Extract-Transform-Load

SDRP Lecture 3

Christian Thomsen
chr@cs.aau.dk

Slides adapted from Christian S. Jensen, Torben Bach Pedersen, and Man Lung Yiu

F-Klub case work

- Very few hand-ins received
- Very little activity last Friday

- There will be an exam question about the F-Klub case
- And the knowledge and experience will also be useful for another question
- If you hand in at the given deadline, you can bring your solutions/hand-ins to the exam
- I strongly recommend that you work on the F-Klub case tomorrow (and after the exercises if you have time)
  - Read the instructions carefully – -v2 uploaded on Moodle
  - Catch up from session 1 if you are behind
  - Let me know if you need some extra time

Last time

- Slowly Changing Dimensions (SCDs)
  - Type 1, Type 2, Type 3, Type 4
- Special dimensions
  - Outriggers
  - Degenerate dimensions
  - Junk dimensions
  - Role-playing dimensions
- Hierarchies
  - Variable-depth hierarchies
  - Unbalanced, non-covering, non-strict hierarchies
  - Multiple and parallel hierarchies

- Today: Extract-Transform-Load (ETL)

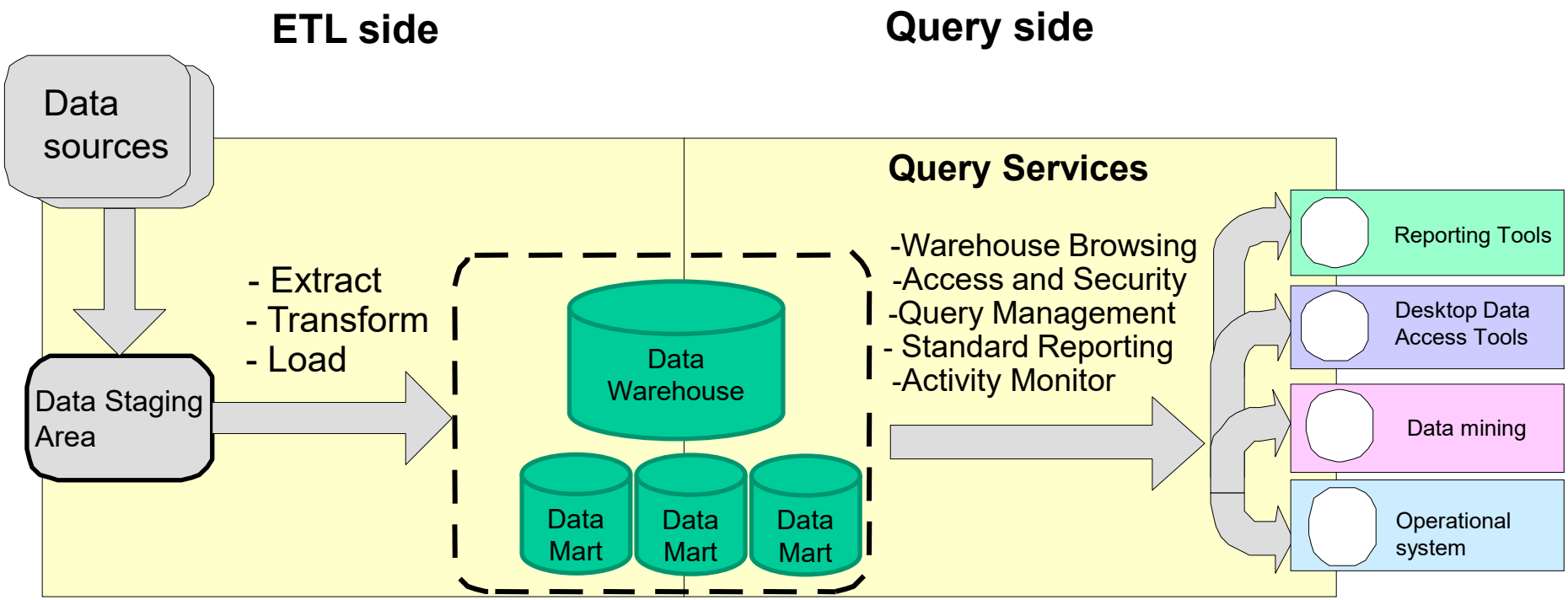
ETL Overview

- The ETL Process
- Extract
- Transform
- Load
- Kimball's ETL Construction Process
- Tools: *Apache Hop* and *pygrametl*

The ETL Process

- The **most underestimated** process in DW development
- The **most time-consuming** process in DW development
  - Up to 70% of the development time is spent on ETL!
- Extract
  - Extract relevant data
- Transform
  - Transform data to DW format
  - Build DW keys, etc.
  - Cleansing of data
- Load
  - Load data into DW
  - Build aggregates, etc.

ETL in the Architecture



Data Staging Area (DSA)

- Transit storage for data in the ETL process
  - Transformations/cleansing done here
- No user queries
- Sequential operations on large data volumes
  - Performed by central ETL logic
  - Easily restarted
  - Reduced need for locking, logging, etc.
  - RDBMS or flat files?
- Finished dimensions copied from DSA to relevant marts
- Allows centralized backup/recovery
  - Better to do this centrally in DSA than in all data marts

Extract

Types of Data Sources

- Cooperative sources
  - Replicated source – publish/subscribe mechanism
  - Call back source – calls external code (ETL) when changes occur
  - Internal action source – only internal actions when changes occur
    - Example: DB with triggers
- Non-cooperative sources
  - Snapshot source – provides only full copy of source, e.g., files
  - Specific source – each is different, e.g., legacy systems
  - Logged source – writes change log, e.g., DB log
  - Queryable source – provides query interface, e.g., RDBMS

- Extract strategy depends on the source types

Extract

- Goal: fast extract of relevant data
  - Extract from source systems can take **long** time
- Types of extracts:
  - Extract applications (SQL): co-existence with other applications
  - DB unload tools: faster than SQL-based extracts
    - e.g., MS SQL Export Wizard, MySQL DB dump
- Extract applications the only solution in some scenarios
- **Too** time consuming to ETL all data at each load
  - Can take days/weeks
  - Drain on the operational systems and DW systems
- Consider only the changes since last load

Computing Deltas

- Delta = changes since last load
- Store sorted, complete extracts in the DSA
  - Delta can be computed from the current and the previous extracts
  - + Always possible
  - + Handles deletions
  - - High extraction and processing time
- Put “audit timestamp” on all rows (in the source)
  - Can be updated by a DB trigger or the source application
    - - Source system must be changed, operational overhead
    - Make sure that the timestamp is reliable
  - Extract only where “timestamp > time for last extract”
    - + Reduces extract time
  - - Cannot (alone) handle deletions

Last extract time: 300

Timestamp	DKK	…
100	10	…
200	20	…
300	15	…
400	60	…
500	33	…

Changed Data Capture (CDC)

- Finding the delta is also called “Changed Data Capture”. Other CDC techniques include:
- Messages
  - Applications insert messages in a queue when they do updates
  - + Works for all types of updates and systems
  - - Operational applications must be changed
  - - Operational overhead
- DB triggers
  - Triggers execute actions on INSERT/UPDATE/DELETE
  - + Operational applications need **not** be changed
  - + Enables real-time update of DW
  - - Operational overhead

CDC (cont.)

- Replication based on DB log
  - Find changes directly in DB log which is written anyway
  - + Operational applications need **not** be changed
  - + No operational overhead
  - - A very complex task… Don’t do this yourself
  - (Tools exist for many systems, incl. SQL Server, Oracle, and DB2)

Transform

Common Transformations

- Data type conversions
  - EBCDIC → ASCII/Unicode
  - String manipulations
  - Date/time format conversions
    - 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

Data Quality

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

Cleansing

- Why cleansing? ***Garbage In Garbage Out***
- BI does not work on “raw” data
  - Pre-processing necessary for BI analysis (+ ML/AI)
- Handle inconsistent data formats
  - Spellings, codings, …
- Remove unnecessary attributes
  - Production keys, comments,…
- Replace codes with text for easy understanding, e.g.,
  - department name instead of a code: *Finance* vs. *DPT8*
  - country name instead of country code: *Ireland* vs. *IE*
- Combine data from multiple sources with common key
  - E.g., customer data from marketing, sales, support, …

Types of Cleansing

- Conversion and normalization
  - Most common type of cleansing
  - Text coding, date formats
    - does 3/2 mean 3rd February or 2nd March?

- Special-purpose cleansing
  - Look-up tables and dictionaries to find valid data, synonyms, abbreviations
  - Normalize spellings of names, addresses, etc.
    - Dorset *Rd* or Dorset *Road*?
    - København or Copenhagen?
    - Aalborg or Ålborg?
  - Remove duplicates, e.g., duplicate customers

Types of Cleansing

- Approximate, “fuzzy” joins on records from different sources
  - Example: two customers are regarded as the same if their respective values match for most of the attributes (e.g., address, phone number)
- Domain-dependent cleansing
  - Specialized solutions for address normalization exist, for example
- Rule-based cleansing
  - User-specified rules: if-then-else style
  - Automatic rules: use data mining to find patterns in data
    - Guess missing sales person based on customer and item

Cleansing

- Uniform treatment of NULL
  - Use NULLs only for measure values (estimates instead?)
  - Use a special dimension value instead of NULL dimension values
    - E.g., for the time dimension, instead of NULL, use special key values to represent “Date not known”, “Soon to happen”, and so on
    - Avoids problems in joins, since NULL is not equal to NULL
    - Yet another good reason to **use surrogate keys**!
- Remove *special codes* (e.g., -1 for “Deactivated”) in your DW data
  - They are hard to understand in query/analysis operations
  - The users won’t understand them

Cleansing

- Apply tests to the data (“age is non-negative”, “salary > …”)

- Mark facts with **Data Status dimension**
  - Normal, abnormal, outside bounds, impossible,…
  - Facts can be taken in/out of analyses

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

- Mark facts with an **Audit dimension** telling about the ETL run

AID	StartTime	EndTime	Processed records	Rejected records	ETL version	…
1	1/1-2020 3:00	1/1-2020 3:35	350295	3	1.26	…
2	…	…	…	…	…	…

Improving Data Quality

- Appoint “data steward”
  - Responsibility for data quality
  - Includes manual inspections and corrections!
- DW-controlled improvement
  - Default values
  - ”Not yet assigned 157” note to data steward
- Source-controlled improvements
- Construct programs that check data quality
  - Are totals as expected?
  - Do results agree with alternative source?
  - Number of NULL values?

Load

Load

- Goal: fast loading into DW
  - Loading deltas is much faster than a total load
- SQL-based update is **slow**
  - Large overhead (optimization, locking, etc.) for every SQL call
  - DB load tools are much faster
- Index on a table can **slow down** the load significantly
  - Consider to drop the index and rebuild it after the load
  - Can be done per index partition
- Parallelization
  - Dimensions can be loaded concurrently
  - Fact tables can be loaded concurrently
  - Partitions can be loaded concurrently

Load

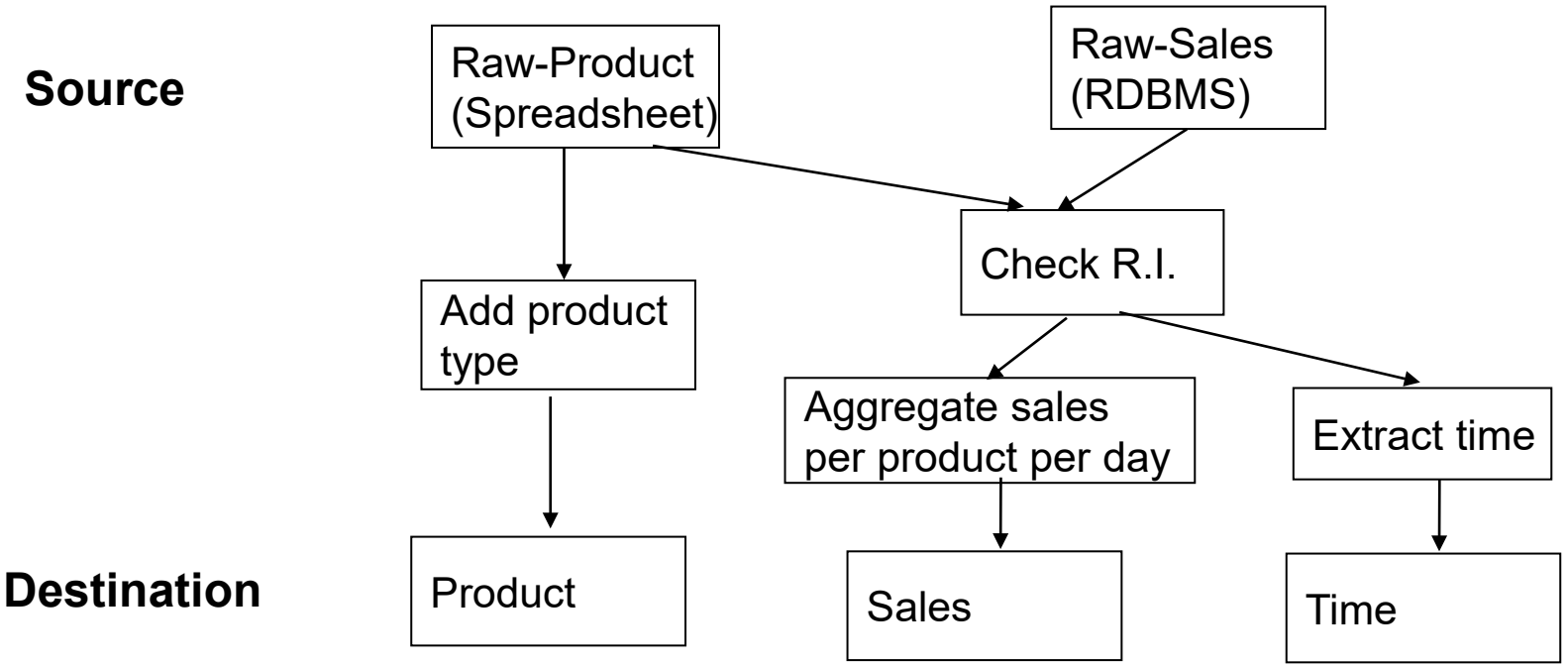
- Relationships in the data
  - Referential integrity and data consistency must be ensured before loading
    - FKs are often not declared or enforced in the DW
- Aggregates
  - Can be built and loaded at the same time as the detail data
- Load tuning
  - Load without log
  - Sort load file first
  - Make only simple transformations in loader
  - Use loader facilities for building aggregates

Kimball's ETL Construction Process

- **Plan**
  1) Make high-level diagram of source-destination flow
  2) Choose an ETL tool
  3) Develop default strategies for dimension management, error handing, etc.
  4) Focus on each target table
- Construction of **one-time historic load** process
  5) Build and test historic dimension loads
  6) Build and test historic fact loads
- Construction of **incremental load** process
  7) Build and test incremental dimension loads
  8) Build and test incremental fact loads
  9) Build and test aggregate loads and/or OLAP processing
  10) Design, construct, and test ETL automation

High-level Diagram

- Step 1: Make high-level diagram of source-destination flow
  - Mainly used for communication purpose
  - 1-2 pages, highlight sources/destinations, questions, and challenges



ETL Tool and Default Strategies

- Step 2: Choose an ETL tool
  - Use graphical ETL tool or hand-coded scripts?
    - Graphical tools are (somewhat) self-documenting – but may have steep learning curves
  - Many tools are available – the "best choice" depends on your case
- Step 3: Develop ***default*** strategies
  - How to extract from the source systems?
  - How and how long should extracted data be archived?
  - How should data quality be checked? What happens to bad data?
  - How are changes (SCDs) handled?
  - How are requirements for availability met?
  - How is auditing done?
  - How is staging (writing to disk for later usage) done?

Individual Targets and Historic Dimension Load

- Step 4: Focus on each target table
  - Where do we get data (for each column) from?
  - Ensure that hierarchies are clean
  - Plan the detailed restructurings
- Step 5: Load dimensions with *historic* data
  - Historic data loading is done *once* – incremental loading is done many times. You may or may not have separate processes
  - Map from production codes to meaningful text equivalents
  - Avoid NULLs
  - Check if one-to-one and one-to-many relationships are as expected
  - Assign surrogate keys
    - Maintain a *map* table between production keys and surrogate keys
  - Load efficiently (more about this soon)
  - Load *static* dimensions (e.g., Date)

Historic Fact Load

- Step 6: Load historic facts
  - Probably you are dealing with huge data volumes
  - Typically, only few transformations needed
  - NULLs may be OK for measure values, but foreign keys to dimensions should NOT be NULL
  - Maintain referential integrity (RI) – do not point to a non-existing dimension value. Lookup the surrogate keys
    - Dimensions should be loaded before the facts
  - Use an audit dimension to track when a fact was added (i.e. to track information about the ETL invocation)
  - Load data efficiently (“bulkload”)

Incremental Loads

- Step 7: Incremental dimension load
  - Identify new and changed data
  - Add new members (and update mapping to surrogate keys)
  - Handle changed attributes for SCDs (update mapping to *newest* surrogate key)
    - Hash values can be used to detect differences
  - For small dimensions, it may be easier to rebuild them completely
- Step 8: Incremental fact load
  - A subset of the map between production keys and surrogate keys can be cached in a preprocessing step
  - Fast loading possible for partitions
  - Parallel processing possible?

Automation

- Step 9: Maintain *aggregates* and build OLAP cubes

- Step 10: Automate the ETL system
  - Scheduling (time and events)
  - Error handling
  - Backups
  - …

A Hint on ETL Design

- **Don’t** implement all transformations in one step!
  - Build first step and check that result is as expected
  - Add second step and execute both, check the result
  - Add third step …
- Try to do *one thing at a time*

Quiz 3: ETL

Q3.1 ETL means

- [ ] A) External Transfer Layer
- [ ] B) Extensible Transformations Language
- [ ] C) Extract-Transform-Load
- [ ] D) Easy Transactional Loading

Q3.2 Creating the ETL flow typically takes

- [ ] A) a month
- [ ] B) little of the entire development time
- [ ] C) much of the entire development time

Q3.3 The DSA is

- [ ] A) used by the ETL process exclusively
- [ ] B) used to answer certain user queries very quickly
- [ ] C) freely available to the end user
- [ ] D) a 1:1 copy of the operational sources

Quiz 3, cont.

Q3.4 In a "delta load",

- [ ] A) the entire source data set is processed into the DW again
- [ ] B) only new and updated source data is processed into the DW
- [ ] C) the ETL process runs with a very low priority

Q3.5 When loading facts into the fact table F, an index on F

- [ ] A) makes the performance worse
- [ ] B) does not affect the performance
- [ ] C) makes the performance better

Q3.6 In a fact table, NULLs

- [ ] A) are OK to use for unknown measure values
- [ ] B) are OK to use for unknown dimension values
- [ ] C) should be avoided by all costs

Q3.7 Typical ETL processing is supported by

- [ ] A) Microsoft Excel
- [ ] B) specialized ETL tools
- [ ] C) plain SQL

Graphical tools

- Most available ETL tools are GUI-based
  - Apache Hop (quick intro next)
  - PDI
  - SQL Server Integration Services
  - Oracle Warehouse Builder
  - IBM DataStage
  - …
- Afterwards, we will look at an alternative: programming of ETL flows in *pygrametl*

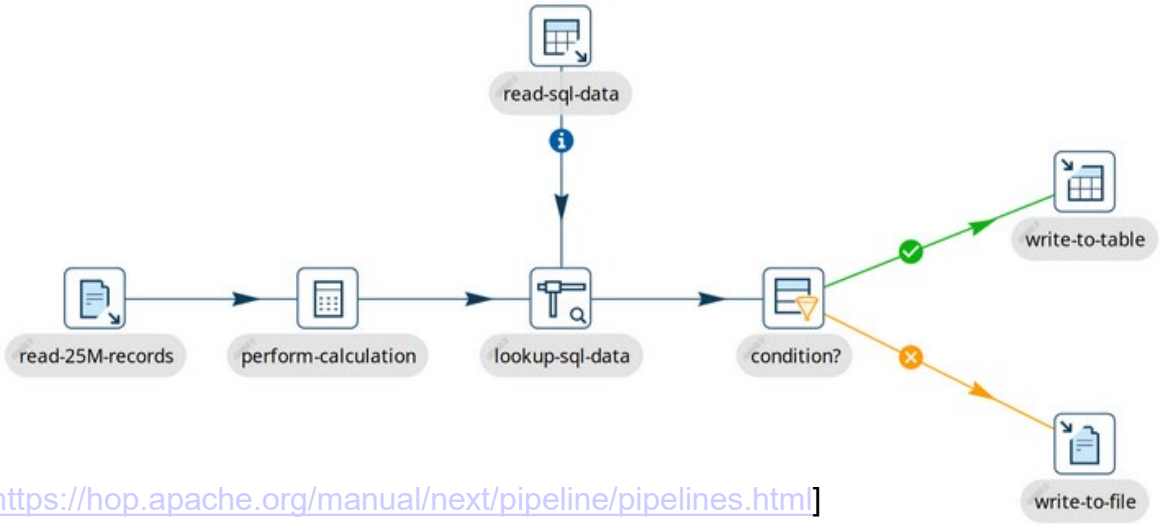
Apache Hop

- Hop = Hop Orchestration Platform
- An open-source ETL tool
- Started as a fork from Pentaho Data Integration (PDI)

- **Hop Gui** is is the visual tool to design workflows and pipelines
- **Workflows** perform orchestration tasks
  - Downloading files, sending emails, …
  - Define dependencies and order, check conditions
- **Pipelines** do the actual work on data
  - A network of logical tasks called **transforms**

Pipeline

- Two main components: **transforms** and **hops**



[Figure from https://hop.apache.org/manual/next/pipeline/pipelines.html]

- 7 transforms in this example (the boxes)
  - Note that the transforms run in parallel
- The data flows via the hops (the arrows show the direction)
  - Hops can be conditional

Transforms

- The building blocks of pipelines
- Many types of transforms in Apache Hop
  - Input, output, dimension lookup/update, scripting, query, join, value mapping, …
  - Currently ~200 transforms

- Picking the right transform for the right task is the key
  - Spend some time to get an overview and read the documentation

pygrametl: ETL in Python

Motivation

- The Extract-Transform-Load (ETL) process is a crucial part for a data warehouse (DW) project
- Many commercial and open source ETL tools exist
- The dominating tools use graphical user interfaces (GUIs)
  - Pros: Easy overview, understood by non-experts, easy to use (?)
  - Cons: A lot of drawing/clicking, missing constructs, inefficient (?)
- GUIs do not automatically lead to high(er) productivity
  - A company experienced similar productivity with coding ETL in C
- Trained specialists use text efficiently
- ETL *developers* are (in our experience) trained specialists

Motivation – cont.

- We wish to challenge the idea that GUIs are always best for ETL
- For some ETL projects, a code-based solution is the right choice
  - "Non-standard" scenarios when …
    - fine-grained control is needed
    - required functionality not available in existing ETL tool
    - doing experimentation
  - Prototyping
  - Teams with limited resources
- Redundancy if each ETL program is coded from scratch
- A framework with common functionality is needed
- ***pygrametl***
  - a Python-based framework for ETL programming

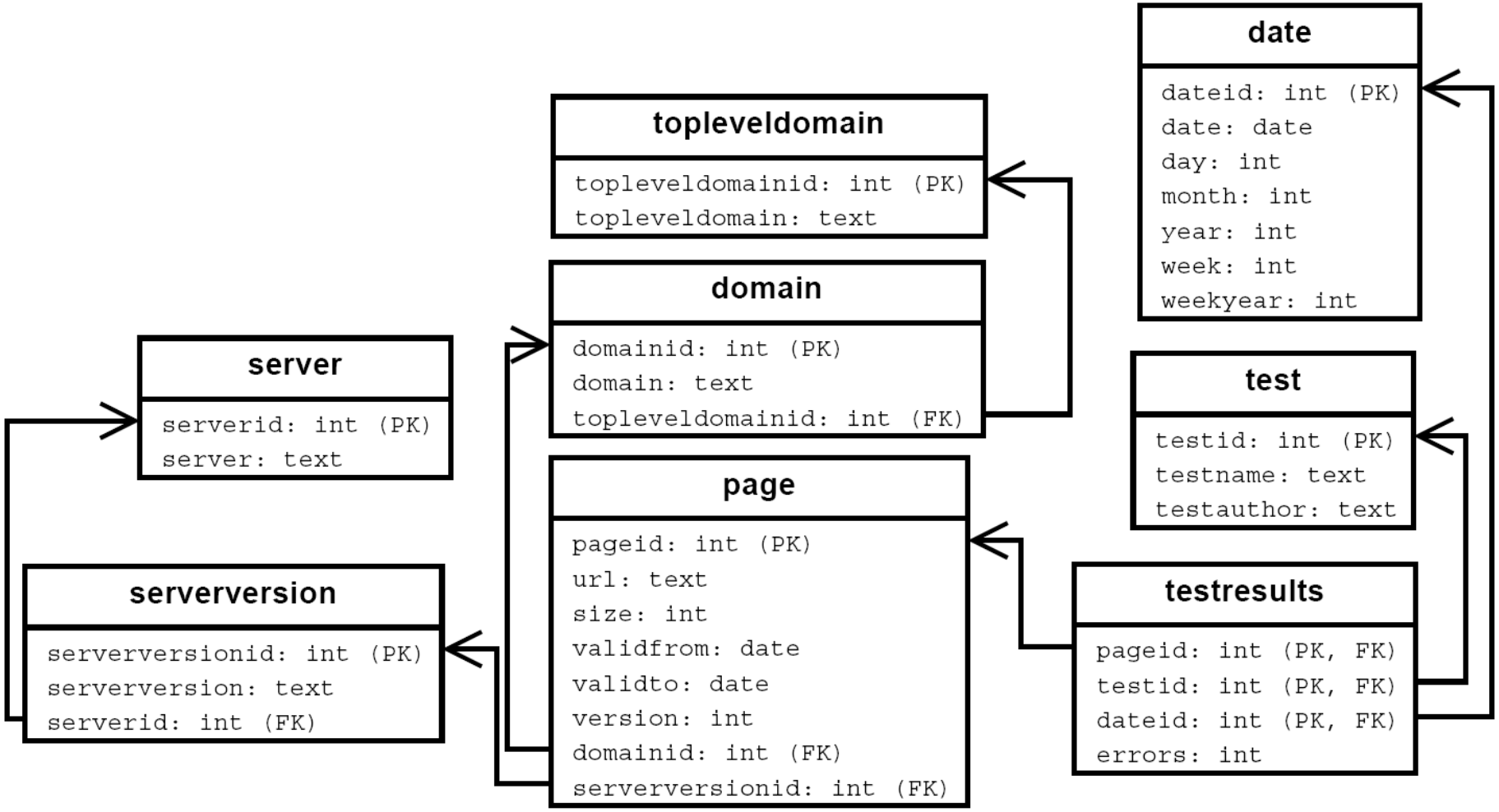
Agenda

- Motivation
- Why Python?
- Example
- Dimension support
- Fact table support
- Flow support
- Evaluation
- New support for testing
- Conclusion and future work

Why Python?

- Designed to support programmer productivity
  - Less typing – a Python program is often 2-10X shorter than a similar Java program
- Good connectivity
- Runs on many platforms (also .NET and Java)
- “Batteries included” – comprehensive standard libraries
- Object-oriented, but also support for functional programming
- Dynamically and strongly typed
- Duck typing

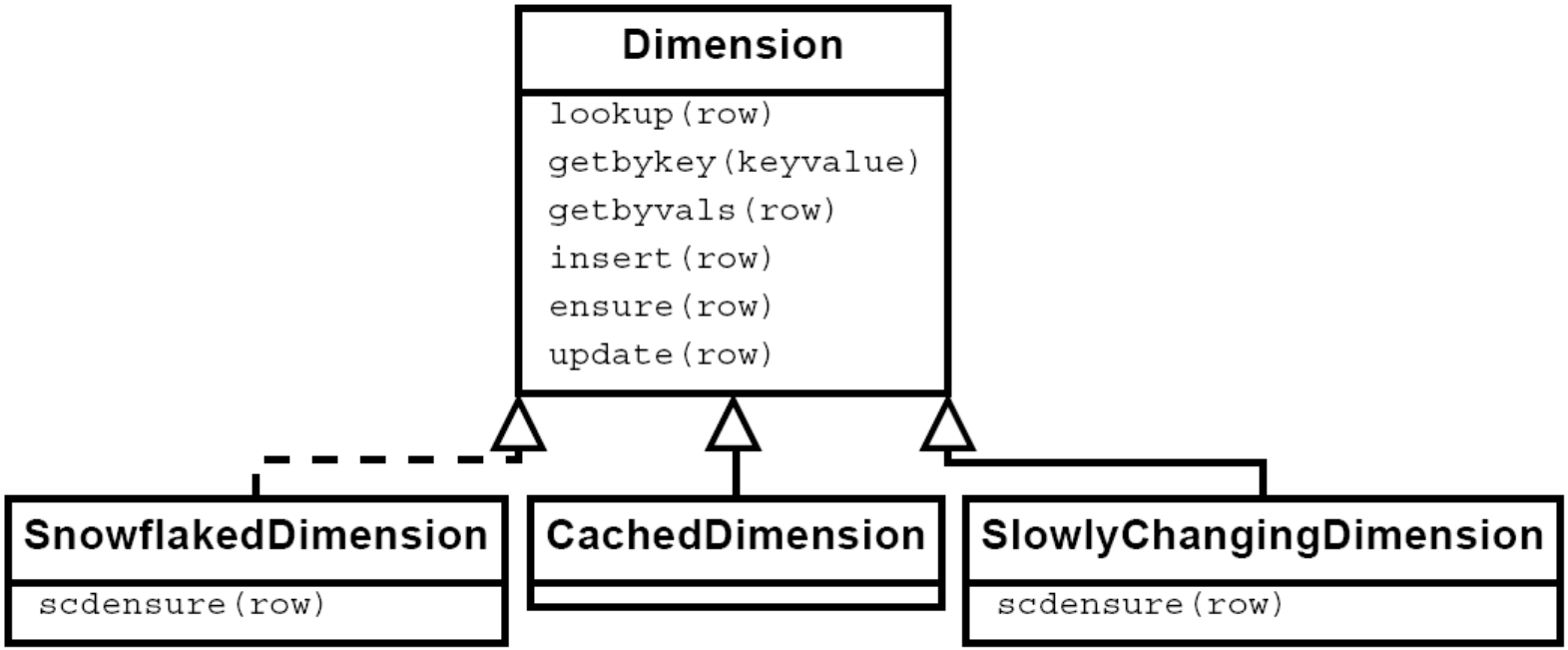
Example


date
dateid: int (PK)
date: date
day: int
month: int
year: int
week: int
weekyear: int
topleveldomain
topleveldomainid: int (PK)
topleveldomain: text
domain
domainid: int (PK)
domain: text
topleveldomainid: int (FK)
server
serverid: int (PK)
server: text
test
testid: int (PK)
testname: text
testauthor: text
page
pageid: int (PK)
url: text
size: int
validfrom: date
validto: date
version: int
domainid: int (FK)
serverversionid: int (FK)
serverversion
serverversionid: int (PK)
serverversion: text
serverid: int (FK)
testresults
pageid: int (PK, FK)
testid: int (PK, FK)
dateid: int (PK, FK)
errors: int

Agenda

- Motivation
- Why Python?
- Example
- ***Dimension support***
- Fact table support
- Flow support
- Evaluation
- New support for testing
- Conclusion and future work

Dimension support



- The general idea is to create one `Dimension` instance for each dimension in the DW and then operate on that instance: `dimobject.insert(row)`

Dimension

```
testdim = Dimension(
    name="test", key="testid",
    attributes=["testname", "testauthor"],
    lookupatts=["testname"],
    defaultidvalue=-1)
```

Required

Optional

Further, we could have set

- `idfinder=somefunction`
  to find key values on-demand when inserting a new member
- `rowexpander=anotherfunction`
  to expand rows on-demand

Dimension's methods

- `lookup(row, namemapping={})`
  Uses the lookup attributes and returns the key value
- `getbykey(keyvalue)`
  Uses the key value and returns the full row
- `getbyvals(row, namemapping={})`
  Uses a subset of the attributes and returns the full row(s)
- `insert(row, namemapping={})`
  Inserts the row (calculates the key value if it is missing)
- `ensure(row, namemapping={})`
  Uses `lookup`. If no result is found, `insert` is used after the optional `rowexpander` has been applied
- `update(row, namemapping={})`
  Updates the row with the given key value to the given values

CachedDimension

- Like a Dimension but with caching (and thus ***much faster!!!***)

```
testdim = CachedDimension(
        name="test", key="testid",
        attributes=["testname", "testauthor"],
        lookupatts=["testname"],
        defaultidvalue=-1
        cachesize=500,
        prefill=True
        cachefullrows=True )
```

SlowlyChangingDimension

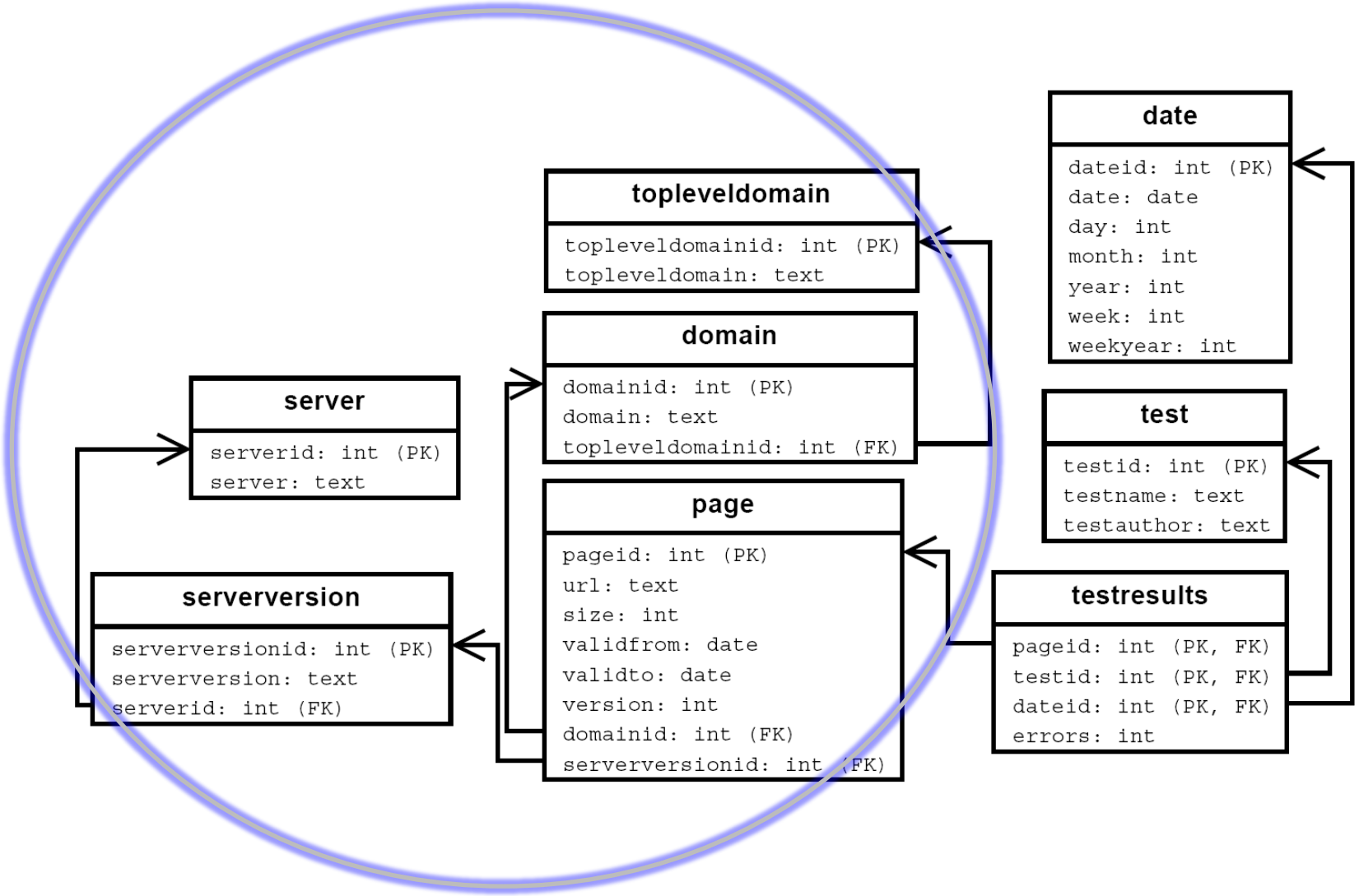
- Supports Slowly Changing Dimensions (type 1 & 2)

```
pagedim = SCDimension(name="page", key="pageid",
  attributes=["url", "size", …],
  lookupatts=["url"],
  fromatt="validfrom",
  fromfinder=pygrametl.datereader("lastmoddate"),
  toatt="validto", versionatt="version")
```

- We could also have given a list of "type 1 attributes", set a `tofinder`, and configured the caching
- Methods like `Dimension` plus `scdensure(row, namemapping={})` that is similar to `ensure` but detects changes and creates new versions if needed

Snowflaked Dimensions

- Recall the running example



SnowflakedDimension

- For filling a *snowflaked* dimension represented by several tables
- Enough to use one method call on a *single* object made from other `Dimension` objects

```
pagesf = SnowflakedDimension \
([   (pagedim, [serverversiondim, domaindim]),
     (serverversiondim, serverdim),
     (domaindim, topleveldim)
 ])
```

- We can then use a single call of `pagesf.ensure(…)` to handle a full lookup/insertion into the snowflaked dimension
- Likewise for `insert`, `scdensure`, etc.

Agenda

- Motivation
- Why Python?
- Example
- Dimension support
- ***Fact table support***
- Flow support
- Evaluation
- New support for testing
- Conclusion and future work

Fact table support

- `FactTable` – a basic representation
  - `insert(row, namemapping={})`
  - `lookup(row, namemapping={})`
  - `ensure(row, namemapping={})`
- `BatchFactTable`
  - Like `FactTable`, but inserts in *batches*.
- `BulkFactTable`
  - Only `insert(row, namemapping={})`
  - Does bulk loading by calling a user-provided function

```
facttbl = BulkFactTable(name="testresults",
    keyrefs=["pageid", "testid", "dateid"],
    measures=["errors"], bulksize=5000000,
    bulkloader=mybulkfunction)
```

Putting it all together

- The ETL program for our example:

```
[Declarations of Dimensions etc.]
 …
def main():
    for row in inputdata:
        extractdomaininfo(row)
        extractserverinfo(row)
        row[“size”] = pygrametl.getint(row[“size”])
        row[“pageid”] = pagesf.scdensure(row)
        row[“dateid”] = datedim.ensure(row)
        row[“testid”] = testdim.lookup(row)
        facttbl.insert(row)
    connection.commit()
```

Flow support

- A good aspect of GUI-based ETL programming is that it is easy to keep different tasks separated
- pygrametl borrows this idea and supports *Steps* (with encapsulated functionality) and flows between them
- A `Step` can have a following `Step`
- The basic class `Step` offers (among other) the methods
  - `defaultworker(row)`
  - `_redirect(row, target)`
  - `_inject(row, target=None)`
- pygrametl has some predefined `Step`s: `MappingStep`, `ValueMappingStep`, `ConditionalStep`, …
- Steps are rarely used by pygrametl users. It seems they want to ”take control” when they code their ETL flow

Agenda

- Motivation
- Why Python?
- Example
- Dimension support
- Fact table support
- Flow support
- ***Evaluation***
- New support for testing
- Conclusion and future work

Evaluation

- We implemented ETL solutions for the example in pygrametl and Pentaho Data Integration (PDI)
  - PDI is a leading open source GUI-based ETL tool
  - Ideally, commercial tools should also have been used but commercial licenses often forbid publication of performance results
- Difficult to make a complete comparison…
  - We have experience with PDI but we wrote pygrametl
  - A full-scale test would require teams with fully trained developers
- We evaluated development time
  - each tool was used twice – in the first use, we had to find a strategy, in the latter use we only found the interaction time
- … and performance
  - on generated data with 100 million facts

Comparison

pygrametl

- 142 lines (incl. whitespace and comments), 56 statements
- 1st use: 1 hour
- 2nd use: 24 minutes
- 9208 facts/sec
- 53% CPU utilization

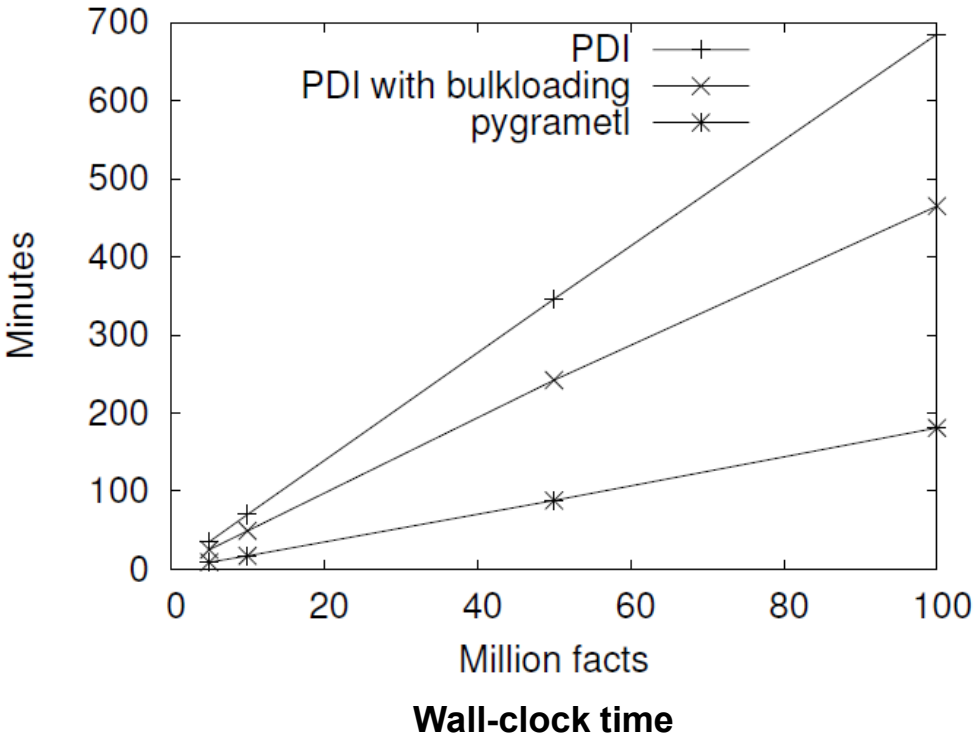
PDI

- 19 boxes and 19 arrows
- 1st use: 2 hours
- 2nd use: 28 minutes
- 2433 (3584) facts/sec
- 83% (93%) CPU util.

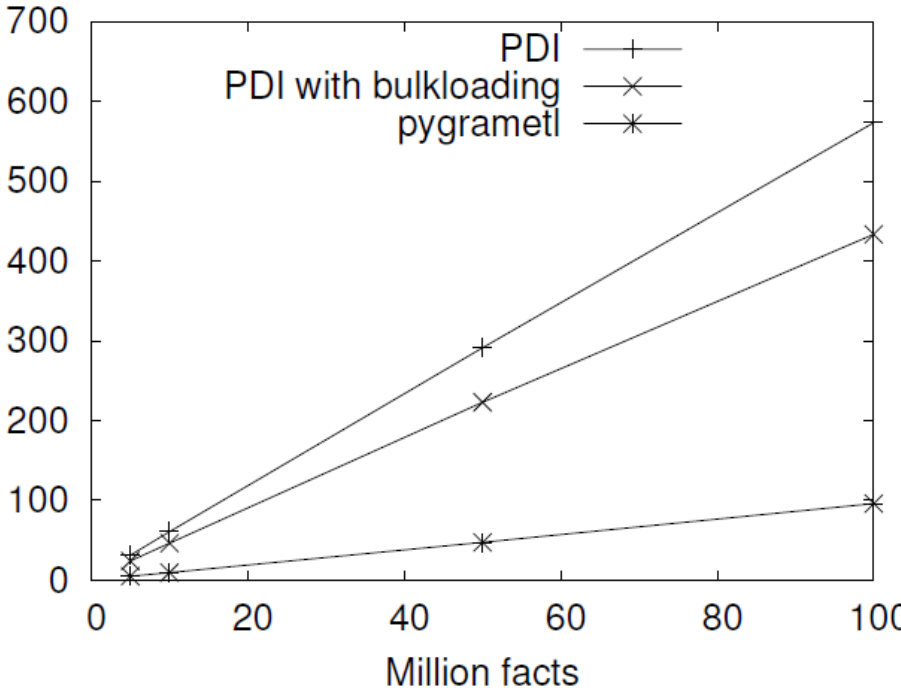
Performance test

- Uses the running example
  - 2,000 domains each with 100 pages and 5 tests
  - ➔ One considered month gives 1 million facts
  - We get ~100,000 new page versions per month after the 1st month
- VirtualBox with 3 virtual CPUs; host has 2.70GHz i7 with 4 cores and hyperthreading
- VirtualBox with 16GB RAM; host has 32GB
- Host has SSD disk
- Linux as guest OS; host runs Windows 10
- Python 3.6, OpenJDK 8, PostgreSQL 9.4
- pygrametl 2.5, PDI 7.1
- Both pygrametl and PDI were allowed to cache all dimension data

Performance



Wall-clock time



CPU time

Agenda

- Motivation
- Why Python?
- Example
- Dimension support
- Fact table support
- Flow support
- Evaluation
- ***New support for testing***
- Conclusion and future work

Testing of ETL

- It is complex to create an ETL flow
- Testing during its development is important
  - ensure correctness
  - correct errors immediately ➔ decreased development time
- It is tedious to define test cases where the state of the database is taken into consideration
  - Use of ”boilerplate code” to load/compare data
- In pygrametl 2.7, we included the ***Drawn Table Testing (DTT) framework***
- DTT makes it easy to define
  - preconditions (the state of the DB *before* the ETL flow runs) and
  - postconditions (the expected state of the DB *after* the flow runs)
- ”Drawn Tables” are used for this

Drawn Table – example

bid:int (pk)	browser:text	os:text
-1	Unknown	Unknown
1	Firefox	Linux
2	Chrome	MacOS
1	Safari	MacOS

- PK, UNIQUE, NOT NULL, FK *table(column)* supported

Drawn Tables

- DTs can be created with the **`Table`** class:

```
tbl = Table("browser", """[Drawn Table]""")
```

- The content can also be loaded from somewhere else **`(loadFrom=`*`iterable`*`)`**
- **`tbl.ensure()`** will create and populate the table if it doesn't already exist with the exact rows (or raise an Error)
- **`tbl.reset()`** will force the table to look like specified

- By default, a SQLite in-memory database is used
  - the user can get and set the **`connectionwrapper`** such that her ETL flow can modify the test database

Working with Drawn Tables

- `expected = tbl + ”| 4 | Chrome | Windows |” \`
  `+ ”| 5 | Opera | Linux |”`
- `tbl.assertEqual()` will check that the table in the DB now looks like this DT
  - If it does not, an `AssertionError` is raised and the differences are printed (this verbosity can optionally be disabled)
- `assertDisjoint` and `assertSubset` are also supported

```
def test_canInsertIntoBrowserDimTable(self):
    exp = tbl + "| 4 | Chrome | Windows |" \
              + "| 5 | Opera | Linux |"
    newrows = exp.additions()
    etl.execute(newrows) # could be any ETL tool
    expected.assertEqual()
```

More about the DTT framework

- Variables are also supported in DTs
  - $*name*
  - When values must be equal, but the exact values are unknown or do not matter
  - $_ is a special variable which is considered equal to anything

- The tool can also be used as a stand-alone tool where DTs are specified in text files (➔ ***no*** Python code needed)
  - Also possible to use DTT with other ETL tools than pygrametl

- If you want to try DTT,
  `import pygrametl.drawntabletesting`

Conclusion and future work

- We challenge the conviction that ETL is always best done by means of a GUI
- We propose to let ETL developers do ETL programming by writing code
- To make this easy, we provide *pygrametl*
  - a Python-based framework for ETL programming
- Some persons prefer a graphical overview of the ETL process
  - The optimal solution includes both a GUI and code
  - Future work includes to make a GUI for creating and connecting steps
  - Updates in code should be visible in GUI and vice versa
    - “Reverse engineering” & “roundtrip engineering”

Getting pygrametl

- Version 2.8 released in September 2023
- The source code, documentation, and the shown example case can be found on **http://pygrametl.org**

- You can also install it with pip
  ```
  pip3 install pygrametl
  ```

- ```
  import pygrametl
  import pygrametl.tables

  mydim = pygrametl.tables.CachedDimension(…)
  ```
  ```

Exercises and F-Klub Case

- Exercises
  - See Moodle
  - Apache Hop and pygrametl

- F-Klub case tomorrow
  - Make the relational schema if you didn't do so already
    - `CREATE TABLE` …
  - If you realize that you made a mistake or forgot something in the first part, you are of course free to change it – include all parts when you submit
  - Make an ETL flow in Apache Hop or pygrametl