Data engineering

Contents

- Data management
- Relational database modelling
- Normal forms and ER diagrams
- NoSQL databases
- Distributed database concepts

Data management

Database Management System (DBMS)

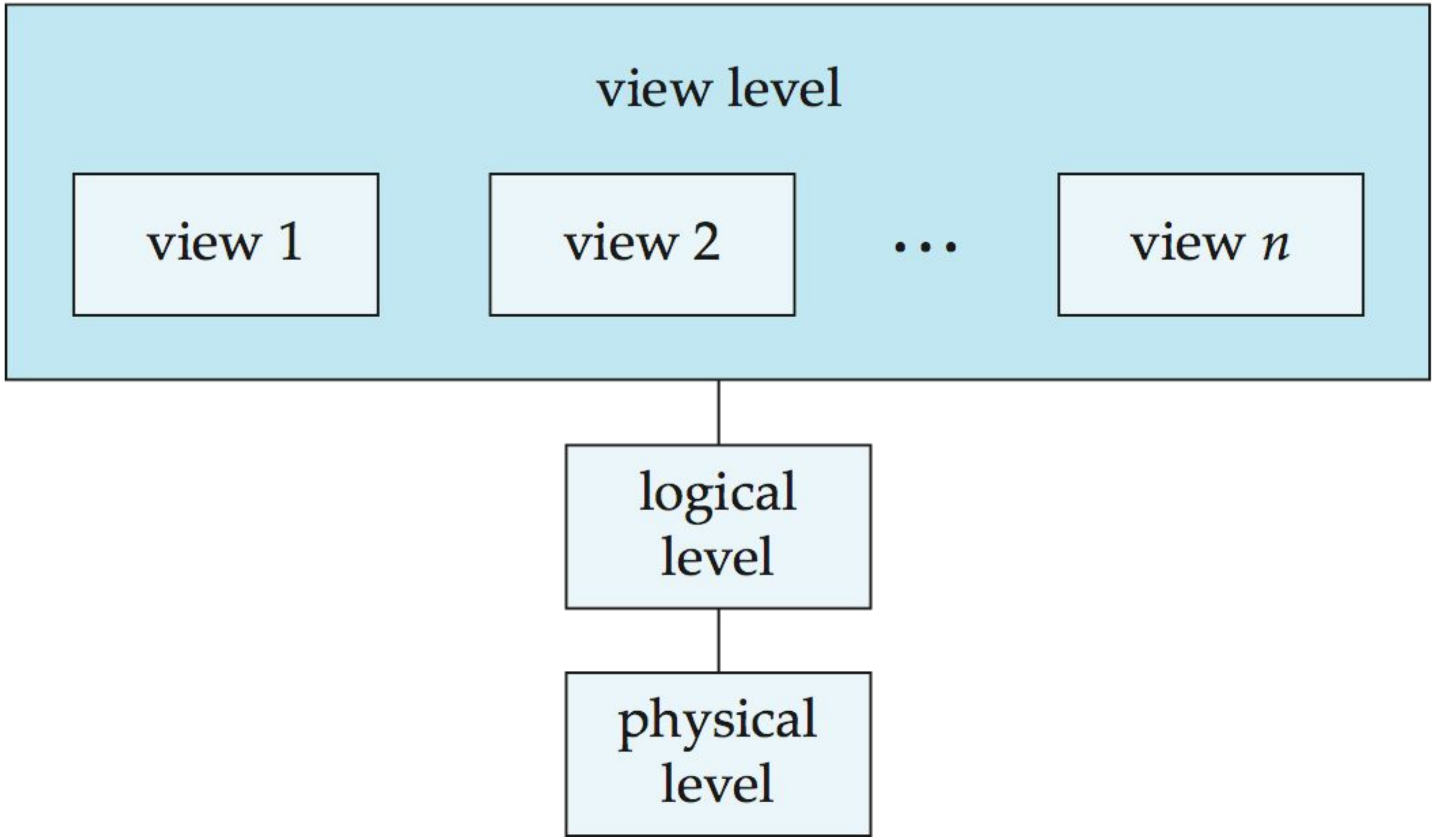
- DBMS contains information about a particular enterprise
    - Collection of interrelated data
    - Set of programs to access the data
    - An environment that is both *convenient* and *efficient* to use
- Database Applications:
    - Banking: transactions
    - Airlines: reservations, schedules
    - Universities: registration, grades
    - Sales: customers, products, purchases
    - Online retailers: order tracking, customized recommendations
    - Manufacturing: production, inventory, orders, supply chain
    - Human resources: employee records, salaries, tax deductions
- Databases can be very large.
- Databases touch all aspects of our lives

University Database Example

- Application program examples
  - Add new students, instructors, and courses
  - Register students for courses, and generate class rosters
  - Assign grades to students, compute grade point averages (GPA) and generate transcripts
- In the early days, database applications were built directly on top of file systems

View of Data

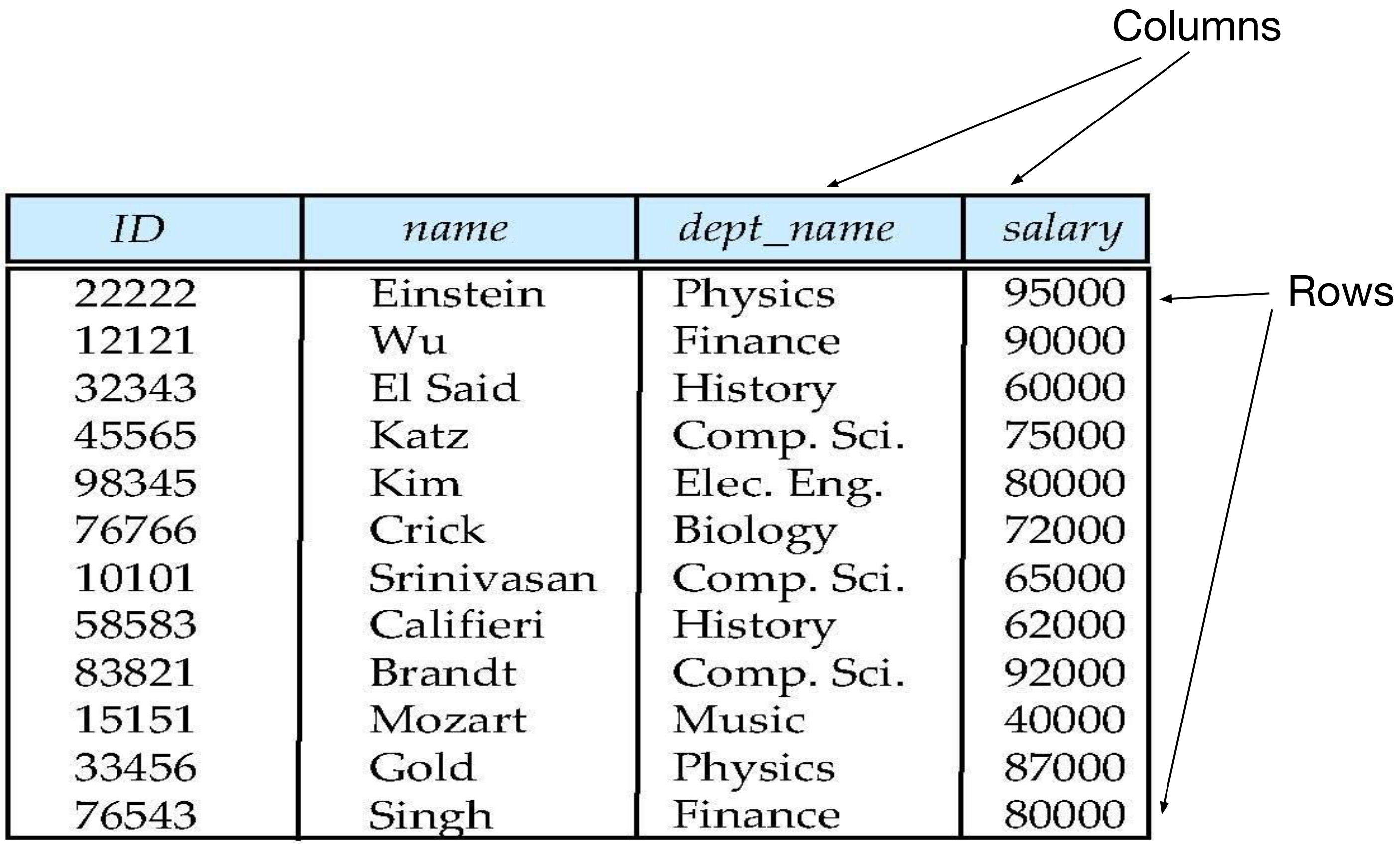
An architecture for a database system



Data Models

- A collection of tools for describing
  - Data
  - Data relationships
  - Data semantics
  - Data constraints
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semistructured data model (XML)
- Other older models:
  - Network model
  - Hierarchical model

Relational Model



Columns

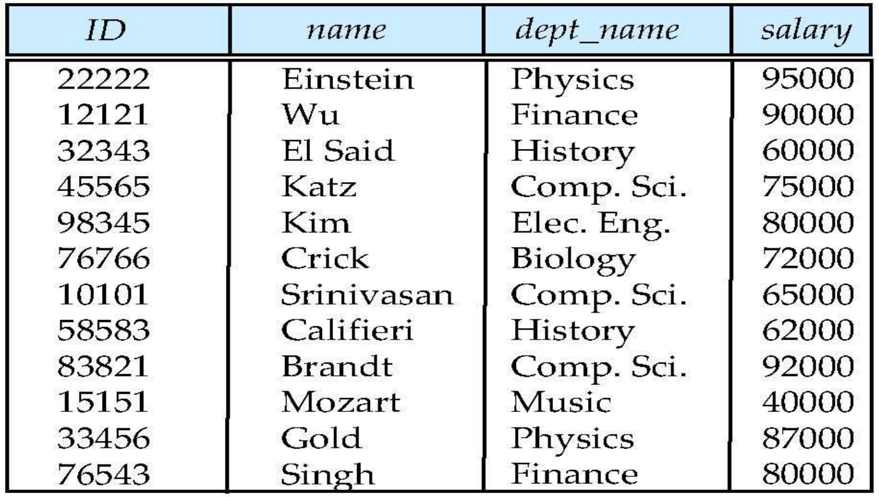
Rows

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table

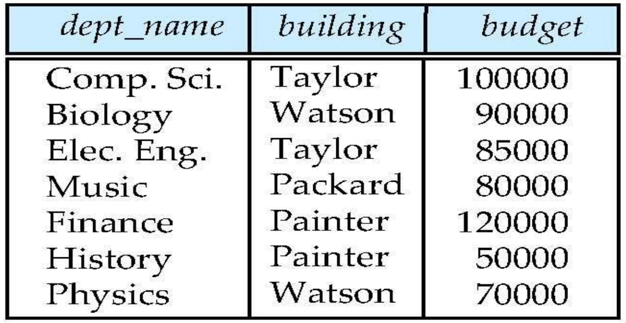
- All the data is stored in various tables.
- Example of tabular data in the relational model

A Sample Relational Database



ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(a) The *instructor* table



dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Data Definition Language (DDL)

- Specification notation for defining the database schema

  Example:

  ```
  create table instructor (
                 ID          char(5),
                 name        varchar(20),
                 dept_name   varchar(20),
                 salary      numeric(8,2))
  ```

- DDL compiler generates a set of table templates stored in a ***data dictionary***
- Data dictionary contains metadata (i.e., data about data)
  - Database schema
  - Integrity constraints
    - Primary key (ID uniquely identifies instructors)
  - Authorization
    - Who can access what

Data Manipulation Language (DML)

- Language for accessing and manipulating the data organized by the appropriate data model
  - DML also known as query language
- Two classes of languages
  - **Pure** – used for proving properties about computational power and for optimization
    - Relational Algebra
    - Tuple relational calculus
    - Domain relational calculus
  - **Commercial** – used in commercial systems
    - SQL is the most widely used commercial language

SQL

- The most widely used commercial language
- SQL is NOT a Turing machine equivalent language
- SQL is NOT a Turing machine equivalent language
- To be able to compute complex functions SQL is usually embedded in some higher-level language
- Application programs generally access databases through one of
  - Language extensions to allow embedded SQL
  - Application program interface (e.g., ODBC/JDBC) which allow SQL queries to be sent to a database

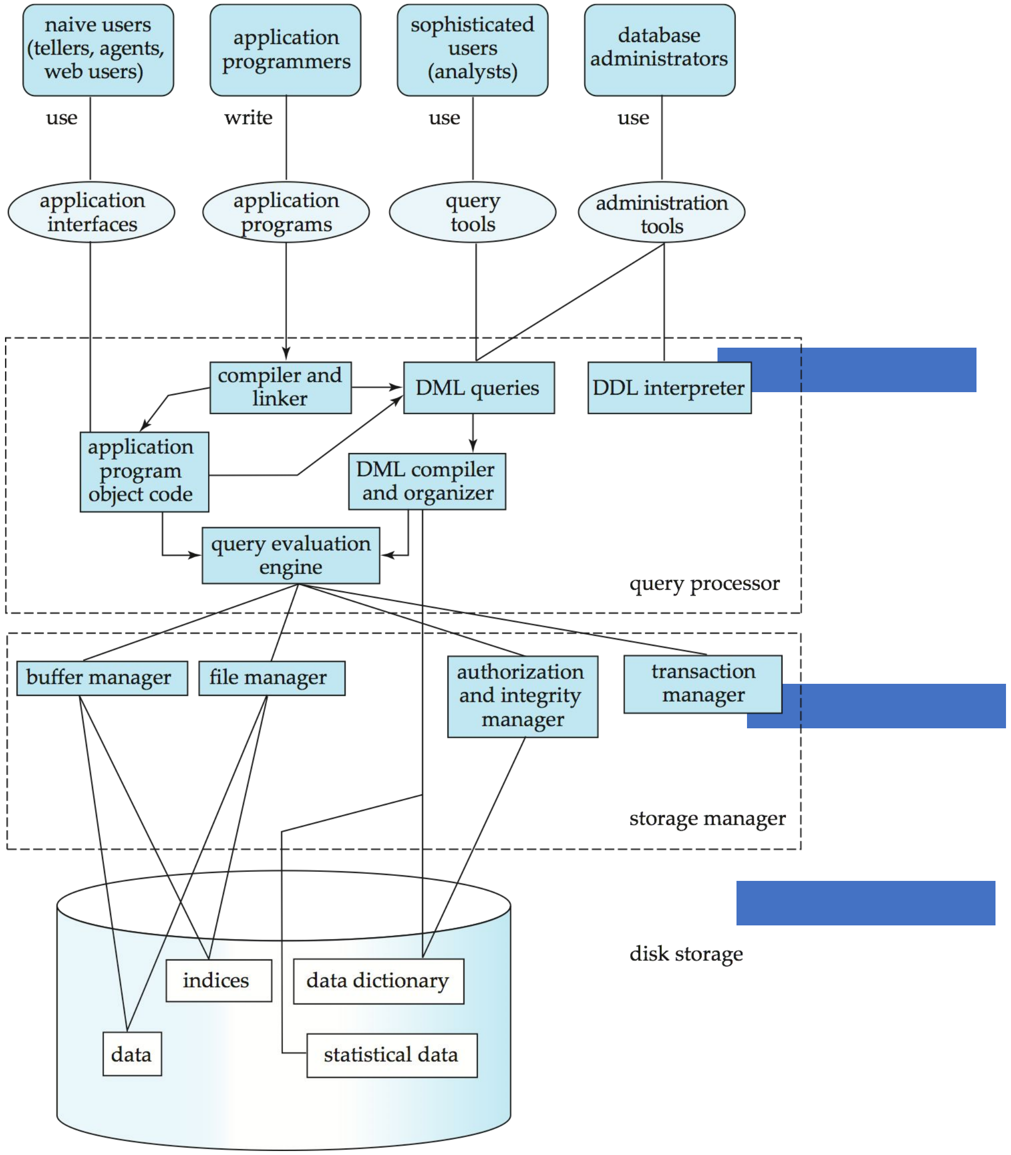
Database Engine

- Storage manager
- Query processing
- Transaction manager

Storage Management

- **Storage manager** is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible to the following tasks:
  - Interaction with the OS file manager
  - Efficient storing, retrieving and updating of data
- Issues:
  - Storage access
  - File organization
  - Indexing and hashing

Database System Internals



Keys

- Let K ⊆ R
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey *K* is a **candidate key** if *K* is minimal

  Example: {*ID*} is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**.
  - Which one?
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation
  - **Referenced** relation
  - Example: *dept_name* in i*nstructor* is a foreign key from *instructor* referencing *department*

ER model -- Database Modeling

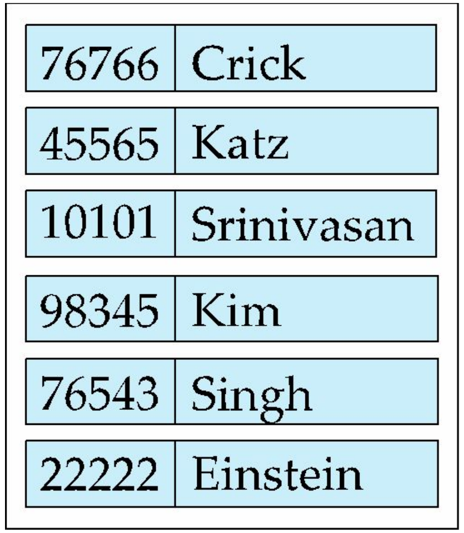
- The ER data mode was developed to facilitate database design by allowing specification of an **enterprise schema** that represents the overall logical structure of a database.
- The ER data model employs three basic concepts:
  - entity sets,
  - relationship sets,
  - attributes.
- The ER model also has an associated diagrammatic representation, the **ER diagram**, which can express the overall logical structure of a database graphically.

Entity Sets

- An **entity** is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- An **entity set** is a set of entities of the same type that share the same properties.
  - Example: set of all persons, companies, trees, holidays
- An entity is represented by a set of attributes; i.e., descriptive properties possessed by all members of an entity set.
  - Example:

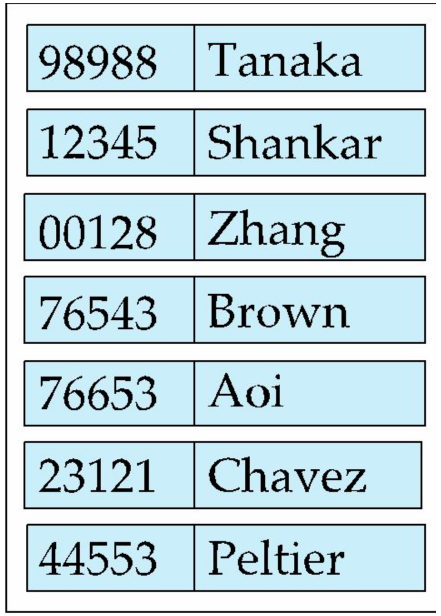
    instructor = (*ID, name, salary*)
    course= (*course_id, title, credits*)
- A subset of the attributes form a **primary key** of the entity set; i.e., uniquely identifying each member of the set.

Entity Sets -- *instructor* and *student*



76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor



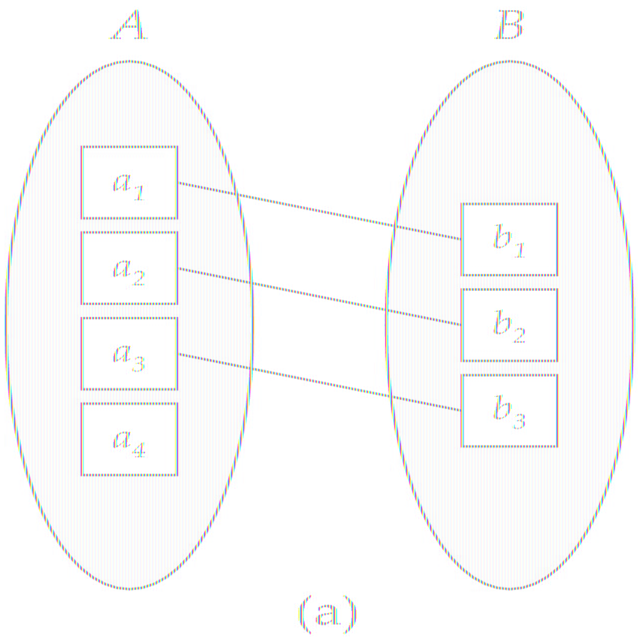
98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

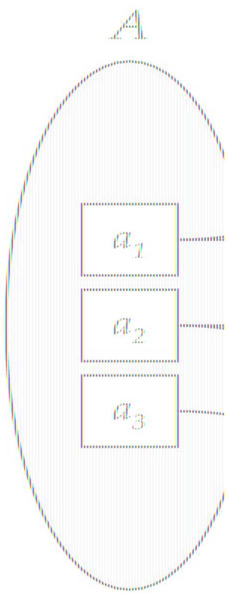
Mapping Cardinality Constraints

- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- For a binary relationship set the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to many

Mapping Cardinalities



One to one

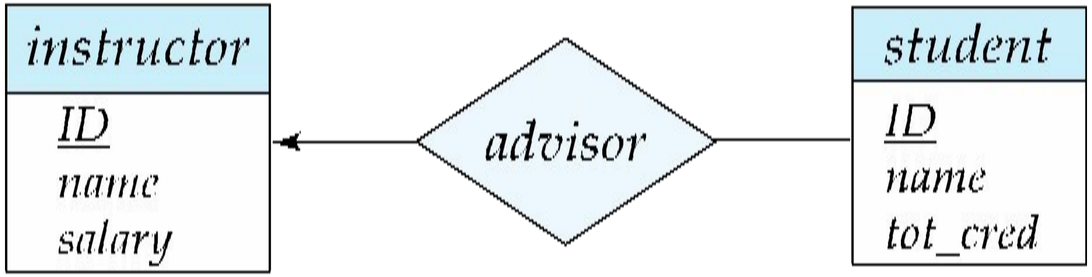


One to many

Note: Some elements in *A* and *B* may not be mapped to any elements in the other set

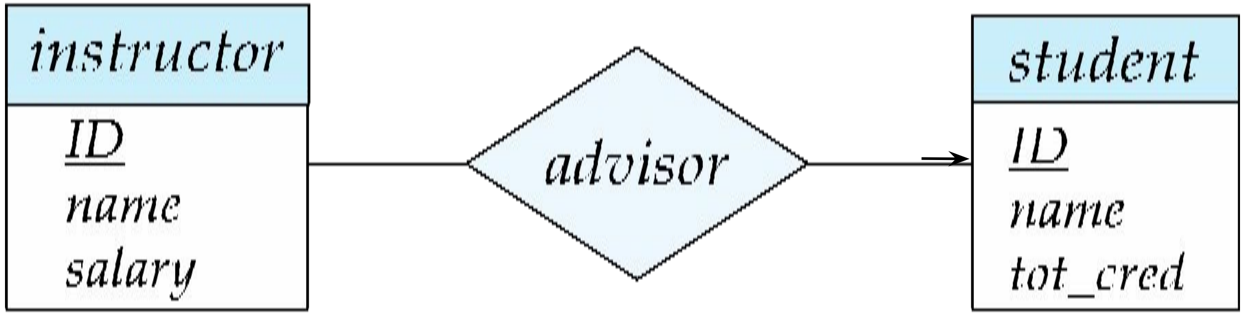
One-to-Many Relationship

- one-to-many relationship between an *instructor* and a *student*
  - an instructor is associated with several (including 0) students via *advisor*
  - a student is associated with at most one instructor via advisor,



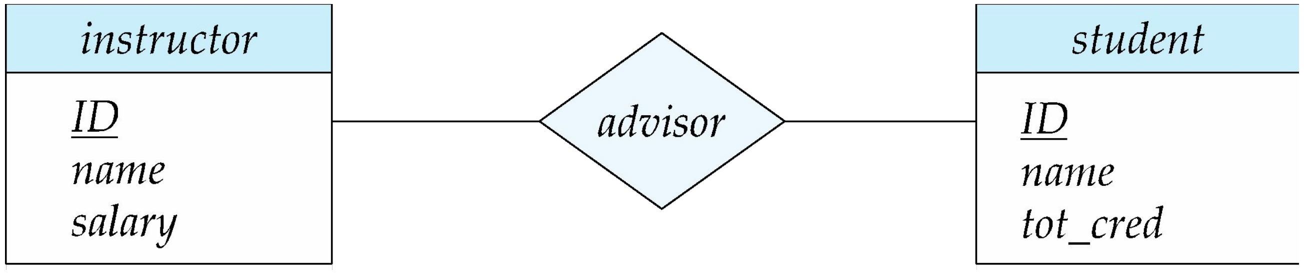
Many-to-One Relationships

- In a many-to-one relationship between an *instructor* and a *student,*
  - an instructor is associated with at most one student via *advisor*,
  - and a student is associated with several (including 0) instructors via *advisor*



Many-to-Many Relationship

- An instructor is associated with several (possibly 0) students via *advisor*
- A student is associated with several (possibly 0) instructors via *advisor*



Primary Key

- Primary keys provide a way to specify how entities and relations are distinguished. We will consider:
    - Entity sets
    - Relationship sets.
    - Weak entity sets

Primary key for Entity Sets

- By definition, individual entities are distinct.
- From database perspective, the differences among them must be expressed in terms of their attributes.
- The values of the attribute values of an entity must be such that they can uniquely identify the entity.
  - No two entities in an entity set are allowed to have exactly the same value for all attributes.
- A key for an entity is a set of attributes that suffice to distinguish entities from each other

Primary Key for Relationship Sets

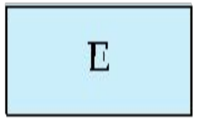
- To distinguish among the various relationships of a relationship set we use the individual primary keys of the entities in the relationship set.
  - Let *R* be a relationship set involving entity sets E1, E2, .. En
  - The primary key for R is consists of the union of the primary keys of entity sets E1, E2, ..En
  - If the relationship set *R* has attributes a1, a2, .., am associated with it, then the primary key of *R* also includes the attributes a1, a2, .., am
- Example: relationship set “advisor”.
  - The primary key consists of *instructor.ID* and s*tudent.ID*
- The choice of the primary key for a relationship set depends on the mapping cardinality of the relationship set.

Choice of Primary key for Binary Relationship

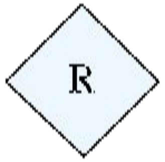
- Many-to-Many relationships. The preceding union of the primary keys is a minimal superkey and is chosen as the primary key.
- One-to-Many relationships . The primary key of the “Many” side is a minimal superkey and is used as the primary key.
- Many-to-one relationships. The primary key of the “Many” side is a minimal superkey and is used as the primary key.
- One-to-one relationships. The primary key of either one of the participating entity sets forms a minimal superkey, and either one can be chosen as the primary key.

E-R Diagram for a University Enterprise

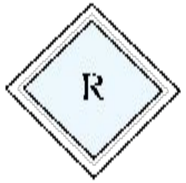
Summary of Symbols Used in E-R Notation



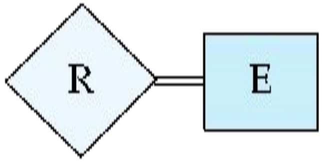
entity set



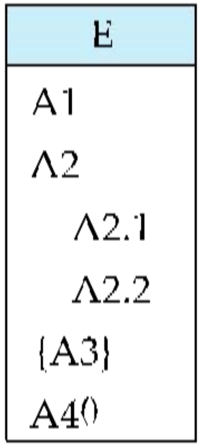
relationship set



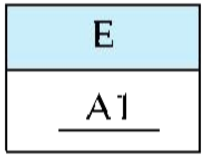
identifying
relationship set
for weak entity set



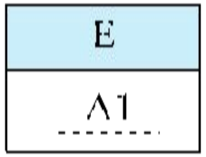
total participation
of entity set in
relationship



attributes:
simple (A1),
composite (A2) and
multivalued (A3)
derived (A4)



primary key



discriminating
attribute of
weak entity set

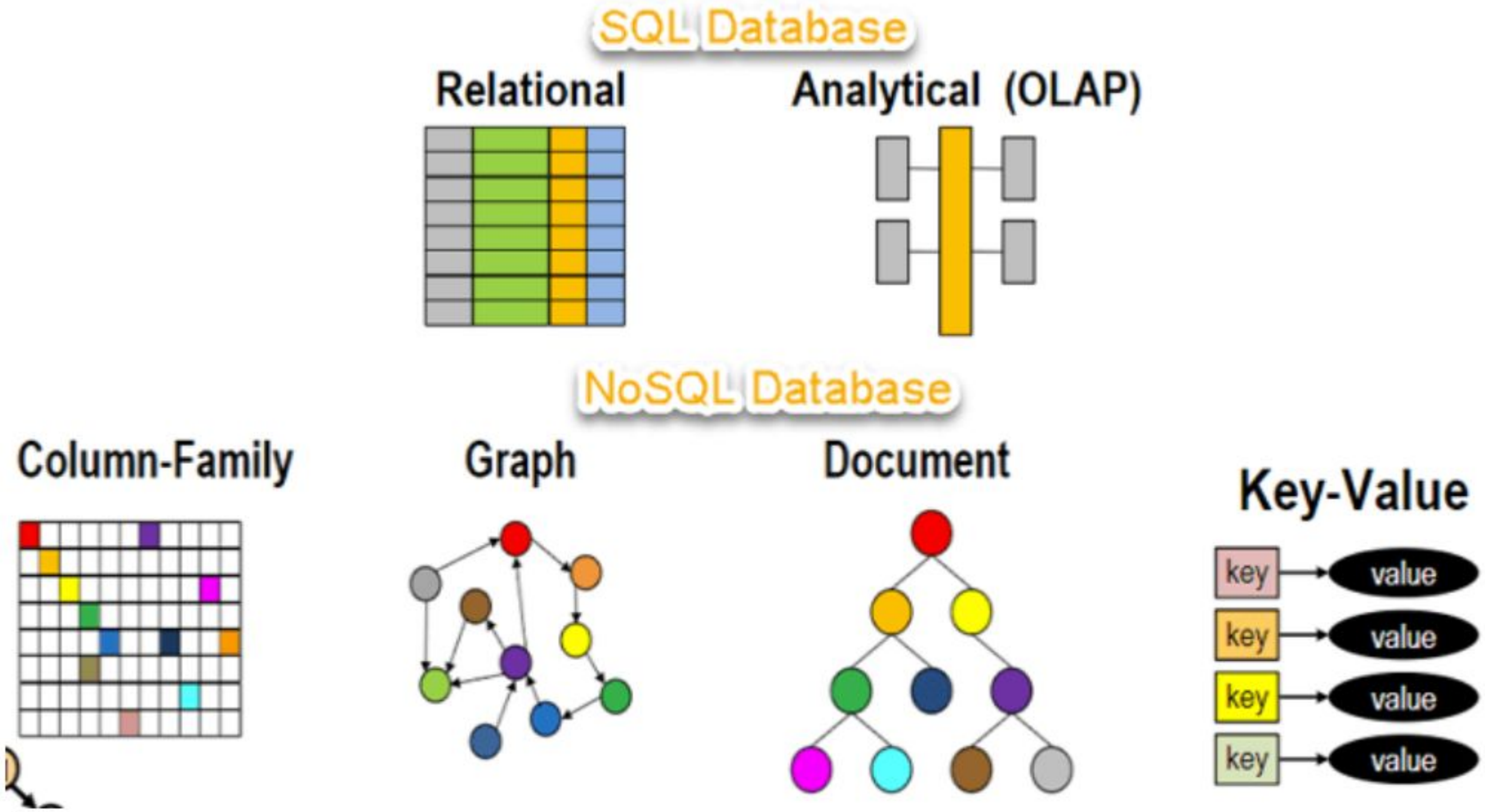
Introduction to NoSQL databases

Conventional relational databases were the product of a great deal of research and testing to obtain optimal storage structures. However, their flaw is that they have a rigid structure which cannot be altered if there are different specifications. To combat this problem, NoSQL databases were created.

Types of NoSQL databases

- **Column** - Data is stored in a columnar form. Some examples of this type of database are: Accumulo, Cassandra, Druid, Vertica etc.
- **Key Value** - These databases are organised as key value pairs, where each key appears exactly once. The keys are usually arranged in a sorted fashion. Examples of this type of databases are: Aerospike, ArangoDB, Couchbase, Dynamo etc.
- **Graph** - The databases are arranged in the form of a graph with the elements connected using the relations between them. Examples are AllegroGraph, InfiniteGraph,MarkLogic etc.
- **Document** - The database is stored in the form of documents that are accessed using a unique key. A single key references a document. Examples of Domument based databases are: Apache CouchDB, BaseX, Clusterpoint erc.

At a glance


SQL Database
Relational
Analytical (OLAP)
NoSQL Database
Column-Family
Graph
Document
Key-Value
key
value
key
value
key
value
key
value

Rating of different NoSQL databases

Data Model	**Performance**	**Scalability**	**Flexibility**	**Complexity**	**Functionality**
Document Oriented	HIGH	HIGH	HIGH	LOW	LOW
Graph Database	VARIABLE	VARIABLE	HIGH	HIGH	GRAPH THEORY
Key value database	HIGH	HIGH	HIGH	NONE	VARIABLE
Column Oriented	HIGH	HIGH	MODERATE	LOW	MINIMAL

Disadvantages of NoSQL databases

- The data is not as consistent in NoSQL databases as in traditional relational databases.
- Moreover, while SQL databases uphold the ACID properties and provide reliable transactions, the NoSQL databases often sacrifice the ACID properties for speed and scalability.
- Basically, both the SQL and NoSQL databases provide different functionalities and compliment each other in the long run.
- It is up to the user to choose whichever type of database suits their needs the best.

Differences between SQL and NoSQL

Key	SQL	NoSQL
Type	SQL databases are classified as Relational databases, i.e., RDBMS.	NoSQL databases are known as non-relational or distributed database.
Language	SQL databases use standard Structured Query Languages, as the name suggests. SQL is an industry-standard and very powerful language to execute complex queries.	NoSQL database has dynamic schema for unstructured data. The data stored in a NoSQL database is not structured. Data could be stored as document-oriented, column oriented, graph-based or organized as a Key-Value store.

Differences between SQL and NoSQL

Key	SQL	NoSQL
Scalability	SQL databases can extend their capacity on a single server by increasing their RAM, CPU or SSD. SQL databases are scalable vertically, as their storage could be increased for the same server by enhancing their storage components.	In order to increase the capacity of a NoSQL database, you would have to install new servers parallel to the parent server. NoSQL databases are horizontally scalable which means they can easily handle more traffic by adding new servers to the database, which makes them a great choice for large and constantly changing databases.
Schema	SQL databases have a fixed, pre-defined schema, which makes the data storage more rigid, static, and restrictive.	NoSQL databases don’t have a pre-defined schema, which makes them schema-less and more flexible.

Differences between SQL and NoSQL

Key	SQL	NoSQL
Internal implementation	SQL follows ACID (Atomicity, Consistency, Isolation and Durability) properties for its operations.	NoSQL is based on CAP (Consistency, Availability, and Partition Tolerance).
Data Storage	SQL databases can only be run on a single system and because of this, they don’t follow the distribution of data and hence they don’t support hierarchical storage of data.	NoSQL Databases can run on multiple systems, and hence, they support data distribution features like data repetition, partition, etc., making them the best option for hierarchical storage of data.
Type of Data	SQL databases are table-based databases which makes them better for multi-row transaction applications.	NoSQL is document-based, key-value pair, and graph databases, which makes them better when there are a lot of changes in the data.

Differences between SQL and NoSQL

Key	SQL	NoSQL
Performance and suitability	SQL databases are best suited for complex queries but are not preferred for hierarchical large data storage.	NoSQL databases are not so good for complex queries because these are not as powerful as SQL queries but are best suited for hierarchical large data storage.
Examples	SQL databases are implemented in both open source and commercial databases such as like Postgres & MySQL as open source and Oracle and Sqlite as commercial.	NoSQL is purely open source. Some of its famous implementation are MongoDB, BigTable, Redis, RavenDB, Cassandra, Hbase, Neo4j, and CouchDB.

Why is NoSQL so popular?

- **Flexibility** – With a SQL database, the way data is stored is much more rigid and set. But with NoSQL, data can be stored in a less structured way, without having to follow strict schemas. This design makes it possible to come up with new ideas and make applications quickly. Without having to worry about schemas, developers can focus on making systems that help their customers more.
- **High Performance** – NoSQL databases are utilized in applications that collect terabytes of data every day and require a highly interactive user experience. Because NoSQL databases can also ingest data and deliver it rapidly and reliably, these databases are employed in apps that collect the data.

Why is NoSQL so popular?

- **High Functional** – NoSQL databases are developed specifically for distributed data stores that have exceptionally high requirements for the amount of data that can be stored. Because of this, NoSQL is the best option for applications dealing with huge data, real-time web apps, online shopping, online gaming, Internet of things, social networks.
- **Scalability** – Instead of adding more servers to scale up, NoSQL databases can use cheap hardware to scale out. This can handle more traffic so that it can keep up with demand with no downtime. NoSQL databases can get bigger and more powerful by scaling out, which is why they are the best choice for data sets that are always changing.

Semi-structured and Unstructured Data

Structured data	Semi-structured data	Unstructured data	
What is it?	Data with a high degree of organization, typically stored in a spreadsheet-like manner	Data with some degree of organization	Data with no predefined organizational form and no specific format
To put it simply	Think of a spreadsheet (e.g. Excel) or data in a tabular format	Think of a TXT file with text that has some structure (headers, paragraphs, etc.)	Essentially anything that is not structured or semi-structured data (which is a lot)
Example formats	•Excel spreadsheets •Comma-separated value file (.csv) •Relational database tables	•Hypertext Markup Language (HTML) files •JavaScript Object Notation (JSON) files •Extensible Markup Language (XML) files	•Images such as .jpeg or .png files •Videos such as .mp4 or m4a files •Sound files such as .mp3 or .wav files •Plain text files •Word files •PDF fi

Semi-structured and Unstructured Data

Structured data	Semi-structured data	Unstructured data	
Characteristics	•Data is structured in a spreadsheet-like manner (e.g. in a table) •Within that table, entries have the same format and a predefined length and follow the same order •Is easily machine-readable and can therefore be analysed without major pre-processing of the data •It is commonly said that around 20% of the world's data is structured	•Data is stored in files that have some degree of organization and structure •Tags or other markers separate elements and enforce hierarchies, but the size of elements can vary and their order is not important •Needs some pre-processing before it can be analysed by a computer •Has gained importance with the emergence of the World Wide Web	•Data that can take any form and thus be stored as any kind of file (formless) •Within that file, there is no structure of content •Typically needs major pre-processing before it can be analysed by a computer, but often easily consumable for humans (e.g. pictures, videos, plain texts) •Most of the data that is created today is unstructured

Why is the distinction important?

- As you can see, the distinction of structured, semi structured and unstructured data breaks down to how organized your data is. But why is the degree of organization so important? There are many reasons, but the two reasons that stand out are:
- Machine-readability
- Implications for data storage
- If data follows a rigorous structured like in a spreadsheet from which there is no deviation, then this make the data highly machine-readable. As a result, we can analyze even large datasets very easily by harnessing computer power.
- In contrast, if data does not follow a rigorous structure, it might still be easy for us consume as humans but is usually not very machine-readable. So to harness computer power to analyze it will be much more difficult.

Why is the distinction important?

- Consider this example to grasp this idea: Let’s say I am organizing a workshop with 10 participants and I want to create list of basic workshop participant data.
- On the one hand, I could have every participants enter their name and age into an Excel sheet upon arrival. Or I could have everyone write down their name and age on a name tag (i.e. physical piece of paper) and then take a picture of all name tags.

Why is the distinction important?

- In the first case I can directly use my computer and perform operation on the data. For example, I could use a simple tool like Excel to display all participants older than 40 years, or I could filter for a participant name to look up their age. I cannot do that with a computer if I had an image of all participant’s name tags. I could of course do it manually, but there is no software tool where I can tell my computer to give me the age of participant X (we are getting there with image classification and object detection, but I hope you get the point).

When to use a NoSQL Database?

In the late 1990s, almost every type of applications were based on SQL type databases. But the rise of Internet has changed everything in the context of application development. These changes have influenced many organizations of all sizes to adopt a new database technology called NoSQL. Below are some of the facts when we should consider using a NoSQL databases.

1) If you want a faster development

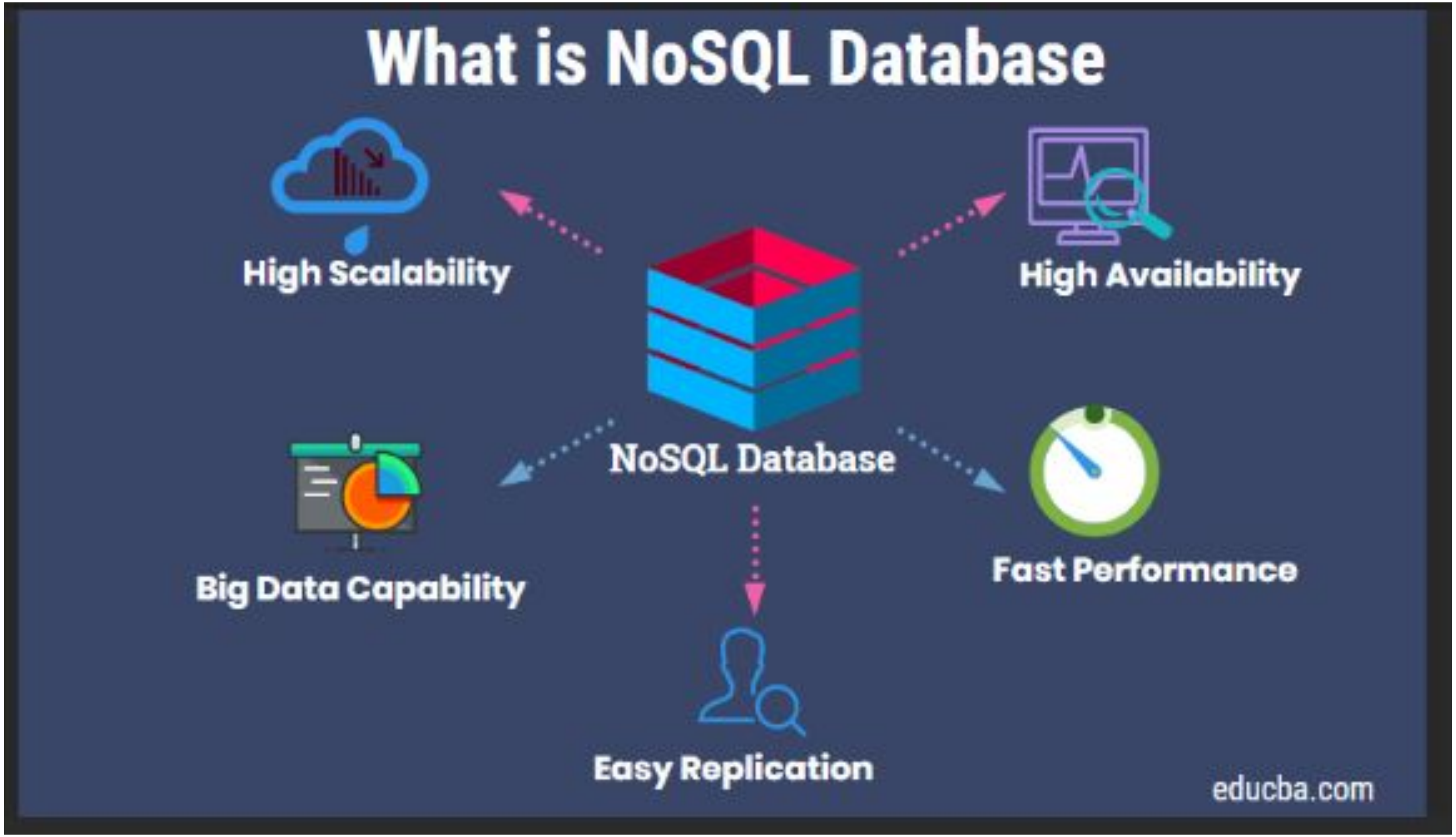
2) If you want structured and semi-structured data storage

3) If you have a requirement of scale-out architecture

4) If you want to handle huge amount of data

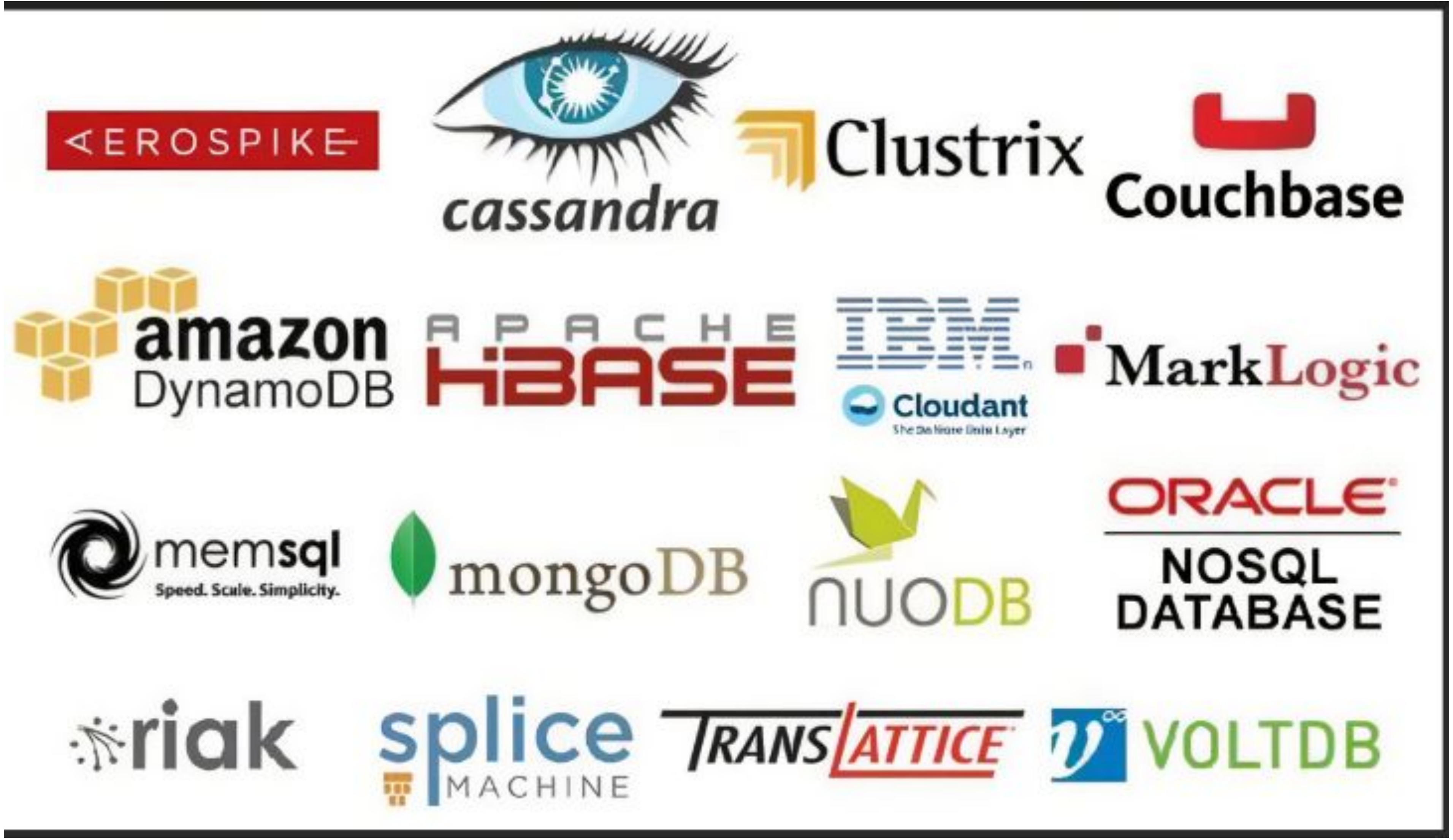
5) If your application is using microservices and real-time streaming

At a glance



What is NoSQL Database
High Scalability
High Availability
NoSQL Database
Big Data Capability
Fast Performance
Easy Replication
educba.com

At a glance

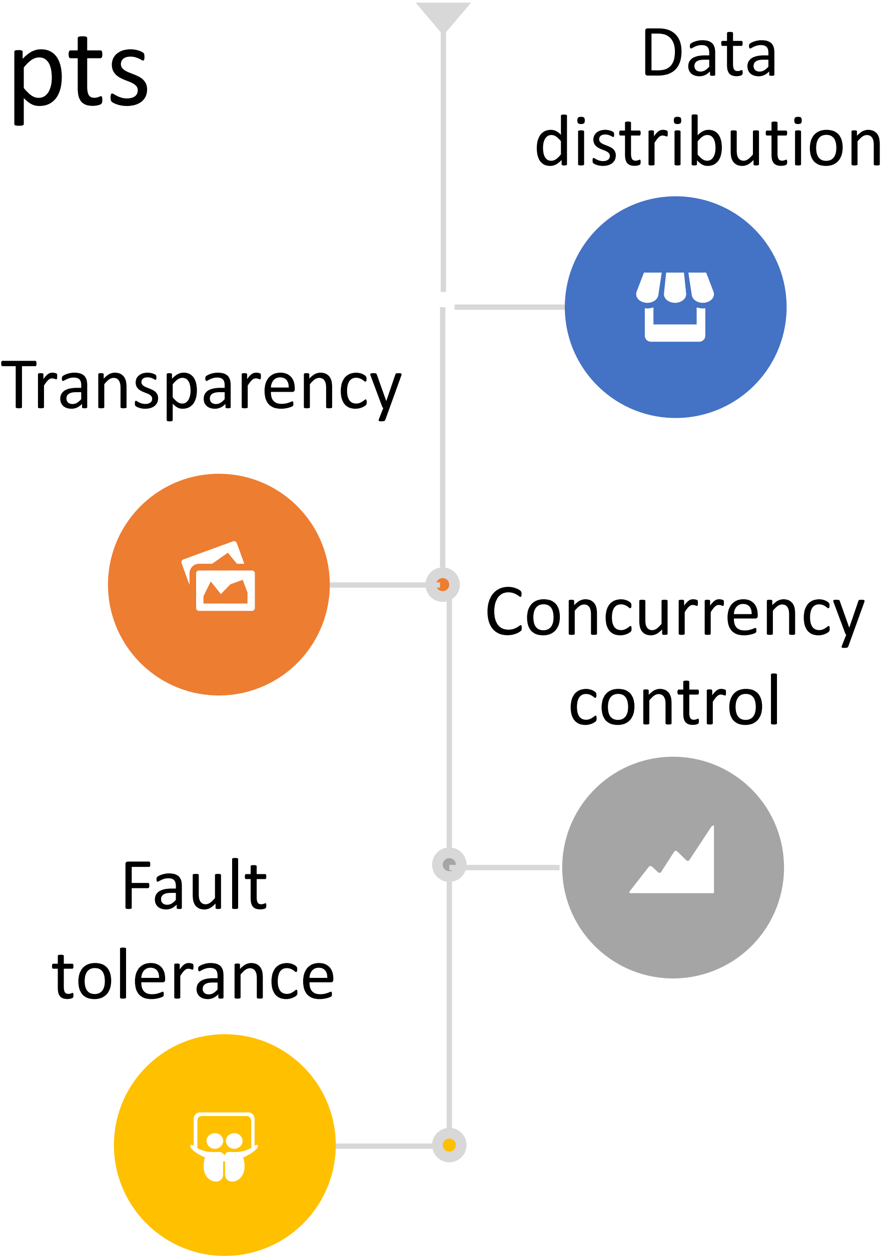

AEROSPIKE
cassandra
Clustrix
Couchbase
amazon DynamoDB
APACHE HBASE
IBM
Cloudant
MarkLogic
memsql
Speed. Scale. Simplicity.
mongoDB
NUODB
ORACLE
NOSQL DATABASE
riak
splice MACHINE
TRANSLATTICE
VOLTDB

Distributed databases

- Collection of multiple, logically interrelated databases distributed over a computer network.
- Data is stored, accessed, and managed efficiently across multiple locations.
- Scalability, fault tolerance, and improved performance.

Examples
MongoDB, Cassandra, Couchbase, DynamoDB and Azure CosmoDB.

Fundamental concepts



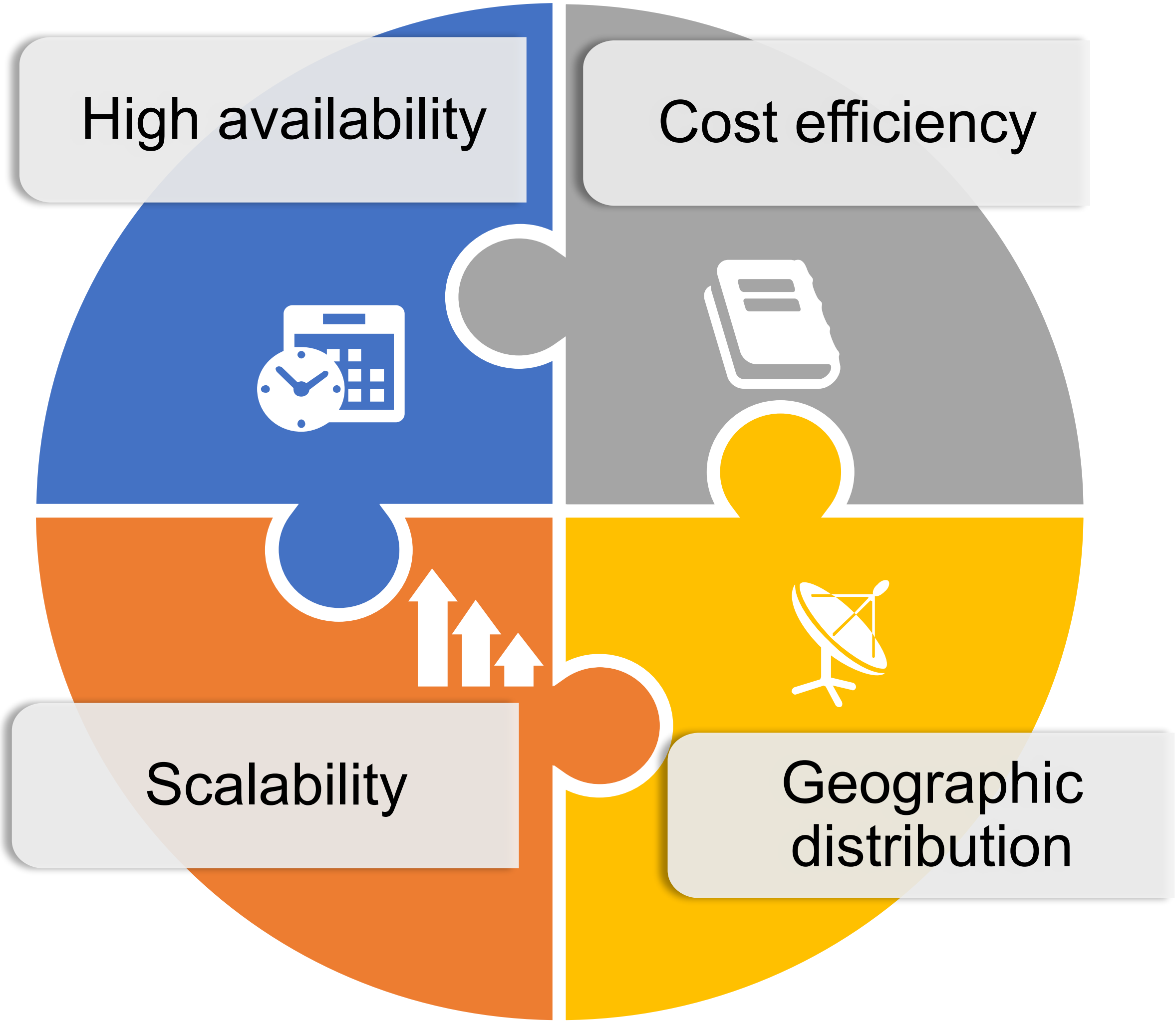
Data distribution

Transparency

Concurrency control

Fault tolerance

Need for Distributed Databases



High availability

Cost efficiency

Scalability

Geographic distribution

Key Challenges

Data consistency

Ensuring data consistency as well as concurrent access and updates

Network latency

Network delays demand optimization techniques

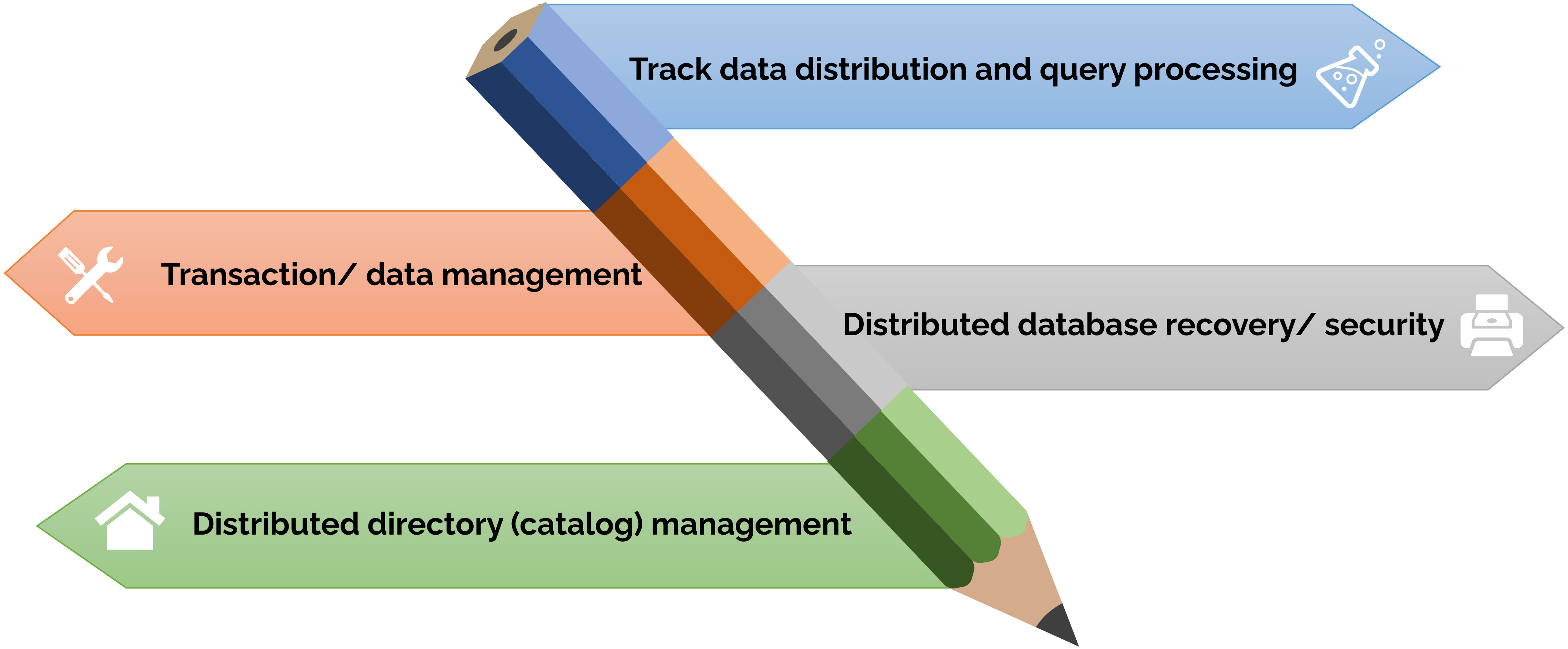
Security and privacy

Protecting sensitive data across distributed environments from unauthorized access

Complexity

Increased complexity in configuration, monitoring, and troubleshooting

Additional functions of DDBMS



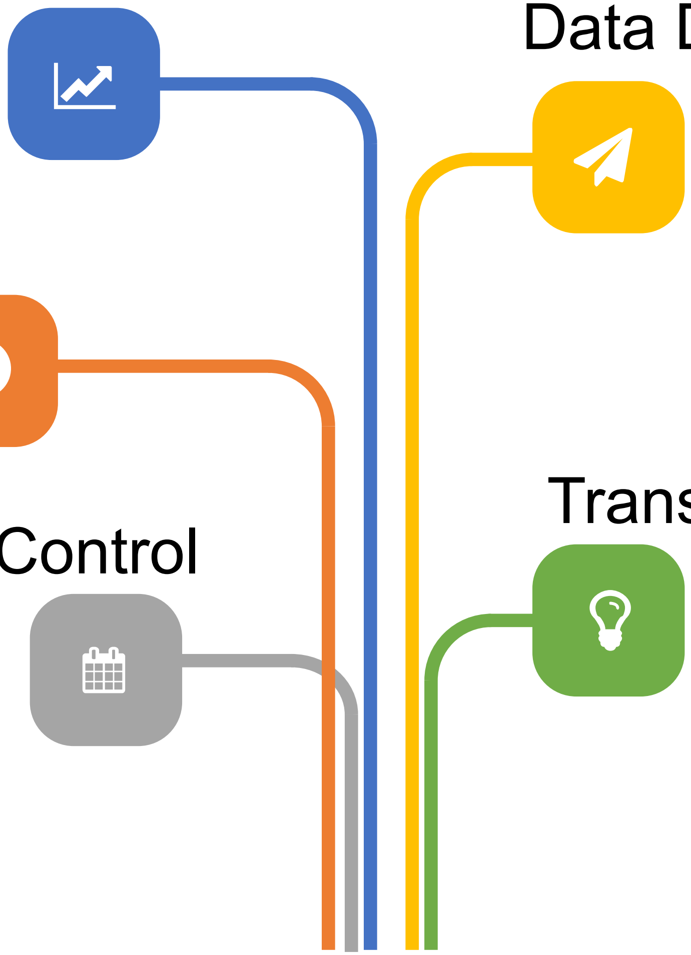
Track data distribution and query processing

Transaction/ data management

Distributed database recovery/ security

Distributed directory (catalog) management

Types of DDBMS



Architecture

1. Homogeneous DDBMS
2. Heterogeneous DDBMS

Control

1. Centralized DDBMS
2. Decentralized DDBMS

Replication Control

1. Synchronous Replication
2. Asynchronous Replication

Data Distribution

1. Replicated DDBMS
2. Partitioned DDBMS

Transparency

1. Location Transparency
2. Fragmentation Transparency