SQL is Dead, Long Live SQL!

Prof. dr. Wilfried Lemahieu

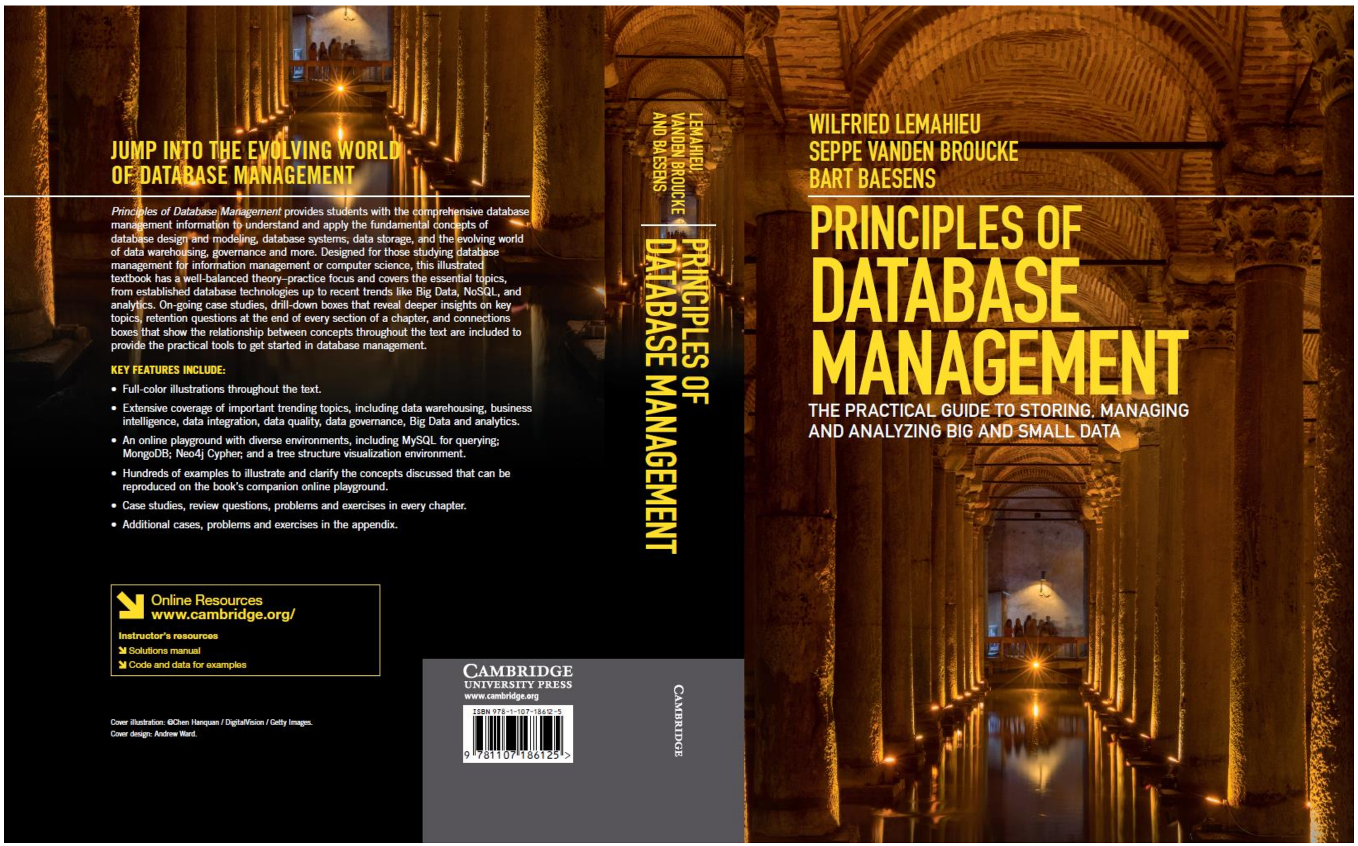
Faculty of Economics and Business

KU Leuven

Wilfried.Lemahieu@kuleuven.be

Our New Book!

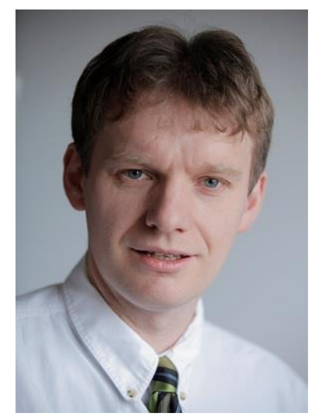
www.pdbmbook.com



Author Team

- Prof. Wilfried Lemahieu
  - professor and dean at KU Leuven (Belgium)
  - Database Management, Data Integration, Data Quality
  - Wilfried.Lemahieu@kuleuven.be

- Prof. Seppe vanden Broucke
  - professor at KU Leuven (Belgium)
  - Process and Data Science
  - Seppe.vandenBroucke@kuleuven.be



- Prof. Bart Baesens
  - professor at KU Leuven (Belgium) and University of Southampton (UK)
  - Analytics, Credit Risk, Fraud Detection
  - Bart.Baesens@kuleuven.be

BAAS (Book as a Service)

- Book website: www.pdbmbook.com
  - Free YouTube lectures
  - Free PowerPoint slides
    - Available in English, Mandarin and Spanish
  - Free on-line multiple choice quiz tool
  - Online playground including MySQL, MongoDB, and Neo4j Cypher (also available as a Dockerfile)
  - Solutions manual

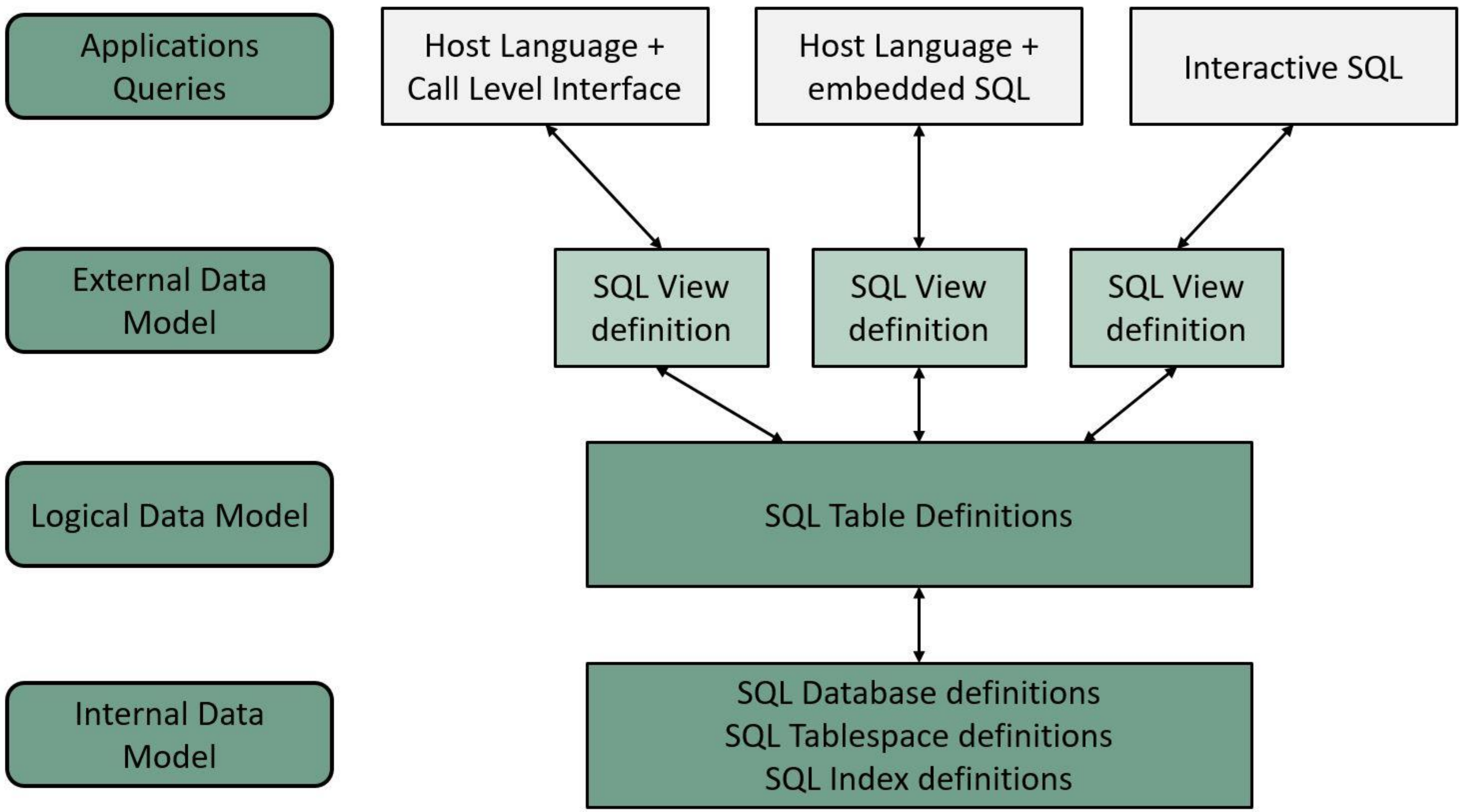
Overview

- SQL and relational databases
- NoSQL
  - Key-value stores
  - Document stores
  - Column-oriented databases
  - Graph databases
  - Transaction management, APIs, other NoSQL types
- Evaluating NoSQL databases: is SQL dead and buried?

SQL

- First version SQL-86 in 1986
- Most recent version in 2016 (SQL:2016)
- Accepted by
  - American National Standards Institute (ANSI) in 1986
  - International Organization for Standardization (ISO) in 1987
- Each vendor provides own implementation
  - SQL dialect

SQL: DDL + DML



Applications Queries
External Data Model
Logical Data Model
Internal Data Model
Host Language + Call Level Interface
Host Language + embedded SQL
Interactive SQL
SQL View definition
SQL View definition
SQL View definition
SQL Table Definitions
SQL Database definitions
SQL Tablespace definitions
SQL Index definitions

SQL: DDL + DML

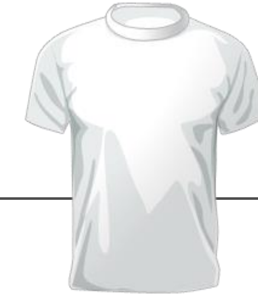
```
CREATE TABLE SUPPLIER
(SUPNR CHAR(4) NOT NULL PRIMARY KEY,
 SUPNAME VARCHAR(40) NOT NULL,
 SUPADDRESS VARCHAR(50),
 SUPCITY VARCHAR(20),
 SUPSTATUS SMALLINT)

SELECT SUPNR, SUPNAME
FROM SUPPLIER
WHERE SUPCITY = 'San Francisco'
AND SUPSTATUS > 80

SELECT SUPNAME
FROM SUPPLIER R
WHERE EXISTS
         (SELECT *
           FROM SUPPLIES S
           WHERE R.SUPNR = S.SUPNR
           AND S.PRODNR = '0178')
```

```
DELETE FROM SUPPLIES S1
WHERE S1.PURCHASE_PRICE >
  (SELECT 2*AVG(S2.PURCHASE_PRICE)
FROM SUPPLIES S2
WHERE S1.PRODNR = S2.PRODNR)

CREATE VIEW ORDEROVERVIEW(PRODNR,
PRODNAME, TOTQUANTITY)
AS SELECT P.PRODNR, P.PRODNAME,
SUM(POL.QUANTITY)
FROM PRODUCT AS P LEFT OUTER JOIN PO_LINE
AS POL
ON (P.PRODNR = POL.PRODNR)
GROUP BY P.PRODNR

CREATE UNIQUE INDEX PRODNR_INDEX
ON PRODUCT(PRODNR ASC)

GRANT SELECT, INSERT, UPDATE, DELETE
ON SUPPLIER TO BBAESENS
```

Key Characteristics of SQL

- Set-oriented and declarative
- Free form language
- Case insensitive
- Executed interactively from command prompt or by a program
- Very powerful language!

Relational databases: one size fits all ?



- Formal data model based on normalised tables, foreign keys, static and strict database schema
- ACID transactions (atomicity, consistency, isolation, durability)
- Vertical scalability, but limited horizontal scalability
- Scalability and availability restricted by strong focus on consistency

- Application domains:
  - Good performance with intensive read/write operations on small(ish) data sets or large batch processes with limited amount of simultaneous transactions
  - Oriented towards structured data, rather than semi-structured and unstructured data
  - Database functionality (complex queries, transaction mgmt, …) is sometimes ‘overkill’ compared to application needs

And then came Web 2.0…

- Volume + Variety + Velocity
- Storage of massive amounts of (semi-)structured and unstructured, highly dynamic data
- Need for flexible storage structures (no fixed schema)
- Availability and performance often favoured over consistency
- Complex query facilities not always needed: just put/get data
- Need for massive horizontal scalability (server clusters) with flexible reallocation of data to server nodes
- Big Data Analytics
- Cloud computing and cloud data services

NoSQL databases

- Read as: "not only SQL"
- Umbrella term for diverse systems with (partially) similar properties:
  - No relational data model
  - No (or limited) schema restrictions
  - Distribution and (nearly linear) horizontal scalability
  - Massive replication for availability (failover) and performance (load balancing)
  - Diverse types of (often very simple) APIs
  - Often emanating from open source community
  - Different transaction paradigm: BASE (basically available, soft state eventually consistent) instead of ACID
- Other key terms:
  - Map/reduce
  - Consistent hashing
  - multi version concurrency control (MVCC)
  - Eventual consistency

Properties of NoSQL databases

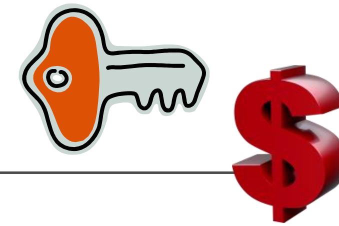
- Classification according to data model
  - Key-value stores
  - Document stores
  - Column-oriented databases
  - Graph-based databases
  - …

- Other
  - Query facilities + APIs
  - Transaction management and concurrency control

NoSQL Versus SQL

	Relational Databases	NoSQL Databases
Data paradigm	Relational tables	Key-value (tuple) based Document based Column based Graph based XML, object based Others: time series, probabilistic, etc.
Distribution	Single-node and distributed	Mainly distributed
Scalability	Vertical scaling, harder to scale horizontally	Easy to scale horizontally, easy data replication
Openness	Closed and open source	Mainly open source
Schema role	Schema-driven	Mainly schema-free or flexible schema
Query language	SQL as query language	No or simple querying facilities, or special-purpose languages
Transaction mechanism	ACID: Atomicity, Consistency, Isolation, Durability	BASE: Basically available, Soft state, Eventual consistency
Feature set	Many features (triggers, views, stored procedures, etc.)	Simple API
Data volume	Capable of handling normal-sized data sets	Capable of handling huge amounts of data and/or very high frequencies of read/write requests

Key-value Stores



- Concept: storage of (key, data value) couples
- The unique keys are the only criterion for data retrieval
- Store and retrieve across multiple nodes by hashing the key (‘consistent hashing’)
- Data values = BLOBs, no meaning, no search criteria
- No schema; data interrelations managed at application level

- Mainly useful for simple put/get functionality, based on (part of) key
- Scalability and performance
- Often foundation layer to more complex systems

- Examples: Memcached, Redis, Membase, Dynamo (Amazon), Bigtable (Google)

Key-value Stores

```
import java.util.HashMap;
import java.util.Map;
public class KeyValueStoreExample {
	public static void main(String...  args) {
		// Keep track of age based on name
		Map<String, Integer> age_by_name = new HashMap<>();

		// Store some entries
		age_by_name.put("wilfried", 34);
		age_by_name.put("seppe", 30);
		age_by_name.put("bart", 46);
		age_by_name.put("jeanne", 19);

		// Get an entry
		int age_of_wilfried = age_by_name.get("wilfried");
		System.out.println("Wilfried's age: " + age_of_wilfried);

		// Keys are unique
		age_by_name.put("seppe", 50); // Overrides previous entry
	}
}
```

Key-value Stores

- Keys (e.g., “bart”, “seppe”) are hashed by means of so-called hash function
  – Hash function takes arbitrary value of arbitrary size and maps it to key with fixed size (hash value)
  – Hash can be mapped to space in computer memory

Key
wilfried
seppe
bart
jeanne

Hash function →

Hash	Key
01	(wilfried,34)
03	(seppe,30)
07	(bart,46)
08	(jeanne,19)

Key-value Stores

- Remember: NoSQL databases built with horizontal scalability support in mind
- Distribute hash table over different locations
- Assume we need to spread hashes over 3 servers
  - Hash every key (“wilfried”, “seppe”) to server identifier
  - index(hash) = mod(hash, nrServers) + 1

Key-value Stores

Key
wilfried
seppe
bart
jeanne

Hash function

Hash	Index (= mod 3 + 1)
01	2
03	1
07	2
08	3

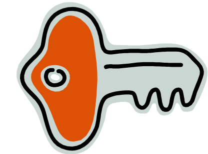
Sharding!

Server #1	Entries
03	(seppe,30)

Server #2	Entries
01	(wilfried,34)
07	(bart,46)

Server #3	Entries
08	(jeanne,19)

Document Stores



- Concept: storage of (key, document) couples
- The DBMS is aware of the document type and interprets the document content
- Document formats: semi-structured data, a.o. XML, JSON (JavaScript Object Notation), YAML (YAML Ain't Markup Language), …
- Documents contain attributes: (key, value) couples. Therefore, we also speak of tuple stores, i.e. the document is a vector of data
- Document processing (add/change attributes); attributes as search criteria
- Complex data structures and nested objects; no fixed schema

- Examples: CouchDB, MongoDB

Document Stores

- Most Document stores (e.g. MongoDB) choose to represent documents using JSON

```
{
        "title": "Harry Potter",
        "authors": ["J.K.  Rowling", "R.J.  Kowling"],
        "price": 32.00,
        "genres": ["fantasy"],
        "dimensions": {
                "width": 8.5,
                "height": 11.0,
                "depth": 0.5
        },
        "pages": 234,
        "in_publication": true,
        "subtitle": null
}
```

Document Stores

- Most NoSQL document stores allow to store items in tables (collections) in schema-less way, but enforce that primary key be specified
- Document stores exhibit many similarities to relational databases
  - Including query, aggregation and indexing facilities
- MapReduce
  - Open source software framework for distributed computing and storage of large data sets

Document Stores

- Map Reduce
  - Map-reduce pipeline starts from series of key-value pairs (k1,v1) and maps each pair to 1 or more output pairs
  - Output entries shuffled and distributed so that all output entries belonging to same key are assigned to same worker (e.g., physical machines)
  - Workers then apply reduce function to each group of key-value pairs having same key, producing new list of values per output key
  - Resulting, final outputs are then (optionally) sorted per key k2 to produce final outcome

Document Stores

- Example: get a summed count of pages for books per genre
- Create list of input keys-value pairs

k1	v1
1	{genre: education, nrPages: 120}
2	{genre: thriller, nrPages: 100}
3	{genre: fantasy, nrPages: 20}
…	…

- Map function is simple conversion to genre-nrPages key-value pair

```
function map(k1, v1)
     emit output record (v1.genre, v1.nrPages)
end function
```

Document Stores

- Workers have produced following 3 output lists, with keys corresponding to genres

Worker 1	
k2	**v2**
education	120
thriller	100
fantasy	20

Worker 2	
k2	**v2**
drama	500
education	200

Worker 3	
k2	**v2**
education	20
fantasy	10

- Working operation started per unique key k2, for which its associated list of values will be reduced
  - E.g., (education,[120,200,20]) will be reduced to its sum, 340

```
function reduce(k2, v2_list)
     emit output record (k2, sum(v2_list))
end function
```

Document Stores

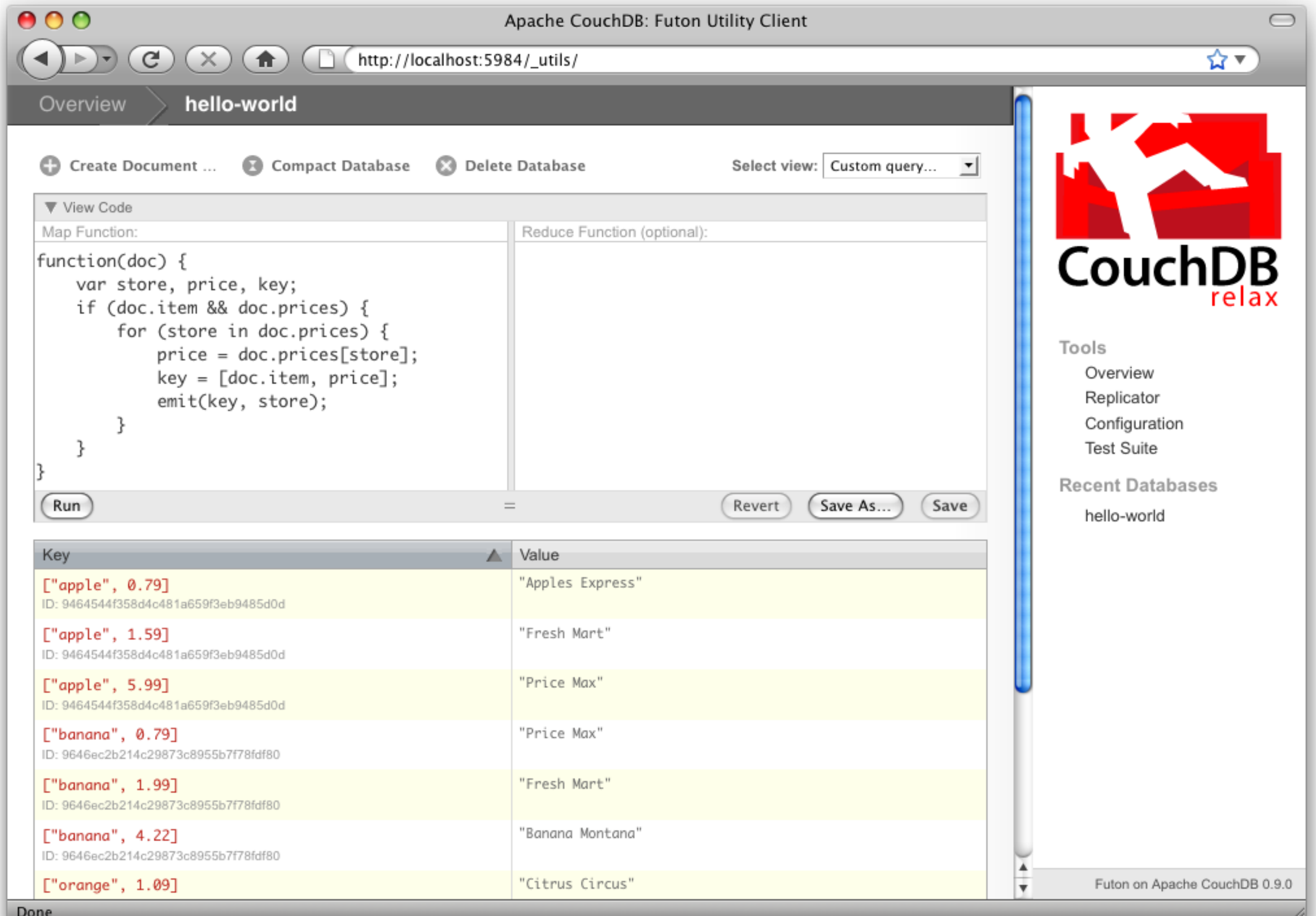
- Final output looks as

k2	v3
education	340
thriller	100
drama	500
fantasy	30

- Can be sorted based on k2 or v3

GROUP BY style SQL queries are convertible to equivalent MapReduce pipeline

Document Stores



Column-oriented Databases

- Concept: unit of storage ≠ record with attribute values (as in RDBMS), but the values of the same attribute type for a set of records (cf. column in RDBMS)
- All values of a storage unit have the same data type
- No storage of null values (“sparse data”) >< RDBMSs

- Aimed at structured data
- Scalability, very efficient aggregation of values (sum, average, …) + sparse data (null values)
- No/limited support for data interrelations or more complex queries (e.g. joins)

- Examples: Cassandra, Hbase, Google BigTable, Parquet

Column-oriented Databases

- Example

Id	Genre	Title	Price	Audiobook price
1	fantasy	My first book	20	30
2	education	Beginners guide	10	null
3	education	SQL strikes back	40	null
4	fantasy	The rise of SQL	10	null

- Row based databases not efficient at performing operations that apply to entire data set
  - Need indexes which add overhead

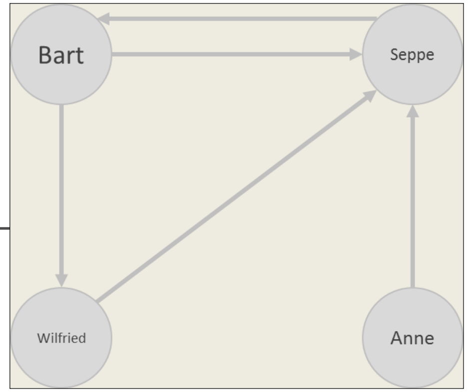
Column-oriented Databases

- In column-oriented database, all values of column are placed together on disk

```
Genre: fantasy:1,4 education:2,3
Title: My first book:1  Beginners guide:2  SQL strikes back:3 The rise of SQL:4
Price: 20:1    10:2,4   40:3
Audiobook price: 30:1
```

- Operations such as: find all records with price equal to 10 can now be executed directly
- Null values do not take up storage space anymore
- But: retrieving all attributes pertaining to a single entity becomes less efficient

Graph-based Databases



- Concept: storage of nodes and links/edges
- Nodes and links have unique IDs and may also contain (key, attribute) couples to represent properties (e.g. distance) or link types (e.g. married_to)
- Focus on data interrelations >< other NoSQL databases
- Flexible interrelations >< RDBMSs
- Navigate links instead of (expensive) joins
- Sometimes schema definition (e.g. constraints on node-link combinations)

- Applications: location based services, knowledge representation, navigation systems, recommender systems, …

- Examples: FlockDB (Twitter), InfiniteGraph, Neo4j (+ Cypher)

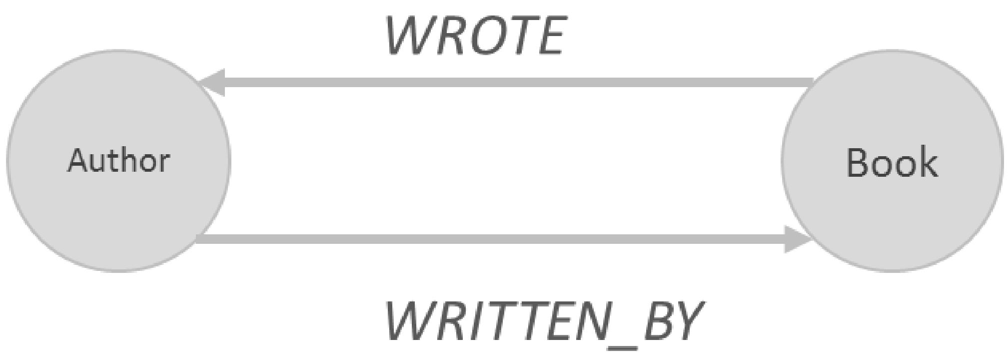
Graph-based Databases

- One-to-one, one-to-many, and many-to-many structures can easily be modeled in a graph
- Consider N-M relationship between books and authors
- RDBMS needs 3 tables: Book, Author and Books_Authors
- SQL query to return all book titles for books written by a particular author would look like follows

```
SELECT title
FROM books, authors, books_authors
WHERE author.id = books_authors.author_id
  AND books.id = books_authors.book_id
  AND author.name = "Bart Baesens"
```

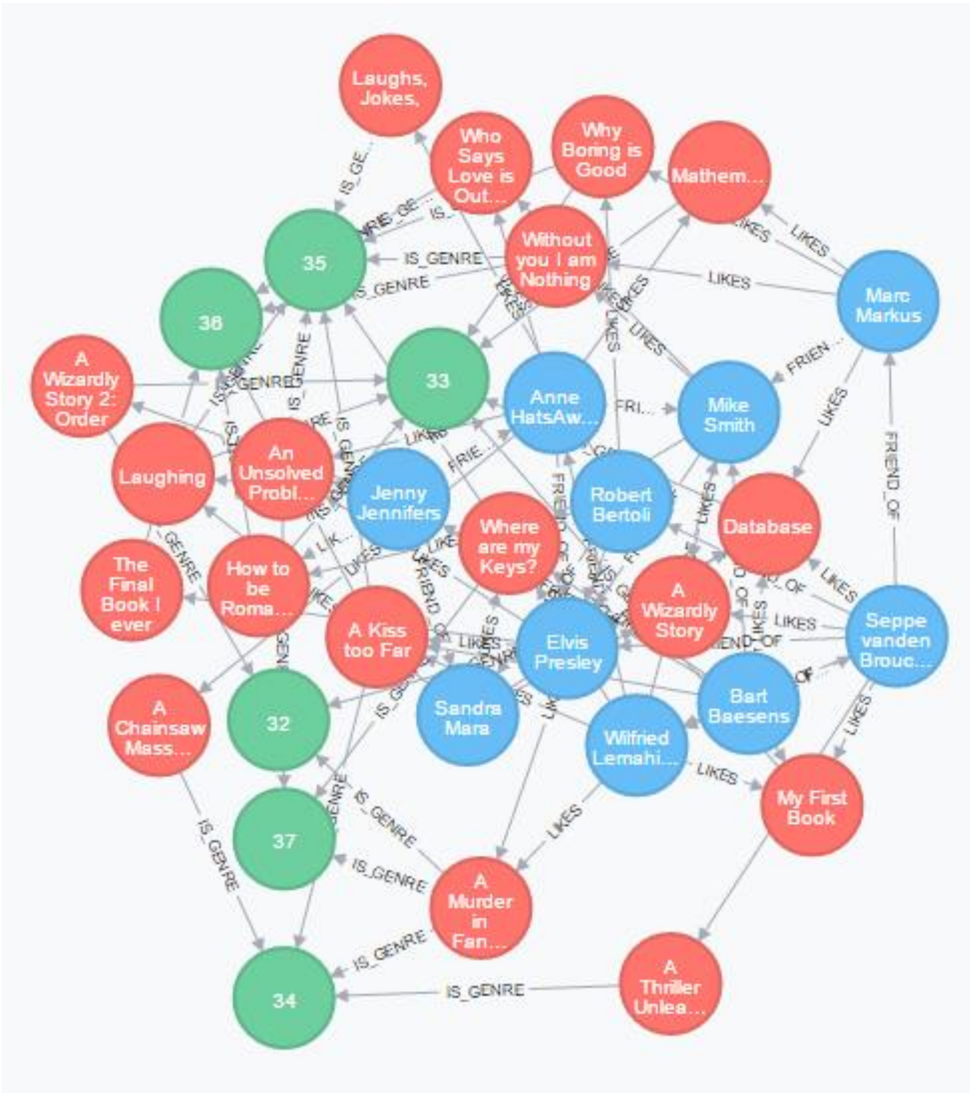
Graph-based Databases

- In a graph database (using Cypher query language from Neo4j)




```
MATCH (b:Book)<-[:WRITTEN_BY]-(a:Author)
WHERE a.name = "Bart Baesens"
RETURN b.title
```

Graph-based Databases



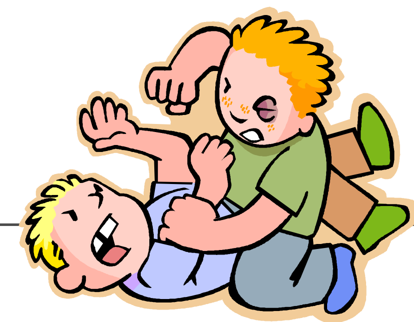
Who likes romance books?

```
MATCH (r:Reader)--(:Book)--(:Genre
{name:'romance'})
RETURN r.name
```

Who are Bart's friends that liked Humor books?

```
MATCH (me:Reader)--(friend:Reader)--
(b:Book)--(g:Genre)
WHERE g.name = 'humor' AND me.name =
'Bart Baesens'
RETURN DISTINCT friend.name
```

Transaction Management and Concurrency Control



- RDBMSs: concurrency control through locking, atomic transactions ('ACID')
  → impact on performance/throughput, esp. with replicated data

- NoSQL: multiversion concurrency control (MVCC): no locking, but multiple versions are stored for a data item in chronological order
  - Read: (hopefully) most recent version(s)
  - Write = create new versions
  - Conflicts are solved by DBMS or client
  - Eventual consistency: asynchronous propagation of updates; all replicas of a data item become consistent 'eventually', but not immediately as with ACID (cf. tweets in a SN)

Transaction Management and Concurrency Control

- NoSQL DBMSs do not give up on consistency altogether
- BASE transactions:
  - Basically Available: measures are in place to guarantee availability under all circumstances, if necessary at the cost of consistency
  - Soft State: the state of the database may evolve, even without external input, due to the asynchronous propagation of updates throughout the system
  - Eventually consistent: the database will become consistent over time, but may not be consistent at any moment and especially not at transaction commit

Query facilities and APIs

- No standard query language or API
- Key-value stores:
  - Just API with key based put() and get() methods
  - Often REST and/or SOAP interface
- Document stores:
  - Richer API; search and manipulate document content
  - ‘Range’ queries on attribute values
- Column oriented databases:
  - Very efficient range and aggregate queries on attribute values
- Graph databases:
  - Graph pattern matching: find parts of graph that match search pattern
  - Graph traversal: navigate graph according to predefined path (breadth-first, depth-first)
  - Very efficient for querying transitive relationships (>< RDBMS)
  - FlockDB: no query language, just look for related data items (‘follows’ in Twitter)
  - Dedicated query languages, e.g. Cypher (Neo4J)

Query facilities and APIs

- MapReduce: parallel searching and processing of large data volumes in distributed storage clusters
- In-database analytics
- Low level programming; considerable ‘plumbing’

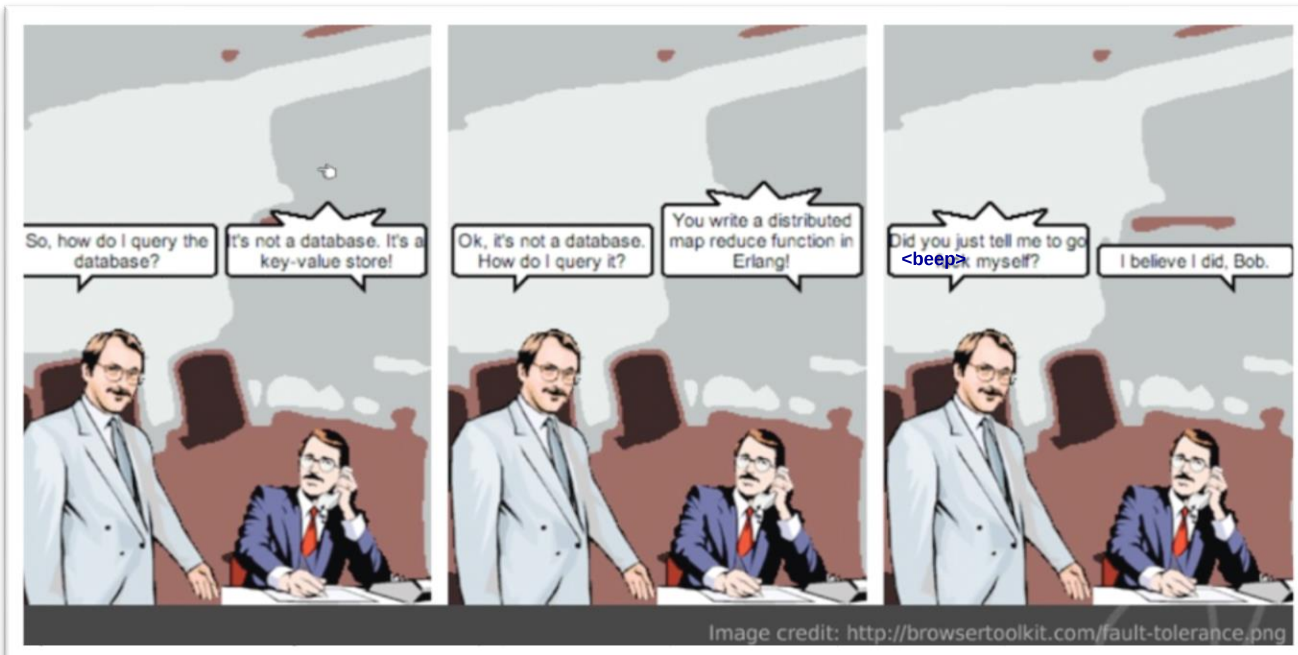


Image credit: http://browsertoolkit.com/fault-tolerance.png

Data warehouses vs data lakes

Traditional data

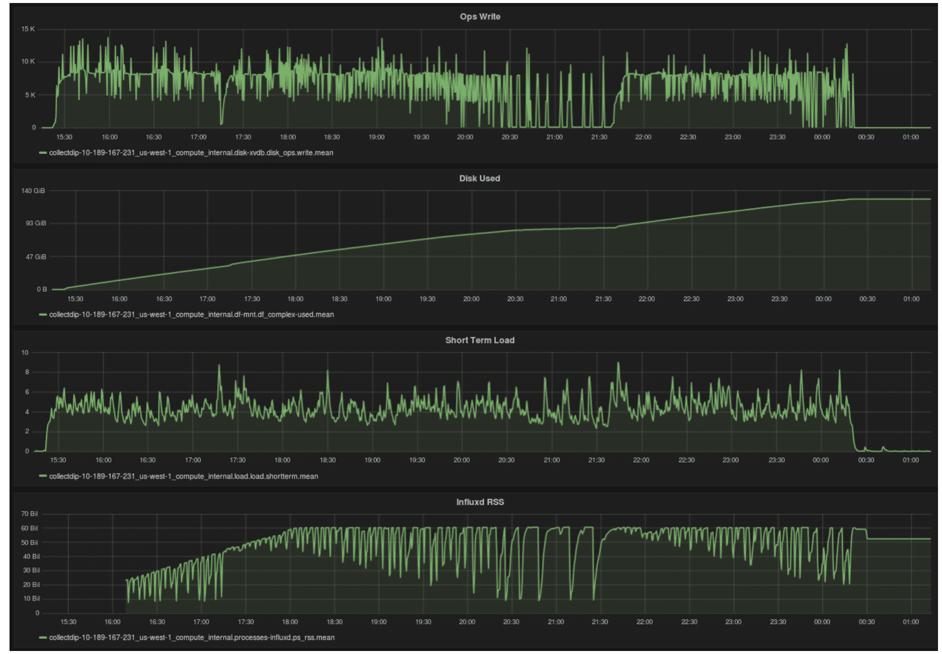
Structure tailored to predictable types of analysis
Manage quality, completeness, …

Server logs
Clickstream data
Social media feeds
Sensor and RFID data
…

'As is' volatile structure
Type(s) of analysis as yet unknown

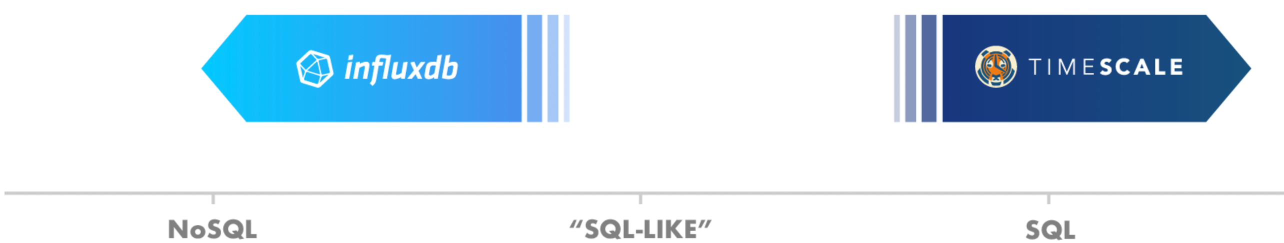
Time-series Databases

- Optimized for time-stamped or time series data
  - Server metrics and performance monitoring
  - Network data
  - Sensor data
  - Events, clicks, trades in a market…
  - Queries based on analysis tasks over time: windowing, aggregating, joining on time series



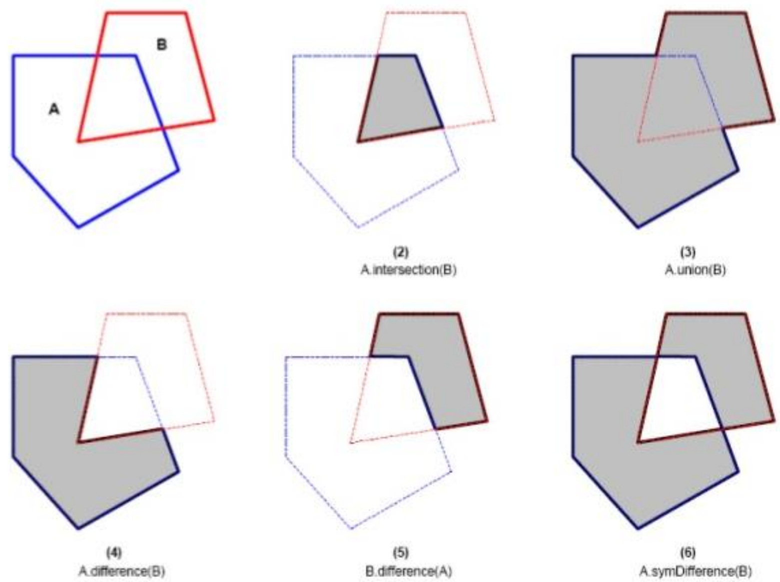
Time-series Databases

- Some popular vendors: InfluxDB, Kdb+, TimescaleDB
  - Some embrace SQL (which also supports many time-based operations in later standards) and extend it (TimescaleDB) whilst others take a NoSQL oriented approach with bespoke query languages (e.g. Flux in InfluxDB)
  - Like graph databases: a growing niche within NoSQL sphere

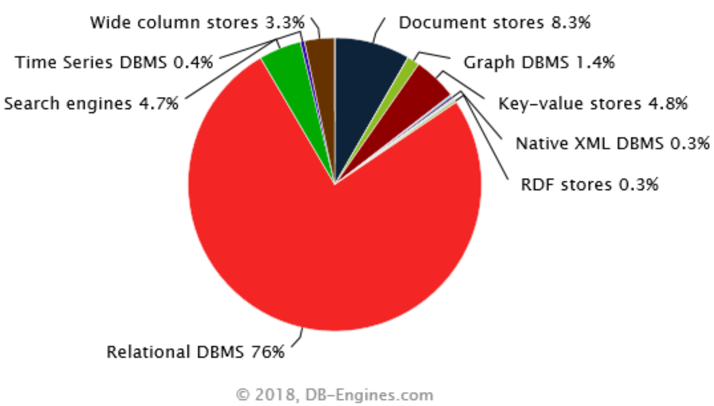


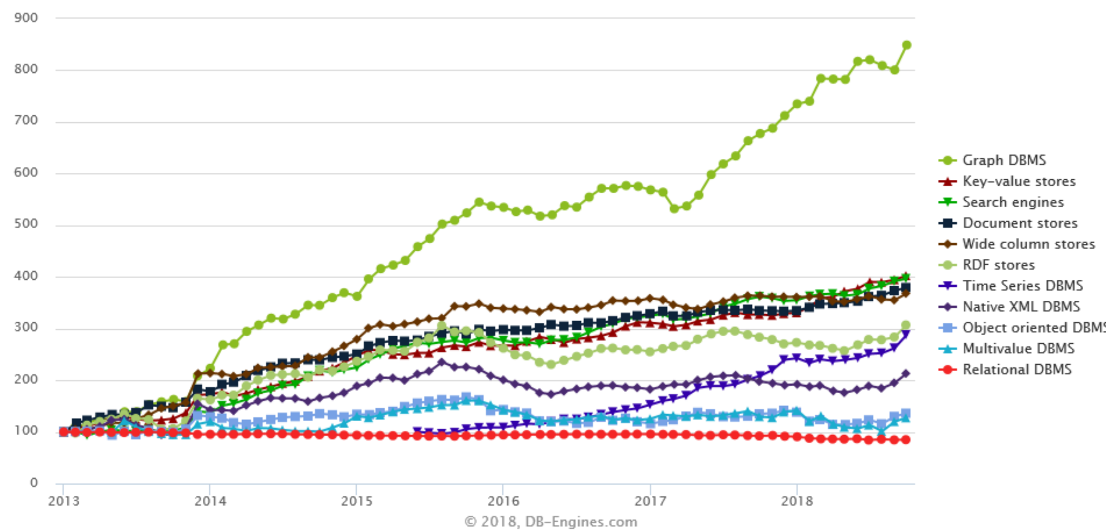
Geospatial Databases

- Optimized for storing and querying data that represents objects defined in a geometric space
  - Represented as points, lines, line segments, polygons, complex polygons with holes
  - Query operations based on spatial operators: spatial indexes to improve query speed
- Popular vendors: PostgreSQL with PostGIS, ESRI GIS Tools (Hadoop extension), Microsoft SQL Server (supports geospatial extensions), GIS tools such as ESRI



Evaluating NoSQL DBMSs





https://db-engines.com/en/

- Document stores already quite popular!
- Graph databases gaining interest!

Evaluating NoSQL DBMSs

- Many NoSQL implementations have yet to prove their true worth
- Some queries or aggregations particularly difficult, with MapReduce interfaces harder to learn and use
- Some early-adaptors of NoSQL were confronted with some sour lessons

- 'Use the right tool for the job !!'
  - Need for complex data model + enforce constraints ?
  - Need for rich query functionality ?
  - Need for graph like structures ?
  - Tradeoff between performance/scalability and consistency

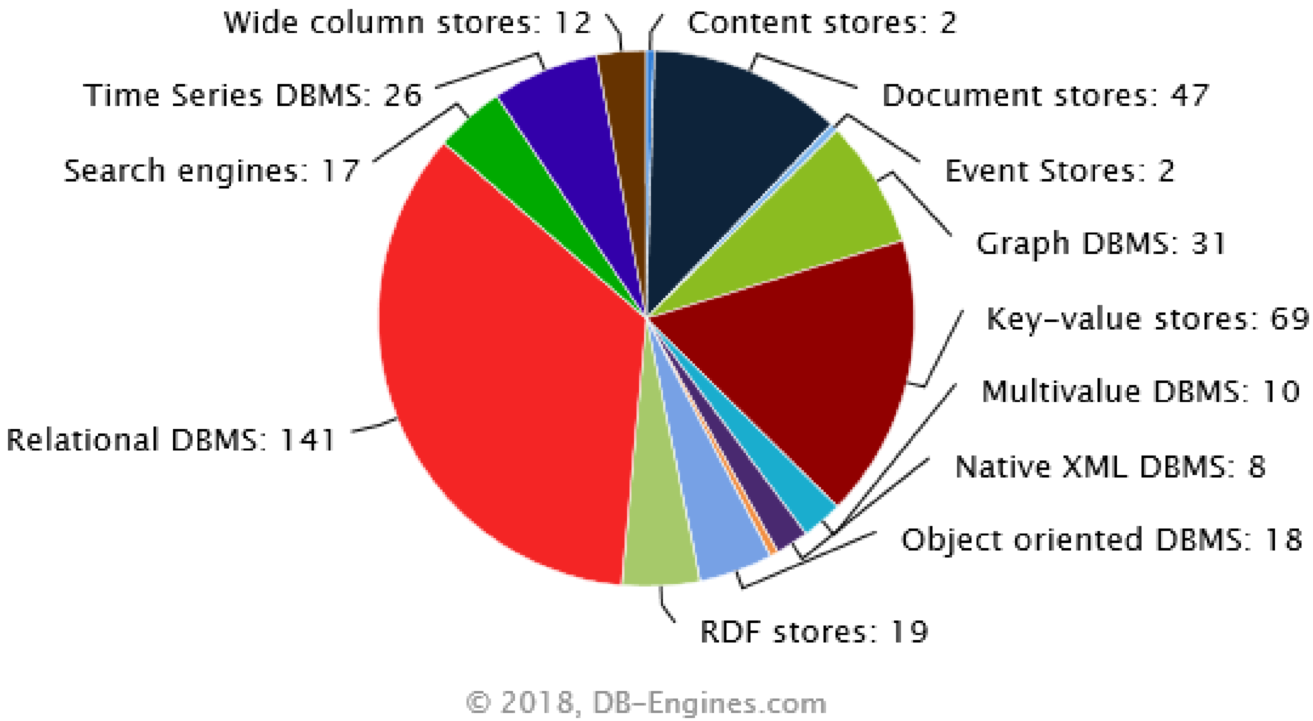
SQL: Dead and Buried?

Rank			DBMS	Database Model	Score		
Oct 2018	**Sep 2018**	**Oct 2017**			**Oct 2018**	**Sep 2018**	**Oct 2017**
1.	1.	1.	Oracle	Relational DBMS	1319.27	+10.15	-29.54
2.	2.	2.	MySQL	Relational DBMS	1178.12	-2.36	-120.71
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1058.33	+7.05	-151.99
4.	4.	4.	PostgreSQL	Relational DBMS	419.39	+12.97	+46.12
5.	5.	5.	MongoDB	Document store	363.19	+4.39	+33.79
6.	6.	6.	DB2	Relational DBMS	179.69	-1.38	-14.90
7.	↑ 8.	↑ 9.	Redis	Key-value store	145.29	+4.35	+23.24
8.	↓ 7.	↑ 10.	Elasticsearch	Search engine	142.33	-0.28	+22.09
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	136.80	+3.41	+7.35
10.	10.	↓ 8.	Cassandra	Wide column store	123.39	+3.83	-1.40

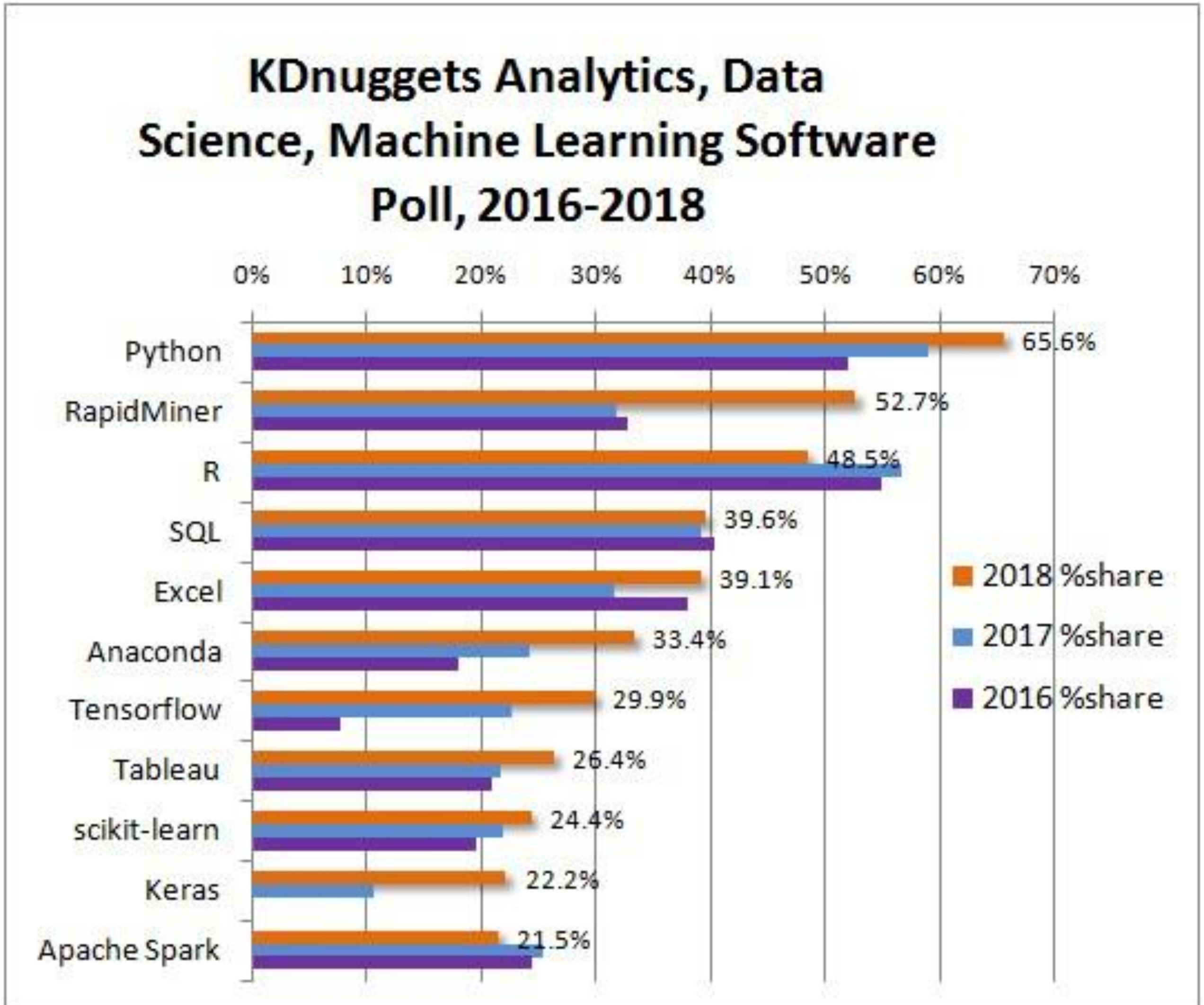
https://db-engines.com/en/ranking

SQL: Dead and Buried?

Number of systems per category:



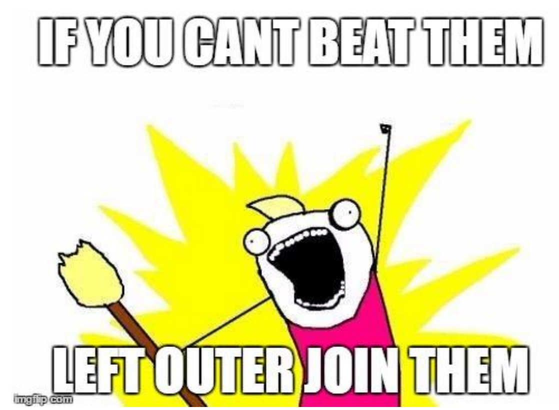
SQL: Dead and Buried?



SQL: Dead and Buried?

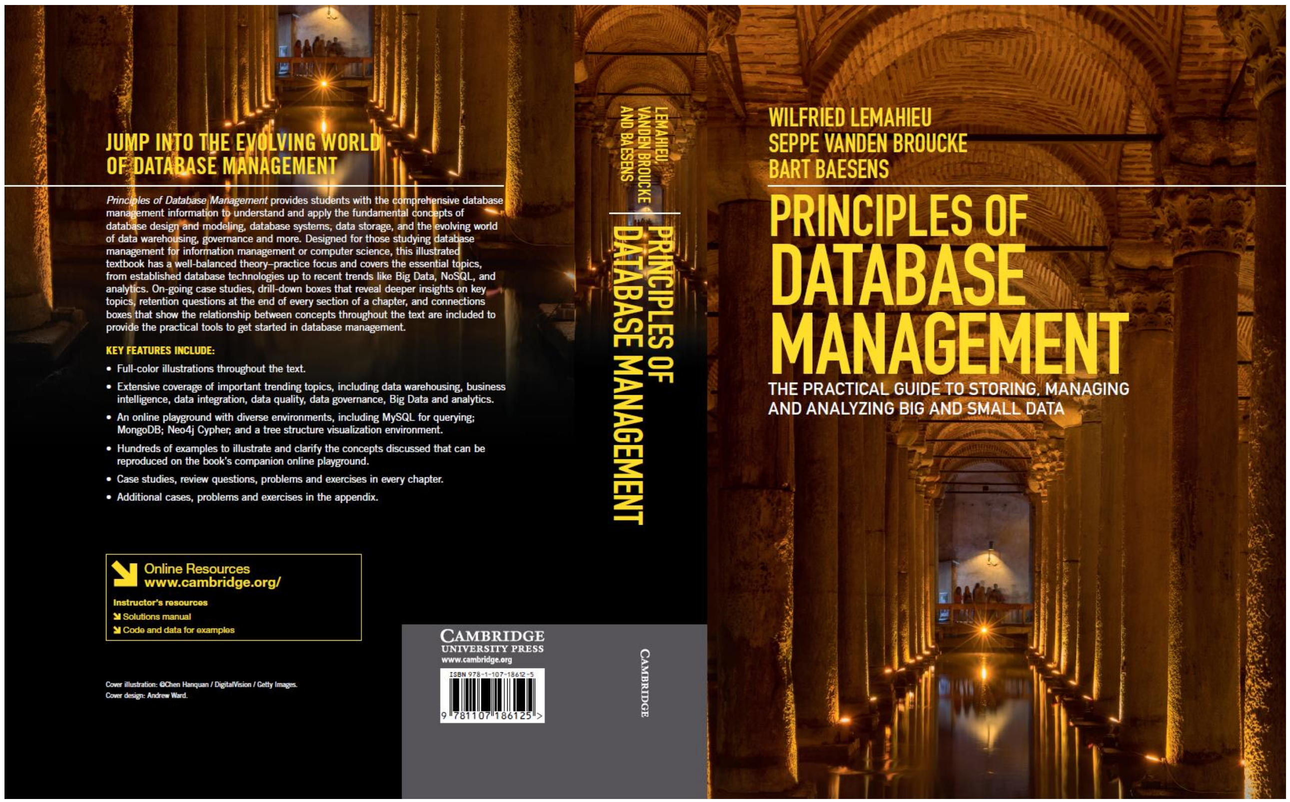
- NoSQL vendors start focusing again on robustness and durability *and on SQL like query languages*

- NewSQL: RDBMS vendors start implementing NoSQL by
  - Focusing on horizontal scalability and distributed querying
  - Dropping schema requirements
  - Support for nested data types or allowing to store JSON directly in tables
  - Support for MapReduce operations
  - Support for special data types, such as geospatial data

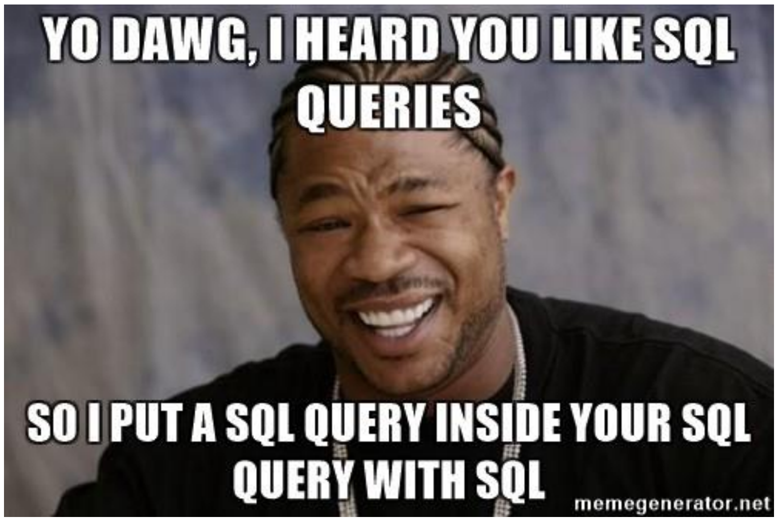


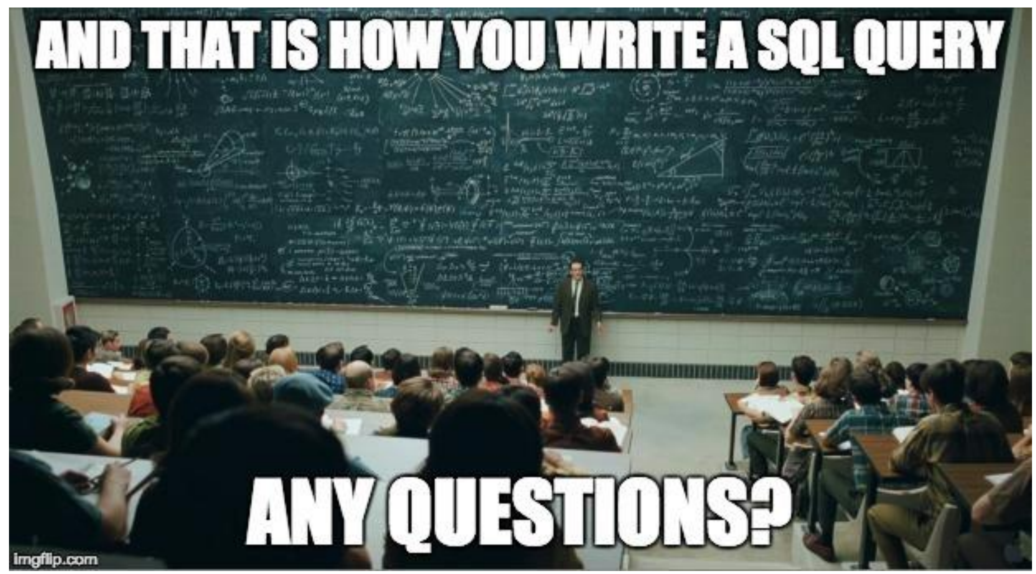
More Information?

www.pdbmbook.com



End


YO DAWG, I HEARD YOU LIKE SQL QUERIES
SO I PUT A SQL QUERY INSIDE YOUR SQL QUERY WITH SQL
memegenerator.net


AND THAT IS HOW YOU WRITE A SQL QUERY
ANY QUESTIONS?
imgflip.com