CS422
Database systems

Introduction

Data-Intensive Applications and Systems (DIAS) Laboratory
École Polytechnique Fédérale de Lausanne

“The world is one big data problem”
– Andrew McAfee

Some slides adapted from:

- Andy Pavlo
- CS-322

Course Overview

- Data management systems:
  - Design
  - Implementation
  - Principles

Prerequisites

- Required courses
  - CS-105: Introduction to object-oriented programming
  - CS-150: Discrete structures
  - **CS-322: Introduction to database systems**

- Recommended courses
  - CS-323: Operating systems

- Refresh your database knowledge

Course Outline

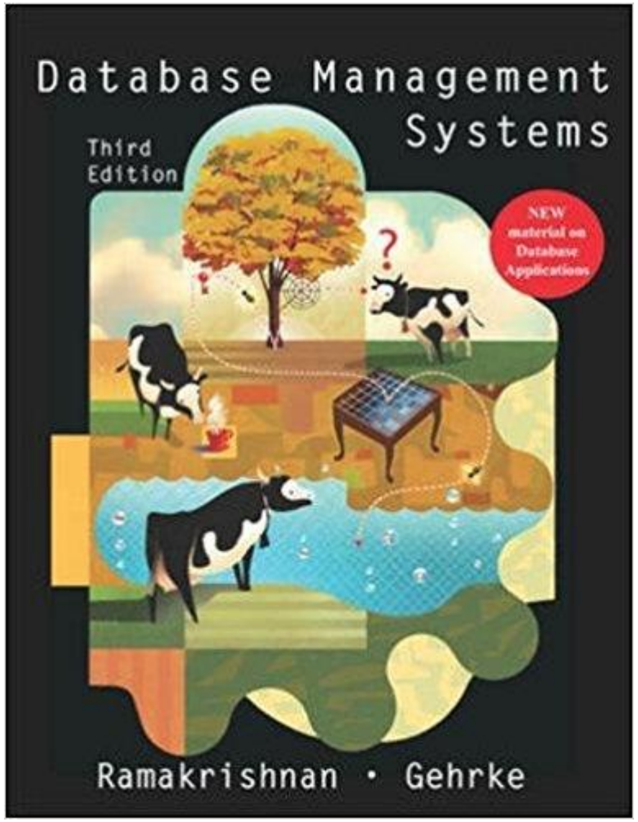
- Relational Databases
- Storage
- Execution
- Optimization
- Data Warehouses
- Distributed Computing
- Stream Processing
- Transaction Processing

Textbook & more

- **Database Management Systems**

3rd Edition

Ramakrishnan & Gehrke



- **Surveys on state-of-art**

- **Seminal Papers & Articles**

Grading Scheme

Projects (30%)

Midterm Exam (30%)

Final Exam (40%)

Projects

Project I: Build your own database engine

- Express database operators using Java
- 10% of overall grade

Project II: Develop distributed computing apps

- Batch, in-memory, and real-time analytics over Spark
- Spark (Java or Scala) API
- 20% of the overall grade

Exams

Mid-term Exam (April 4th)

Final Exam (May 29th)

Closed book.

Lecturers

Anastasia Ailamaki

Angelos Anadiotis

TAs

When: Wednesdays, 13:15, @ SG0213

Thursday, 10:15, @BC01

- Matt Olma
- Stella Giannakopoulou
- Periklis Chrysogelos
- Panos Sioulas

Exercise sessions

- Further discussion on material (esp. papers)
- Answering questions

DATABASE

An organized collection of inter-related data that models some aspect of the real world.

Databases are the core component of most computer applications.

DATABASE EXAMPLE

Create a database that models a digital music store

Things to store:

- Information about **Artists**
- What **Albums** those **Artists** released
- The **Tracks** on those **Albums**

Entity-relationship diagram

Artists have

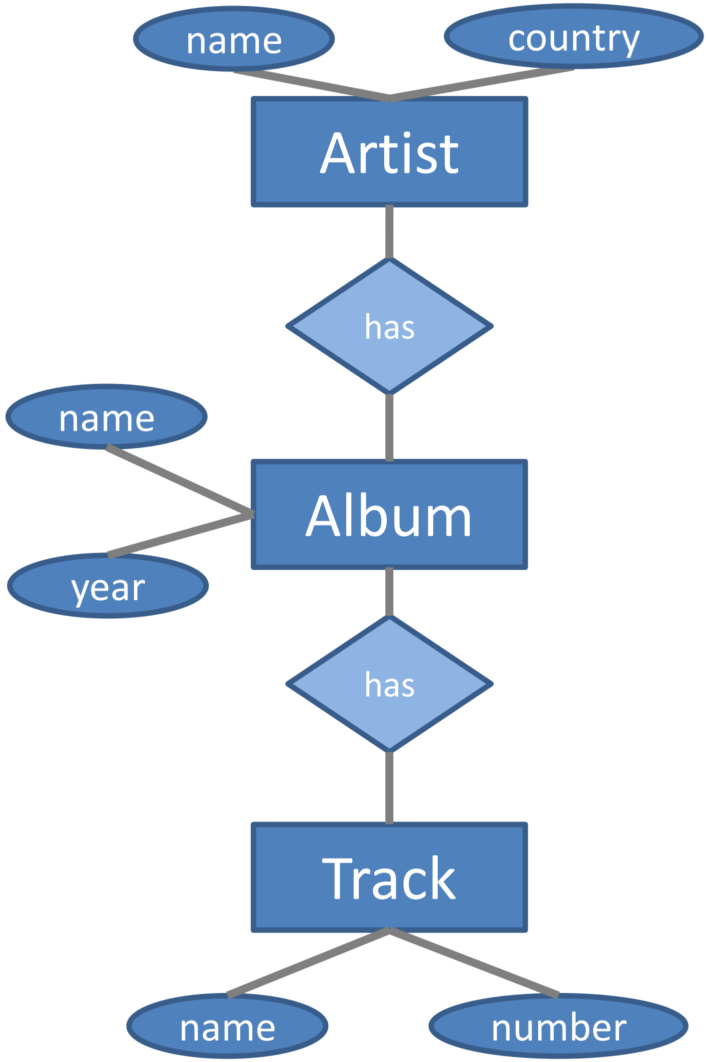
- name
- country of origin

Albums have

- name
- release year

Tracks have

- name
- number



Describe data via objects and their relationships

Storage Option : Flat CSV (Excel) files

- File per entity
- Application has to parse data for every record read/update

Artist(name, country)

```
“Mogwai”, “Scotland”
“Justin Bieber”,“Canada”
“Eluveitie”, “Switzerland”
```

Album(name, artist, year)

```
“Mr.Beast”,“Mogwai”,2006
“Atomic”,“Mogwai”,2015
“Helvetios”,Eluveitie”,2012
```

Storage Option : Flat CSV (Excel) files

- File per entity
- Application has to parse data for every record read/update

Album(name, artist, year)

```
“Mr.Beast”,“Mogwai”,2006
“Atomic”,“Mogwai”,2015
“Helvetios”,Eluveitie”,2012
```

Example:

Get albums published in 2012

```
for line in file:
  record = parse(line)
  if 2012 == int(record[2])
    print record[0]
```

Flat files: Data Integrity

- How do we ensure that the artist is the same for each album entry?
- What if somebody overwrites the album year with an invalid string?
- How do we store that there are multiple artists on an album?

Flat files: Implementation

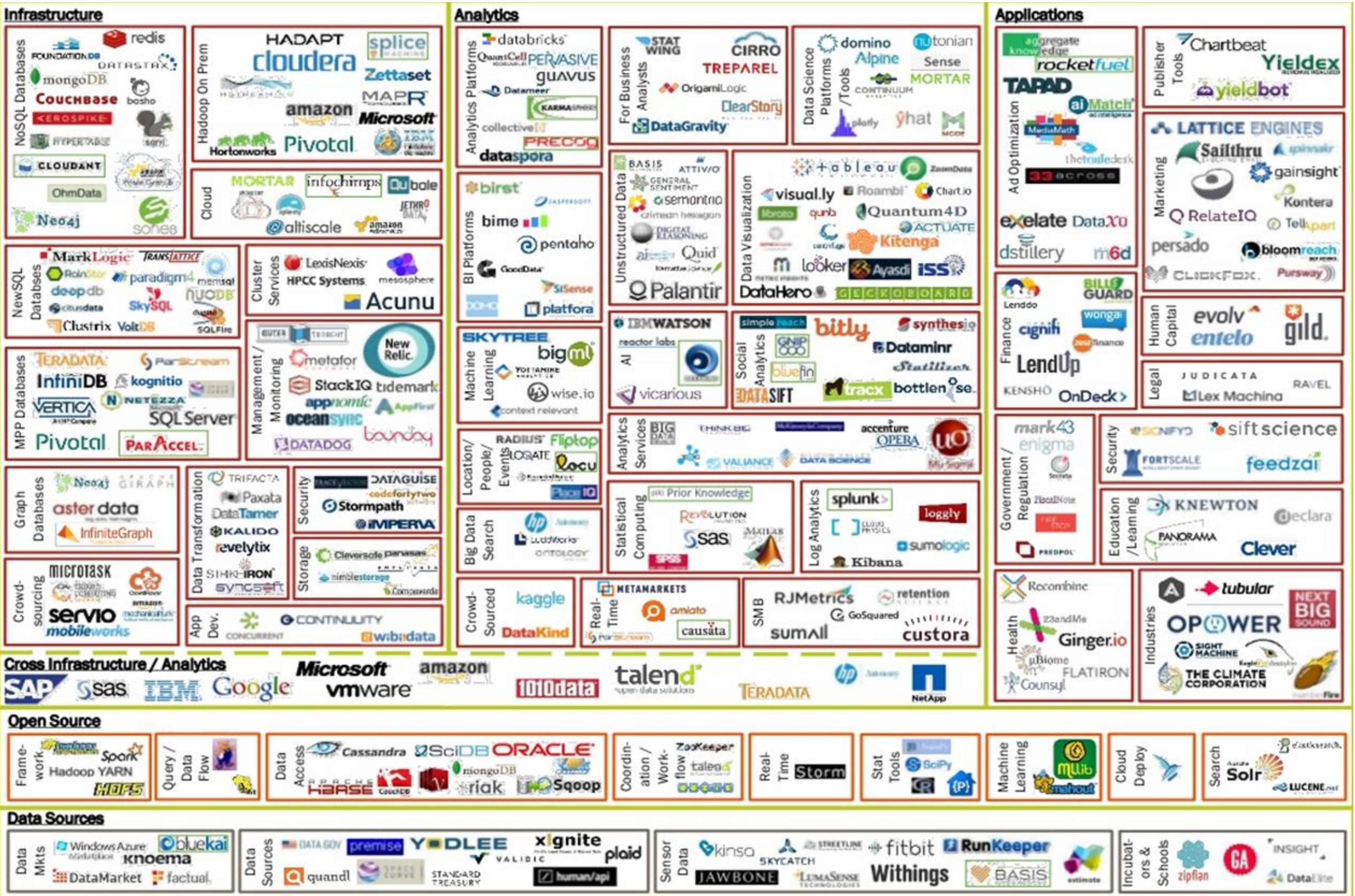
- How do you find a particular record?
- What if we now want to create a new application that uses the same database?
- What if two threads try to write to the same file at the same time?

Flat files: Durability

- What if the machine crashes while we're updating a record?
- What if we want to replicate the database on multiple machines for high availability?

Database Management System

- A DBMS is software that allows applications to store and analyze information in a database.
- A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases.
- DBMSs are used in (almost) every application, web site, software system you can think of.

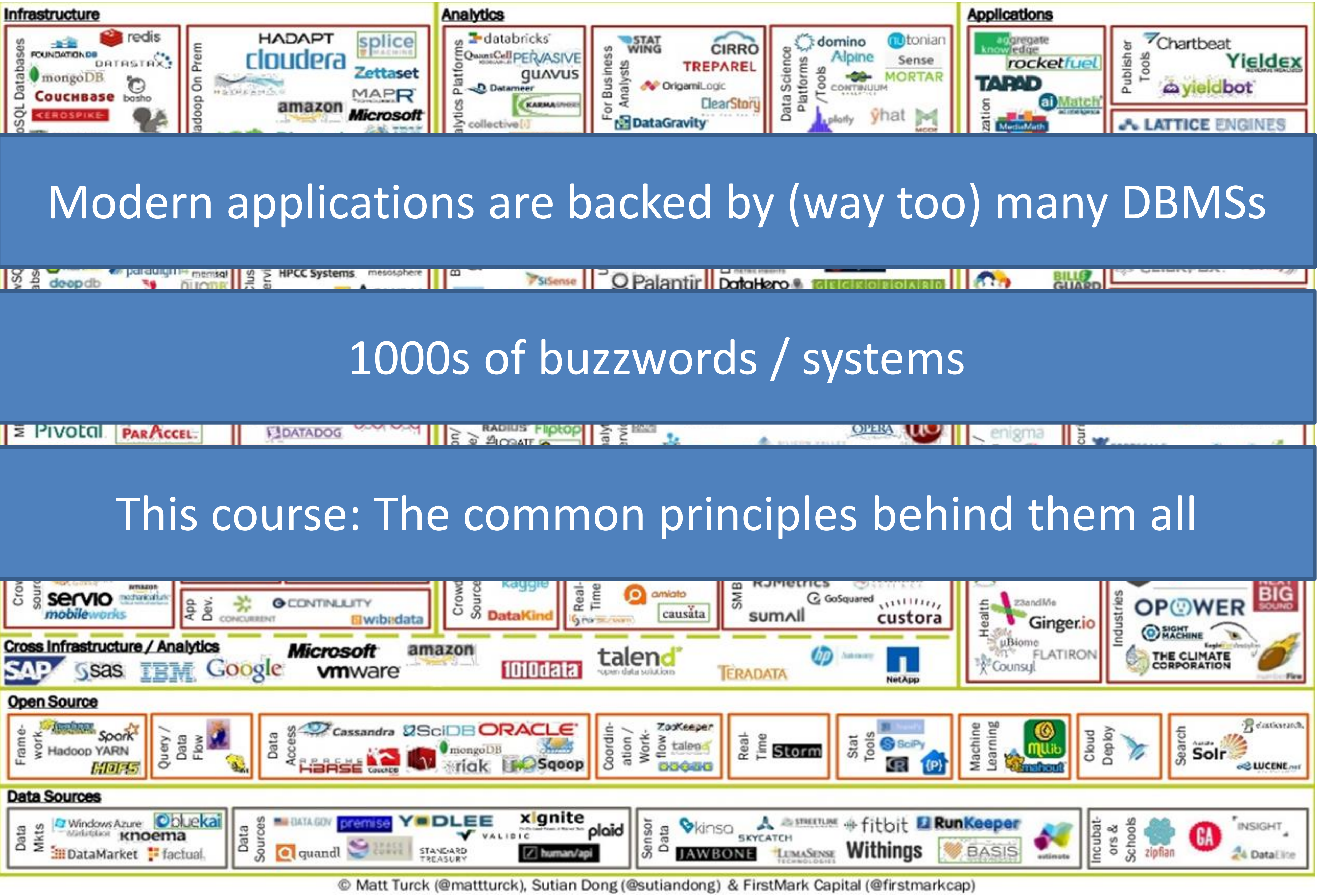


© Matt Turck (@mattturck), Sutian Dong (@sutiandong) & FirstMark Capital (@firstmarkcap)

Modern applications are backed by (way too) many DBMSs

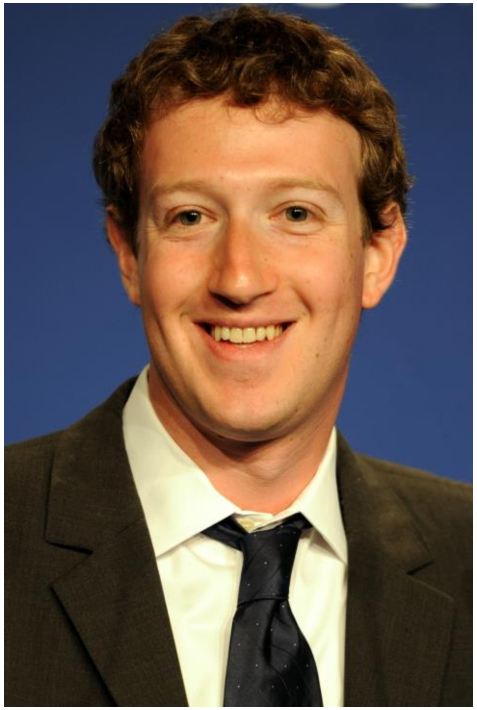
1000s of buzzwords / systems

This course: The common principles behind them all



Why data management principles matter

friendster®



Course Orientation

A broad perspective of
database management systems

- Database internals: storage, execution, optimization, transa
  column stores, optimizations, transaction & analytical processing
- Distributed computing: Big Data, data analytics, distributed databases
- Programming paradigms: MapReduce, stream processing

Database Management System

A DBMS is software that allows applications to store and analyze information in a database.

A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases.

DBMS Types: Target Workloads

On-line Transaction Processing

- Fast operations that only read/update a small amount of data each time.

On-line Analytical Processing

- Complex queries that read a lot of data to compute aggregates.

Hybrids do exist!

DBMS Types: Data Model

- Relational ← **Most DBMSs**
- Key/Value
- Graph
- Document
- Column-family

← **NoSQL**

- Array / Matrix ← **Machine Learning**
- Hierarchical
- Network

← **Rare**

Relational Model

A relation is an unordered set that contain the relationship of attributes that represent entities.

A tuple is a sequence of attribute values in the relation

Artist(name, country)

name	country
Mogwai	Scotland
Justin Bieber	Canada
Eluveitie	Switzerland

Relational Model: Primary Keys

A relation's ***primary key*** uniquely identifies a single tuple.

Some DBMSs support auto-generation of unique integer primary keys.

- SEQUENCE (SQL:2003)
- AUTO_INCREMENT (MySQL)

Artist(id, name, country)

id	name	country
123	Mogwai	Scotland
456	Justin Bieber	Canada
789	Eluveitie	Switzerland

Relational Model: Foreign Keys

A ***foreign key*** specifies that an attribute from one relation has to map to a tuple in another relation.

Artist(id, name, country)

id	name	country
123	Mogwai	Scotland
456	Justin Bieber	Canada
789	Eluveitie	Switzerland

Album(id, name, artist, year)

id	name	artist	year
11	Mr. Beast	Mogwai	2006
22	Atomic	Mogwai	2015

Album(id, name, artist_id, year)

id	name	artist_id	year
11	Mr. Beast	123	2006
22	Atomic	123	2015

Relational Model: Queries

The relational model is independent of any query language implementation.

SQL is the de facto standard.

```
for line in file:
  record = parse(line)
  if 2012 == int(record[2])
    print record[0]
```

```
SELECT name
FROM album
WHERE year = 2012
```

Data Manipulation Languages (DML)

How to store and retrieve information from a db.

(Procedural) Relational Algebra:
Query specifiers high-level strategy the DBMS should use to find the desired result

(Non-procedural) Relational Calculus:
The query specifies only what data is wanted and **not** how to find it

Data Manipulation Languages (DML)

How to store and retrieve information from a db.

(Non-procedural) Relational Calculus:
The query specifies only what data is wanted and **not** how to find it

(Procedural) Relational Algebra:
Query specifiers high-level strategy the DBMS should use to find the desired result

Relational Calculus

Tuple Relational Calculus:

Uses variables whose permitted values are **tuples**

$$\{t | t \in R \wedge t.a_{id} = 'a2' \wedge t.b\_id > 102\}$$

Domain Relational Calculus:

Uses variables to select attributes (instead of tuples).

$$\{< a_{id}, b\_id > | < a_{id}, b_{id} > \in R \wedge a_{id} = 'a2' \wedge b\_id > 102\}$$

Both are equivalent to relational algebra!

Relational Algebra

Fundamental operations to retrieve and manipulate tuples in a relation

- Based on set algebra

Each operator takes on or more relations as its inputs and outputs a new relation

- We can “chain” operators together to create more complex operations

Relational Algebra: 5 Basic Operations

- **Selection** (σ) Selects a subset of *rows* from relation (horizontal).
- **Projection** (π) Retains only wanted *columns* from relation (vertical).
- **Cross-product** ($\times$) Allows us to combine two relations.
- **Set-difference** ($-$) Tuples in r1, but not in r2.
- **Union** ($\cup$) Tuples in r1 and/or in r2.

Since each operation returns a relation, operations can be *composed!* (Algebra is "closed").

Example Schema and Instances

- Boats(*bid*: integer, *bname*: string, *color*: string)
- Sailors(*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
- Reserves(*sid*: integer, *bid*: integer, *day*:date)

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Simplest Relational Algebra Expression

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

$$Boats$$

Output

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

$$S2$$

Output

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Relational Algebra

- Selection & Projection
- Union, Set Difference & Intersection
- Cross product & Joins

Selection Operator: (σ)

- Selects rows that satisfy *selection condition*.
- ***Output schema*** of result is same as that of the input relation

S2

sid	sname	rating	age
~~28~~	~~yuppy~~	~~9~~	~~35.0~~
31	Lubber	8	55.5
44	guppy	5	35.0
~~58~~	~~Rusty~~	~~10~~	~~35.0~~

$$\sigma_{rating<9}(S2)$$

Output

sid	sname	rating	age
31	Lubber	8	55.5
44	guppy	5	35.0

S2

sid	sname	rating	age
~~28~~	~~yuppy~~	~~9~~	~~35.0~~
31	Lubber	8	55.5
~~44~~	~~guppy~~	~~5~~	~~35.0~~
~~58~~	~~Rusty~~	~~10~~	~~35.0~~

$$\sigma_{\substack{rating<9\wedge \\ age>50}}(S2)$$

Output

sid	sname	rating	age
31	Lubber	8	55.5

Projection Operator (π)

- Retains only attributes that are in the *projection list.*
- **Output schema** is exactly the fields in the projection list, with the same names that they had in the input relation.

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

$$\pi_{sname,rating}(S2)$$

Output

sname	rating
yuppy	9
Lubber	8
guppy	5
Rusty	10

Projection Operator (π): Duplicate Elimination

- Projection operator has to *eliminate duplicates* (How do they arise? Why remove them?)
- Relational algebra is set based while SQL is bag (multiset) based

$S2$

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

$$\pi_{age}(S2)$$

$Output$

age
35.0
55.5

Composing multiple operators

- Output of one operator can become input to another operator

S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

Output

sname	rating
yuppy	9
Rusty	10

$$\pi_{sname,rating}\left(\sigma_{rating>8}(S2)\right)$$

Relational Algebra

- Selection & Projection
- Union, Set Difference & Intersection
- Cross product & Joins

Union and Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - “Corresponding” fields have the same type.
- For which, if any, is duplicate elimination required?

Union operator ($\cup$)

$S1$

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

$S2$

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

$S1 \cup S2$

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

Set Difference Operator (–)

$S1$

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

$S2$

sid	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

$S1 - S2$

sid	sname	rating	age
22	Dustin	7	45.0

$S2 - S1$

sid	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

Compound Operator: Intersection

- Alongside the 5 basic operators, there are several additional **Compound Operators**:
  - These add no computational power to the language, but are useful shorthands.
  - Can be expressed solely with the basic ops.
- Intersection takes two input relations, which must be union-compatible.
- Q: How to express it using basic operators?

$$R \cap S = R - (R - S)$$

Intersection operator (∩)

$S1$

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

$S2$

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

$S2 \cap S1$

<u>sid</u>	sname	rating	age
31	Lubber	8	55.5
58	Rusty	10	35.0

Relational Algebra

- Selection & Projection
- Union, Set Difference & Intersection
- Cross product & Joins

Renaming Operator (ρ)

- Renames the list of attributes specified in the form of oldname → newname or position → newname
- **Output schema** is same as input except for the renamed attributes.
- Returns same tuples as input
- Can also be used to rename the name of the output relation

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

$$\rho_{bname \to boatname, color \to boatcolor}(Boats)$$

bid	boatname	boatcolor
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

$$\rho_{2 \to boatname, 3 \to boatcolor}(Boats)$$

Cross-Product (×)

- S1 x R1: Each row of S1 paired with each row of R1.
- Q: How many rows in the result?
- **Result schema** has one field per field of S1 and R1, with field names "inherited" if possible.
  - *May have a naming conflict*: Both S1 and R1 have a field with the same name.
  - In this case, can use the *renaming operator*:

$$\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$$

Call C the result of S1×R1 and respectively rename the 1st & 5th fields of C to sid1 & sid2

Cross-Product Example

$S1 \times R1$

sid	sname	rating	age	sid	bid	day
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

$S1$

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

$R1$

<u>sid</u>	<u>bid</u>	day
22	101	10/10/96
58	103	11/12/96

$\rho_{1 \to sid1, 5 \to sid2}(S1 \times R1)$

sid1	sname	rating	age	sid2	bid	day
22	Dustin	7	45.0	22	101	10/10/96
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	22	101	10/10/96
31	Lubber	8	55.5	58	103	11/12/96
58	Rusty	10	35.0	22	101	10/10/96
58	Rusty	10	35.0	58	103	11/12/96

Compound Operator: Join

- Joins are compound operators involving cross product, selection, and (sometimes) projection.
- Most common type of join is a **natural join** (often just called "join"). R⋈S conceptually is:
  - Compute R × S
  - Select rows where attributes that appear in both relations have equal values
  - Project all unique attributes and one copy of each of the common ones.
- Note: Usually done much more efficiently than this.
- Useful for putting "normalized" relations back together.

Natural Join Example

$$\pi_{S1.sid,sname,\ldots}\left(\sigma_{S1.sid=R1.sid}(\boldsymbol{S1} \times \boldsymbol{R1})\right)$$

S1

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

R1

<u>sid</u>	<u>bid</u>	day
22	101	10/10/96
58	103	11/12/96

sid	sname	rating	age	~~sid~~	bid	day
22	Dustin	7	45.0	~~22~~	101	10/10/96
~~22~~	~~Dustin~~	~~7~~	~~45.0~~	~~58~~	~~103~~	~~11/12/96~~
~~31~~	~~Lubber~~	~~8~~	~~55.5~~	~~22~~	~~101~~	~~10/10/96~~
~~31~~	~~Lubber~~	~~8~~	~~55.5~~	~~58~~	~~103~~	~~11/12/96~~
~~58~~	~~Rusty~~	~~10~~	~~35.0~~	~~22~~	~~101~~	~~10/10/96~~
58	Rusty	10	35.0	~~58~~	103	11/12/96

$$\boldsymbol{S1} \bowtie \boldsymbol{R1}$$

sid	sname	rating	age	bid	day
22	Dustin	7	45.0	101	10/10/96
58	Rusty	10	35.0	103	11/12/96

Condition Join or Theta-Join

$$R \bowtie_C C = \sigma_C(R \times S)$$

- **Output schema** same as that of cross-product.
- May have fewer tuples than cross-product.

$S1$

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

$R1$

sid	bid	day
22	101	10/10/96
58	103	11/12/96

$$S1 \bowtie_{S1.sid<R1.sid} R1$$

sid	sname	rating	age	sid	bid	day
22	Dustin	7	45.0	58	103	11/12/96
31	Lubber	8	55.5	58	103	11/12/96

Equi-Join

- Special case of theta-join: condition *c* contains only conjunction of *equalities*.
- Find **all pairs** of sailors in S2 who have same age.

S2

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

- $\boldsymbol{S1} \bowtie_{S1.age=S2.age} \boldsymbol{S2}$
- $\boldsymbol{S1} \bowtie_{age=age2} \rho_{age\to age2}(\boldsymbol{S2})$
- $\sigma_{\boldsymbol{sid1!=sid2}}\left(\boldsymbol{S1} \bowtie_{age=age2} \rho_{\substack{age\to age2 \\ sid\to sid2}}(\boldsymbol{S2})\right)$
- $\sigma_{\boldsymbol{sid1<sid2}}\left(\boldsymbol{S1} \bowtie_{age=age2} \rho_{\substack{age\to age2 \\ sid\to sid2}}(\boldsymbol{S2})\right)$

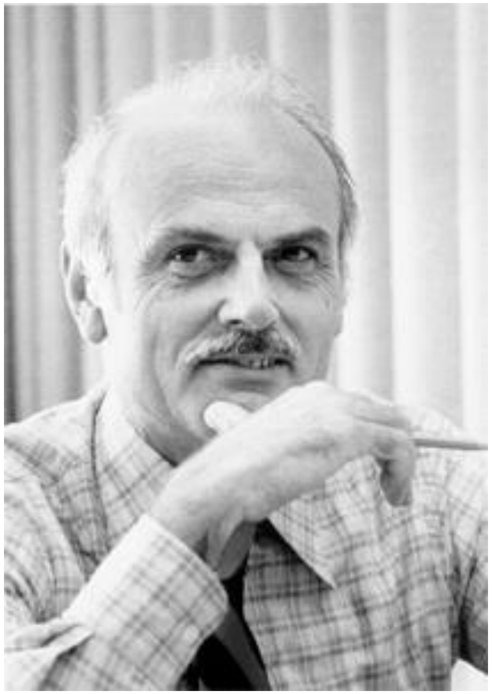

OTHERSIDE
RED HOT CHILI PEPPERS

The SQL Wars

Round I: CODASYL vs SQL

Charles Bachman

- Access db using COBOL
- Network data model
  - Complex queries
  - Easily corrupted
- No data independence



Ted Codd

- Introduce relational model
- Store db in simple structures
- Access data via high-level lang
- Physical storage up to DBMS

The great debate @ SIGMOD 1975

COBOL crew:

- “Relational model is too mathematical”
- “Rel. model cannot be implemented efficiently”
- “Transaction processing systems want (procedural) tuple-at-a-time operations; not declarative

Relational crew:

- “CODASYL proposal too complicated”
- “Queries too hard to program”
- “CODASYL has no formal foundation”

IBM DB2 used the relational model; the rest followed

Round II: The (No)SQL Wars

Data Models

- Relational
- Key/Value
- Graph
- Document
- Column-family
- Array / Matrix
- Hierarchical
- Network

NoSQL

Birth of NoSQL (Not-Only SQL)

- Non-relational data model
  - Claim: Non-relational data (unstructured, semistructured, hierarchical..)
- Flexible or evolving schema
  - Claim: “Schema-upfront” DBMS approach not agile
- Simple call-level interface (not SQL)
  - Claim: Cater for specific application (memcached), complex analytics
- **B**asically **A**vailable **S**oft state **E**ventual consistency
  - Claim: ACID serializability doesn’t scale
- Horizontal scaling
  - Claim: Relational DBMS only scales vertically

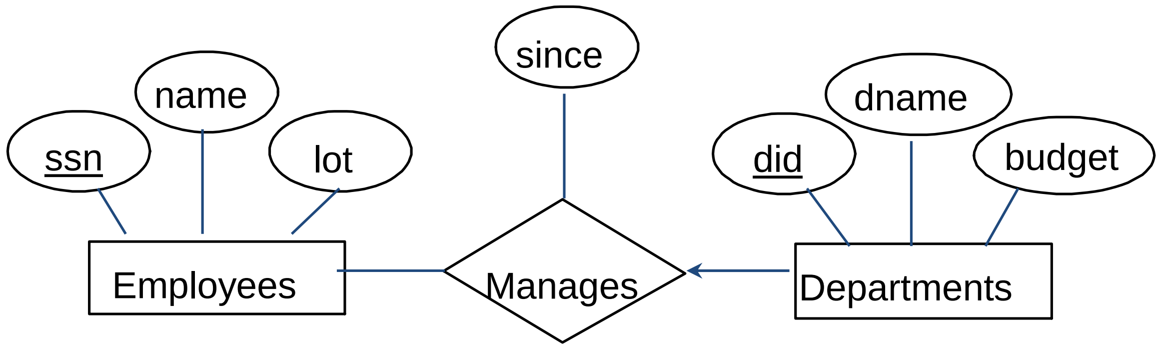
NoSQL systems: Data models

- Key-Value stores
  - Ex: Voldemort, Riak, Redis
  - Structural aspect: Associative array
  - Manipulation aspect: get, put
- Document stores
  - Ex: SimpleDB, <u>CouchDB</u>, MongoDB
  - Structural aspect: “Document” (= <scalar> <document/list>)
  - Manipulation aspect: “View” (queries in javascript)

NoSQL systems: Data models

- Extensible record stores (wide column stores)
  - Ex: BigTable, Hbase, HyperTable, Cassandra
  - Structural aspect: “Tables” rows and columns
  - Rows are more like documents than tuples
    - variable #columns
    - Rows grouped into a table
  - Columns similar to attributes in relational model
    - Scalar, Usually no nested types
  - Manipulation aspect: record-level get/put

NoSQL vs Relational: Example

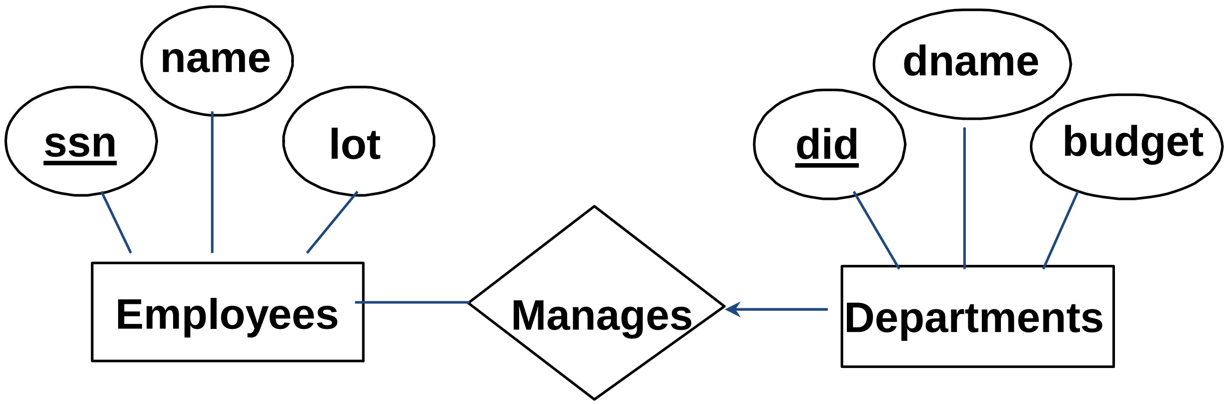


- One table for Employees, Department entities
- Merge Manages relationship with department

```
CREATE TABLE Employee (
 ssn CHAR(11),
 name CHAR(20),
 lot  INTEGER)
```

```
CREATE TABLE  Dept_Mgr(
 did  INTEGER,
 dname  CHAR(20),
 budget  REAL,
 ssn  CHAR(11),
 since  DATE)
```

Same data in key-value store

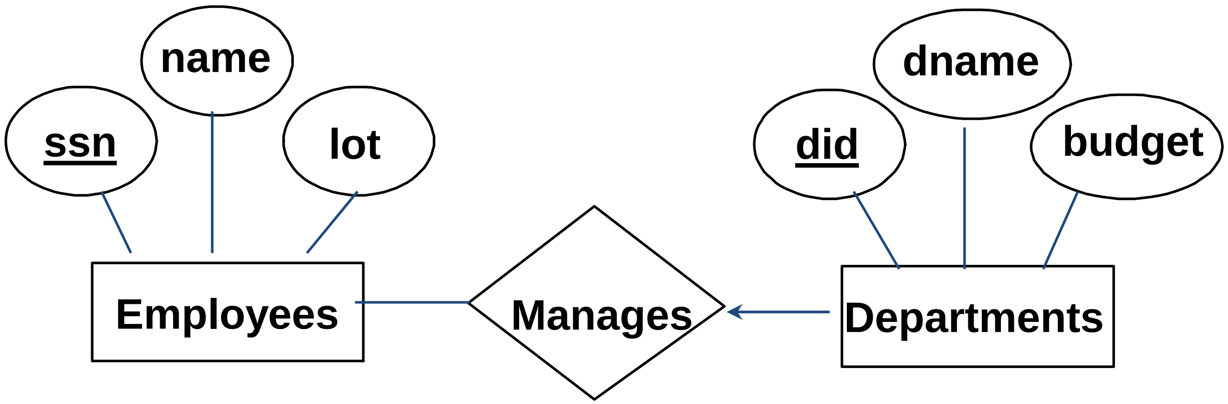


- Store all employee info in employee object
- Store employee object as the "value" for a "key"

```
class Employee
{
 int ssn;
 string name;
 int lot;
 list<Department> managedDepts;
}
```

```
private class Department
{
 int did;
 string dname;
 int budget;
}
```

Same data in key-value store



- Store all employee info in employee object
- Store employee object as the "value" for a "key"

123-22-3666 →	Attishoo	48	[(51,IT,1000),(56,Accounting,3000)]
231-31-5368 →	Smiley	22	[(51,IT,1000)]
131-24-3650 →	Smethurst	35	[]

Same data in document store

- Store data + schema in a JSON document

```
JSON Document {
  "id": 123223666, "name": "Attishoo",
  "manages": [{"did": 51, "name": "IT",
               "budget": 1000},
              {"did": 56, "name": "Accounting",
               "budget": 3000}]
}
```

NoSQL: Not-so-new SQL (atleast wrt data model)

- Object-oriented databases
  - Persisting objects in OO applications, supports nesting
  - Ex: Gemstone
- Graph databases
  - Supports complex graph traversals, graph querying
  - Ex: Neo4j
- Relational++
  - Object-relational extensions
  - Text, XML, JSON, graph extensions to SQL/relational DBMS

NoSQL: Not-so great SQL (atleast wrt features)

BLOG@CACM

The "NoSQL" Discussion has Nothing to Do With SQL

By Michael Stonebraker
November 4, 2009
Comments (12)

VIEW AS: SHARE:

Recently, there has been a lot of buzz about "No SQL" databases. In fact there are at least two conferences on the topic in 2009, one on each coast. Seemingly this buzz comes from people who are proponents of:

• document-style stores in which a database record consists of a collection of (key, value) pairs plus a payload. Examples of this class of system include CouchDB and MongoDB, and we call such systems **document stores** for simplicity

• key-value stores whose records consist of (key, payload) pairs. Usually, these are implemented by distributed hash tables (DHTs), and we call these **key-value stores** for simplicity. Examples include Memcachedb and Dynamo.

In either case, one usually gets a low-level record-at-a-time DBMS interface, instead of SQL. Hence, this group identifies itself as advocating "No SQL."

"It's all about performance or flexibility, not about SQL"

Conclusion

- DBMS used to maintain, query large datasets
- Most DBMS have a relational flavor,
  for legacy / performance reasons
- This course:
  DBMS principles, and
  DBMS evolution to BDMS (“big data management systems”)