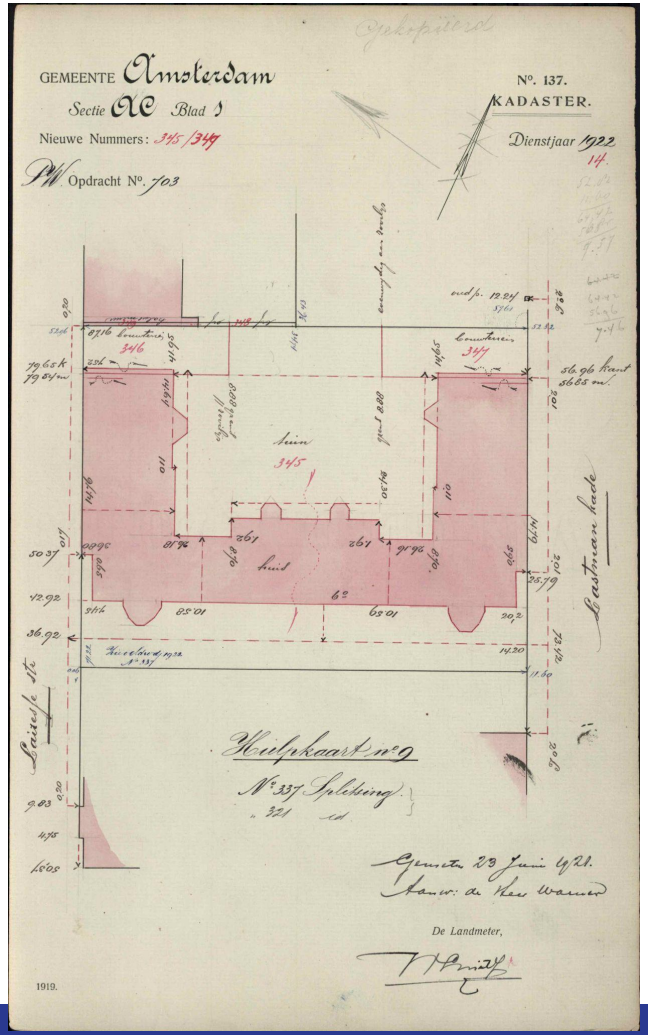
Databases and APIs

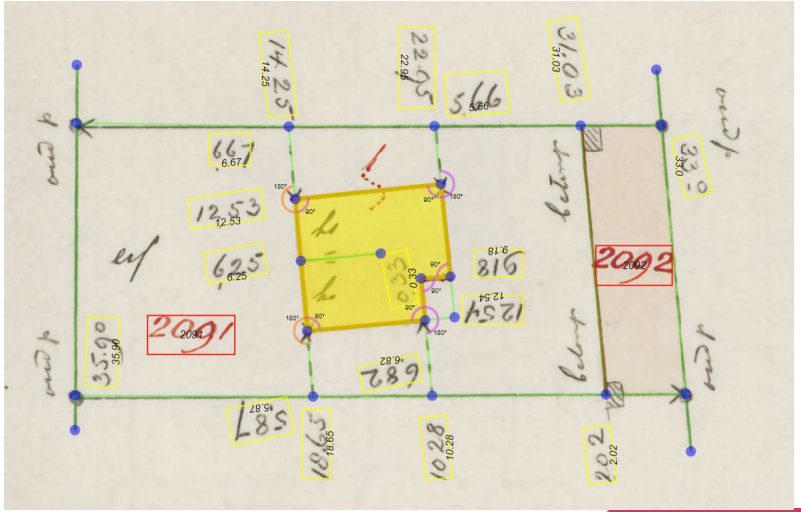
Introduction

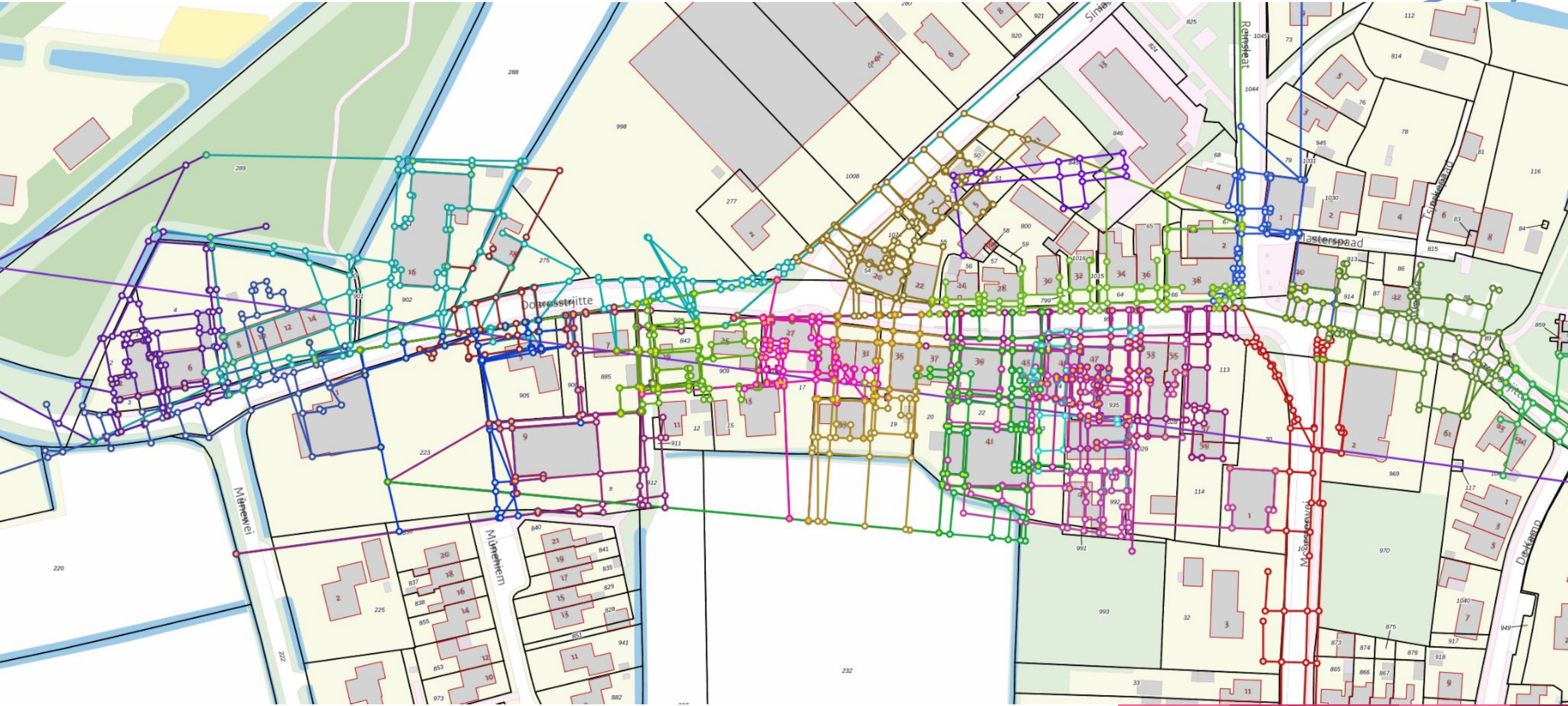
- KPMG
  - Junior Consultant -> Senior Manager
  - Lead of the Architecture and Engineering team
  - Design Data Lake for Monetary Authority of Singapore
  - Design Data Lake for State bank of India
  - Implement Data Lake Fubon Bank in Taiwan
- Independent since 2019
  - Kadaster
  - Sensor analytics for manufacturing process with Philips
  - Model Landscape for DLL
  - Very rarely trainings for Harvest

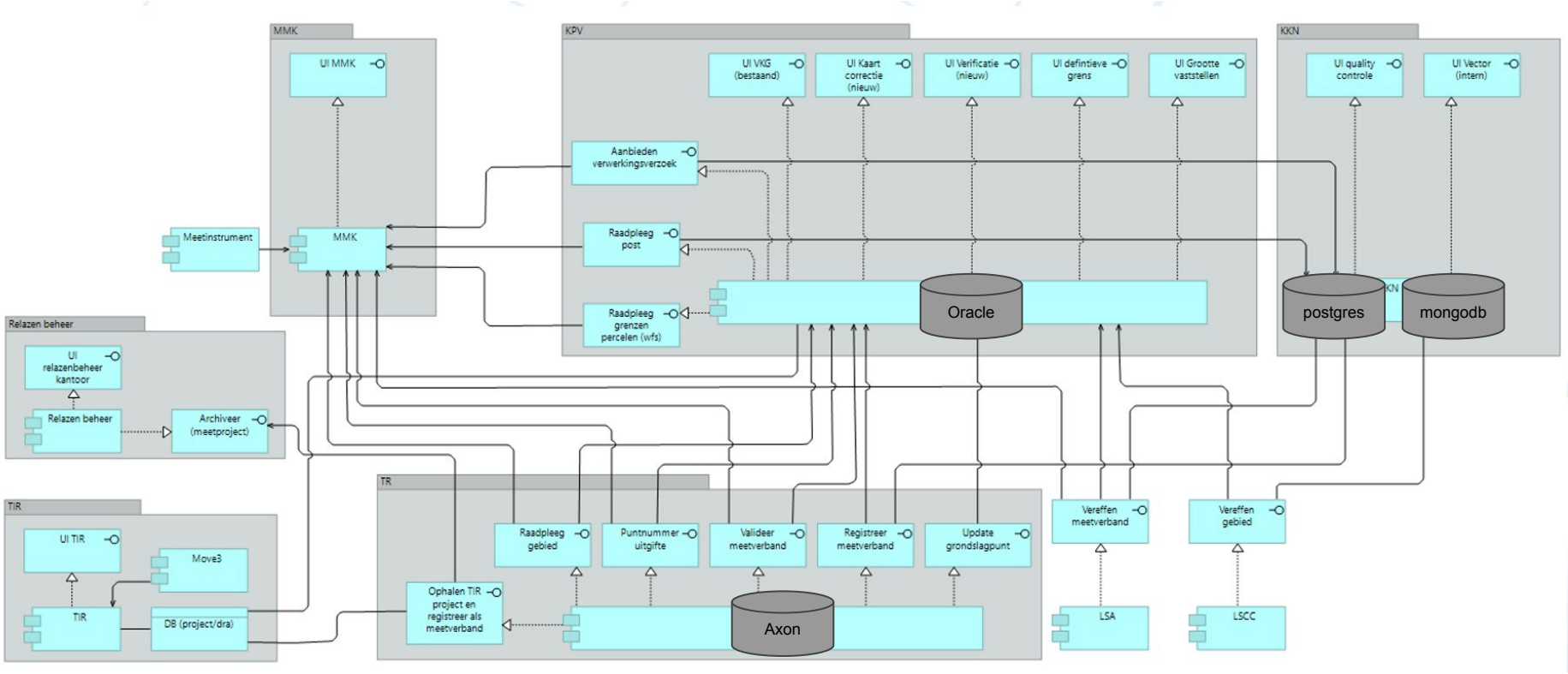
- Architect
  - Software
  - Solution
  - Enterprise
- Data Engineer
- Hadoop specialist
- Kubernetes specialist



5.000.000 Veldwerken
Oude data 1880 - 1950
Hoge compressie JPEG

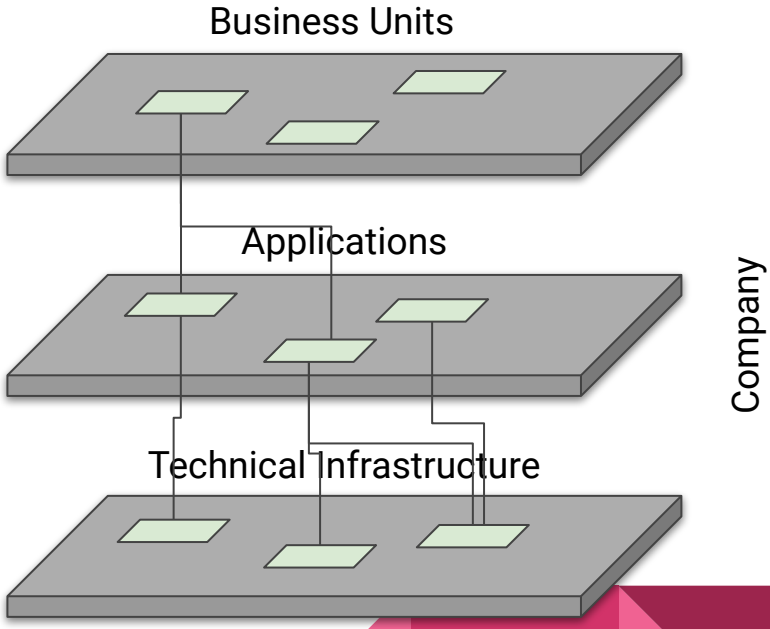



Mûnewei
Mûnehiem

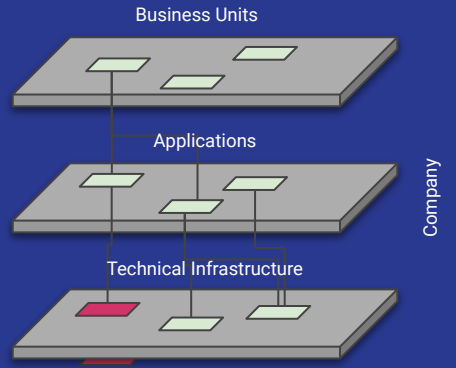

MMK
UI MMK
MMK
Meetinstrument
KPV
UI VKG (bestaand)
UI Kaart correctie (nieuw)
UI Verificatie (nieuw)
UI definitieve grens
UI Grootte vaststellen
Aanbieden verwerkingsverzoek
Raadpleeg post
Raadpleeg grenzen percelen (wfs)
Oracle
KKN
UI quality controle
UI Vector (intern)
postgres
mongodb
Relazen beheer
UI relazenbeheer kantoor
Relazen beheer
Archiveer (meetproject)
TIR
UI TIR
Move3
TIR
DB (project/dra)
TR
Raadpleeg gebied
Puntnummer uitgifte
Valideer meetverband
Registreer meetverband
Update grondslagpunt
Ophalen TIR project en registreer als meetverband
Axon
Vereffen meetverband
Vereffen gebied
LSA
LSCC

Outline

- Databases and structured data
- Building a Data Lake
- Retrieving data from structured and unstructured data sources
- Rest APIs



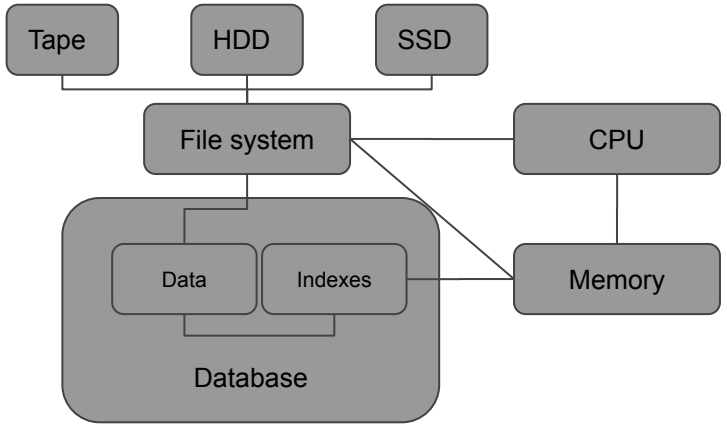
Databases and structured data



General idea

Databases is a bit about history. There are a lot of nuances which we will ignore but everything starts with tape. Data was just a list of bits. How do you find anything? And how can you make it fast?

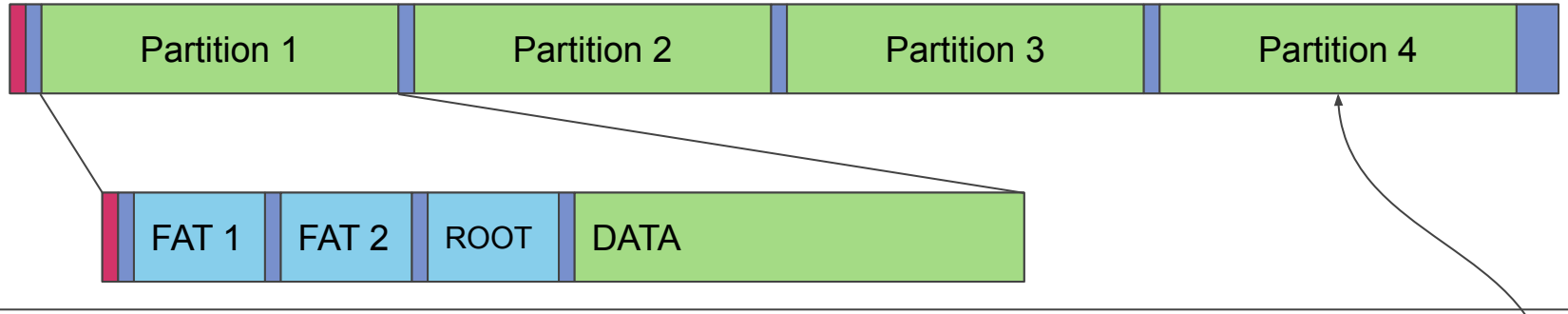
- **Tape drives:** Just a linear length of bits with data.
- **HDD:** Platters with circular storage, however it's still read as a list of bits,
- **SSD:** Random read!

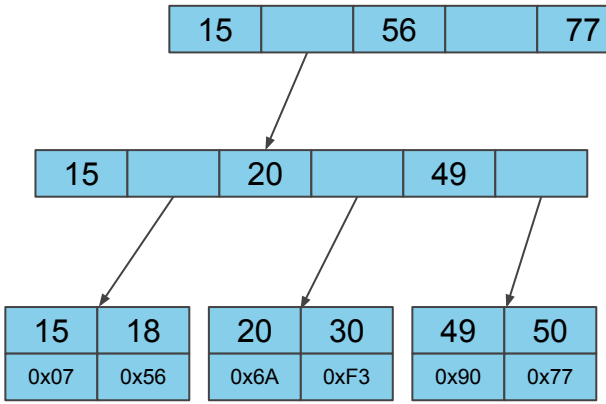


A Query still is the same though:

1. Query comes in
2. Data is selected via indexes if possible
3. Data is transferred from disk to memory
4. Data is passed from memory through CPU for query resolution
5. Result is stored in Memory
6. Result is presented to client

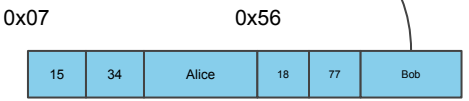
File system and a database (MyISAM)





Fixed width

	Col1	Col2	Col3
0x07	15	34	Alice
0x56	18	77	Bob
0x6A	20	5	Charley
0xF3	30	91	Dave
0x90	49	22	Edward
0x77	50	89	Florence



Bottom line

How can I store my data in a consistent way?

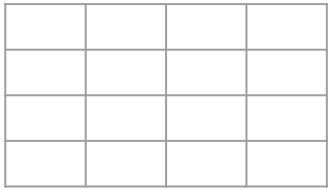
How can I effectively read/write this data?

Indexes

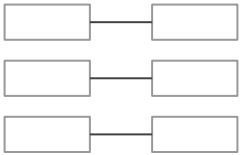
- Binary Tree index
- Hash Index
- Combined (multi-key) index
- Bitmap index
- Partial index
- Spatial index

Types of databases (non-exhaustive)

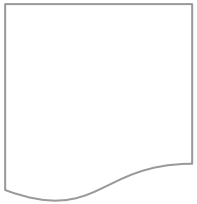
Relational



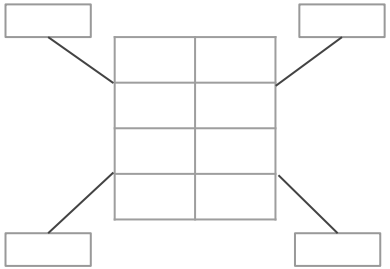
key-value



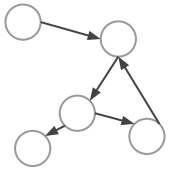
Document



OLAP

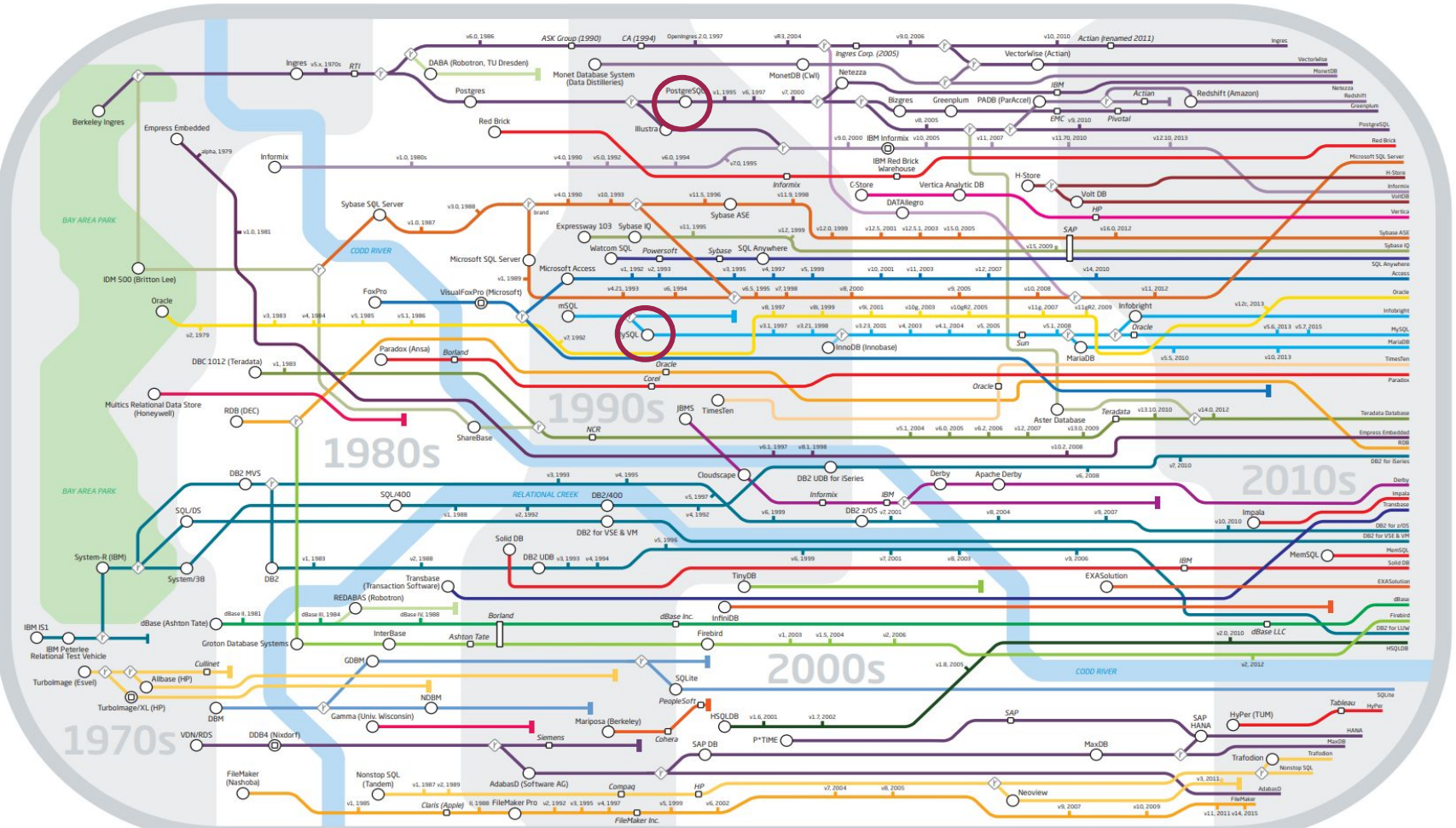


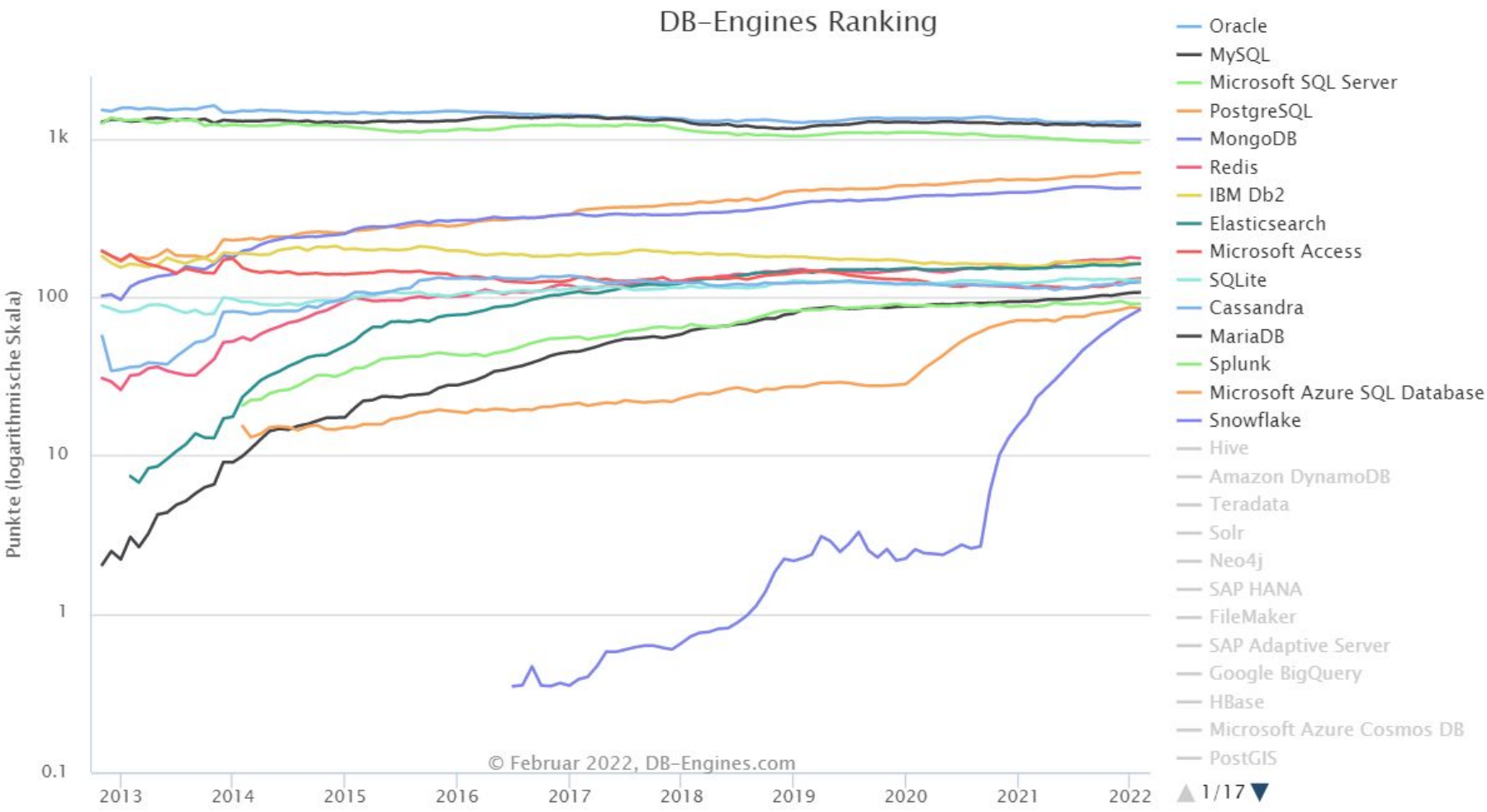
Graph

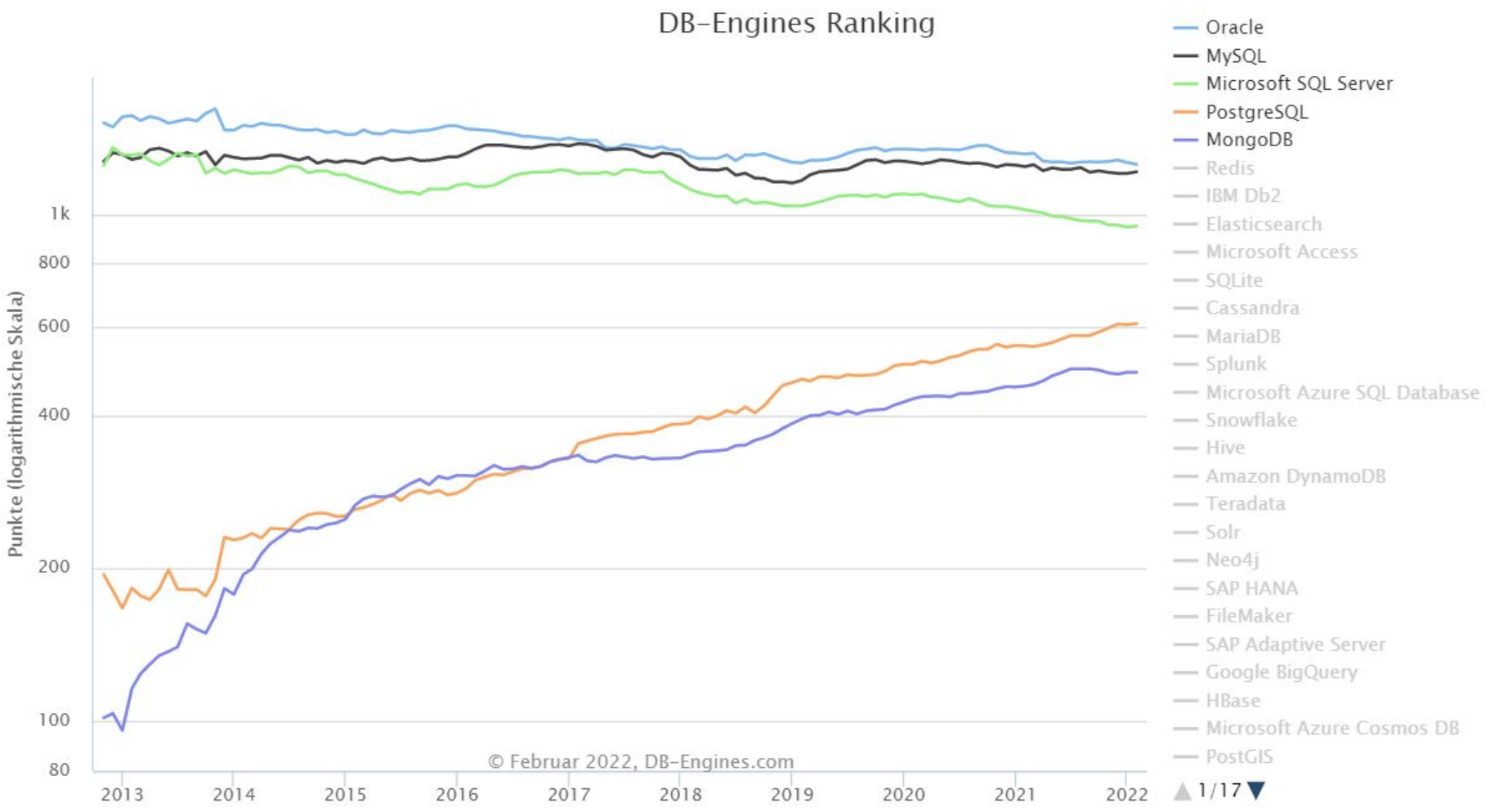


Search engine

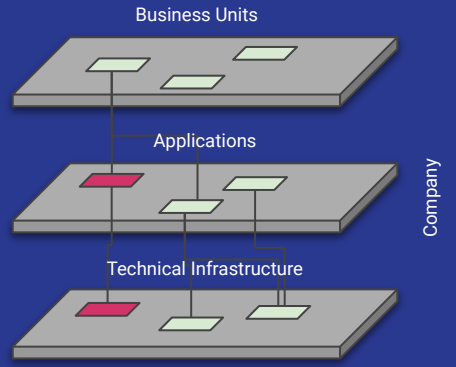
Genealogy of Relational Database Management Systems



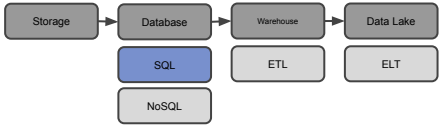

DB-Engines Ranking
Punkte (logarithmische Skala)
1k
100
10
1
0.1
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
© Februar 2022, DB-Engines.com
Oracle
MySQL
Microsoft SQL Server
PostgreSQL
MongoDB
Redis
IBM Db2
Elasticsearch
Microsoft Access
SQLite
Cassandra
MariaDB
Splunk
Microsoft Azure SQL Database
Snowflake
Hive
Amazon DynamoDB
Teradata
Solr
Neo4j
SAP HANA
FileMaker
SAP Adaptive Server
Google BigQuery
HBase
Microsoft Azure Cosmos DB
PostGIS
1/17


DB-Engines Ranking
Punkte (logarithmische Skala)
1k
800
600
400
200
100
80
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
© Februar 2022, DB-Engines.com
Oracle
MySQL
Microsoft SQL Server
PostgreSQL
MongoDB
Redis
IBM Db2
Elasticsearch
Microsoft Access
SQLite
Cassandra
MariaDB
Splunk
Microsoft Azure SQL Database
Snowflake
Hive
Amazon DynamoDB
Teradata
Solr
Neo4j
SAP HANA
FileMaker
SAP Adaptive Server
Google BigQuery
HBase
Microsoft Azure Cosmos DB
PostGIS
1/17

Application Point of View RDBMs



Data normalisation



SCH701		**REGISTRATION**			14FEB21
STUDENT DETAILS					
STUDENT NR	**NAME**	Derksen Th.			
1234	**ADDRESS**	Keistraat 31			
	CITY	Amersfoort			
COURSE NR	**COURSE NAME**	**START DATE**	**BUILDING NR**	**BUILDING NAME**	**LOCATION**
SO	syteemontwerp	12MEI21	2	paviljoen	Utrecht
SA	systeemanalyse	20JUN21	1	toren	Utrecht
PR	programmeren	11AUG21	7	bastille	Enschede
IA	informatie analyse	16OKT21	4	Vrijhof	Zwolle

0e Normaal vorm

SCH701		**REGISTRATION**			14FEB21
STUDENT DETAILS					
STUDENT NR	**NAME**	Derksen Th.			
1234	**ADDRESS**	Keistraat 31			
	CITY	Amersfoort			
COURSE NR	**COURSE NAME**	**START DATE**	**BUILDING NR**	**BUILDING NAME**	**LOCATION**
SO	syteemontwerp	12MEI21	2	paviljoen	Utrecht
SA	systeemanalyse	20JUN21	1	toren	Utrecht
PR	programmeren	11AUG21	7	bastille	Enschede
IA	informatie analyse	16OKT21	4	Vrijhof	Zwolle

Inventariseer alle elementaire gegevens.
Geef procesgegevens en de repeterende groep aan.
Bepaal de sleutel van de hele groep

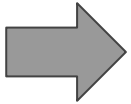
0e

studentnr
name
address
city
course_code
course_name
start_date
building_nr
building_name
location

1e Normaal vorm

0e

<u>studentnr</u>
name
address
city
course_code
course_name
start_date
building_nr
building_name
location



1e

<u>studentnr</u>
name
address
city

<u>studentnr</u>
<u>course_code</u>
course_name
start_date
building_nr
building_name
location

Verwijder de procesgegevens
Zolang een entiteittype een repeterende groep bevat doe:

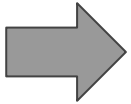
- Splits de repeterende groep af
- Verwijder repeterende groep
- Bepaal sleutel van nieuwe groep

2e Normaal vorm

1e

studentnr
name
address
city

studentnr
course_code
course_name
start_date
building_nr
building_name
location



2e

studentnr
name
address
city

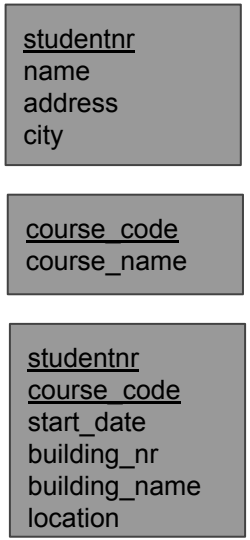
course_code
course_name

studentnr
course_code
start_date
building_nr
building_name
location

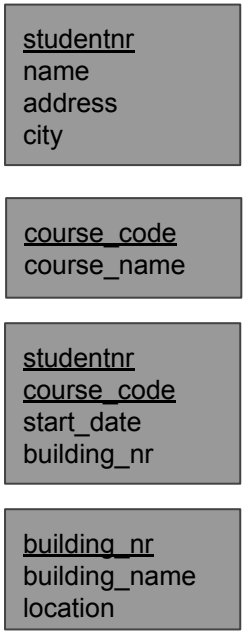
Geef aan of er attributen zijn die niet afhankelijk zijn van de volledige sleutel.
Vorm een aparte groep voor de aangewezen attributen en dat deel van de sleutel waarvan zij functioneel afhankelijk zijn
Verwijder alleen de aangewezen attributen uit de oorspronkelijke groep.

3e Normaal vorm

2e



3e



Geef aan of er attributen voorkomen die ook functioneel afhankelijk zijn van niet-sleutelattributen. Vorm een aparte groep voor de aangewezen attributen en het attribuut waarvan zij functioneel afhankelijk zijn. Verwijder alleen de aangewezen attributen uit de oorspronkelijke groep.

Assignment 1 - Normalize a webshop DB

Publisher Webshop					
Client information					
CustomerNr	**Name**	Derksen Th.			
1234	**Adres**	Keistraat 31			
	City	Amersfoort			
ArticleNr	**Name**	**Price**	**Amount**	**Warehouse**	**Shelf**
50001	Dune	25.95	2	2	12
50002	The Martian	14.00	1	1	14
50003	zen and the art of motorcycle maintenance	15.95	1	7	13

https://dbdiagram.io/

0e Inventariseer alle elementaire gegevens. Geef procesgegevens en de repeterende groep aan. Bepaal de sleutel van de hele groep

1e Verwijder de procesgegevens Zolang een entiteittype een repeterende groep bevat doe:

- Splits de repeterende groep af
- Verwijder repeterende groep
- Bepaal sleutel van nieuwe groep

2e Geef aan of er attributen zijn die niet afhankelijk zijn van de volledige sleutel. Vorm een aparte groep voor de aangewezen attributen en dat deel van de sleutel waarvan zij functioneel afhankelijk zijn
Verwijder alleen de aangewezen attributen uit de oorspronkelijke groep.

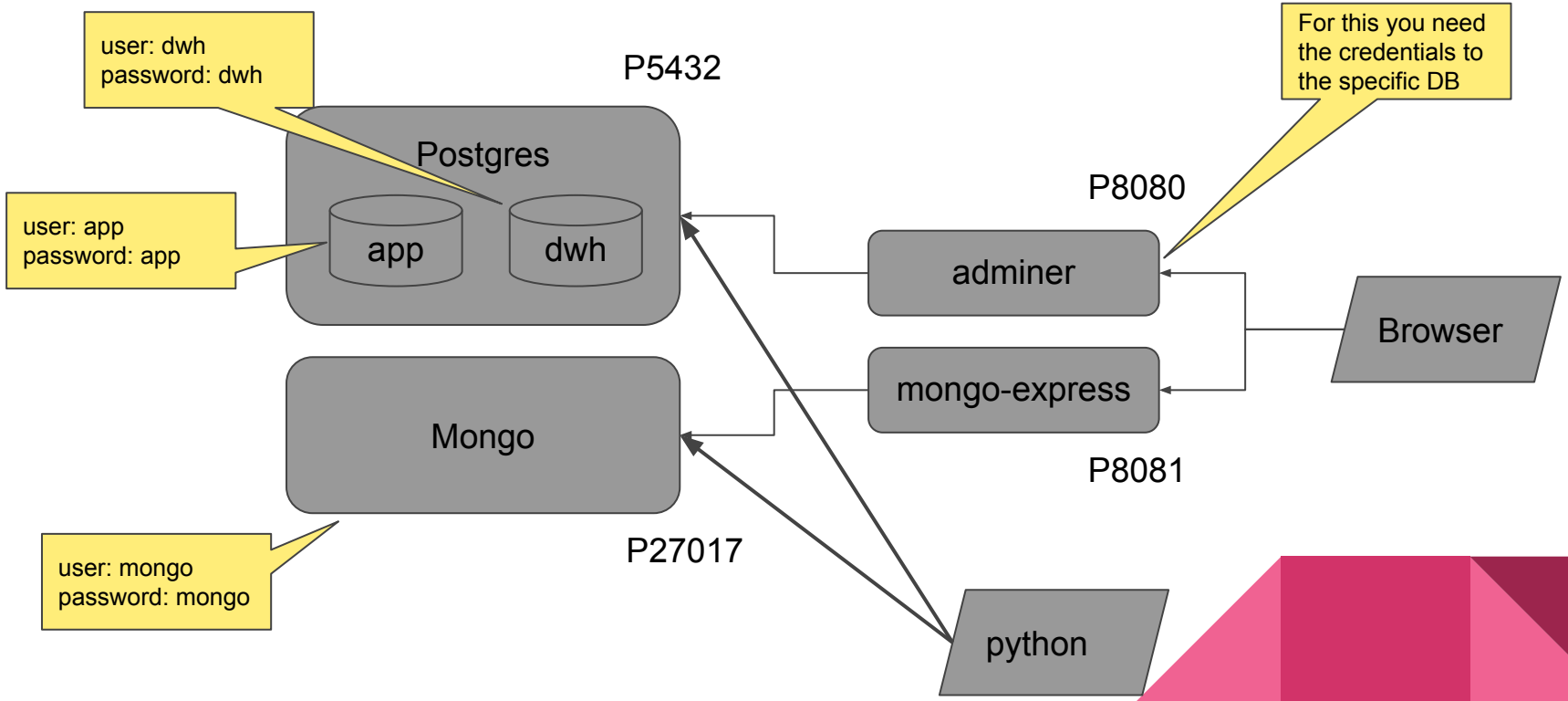
3e Geef aan of er attributen voorkomen die ook functioneel afhankelijk zijn van niet-sleutelattributen. Vorm een aparte groep voor de aangewezen attributen en het attribuut waarvan zij functioneel afhankelijk zijn. Verwijder alleen de aangewezen attributen uit de oorspronkelijke groep.

Pre-flight check

- Downloaded the git repo
- Created the python environment
- Started the compose environment

```
docker-compose up
```

Docker-compose environment



Assignment 2 - Deploy

- Part 1. Deploy the created schema.
  - (easy) Via the gui at http://localhost:8081
  - (better) Via an Alembic migration
- Part 2. Load the data from data.sql

For the assignment you need:

- docker-compose.yml
- the files:
  - assignment_2/schema.sql
  - assignment_2/alembic/

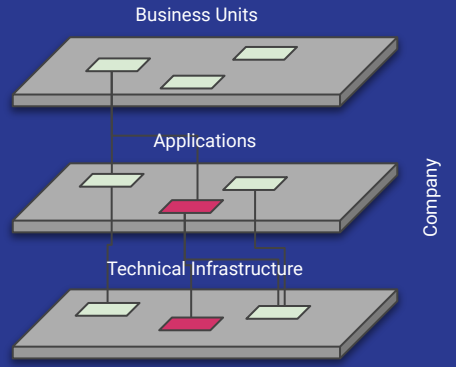
Assignment 3 - Load the data

- Struggle with the CSV and regex
- Via an Alembic migration
- Via ETL bonobo

For the assignment you need:

- docker-compose.yml
- the files:
  - data/
  - assignment_3/bonobo/
  - assignment_3/alembic/

Application Point of View NoSQL



MongoDB

- Document database with a flexible schema based on json
- Javascript as query language.
- No traditional joins
- Features
  - High-performance
  - Replication and failover
  - Auto sharding
  - Map / Reduce aggregations

RDBMS	**MongoDB**
Database	Database
Table	Collection
Row	Document
Column	Field
Index	Index
Join	Embedded Document or lookup reference
Foreign Key	Reference
Partition Key	Shard Key

MongoDB

SCH701		**REGISTRATION**			14FEB21
STUDENT DETAILS					
STUDENT NR	**NAME**	Derksen Th.			
1234	**ADDRESS**	Keistraat 31			
	CITY	Amersfoort			
COURSE NR	**COURSE NAME**	**START DATE**	**BUILDING NR**	**BUILDING NAME**	**LOCATION**
SO	syteemontwerp	12MEI21	2	paviljoen	Utrecht
SA	systeemanalyse	20JUN21	1	toren	Utrecht
PR	programmeren	11AUG21	7	bastille	Enschede
IA	informatie analyse	16OKT21	4	Vrijhof	Zwolle

```
{
        studentnr: "1234"
        name: "Derksen Th."
        address: "Keistraat 31"
        city: "Amersfoort"
        courses: [
                {
                        course_code: "SO"
                        course_name: "systeemontwerp"
                        start_date: "12MEI2021"
                        building_nr: "2"
                        building_name: "paviljoen"
                        location: "Utrecht"
                }
        ]
}
```

Mongo Normalization

There is no formal process for normalization within the MongoDB community. The choice is between embedding and referencing.

```
{
        studentnr: "1234"
        name: "Derksen Th."
        address: "Keistraat 31"
        city: "Amersfoort"
        courses: [
                {
                        course_code: "SO"
                        course_name: "systeemontwerp"
                        start_date: "12MEI2021"
                        building_nr: "2"
                        building_name: "paviljoen"
                        location: "Utrecht"
                }
        ]
}
```

```
{
        _id: ObjectID('AAAA')
        course_code: "SO"
        course_name:
        "systeemontwerp"
        start_date: "12MEI2021"
        building_nr: "2"
        building_name:
        "paviljoen"
        location: "Utrecht"
}
```

```
{
        studentnr: "1234"
        name: "Derksen Th."
        address: "Keistraat 31"
        city: "Amersfoort"
        courses: [
                ObjectID('AAAA')
        ]
}
```

- Embedded object are prefered
- Intended purpose / usage
  - Read only as in a analytical database?
  - Do the sub-entities need to stand alone?
  - Are the sub-entities referenced by more entities?
- Cardinality
  - One-to-one or one-to-many or one-to-millions?
  - Mongo objects are limited to 16MB, will this fit?

Assignment - ‘Normalize’ noSQL data

Webshop Suppliers		
SupplierNr	**Name**	books’r’us
1	**Adres**	Leidsche 12
	City	Haarlem
ArticleNr	**Name**	**Price**
50001	Dune	10.00
50002	The Martian	8.00
50003	zen and the art of motorcycle maintenance	12.00

‘Normalize’ by using a notepad and create a suitable json structure to capture this data.

Assignment - Deploy

- Part 1. Load the data from data.json
  - (easy) Via the gui at http://localhost:8081
  - (better) Via an bonobo transform in load.py

For the assignment you need:

- docker-compose.yml
- the files:
  - assignment_2/data.json
  - assignment_2/load.py

Case study

Covariantie matrix
Deze matrix is een sparse driehoek matrix. Waarin de covarianties tussen twee punten worden opgeslagen als een 32bit real.

Deze is sparse omdat er een limiet wordt gesteld aan de correlatie afstand tussen de verschillend punten, de correlatie afstand zal verschillen tussen verschillende gebieden aan de hand van een latere analyse.

Het is een driehoek matrix omdat de correlaties gelijkwaardig zijn tussen de verschillende punten.

De Punten zouden geïdentificeerd kunnen worden doormiddel van BRK-Ids. Het formaat hiervan wordt ook geschat op een 32bit.

Geschat aantal punten: 80 000 000
Oppervlakte Nederland: 41 543 km2
Correlatie afstand: tussen 100 en 1 000 meter

	Punt A	Punt B	Punt C	Punt D
Punt A				
Punt B				
Punt C				
Punt D				

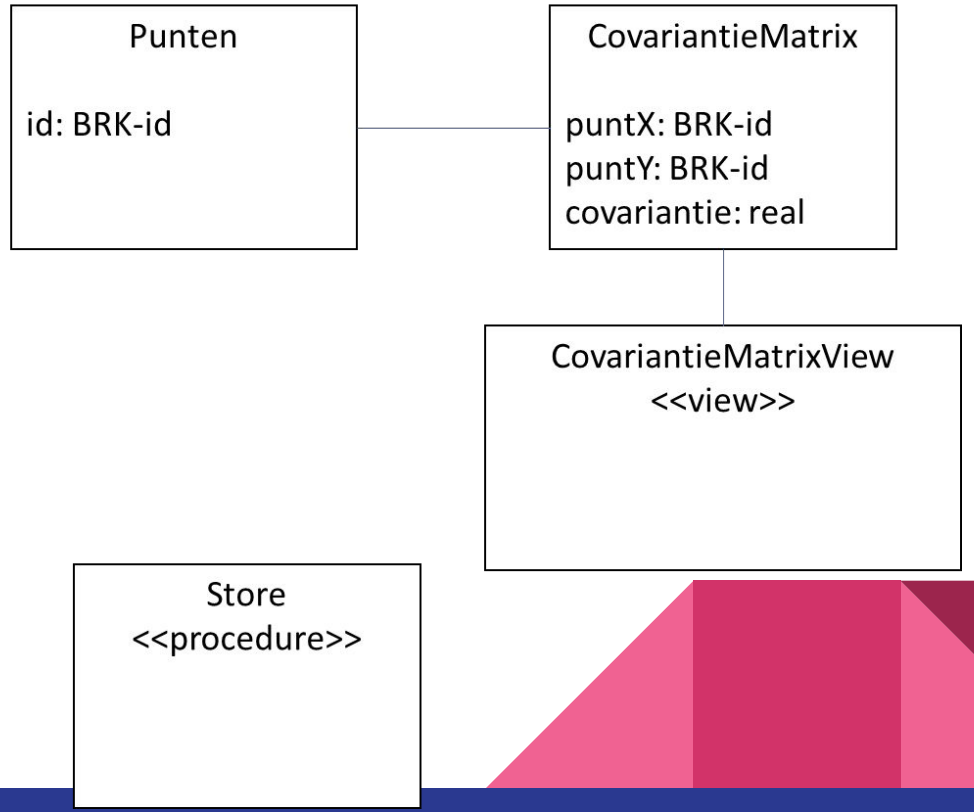
Als wij de correlatie afstand tussen 100 en 1000 afschatten op 500 meter kunnen wij punten per km2 gebruiken als aantal correlaties per punt.

80M punten / 41 543 km^2 = 1925,72 punten per km2 (2000)
80M punten * 2 000 relaties / 2 - 80M punten = 79 920M
32bit covariantie waarde + 2 * 32bit BRK-Ids = 96bit

79 920M * 96bit = 7 672 320M bit = 959GB

Postgresql

- Matrix opslaan als een set van relaties.
- Toepassen van een view om het bevragen van de relaties bidirectioneel te laten werken. Elke select zal 2x plaats moeten vinden
- HashIndex mogelijk beter dan Btree index voor de punten. Bevraging vindt plaats op equality. Btree is al O(log(n)) dus zou op zich snel moeten zijn.
- Aparte procedure zou nodig zijn om te waarborgen dat relaties niet dubbel worden opgeslagen.
- Partitioneren, Sharden en clusteren is wellicht een uitdaging gezien de natuur van de data. 1TB niet gigantisch mochten er performance issues ontstaan dan zijn read-slaves mogelijk al effectief genoeg.



MongoDB

- Structuur vergelijkbaar met postgres

- Optie B scheelt iets in het opslag volume. De vraag is alleen of de indexering binnen de array goed ondersteund word.

CovariantieMatrix optie A

```
{
  puntX: BRK-Id
  puntY: BRK-Id
  covariantie: real
}
```

CovariantieMatrix optie B

```
{
  id: BRK-Id
  covarianties: [
    {
      punt: BRK-Id
      covariantie: real
    }, …
  ]
}
```

Key-Value store

- Te implementeren met MongoDB. Maar ook met meer dedicated Key-Value stores

- Voordeel is dat deze erg snel kunnen zijn en eenvoudig te parallelliseren.

- Nadeel van optie A is fundamenteel. Relatie valt niet terug te leggen.

- Optie B mogelijk als apart alle punten in de correlatie afstand op te halen zijn. Vervolgens deze allemaal opvragen. Onwaarschijnlijk dat dit efficiënt is.

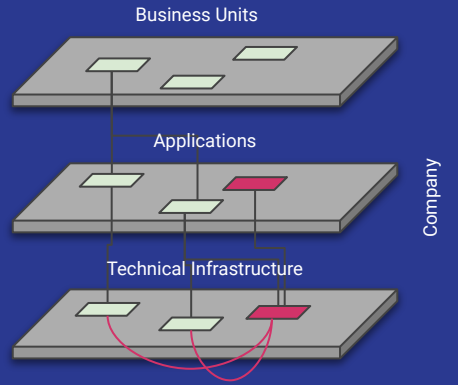
CovariantieMatrix
Optie A

```
key: BRK-Id
value: [
  {
    puntX: puntY: BRK-Id
    covariantie: real
  },...
]
```

CovariantieMatrix
Optie B

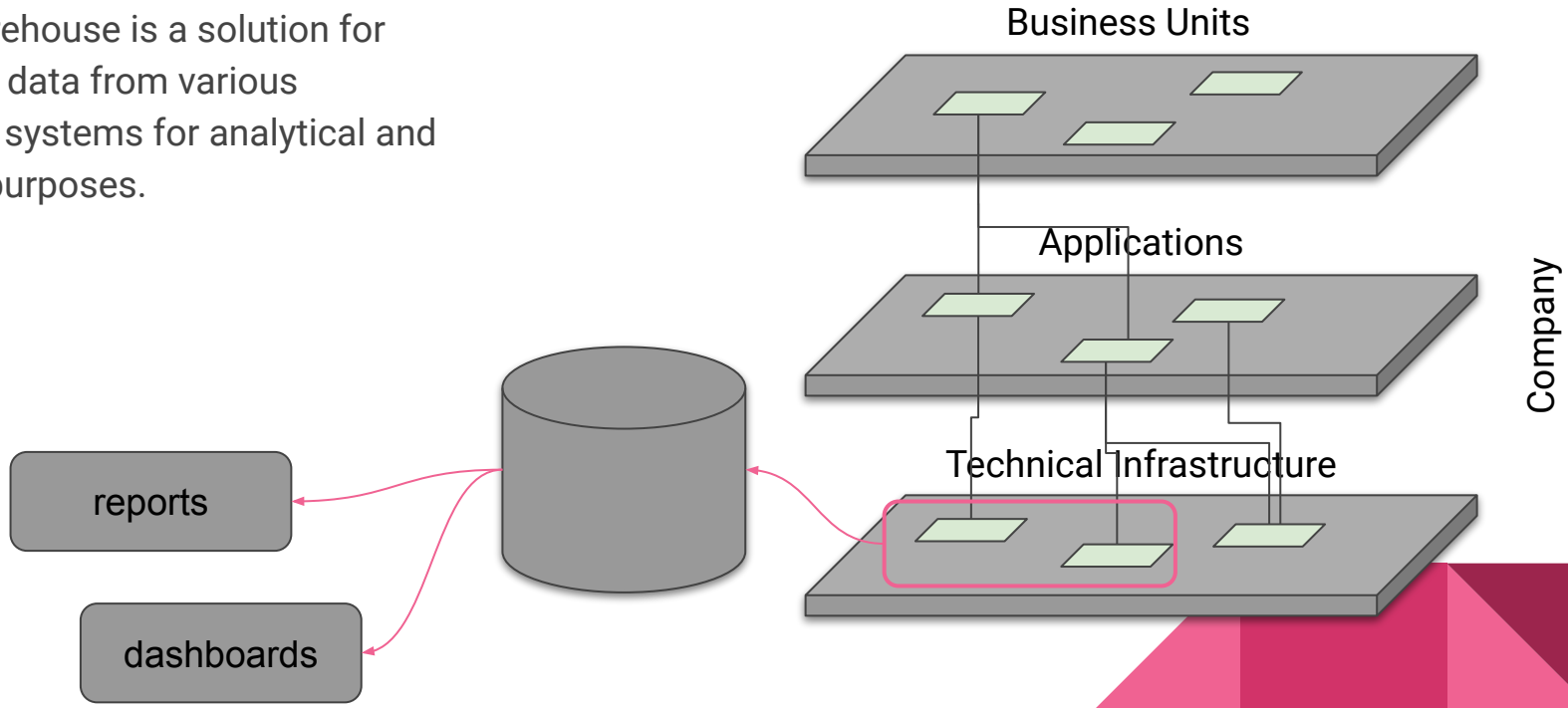
```
key: BRK-Id - BRK-Id
value: real
```

Warehousing Point of View



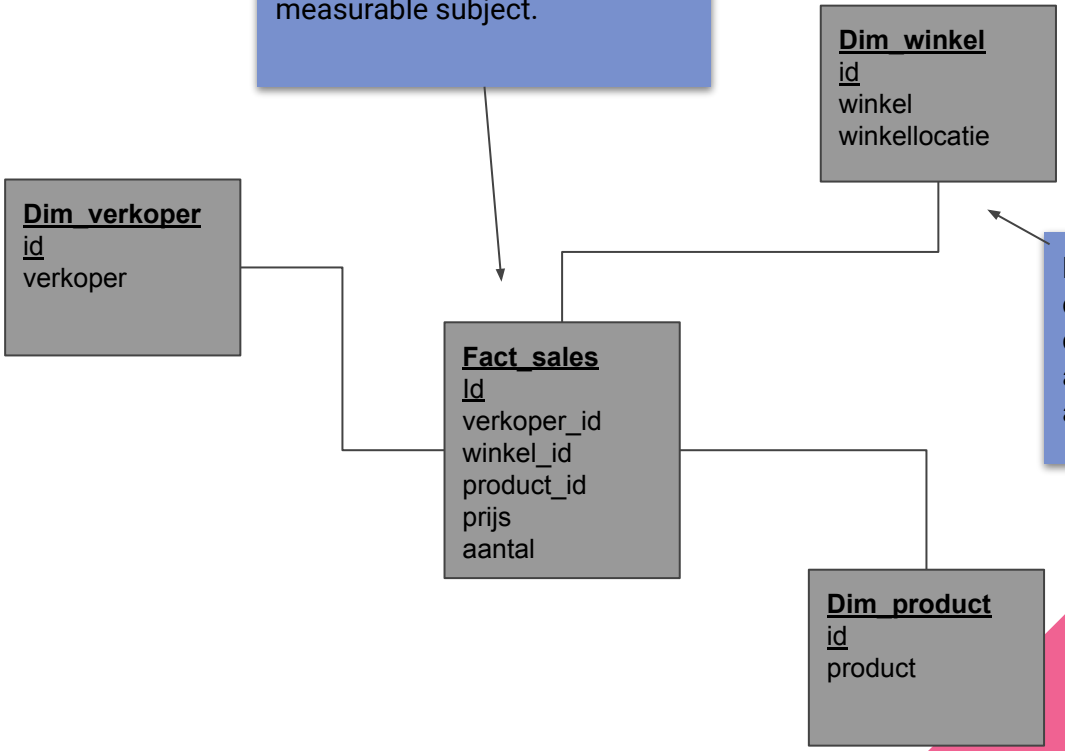
Data Warehouse

A data warehouse is a solution for combining data from various underlying systems for analytical and reporting purposes.



Star-Schema

The core consists of a fact table containing the prime measurable subject.



Fact tables are surrounded by dimension tables. Usually dimension tables are smaller and enrich the fact with additional information.

Normaliseren van een star-schema

Product	Verkoper	Winkel	Winkel locatie	Prijs	Aantal
A	Henk	Hoofd	Dreef 12	11	5
B	Henk	Hoofd	Dreef 12	200	1
C	Frits	Branch	Dorpstraat 2	13	2

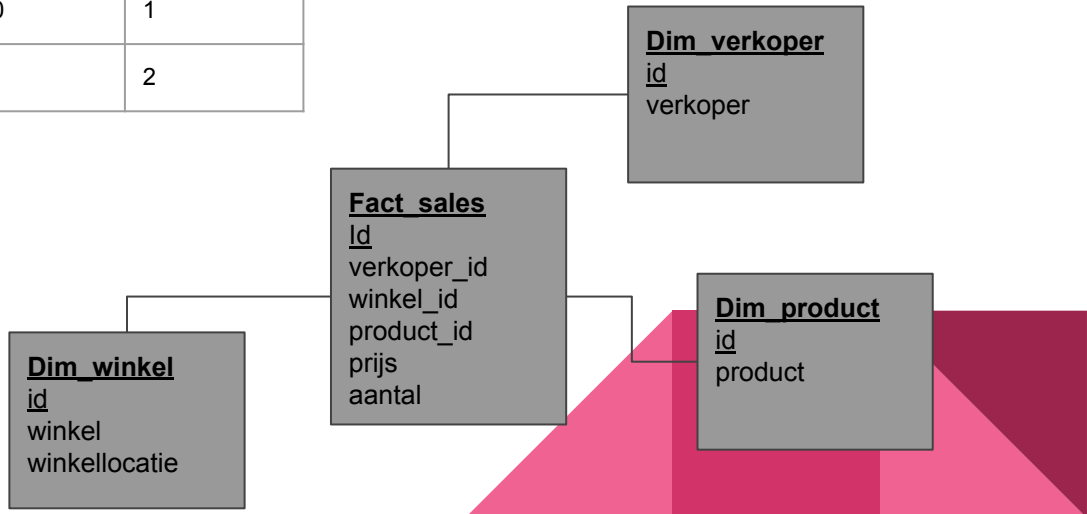
Plaats alle sommeerbare getallen (measures) in een centrale fact tabel.

Fact_sales
id
prijs
aantal

Normaliseren van een star-schema

Product	Verkoper	Winkel	Winkel locatie	Prijs	Aantal
A	Henk	Hoofd	Dreef 12	11	5
B	Henk	Hoofd	Dreef 12	200	1
C	Frits	Branch	Dorpstraat 2	13	2

Creëer een dimensie voor alle deel verzamelingen en voeg een referentie toe in de fact table.



Assignment - Normalize for a data warehouse

Publisher Webshop					
Client information					
CustomerNr	**Name**	Derksen Th.			
1234	**Adres**	Keistraat 31			
	City	Amersfoort			
ArticleNr	**Name**	**Price**	**Amount**	**Warehouse**	**Shelf**
50001	Dune	25.95	2	2	12
50002	The Martian	14.00	1	1	14
50003	zen and the art of motorcycle maintenance	15.95	1	7	13

Start with your own normalized versions of the source data.

Webshop Suppliers		
SupplierNr	**Name**	books'r'us
1	**Adres**	Leidsche 12
	City	Haarlem
ArticleNr	**Name**	**Price**
50001	Dune	10.00
50002	The Martian	8.00
50003	zen and the art of motorcycle maintenance	12.00

Use: https://dbdiagram.io/

The objective is to create a viable star schema to encompass the data from both systems and allow the creation of a report to calculate the profit margin for the various books.

Extract Transform Load (ETL)

ETL vendors

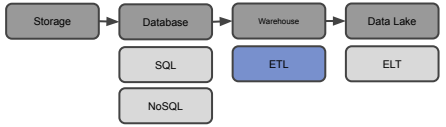
Bonobo

Pushdown processing

Flow processing

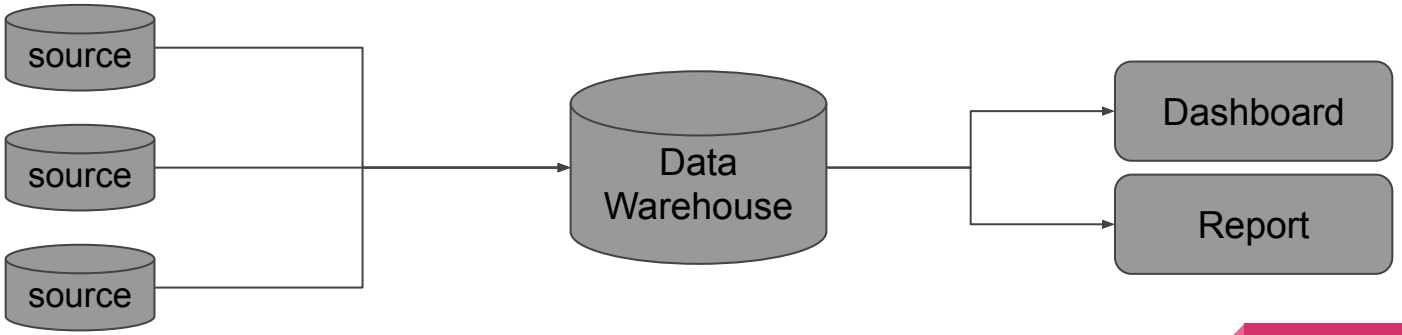
Workflow schedulers

Extract Transform Load

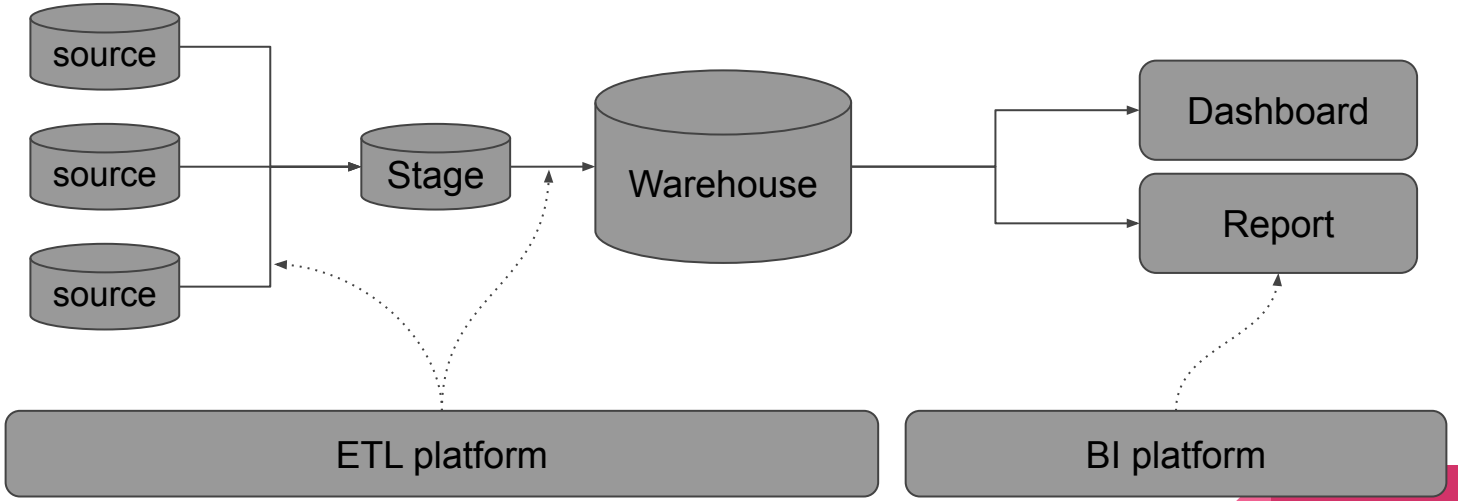


ETL is the process of extracting data from a source system, transforming it to a different format and loading it in a target system.

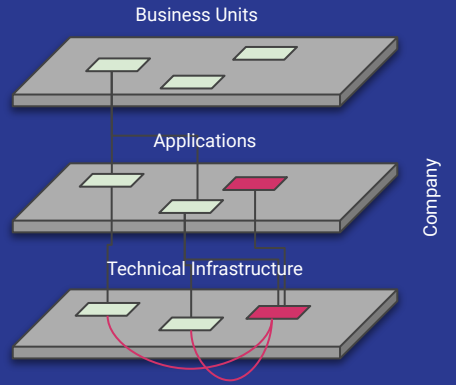
This process is the common way of loading data in a data warehouse.



Extract Transform Load



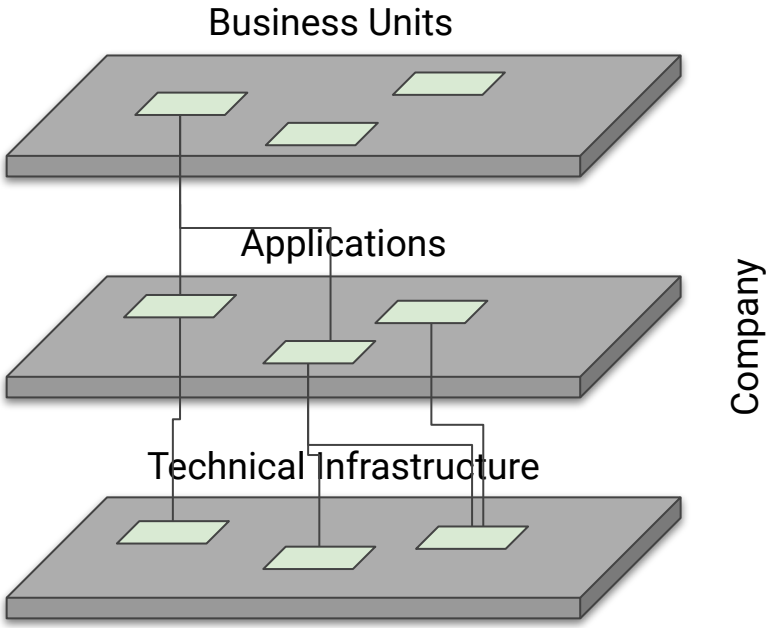
Data Lake Point of View



Data Warehouse and Data Lake

Data Warehouse:
Large Structured RDBMS combining data from multiple source systems.

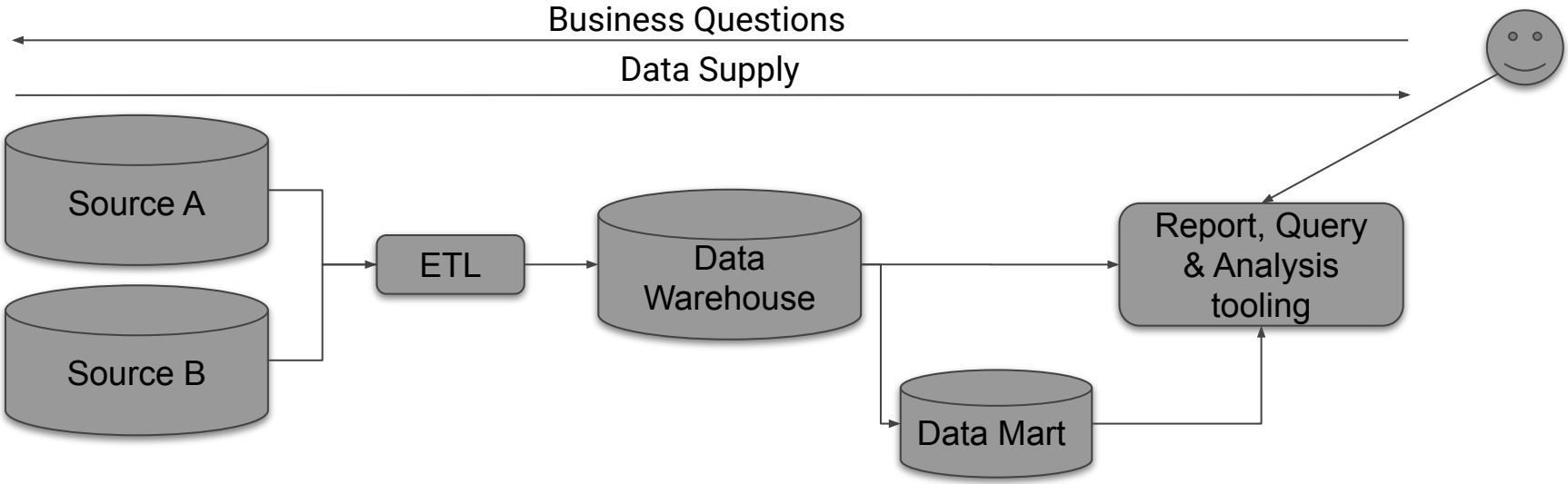
Design is driven by a direct business need and requires a clear purpose and a rigid structure.



Data Lake
Massive storage solution for structured and unstructured data.

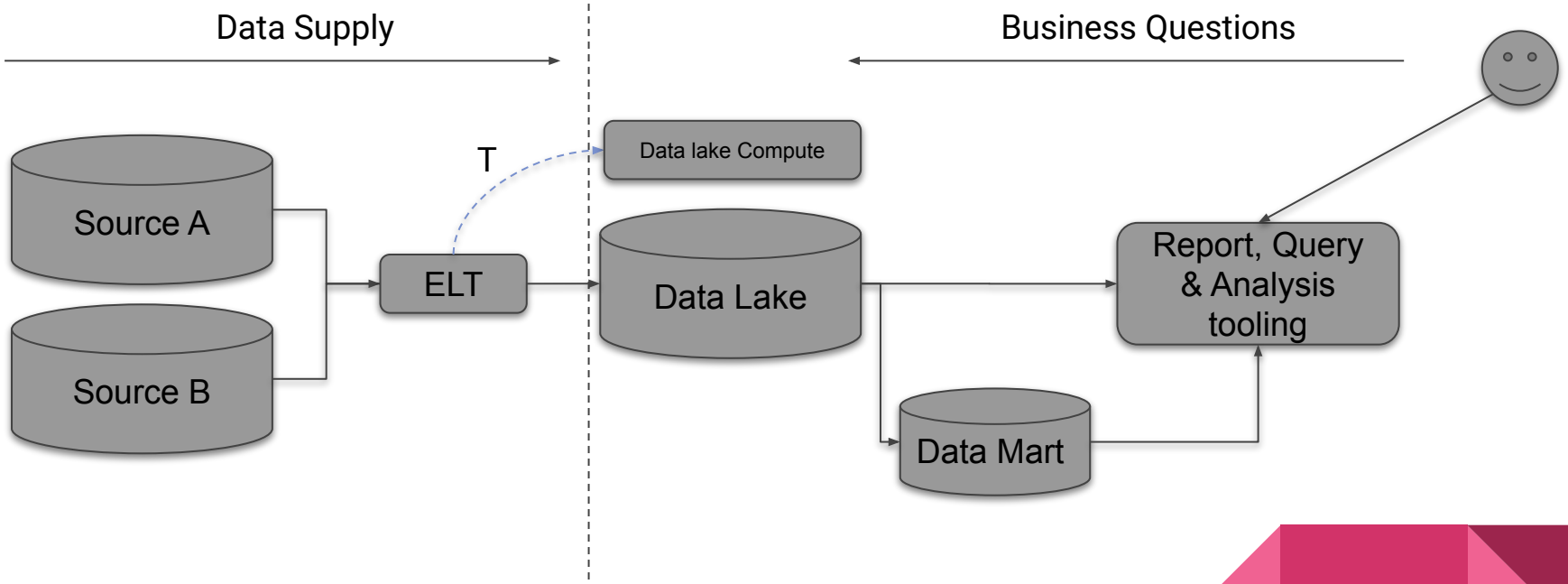
Intended to capture the full source data so the integration towards the source system has to be done only once

Data Warehouse



Extract **T**ransform **L**oad

Data Lake



Extract **L**oad **T**ransform

Data lake layers

Raw

Duplicates of source systems. The data formats can be varied and difficult to interpret.

Can still contain sensitive data or items which are ambiguous to interpret.

Curated

Cleaned versions of the source data. Data in here should be homogenized and well described.

Ideal place for data scientists to tap into. Reliable, understandable data which is still closely related to the source system

Product

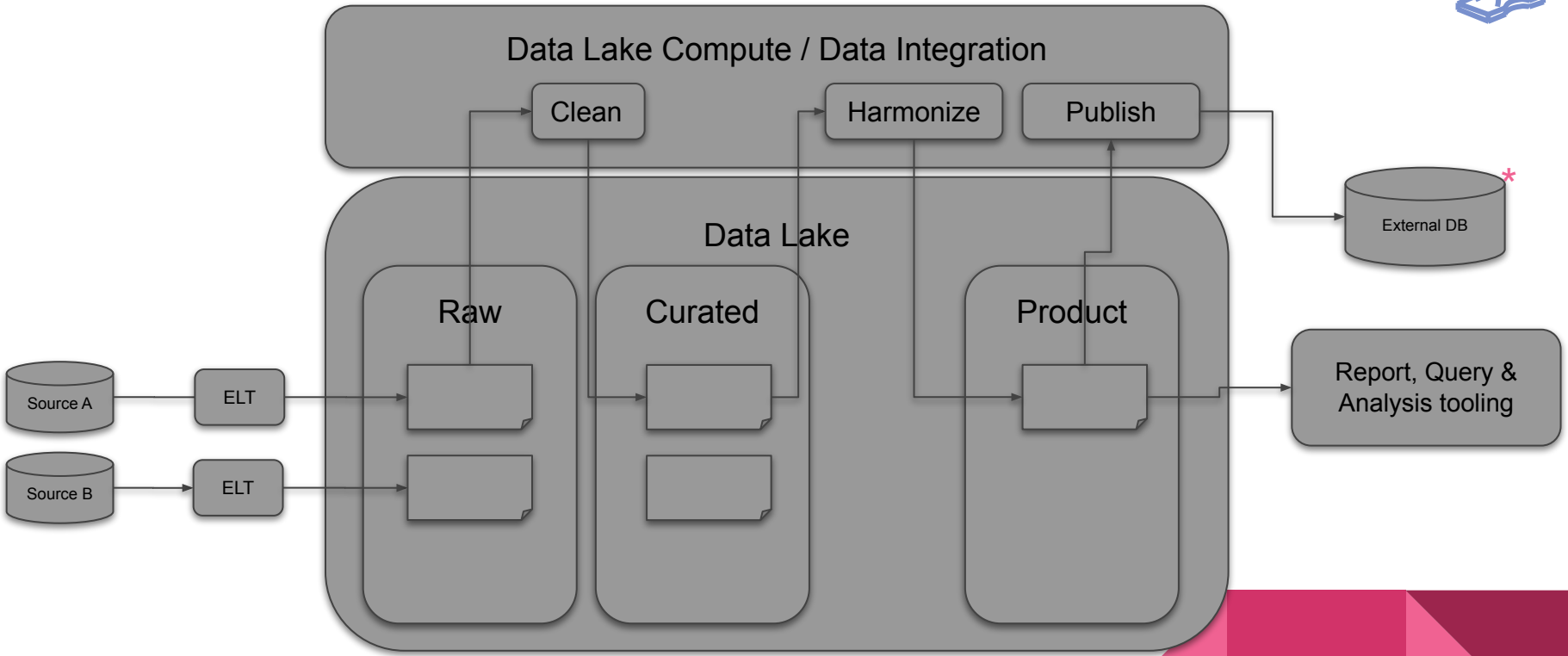
Final layer used for integrating data sets into other systems or tools.

Usually this is tailored to a specific use reminiscent of the structure in a Data Warehouse.

There is a plethora of variations to this staging, every company runs their own. The basics are very comparable though.

Examples:

- Raw, Cooked, Presentation
- Grey, Brown, Blue, Green



Reference to Part 1

Retrieving semi-structured data

CSV / TSV

- In my experience this is most common as your first dataset.

```
John,Doe,120 jefferson st.,Riverside, NJ, 08075
Jack,McGinnis,220 hobo Av.,Phila, PA,09119
John "Da Man",Repici,120 Jefferson St.,Riverside, NJ,08075
Stephen,Tyler,"7452 Terrace ""At the Plaza""
road",SomeTown,SD, 91234
,Blankman,,SomeTown, SD, 00298
Joan "the bone", Anne,Jet,"9th, at Terrace plc",Desert
City,CO,00123
```

Obvious things to look out for:

- Header row
- Delimiter
- Quote Character
- Line ending (most parsers deal with this)
- Character Encoding
- Size (plain text is very inefficient)

XML

- This is more difficult to work with since it needs to be parsed and complex structures can be created.

```
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Obvious things to look out for:

- Parse and convert to tabular can be very helpful
- DOM Parser: Parse the entire document into a hierarchical tree. Memory intensive!
- SAX Parser: Parse element tag-by-tag. More difficult.
- XPath Parser: Query specific parts of the document

Log files (or any other append file)

- Quite common

```
111.222.333.123 HOME - [01/Feb/1998:01:08:39 -0800] "GET /bannerad/ad.htm
HTTP/1.0" 200 198 "http://www.referrer.com/bannerad/ba_intro.htm" "Mozilla/4.01
(Macintosh; I; PPC)"
111.222.333.123 HOME - [01/Feb/1998:01:08:46 -0800] "GET /bannerad/ad.htm
HTTP/1.0" 200 28083 "http://www.referrer.com/bannerad/ba_intro.htm" "Mozilla/4.01
(Macintosh; I; PPC)"
111.222.333.123 AWAY - [01/Feb/1998:01:08:53 -0800] "GET /bannerad/ad7.gif
HTTP/1.0" 200 9332 "http://www.referrer.com/bannerad/ba_ad.htm" "Mozilla/4.01
(Macintosh; I; PPC)"
```

Obvious things to look out for:

- Ask for the specification, this will save you time.
- Parsing lines via Regex is a good solution
- Use a regex interface: https://regex101.com/

Regex - Example

```
111.222.333.123 HOME - [01/Feb/1998:01:08:39 -0800] "GET /bannerad/ad.htm HTTP/1.0" 200 198
"http://www.referrer.com/bannerad/ba_intro.htm" "Mozilla/4.01 (Macintosh; I; PPC)"
```

Regex - Example

Regex	Description	Example	Match
.	Match any character	.	Lorum Ipsum
\	Escape the next char	\. \.	Lorum Ipsum ...
\d	Match a number	\d	123
*	Match 0 or more of the previous match (greedy)	\d* .*	123 Lorum Ipsum 123 Lorum Ipsum
+	Match 1 or more of the previous match (greedy)	\d+ .+	123 Lorum Ipsum 123 Lorum Ipsum
()	Make a capture group	Lorum (.+)	123 Lorum Ipsum
[A-Z]	Character class	[A-Z]* [A-Za-z]*	123 Lorum Ipsum 123 Lorum Ipsum

Regex - Example

111.222.333.123 HOME - [01/Feb/1998:01:08:39 -0800] "GET /bannerad/ad.htm HTTP/1.0" 200 198 "http://www.referrer.com/bannerad/ba_intro.htm" "Mozilla/4.01 (Macintosh; I; PPC)"

(\d+\.\d+\.\d+\.\d+) ([A-Z]+) - (\[.*\]) "(.*)" (\d+) (\d+) "(.*)" "(.*)"

Parenthesis: a capture group

\. exactly one dot
\d any digit
+ one or more

Character class of capital letters

. any character
* zero or more

Assignment 5

We are looking at: http://www.almhuette-raith.at/apache-log/access.log

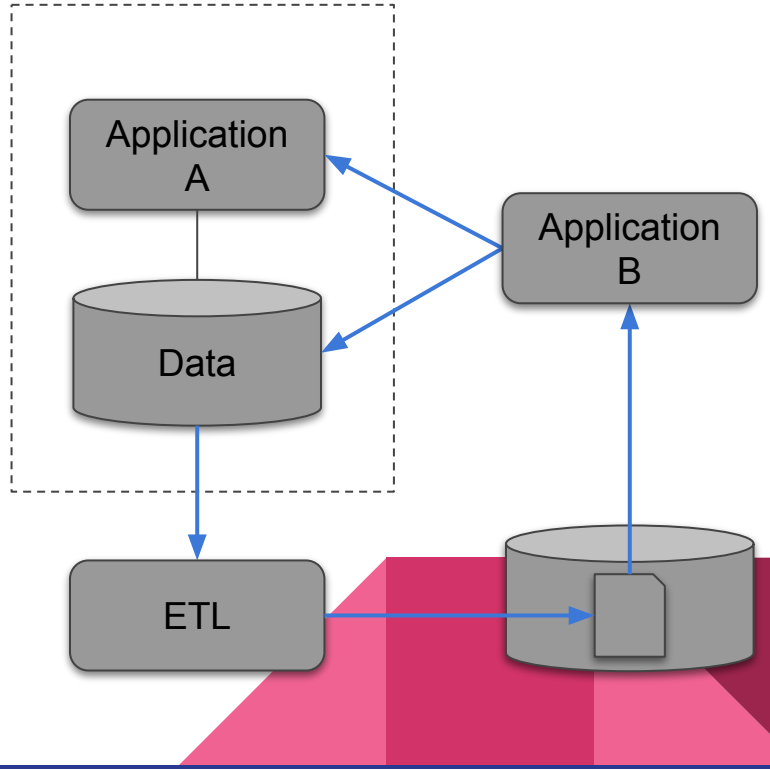
This is a site that is very likely to have been hacked and the poor admins have left the access log downloadable. Let try and figure out what is going on here!

- Files:
  - data/log.txt (big file)
  - data/log-0-100.txt (first 100 lines of log.txt)
  - assignment_5/bonobo
- Use regex101 to build a regex
- Parse the log file with python
- Persist

Part 3 - Rest APIs

Many ways you can get data

- Just a file*
- Database Connection
- Custom protocol
- Common protocol
  - SOAP
  - PUB/SUB
  - REST APIs



Reference to Part 2

RESTful APIs

It’s in its core nothing more than http traffic. Read: https://tools.ietf.org/html/rfc2616. If you work with web traffic or REST this is the core of what you are doing.

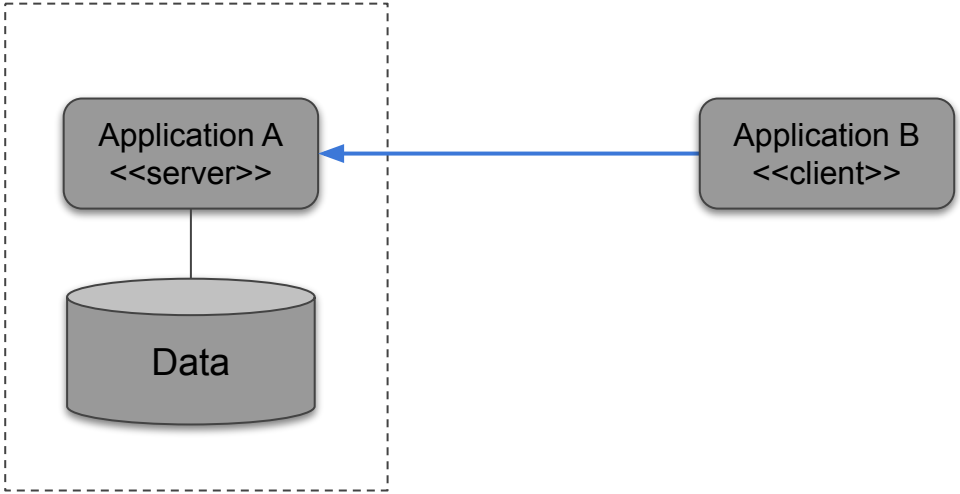
HTTP methods	**Resources that manipulate data, such as https://api.example.com/collection or https://api.example.com/collection/item3**	**Resources that invoke operations, such as https://api.example.com/clusters/1234/create-vm**
POST		Invoke the operation provided by the resource. If the request results in the creation of a new resource, its URI is returned in the response Location header.
GET	Retrieve a representation of the data in the response body.	Retrieve the status of an asynchronous operation in the response body.
PUT	Store the representation in the request body as the (new) state of the resource. Subsequent GET access on the resource is expected to return this representation with HTTP status 200 until PATCHed, a different representation is PUT, or the state of the resource is DELETEd.	
PATCH	Update some part of the resource's state using the instructions in the request body.	
DELETE	Delete the state of the resource. Subsequent GET access on the resource is expected to return HTTP status 404 until new state is PUT to it.	Cancel an asynchronous operation.

Architectural constraints of RESTful system

- Client-server architecture
- Statelessness
- Cacheability
- Layered system
- Code on demand (optional)
- Uniform interface
  - Resource identification in requests
  - Resource manipulation through representations
  - Self-descriptive messages
  - Hypermedia as the engine of application state

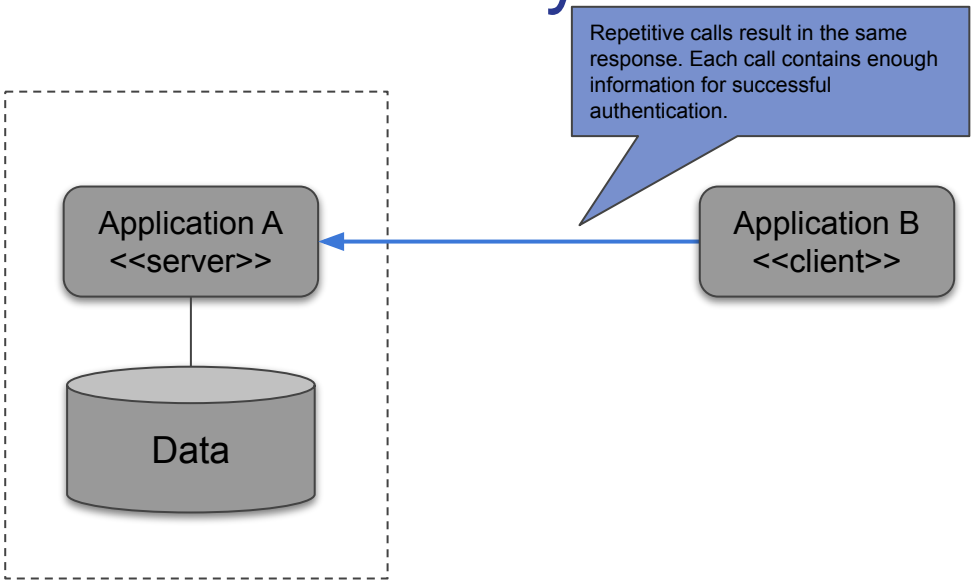
Architectural constraints of RESTful system

- Client-server architecture



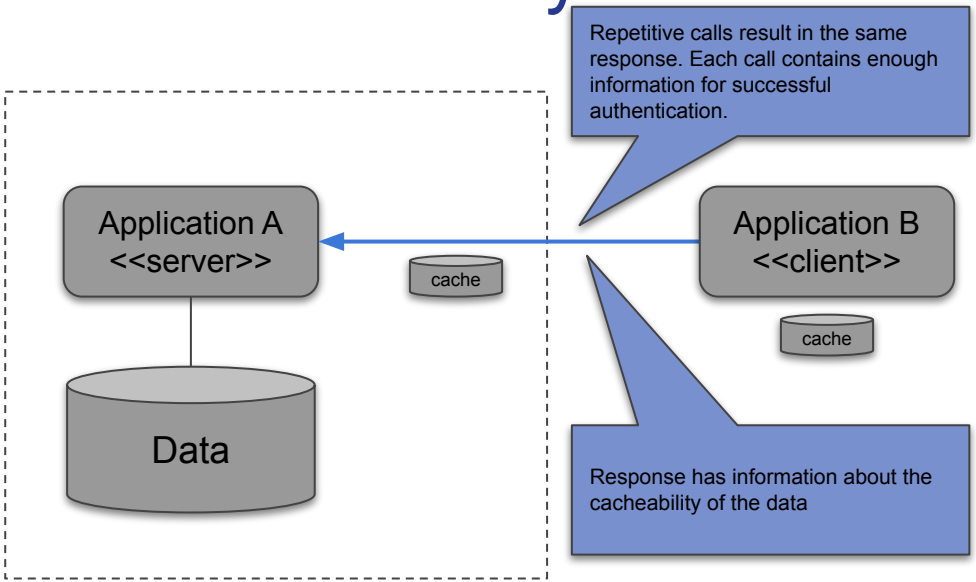
Architectural constraints of RESTful system

- Client-server architecture
- Statelessness



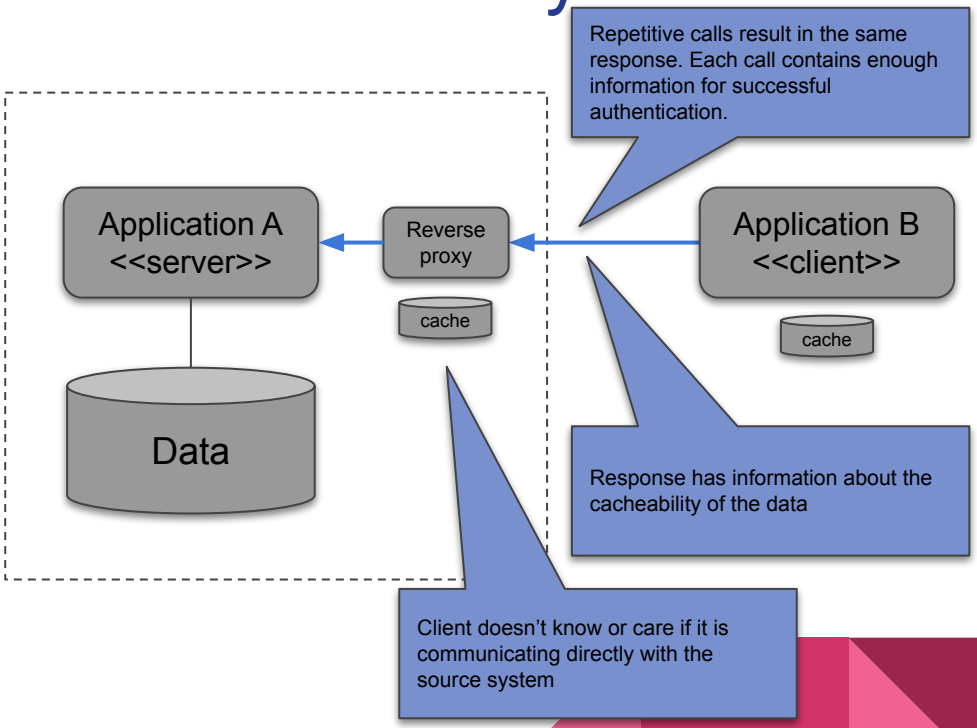
Architectural constraints of RESTful system

- Client-server architecture
- Statelessness
- Cacheability



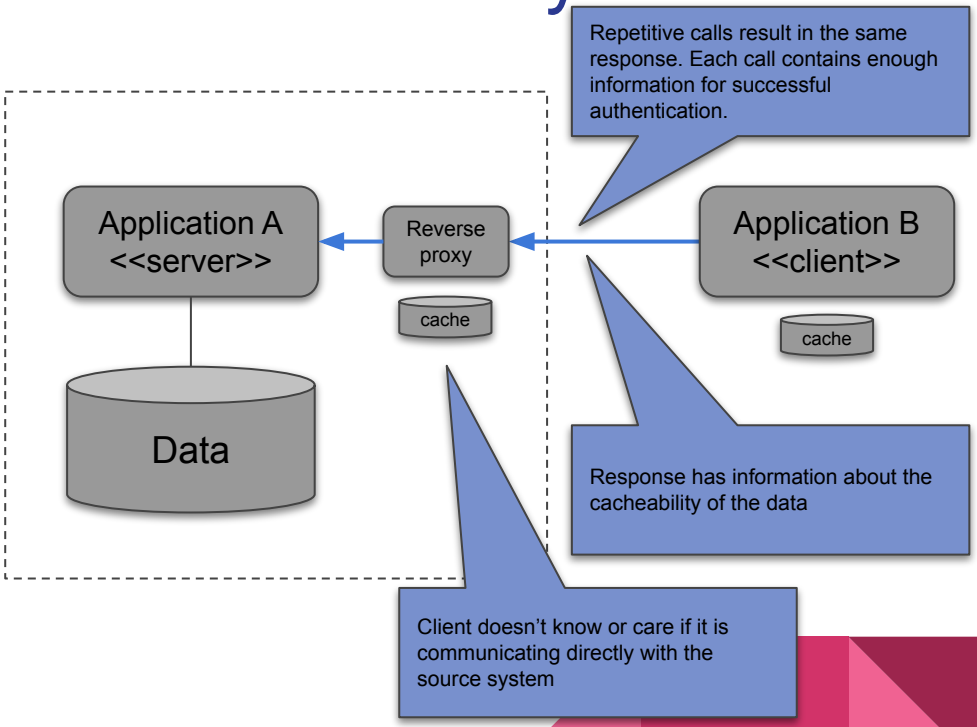
Architectural constraints of RESTful system

- Client-server architecture
- Statelessness
- Cacheability
- Layered system



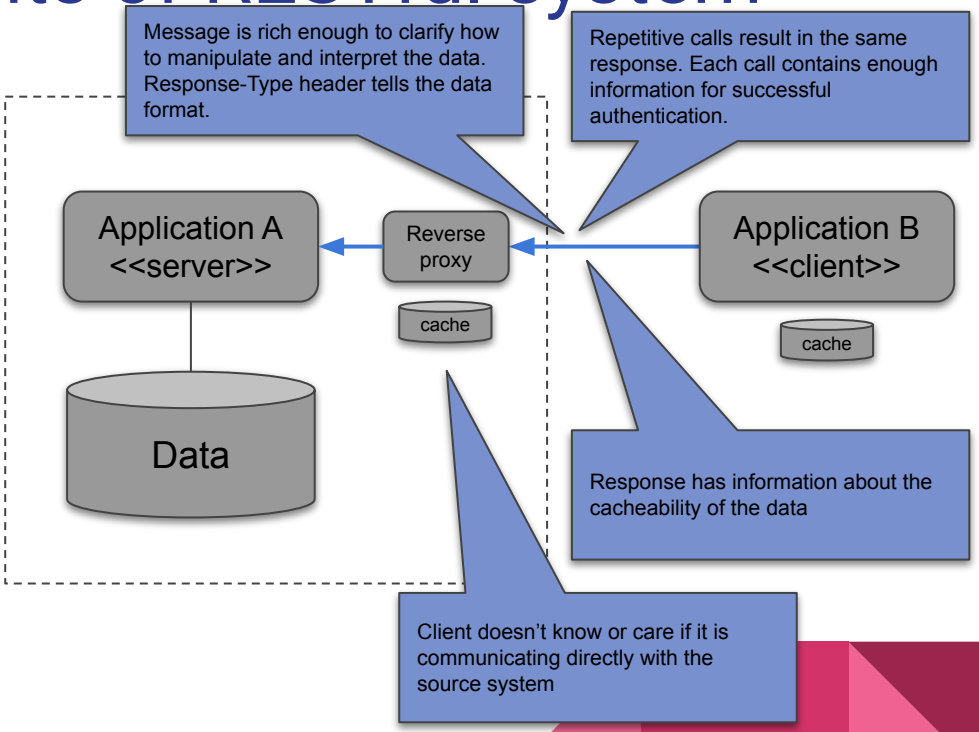
Architectural constraints of RESTful system

- Client-server architecture
- Statelessness
- Cacheability
- Layered system
- ~~Code on demand (optional)~~



Architectural constraints of RESTful system

- Client-server architecture
- Statelessness
- Cacheability
- Layered system
- ~~Code on demand (optional)~~
- Uniform interface
  - Resource identification in requests
  - Resource manipulation through representations
  - Self-descriptive messages
  - ~~Hypermedia as the engine of application state~~



Example

https://en.wikipedia.org/api/rest_v1/#/

https://ipwhois.io/documentation

Python - pandas

```
>>> import requests
>>> import json
>>> import pandas as pd
>>> r = requests.get('http://ipwhois.app/json/8.8.8.8')
>>> j = r.json()
```

http://ipwhois.app/json/8.8.8.8

Assignment 6

Tell me some things:

- Where do the requests from the apache log come from?
- Can you find information on the organisations behind them?

Try and limit your requests!

Curriculum

- https://tools.ietf.org/html/rfc2616 read this at least once in your life.
- https://regexone.com/ good tutorial on regex
- Learn Docker and Kubernetes. The coming years these will become even more pervasive.
- Write code. No lecture will be better than you getting your hands dirty.

The end.

Hortonwork Data Platform

Hortonworks

GOVERNANCE & INTEGRATION

Data Workflow, Lifecycle & Governance

Falcon
Sqoop
Flume
NFS
webHDFS

DATA ACCESS

Batch	Script	SQL	NoSQL	Stream	Others
Map Reduce	Pig	Hive/Tez HCatalog	Hbase Accumulo	Storm	In-Memory Analytics ISV Engines

YARN : Data Operating System

DATA MANAGEMENT

HDFS
(Hadoop Distributed File System)

SECURITY

Authentication
Authorization
Accounting
Data Protection

Storage: HDFS
Resources: YARN
Access: Hive, ...
Pipeline: Falcon
Cluster: Knox

OPERATIONS

Provision, Manage & Monitor

Ambari
Zookeeper

Scheduling

Oozie

Normalisation

	UNF (1970)	1NF (1970)	2NF (1971)	3NF (1971)	EKNF (1982)	BCNF (1974)	4NF (1977)	ETNF (2012)	5NF (1979)	DKNF (1981)	6NF (2003)
Primary key (no duplicate tuples)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
No repeating groups	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Atomic columns (cells have single value)[8]	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either does not begin with a proper subset of a candidate key or ends with a prime attribute (no partial functional dependencies of non-prime attributes on candidate keys)[8]	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency begins with a superkey or ends with a prime attribute (no transitive functional dependencies of non-prime attributes on candidate keys)[8]	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with an elementary prime attribute[8]	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	N/A
Every non-trivial functional dependency begins with a superkey[8]	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	N/A
Every non-trivial multivalued dependency begins with a superkey[8]	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	N/A
Every join dependency has a superkey component[9]	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	N/A
Every join dependency has only superkey components[8]	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	N/A
Every constraint is a consequence of domain constraints and key constraints[8]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	N/A
Every join dependency is trivial[8]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Python - getting a bit cheeky

Sometime you don’t get a nice RESTful API or a friendly IT team that will provide you with a nice data dump. At this point we’re going to get a bit cheeky and get the data ourselves.

```
import scrapy

class BlogSpider(scrapy.Spider):
    name = 'blogspider'
    start_urls = ['https://blog.scrapinghub.com']

    def parse(self, response):
        for title in response.css('.post-header>h2'):
            yield {'title': title.css('a ::text').get()}

        for next_page in response.css('a.next-posts-link'):
            yield response.follow(next_page, self.parse)
```

Important!
Sending a request is way easier than responding to one. If you must go this route be sure that you don’t overload your source and you don’t violate their robots.txt

Assignment 1.1

https://github.com/dataturgy/Harvest-Assignments/tree/master/Assignment%201.1

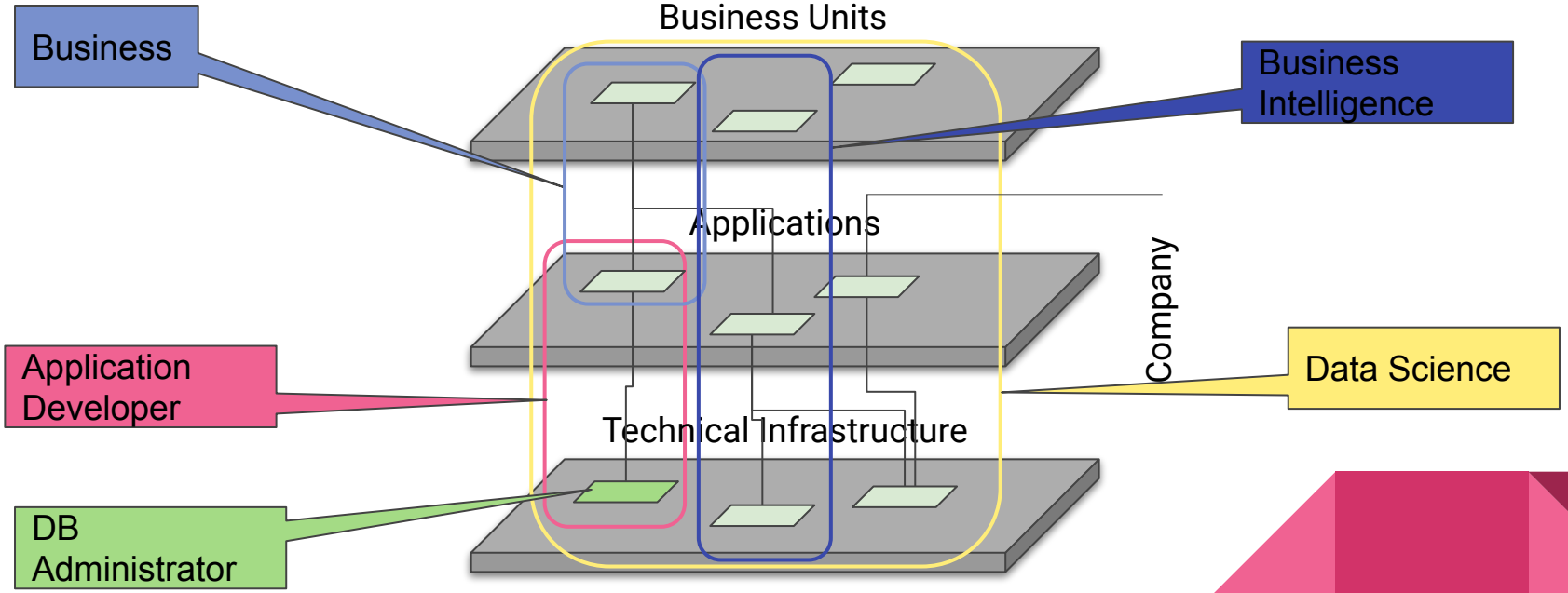
- Load the demo diagram from dbdiagram.txt in https://dbdiagram.io/
  - Try to understand what kind of application this would be
- Download as the schema as PostgreSQL
- Connect my PostgreSQL at with pgAdmin
  - url : harvest-postgres.postgres.database.azure.com
  - Username: FIRSTNAME@harvest-postgres.postgres.database.azure.com
  - Password: aP2hMxMmajuHFFGG
- Load the schema
- Download the data set from the github (Assignment 1.1/data.tar)
- Load the dataset

Assignment 1.2

Tell me some things:

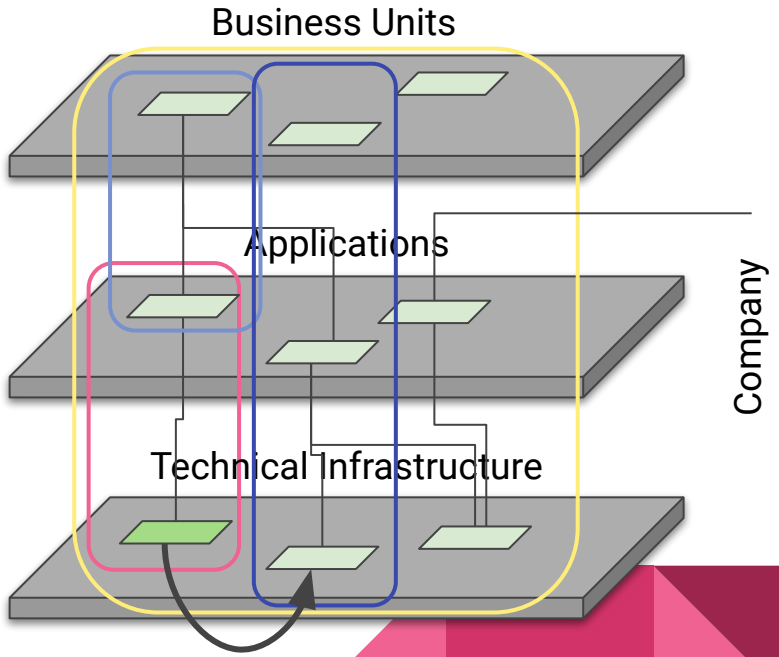
- How many users in the system?
- Which is the most popular product?
- Total actually sold by each merchant?

How does the real world look like - perspectives



Reiterate on the perspectives

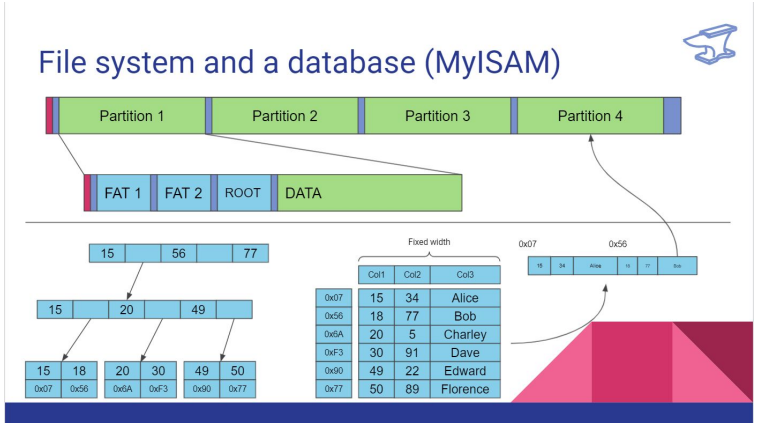
- What perspective were you using to look at the database?
- What would you be doing in different with different perspective?
- How to combine data from these different systems?

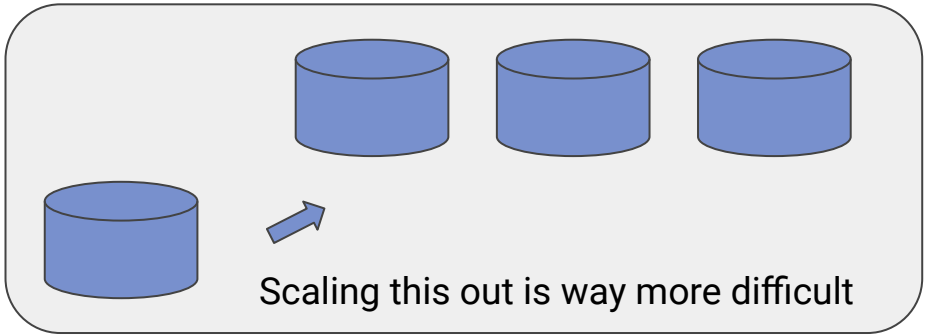


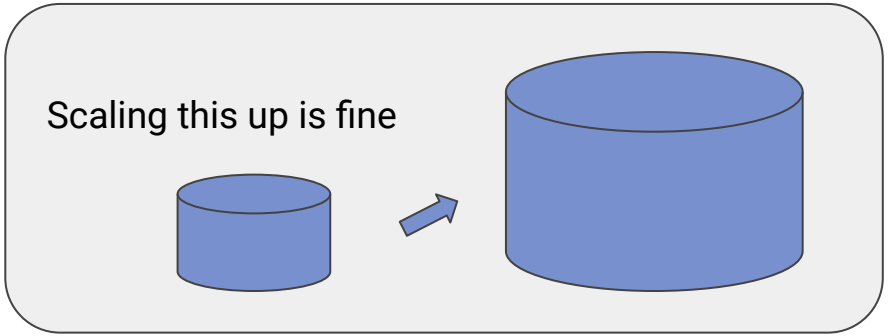
ETL

End slides

Scaling a database







CAP theorem

When scaling a database out we have to consider the CAP theorem. The CAP theorem suggests that there is a trichotomy of which only two values can be true at the same time. The statements are:

Consistency: Every read receives the most recent write or an error

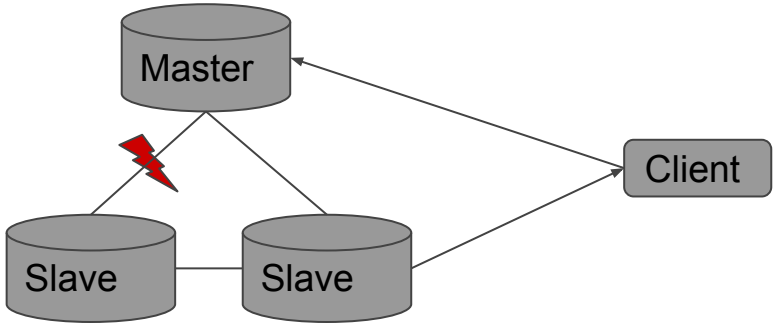
Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write.

Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes.

CAP is not the only way to look at this issue:

- PACELC
- ACID

Master/Slave read write sepparation



CP Proof
If a network partition occurs the results might be delayed or contain an error

AP Proof
If a network partition occurs the and we guarantee availability we can’t be consistent

Disambiguation needed for Data Lakes

Hadoop
The term Data Lake originated at the rise of Hadoop. With Hadoop horizontally scalable storage was combined with horizontally scalable compute power. This resulted in a swiss army knife like solution which is very powerful but very complex.

Cloud file system (Azure Data Lake, Amazon S3, ...)
The public have made their storage solutions compatible with the concept of a Data Lake. This is widely used. Keep in mind that there is a distinction between using technical component called a 'Data Lake' and implementing a 'Data Lake' architecture.

Hybrid Data Warehouse / Data Lake Solutions
Most organisations have Data Warehouses operational for years already. Don't think a DWH is an outdated concept it is still very powerful. In these cases some harmonization is sought between and the can run in tandem. Either the DWH feeds the DL or the other way around.

Data Platform
A general architecture to build multiple solutions which could contain a Data Lake. This is common terminology in more mature Organizations. This stems from the realisation that just a DL has it drawbacks, that you want to integrate it with your DWH and that your Scientists also have to work with it.

MySQL vs PostgreSQL

MySQL	PostgreSQL
relational-based DBMS.	**object-based** relational DBMS
Standard indexes	Supports partial, bitmap, and expression indexes
Standard views	Support for Materialised views and Table inheritance.
Supports **Standard data types**.	Supports **Advanced data types** such as arrays, hstore, geometry, json, and user-defined types
Simple, and faster.	Slower and more complex.