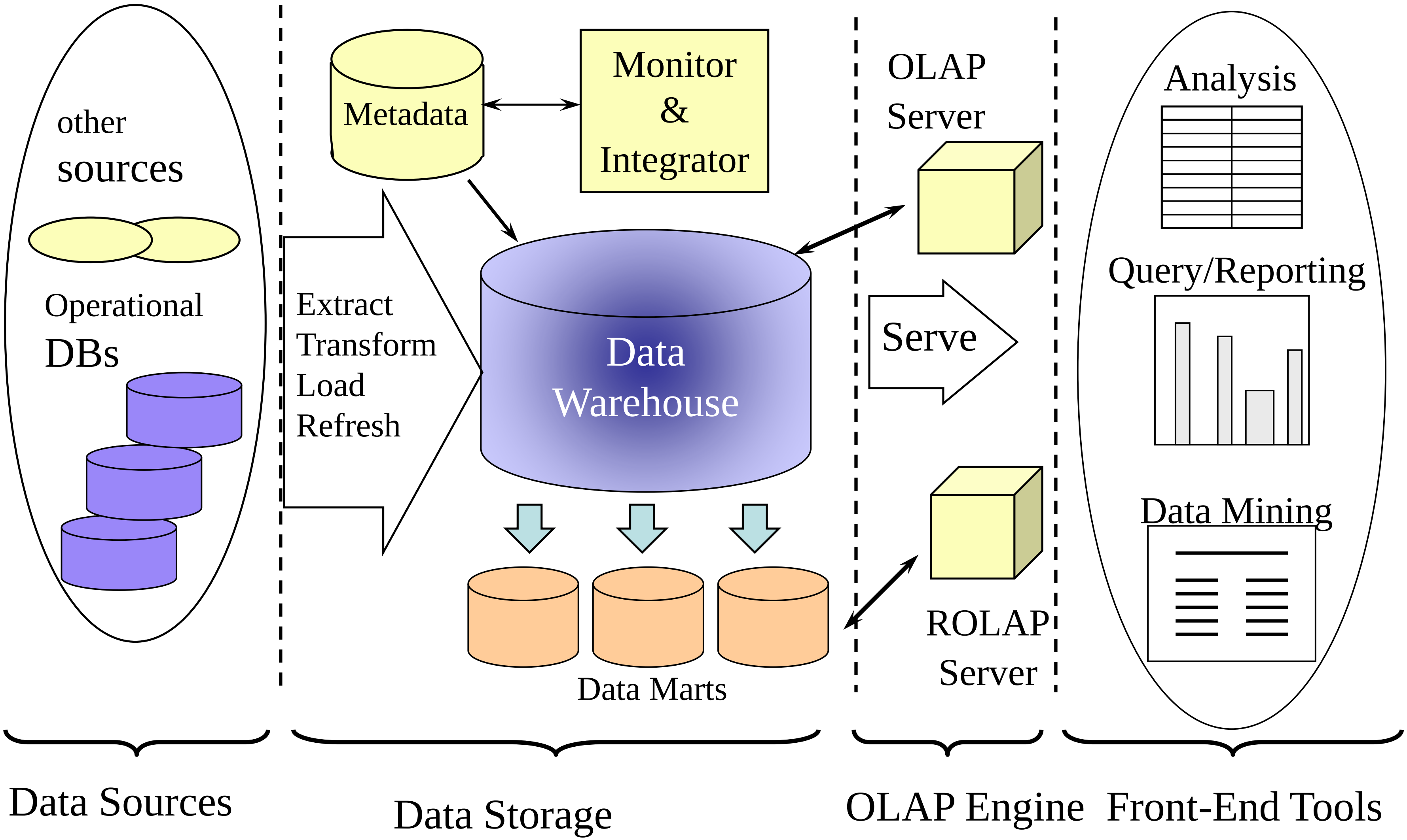
Data Warehousing
ETL

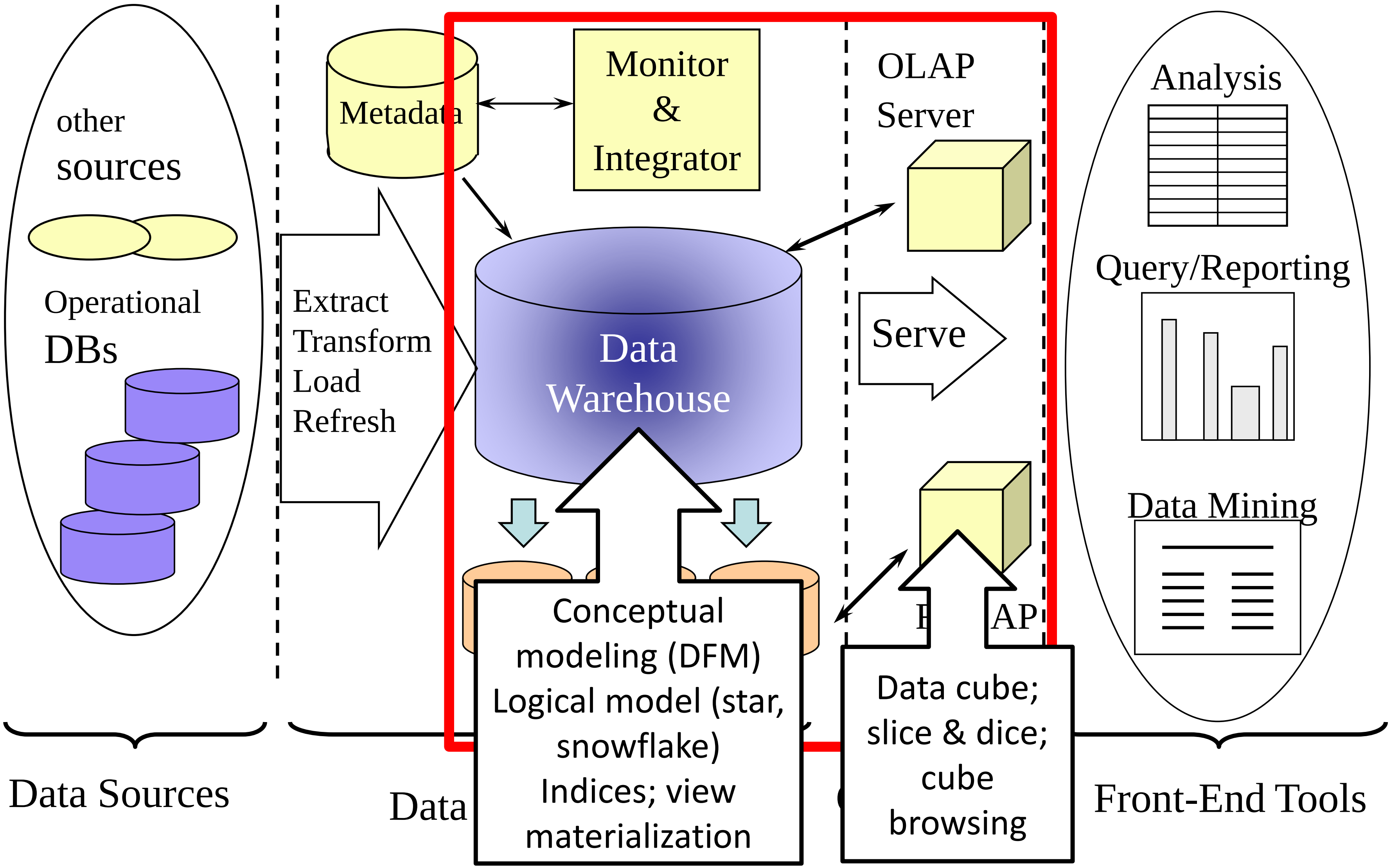
Esteban Zimányi
ezimanyi@ulb.ac.be
Slides by Toon Calders

ULB
ECOLE
POLYTECHNIQUE
DE BRUXELLES

Overview Picture

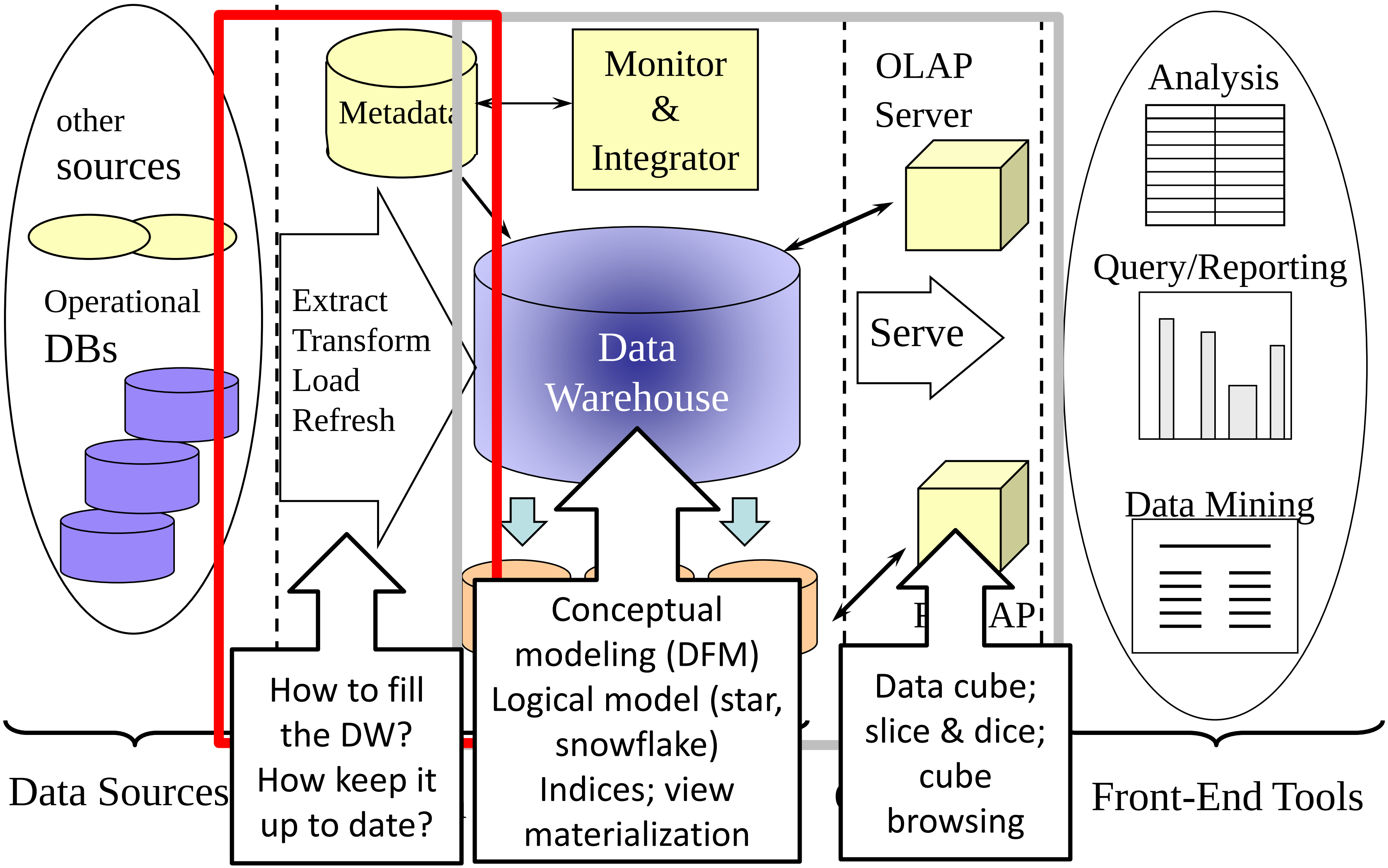


Until now



other
sources
Operational
DBs
Metadata
Monitor
&
Integrator
Extract
Transform
Load
Refresh
Data
Warehouse
OLAP
Server
Serve
AP
Analysis
Query/Reporting
Data Mining
Conceptual
modeling (DFM)
Logical model (star,
snowflake)
Indices; view
materialization
Data cube;
slice & dice;
cube
browsing
Data Sources
Data
Front-End Tools

This lecture



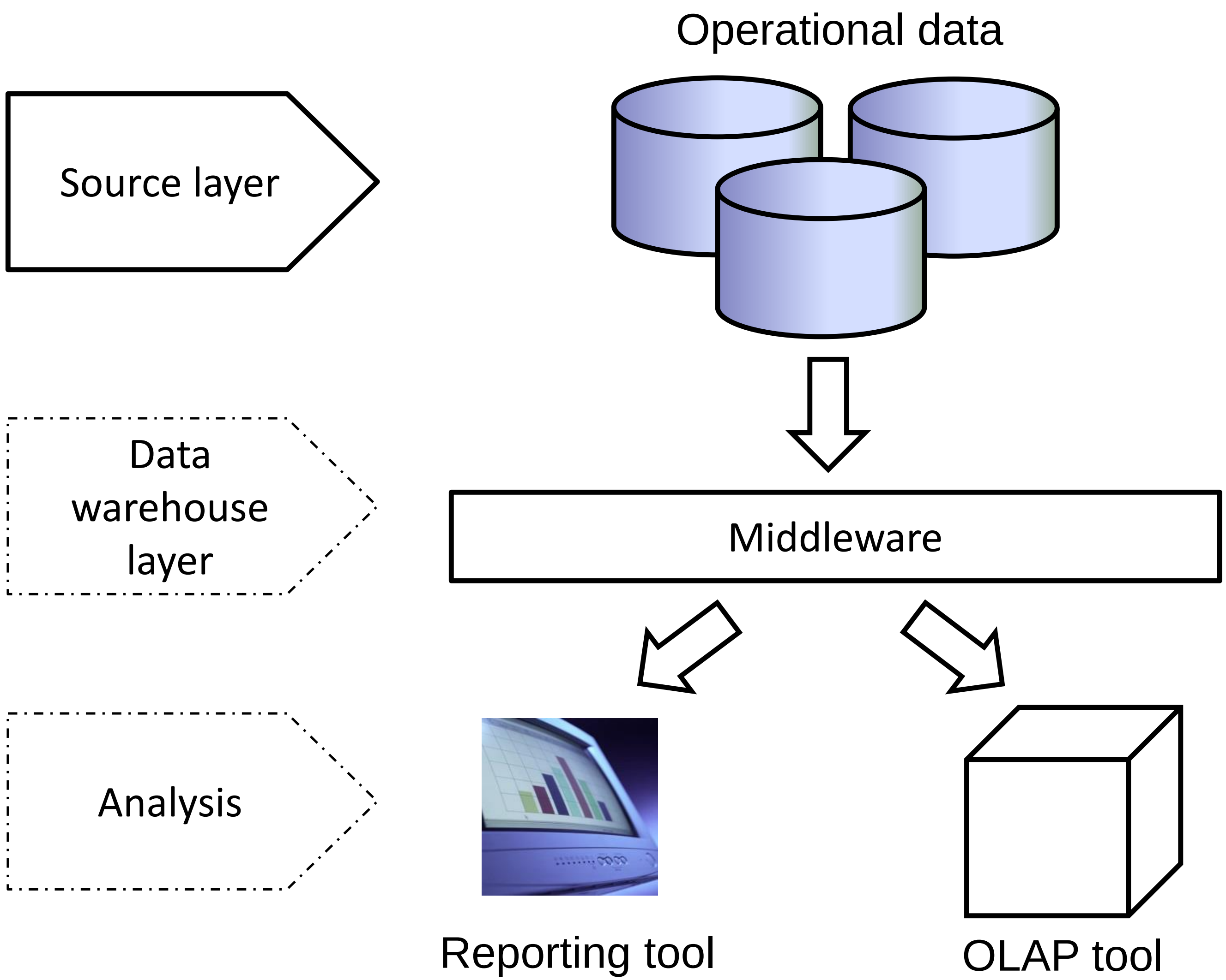
Summary

- Data Warehouse architectures
  - Single-, two-, and three-layer architectures

- ETL process
  - Extract
  - Transform/Cleanse
  - Load

Sec. 1.3, 1.4; Ch. 10 of Golfarelli and Rizzi book

Single Layer Architecture



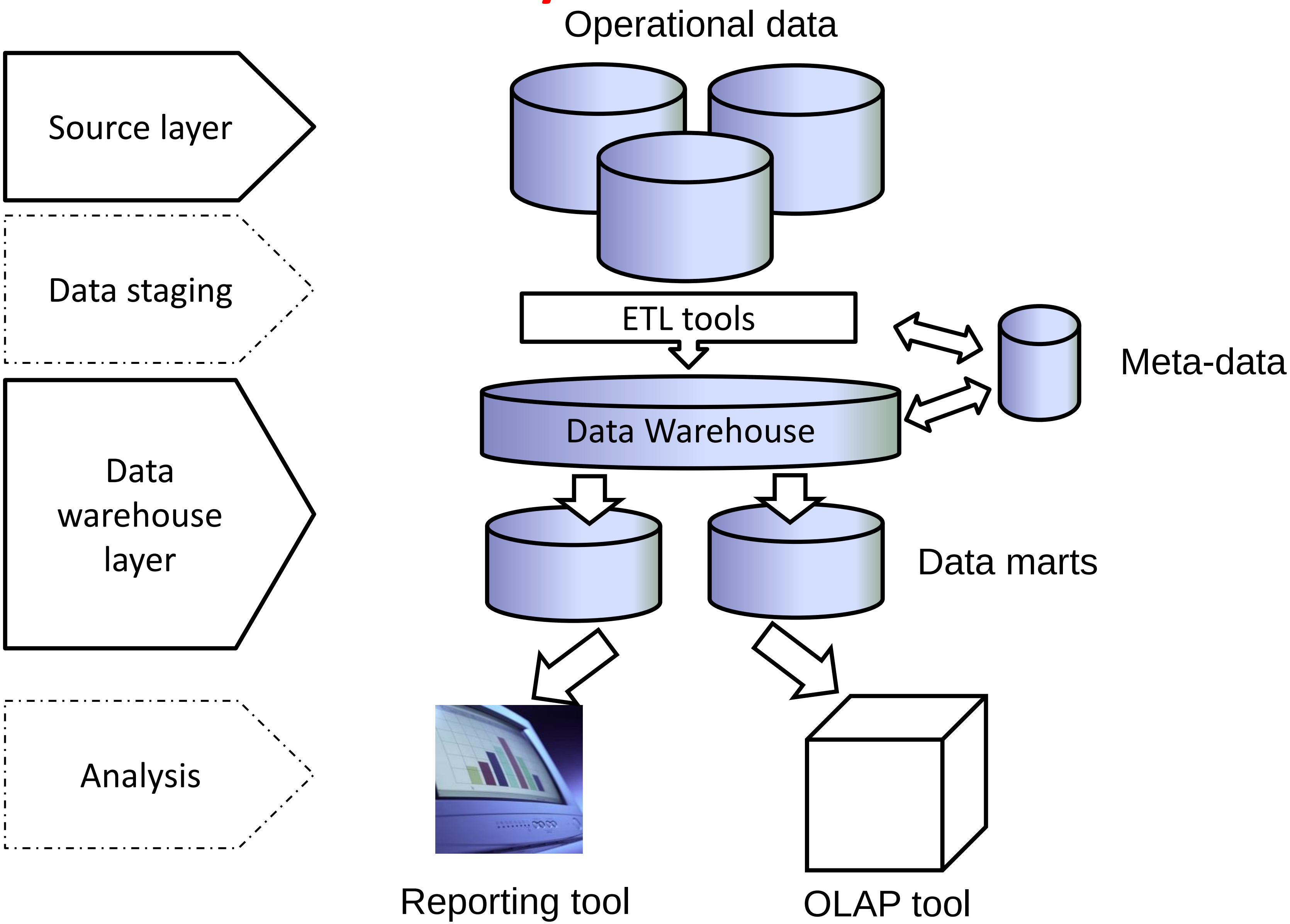
Single Layer Architecture

- Not frequently used
- Reconciled data is not materialized
  - Middleware makes heterogeneity transparent to the user

+ No data duplication

- No separation of analytical and transactional processing

Two-Layer Architecture



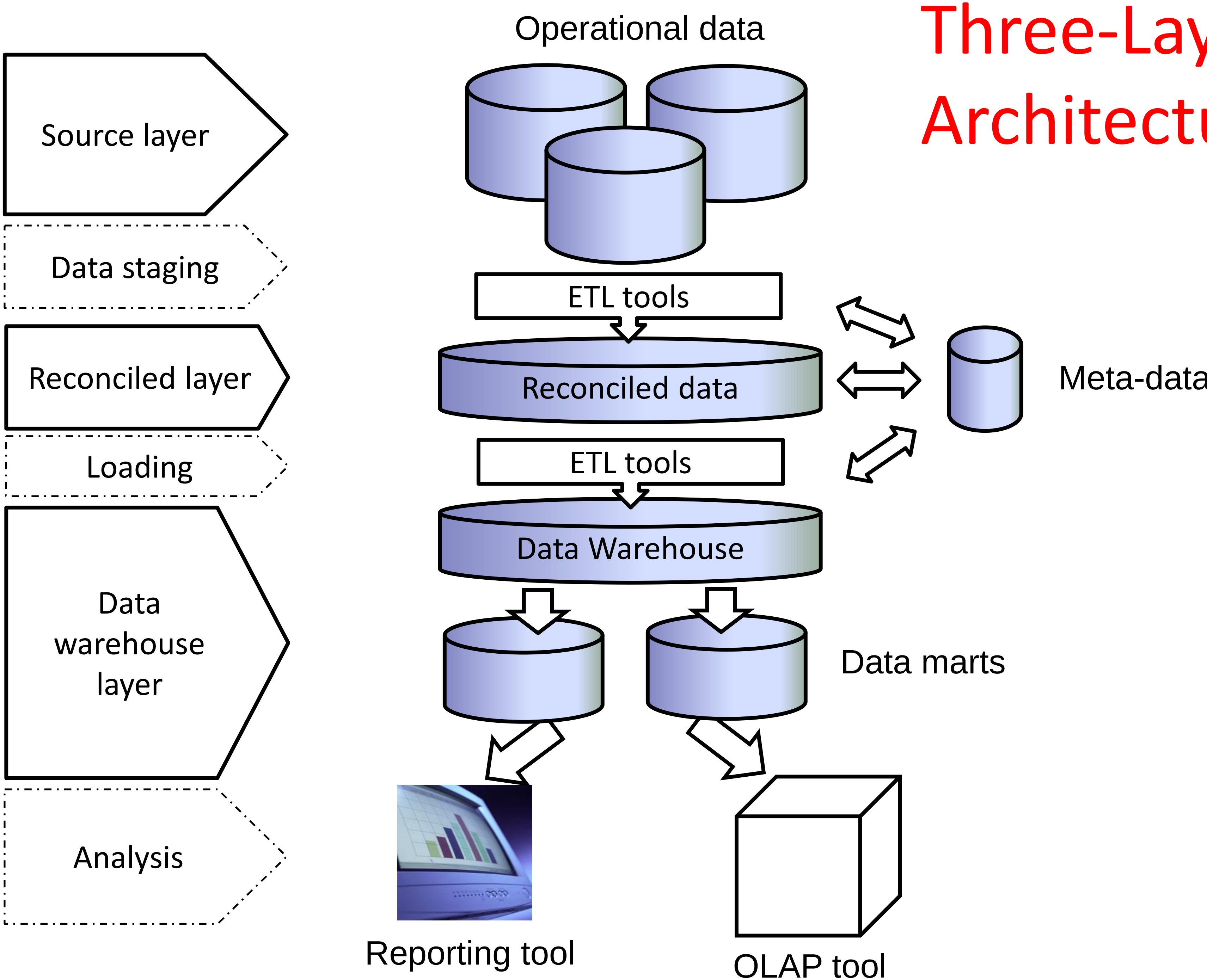
Two-Layered Architecture

- Introduction of data warehouse layer
  - Data is materialized
- Historical data management
- ETL runs regularly to keep data warehouse up-to-date

- Data duplication

+ Separation of analytics and operations

Three-Layer Architecture



Three-Layer Architecture

- Data from different sources needs to be reconciled
  - Different schemas need to be integrated
  - Source data needs to be cleansed
- In 3-tier architecture the reconciled data is materialized as well
  - Useful in its own respect
  - Makes ETL more transparent
  - Neither historical, nor dimensional

Data Staging

- Often the data staging area is materialized as well
  - Store helper tables
  - Take burden away from source systems

Summary

- Data Warehouse architectures
  - Single-, two-, and three-layer architectures

- ETL process
  - Extract
  - Transform/Cleanse
  - Load

Extract

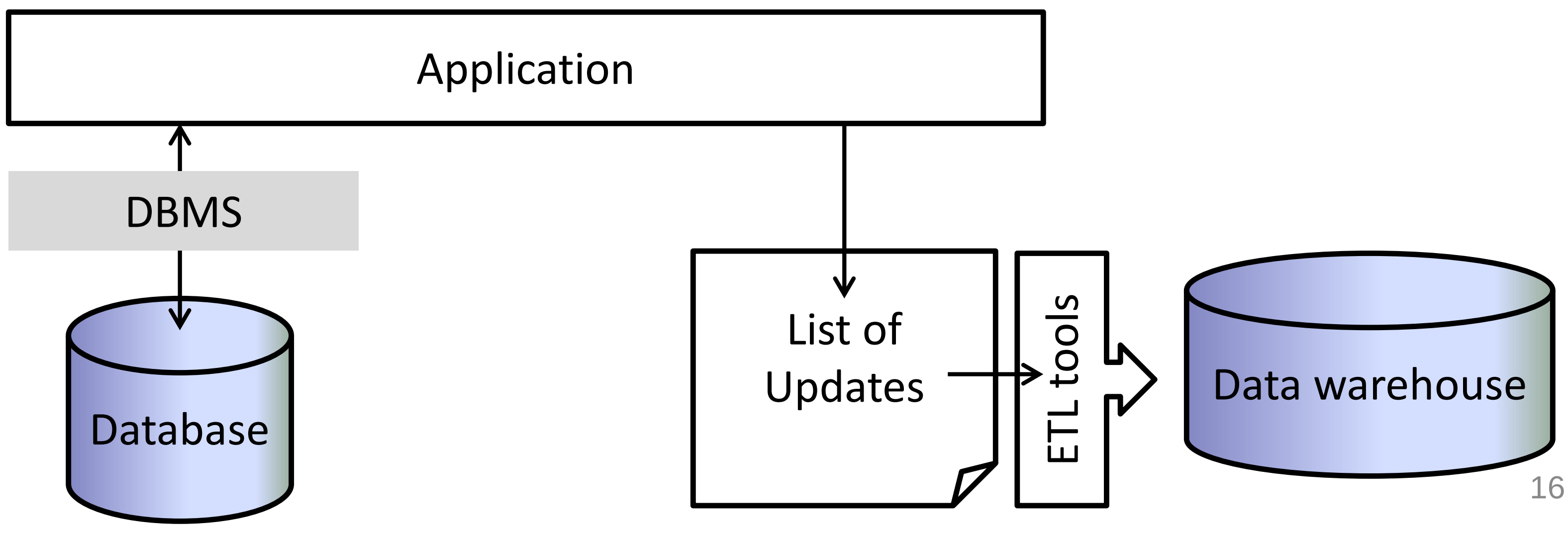
- Relevant data obtained from data sources
  - First load: extract all data
  - Subsequent loads: extract changes
- In 3-Tier infrastructure:
  - From source data to reconciled database
    - Schema integration, transformation, cleansing
  - From reconciled database to data warehouse
    - De-normalization, introduction of surrogate keys
    - Calculation of derived data

Extract

- Static vs incremental extraction
- Incremental & immediate
  - Application-assisted
  - Trigger-based extraction
  - Log-based
- Incremental & delayed
  - Timestamp-based
  - File Comparison

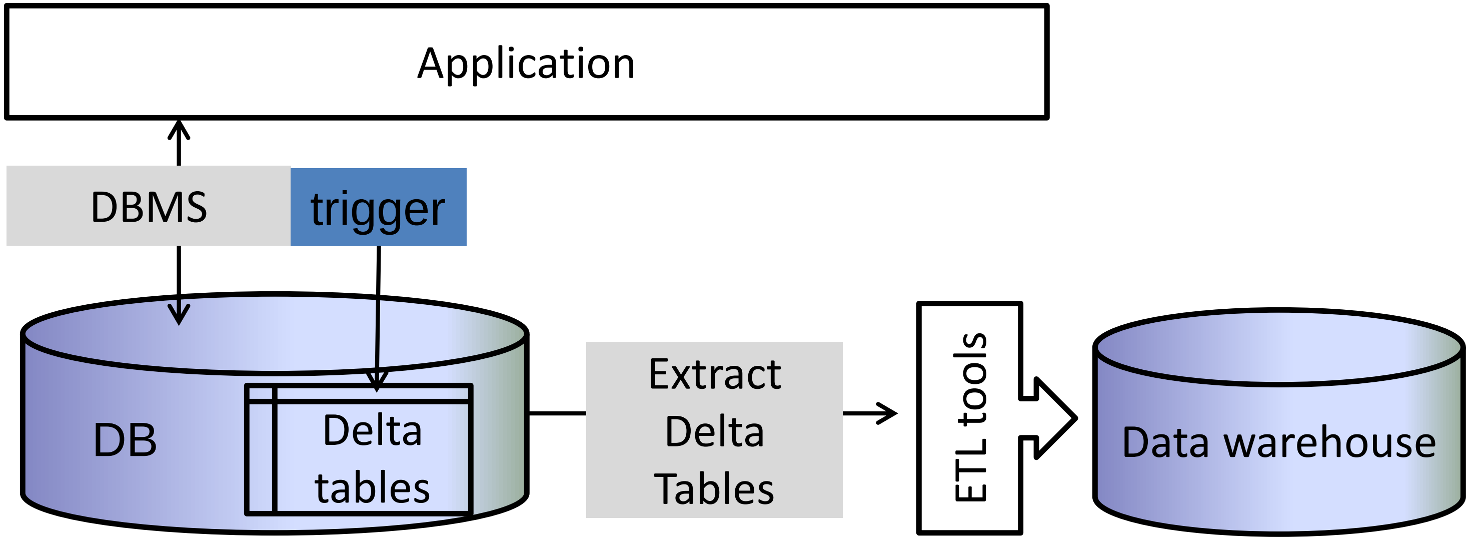
Application-Assisted

- Update of data warehouse deeply integrated into the application software
  - Immediate
  - Requires adaptation of existing applications
  - Hard to maintain



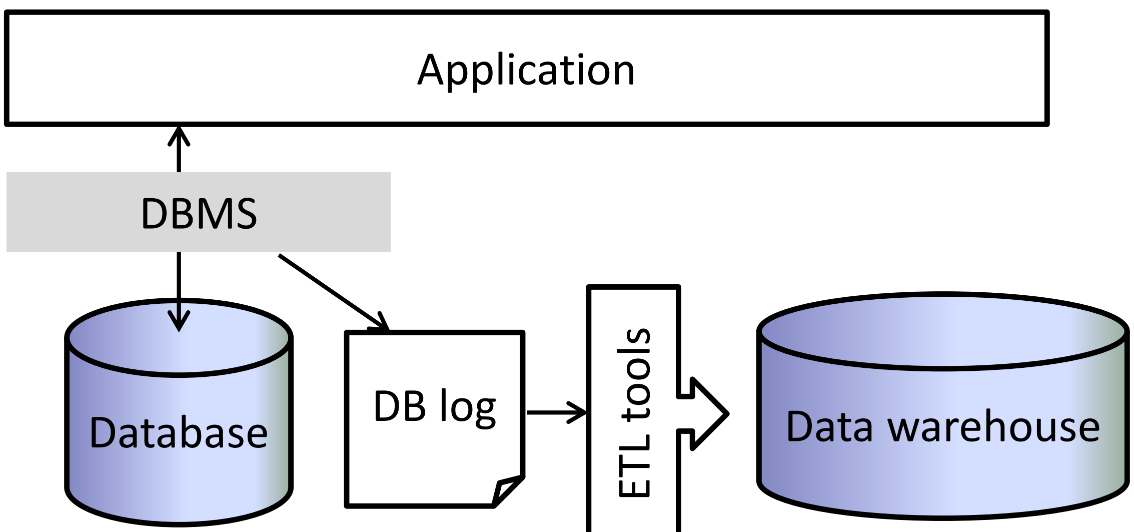
Trigger-Based

- Closely related to application-based
  - Triggers to store updates
  - Huge performance hit for database
  - More transparent for application layer
  - Only possible for data maintained in database



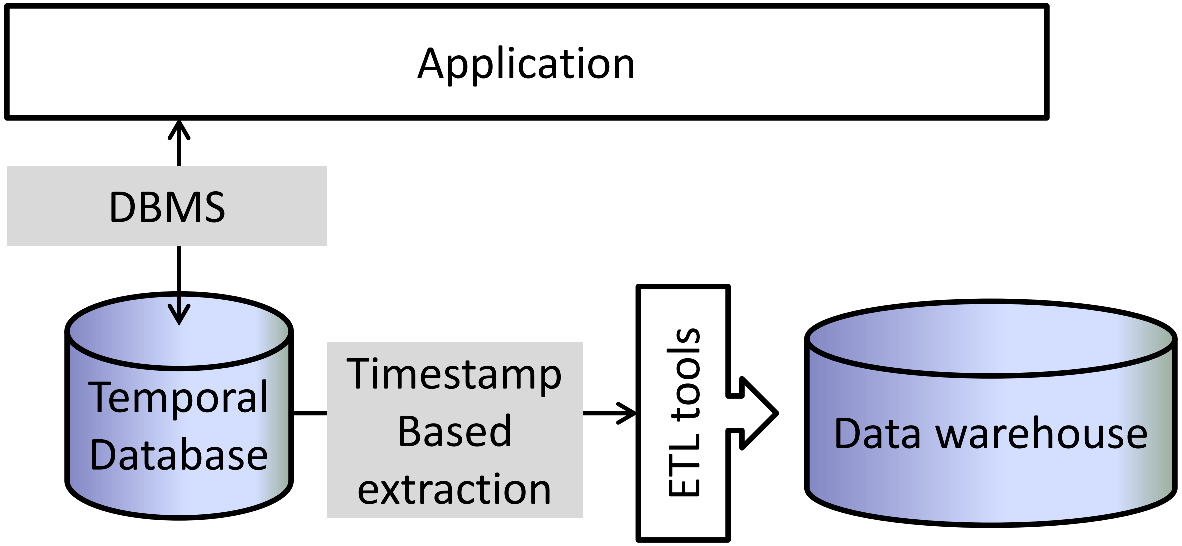
Log-Based

- Many database systems keep change logs
  - To ensure durability; in-between checkpoints all transactions to the database are logged
  - Use these change logs for updating data warehouse
  - Transparent to the user



Timestamp-Based

- Operational database can be changed to keep whole history
  - Different levels of timestamps
    - Timestamp per tuple ←→ Temporal database



Timestamp-Based

- Only last update date may lose information

Capture

Customer	Gender	Age	Segment	Date
John	M	36	1	D1
Karl	M	52	2	D1
Betty	F	18	1	D1

D1

Timestamp-Based

- Only last update date may lose information

Capture

Customer	Gender	Age	Segment	Date
John	M	36	1	D1
Karl	M	52	2	D1
Betty	F	18	1	D1

D1

Customer	Gender	Age	Segment	Date
John	M	36	2	D2
Betty	F	18	1	D1

D2

Timestamp-Based

- Only last update date may lose information

Capture

Customer	Gender	Age	Segment	Date
John	M	36	1	D1
Karl	M	52	2	D1
Betty	F	18	1	D1

D1

Customer	Gender	Age	Segment	Date
John	M	36	2	D2
Betty	F	18	1	D1

D2

Customer	Gender	Age	Segment	Date
John	M	36	3	D3
Betty	F	18	1	D1
Martin	M	64	2	D3

D3

Timestamp-Based

- Only last update date may lose information

Capture

Customer	Gender	Age	Segment	Date
John	M	36	1	D1
Karl	M	52	2	D1
Betty	F	18	1	D1

D1

Karl? John's segment ?

Customer	Gender	Age	Segment	Date
John	M	36	2	D2
Betty	F	18	1	D1

D2

Capture

Customer	Gender	Age	Segment	Date
John	M	36	3	D3
Betty	F	18	1	D1
Martin	M	64	2	D3

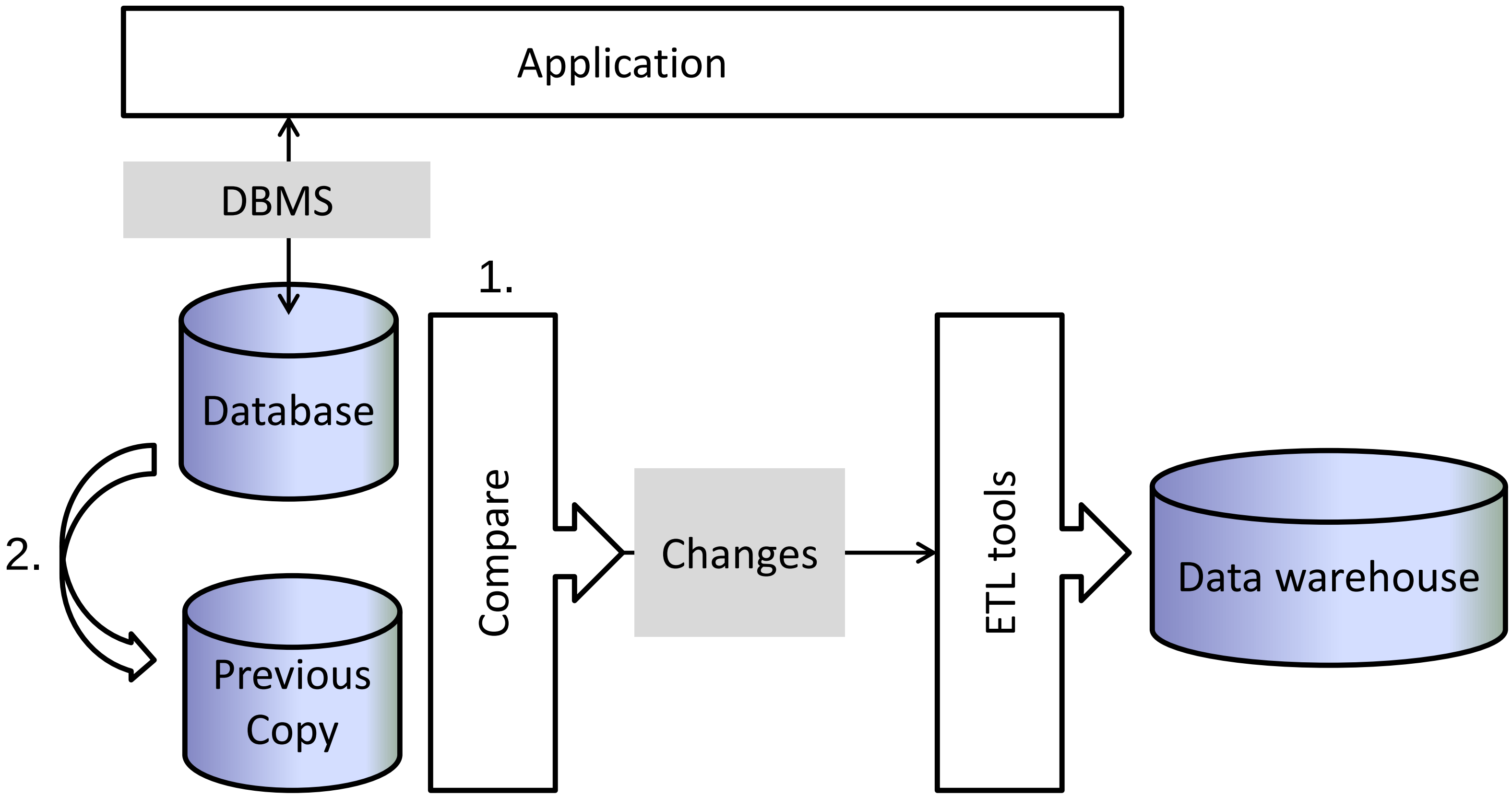
D3

Timestamp-Based

- Based on the application field and update frequency, one timestamp per tuple may be acceptable
  - Type-2 change for attributes that change several times between updates?!
- If more fine-grained behavior is required we can go to full temporal database
  - Register valid time (start-end)

File Comparison

- Easy; works even for flat files
- Does not capture intermediate changes



Extraction Methods - Overview

Method	Maintain full history?	Impact on performance operational level	Complexity extraction procedures	DBMS dependent	Requires changes to application layer	Maintenance
Static	No	No	Low	No	None	Easy
Application assisted	Yes	Medium	High	No	Many	Difficult
Trigger based	Yes	Medium	Medium	Yes	None	Medium
Log based	Yes	No	Low	Limited	None	Easy
Timestamp based	(yes)	Some	Low	Limited	Some	Medium
File comparison	Not all	No	Medium	No	None	Easy

Summary

- Data Warehouse architectures
  - Single, two-, and three-layer architectures
- ETL process
  - Extract
  - Transform/Cleanse
  - Load

Transform/Cleanse

- Conversion
  - Different date types
- Enrichment
  - Combine information of different attributes to create a new one
- Joining data without key;
  - Entity resolution
- Correcting errors
  - De-duplication

De-duplication & Entity Resolution

- De-duplication
  - Recognize duplicate entries
- Entity resolution
  - Recognize if two entities from different sources are actually referring to the same object
    - Not always obvious if there is no key shared between different data sources
  - In most extreme case need for methods to directly compare strings

Example: Cleansing customer data

John White
Downing St. 10
TW1A 2AA London (UK)

normalizing

Fname: John
Sname: White
Adress: Downing St. 10
ZIP: TW1A 2AA
City: London
Country: UK

standardizing

Fname: John
Sname: White
Adress: 10**,** Downing St.
ZIP: TW1A 2AA
City: London
Country: United Kingdom

correcting

Fname: John
Sname: White
Adress: 10**,** Downing St.
ZIP: SW1A 2AA
City: London
Country: United Kingdom

Example from Golfarelli and Rizzi book

Comparing Strings

- How to compare strings?

  Patrick Smith vs. Patrik Smyth

- Popular distance measure: edit distance
  - Insert, Delete, Overwrite
  - Shortest sequence of operations to turn one string into another

Patri<u>c</u>k Smith $\rightarrow_D$ Patrik Sm<u>i</u>th $\rightarrow_R$ Patrik Sm<u>y</u>th

Edit Distance

- Relatively easy to compute via dynamic programming

Smyth vs Simte

d(S,S)

d(Smyt,Si)

		S	m	y	t	h
	0	1	2	3	4	5
S	1	0	1	2	3	4
i	2	1	1	2	3	4
m	3	2	1	2	3	4
t	4	3	2	2	2	3
e	5	4	3	3	3	3

d(Smyth,Simte)

d(Smyt,Simt)

Edit Distance

- Compute content of a cell as follows:

Example:

d(Smyth,Simte):

d(Smyth,Simt) + insert e at the end

or delete h + d(Smyt,Simte)

or d(Smyt,Simt) + overwrite h with e

Edit Distance

- Compute content of a cell as follows:

Example:

d(Smyth,Simte) = min(

d(Smyth,Simt) + 1

d(Smyt,Simte) + 1,

d(Smyt,Simt) + 1

)

Edit Distance

- Compute content of a cell as follows:

Example:

d(Smyt,Simt):

d(Smyt,Sim) + insert t at the end

or delete t + d(Smy,Simt)

or d(Smy,Sim)

Edit Distance

- Compute content of a cell as follows:

Example:

d(Smyt,Simt) = min(

d(Smyt,Sim) + 1

d(Smy,Simt) + 1,

d(Smy,Sim) **+ 0**

)

Edit Distance

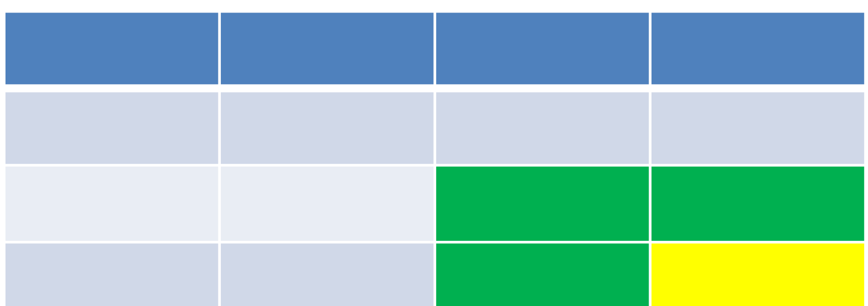
- Compute content of a cell as follows:

if $string_1[k] = string_2[l]$

$$M[k,l] = \min(M[k-1,l-1],\ M[k-1,l]+1,\ M[k,l-1]+1)$$

else:

$$M[k,l] = \min(M[k-1,l-1]+1,\ M[k-1,l]+1,\ M[k,l-1]+1)$$



Summary

- Data Warehousing architectures
  – Single, two-, and three-layer architectures

- ETL process
  – Extract
  – Transform/Cleanse
  – Load

Populating Dimension Tables

- Type 1 and type 2 changes
  - May require dimension table to be loaded in the data staging area
- Introduction of surrogate keys
  - Permanently maintain mapping tables in the staging area
    - In case of a type-1 change: change the value in the dimension table
    - In case of a type-2 change: introduce a new tuple with a new surrogate key

Populating Fact Tables

- Always update dimension tables before fact table (if possible)
  – Referential integrity
  – Not always possible: introduce “dummy” value in dimension tables if dimension key lookup fails
- Update materialized views
  – Can be optimized if measures are distributive

Computation of Aggregations

- Part of the data warehouse loading involves computation of aggregations → materialized views
  - Order in which they are computed can be important
  - Sort-based or Hash-based

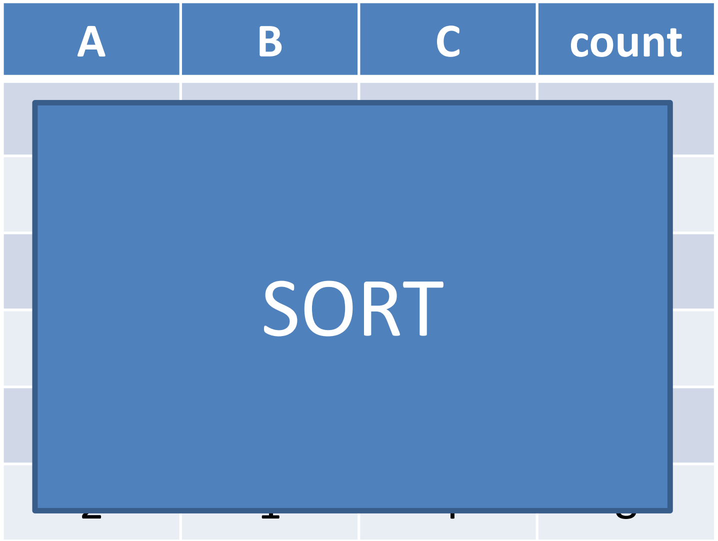
See, e.g.: Agarwal et al. On the computation of multidimensional aggregates. VLDB 1996

Sort-Based Aggregation

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

Sort-Based Aggregation



SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	8

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14
1	8	6	10

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14
1	8	6	10
2	1	4	8

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14
1	8	6	10
2	1	4	17

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14
1	8	6	10
2	1	4	17
3	3	3	5

Sort-Based Aggregation

A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

SCAN

SELECT A, B, C, sum(count)
FROM R
GROUP BY A, B, C;

A	B	C	SUM
1	5	6	14
1	8	6	10
2	1	4	17
3	3	3	5

Sort-Based Aggregation

- Observation:
  - Table sorted on ABC

    = sorted on AB

    = sorted on A
  - One sort supports 3 aggregations

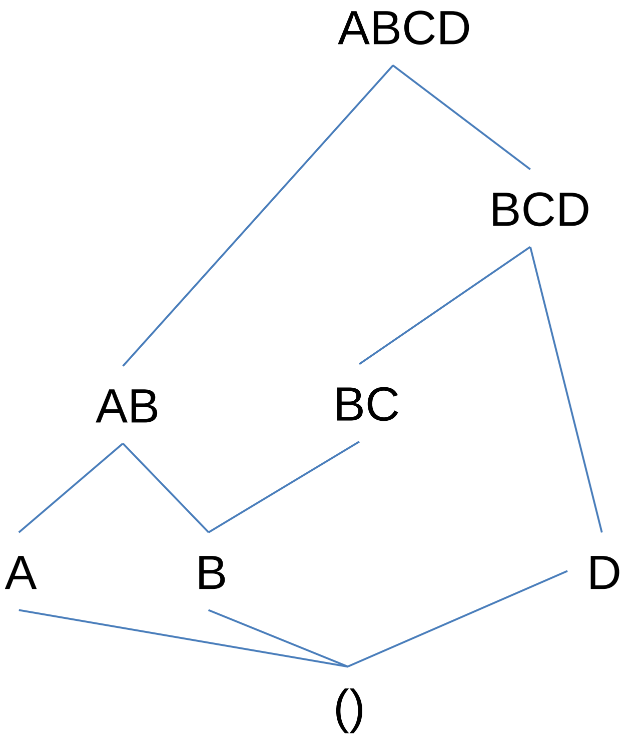
A	B	C	count
1	5	6	8
1	5	6	6
1	8	6	10
2	1	4	8
2	1	4	9
3	3	3	5

Pipe-Sort

- Sort on ABC
- Scan sorted relation
  – As long as next tuple is the same as previous
    - Update aggregated tuple; add count of current
  – Else
    - Ship aggregated tuple to disk
    - Pipe tuple to procedure computing aggregation on AB

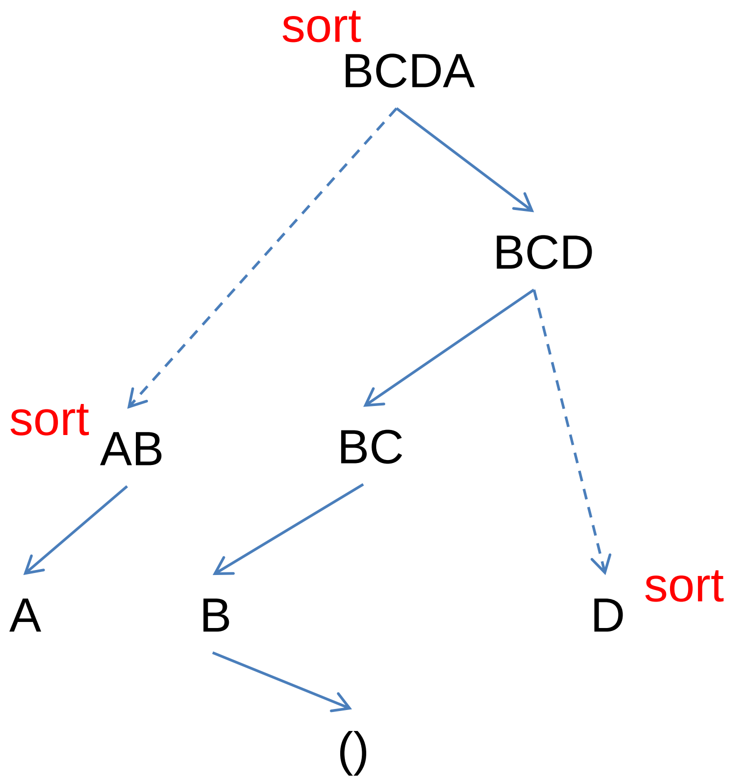
Pipe-Sort

- Key problem: divide materialized views lattice into “pipes”, minimizing sorts



Pipe-Sort

- Key problem: divide materialized views lattice into "pipes", minimizing sorts



Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	8

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	8
2	1	4	9

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	8
2	1	4	9
1	8	6	10

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	**14**
2	1	4	9
1	8	6	10

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	14
2	1	4	9
1	8	6	10
3	3	3	5

Hash-Based Optimization

- If aggregate tables fit into memory

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

SCAN

A	B	C	SUM
1	5	6	14
2	1	4	**17**
1	8	6	10
3	3	3	5

Hash-Based Optimization

- Multiple hash tables may fit at the same time

A	B	C	count
1	5	6	8
2	1	4	9
1	8	6	10
1	5	6	6
3	3	3	5
2	1	4	8

A	B	C	SUM
1	5	6	14
2	1	4	17
1	8	6	10
3	3	3	5

A	C	SUM
1	6	24
2	4	17
3	3	5

C	SUM
6	24
4	17
3	5

Conclusion

- ETL is often the most time consuming parts of a datawarehousing project
  - Reported to take up to 80% of time
- Different ways to capture changes
  - Application-assisted
  - Trigger-based extraction
  - Log-based
  - Timestamp-based
  - File Comparison

Conclusion

- Transform/Cleanse involves
  - Normalization
  - Standardization
  - De-duplication
  - Entity resolution
- Load
  - Surrogate keys (lookup tables in staging area)
  - First update dimension tables, only then fact table