Systems for large-scale data management

1

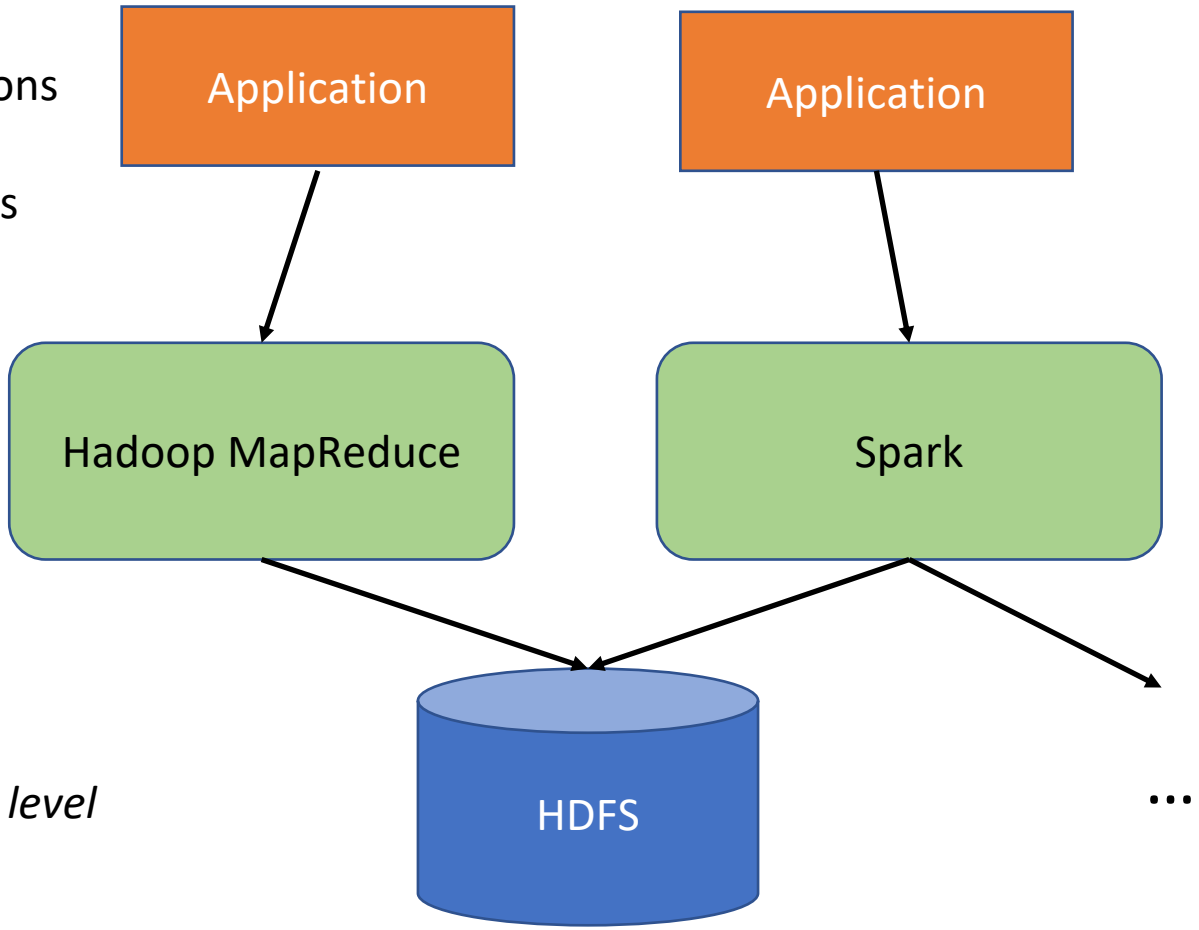
Introduction

Large-scale distributed (cluster) computing

- Work on clusters
- Provide a framework that
  - hides the complexity of distribution to developers (load balancing, fault tolerance)
  - provides proprietary data storage capabilities, in terms of distributed file systems
  - focuses on the fundamental nuggets of the computation, supporting new computation models, for processing data in parallel
- Principles
  - Scale “out", not “up"
  - Move processing to data (data locality)
  - Focus on (read) batch access
  - Tailored to *analytical processing*

Large-scale (cluster) computing

logical level
inside applications
in terms of
(key-value) pairs



physical level

Is this enough?

- In 70's, relational database management systems have been introduced with the aim of overcoming the limitations of file systems for data management

Can the (relational) data management system technology be a reference technology also for large-scale data management?

Relational DBMSs (RDBMSs)

Mostly standard

Clear separation between logical (relational model) and physical levels (files)

Declarative languages at the logical level (SQL)

Well established technology

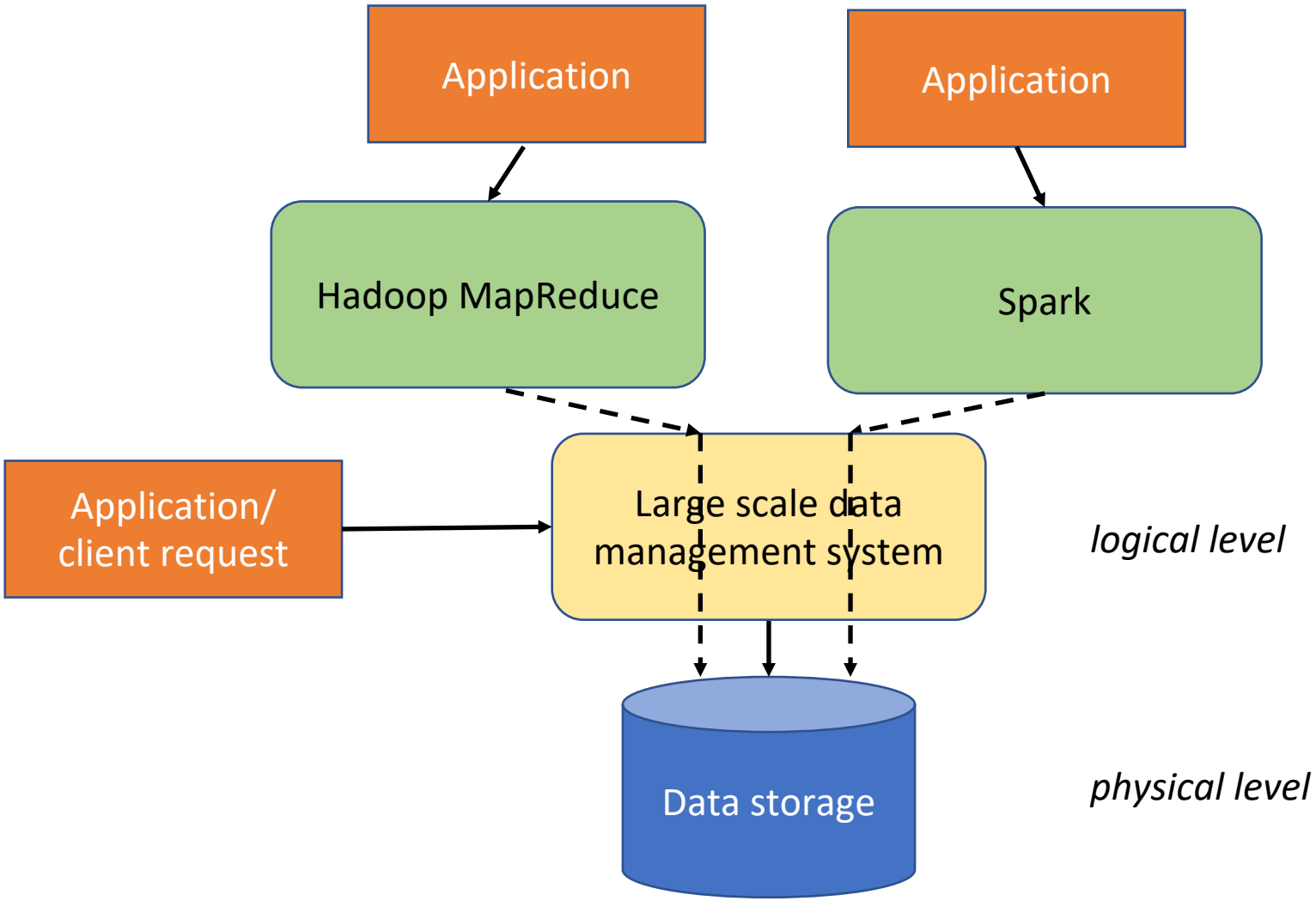
Are RDBMSs enough in a large-scale environment?

- No!
- Traditional RDBMSs can be made to run over clusters, but weren't really designed for it
- Typical DBMS functionalities can be extended to distributed architectures
- When considering large-scale distributed architectures, traditional algorithms and protocols have to be extended
  - Very complex algorithms and protocols
  - New issues to take into account (e.g., replica consistency) – very costly

Large-scale data management

- Work on clusters
- Provide a framework that
  - hides the complexity of distribution to developers (load balancing, fault tolerance)
  - provides data storage capabilities (physical level), that can be used by large-scale data processing frameworks as well
  - provides a logical level, partially independent from the physical one
  - focuses on data representation and querying
- Principles
  - Scale “out", not “up"
  - Move processing to data (data locality)
  - Focus also on read and write random access
  - Tailored to transactional (and analytical) *processing*

Large-scale data management

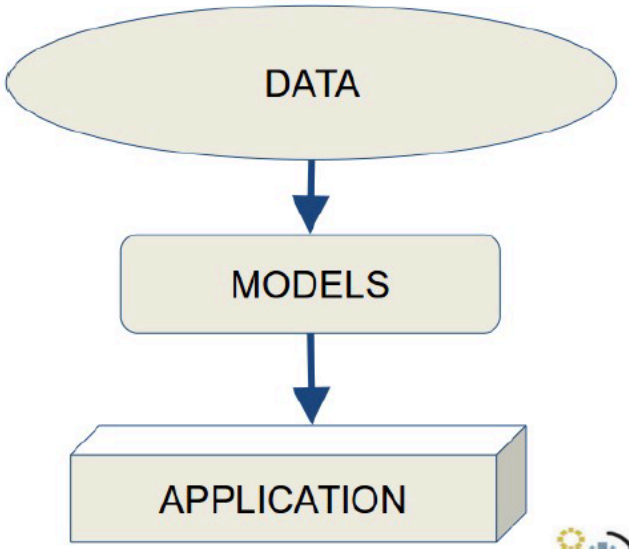


Main differences

Relational data management	Large-scale data management
1. Integration database	1. Application database
2. Rigid schema, structured data	2. Flexible schema, possibly unstructured data
3. Flat data	3. More complex data
4. Tailored to normalization and join	4. Joins are an issue, normalization is no more a reference principle
5. Fully featured declarative language (SQL)	5. Mainly procedural code

Main differences:
(1) integration vs application databases

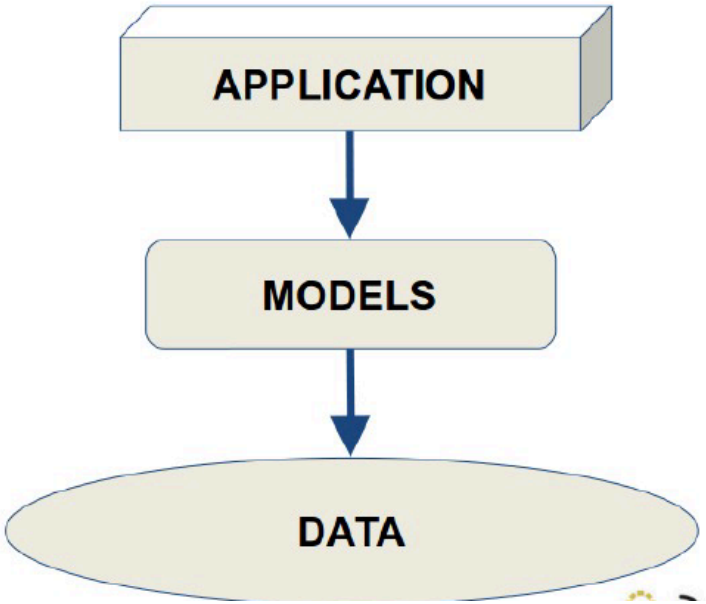
- Integration databases
  - One database may serve many applications, all access the same relational schema (logical level)
  - The schema is usually more complex than any single application needs
  - Different teams access different views of data
  - Complex integrity constraint checking and access control at the DBMS level



Workflow in integration DB

Main differences:
(1) integration vs application databases

- Application databases
  - Directly accessed by a single application
  - Data exchange between applications based on specific format (e.g., XML or JSON)
  - Rather than using RDBMS for everything, developers can choose the right DB for the right application
  - Only the team using the application needs to know about the database structure, which makes it much easier to maintain and evolve the schema
  - Since the application team controls both the database and the application code, many features typically supported by DBMSs (integrity checking, access control) can be put in the application code



Workflow in application DB

Main differences:
(2) rigid vs flexible schema

- In a relational database, all the tuples inside one relation share the same schema
- Trend accelerated by the decentralization and individualization of content generation (crowdsourced data), leads to the need for relaxing the notion of schema
- Different records of the same collection might contain different fields
- A field might hold several values

- Semi-structured data representation

Example

- In the relational model, the only way to cope with flexibility is to
  - Identify in advance all possibile properties - not always possible
  - Rely on null values – not always convenient

Product: iPhone 12
Price: 1.100
Camera: Fine
Screen: Very good
Accessories: {Headphone, Case, ….}

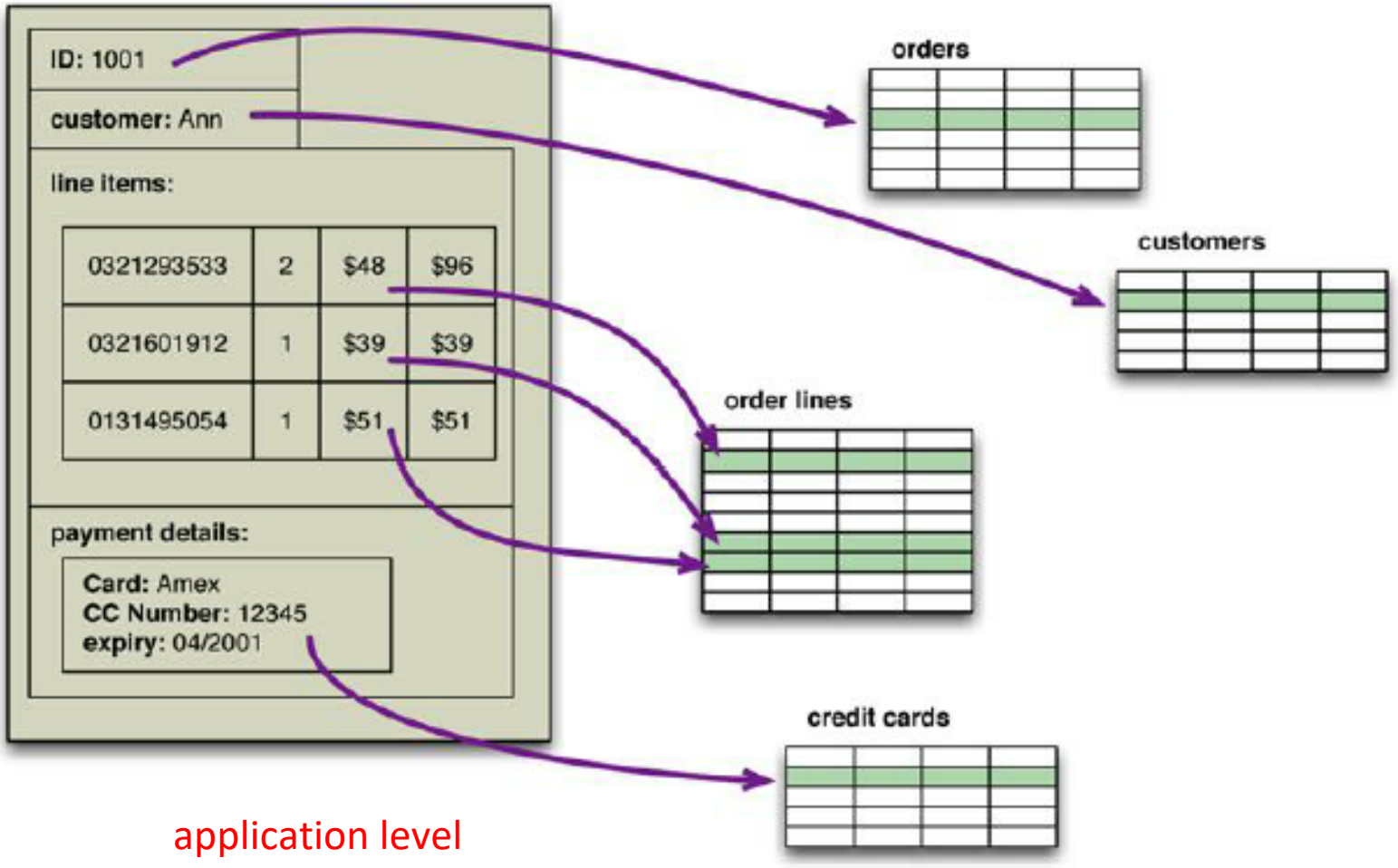
Product: iPhone 12
Price: 1.000
Camera: Excellent
Screen: Poor in sunshine
Operating system: Easy to use

Product	Price	Camera	Screen	Accessories	Operating System
IPhone 12	1.100	Fine	Very good	Headphone, Case, …	NULL
IPhone 12	1.000	Excellent	Poor in sunshine	NULL	Easy to use

Main differences:
(3) flat vs complex data

- In the relational model, each attribute value belongs to an atomic domain: number, string, date,..
- No sets, no lists
- But programming languages usually rely on more complex models (e.g., object-based models)
- Impedence mismatch between the persistent data model of the database (e.g., relational) and the in-memory data structures used by programs (e.g., object-based)
- In application databases, the relation between application and database is tighter, thus the database model should be closer to the application model

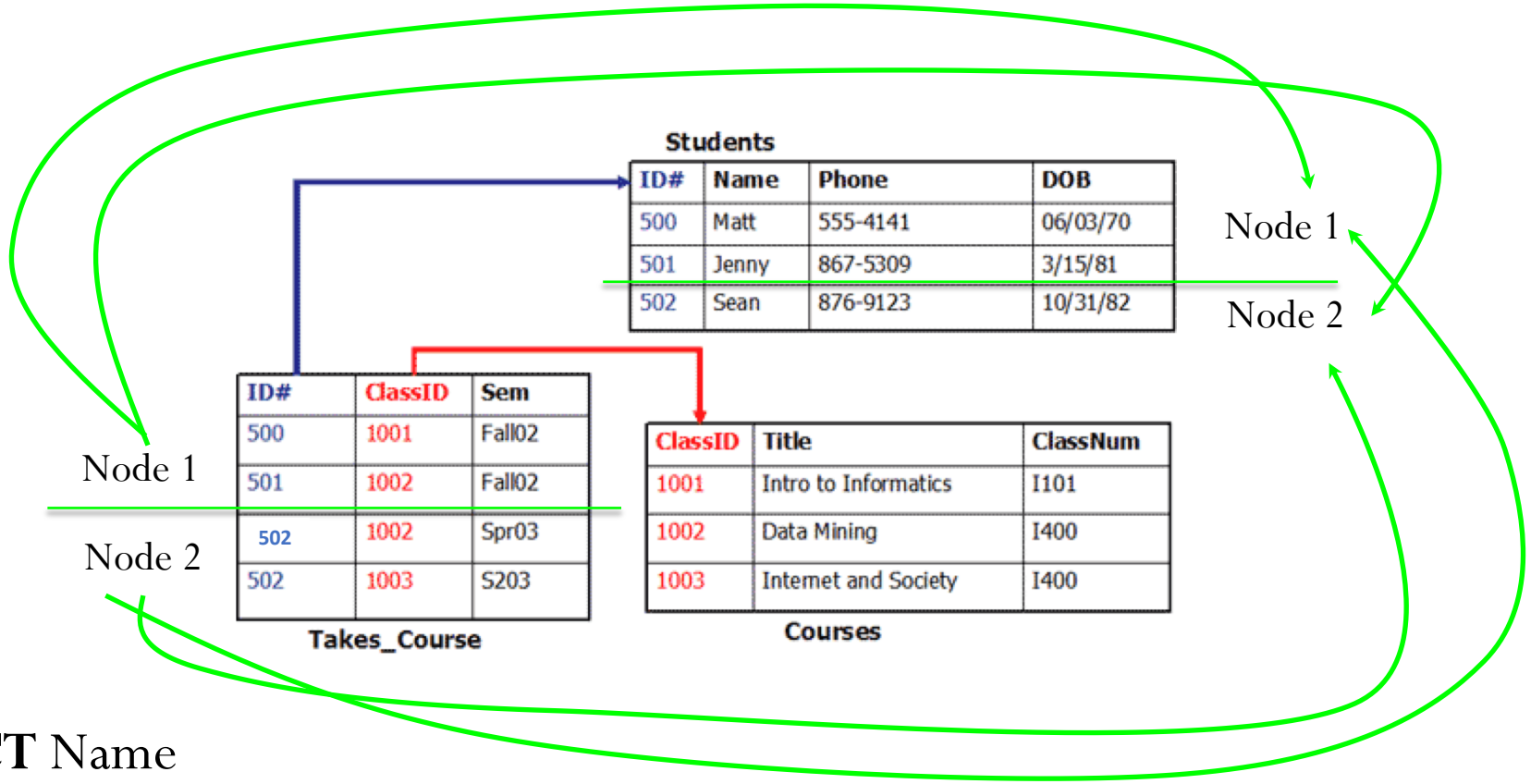
Example


ID: 1001
customer: Ann
line items:
0321293533
2
$48
$96
0321601912
1
$39
$39
0131495054
1
$51
$51
payment details:
Card: Amex
CC Number: 12345
expiry: 04/2001
orders
customers
order lines
credit cards
application level
database level

Main differences:
(4) normalization and joins

- In RDBMs, data are split into tables, according to some normal form (often 3NF)
- Queries can merge data contained in different tables using joins
- But joins are inefficient for large volumes of data
- Solution: take into account joins at design time
- Query-based design

Example

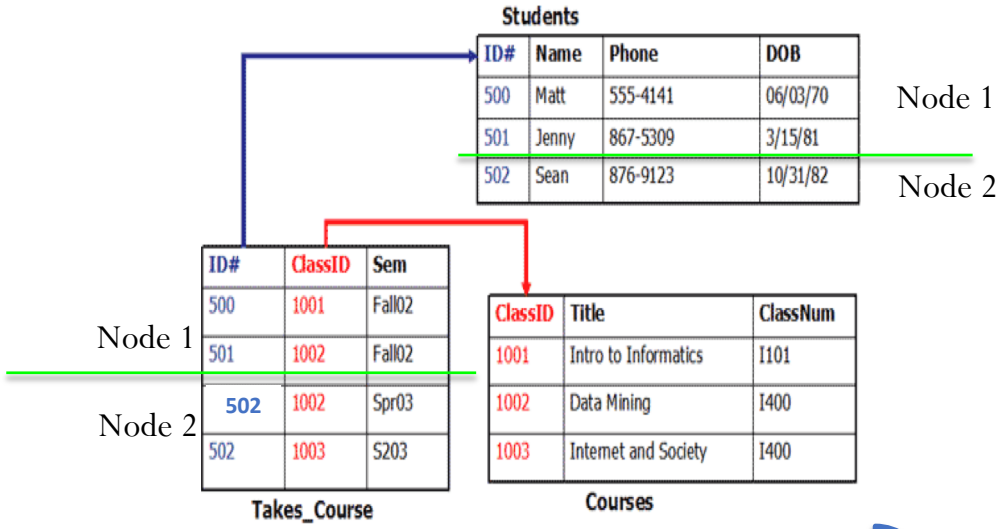


SELECT Name
FROM Students **NATURAL JOIN** Takes_Course
WHERE ClassID = 1001

High data communication across the network

Example

Query-based design



SELECT Name
FROM Students_Take_Courses
WHERE ClassID = 1001

Students_Take_Courses

	ID#	ClassID	Sem	Name	Phone	...
Node 1	500	1001	Fall02	Matt	555-4141	...
	501	1002	Fall02	Jenny	867-5309	...
Node 2	502	1002	Spr03	Jenny	867-5309	
	502	1003	S203	Sean	876-9123	

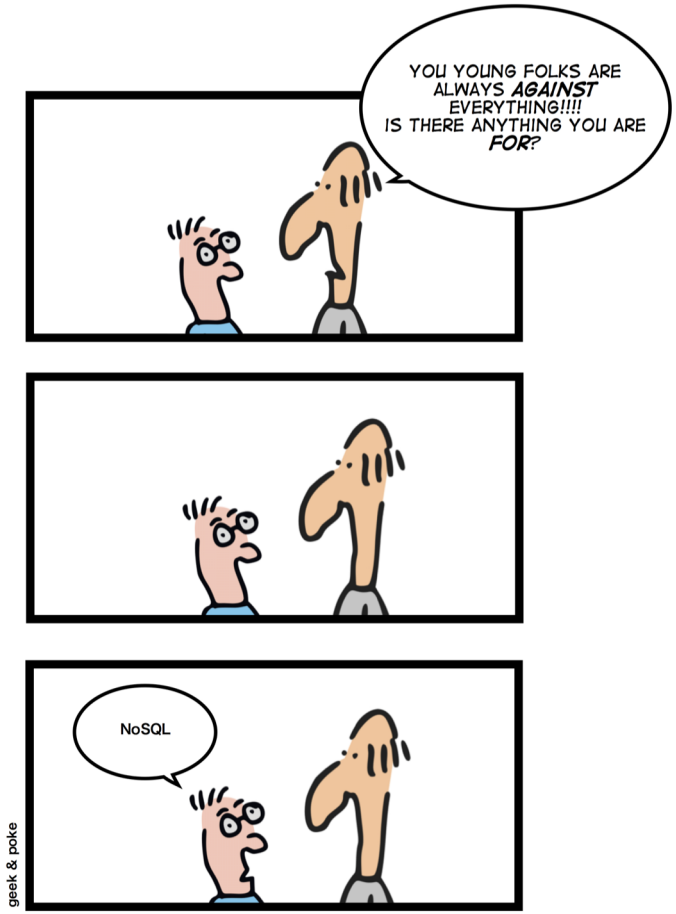
Main differences:
(5) interaction languages

- Moving the complexity at the application layer
  - In application databases, many DBMS functionalities can be moved at the application level
  - There is no more the need for the DBMS to provide a fully featured declarative language
  - Most of the code is procedural and executed at the application side (e.g., using MapReduce)
- Interfaces with the database become simpler
  - read and writes of one or a small number of records
  - avoid operations that require a large data communication (no complex queries or joins)
  - SQL-like languages are still provided for continuity with the past: *similar syntax, very different underlying model*

Introduction to NoSQL systems

NoSQL systems (or data stores)

- New data management systems tailored to large-scale data management requirements
- (N)ot (O)nly SQL not necessarily “Anti SQL”
- Movement more than “one” technology



NoSQL data stores

SYSTEM ARCHITECTURE

- Designed to run on large clusters
- Data distribution (partitioning, replication)
- Strong or eventual consistency
- Often non-ACID transactions
- Application database oriented
- Optimized for specific scenarios (analytical/read intensive, transactional/read-write intensive)

DATA MODEL

- No relational model
- Flexible data model
- Two main modeling approaches: aggregate-oriented and graph-oriented
- Limited impedence mismatch
- Query-based design

STYLE OF INTERACTION

- Simple API and powerful integration with parallel programming frameworks
- Sometimes, SQL-like query languages

Just another temporary trend?

- There have been other trends here before
  - object databases, XML databases, etc.

- But NoSQL data stores:
  - are answer to real practical problems big companies have
  - are often developed by the biggest players
  - outside academia but based on solid theoretical results (e.g., well established results on distributed processing)
  - widely used
  - tipically open-source

Who is NoSQL?


CouchDB
relax


twitter


LinkedIn


mongoDB


facebook


Google


amazon


sones
neotechnology
value in relationships


foursquare


IMDb
Earth's Biggest Movie Database™

(Some) disadvantages of NoSQL

- New and sometimes buggy
- Data is generally replicated, potential for inconsistency
- No standardized data model
- No standardised language for data access
- Data integrity often enforced at the application level

… but

- More and more companies accept the weak points and choose NoSQL databases for their strengths
- NoSQL technologies are also often used as secondary databases for specific data processing applications (in line with the application database concept)

The end of relational databases?

- Relational databases are not going away
  - are ideal for a lot of structured data, reliable, mature, etc.
- RDBMS became one option for data storage
- Polyglot persistence – using different data stores in different circumstances

NoSQL logical data models

- Two main groups of models
  - aggregate-oriented
  - graph-oriented

- For each model: no standard

Take away

- Relational databases have been a successful technology for twenty years, providing persistence, concurrency control, and an integration mechanism
- Application developers have been frustrated with the impedance mismatch between the relational model and the in-memory data structures
- There is a movement away from using databases as integration points towards encapsulating databases within applications and integrating through services
- The vital factor for a change in data storage was the need to support large volumes of data by running on clusters. Relational databases are not designed to run efficiently on clusters
- NoSQL is an accidental neologism. There is no prescriptive definition—all you can make is an observation of common characteristics
  - Not using the relational model, schedules
  - Various interaction protocols
  - Horizontal scaling
  - Replication and sharding
  - Relaxing consistency, no more ACID transactions
  - Open-source