**CHAPTER 1**

**INTRODUCTION**

**1.1 DATABASE MANAGEMENT SYSTEM**

A database management system (DBMS) refers to the technology for creating and managing databases. Basically DBMS is a software tool to organize (create, retrieve, update and manage) data in a database. The main aim of a DBMS is to supply a way to store up and retrieve database information that is both convenient and efficient.

Database systems are meant to handle large collection of information. Management of data involves both defining structures for storage of information and providing mechanisms that can do the manipulation of those stored information. Moreover, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access.

**1.2 HISTORY**

Following the [technology](https://en.wikipedia.org/wiki/Technology) progress in the areas of [processors](https://en.wikipedia.org/wiki/Processors), [computer memory](https://en.wikipedia.org/wiki/Computer_memory), [computer storage](https://en.wikipedia.org/wiki/Computer_storage), and [computer networks](https://en.wikipedia.org/wiki/Computer_networks), the sizes, capabilities, and performance of databases and their respective DBMSs have grown in orders of magnitude. The development of database technology can be divided into three eras based on data model or structure: [navigational](https://en.wikipedia.org/wiki/Navigational_database), SQL/[relational](https://en.wikipedia.org/wiki/Relational_database), and post-relational.

The two main early navigational data models were the [hierarchical model](https://en.wikipedia.org/wiki/Hierarchical_database_model), epitomized by IBM's IMS system, and the [CODASYL](https://en.wikipedia.org/wiki/CODASYL) model ([network model](https://en.wikipedia.org/wiki/Network_model)), implemented in a number of products such as [IDMS](https://en.wikipedia.org/wiki/IDMS).

The [relational model](https://en.wikipedia.org/wiki/Relational_model), first proposed in 1970 by [Edgar F. Codd](https://en.wikipedia.org/wiki/Edgar_F._Codd), departed from this tradition by insisting that applications should search for data by content, rather than by following links. The relational model employs sets of ledger-style tables, each used for a different type of entity. Only in the mid-1980s did computing hardware become powerful enough to allow the wide deployment of relational systems (DBMSs plus applications). By the early 1990s, however, relational systems dominated in all large-scale [data processing](https://en.wikipedia.org/wiki/Data_processing) applications, and as of 2015 they remain dominant: [IBM DB2](https://en.wikipedia.org/wiki/IBM_DB2), [Oracle](https://en.wikipedia.org/wiki/Oracle_database), [MySQL](https://en.wikipedia.org/wiki/MySQL), and [Microsoft SQL Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server) are the top [DBMS](https://en.wikipedia.org/wiki/DBMS). The dominant database language, standardised SQL for the relational model, has influenced database languages for other data models.

[Object databases](https://en.wikipedia.org/wiki/Object_database) were developed in the 1980s to overcome the inconvenience of [object-relational impedance mismatch](https://en.wikipedia.org/wiki/Object-relational_impedance_mismatch), which led to the coining of the term "post-relational" and also the development of hybrid [object-relational databases](https://en.wikipedia.org/wiki/Object-relational_database).

The next generation of post-relational databases in the late 2000s became known as [NOSQL](https://en.wikipedia.org/wiki/NoSQL) databases, introducing fast [key-value stores](https://en.wikipedia.org/wiki/Key-value_store) and [document-oriented databases](https://en.wikipedia.org/wiki/Document-oriented_database). A competing "next generation" known as [New SQL](https://en.wikipedia.org/wiki/NewSQL) databases attempted new implementations that retained the relational/SQL model while aiming to match the high performance of NoSQL compared to commercially available relational DBMSs.

**1.3 OVERVIEW OF THE PROJECT**

**1.4 OBJECTIVES**

**CHAPTER 2**

**REQUIREMENT SPECIFICATION**

**2.1 HARDWARE REQUIREMENTS**

* The hardware required for the development of the project is:
* PROCESSOR :
* PROCESSOR SPEED :
* RAM :
* HARD DISK :

**2.2 SOFTWARE REQUIREMENTS**

The software required for the development of the project is:

* OPERATING SYSTEM :
* ENVIRONMENT :
* BROWSER :
* LANGUAGE : PHP, HTML, CSS
* BACKEND : XAMPP SERVER 3.1.0

**CHAPTER 3**

**ENTITY** - **RELATIONSHIP DIAGRAM**

**3.1 DESCRIPTION**

The ER Model figure shows the proposed system. It defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases. So, let’s see each entity in details:

**CHAPTER 4**

**SCHEMA DIAGRAM**

**4.1 SEVEN STEPS FOR ER TO SCHEMA CONVERSION**

**Step 1: Mapping of Regular Entity Types.**

For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E. Include only the simple component attributes of a composite attribute. Choose one of the key attributes of E as the primary key for R. If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R. If multiple keys were identified for E during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify secondary (unique) keys of relation R. Knowledge about keys is also kept for indexing purposes and other types of analyses.

**Step 2: Mapping of Weak Entity Types.**

For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R. In addition, include as foreign key attributes of R, the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s); this takes care of mapping the identifying relationship type of W. The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any. If there is a weak entity type E2 whose owner is also a weak entity type E1, then E1 should be mapped before E2 to determine its primary key first.

**Step 3: Mapping of Binary 1:1 Relationship Types.**

For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. There are three possible approaches:

1. The foreign key approach.
2. The merged relationship approach, and
3. The cross-reference or relationship relation approach.

The first approach is the most useful and should be followed unless special conditions exist, as we discuss below.

1. ***Foreign key approach:***

Choose one of the relations—S, say—and include as a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S.

Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type R as attributes of S.

1. ***Merged relation approach:***

An alternative mapping of a 1:1 relationship type is to merge the two entity types and the relationship into a single relation. This is possible when both participations are total, as this would indicate that the two tables will have the exact same number of tuples at all times.

1. ***Cross-reference or relationship relation approach:***

The third option is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types. As we will see, this approach is required for binary M:N relationships. The relation R is called a relationship relation (or sometimes a lookup table), because each tuple in R represents a relationship instance that relates one tuple from S with one tuple from T. The relation R will include the primary key attributes of S and T as foreign keys to S and T. The primary key of R will be one of the two foreign keys, and the other foreign key will be a unique key of R. The drawback is having an extra relation, and requiring an extra join operation when combining related tuples from the tables.

**Step 4: Mapping of Binary 1:N Relationship Types.**

For each regular binary 1:N relationship type R, identify the relation S that represents the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R; we do this because each entity instance on the N-side is related to at most one entity instance on the 1-side of the relationship type. Include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of S.

**Step 5: Mapping of Binary M:N Relationship Types.**

For each binary M:N relationship type R, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S. Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S. Notice that we cannot represent an M:N relationship type by a single foreign key attribute in one of the participating relations (as we did for 1:1 or 1:N relationship types) because of the M:N cardinality ratio; we must create a separate relationship relation S.

**Step 6: Mapping of Multivalued Attributes.**

For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K—as a foreign key in R—of the relation that represents the entity type or relationship type that has A as a multivalued attribute. The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.

**Step 7: Mapping of N-array Relationship Types.**

For each n-array relationship type R, where n > 2, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types. Also include any simple attributes of the n-array relationship type (or simple components of composite attributes) as attributes of S. The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types. However, if the cardinality constraints on any of the entity types E participating in R is 1, then the primary key of S should not include the foreign key attribute that references the relation E' corresponding to E.

**CHAPTER 5**

**TABLE DESCRIPTION**

**User**

**CHAPTER 6**

**RESULTS**

**CONCLUSION**