Data in the Wild

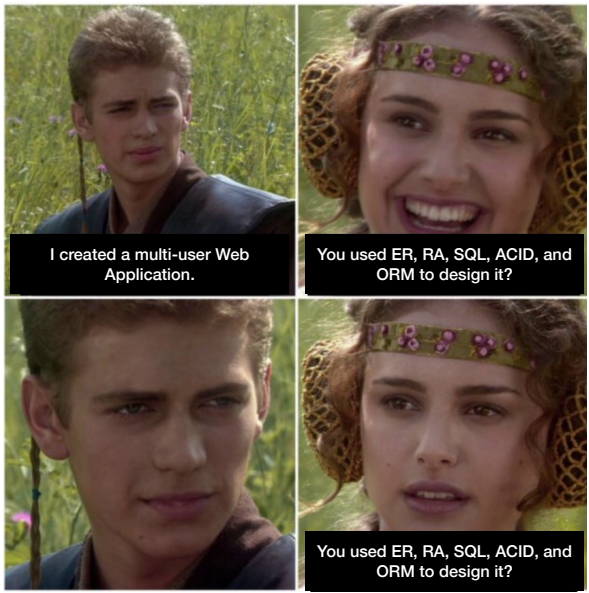
Big Data Engineering
(formerly Informationssysteme)

Prof. Dr. Jens Dittrich

bigdata.uni-saarland.de

July 13, 2023

Databases at University vs Databases in Reality



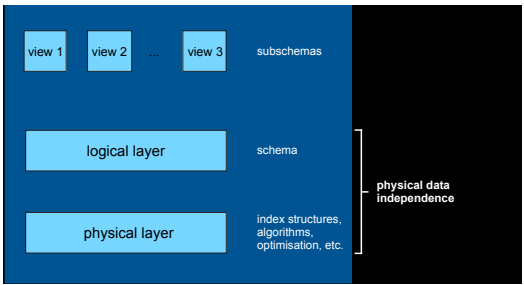
Uni vs Reality (Part 1)

Uni	Reality	Note
ER	No modelling, no interaction with customers, prefer to get started straight away; alternative: another data model: XML, JSON, OO, Graph; no sub-schemas, no logical data independence	logical data independence (LDI)
RM	Tables, iteratively extended over years (attributes and tables continuously added), partially hidden by object-relational mapping (ORM) wrappers (like e.g. in Django, but can also be separated!), many redundancies, no normalisation, many legacy fields (who actually uses this field?), no views, too broadly defined domains	subsequent normalisation, ORM, Django, LDI
RA	Some programming library that reinvents the wheel (e.g. Pandas), based on CSV files; queries formulated procedurally in library, hard-coded query plans	hard-coded vs Spark

Physical Data Independence

Physical Data Independence

The database schema is independent of its physical realisation. Changes to the physical representation of the data (hardware, indexes, etc.) have no effect on the database schema.



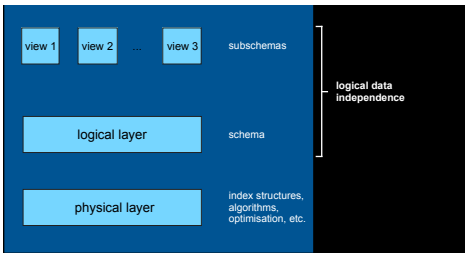
Advantages:
physical layer can be changed subsequently

Disadvantages:
Effort for abstraction (no direct hard-coding of physical actions)

Logical Data Independence

Logical Data Independence

The views of the end users are independent of the database schema. Changes to the database schema have no effect on the users' views.



Advantages:
logical layer can be changed subsequently

Disadvantages:

1. effort for creating views
2. possible problems with updates through views
3. too complicated to maintain for small apps

Logical Data Independence and Web Applications

Logical Data Independence and Web Applications

For Web Applications that run on a database that is not shared with other applications, logical data independence is typically **not** implemented on the database-layer but implemented by the application server.

However, if you are developing a Web Application that shares data with another app, you may want to consider implementing LDI on the database-layer.

Normalisation

Normalisation

Theory for determining the quality of relations with the help of functional dependencies.
Algorithms for improving the quality of relations.

Advantages:
good tool, in particular to reverse-engineer existing (messed-up) database schemas

Disadvantages:

1. a bit outdated
2. too strongly oriented towards atomic domains (as of SQL 92)
3. first normal form is already a contradiction to modern SQL

Hard-coded vs Spark

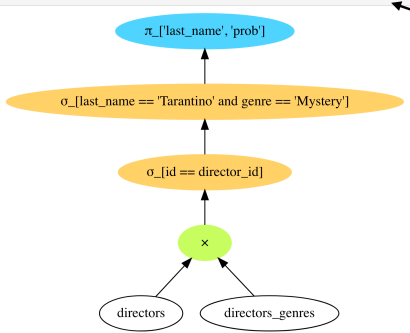
Procedural vs Declarative Way of Reading

Just because it looks like it is a procedural program does not mean we have to run it in that order. (see Rule-based Optimization.ipynb)

```
In [4]: # Unoptimized plan
        cp = Cartesian_Product(directors, directors_genres)
        sel1 = Selection(cp, "id == director_id")
        sel2 = Selection(sel1, "last_name == 'Tarantino' and genre == 'Mystery'")
        proj = Projection(sel2, ['last_name', 'prob'])

        graph = proj.get_graph()
        Source(graph)
```

Out[4]:



Interpretation 1
This is procedural code that should/ must be executed exactly in this order.

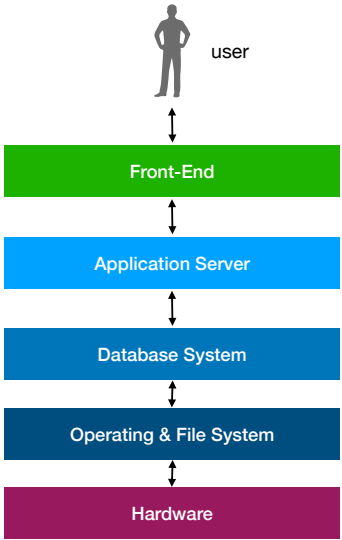
Interpretation 2
This is just a special form of declarative programming. Even though it looks like a procedural program, we can do what we want as long as nothing changes in the calculated result.

(similar condition as with isolation)

Uni vs Reality (Part 2)

Uni	Reality	Note
SQL	only SQL 92, and only partially; application logic is only used to read relations into the application, reinvents parts of SQL and query optimiser, poor scalability; SQL-hints, materialised views.	user logic horror story, cut between pre- and post-processing
Data Economy	mostly not followed, on the contrary: data collection mania is the standard	
A	often only used per tuple: key/value stores, NoSQL	KV semantics
C	hardly used beyond foreign keys; especially triggers hardly used; or simply no C-conditions at all; settings of the DBMS incorrectly used (transactional triggers are not that easy to use)	Trigger
I	all possible relaxations; with or without knowledge of the consequences, see A	eventual consistency
D	somehow part of the backup strategy	

Where to implement which functionality?



Example: filter()

filter(): all data is sent up to the user

filter(): all data is sent up to the front end

filter(): all data is sent up to the app server...

filter(): all data is sent up to the DB...

filter(): on the storage medium, only results or suitable superset is sent up

User Logic: Where to Implement Which Functionality?

Data intensive operations

The general rule is to execute functionality as far down the stack as possible.

Advantages: less data is sent around

Disadvantages:
not always easy to realise
often not necessary for development, as no performance problem

DB Functionality

In doubt, the functionality should be implemented in the DBMS or (partially) in lower layers. DB functionality in higher layers bears the risk of eventually becoming a performance problem and/or another problem that could be solved by the DBMS.

Example:
Transaction logic in the application server

Key/Value Stores

Key/Value Stores

1. Allow for the efficient storage and retrieval of keys mapped to arbitrary values.
2. Transactions over multiple keys are typically not supported.
3. Functionality and queries that cannot be mapped to key/value semantics are usually insufficiently supported.

Examples:

URL	$\mapsto$	content of a website (aka: the Internet)
hierarchical file path	$\mapsto$	content of a file (aka: a file system)
ID	$\mapsto$	JSON document (aka: a document store, e.g. MongoDB)

Key/Value Stores

Advantages:

- very efficient and sufficient if no other access patterns are needed
- some products usually also offer scale-out (distribution to several servers)

Disadvantages:

- very limited use cases
- very slow for queries that do not query via the key
- basically nothing but **an** index/hash map

ECA Rules and Triggers (1/2)

Event Condition Action (ECA) Rules

1. **E**vent: specifies an event
2. **C**ondition: specifies a condition that is checked for an event
3. **A**ction: specifies an action that is executed if the condition is fulfilled

Database trigger

A database trigger allows ECA rules to be formulated directly in the DBMS.

ECA Rules and Triggers (2/2)

Example:

```
CREATE TRIGGER log_update
    AFTER UPDATE ON accounts
    FOR EACH ROW
    WHEN (OLD.* IS DISTINCT FROM NEW.*)
    EXECUTE FUNCTION log_account_update();
```

Call a function that logs updates. But only if something has changed.

see [PostgreSQL documentation]

Video (in German):
https://youtu.be/aTeRR9XmPWE

Advantages:

- very powerful tool
- very suitable for complex consistency conditions, tracing, event-based changing of relations, i.e. these things can be implemented where they belong: in the DBMS!

Disadvantages:

- not quite easy to use (different syntax depending on DBMS)
- difficult to debug
- possibly undetected infinite loops
- rather slow

Uni vs Reality (Part 3)

Uni	Reality	Note
Security	designed or hacked in afterwards	
Privacy	conceived or hacked in afterwards, if it is considered an issue at all	
Quality Assurance	no staging environment	staging
Automatic Testing	app testing vs consistency of the DB (see trigger)	trigger vs test
Documentation	meaningless names of tables and attributes; semantics unclear; user roles (ACLs for reading and/or writing) of tables unclear; effects across tables unclear	LDI subsequently, ACL for users
Extensibility	difficult to impossible due to missing interfaces and views; unknown dependencies in the code	LDI subsequently
Physical Design	buy new hardware? buy new DBMS? indexes? refresh database statistics?	physical design advisory, performance tuning

Deployment Environment/Staging

Deployment Environment

In a deployment environment, several versions of the same system are arranged in a pipeline.

1. **Development:** system for developing new features on any sample data
2. **Test:** system for testing new features, typically connected through continuous integration, should test appropriate benchmarks and scenarios
3. **Staging:** system for testing new features; staging should correspond exactly to the production system, especially with regard to the state of the data. This means ideally a replica of the production system. Also useful for load testing.
4. **Production:** production system with real data, real customers

[https://en.wikipedia.org/wiki/Deployment_environment]

Example:

Remember the motivation from the slide set "Trading, Banking, Ticket Systems..."? Guess what the reason was for one of the crashes of one of these banks...

Physical Design Advisory and Performance Tuning

Physical data independence is great, but: ultimately, someone has to define and configure how the data is physically stored (indexes, hardware, data distribution, etc.). This is typically done by a database administrator (DBA). Many DBMSs provide extensive tools for this purpose.

Physical Design Advisory

Tool for semi-automatically setting the physical configuration of a DBMS.

Example:
Database optimisation guide https://learn.microsoft.com/en-us/sql/tools/dta/tutorial-database-engine-tuning-advisor?view=sql-server-ver15

Auto tuning

Tool for fully automatic setting of the physical configuration of a DBMS.

Example:
https://ottertune.com/

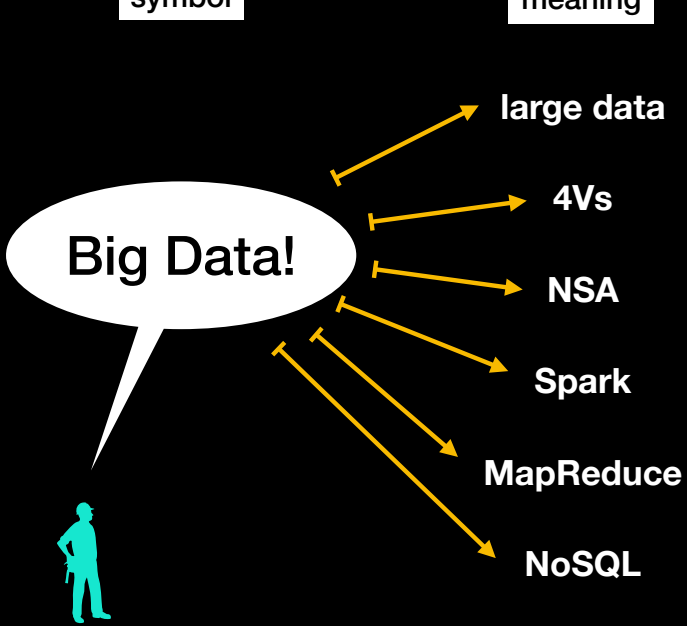
Uni vs Reality (Part 4)

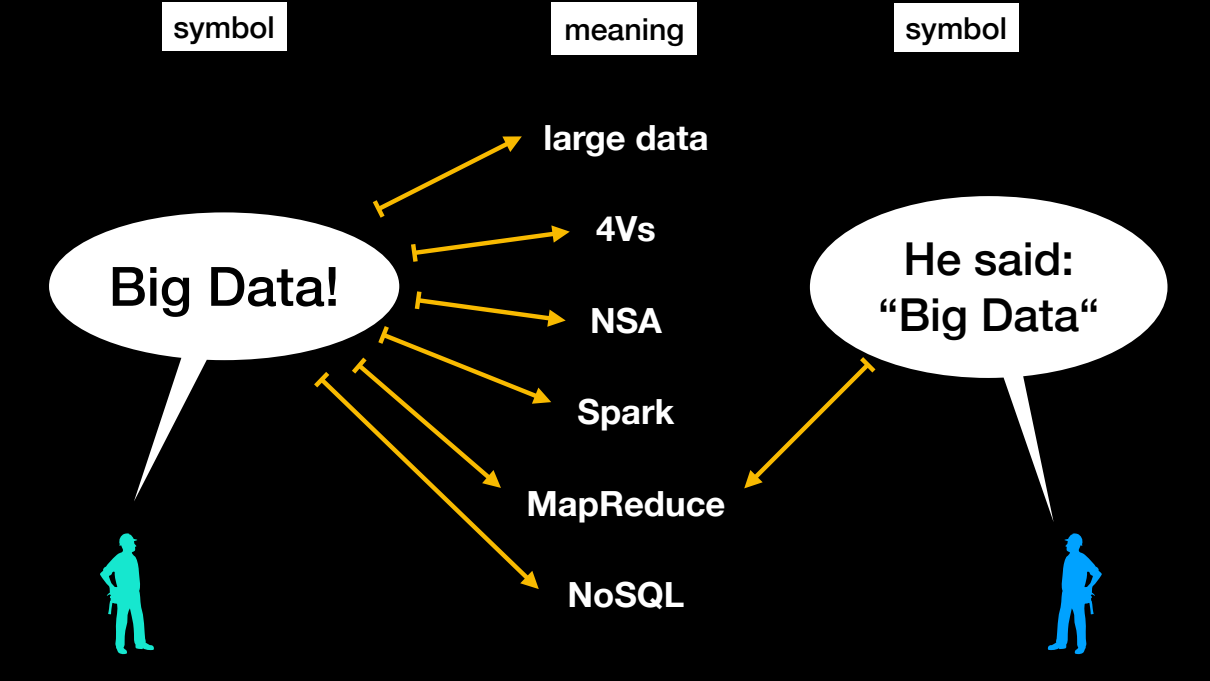
Uni	Reality	Note
Data Management	often alignment with organisational hierarchy necessary, social aspects especially power games; data distributed across many different systems and databases; query processing along organisational hierarchy; many data integration problems	business processes vs IT architecture
Genuine Knowledge, Competence	perceived knowledge or active concealment and/or ignoring of one's own lack of knowledge (aka smart-assing and dumbing down)[1]	delimitation of own knowledge
Concepts	specific implementations, tools and products that also somehow use certain ancient concepts (but do not necessarily make this clear)	concept vs mapping to technology
Technical Terms	buzzwords (used unconsciously or consciously)	buzzword bullshit bingo

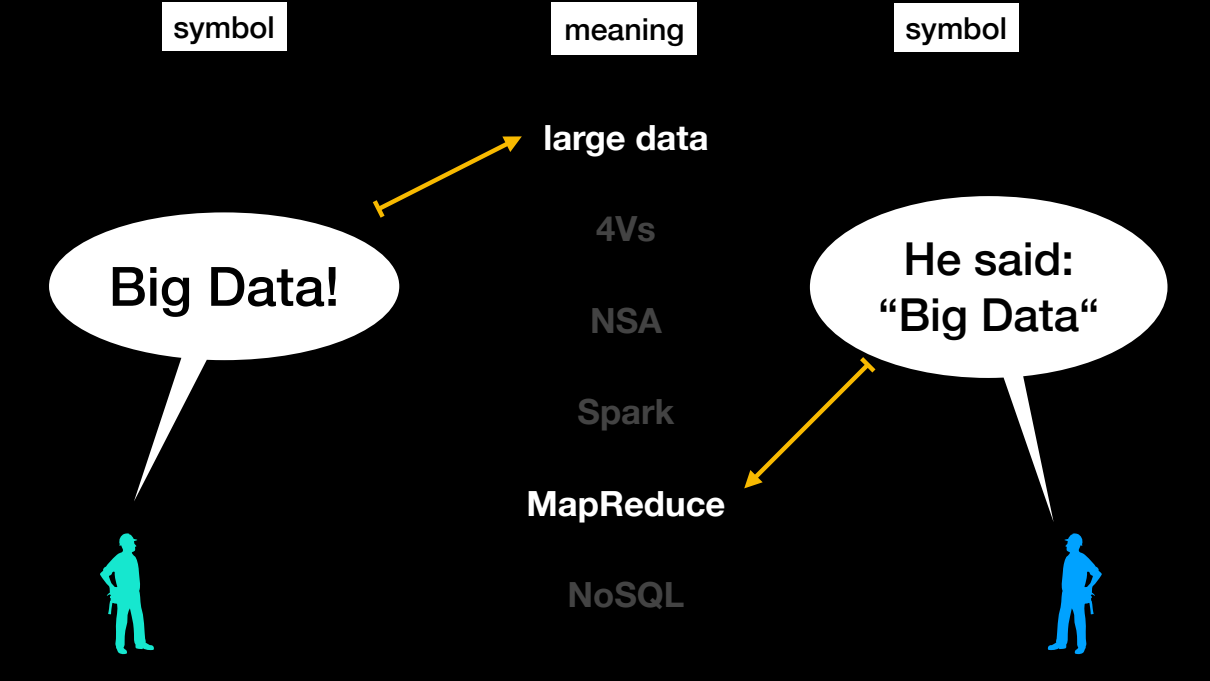
[1] https://thedailywtf.com/articles/classic-wtf-the-mainframe-database

Buzzword Bullshit Bingo

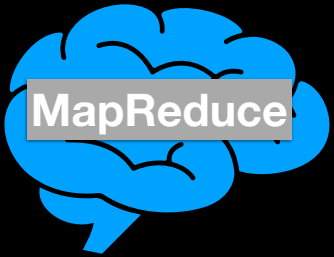
Problem 1:
ambiguous communication


symbol
meaning
large data
4Vs
Big Data!
NSA
Spark
MapReduce
NoSQL


symbol
meaning
symbol
large data
4Vs
NSA
Spark
MapReduce
NoSQL
Big Data!
He said:
“Big Data“


symbol
meaning
symbol
large data
4Vs
NSA
Spark
MapReduce
NoSQL
Big Data!
He said:
“Big Data“

translated to:



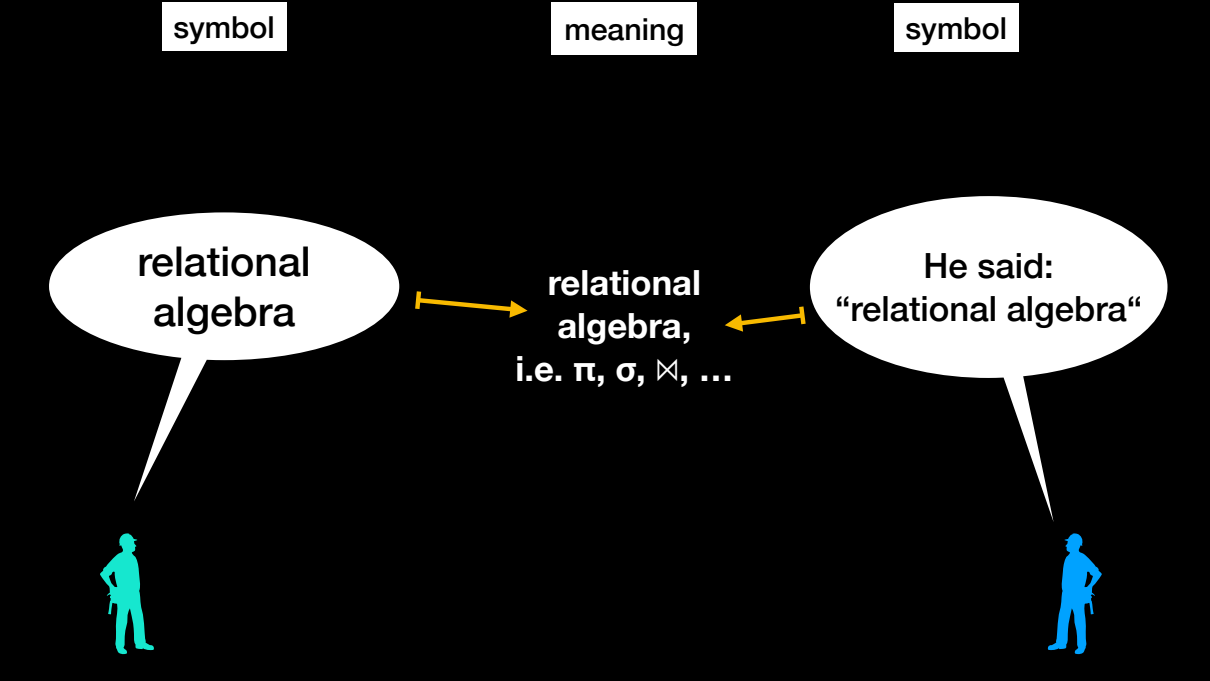
clear communication:

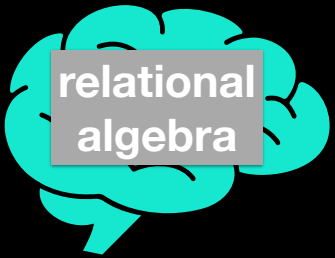
symbol

meaning

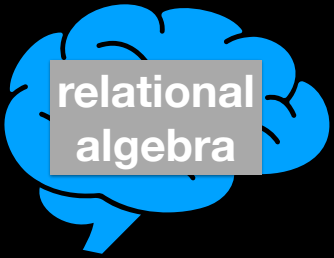
relational algebra

relational algebra, i.e. π, σ, ⋈, …

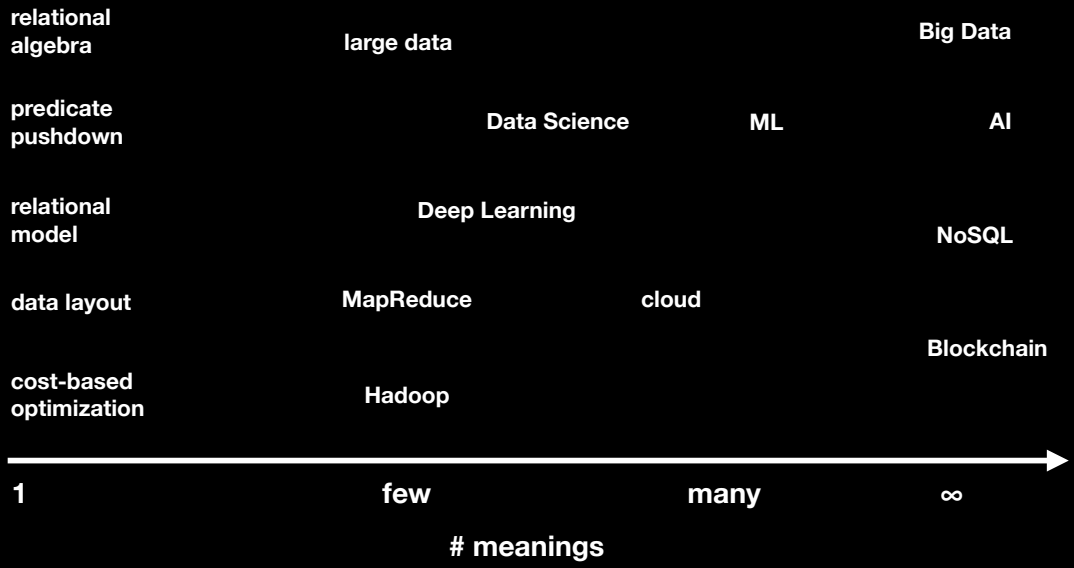

symbol
meaning
symbol
relational algebra
relational algebra, i.e. π, σ, ⋈, …
He said: “relational algebra“



translated to:

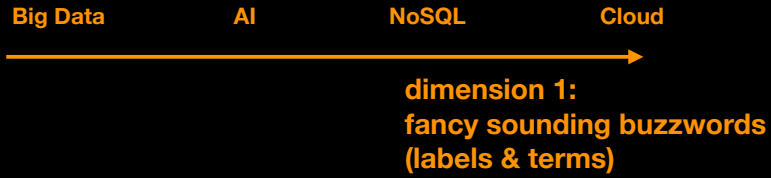


The Buzzword Bullshit Bingo Landscape



Concept vs Technology

Problem 2:
confusion of dimensions


Big Data
AI
NoSQL
Cloud
dimension 1:
fancy sounding buzzwords
(labels & terms)

dimension 2:
technical principles
and patterns
(concepts, best
practices)

predicate pushdown

relational model

relational algebra

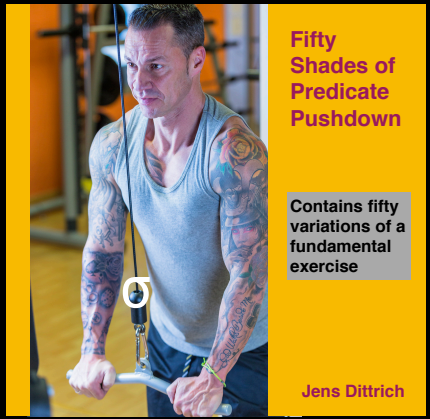
data layouts,
e.g. column vs row

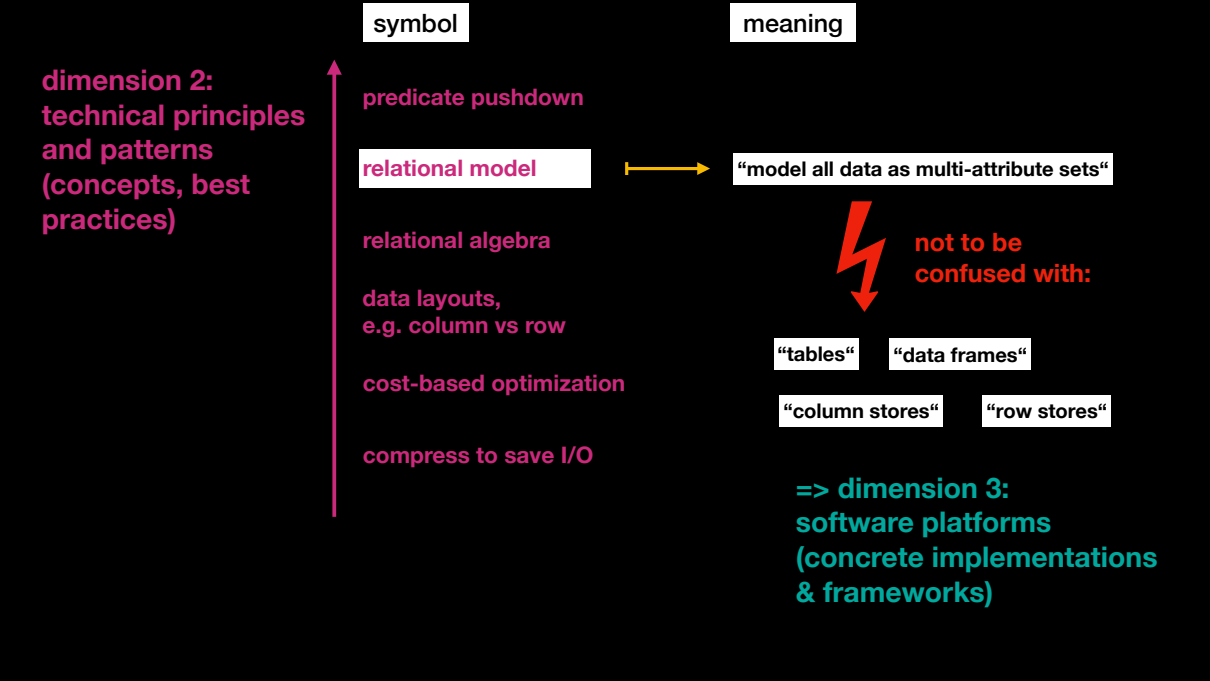
cost-based optimization

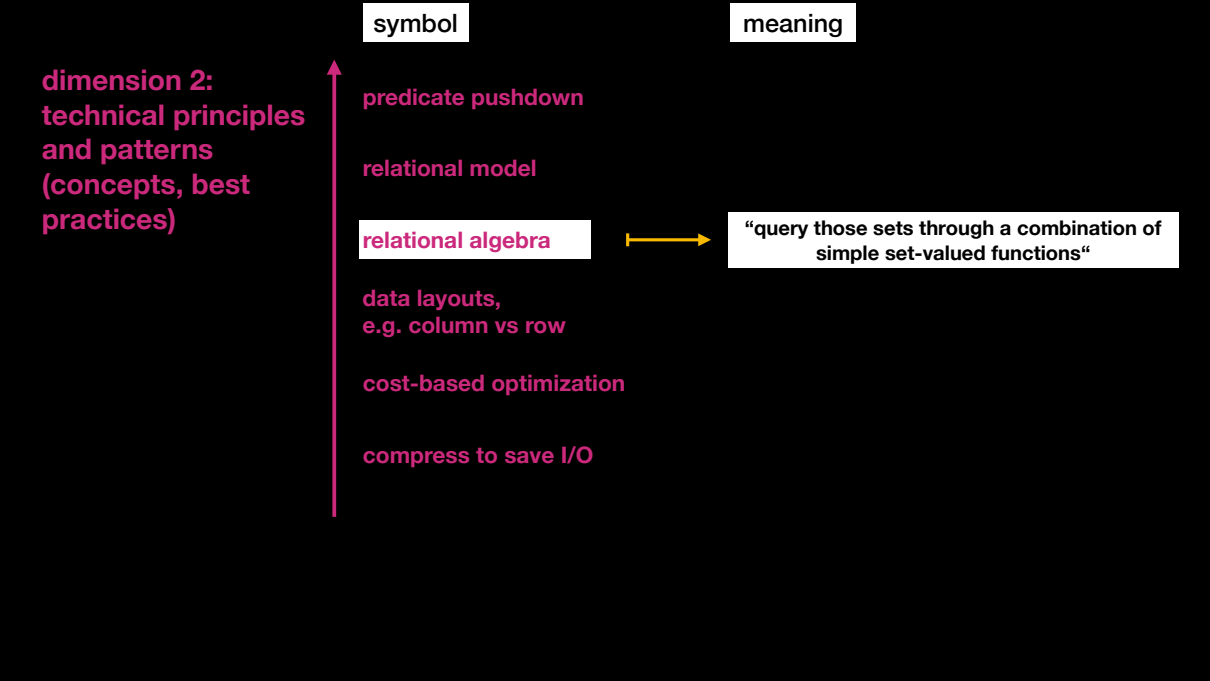
compress to save I/O

dimension 2: technical principles and patterns (concepts, best practices)

symbol	meaning
predicate pushdown	“filter and project data as early as possible“
relational model	
relational algebra	
data layouts, e.g. column vs row	
cost-based optimization	
compress to save I/O	




symbol
meaning
dimension 2:
technical principles
and patterns
(concepts, best
practices)
predicate pushdown
relational model
“model all data as multi-attribute sets“
relational algebra
not to be
confused with:
data layouts,
e.g. column vs row
“tables“
“data frames“
cost-based optimization
“column stores“
“row stores“
compress to save I/O
=> dimension 3:
software platforms
(concrete implementations
& frameworks)


symbol
meaning
dimension 2:
technical principles
and patterns
(concepts, best
practices)
predicate pushdown
relational model
relational algebra
“query those sets through a combination of simple set-valued functions“
data layouts,
e.g. column vs row
cost-based optimization
compress to save I/O

dimension 2:
technical principles
and patterns
(concepts, best
practices)

predicate pushdown

relational model

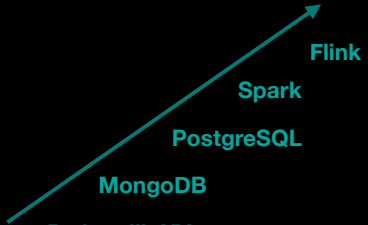
relational algebra

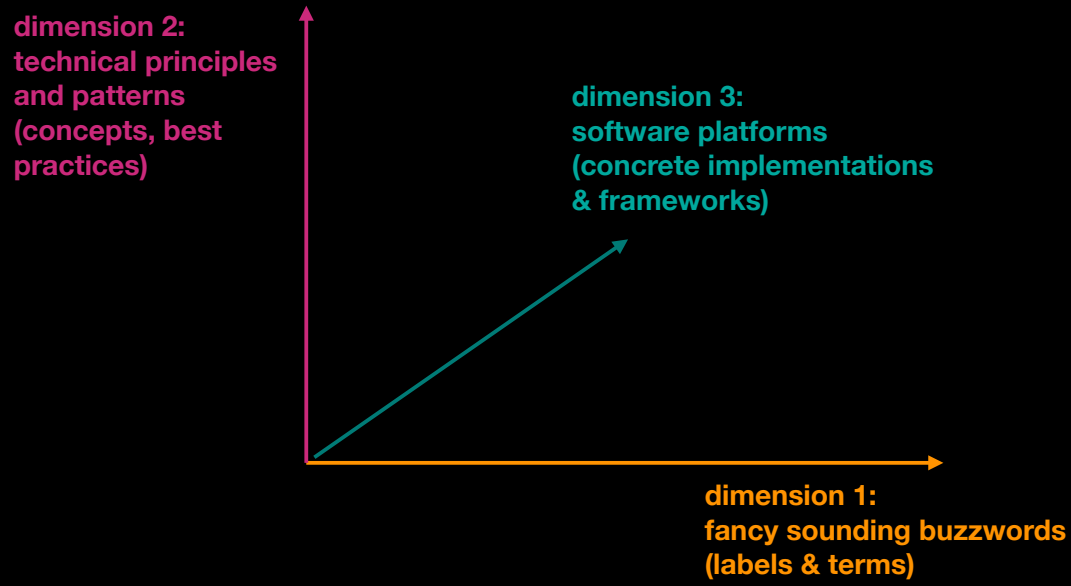
data layouts,
e.g. column vs row

cost-based optimization

compress to save I/O

dimension 3:
software platforms
(concrete implementations
& frameworks)




dimension 2:
technical principles
and patterns
(concepts, best
practices)
dimension 3:
software platforms
(concrete implementations
& frameworks)
dimension 1:
fancy sounding buzzwords
(labels & terms)

Example Consultation Scenario

(experienced in 2022...)

Problem (Part 1)

- large web application with terribly slow performance: **minutes** instead of milliseconds
- i.e. we are talking about performance problems on the order of **factor 10,000 and more slower than it should be!!!**
- SAP system with approx. 100,000 tables, apparently no or hardly any views?

→ completely ridiculous performance numbers!

Solution Space

- profile runtime of the application vs runtime of the database query
- profile runtime of the database query on system with and without load
- start: profile DB performance on under-utilised system (test, staging or development system) and troubleshoot wherever possible.
- physical design, including indexes, and partitioning

Example Consultation Scenario

Problem (Part 2)

- more than 30,000 users

Solution Space

- clarify load distribution (**after** you have ruled out that it is due to individual queries without load).
- if app server and database server are on the same server, clarify whether this is a load problem, if so: put the database server on its own server
- clear performance monitoring: when does which part of the system cause a problem? SLAs!
- KIWI: Kill it with iron, SSDs vs hard disks
- performance load tests on a staging server

Example Consultation Scenario

Problem (Part 3)

- project setup: three "companies" involved in the development, each with several managers, project leaders, sub-project leaders and developers, with very different database knowledge: everything from very good (the DB admin) to individual managers, project leaders, application developers (little to zero knowledge), mostly non-computer scientists
- fingerpointing: "the others are to blame!"

Solution Space

- profiling a query can very quickly locate where the problem is (DB or App), no matter who is ultimately responsible
- problem here: several performance problems combined
- identify and address knowledge deficits in staff: either train people appropriately or fire them (I mean that the way I say it).
- establish clear role/responsibility for physical design of the database

Summary

Problem: Lack of Fit between Training and Task

- database technology is often used by people who have no real clue
- aggravated by the fact that these people often believe they have a clue
- in addition, these people are often also resistant to advice
- this unpleasant mixture leads to a variety of:
  - performance problems
  - security issues
  - architectural mistakes (including purchasing decisions)
  - enormously high costs

Would you have your teeth treated by someone who actually learned to be a baker?
Would you let yourself be operated on by someone who actually trained as a butcher?

Solution

Ensure that those who use a tool understand that tool.

To Put it a Little More Pointedly

Advice:

If you are managed by someone who only has business knowledge but no technical knowledge, then run!

Summary

Main Takeaway

When it comes to using database technology in IT projects, there can be quite a discrepancy between what should be done and what is done.

How to Cope with That

1. If you have to deal with database technology, be careful of the pitfalls.
2. Most of the perceived database problems with web applications can be solved relatively easily. This includes most (if not all) performance problems.
3. Be extra careful when it comes to security and privacy risks/issues.
4. If in doubt, ask an expert (not a wannabe expert).