Introduction to RDBMS

Introduction to RDBMS

- **Data** is nothing but facts and statistics stored or free flowing over a network, generally it's raw and unprocessed.
- A **Relational Database** is a collection of related data organized in a way that data can be easily accessed, managed and updated. Database can be software based or hardware based, with one sole purpose, storing data. Some of popular Database Systems are,
  - MySQL
  - Oracle
  - SQL Server
  - IBM DB2
  - PostgreSQL

Characteristics of RDBMS

- **Data stored into Tables:** Data is never directly stored into the database. Data is stored into tables, created inside the database. DBMS also allows to have relationships between tables which makes the data more meaningful and connected, which can be easily understandable about the type of data is stored and where by looking at all the tables created in a database.
- **Reduced Redundancy:** In the modern world hard drives are very cheap, but earlier when hard drives were too expensive, unnecessary repetition of data in database was a big problem. But DBMS follows Normalization which divides the data in such a way that repetition is minimum.
- **Data Consistency:** On Live data, i.e. data that is being continuously updated and added, maintaining the consistency of data can become a challenge. But DBMS handles it all by itself.

Characteristics of RDBMS

- **Support Multiple user and Concurrent Access:** DBMS allows multiple users to work on simultaneously (update, insert, delete data) at the same time and still manages to maintain the data consistency.
- **Query Language:** DBMS provides users with a simple Query language, using which data can be easily fetched, inserted, deleted and updated in a database.
- **Security:** The DBMS also takes care of the security of data, protecting the data from un-authorised access. In a typical DBMS, we can create user accounts with different access permissions, using which we can easily secure our data by restricting user access.
- DBMS supports **transactions**, which allows us to better handle and manage data integrity in real world applications where multi-threading is extensively used.

ER Model

- **Entity-Relationship** model is a model used for design and representation of relationships between data.
- An **Entity or Table** is generally a real-world object which has characteristics and holds relationships in a DBMS. For **a School Management Software**, we will have to store **Student** information, **Teacher** information, **Classes**, **Subjects** taught in each class etc.
- An **Attribute or Field** is a smaller entity in the Table. For example, the **Student** entity can have **Roll Number, Name, Age, etc** as attributes. There are multiple type of Attributes, they are,
  - **Simple Attributes :** The attributes with values that are atomic and cannot be broken down further are simple attributes. For example, student's **age**.
  - **Composite attribute:** A composite attribute is made up of more than one simple attribute. For example, student's address will contain, house no., street name, pincode, etc.
  - **Derived attribute:** These are the attributes which are not present in the whole database management system, but are derived using other attributes. For example, *average age of students in a class*.

ER Model

- **Keys:** Keys are very important part of Relational database model. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table. A Key can be a single attribute or a group of attributes, where the combination may act as a key.
  - Following are the types of Keys:
    - Super Key
    - Candidate Key
    - Primary Key
    - Composite Key

Let us see the differences between these keys using an example.

In the table defined above super key would include `student_id`, `(student_id, name)`, `phone` etc.

ER Model

student_id	name	phone	age
1	Akon	9876723452	17
2	Akon	9991165674	19
3	Bkon	7898756543	18

- **Super Key** is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a superset of Candidate key. In the table defined above super key would include **student_id, (student_id, name), phone** etc.
- In the above example, the records are uniquely identified using,
  - Student_id
  - Phone
  - (Student_id, name) or (Name, phone). We can't identify the record uniquely using Name alone.

In the table defined above super key would include `student_id`, `(student_id, name)`, `phone` etc.

ER Model

- **Candidate keys** are defined as the minimal set of fields which can uniquely identify each record in a table. It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table.
- In our example, **student_id** and **phone** both are candidate keys for table Student.
  - A candidate key can never be NULL or empty. And its value should be unique.
  - There can be more than one candidate keys for a table.
  - A candidate key can be a combination of more than one column (attributes).

In the table defined above super key would include `student_id`, `(student_id, name)`, `phone` etc.

ER Model

- **Primary Key: Primary key** is a candidate key that is most appropriate to become the main key for any table. It is a key that can uniquely identify each record in a table.
- In our example, **student_id** is an appropriate candidate key to become a **Primary Key** for table Student.
- In the example, even though **Phone** is also a candidate key, in the real world there is a possibility that more than one student can use the same phone number in case of siblings are studying in the same school.

In the table defined above super key would include `student_id`, `(student_id, name)`, `phone` etc.

ER Model

- **Composite Key:** Key that consists of two or more attributes that uniquely identify any record in a table is called Composite key. But the attributes which together form the Composite key are not a key independently or individually.
- In the Student Management System, we need an entity to record subject wise marks for all the students. The **Student_Marks** table can have (student_id, subject_id, marks, total_marks, exam_name) columns.
- In the above table, (student_id, subject_id) will form the **Primary key** to uniquely identify the record since a student will study more than one subject and a subject is studied by more than one student.

In the table defined above super key would include `student_id`, `(student_id, name)`, `phone` etc.

ER Model

- **Secondary or Alternative key :** The candidate key which are not selected as primary key are known as secondary keys or alternative keys.
- **Non-key Attributes:** Non-key attributes are the attributes or fields of a table, other than candidate key attributes/fields in a table.
- **Non-prime Attributes:** Non-prime Attributes are attributes other than Primary Key attribute(s)..

ER Model

- **Relationship:** When an Entity is related to another Entity, they are said to have a **Relationship**. For example, A **Class** Entity is related to **Student** entity, because students' study in classes, hence this is a relationship.
  - Relationships are many types:
    - One-to-One
    - One-to-Many
    - Many-to-One
    - Many-to-Many

ER Model

- **Relationship:** When an Entity is related to another Entity, they are said to have a **Relationship**. For example, A **Class** Entity is related to **Student** entity, because students' study in classes, hence this is a relationship.
- Relationships are many types:
  - **One-to-One** : Suppose, we plan to store Students' Parent details in student_parent Entity, then one student will have only one record associated in student_parent table. Then it is an one-to-one relationship.
  - **One-to-Many : Student** and **Courses** entities will have one-to-many relationship. One student can study more than one course.
  - **Many-to-One : Student and Courses** entitles will also have many-to-one relationship. One Course will have more than one Student.
  - **Many-to-Many :** Many students can study more than one course and Course also have many students. Then, it is many-to-many relationship. In the e-r model, the many-to-many relationship is resolved using an additional entity to make it a one-to-many relationship. In the above example, we can have student_course table with student_id and Course_id (composite key) as a Primary key, which resolves the many-to-many relationship.

ER Model

- **Many-to-One : Student and Courses** entitles will also have many-to-one relationship. One Course will have more than one Student.
- **Many-to-Many :** Many students can study more than one course and Course also have many students. Then, it is many-to-many relationship. In the e-r model, the many-to-many relationship is resolved using an additional entity to make it a one-to-many relationship. In the above example, we can have student_course table with student_id and Course_id (composite key) as a Primary key, which resolves the many-to-many relationship.

- **Recursive Relationship :** When an Entity is related with itself it is known as Recursive Relationship. For example, in an employee table, all the employees are stored. It can also have manager_id as an attribute to store under whom the employee is working, where the manager_id itself is an employee_id which are stored in the same table.

Relational Integrity Constraints

Every relation in a relational database model should abide by or follow a few constraints to be a valid relation, these constraints are called as **Relational Integrity Constraints**.

The three main Integrity Constraints are:

- Key Constraints
- Domain Constraints
- Referential integrity Constraints

Key Constraints: The Key Constraint specifies that there should be such an attribute(column) in a relation(table), which can be used to fetch data for any tuple(row).

- The Key attribute should never be **NULL** or same for two different row of data.
- For example, in the Employee table we can use the attribute ID to fetch data for each of the employee. No value of ID is null and it is unique for every row, hence it can be our Key attribute.

Relational Integrity Constraints

Domain Constraint : Domain constraints refers to the rules defined for the values that can be stored for a certain attribute.

As an example, the **Address** of employee should not be stored in the column for **Name.** Similarly, a mobile number cannot exceed 10 digits.

Referential Integrity Constraint : If a table reference to some data from another table, then that table and that data should be present for referential integrity constraint to hold true.

For example, Employee table references a Department Id from Department Table, then the Department Id should be present Department Table. Suppose, if department id deleted from Department table, the referential integrity constraint prevent the deletion of any employee id assigned to the deleted department id.

Normalization

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy (repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies.

Normalization is used for mainly two purposes,

- Eliminating redundant (useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

Normalization

Normalization guidelines are divided into following forms :

First Normal Form(1NF) :

- Each Column to be single valued.
- Entries in a column are same type.
- Unique name for Attributes/Columns.
- The order in which data is stored, does not matter.

roll_no	name	subject
101	Akon	OS, CN
103	Ckon	Java
102	Bkon	C, C++

Normalization

Second Normal Form(2NF) :

- The Table should be in the First Normal Form.
- All attributes should be dependent on the key.

subject_id	subject_name
1	Java
2	C++
3	Php

score_id	student_id	subject_id	marks	teacher
1	10	1	70	Java Teacher
2	10	2	75	C++ Teacher
3	11	1	80	Java Teacher

Subject_id	Subject_name	Teacher

Normalization

Third Normal Form(3NF) :

- The Table should be in the Second Normal Form.
- All field/columns should be identified only by the key.

Scores Table

Score Id	Student_id	Subject_id	Marks	Exam_name	Total_Marks

Exam_id	Exam_name	Total_Marks

Score Id	Student_id	Subject_id	Marks	Exam_id

Normalization

Fourth Normal Form(4NF) :

- No multi valued dependencies.

What is Multi-valued Dependency?

A table is said to have multi-valued dependency, if the following conditions are true,

- For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
- Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
- And, for a relation R(A,B,C), if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

Normalization

Below we have a college enrolment table with columns s_id, course and hobby.

Student_id	Course	Hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	MySQL	Hockey

It will introduce 4 more additional rows since the student Ids 1 and 2 have 2 hobbies each. To map with Subject, we need to add 4 more rows. This multi dependency can be resolved by splitting the data into 2 tables namely, **CourseOpted** Table and **StudentHobby** table


thank
you