E-R Models

Three-Level Architecture

The three-level architecture refers to the **three different layers** at which a DBMS can be viewed. The three levels are as follows:

- **Internal level**: The internal level involves the physical storage of data on storage devices.
- **Logical level**: The logical level involves the schema design for database implementation.
- **View level**: The view level involves the applications required to access the database.

The implementation of all three levels is independent of each other.

Data Models

A **data model** describes the schema in which data is stored in a database, and it explains how the different elements of this data are related to each other.
Data models are logical designs that are first created on paper and then implemented physically. Business users use data models to understand a database.

There are different types of data models: **Entity-Relation or E-R** model, **Relational** model, **Network** model, **Hierarchical** model and **Dimensional** model. All of these models represent data in different schemas. In this module, you will learn about E-R models and relational models.

In a relational data model, all data elements are related to each other. The E-R model stores information about real-life objects in **entities** and **attributes**. The entities are related to each other in some way or the other. Hence, you can build a relational model upon an E-R model. Every entity is a **table** and every attribute is a **column** in a relational model. You can implement the relations in an E-R model using the foreign keys in a relational model.

Building an E-R Model

Entities

An **entity** defines the object about which the information is stored in a database. It is important for a business to understand the main components about which information is necessary for the business. An E-R model describes these business components as entities. An E-R model maps these real-world objects into entities such as customer, student, employee, user, product and department.

Attributes

Attributes in an E-R model represent the properties of a real-world object such as name, weight, type, category, size and brand. Attributes in an E-R model store such properties of every entity. They define what information is required regarding a business entity.

Relations

A **relation** refers to how two different entities are related to each other. A degree of relation defines how two entities participate in a relation. The various degrees of relations are as follows:

- **One-to-one relation:** A one-to-one relation defines that each record of entity 'A' is related to, at most, one record of entity 'B'.
- **One-to-many relation:** A one-to-many relation between two entities 'A' and 'B' defines that each record of entity 'A' can be related to many records of entity 'B', but each record of entity B can only be related to one record of entity A.
- **Many-to-many relation:** A many-to-many relation defines that each record of entity 'A' can be related to any number of records in entity 'B', and vice versa.

Now, let's consider the following statements one by one and discuss what they mean:
One team can manage many projects.

- This statement defines that one team can handle either one or many projects. The participation of the entity 'Project' is many because, for each row in the entity 'Team', there can be multiple rows in the entity 'Project'.
- It does not define the participation of the entity 'Team', which will be one if every project can be handled by only one team. This means that for every row of the entity 'Project', there can be only a single row of the entity 'Team'. The participation of the entity 'Team' will be many if every project can be handled by many teams. This means that for every row of the entity 'Project', there can be many rows of the entity 'Team'.
- The participation of the entity 'Project' is known but that of the entity 'Team' is not known.

One team can manage only one project, but one project can be managed by many teams.

- This statement defines that the relation between the entity 'Team' and the entity 'Project' is many-to-one.
- To understand this, consider five teams, Team A, Team B, Team C, Team D and Team

E. Also, consider three projects, Project A, Project B and Project C.

- As multiple teams can work on a single project, Team A, Team B and Team C are working on Project A. This means that Team A, Team B and Team C cannot work on any other project, as one team can work on only one project at a time.
- Suppose Team D is working on Project B and Team E is working on Project C. This indicates that one project may be handled by only one team or by multiple teams.
- A many-to-one relation between the entity 'Team' and the entity 'Project' indicates that many teams can work on a single project, but one team cannot handle more than one project.

Cardinality

Cardinality defines participation of each entity in the relation. The minimum and maximum cardinality of each entity in the relation can be different. **Minimum cardinality** defines the minimum participation of an entity in a relation, and **maximum cardinality** defines the maximum participation of an entity in a relation.

If the minimum cardinality of entity 'A' is 0, then every row of entity 'A' may or may not have a corresponding row in entity 'B'. This means that a team may or may not manage any project.
If the maximum cardinality of entity 'A' is 1, then for any number of rows of entity 'B', there will be a maximum of one corresponding row in entity 'A'.

If the maximum cardinality of both the entities in a relation is 1, then the degree of relation is **one-to-one**.
If the maximum cardinality of one entity is 1, whereas that of the other entity is N, then the degree of relation is **one-to-many**.
If the maximum cardinality of both the entities in a relation is N, then the degree of relation is **many-to-many**.

Relational Models

A relational model stores data in the form of **tables**; these tables are called **relations**. A relational model is based on the fact that tables are related to each other.

Every table in a relational model has the following characteristics:

1. The name of every table must be unique.
2. The name of every attribute of a table must be unique to that table.
3. Every table must have a key attribute. The value of this key attribute cannot be the same for any two rows of that table.

4. The data type for each attribute must be defined.
5. The order of the rows does not matter in relation.
6. The order of the attributes does not matter in relation.
7. Every row or a tuple in a table represents one data record, i.e., information about one particular type of entity.

A relational model is a more detailed implementation of the E-R model. The E-R model is more of a design that is built to understand the business requirements and data that the business wants to store. A relational model is more detailed in how actually the schema will be implemented in a system. An entity becomes a table, an attribute becomes a column and the relations are established using foreign keys.

Database Keys

A **key** is a name given to one column or a combination of columns, which has a unique value for each row in the table. In a relational model, there are many types of keys, such as super keys, candidate keys, primary keys, composite keys and foreign keys.

Super Key

Super keys are combinations of all the possible attributes that can uniquely identify each row of a table. There can be many such combinations of columns available. All of these combinations are the super keys of a table.
In a table containing the attributes <Student ID, Name>, the Student ID can uniquely identify each row. We know that the value of Student ID is different for each row. Suppose you have two rows: <15, Rohit> and <20, Rohit>. Here, Student ID is sufficient to identify each row: 15 for the first row and 20 for the second row. If we consider the combination of Student ID and Name: 15, Rohit and 20, Rohit, we see that we can identify each row uniquely because of the presence of the Student ID column.
The super keys are as follows:

1. <Student ID>
2. <Student ID, Name>

The name attribute alone cannot be used to identify each row. Thus, <name> is not a super key.

Candidate Key

Candidate keys are the super keys that contain only the necessary attributes required to identify each row of a table. In the table mentioned above, only <Student ID> is necessary to identify each row. So, the candidate keys include only <Student ID>. This shows us that if we use different combinations of columns with a candidate key, we get a super key. **Candidate keys are a subset of super keys**.

Primary Key

If a table has many candidate keys, then one of them is chosen by the database designer to uniquely identify each row; this chosen key is known as the **primary key of that table**.

Note: For a particular entity in an entity-relationship diagram (ERD), the primary key is highlighted by placing an asterisk (*) in front of it.

Composite Key

If this primary key contains more than one attribute, then it is a **composite key**.

Note: For a particular entity in an ERD, if more than one column name contains an asterisk (*) in front of it, that means that the entity has a composite key. This composite key will be the combination of all the columns with asterisks (*) in front of them.

Foreign Key

Foreign keys are used to implement relations between tables. One of the tables (of two related tables) contains the foreign key that refers to the primary key of the other table. The values present in the foreign key column of one table must be present in the primary key column of the other table.

Consider a table A with attributes <A1, A2, A3, A4, A5> and a table B with attributes <B1, B2, B3, B4, B5>. To relate these two tables, we need to have a common column in both the tables. Suppose we keep the A1 column from table A in table B; A1 is the foreign key here.

- Table A: <A1, A2, A3, A4, A5>
- Table B: <B1, B2, B3, B4, B5, A5>

The value in A5 in table B must be present in A1 in table A.

Subqueries

Subqueries are a type of complex queries that are used to retrieve complex data. Subqueries consist of two parts: an inner query and an outer query. These subqueries are further divided into two types:

1. Nested

2. Correlated

Nested Subqueries

In **nested subqueries**, first, the inner query is executed completely, and the result of this inner query is then used to operate on the outer query. The steps to create any subquery are as follows:

1. Identify the tables that are required to form the required query.
2. Identify the common thing (attribute) between these tables.
3. Identify what should be the inner query and what should be the outer query.

Correlated Subqueries

Correlated subqueries work in a different manner. These subqueries work on the principle that for each record present in the outer query, each record in the inner query is processed. The outer query is dependent on the inner query. The steps to create any subquery are as follows:

1. Identify the tables that are required to form the required query.
2. Identify the common thing (attribute) between these tables.
3. Identify what should be the inner query and what should be the outer query.

Building a Relational Model

A relational model is created by mapping an E-R model to a relational model and implementing the relations between the entities. A relation between two entities in an E-R model is implemented using foreign keys.
Let's say a bus entity and a customer entity have a relation 'ticket' in the E-R model. The relation is many-to-many. This ticket relation will be a new table that contains the foreign keys for both the tables. Sometimes, a relation is implemented by foreign keys, but a new table is not created. For example, a department has employees. The relation is one-to-many and the department's primary key becomes a foreign key in the employee's table.
Note: A new table is created for a relation when the relation is **many-to-many** as in the case of a bus and customers.

If the relation between two entities is one-to-one:

a. If both entities have mandatory participation, the foreign key column can be put in any table. However, it must be marked as Non-Null, as participation is mandatory. Therefore, every row in the table must have a foreign key value.
b. If both entities have optional participation, the foreign key column can be put in any table. But, it should not be marked as Non-Null because the participation is optional,

which means that not every row in the first table is related to some other row in the second table or not every row must have a value for the foreign key column.

c. If one entity A has mandatory participation and one entity B has optional participation, the foreign key is kept in the entity A (entity having mandatory participation). This ensures that, as the participation is mandatory, every row in table A will have a value for the foreign key. Here, the column will be marked as Non-Null.
d. The foreign key column is made unique in the one-to-one relation, as one row from one table can correspond to only one row in another table. This means that each value in the foreign key column is also unique for each row.

If the relation between two entities is one-to-many:
In this case, it is recommended that the foreign key be placed in the entity on the ‘many’ side.

Consider two tables: team and project. One team can handle many projects but one project can be handled by only one team. This means that the relation between team and project is one-to-many.

Consider the team table:

Team ID	**Team Name**	**Team Category**
101	Team A	Technical
102	Team B	Sales
105	Team C	Marketing

Consider the project table:

Project ID	**Project Name**
150	Project A
200	Project B
250	Project C
300	Project D
350	Project C

We know the foreign key must be placed in the table on the ‘many’ side. This means that the foreign key column must go in the project table. But let’s say we put it in the team table.
Consider that team A manages project A and project C, team B manages project B, and team C manages project D and project E. This causes Project D in the team table to have

multiple values. As Team A manages two projects, there are two values in the foreign key column. It is difficult to retrieve one value when multiple values are present in a field, as the values are not stored in the form of a list but as a single value.
This is not efficient while querying a database. We can solve this issue by separating the values into new rows. This way, the fields do not contain multiple values, but the information gets repeated. This is the reason the foreign key column is not kept in the table on 'one' side of the relation.

If the relation between two entities is many-to-many:
If the relation is many-to-many, keeping the foreign key in any entity will cause multiple values in the fields of the foreign key column. A new entity that contains the primary key of both entities will be created.

ACID Property

A database built using a relational model is a relational database. ACID properties of a transaction in a relational model are as follows:

1. **Atomicity**: This property ensures that the transaction either happens completely or does not happen at all. Every transaction deletes or updates certain data values in a table. It may also add a new row to the table. Therefore, it is necessary that when a transaction happens, either all data values or rows are updated or none of the values are changed.
2. **Consistency:** This property ensures that the data is consistent in every table. Data must be consistent before and after a transaction is made.
3. **Isolation:** If two transactions are happening at the same place, they should happen independent of each other, or one of these transactions must happen first.
4. **Durability:** Every transaction must be durable. This means that if a transaction occurs, then the changes made by this transaction to the database remain even in the event of a system failure.