Large-Scale and Multi-Structured Databases

The Database Revolutions

Prof Pietro Ducange

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITÀ DI PISA

CROSSLAB Innovation for industry 4.0

Timeline of Database Frameworks

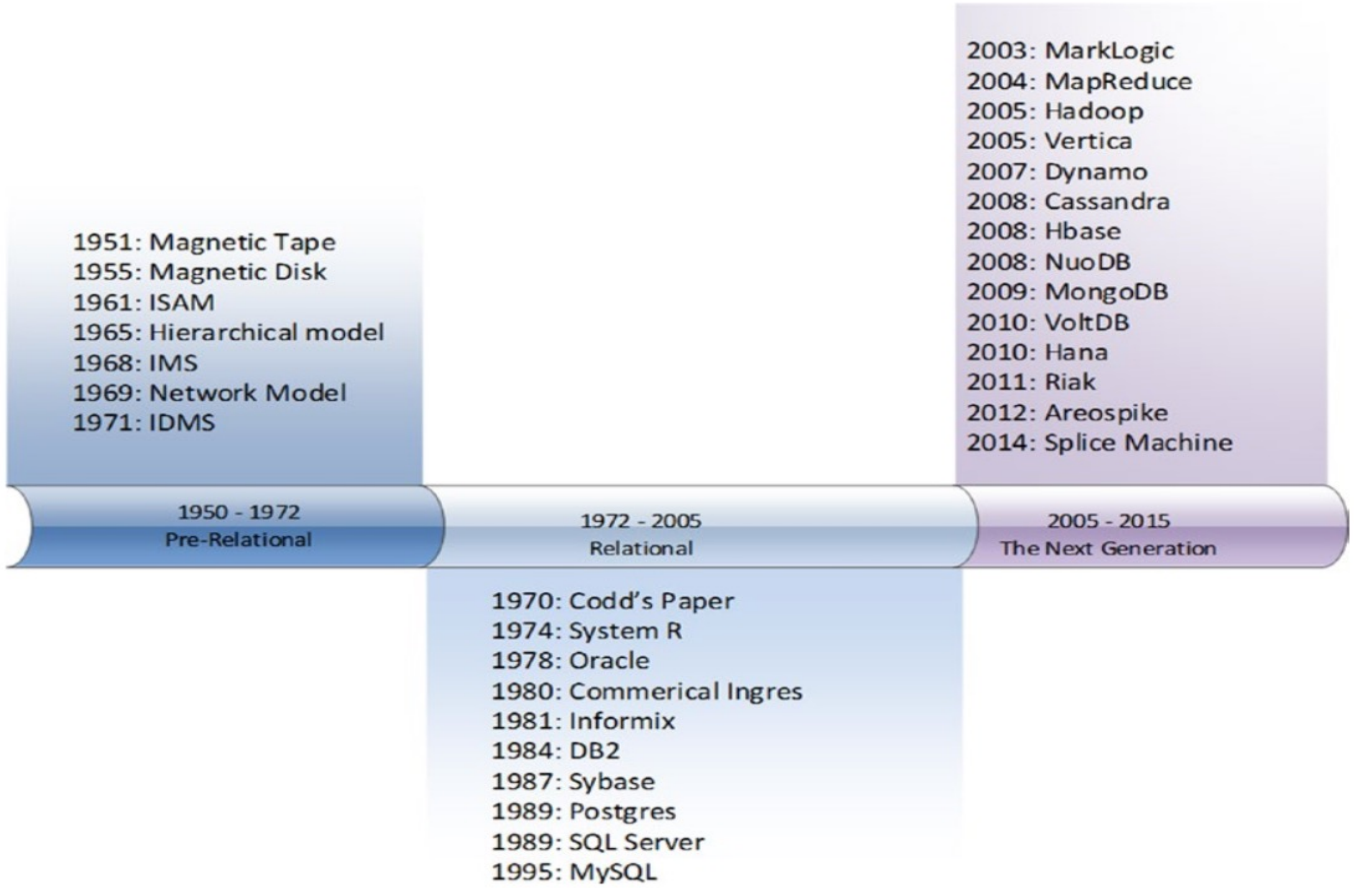


Image extracted from "Guy Harrison, Next Generation Databases, Apress, 2015"

The “Pre historic” Databases

Database is an organized collection of data. Thus:

- ***Books*** with a strongly enforced structure such as ***dictionaries*** and ***encyclopaedias***
- ***Libraries*** and other ***indexed archives*** of information

Can be considered the first databases!

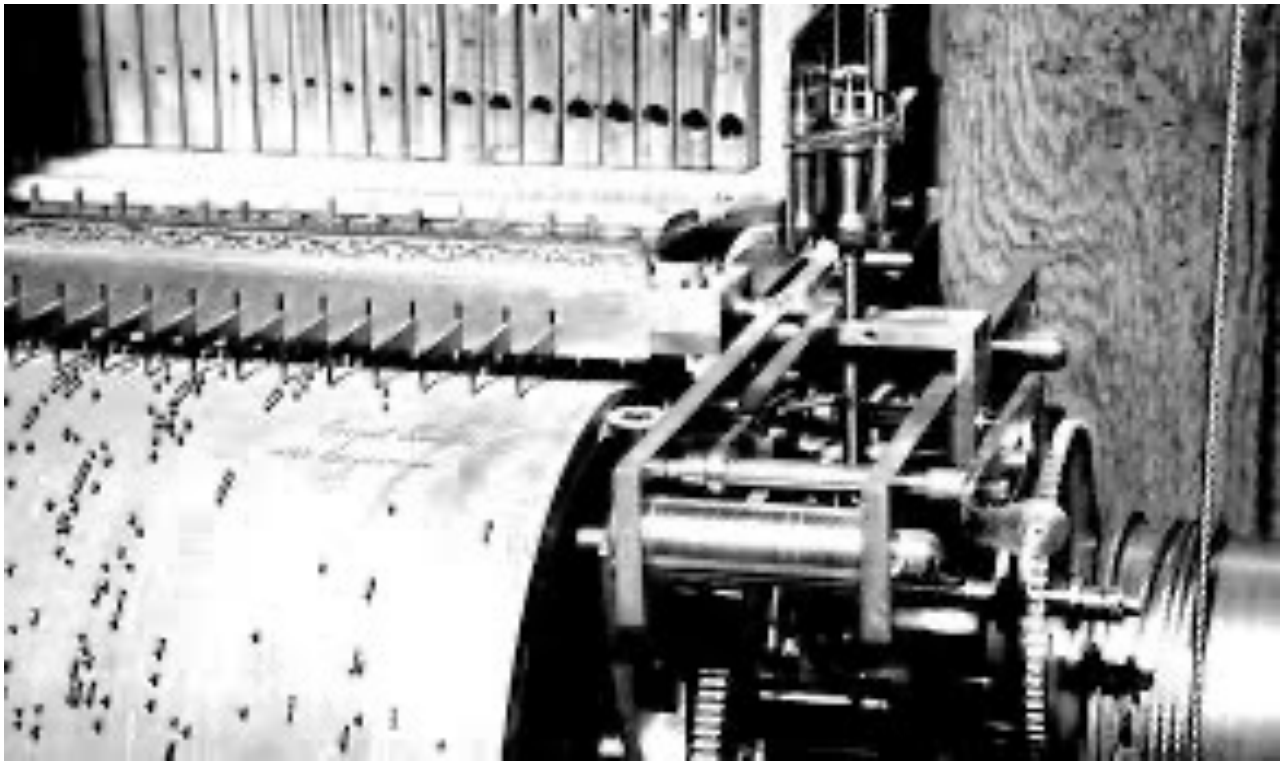
Mechanical Databases (19th Century)

Loom cards for creating complex fabric patterns

***Jacquard Loom**: Image extracted from Wikipedia*

Mechanical Databases (19th Century)

Perforated paper strips for musical boxes



Mechanical Databases (19th Century)

Punched cards used in tabulating machines for producing census statistics

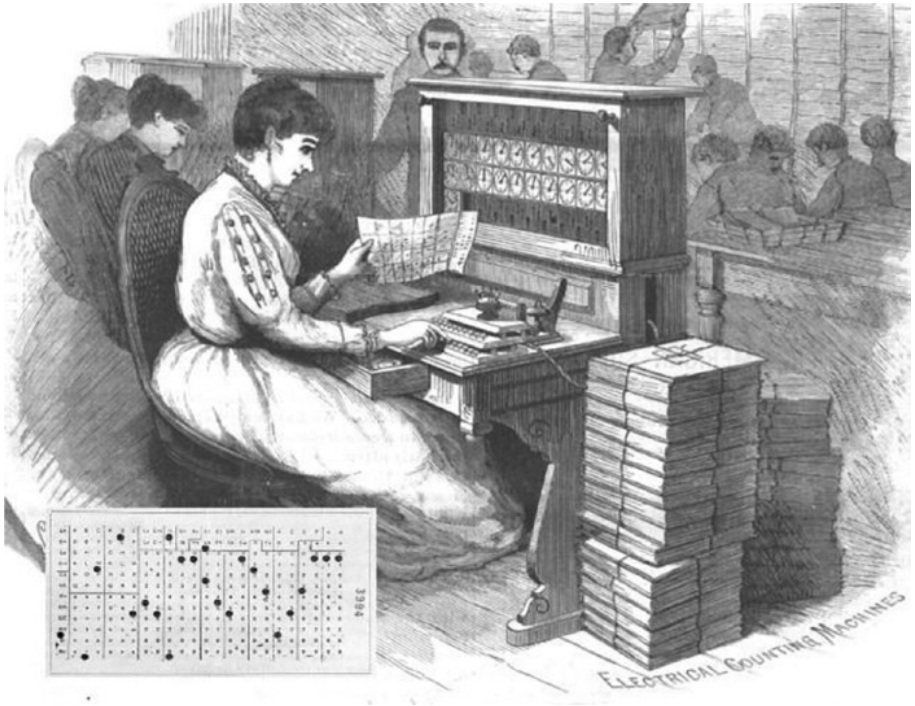


Image extracted from "Guy Harrison, Next Generation Databases, Apress, 2015"

The Tape Revolution (after WWII)

Data were stored ***sequentially*** adopting:

- ***Paper tape***
- ***Magnetic tape***

Features: “fast forward” and “rewind” through the datasets

Problems: no direct high-speed access to individual records!

The First Electronic Databases

Around the Sixties of the 20th Century, the first ***OLTP*** (On-line Transaction Processing) computer systems appeared

Main features: ISAM (Index Sequential Access Method) and other similar indexing strategies

Drawbacks: there were databases but ***no Database Management Systems (DBMS)***. The application had the ***control*** of the database!

Working Without DBMS

Without DBMS, every application had to ***reinvent the database wheel*** !

Problems:

- ***Errors*** in application data handling ***code*** led inevitably to ***corrupted data***.
- Allowing ***multiple users*** to concurrently access or change data requires ***sophisticated coding***.
- ***Optimization*** of data access requires ***complicated*** and ***specialized*** algorithms that could not easily be ***duplicated*** in each application.

The First Database Revolution

DBMS represented a ***new layer*** that allowed the ***separation*** between the database handling logic and the application itself.

As a results ***programmer overhead*** was minimized and the ***performance*** and ***integrity*** of data access routines were ensured.

Main Features:

the definition of both i) a ***scheme*** for the data to store in the database and ii) an ***access path*** for navigating among data records.

The “Navigational” Models

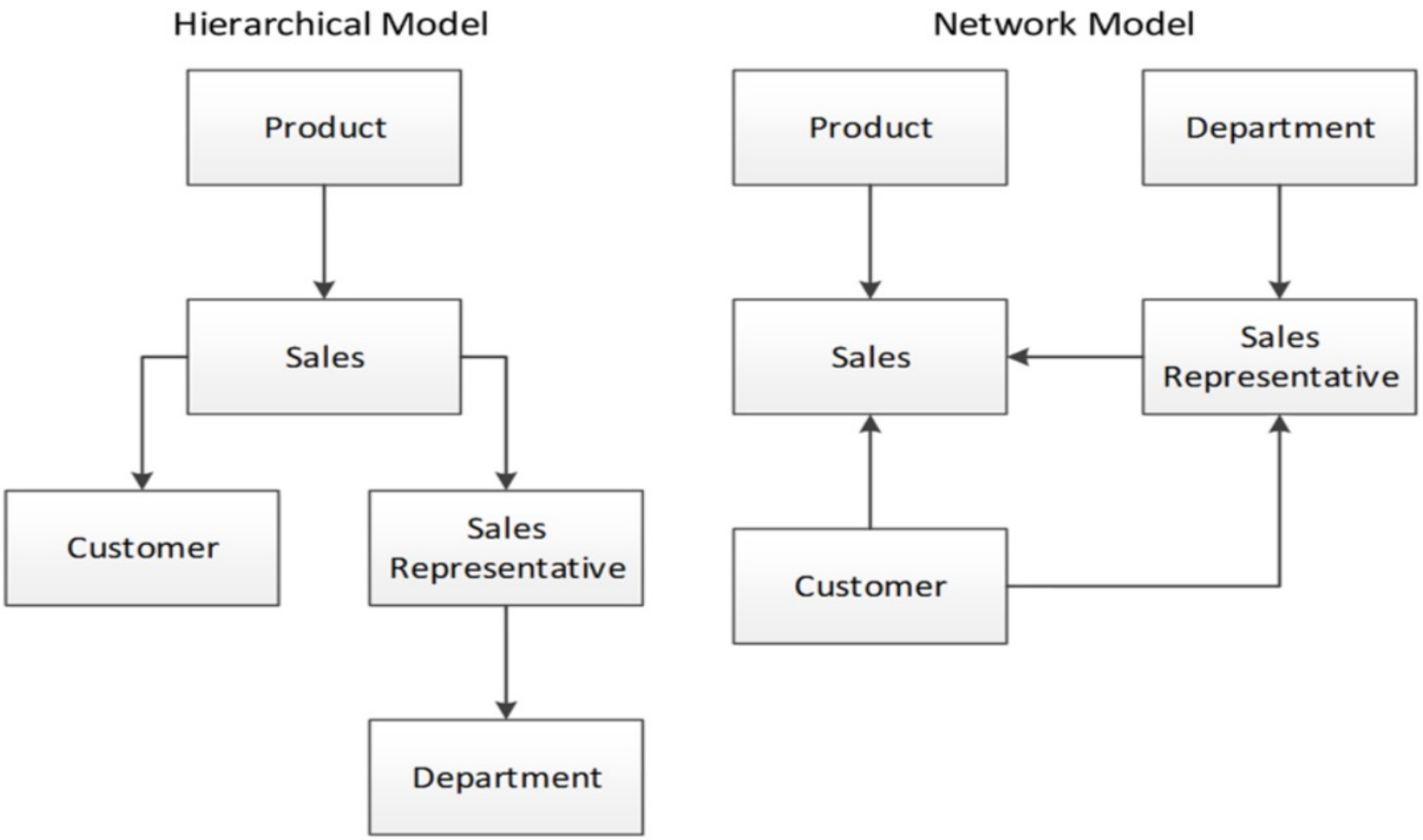


Image extracted from “Guy Harrison, Next Generation Databases, Apress, 2015”

Main Issues of the Navigational Models

- They ***ran*** exclusively on the ***mainframe computer*** systems of the day (largely IBM mainframes)
- Only ***queries*** that could be ***anticipated*** during the initial design phase were possible
- It was ***extremely difficult*** to add new data elements to an existing system
- ***CRUD*** (Create, Read, Update, Delete) operations oriented-> complex analytic queries required ***hard coding***

The Second Database Revolution: Backgrounds

- The business demands for ***analytic-style reports*** were growing rapidly
- Existing databases were ***too hard*** to use
- Existing databases ***lacked*** a ***theoretical*** foundation
- Existing databases ***mixed*** logical and physical implementations

Relational Theory: Tuples and Relations

The theory was discussed for the first time by Edgar Codd in his paper “A Relational Model of Data for Large Shared Data Banks, 1970 ”

Main Features:

Tuples: an unordered set of attribute values. In a database system, a tuple corresponds to a row, and an attribute to a column value.

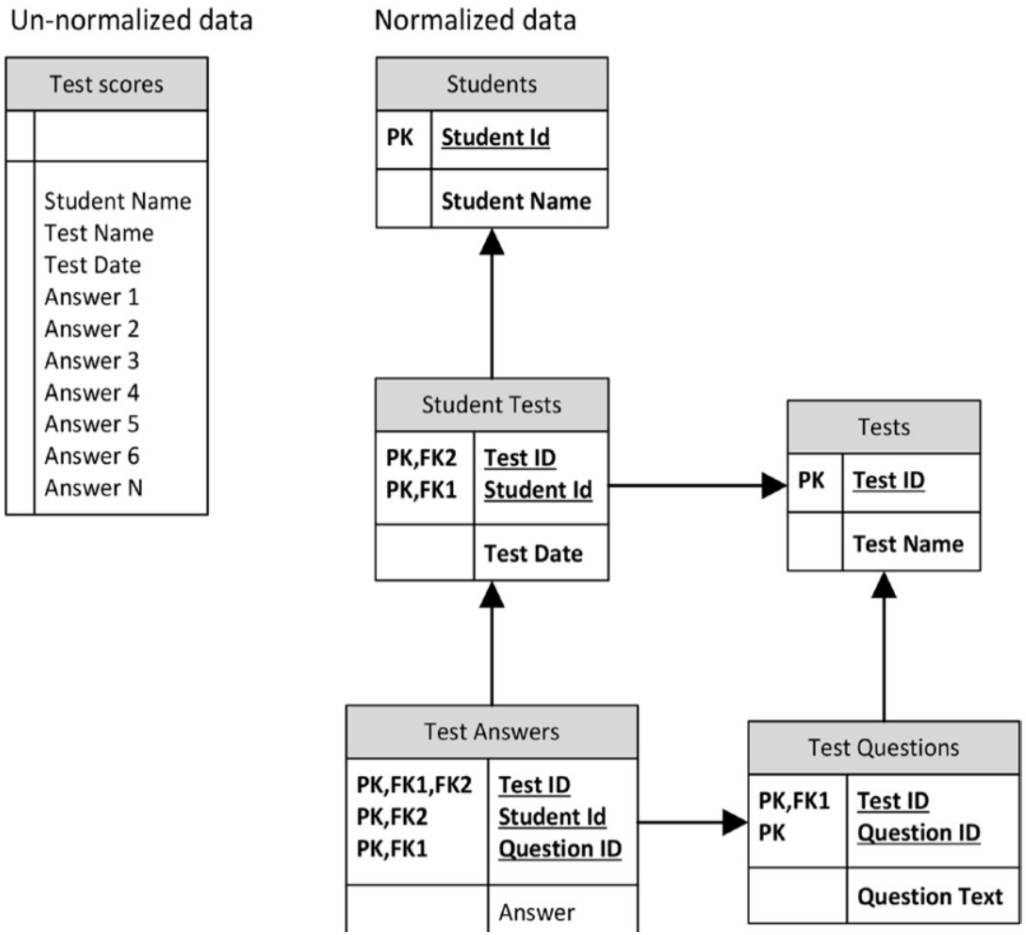
Relations: collections of distinct tuples and correspond to tables in relational database implementations.

Relational Theory: Constrains and Operations

Constraints: enforce consistency of the database. ***Key constraints*** are used to identify tuples and relationships between tuples.

Operations on relations (such as joins, projections, unions). These operations always return relations. In practice, this means that a query on a table returns data in a tabular format.

Relational Theory: Normal Forms



The third normal form: non-key attributes must be dependent on "the key, the whole key, and nothing but the key".

Image extracted from "Guy Harrison, Next Generation Databases, Apress, 2015"

Transactions

Definition: concurrent data change requests on a database system.

Issues: need to ensure consistency and integrity of data.

Solution: ACID Transactions.

Where: mostly in the relational databases that appeared around the Seventies of the 20th Century.

ACID Transactions (I)

Jim Gray definition: "*A transaction is a transformation of state which has the properties of atomicity (all or nothing), durability (effects survive failures) and consistency (a correct transformation).*"

An ACID transaction should be:

Atomic: The transaction is ***indivisible***—either all the statements in the transaction are applied to the database or none are.

Consistent: The database remains in a consistent state ***before*** and ***after*** transaction execution.

ACID Transactions (II)

Isolated: While multiple transactions can be executed by one or more users simultaneously, one transaction should ***not see the effects*** of other in-progress transactions.

Durable: Once a transaction is saved to the database, its changes are expected to ***persist*** even if there is a ***failure*** of operating system or hardware.

ACID Transactions: An Example

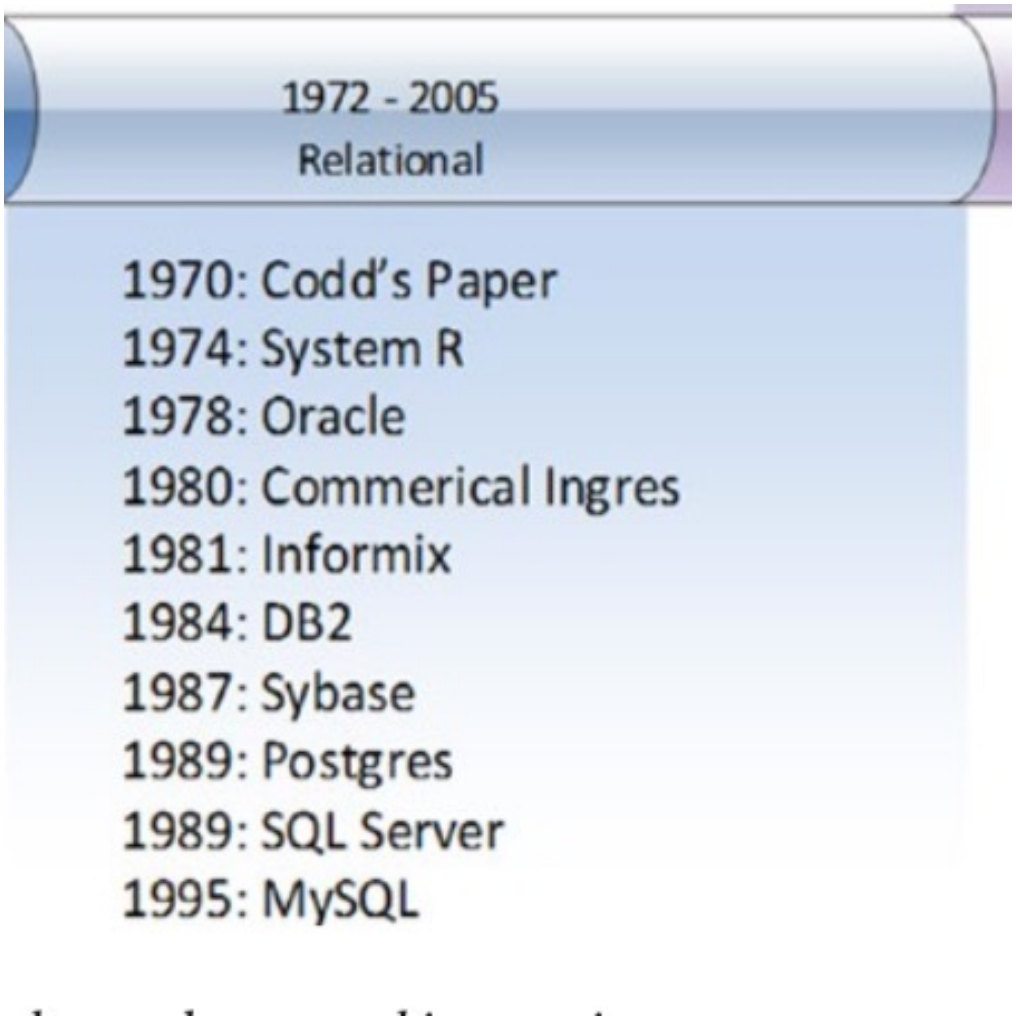
Consider the following transaction T consisting of T1 and T2: Transfer of 100 from account X to account Y.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X − 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction ***fails*** after completion of T1 but before completion of T2.(say, after write(X) but before write(Y)), then amount has been deducted from X but not added to Y. This results in an ***inconsistent*** database state.

Image extracted from: https://www.geeksforgeeks.org/acid-properties-in-dbms/

Relational DBMS



SQL Language: pioneered by IBM in its System R in 1974

Database Wars: in the eighties of the 20th Century, mainly between IBM and ORACLE

By the mid eighties, most of the Relational DBMSs adopted ***SQL*** as a ***standard*** query language.

The database users appreciated the possibility of ***quickly*** writing reports and analytics queries.

Client-Server Computing (late 1980s)

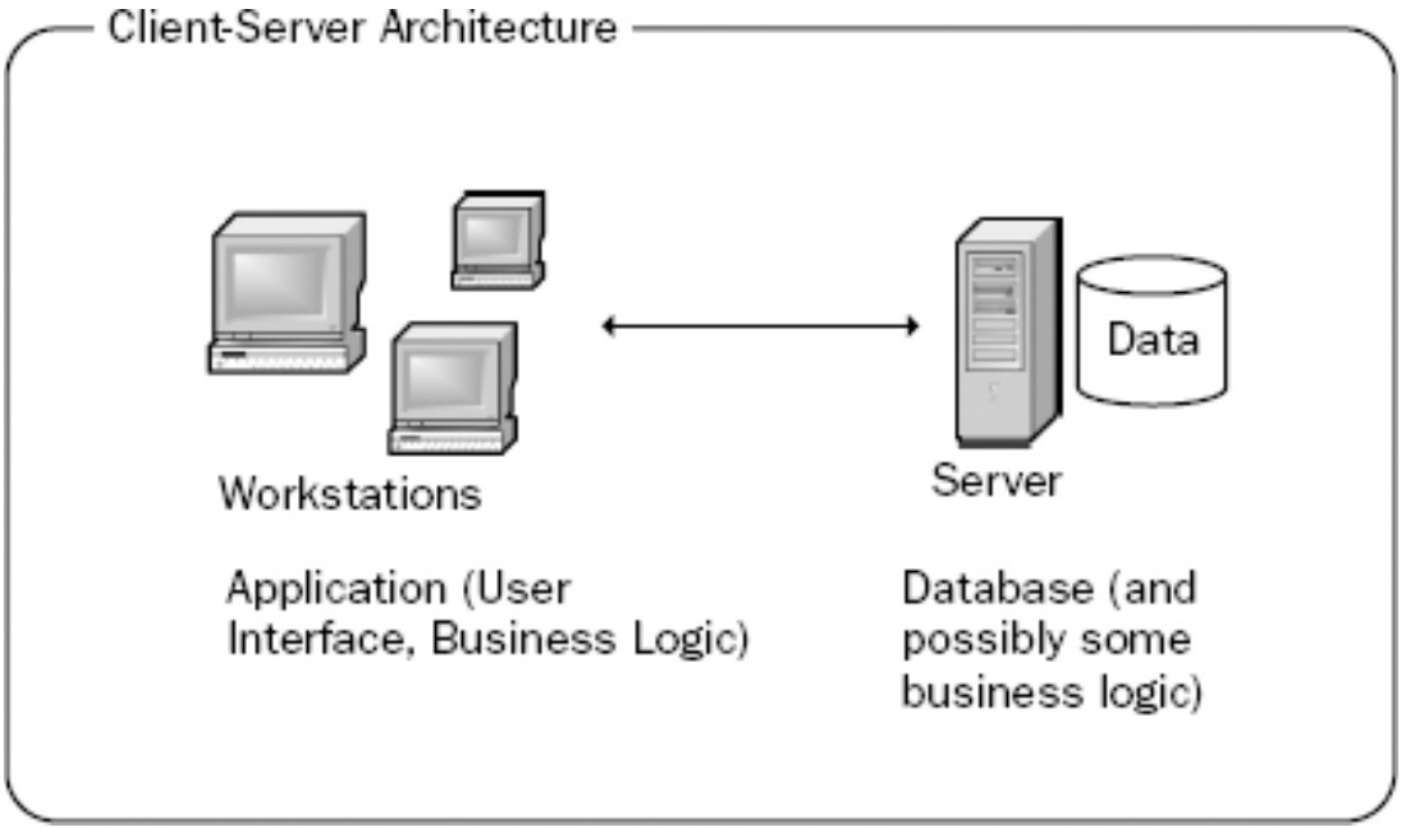


Image extracted from: https://www.oreilly.com/library/view/beginning-accesstm-2007/9780470046845/9780470046845_access_projects_number_symble_using_acce.html

Relational DBMSs in the Client-Server Computing ERA (late 1980s)

- In the client-server model, ***presentation*** logic was hosted on a PC terminal typically running an operating systems with a GUI.
- Database system were hosted on, usually, ***remote servers***.
- ***Application logic*** was often concentrated on the client side, but could also be located within the database server using the ***stored procedures*** *(*programs that ran inside the database).
- Application based on the Client-Server paradigm exploited relational DBMSs as ***backend*** and assumed SQL as the ***vehicle for all requests*** between client and server.

Object-oriented Programming vs RDBMS

- Object-oriented (OO) developers were ***frustrated*** by the mismatch between the object-oriented representations of their data within their programs and the relational representation within the database.

- In an OO program, all the details relevant to a ***logical unit of work*** would be stored within the ***one class*** or directly linked to that class.

- When an object was stored into or retrieved from a *relational database*, ***multiple SQL operations*** would be required to convert from the object-oriented representation to the relational representation

An Example of OO Programmer Frustration

"A relational database is like a garage that forces you to take your car apart and store the pieces in little drawers"

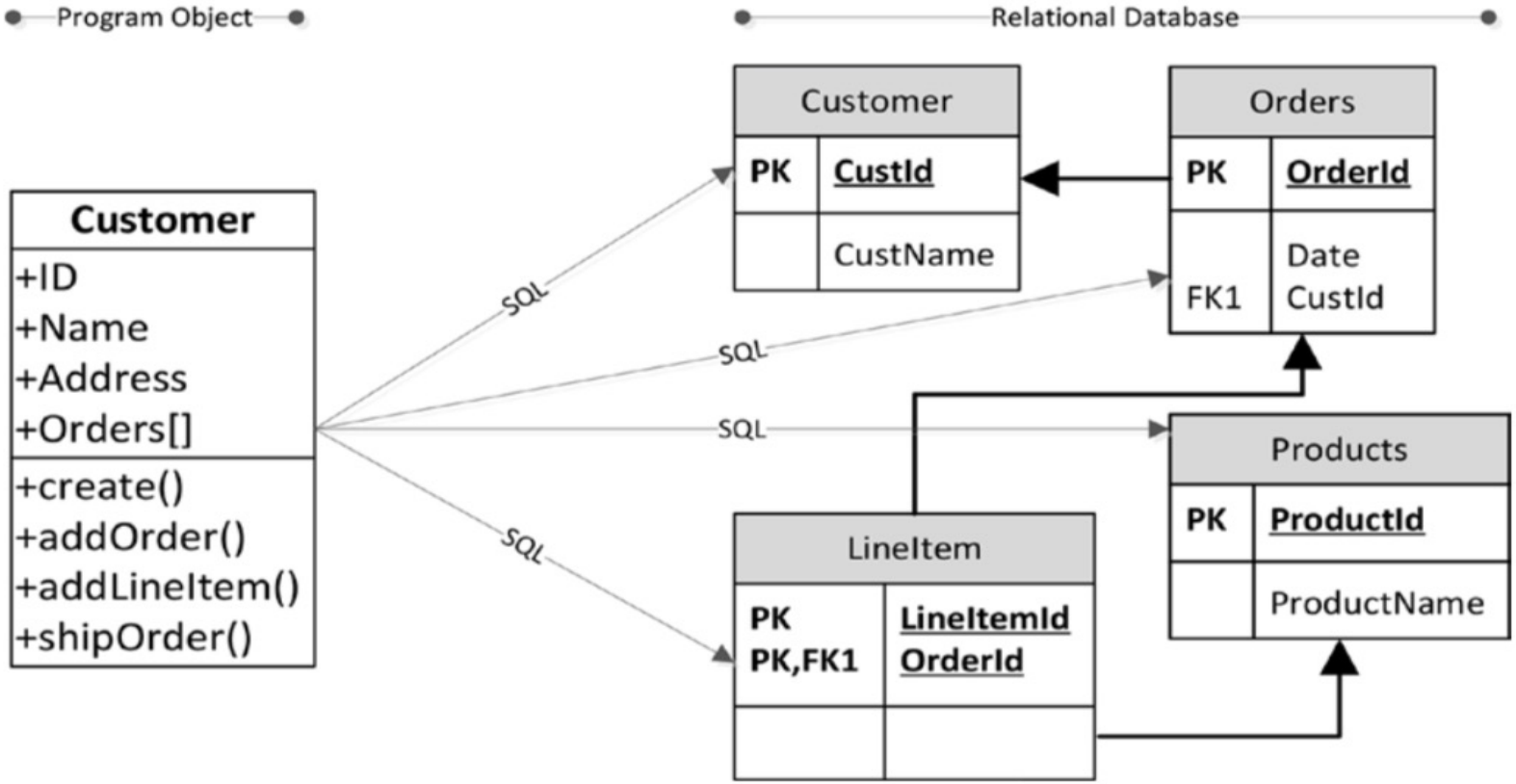


Image extracted from "Guy Harrison, Next Generation Databases, Apress, 2015"

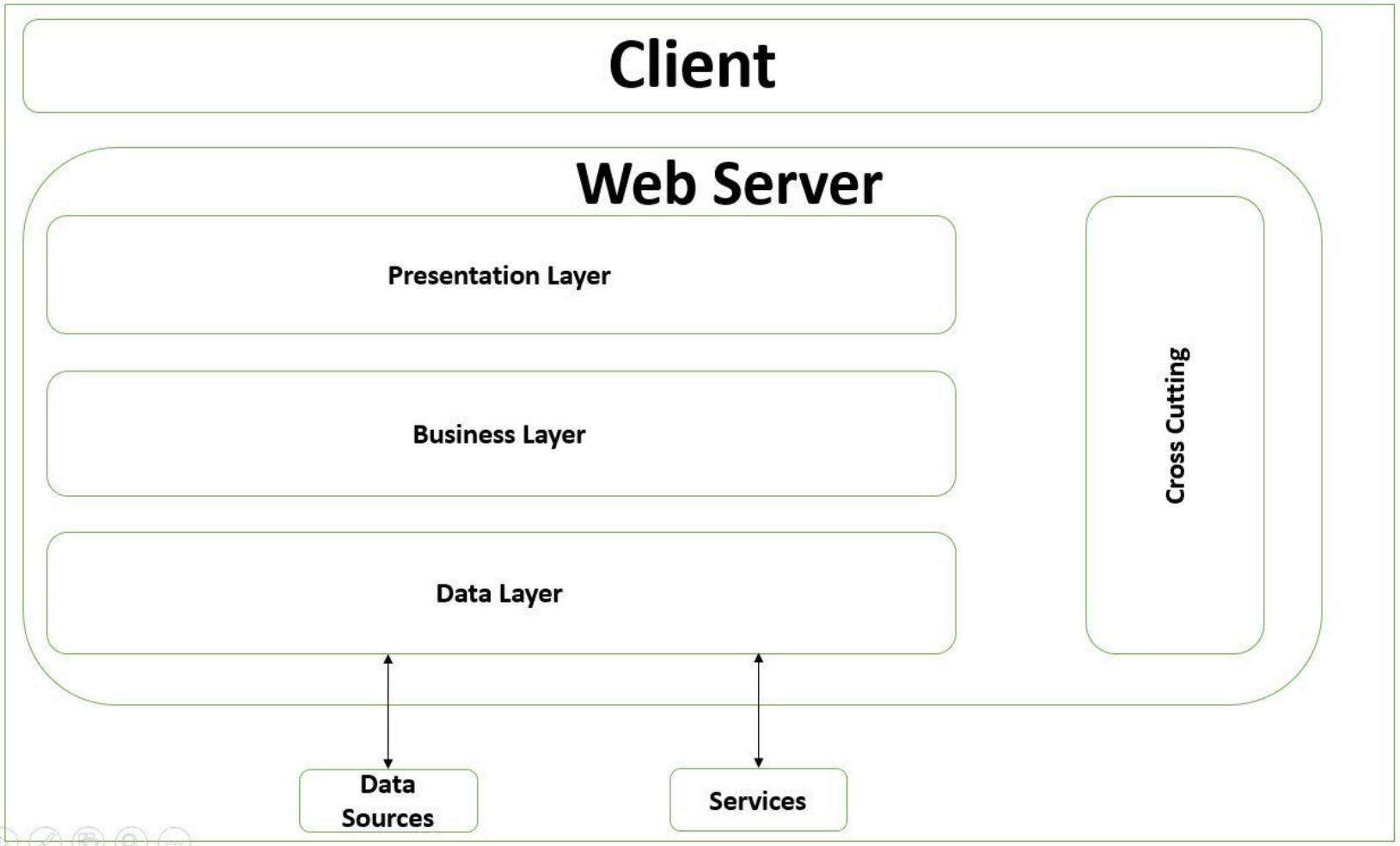
Solutions for Frustrations

- A new DBMS model, namely ***Object Oriented Database Management System*** **(OODBMS)**
- OODBMS would allow to ***store*** program objects ***without normalization***
- Applications would be able to ***load and store*** their object very ***easily***
- The ***implementation*** resembles the "***Navigation***" models
- However, OODBMS systems had ***completely failed*** to gain market share
- OO programmers became ***resigned*** to the use of RDBMS systems to persist objects
- ***Object-Relational Mapping*** (*ORM*) frameworks (Hibernate for example) alleviated the pain of OO programmers

The Second Database Revolution: some considerations

- For a period of roughly ***10 years*** (1995–2005), no significant new databases were introduced
- Although in this period the ***Internet*** has started to ***pervade*** the life of each of us, no new database architectures have emerged
- However, after 2005, the ***era of massive web-scale applications*** created pressures on the relational database model: its supremacy was just about to end!

Web Application Architecture



Massive web-scale applications (MWSAs)

The main actors of these kind of applications are companies such as ***Google, Amazon, Facebook and Yahoo! .*** They need:

- ***Large*** volumes of read and write operations
- ***Low*** latency response times
- ***High*** availability

Limits of RDBMSs for MWSAs

- The performance of RDMSs may be improved by ***upgrading*** CPUs, ***adding*** memory and ***faster*** storage devices (***vertical scaling*** of the servers) .

- Vertical scaling is ***costly***, also in terms of maintenance.

- There are ***limitations*** on the number of CPUs and memory dimension supported by a server.

Limits of RDBMSs for MWSAs

- It's ***hard*** to modify the structure of existing data (i.e. adding an attribute to an object stored in a table).
- RDBMSs may be deployed on ***multiple servers:*** it is very hard to manage such a solutions.
- Moreover, there are ***performance issues*** related to some operations that involve tables stored on different nodes.
- Finally, procedures for ***ACID transactions*** are hard to implement on multiple servers.

The Third Database Revolution: Motivations

MWSAs serving a ***huge amount of users*** (up to millions in social networks, for instance) needs large-scalable DBMSs characterized by:

- Scalability
- Flexibility
- Availability
- Low cost

Scalability

Use case: a spike of traffic to a website.

Solution: adding a new server and then shut it down when the traffic become normal.

Issues: Hard a to adopt the proposed solutions with RBDMs (when possible it results too costly).

Scaling up vs Scaling out

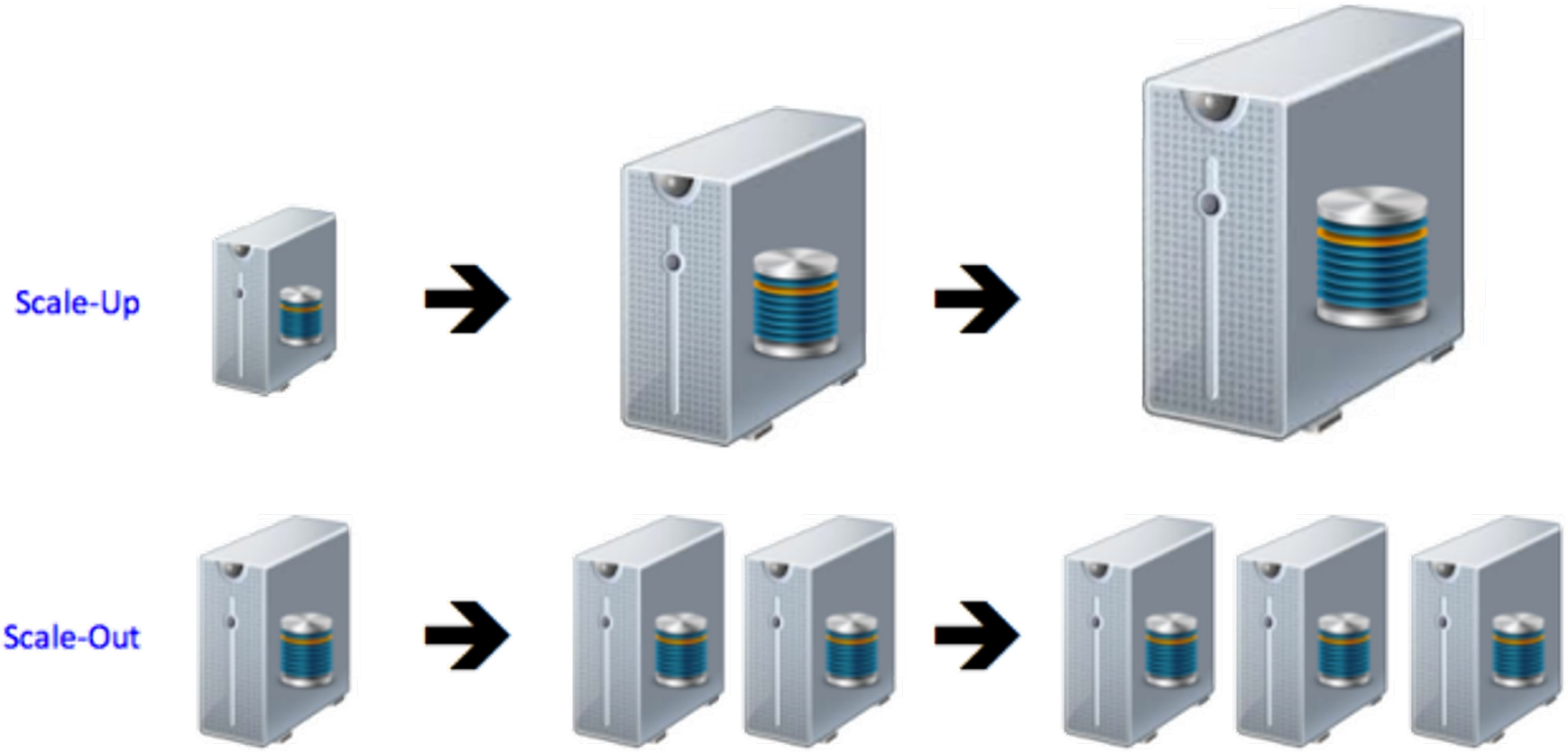


Image extracted from: https://hadoop4usa.wordpress.com/2012/04/13/scale-out-up/

Flexibility

Use case: e-commerce application which handles several kind of items. Items of the same type may also have different descriptions and missing values.

Solution: adding/removing/modifying one or more fields which describe the items in the database

Issues: to modify the schemes of RDBMs is not an easy task and involves the modification also of the application code. Designing relational models often involves defining several tables and needs to know in advance their schemes.

Schema vs Schema-less Models

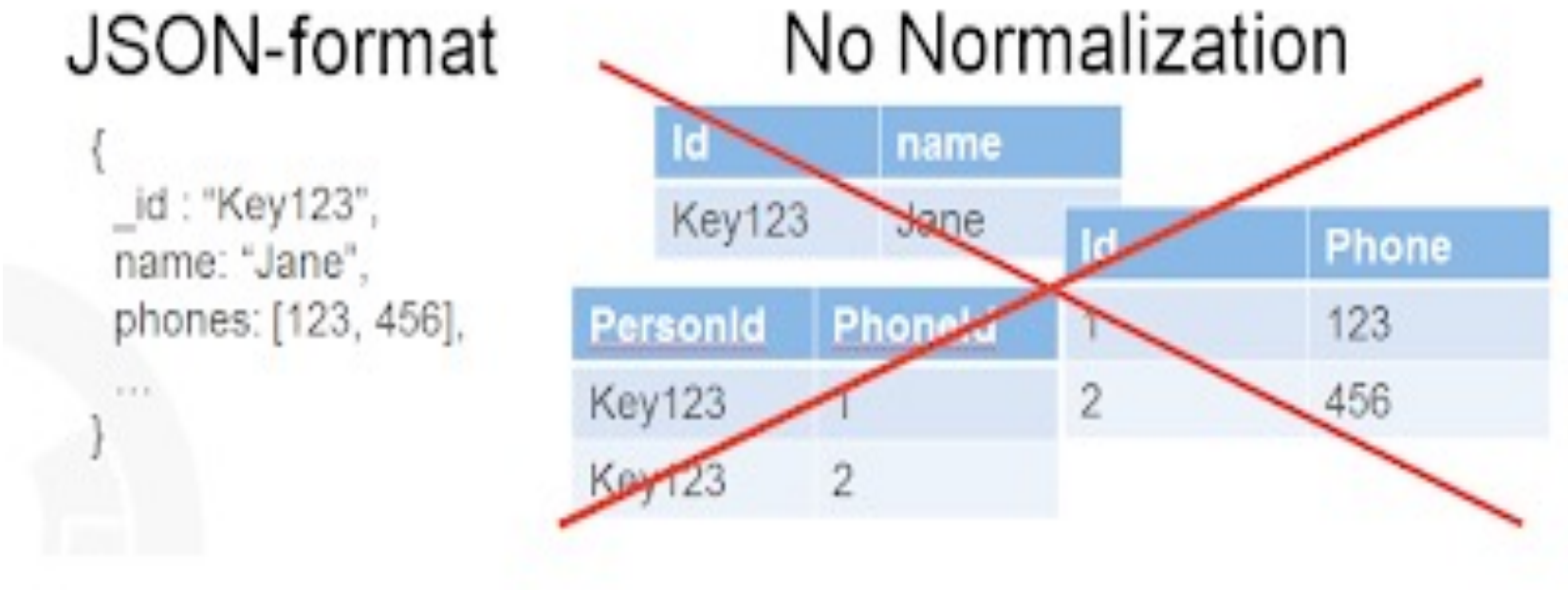


Image extracted from: http://sql-vs-nosql.blogspot.com/2013/10/the-base-difference-between-sql-and.html

Availability

Use case: websites such as Social Networks or e-commerce services must be always available for users. Frequently out-of-services are not permitted for their business.

Solution: to deploy the whole systems on multiple servers. If one server goes down, the services continue to be available for users, even though performances (and also data consistency) may be reduced.

Issues: usually RDMSs run on a single server. If it fails, the services will not be available unless there is a backup server. However, this solutions is not efficient because the second server will not improve the performance of the DBMSs during its normal workflow.

Replication and Sharding

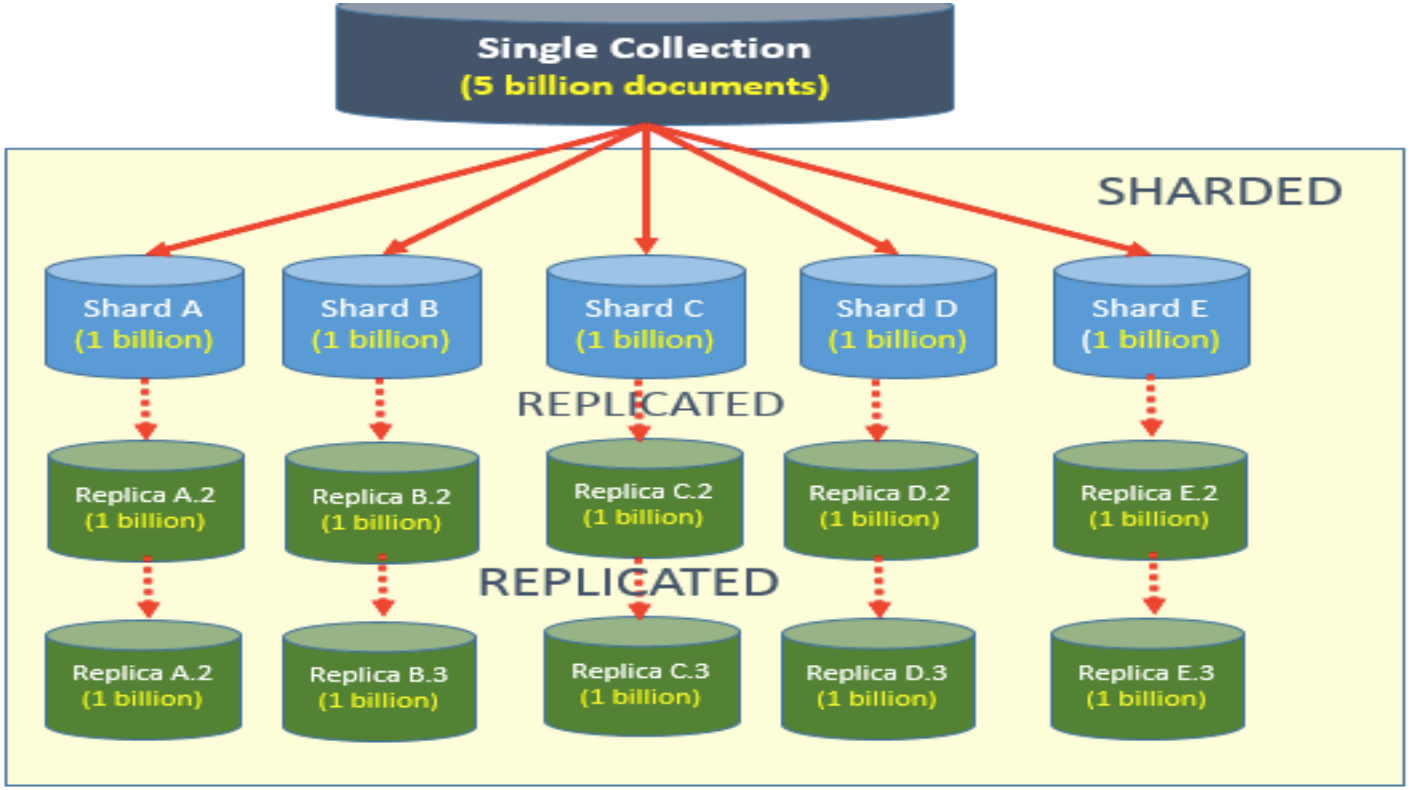


Image extracted from: https://www.red-gate.com/simple-talk/cloud/cloud-data/mongodb-vs-azure-documentdb/

Costs

Use case: Most of the RDBMSs used by enterprises and organizations have often licensing costs. These costs depends on the number of users.

Solution: to use open source DBMSs. Most of the systems born during the third revolutions offers open source solutions. Often, companies (both the same offering the solution or third party) offer consulting services for technical supports.

Issues: enterprises and organizations should change their database architectures and even re-design entirely their applications.

NoSQL Databases

NoSQL is the acronym of *Not Only SQL*

NoSQL considers ***a set of data models*** and related software.

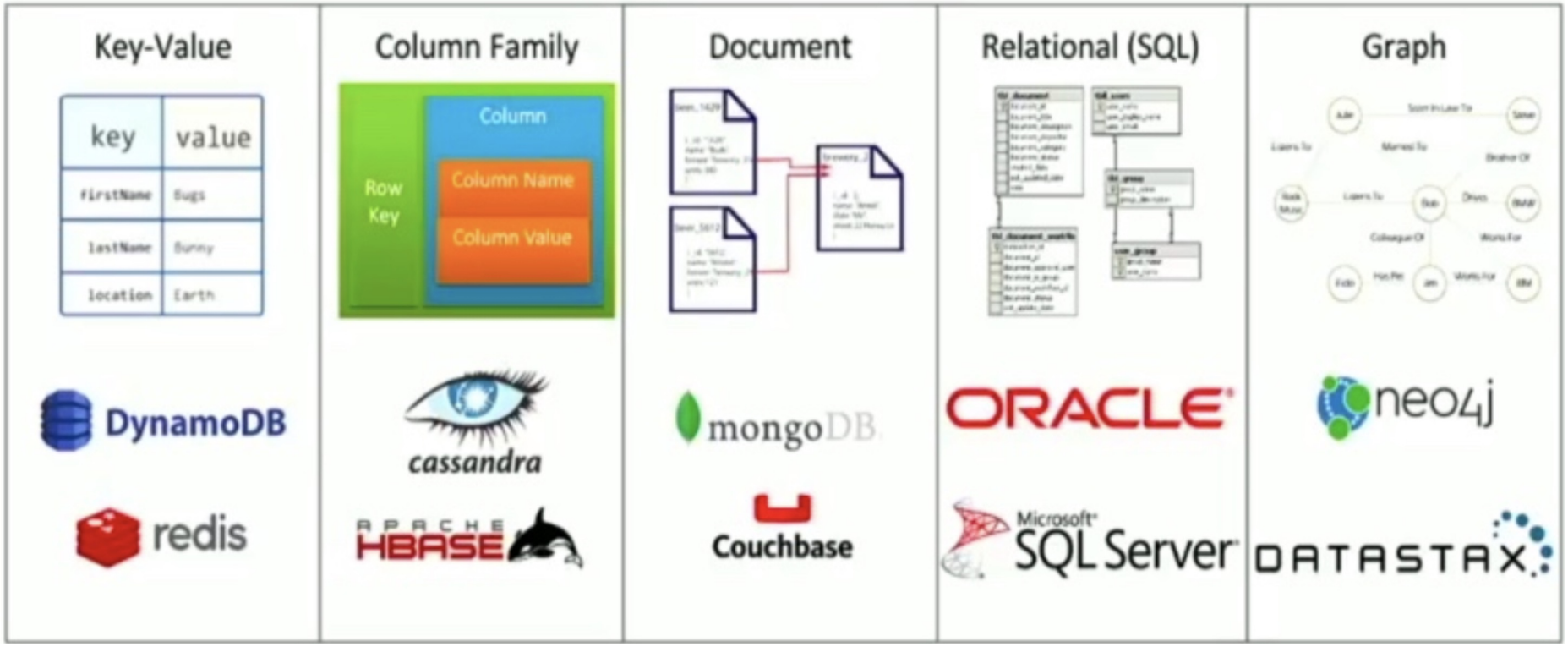
Most of NoSQL solutions ***ensures*** Scalability, Availability, Flexibility and are often open source.

ACID transactions may be ***not supported*** by NoSQL databases.

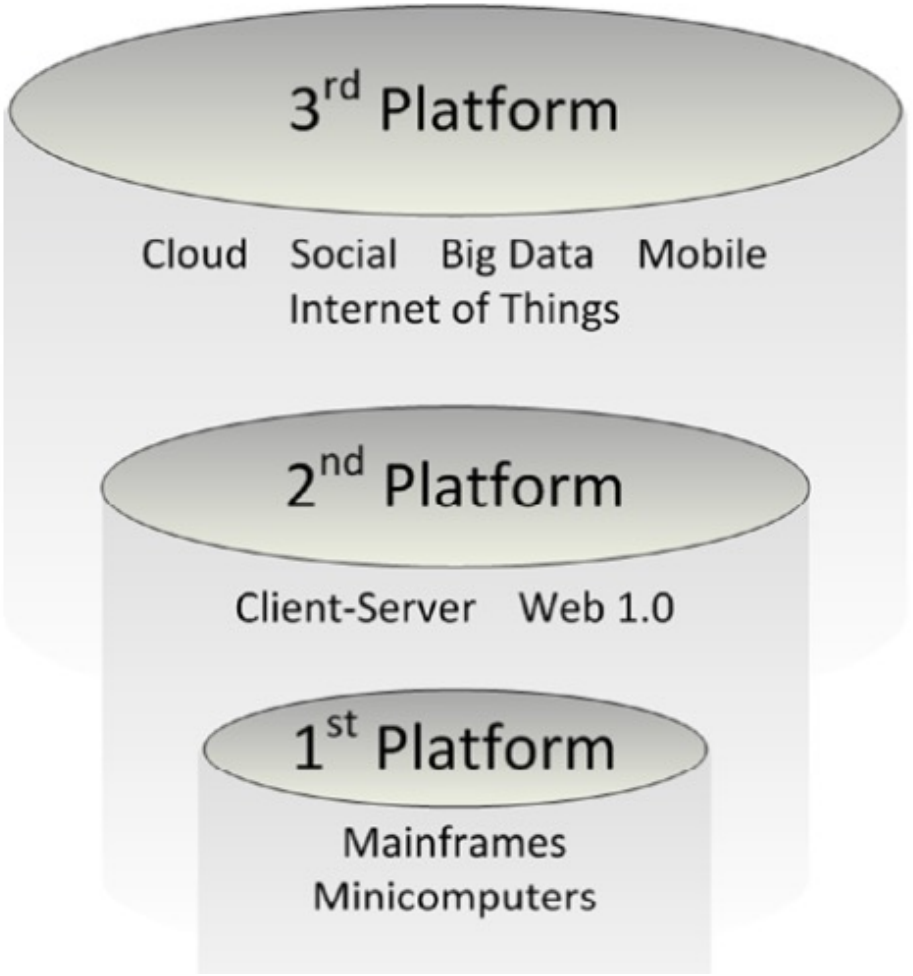
Roughly speaking, , NoSQL databases ***reject*** the constraints of the ***relational model***, including strict consistency and schemas.

There exists also ***NewSQL Databases*** which retain many features of the relational model but amend the underlying technology in significant ways.

NoSQL Databases Galaxy



The database and computer architecture waves



Relational Database
NoSQL
NewSQL
Big Data platforms

Relational Database

Hierarchical Database
Network Database
ISAM files

For Nostalgics of SQL

Currently, ***scalable services*** for handling relational databases are offered by big enterprises such as ***Google*** and ***Amazon***.

These services are ***cloud-based*** and ***pay-per-use services***.

Using these services, enterprises ***may continue*** to use their software without the needs to migrate towards NoSQL databases.

The offered solutions are often ***scalable*** up to thousands of nodes and ensure ***high availability*** of the service.

Issues of SQL on Cloud Services

Even though the ***availability*** and ***scalability*** requirements are ensured, the ***flexibility is reduced*** as in classical SQL-based DBMSs.

Data and services are managed on the ***cloud***, thus some ***privacy*** and ***security*** issues may arise.

Solutions with DBMSs installed and managed ***on site*** cannot be deployed.

Even though the costs seem to be reduced, a ***spending manager*** is needed for taking the costs ***under control***.

SQL Cloud-Based Solutions

Amazon RDS
https://aws.amazon.com/it/rds/

Google Cloud Spanner
https://cloud.google.com/spanner/

Microsoft Azure SQL
https://azure.microsoft.com/services/sql-database/

Suggested Readings

Chapter 1 of the book "*Guy Harrison, Next Generation Databases, Apress, 2015*"

Chapter 1 of the book "*Dan Sullivan, NoSQL For Mere Mortals, Addison-Wesley, 2015*"

Check also: http://nosql-database.org/