDIF Input
Mondrian Input
OLAP Input
Property Input
RSS Input
S3 CSV Input
Salesforce Input
SAP Input
Table input
Text file input
XBase input
Output
Transform
Utility
Flow
Scripting
Lookup
Joins
Data Warehouse
Validation
Statistics
Sample Transformation
100%
Read Sales Data
Filter Missing Zips
Value Mapper
Select values
Number range
Write to Database
Prepare Field Layout
Read Postal Codes
Lookup Missing Zips
To test this transformation, you will need to:
- Make sure the Hypersonic sample database is running (.\pdi-ee\data-integration-server\data\start_hypersonic.bat)
- Open the Table Output step and click the SQL button to create the target output table
Execution Results
Execution History
Logging
Step Metrics
Performance Graph
Stepname
Copynr
Read
Written
Input
Output
Updated
Rejected
Errors
Active
Time
Speed (r/s)
1 Filter Missing Zips 0 2823 2823 0 0 0 0 0 Finished 0.5 6019.1
2 Lookup Missing Zips 0 21455 76 0 0 0 0 0 Finished 0.9 24520.0
3 Read Postal Codes 0 0 21379 21380 0 1 0 0 Finished 0.7 31815.4
4 Prepare Field Layout 0 76 76 0 0 0 0 0 Finished 0.9 85.2
5 Value Mapper 0 2823 2823 0 0 0 0 0 Finished 0.9 3112.4
6 Read Sales Data 0 0 2823 2824 0 1 0 0 Finished 0.3 8209.3
7 Select values 0 2823 2823 0 0 0 0 0 Finished 0.9 3112.4
8 Number range 0 2823 2823 0 0 0 0 0 Finished 0.9 3061.8
9 Write to Database 0 2823 2823 0 2823 0 0 0 Finished 1.1 2543.2

Pentaho

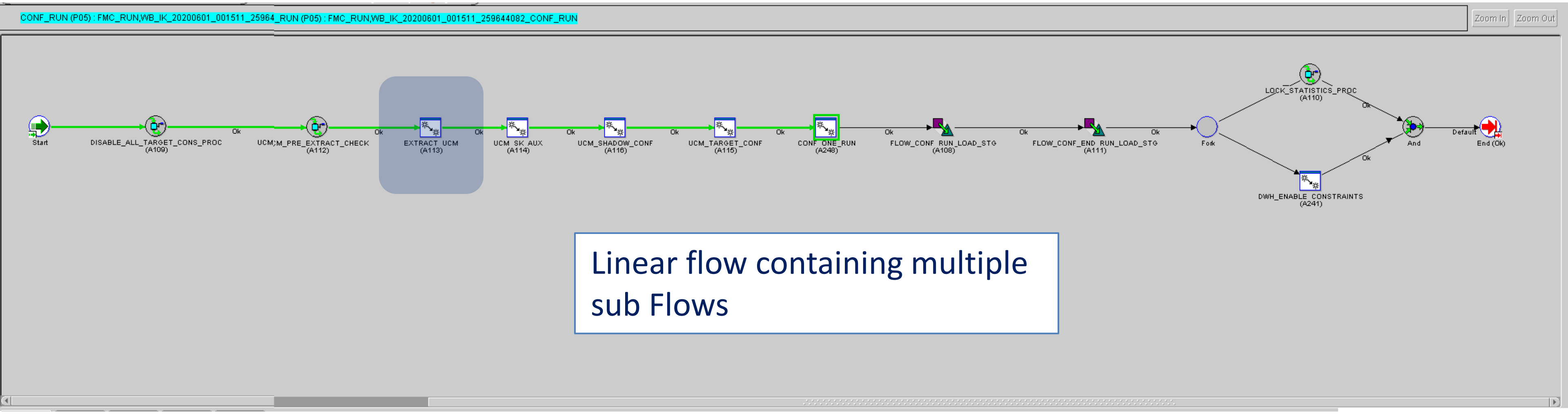
Oracle Data Integrator ODI



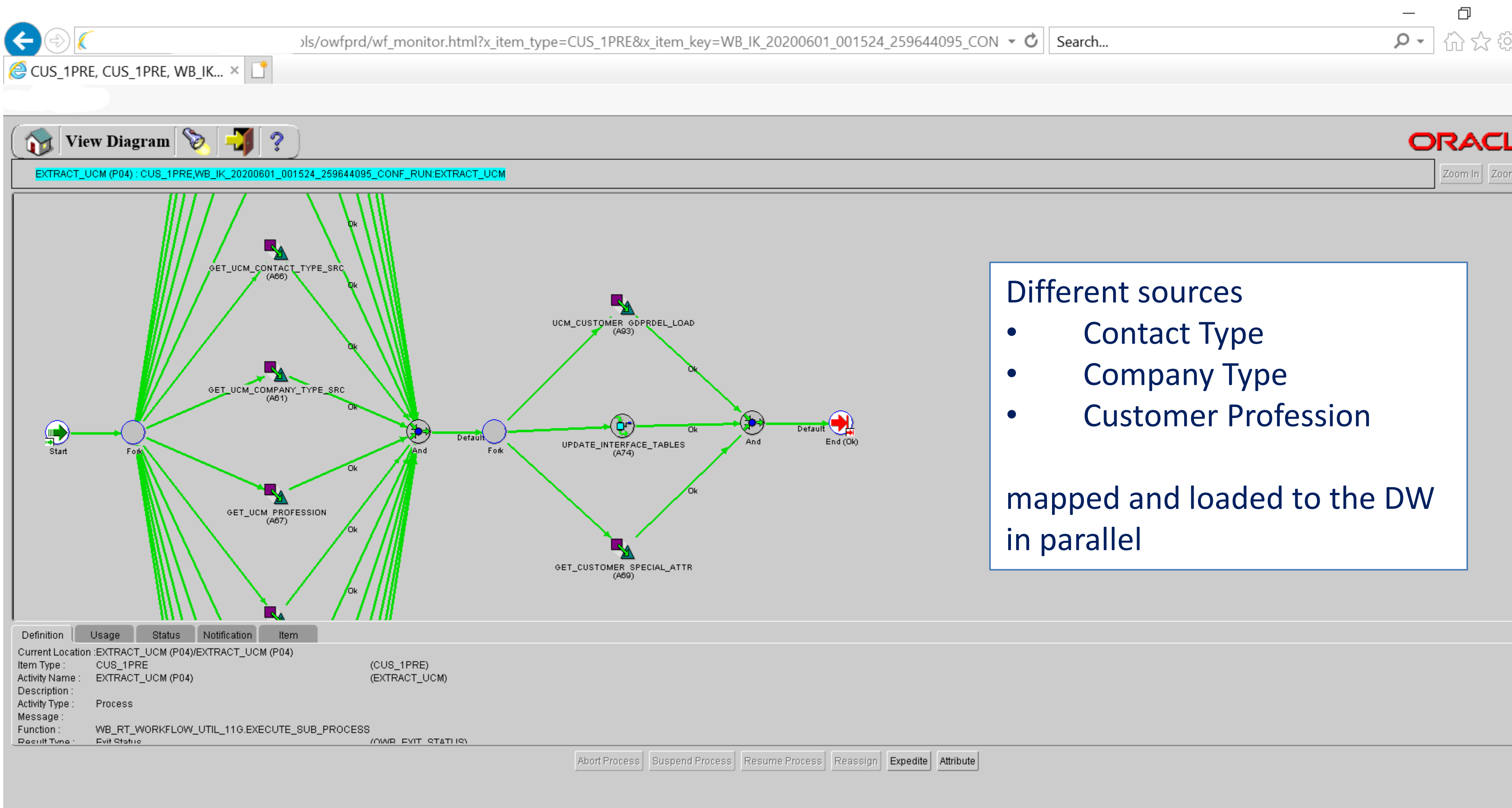
… how ETL works in production?

ETL IN ACTION

ETL for Loading Customers in DW



Sub Flow for EXTRACT_UCM operation

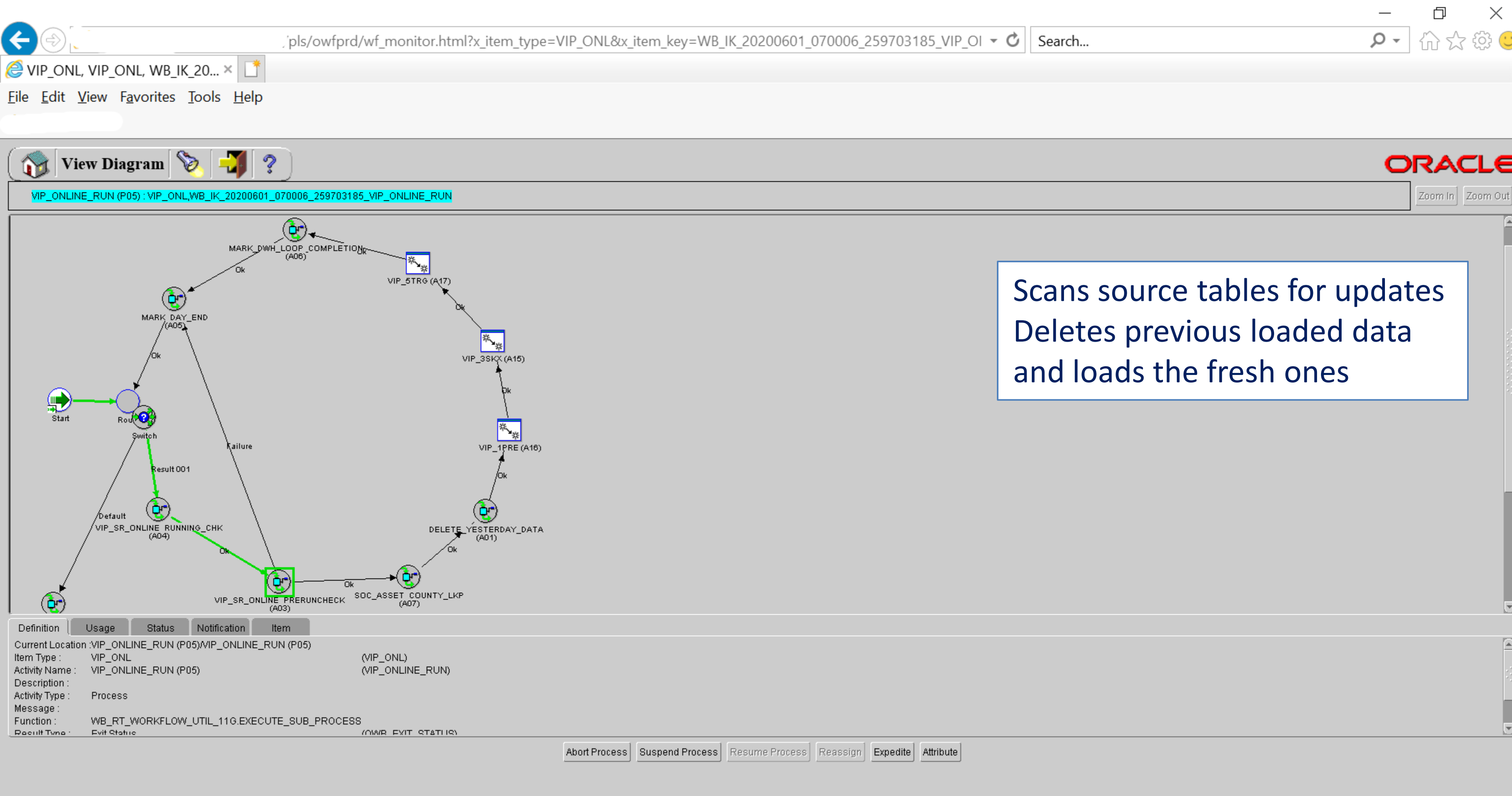


Different sources

- Contact Type
- Company Type
- Customer Profession

mapped and loaded to the DW in parallel

Cyclic flow that runs every 20min for near real-time reporting



Scans source tables for updates
Deletes previous loaded data
and loads the fresh ones

THANK YOU!