# CHAPTER 5

# SYSTEM DESIGN

**5.1 ENTITY RELATIONSHIP DIAGRAM:**

An Entity relationship diagram shows the relationships of entity sets stored in databases. An entity in this context is an object, a component of data. An entity set is a collection of similar entities. These entities can have attributes that define its properties.

An entity relationship diagram is a snapshot of data structure. An entity relationship diagram shows entities (tables) in a database and relationships between tables within that database. For a good database design, it is essential to have an entity relationship diagram.

There are three basic elements in entity relationship diagram

* Entities are the things for which we want to store information. An entity is a person, place, thing or event.
* Attributes are the data we want to collect for an entity.
* Relationships describe the relations between the entities.

An entity-relationship diagram (ERD) is a data modeling technique that graphically illustrates an information system’s entities and the relationships between those entities. An entity-relationship diagram is a conceptual and representational model of data used to represent the entity framework infrastructure.

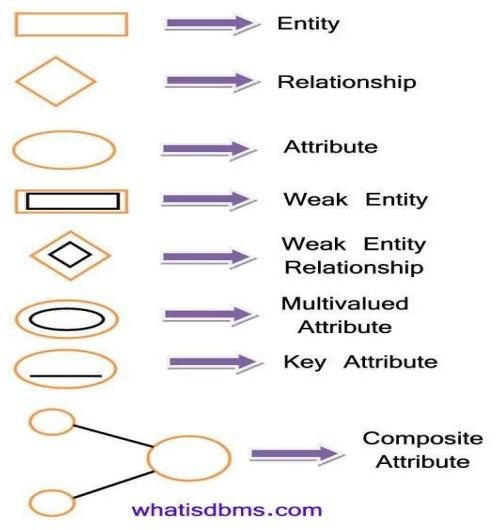
The elements of an entity-relationship diagram are:

* Entities
* Relationships
* Attributes

Steps involved in creating an entity-relationship diagram include:

* Identifying and defining the entities
* Determining all interactions between the entities
* Analyzing the nature of interactions/determining the cardinality of the relationship
* Creating the entity-relationship diagram

### **Notations for ER diagram:**



**Fig.5.1 Notations for ER Diagram**

**Entities and their Attributes:**

* **Customer entity:** Attributes of customer are cus\_id, Username, Fname, Lname Phone\_no, Email, Password,Address
* **Car entity:** Attributes of car are car\_id, car\_name, cost, no\_of\_seats
* **Booking entity:** Attributes of booking are booking\_id, booking\_from , booking\_to , from\_date, to\_date,username
* **Payment entity:** Attributes of payment are pay\_id ,payment\_date, total\_amt
* **Admin entity:** Attributes of admin are username, password

**ER DIAGRAM**

**DOES**

# 1

**CUSTOMER**

# 

# 1

# 

# 

**MANAGES**

# 1 1

**CARS**

**BOOKING**

# 

# M 1

**OWNS**

**COMPLETES**

# 

# 1

# 1

# 

**ADMIN**

PAYMENT

**Fig:5.2 ER Diagram for Vehicle Management system**

**5.2 RELATIONAL DATABASE SCHEMA**

CUSTOMER

Cus \_id email \_id Username F name L name Phone number password

BOOKING

Booking \_id Book \_from Book \_to from \_date to\_date Car \_id Username cus\_id

CARS

Car \_id Car \_name No \_of \_seats cost A\_username

PAYMENT

Pay \_id Total \_amt Payment \_date Booking \_id

ADMINs

Username Password

**Fig:5.3 Relational Database Schema Diagram for Vehicle Management system**

### **5.2.1 Mapping of 1:1 Relationship type:**

For each 1:1 relationship type R can be migrated to any participating entity types. This relationship type ensures that each user in the database can lodge one complaint.

There is 1:1 relationship between vehicle-to-vehicle history and vehicle to booked.

* + 1. **Mapping of 1: N Relationship type:**

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 the foreign key in S the primary key of relation T that represents other entity type in R. We have to do this because each entity instance on N side is related to at most one entity instance on 1-side of relationship type.

There is 1: N relationship from user to vehicle and vehicle to fuel.

### **Mapping of M: N Relationship type:**

For each 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 type, their combination will form the primary key of S. Also include any simple attributes of m: n relationship type.

There is a M: N relationship between driver and vehicle.

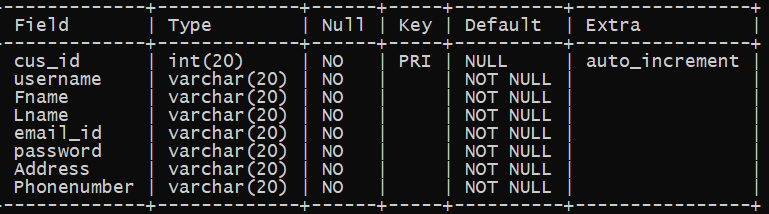
# Database Tables / Relations

A table is a collection of related data held in a structured format within a database. It consists of columns, and rows. In relational databases, and flat file databases, a table is a set of data elements (values) using a model of vertical columns (identifiable by name) and horizontal rows, the cell being the unit where a row and column intersect. A table has a specified number of columns, but can have any number of rows.

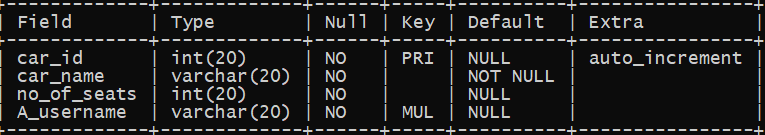
Each row in a table is identified by one or more values appearing in a particular column subset. The columns subset which uniquely identifies a row is called the primary key. A table can be considered as a convenient representation of a relation. A table is perceived a two-dimensional structure composed of rows and columns. The order of rows and columns is immaterial to the DBMS.

**Description of Tables:**

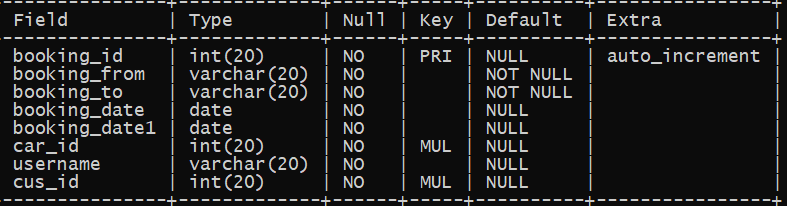
Customer Table

****

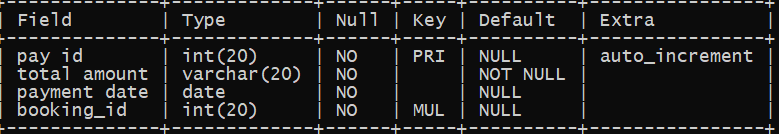
Cars Table



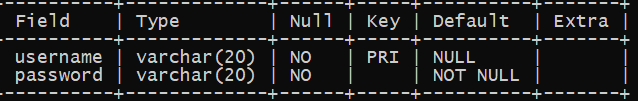
Booking Table



Payment Table



Admin Table



# 5.4 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. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

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

### **1. First Normal Form –**

If a relation contains composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **Name** | **Subject** |
| 101 | Ram | OS, CN |
| 103 | Suresh | Java |
| 102 | Akshaya | C, C++ |

**Example:** We will create a table to store student data which will have student's roll no., their name and the name of subjects they have opted for.

# 

All our column names are unique, we have stored data in the order we wanted to and we have not inter-mixed different type of data in columns.

But out of the 3 different students in our table, 2 have opted for more than 1 subject. And we have stored the subject names in a single column. But as per the 1st Normal form each column must contain atomic value.

All we have to do is break the values into atomic values.

Here is our updated table and it now satisfies the First Normal Form.

|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **Name** | **Subject** |
| 101 | Ram | OS |
| 101 | Ram | CN |
| 103 | Suresh | Java |
| 102 | Akshay | C |
| 102 | Akshay | C++ |

By doing so, although a few values are getting repeated but values for the subject column are now atomic for each record/row.

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

### **2. Second Normal Form –**

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has **No Partial Dependency,**i.e.**,**no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

**Partial Dependency –** If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

# Example: Let’s take an example of a Student table with columns Student\_id, Name, Reg\_no, Branch and Address.

In this table, Student\_id is the primary key and will be unique for every row, hence we can use Student\_id to fetch any row of data from this table.

Even for a case, where student names are same, if we know the Student\_id we can easily fetch the correct record.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student\_id** | **Name** | **Reg\_no** | **Branch** | **Address** |
| 10 | Ram | 07-WY | CSE | Kerala |
| 11 | Ram | 08-WY | IT | Gujarat |

# Let’s create another table for subject, which will have Subject\_id(Primary key), and Subject\_name fields.

|  |  |
| --- | --- |
| **Subject\_id** | **Subject\_name** |
| 1 | Java |
| 2 | C++ |
| 3 | Php |

# Let's create another table Score, to store the marks obtained by students in the respective subjects. We will also be saving name of the teacher who teaches that subject along with marks.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

# Now if you look at the Score table, we have a column name Teacher which is only dependent on the Subject, for Java it's Java Teacher and for C++ it's C++ Teacher & so on. This is Partial Dependency, where an attribute in a table depends on only a part of the primary key and not on the whole key.The simplest solution is to remove columns Teacher from Score table and add it to the Subject table. Hence, the Subject table will become:

|  |  |  |
| --- | --- | --- |
| **Subject\_id** | **Subject\_name** | **Teacher** |
| 1 | Java | Java Teacher |
| 2 | C++ | C++ Teacher |
| 3 | Php | Php Teacher |

# And our Score table is now in the second normal form, with no partial dependency.

|  |  |  |  |
| --- | --- | --- | --- |
| **Score\_id** | **Student\_id** | **Subject\_id** | **Marks** |
| 1 | 10 | 1 | 70 |
| 2 | 10 | 2 | 75 |
| 3 | 11 | 1 | 80 |

### **3. Third Normal Form –**

A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes as well as it is in second normal form.  
A relation is in 3NF if **at least one of the following condition holds** in every non-trivial function dependency X –> Y

* 1. X is a super key.
  2. Y is a prime attribute (each element of Y is part of some candidate key).

# **Transitive dependency –**If A->B and B->C are two FDs then A->C is called transitive dependency.

# Example: We have 3 tables Student, Subject and Score.

# Student Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Student\_id** | **Name** | **Reg\_no** | **Branch** | **Address** |
| 10 | Ram | 07-WY | CSE | Kerala |
| 11 | Ram | 08-WY | IT | Gujarat |
| 12 | Akshay | 09-WY | IT | Rajasthan |

# 

# Subject Table

|  |  |  |
| --- | --- | --- |
| **Subject\_id** | **Subject\_name** | **Teacher** |
| 1 | Java | Java Teacher |
| 2 | C++ | C++ Teacher |
| 3 | Php | Php Teacher |

# Score Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Score\_id** | **Student\_id** | **Subject\_id** | **Marks** |
| 1 | 10 | 1 | 70 |
| 2 | 10 | 2 | 75 |
| 3 | 11 | 1 | 80 |

# In the Score table, we need to store some more information, which is the exam name and total marks, so let's add 2 more columns to the Score table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Score\_id** | **Student\_id** | **Subject\_id** | **Marks** | **Exam\_name** | **Total\_marks** |

# Our new column Exam\_name depends on both student and subject. For example, a mechanical engineering student will have Workshop exam but a computer science student won't. And for some subjects you have Practical exams and for some you don't. So we can say that Exam\_name is dependent on both Student\_id and Subject\_id.

# The column Total\_marks depends on Exam\_name as with exam type the total score changes. For example, practicals are of less marks while theory exams are of more marks.

# Exam\_name is just another column in the score table. It is not a primary key or even a part of the primary key, and Total\_marks depends on it.

This is **Transitive Dependency**. When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key.

# We take out the columns Exam\_name and Total\_marks from Score table and put them in an Exam table and use the Exam\_id wherever required.

#### **Score Table:** In 3rd Normal Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Score\_id** | **Student\_id** | **Subject\_id** | **Marks** | **Exam\_id** |

#### The new **Exam table :**

|  |  |  |
| --- | --- | --- |
| **Exam\_id** | **Exam\_name** | **Total\_marks** |
| 1 | Workshop | 200 |
| 2 | Mains | 70 |
| 3 | Practicals | 30 |

The advantage of removing transitive dependency is,

* Amount of data duplication is reduced.
* Data integrity achieved.