

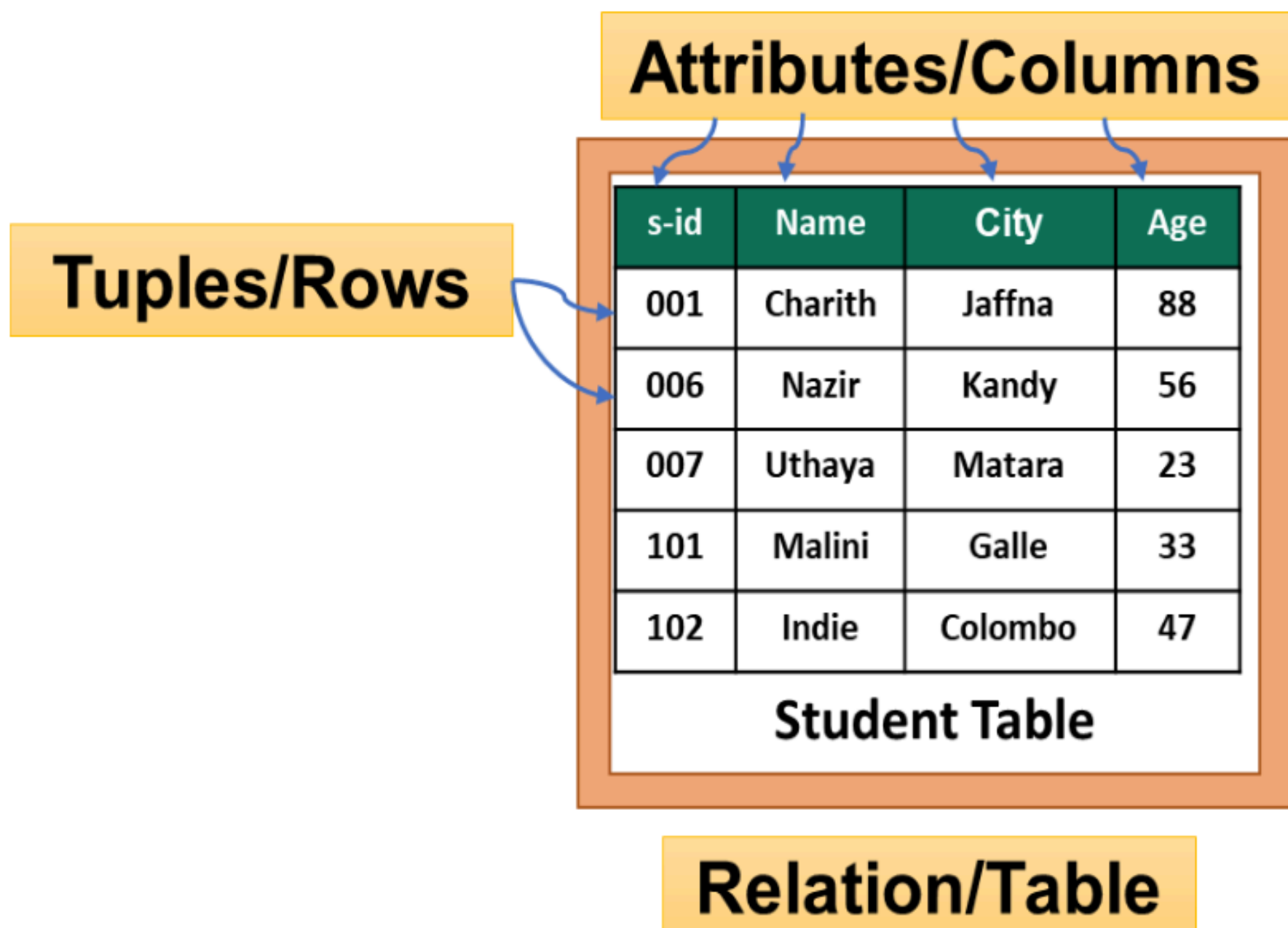
3.2 Relational vs. Non-Relational DBs



3.2 Relational vs. Non-Relational DBs

Fundamentals of RDBMS

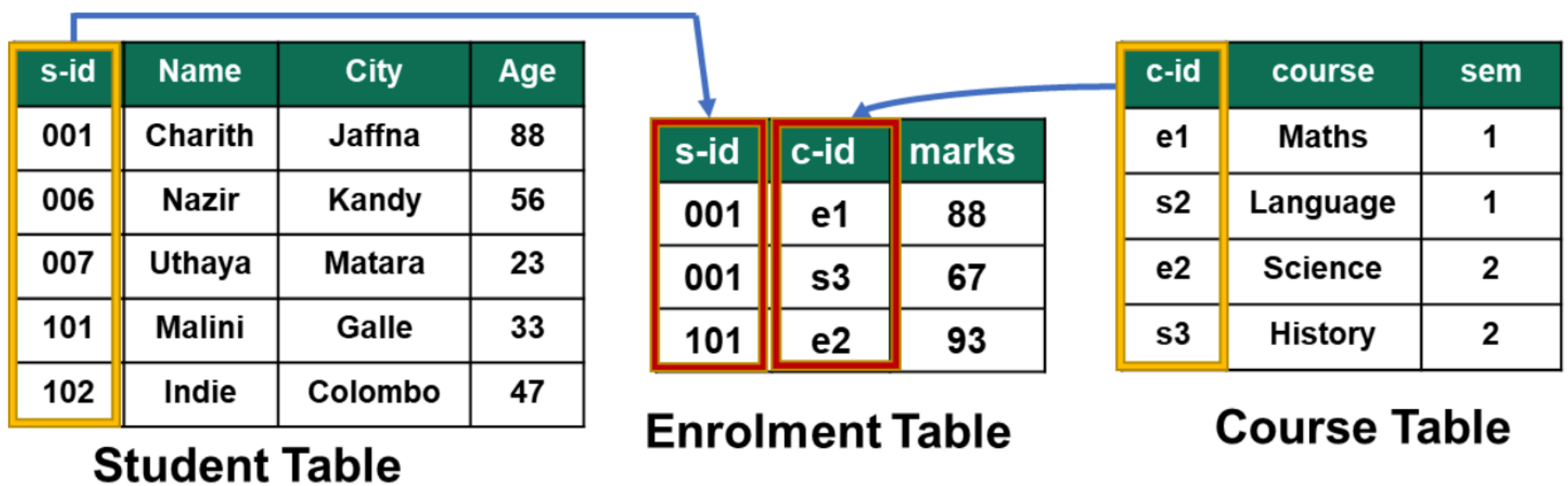
- In RDBMS, a table(Relation) organizes data into rows and columns.
- The columns are known as attributes.
- The rows are known as records/tuples.
- Consider the Student Table here.
student-id, name, city and age are the attributes.
- Student table is a relation.
 - Formally relation is a set of tuples.
 - Which means no duplicate tuple is allowed.
- Each table has a unique primary key that identifies the information in a table.



Relationship

- The relationship between tables can be set via the use of foreign keys.
 - A field in a table that links to the primary key of another table is what is known as a Foreign Key.
 - Here Student table and Course table are brought into the Enrollment table.
 - For each student, the course he took and the marks that he obtained are included in the Enrollment table.
 - Student-id is the primary key of the Student table.

- Course-id is the primary key of the Course table.
- Student-id and Course-id columns from 'Enrolment Table' are foreign keys. (Both becomes foreign keys for the enrollment table.)
- When put together, Student-id and Course-id will form the primary key of the Enrollment table.



Index

One important feature that is required by a DBMS is the ability to fetch a record faster. Indexes support that feature.

- Index.
 - Indexing is used to optimize the performance of a database by minimizing the number of disk accesses.
 - The index is a type of data structure.
 - It is used to locate and access the data in a database table quickly.

SQL

- SQL - Structured Query Language
- SQL is the standard language for dealing with Relational Databases.
- Most of the programmers interact with RBDMS using SQL.
- SQL is used to insert, search, update and delete database records.
 - C (Create) ← → Insert
 - R (Read) ← → Select
 - U (Update) ← → Update
 - D (Delete) ← → Delete
 - This is also abbreviated as CRUD.
- SQL keywords are not case sensitive.

Domains & Cardinalities

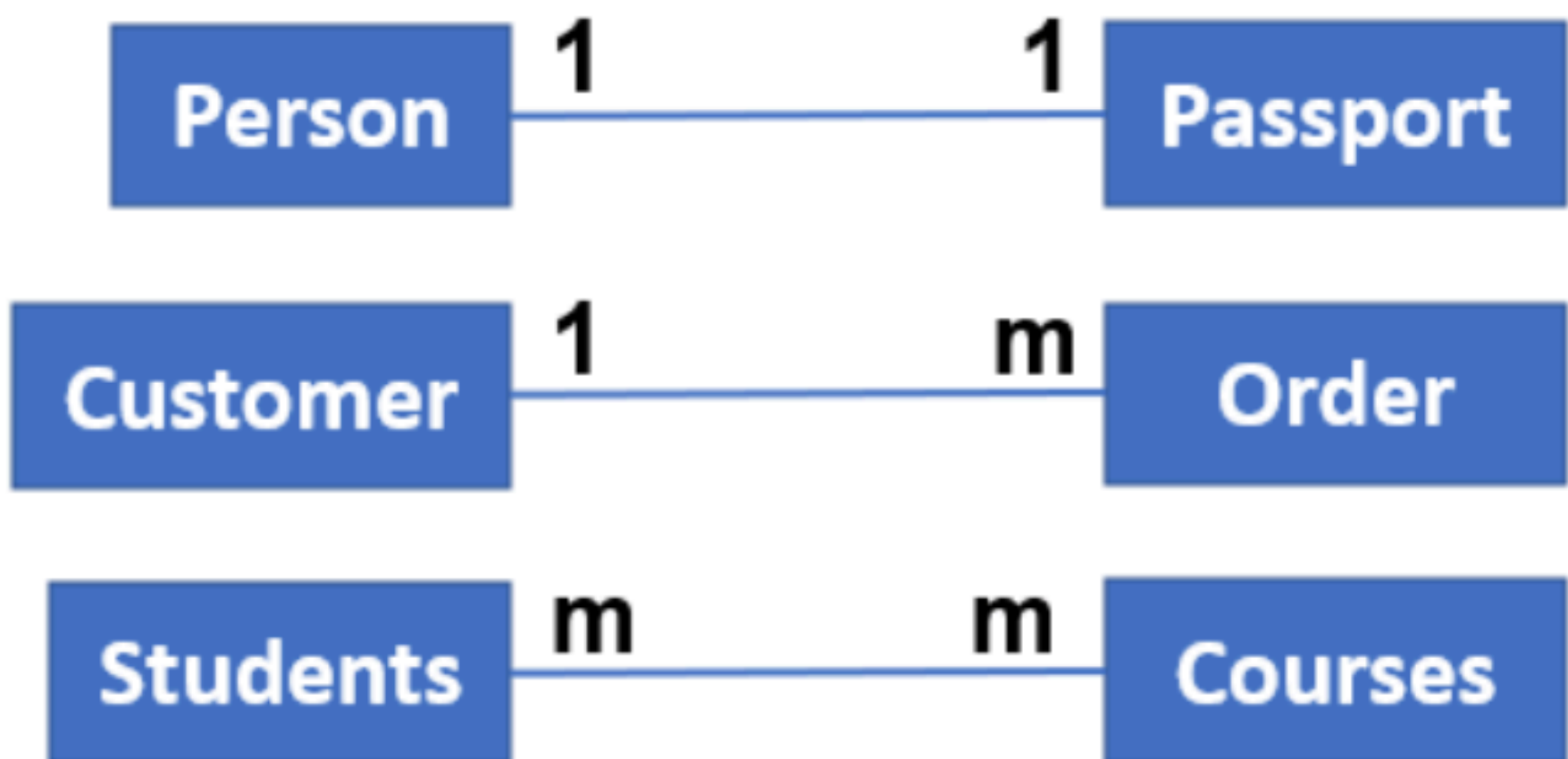
- **Domains** - All the possible allowable values for an attribute.
 - This is slightly different to the data type of the attribute.
 - For example, a field may have an integer number data type, which defines that it can only allow whole numbers to be entered.
 - A classic example would be Age.
- **Cardinality** - How unique an attribute is in terms of its data values. Some attributes will have a wide range of different data values entered.
 - For example, the primary key field will have a completely unique value for every record.
 - Where there is a large percentage of unique values, this is known as **High Cardinality**.
 - Where there are a lot of repeated values across the entities tuples, this is known as having **Low Cardinality**.

Integrity Constraints

- Integrity constraints are used to ensure the accuracy and consistency of data in your relational database.
- The main two types of integrity constraints are
 - **Entity Integrity**
 - This rule states that every entity (table) must have a primary key field and the data in that primary key field must be unique to each record.
 - **Referential Integrity**
 - This rule states that every foreign key value must match the primary key value of a record in another table, or must be null.

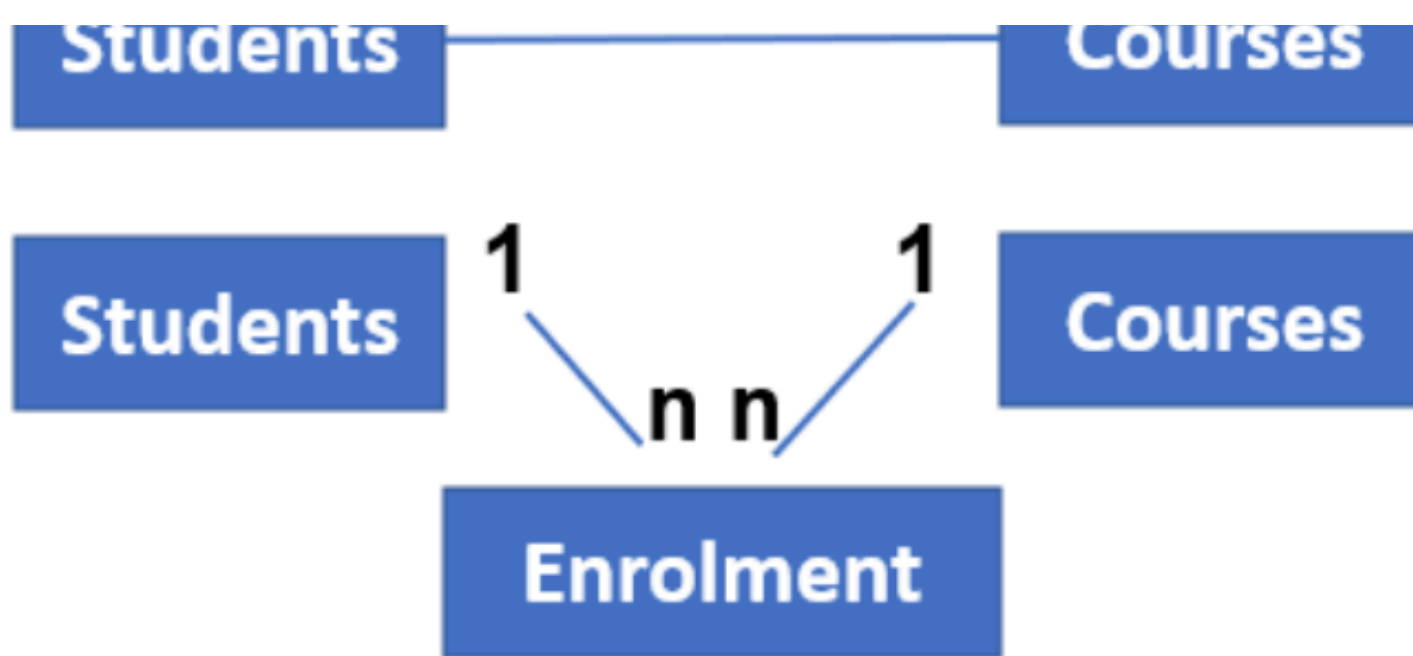
Entity Relationships

- **One-to-One**
 - A single record of a table is related to a single record of another table.
 - This could be the relationship between a person and their passport.
- **One-to-Many**
 - A single record of a table is related to more than one record in another table.
 - In this relationship, the foreign key would appear on many sides of the relationship.
 - This could be the relationship between a customer and their orders.
- **Many-to-Many**
 - More than one record of a table is related to more than one record of another table.
 - This could be the relationship between students and their courses.



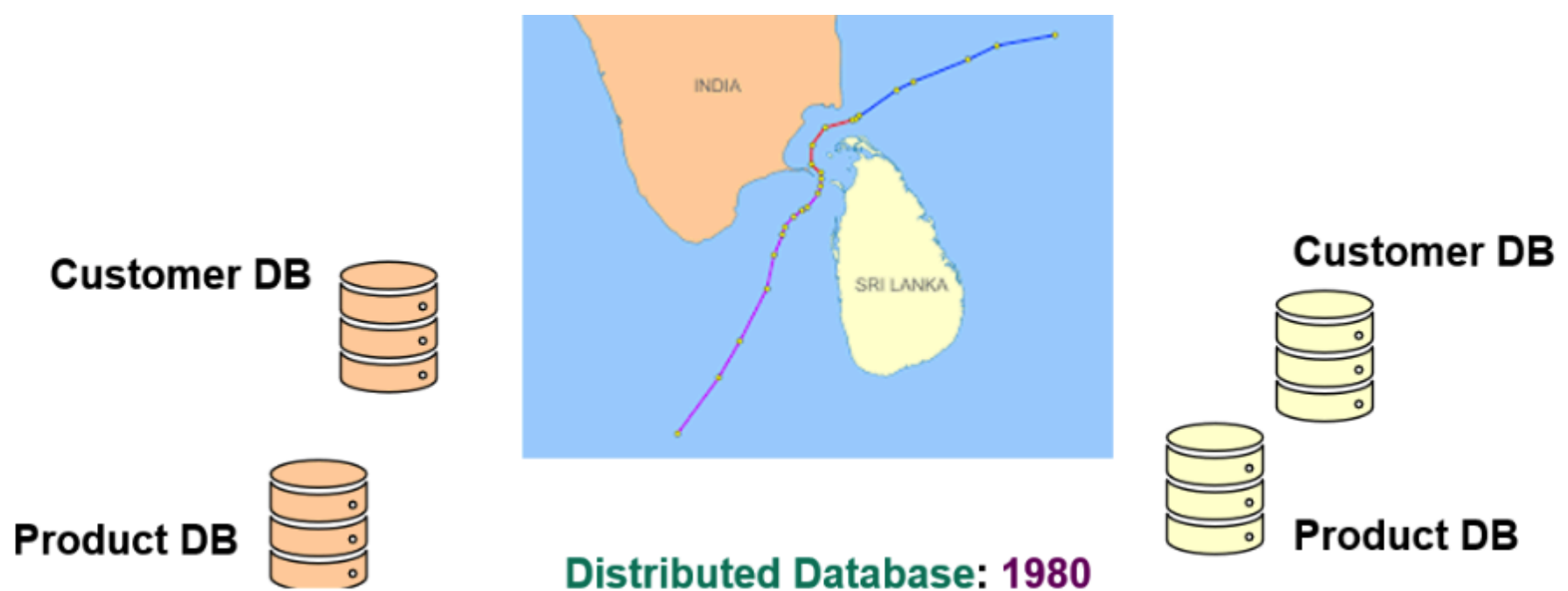
- Unfortunately, many-to-many relationships cannot be represented by a relational database.
- In Navigational Model we learn it supports one-to-many relationship model by default. However, to envision many-to-many we need a hack. The same thing is applied to the relational model as well. To achieve a many-to-many relationship directly is not trivial in RDB.
- Where a many-to-many relationship exists, we must resolve this into two one-to-many relationships.
- Example:





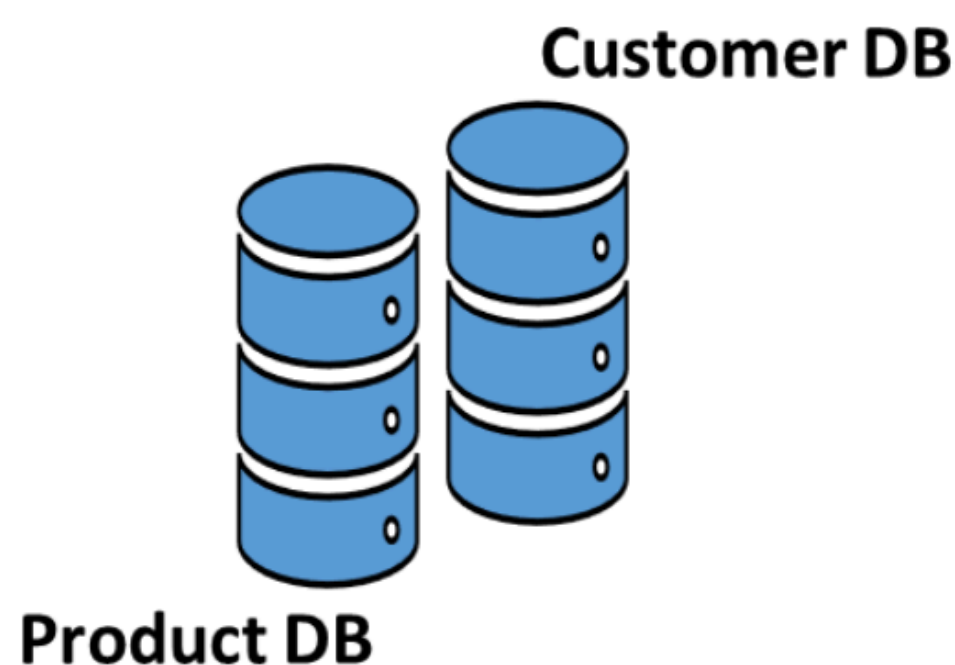
What Happens When Businesses Expands?

Let's say that there is a business that is being built in Sri Lanka with its customers and products, and they have a database that stores the information about their customers as well as the products, orders and business operations etc. Now they want to venture into India. They want to grow their business and they have started that. Now they've grown bigger. But there may be some differences in the way they do business in India. Subsequently, the rules and the types of information that you want to store in the database in India might be slightly different. Because the product is the same. How do we support it? As the business keeps expanding, we may not be able to keep storing in our traditional RDBMS. The solution for this is Distributed Database which was introduced in 1980.



Centralized Vs Distributed DBs

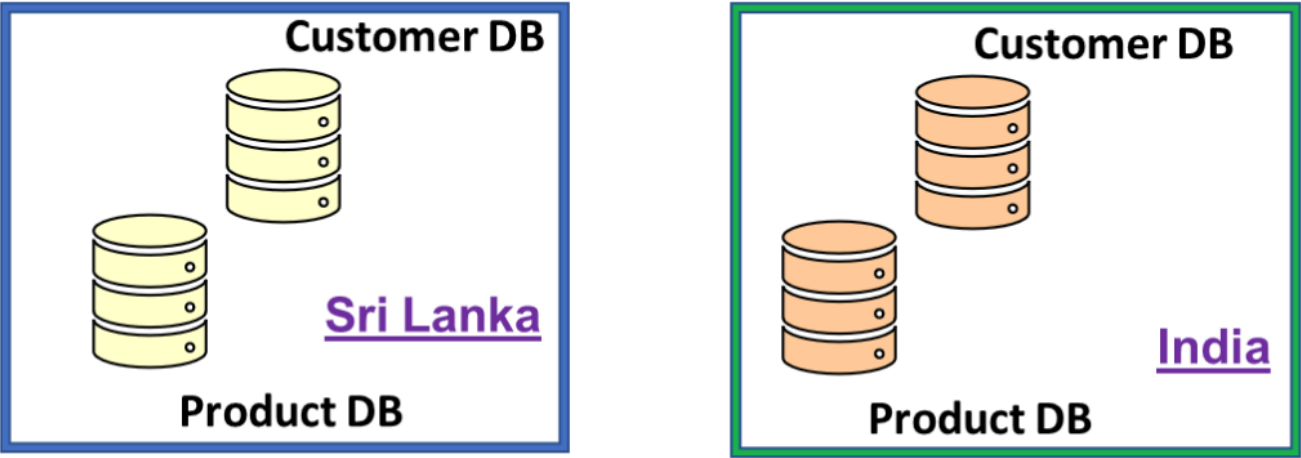
- Centralised
 - Both Sri Lankan and Indian data together.



- **Distributed**

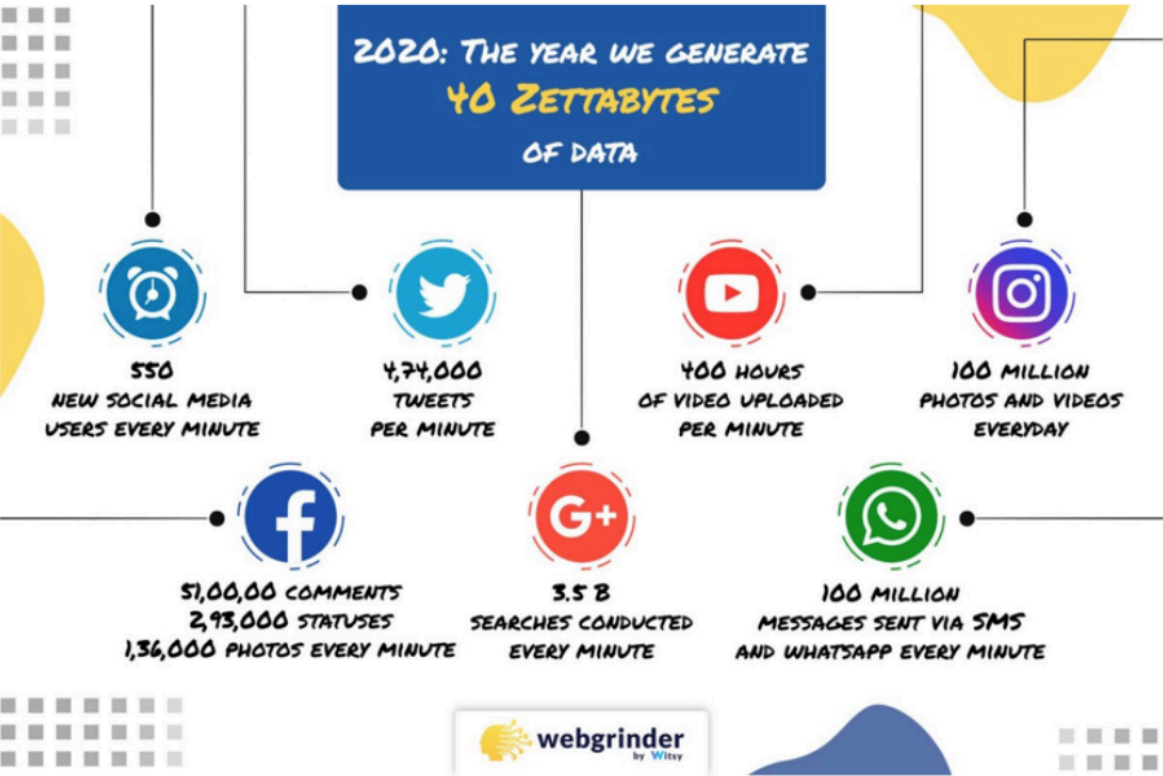
◦ Sri Lankan customer data in Sri Lankan site, and Indian customer data in India. Product data is duplicated to Indian site and

- Sri Lankan customer data in Sri Lankan site and Indian customer data in India. Product data is duplicated to Indian site and synchronized.



Distributed DBS	Centralized DBS
Multiple DBs located at different sites.	Single central probably large DB.
Allows multiple users to access and manipulate.	Bottlenecks when multiple users access the same data together.
Data is delivered quickly from the nearest location.	Data may take longer for users far from the central system.
If one site fails, data is still retrievable.	If one site fails, the whole system is down.
Multiple copies of data need to be synchronized.	Simpler to update and manage.

The Social Media Use case: Data Deluge



BigData

BigData is characterised using 4 Vs.

- Volume
 - Large amounts of data
- Variety
 - Semi-Structured: XML, JSON...
 - Unstructured: Video, Text, Images
- Velocity

- Large amounts of data generated every minute.
- Veracity
 - Greater uncertainty about the quality of that data.
 - It is estimated that over 15% of accounts on Twitter are automated bot accounts.

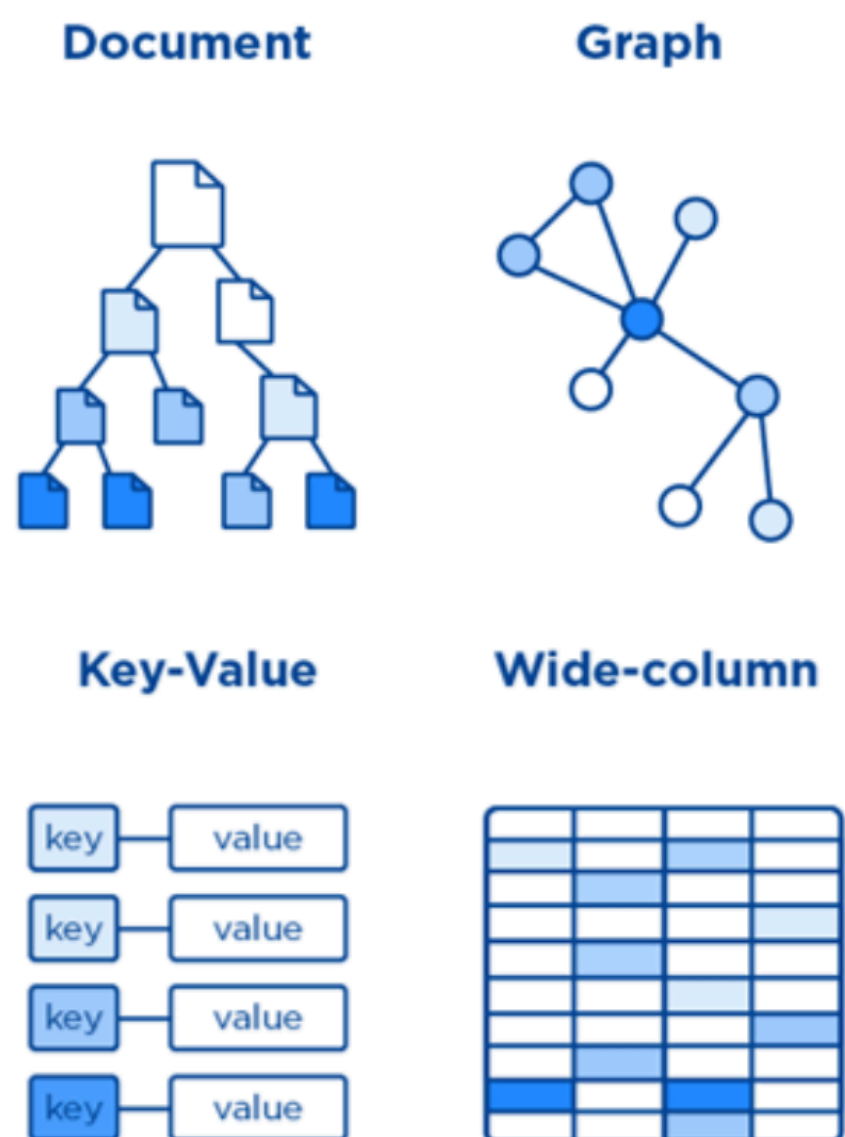
Why RDBMS Fails for BigData?

- Volume
 - Scaling is very hard to achieve in RDBMS on par with Big Data Volume.
- Velocity
 - Supporting the speed and access pattern of Big Data is a challenge.
- Variety
 - Storing and processing unstructured data is not native.

In short, IT is not designed for Big Data use cases but traditional 'operational' use-cases.


Meet NoSQL


- NoSQL databases (AKA "Not Only SQL") are non-tabular databases and store data differently than relational tables.
- Four types of NoSQL DBs:
 - Document Store
 - Data and metadata are stored hierarchically in JSON-based documents inside the database.
 - Key-Value Store
 - The simplest of the NoSQL databases, data is represented as a collection of key-value pairs.
 - Wide-Column Store
 - Related data is stored as a set of nested-key/value pairs within a single column.
 - Graph Store
 - Data is stored in a graph structure as a node, edge, and data properties.





Jump to...



GET IN TOUCH

 University of Moratuwa
Centre for Open & Distance Learning
CODL

 011 308 2787/8

 011 265 0301 ext. 3850,3851

 open@uom.lk

-  [University Website](#)
-  [CODL Website](#)

[Data retention summary](#)