Database:

Database is collection of data which is related by some aspect. Data is collection of facts and figures which can be processed to produce information.

Mostly data represents recordable facts. Data aids in producing information which is based on facts.

A database management system stores data, in such a way which is easier to retrieve, manipulate and helps to produce information.

CHARACTERISTICS:

Traditionally data was organized in file formats. DBMS was all new concepts then and all the research was done to make it to overcome all the deficiencies in traditional style of data management. Modern DBMS has the following characteristics:

- **Real-world entity**: Modern DBMS are more realistic and uses real world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use student as entity and their age as their attribute.
- **Relation-based tables**: DBMS allows entities and relations among them to form as tables. This eases the concept of data saving. A user can understand the architecture of database just by looking at table names etc.
- **Isolation of data and application**: A database system is entirely different than its data. Where database is said to active entity, data is said to be passive one on which the database works and organizes. DBMS also stores metadata which is data about data, to ease its own process.
- **Less redundancy**: DBMS follows rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Following normalization, which itself is a mathematically rich and scientific process, make the entire database to contain as less redundancy as possible.
- **Consistency**: DBMS always enjoy the state on consistency where the previous form of data storing applications like file processing does not guarantee this. Consistency is a state where every relation in database remains consistent. There

exist methods and techniques, which can detect attempt of leaving database in inconsistent state.

- **Query Language**: DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and different filtering options, as he or she wants. Traditionally it was not possible where file-processing system was used.
- **ACID Properties**: DBMS follows the concepts for ACID properties, which stands for Atomicity, Consistency, Isolation and Durability. These concepts are applied on transactions, which manipulate data in database. ACID properties maintains database in healthy state in multi-transactional environment and in case of failure.
- **Multiuser and Concurrent Access**: DBMS support multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when they attempt to handle same data item, but users are always unaware of them.
- **Multiple views**: DBMS offers multiples views for different users. A user who is in sales department will have a different view of database than a person working in production department. This enables user to have a concentrate view of database according to their requirements.
- **Security**: Features like multiple views offers security at some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into database and retrieving data at later stage. DBMS offers many different levels of security features, which enables multiple users to have different view with different features. For example, a user in sales department cannot see data of purchase department is one thing, additionally how much data of sales department he can see, can also be managed. Because DBMS is not saved on disk as traditional file system it is very hard for a thief to break the code.

CODD’S RULE:

Dr Edgar F. Codd did some extensive research in Relational Model of database systems and came up with twelve rules of his own which according to him, a database must obey in order to be a true relational database.

These rules can be applied on a database system that is capable of managing is stored data using only its relational capabilities. This is a foundation rule, which provides a base to imply other rules on it.

Rule 1: Information rule

This rule states that all information (data), which is stored in the database, must be a value of some table cell. Everything in a database must be stored in table formats. This information can be user data or meta-data.

Rule 2: Guaranteed Access rule

This rule states that every single data element (value) is guaranteed to be accessible logically with combination of table-name, primary-key (row value) and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL values

This rule states the NULL values in the database must be given a systematic treatment. As a NULL may have several meanings, i.e. NULL can be interpreted as one the following: data is missing, data is not known, data is not applicable etc.

Rule 4: Active online catalog

This rule states that the structure description of whole database must be stored in an online catalog, i.e. data dictionary, which can be accessed by the authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive data sub-language rule

This rule states that a database must have a support for a language which has linear syntax which is capable of data definition, data manipulation and transaction management operations. Database can be accessed by means of this language only, either directly or by means of some application. If the database can be accessed or manipulated in some way without any help of this language, it is then a violation.

Rule 6: View updating rule

This rule states that all views of database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-level insert, update and delete rule

This rule states the database must employ support high-level insertion, updation and deletion. This must not be limited to a single row that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical data independence

This rule states that the application should not have any concern about how the data is physically stored. Also, any change in its physical structure must not have any impact on application.

Rule 9: Logical data independence

This rule states that the logical data must be independent of its user's view (application). Any change in logical data must not imply any change in the application using it. For example, if two tables are merged or one is split into two different tables, there should be no impact the change on user application. This is one of the most difficult rule to apply.

Rule 10: Integrity independence

This rule states that the database must be independent of the application using it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes database independent of the front-end application and its interface.

Rule 11: Distribution independence

This rule states that the end user must not be able to see that the data is distributed over various locations. User must also see that data is located at one site only. This rule has been proven as a foundation of distributed database systems.

Rule 12: Non-subversion rule

This rule states that if a system has an interface that provides access to low level records, this interface then must not be able to subvert the system and bypass security and integrity constraints.

RELATIONAL ALGEBRA

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either unary or binary. They accept relations as their input and yields relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

Fundamental operations of Relational algebra:

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

I. SELECT OPERATION (σ)

Selects tuples that satisfy the given predicate from a relation.

Notation $\sigma_p(r)$

Where *p* stands for selection predicate and r stands for relation. *p* is prepositional logic formulae which may use connectors like and, or and not. These terms may use relational operators like: =, ≠, ≥, < , >, ≤.

For example:

1. $\sigma_{subject="database"}(\text{Books})$

Output :

Selects tuples from books where subject is 'database'.

2. $\sigma_{\text{subject="database" and price="450"}}(\text{Books})$

Output :

Selects tuples from books where subject is 'database' and 'price' is 450.

3. $\sigma_{\text{subject="database" and price < "450" or year > "2010"}}(\text{Books})$

Output :

Selects tuples from books where subject is 'database' and 'price' is 450 or the publication year is greater than 2010, that is published after 2010.

II. PROJECT OPERATION (∏)

Projects column(s) that satisfy given predicate.

Notation: $\prod_{A1,\ A2,\ An}(r)$

Where a_1, a_2 , a_n are attribute names of relation r.

Duplicate rows are automatically eliminated, as relation is a set.

for example:

$\prod_{subject, author}$ (Books)

Selects and projects columns named as subject and author from relation Books.

III. UNION OPERATION (∪)

Union operation performs binary union between two given relations and is defined as:

$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$

Notion: r U s

Where r and s are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold:

- r, s must have same number of attributes.
- Attribute domains must be compatible.

Duplicate tuples are automatically eliminated.

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

Output : Projects the name of author who has either written a book or an article or both.

IV. SET DIFFERENCE (−)

The result of set difference operation is tuples which present in one relation but are not in the second relation.

Notation: $r - s$

Finds all tuples that are present in r but not s.

$\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

Output: Results the name of authors who has written books but not articles.

V. CARTESIAN `PRODUCT (X)

Combines information of two different relations into one.

Notation: r X s

Where r and s are relations and there output will be defined as:

r X s = { q t | q ∈ r and t ∈ s}

$\prod_{\text{author = 'ramez elmasri'}}$(Books X Articles)

Output :

yields a relation as result which shows all books and articles written by ramez elmasri.

VI. RENAME OPERATION (P)

Results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation.

Rename operation is denoted with small Greek letter rho ρ

Notation: ρ_x (E)

Where the result of expression E is saved with name of x.

Additional operations are:

- Set intersection
- Assignment
- Natural join

Data Model

One fundamental characteristic of the database approach is that it provides some level of data abstraction by hiding details of data storage that are not needed by most database users. A data model is the main tool for providing this abstraction.

A data model is a set of concepts that can be used to describe the structure of a database. It is a collection of high-level data description constructs that hide many low-level storage details.

Categories of Data Models

Many data models have been proposed. We can categorize data models based on the types of concepts they provide to describe the data structure.

High Level or conceptual data models: Provide concepts that are close to the way many users perceive data. Use concepts, such as entities, attributes and relationships, where:

- *Entity* represents a real world object (e.g., student, employee) or concepts (e.g., course, company);
- *Attribute* represents properties that describes objects (e.g., color, name);
- *Relationships* represent an interaction or links among entities (e.g., works-on, is-a, has, etc.).

Low-level or physical data models:
Provide concepts that describe the details of how data is stored in the computer. Concepts provided by low-level data models are generally meant for computer specialists, not for typical end users. They represent information, such as record formats, record orderings and access paths (structure that makes the search for particular database records efficient, i.e. Indexing).

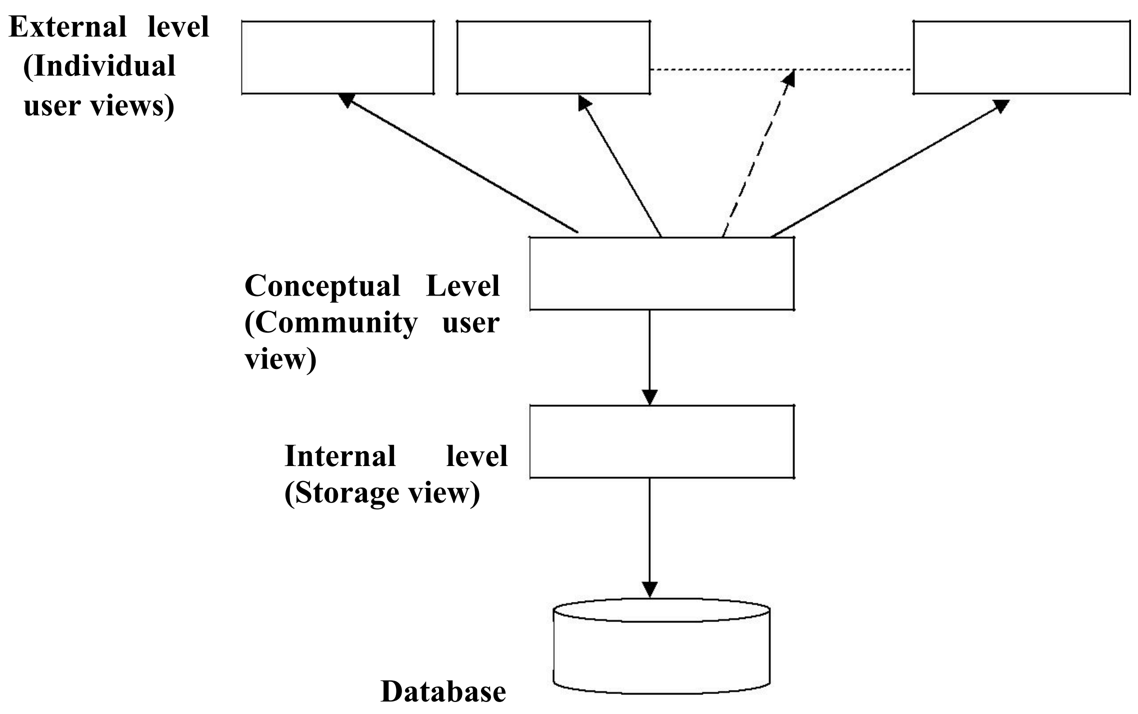
Representational or implementation data models: Between above two extremes is a class of representational (or implementation) data models, which provide concepts that may be understood by end users, but that are not too far removed from the way data is organized within the computer. Representational data models hide some details of data storage, but can be implemented on a computer system in a direct way.

The three important characteristics of the database approach are:

(a) Insulation of programs and data (program-data and program-operation independence).

(b) Support of multiple user views.

(b) Use of a catalog to store database description.


External level (Individual user views)
Conceptual Level (Community user view)
Internal level (Storage view)
Database

<u>**Database Architecture**</u>

Evolution of RDBMS

Before the acceptance of Codd's Relational Model, database management systems was just an ad hoc collection of data designed to solve a particular type of problem, later extended to solve more basic purposes. This led to complex systems, which were difficult to understand, install, maintain and use. These database systems were plagued with the following problems:

- They required large budgets and staffs of people with special skills that were in short supply.
- Database administrators' staff and application developers required prior preparation to access these database systems.
- End-user access to the data was rarely provided.
- These database systems did not support the implementation of business logic as a DBMS responsibility.

Hence, the objective of developing a relational model was to address each and every one of the shortcomings that plagued those systems that existed at the end of the 1960s decade, and make DBMS products more widely appealing to all kinds of users.

The existing relational database management systems offer powerful, yet simple solutions for a wide variety of commercial and scientific application problems. Almost every industry uses relational systems to store, update and retrieve data for operational, transaction, as well as decision support systems.

What is a Relational Database?

A relational database is a database system in which the database is organized and accessed according to the relationships between data items without the need for any consideration of physical orientation and relationship. Relationships between data items are expressed by means of **tables**.

It is a tool, which can help you store, manage and disseminate information of various kinds. It is a collection of objects, tables, queries, forms, reports, and macros, all stored in a computer program all of which are inter-related.

It is a method of structuring data in the form of records, so that relations between different entities and attributes can be used for data access and transformation.

What is a Relational Database Management System?

A Relational Database Management System (RDBMS) is a system, which allows us to perceive data as tables (and nothing but tables), and ***operators*** necessary to manipulate that data are at the user's disposal.

Features of an RDBMS

The features of a relational database are as follows:

- The ability to create multiple relations (tables) and enter data into them
- An interactive query language
- Retrieval of information stored in more than one table
- Provides a *Catalog* or *Dictionary*, which itself consists of tables (called *system* tables)

Basic Relational Database Terminology

Catalog:

A catalog consists of all the information of the various schemas (external, conceptual and internal) and also all of the corresponding mappings (external/conceptual, conceptual/internal).

It contains detailed information regarding the various objects that are of interest to the system itself; e.g., tables, views, indexes, users, integrity rules, security rules, etc.

In a relational database, the entities of the ERD are represented as *tables* and their attributes as the *columns* of their respective tables in a database schema.

It includes some important terms, such as:

- *Table*: Tables are the basic storage structures of a database where data about something in the real world is stored. It is also called a *relation or an entity.*

- *Row:* Rows represent collection of data required for a particular entity. In order to identify each row as unique there should be a *unique identifier* called the *primary key*, which allows no duplicate rows. For example in a library every member is unique and hence is given a membership number, which uniquely identifies each member. A row is also called a *record* or a *tuple.*

- *Column:* Columns represent characteristics or attributes of an entity. Each attribute maps onto a column of a table. Hence, a column is also known as an *attribute*.

- *Relationship:* Relationships represent a logical link between two tables. A relationship is depicted by a *foreign key* column.

- Degree: number of attributes

- Cardinality: number of tuples

- An attribute of an entity has a particular value. The set of possible values That a given attribute can have is called its *domain*.

Keys and Their Use

Key: An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a key for that entity set.

Super Key: If we add additional attributes to a key, the resulting combination would still uniquely identify an instance of the entity set. Such augmented keys are called super keys.

Primary Key: It is a minimum super key.

It is *a unique **identifier** for the table* (a column or a column combination with the property that at any given time no two rows of the table contain the same value in that column or column combination).

Candidate Key: There may be two or more attributes or combinations of attributes that uniquely identify an instance of an entity set. These attributes or combinations of attributes are called candidate keys.

Secondary Key: A secondary key is an attribute or combination of attributes that may not be a candidate key, but that classifies the entity set on a particular characteristic.

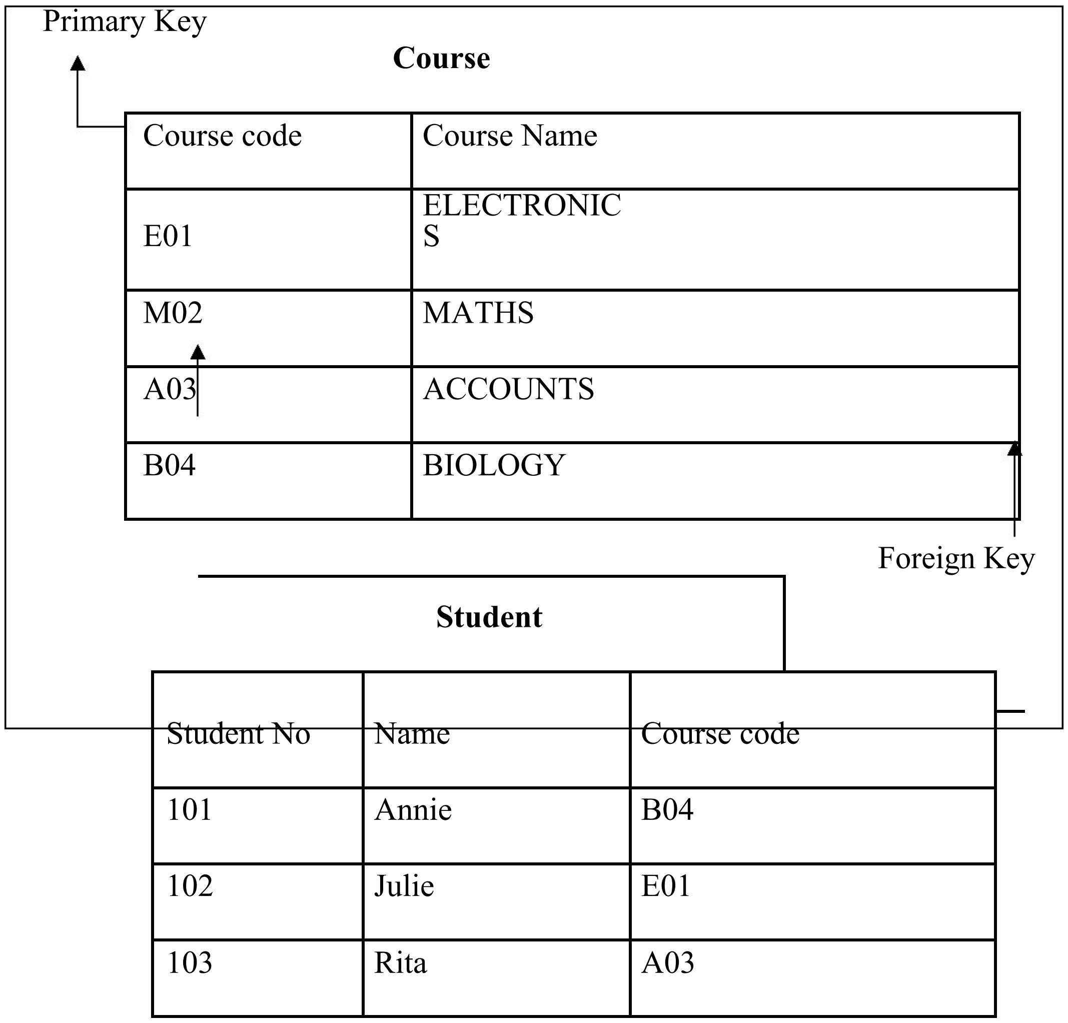
Any key consisting of a single attribute is called a **simple key,** while that consisting of a combination of attributes is called a **composite key**.

Referential Integrity

Referential Integrity can be defined as an integrity constraint that specifies that the value (or existence) of an attribute in one relation depend on the value (or existence) of an attribute in the same or another relation.

Referential integrity in a relational database is consistency between coupled tables. It is usually enforced by the combination of a primary key and a foreign key.

For referential integrity to hold, any field in a table that is declared a foreign key can contain only values from a parent table's primary key field. For instance, deleting a record that contains a value referred to by a foreign key in another table would break referential integrity.



Course

Course code	Course Name
E01	ELECTRONICS
M02	MATHS
A03	ACCOUNTS
B04	BIOLOGY

Student

Student No	Name	Course code
101	Annie	B04
102	Julie	E01
103	Rita	A03

Overview of Database Design

The database design process comprises the following steps:

- Requirement Analysis
- Conceptual Design (ER Model is used at this stage)
- Schema Refinement (Normalization)
- Logical Design
- Physical Database Design and Tuning

- **Requirement Collection & Analysis:** The database designers interview prospective database users to understand and document their data requirements. The result of this step is concisely written set of users requirements.
- This concept of user-defined operations will be applied to the database and they include both retrievals and updates in software design.

- **Conceptual Design:** It is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships and constraints. They are expressed using the concepts provided by the high level data model.

- **Logical Design:** Identification of data model mapping is done here - RDBMS / DBMS / Object Model.

- **Schema Refinement (Normalization):** Check the relational schema for redundancies and related anomalies.

- **Physical Design:** Here, the internal storage structures/ access paths and file organizations for the database files are specified. These activities and application programs are designed and implemented as database transactions corresponding to the high level specifications.

E-R Modeling

The Entity-Relationship model (ER Model in short) is a graphical designing tool for implementation of database systems. It provides a common, informal and convenient model for communication between users and the DBA for the purpose of modeling the structure of data.
The following components are used in developing an E-R Model:

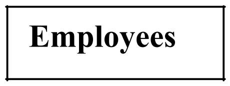
- **Entity**
- **Entity Set**
- **Instance**

- **Attribute**
- **Relationship**
- **Cardinality**
- **Keys**

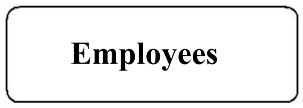
Entity: An *entity* is anything that exists and is distinguishable. For example, each chair is an entity. So is each person and each automobile. Entities can have concrete existence or constitute ideas or concepts. Concepts like love and hate are entities.

Entities can be classified as Regular entities and Weak entities.

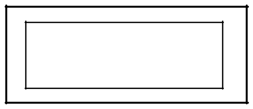
A ***regular (independent) entity*** does not depend on any other entity for its existence. For example, Employee is a regular entity. A regular entity is depicted using a **rectangle.**



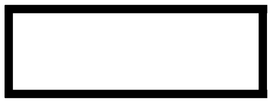
It can also be represented as:



An entity whose existence depends on the existence of another entity is called a ***weak (or dependent) entity***. For example, the dependent of an employee is a weak entity, whose existence depends on the entity Employee. A dependent entity is depicted in a **double-lined box**, or a **darkened rectangle**.



Or



During the design phase, an entity is processed further as Tables.

Entity Set: A group of similar entities forms an entity set.
Examples of entity sets are:

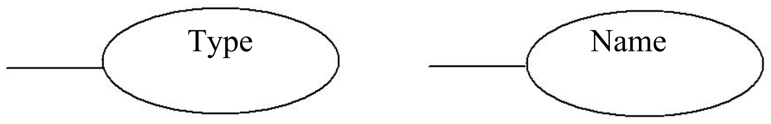
1. All persons
2. All automobiles
3. All emotions

Instance: A specific type of entity is called an instance.
Example: - Smith, Jones, Ally are all employees.

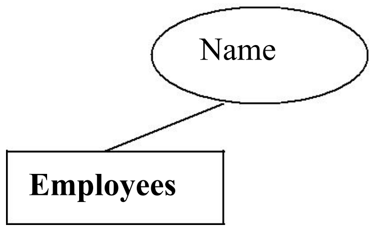
Attributes: Attributes are the properties that characterize an entity set.
For example, employees of an organization are modeled by the entity set EMPLOYEE.

We must include in the model the properties of the employees that may be useful to the organization. Some of these properties are name, address, skill, etc.

An attribute is denoted by an ellipse with its type written inside thereby attached to their respective entity.



An attribute is attached to its entity in the following manner.



During design phase, an attribute is processed further as Column of a table.

Relationship: It is an association between two or more entities or same entity set.

For example, we may have the relationship that an employee ***works in*** a department.

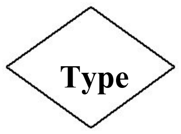
Same entity set could participate in different relationship sets, or in different "roles" in same set.

A relationship is depicted by a **diamond**, with the name of the relationship type. A

relation can be of following types:

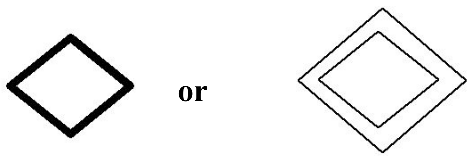
Strong Relationship: A Strong relationship can have Attributes.

It is shown using -



Weak relationship: A weak relationship cannot have any attributes. It is

shown using -



Degree of Relationship:

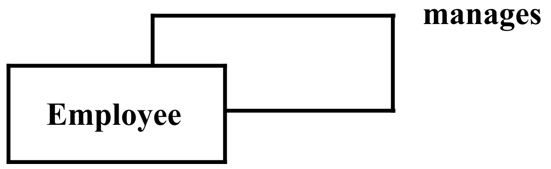
The number of participating entities in a relationship is known as degree of the relationship.

According to degree of relationship, there can be three types of relationships.

- Unary Relationship
- Binary Relationship
- Ternary Relationship
- N-ary Relationship

Unary Relationship:

A relationship where only one entity participates in more than role, is called a *Unary* Relationship.

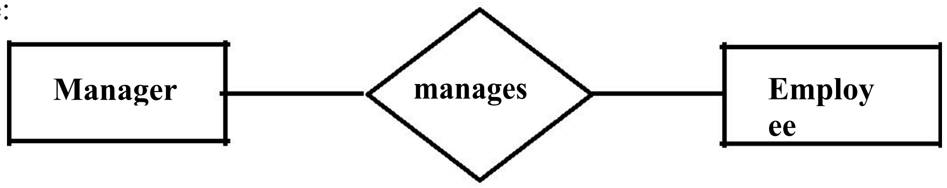


Binary Relationship:

A relationship where there are two entities participating in a relationship, it is called

a *Binary* relationship.

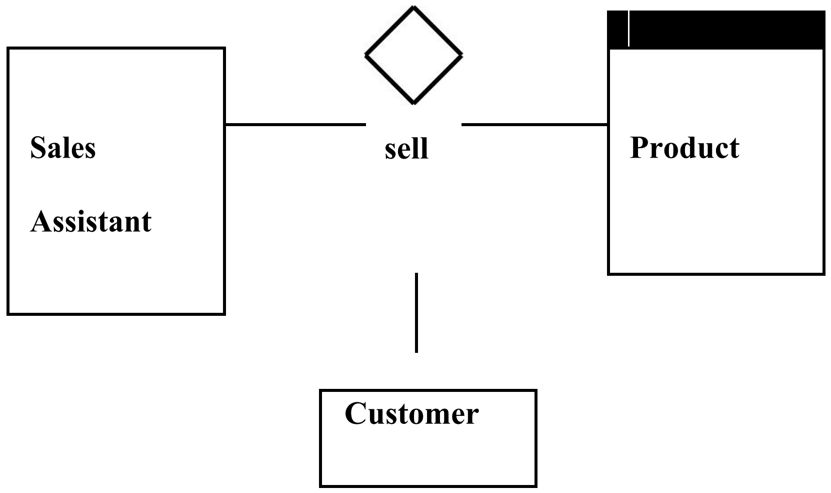
Example:



Ternary relationship:

A relationship where three entity types are involved is called a *ternary* relationship.

Example:



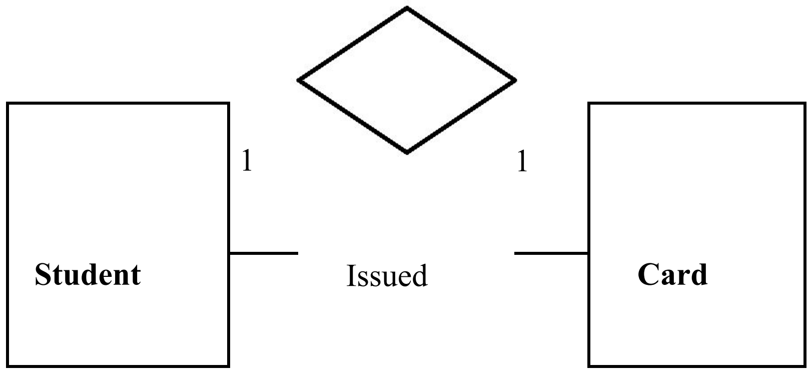
N-ary Relationship:

An n-ary relationship set R relates n entity sets E1...En; each relationship in R involves entities e1 E1, ..., and En.

Cardinality: It defines the numeric relationship between occurrences of entities on either end of the relationship line.

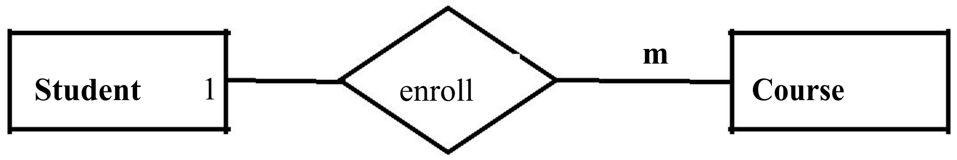
Relationships can be classified into three types based on cardinality:

One-to-one: One student is issued only one card (and vice-versa).

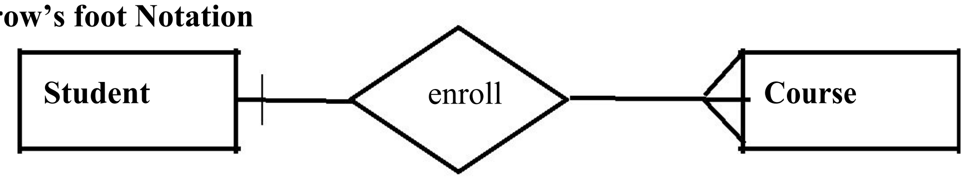


One-to-many (or many-to-one): One student can enroll for only one course, but one course can be offered to many students.

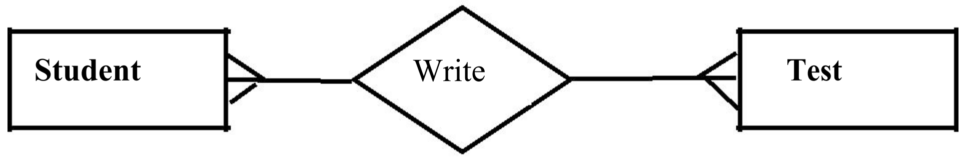
Chen- notation



Crow's foot Notation

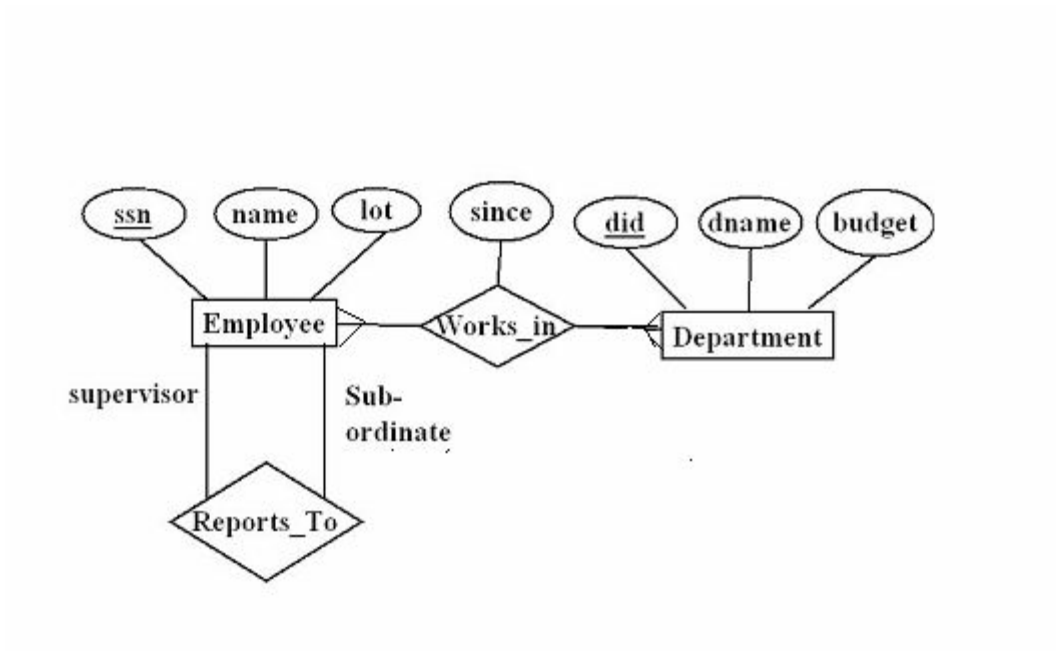


Many-to-many: One student can take many tests, and one test can be taken by many students.



E-R Model Example:

Let us now see how the E-R model is implemented using the above discussed notations. Consider that an employee works in a department and his details stored in the database include his id, name, department name, department id etc.



Normalization

Normalization is a process of designing a consistent Database by minimizing redundancy and ensuring Data Integrity through the principle of Non-loss decomposition.

Why Normalization?

In order to produce good database design, we should ask questions like:

a. Does the design ensure that all database operations will be efficiently performed and that the design does not make the DBMS perform expensive consistency checks, which could be avoided?

b. Is the information unnecessarily replicated?

Database normalization:

Ensures Data Integrity

Now, let us see what is Data Integrity.

Data integrity ensures the correctness of data stored within the database. It is achieved by imposing integrity constraints.
An integrity constraint is a rule, which restricts values present in the database.

There are three integrity constraints:

♦ *Entity constraints:*

The entity integrity rule states that the value of the primary key can never be a null value (a null value is one that has no value and is not the same as a blank). Because a primary key is used to identify a unique row in a relational table, its value must always be specified and should never be unknown. The integrity rule requires that insert, update and delete operations maintain the uniqueness and existence of all primary keys.

♦ *Domain Constraints:*
Only permissible values of an attribute are allowed in a relation.

♦ *Referential Integrity constraints:*
The referential integrity rule states that if a relational table has a foreign key, then every value of the foreign key must either be null or match the values in the relational table in which that foreign key is a primary key.

Prevents Redundancy in data

A non-normalized database is vulnerable to data anomalies, if it stores data redundantly. If data is stored in two locations, but later updated in only one of the locations, then the data is inconsistent; this is referred to as an "update anomaly". A normalized database stores non-primary key data in only one location.

Redundancy can be:

- ♦ *Direct Redundancy:*
  Direct redundancy can result due to the presence of same data in two different locations, thereby, leading to anomalies such as reading, writing, updating and deleting.

- ♦ *Indirect redundancy:*
  Indirect Redundancy results due to storing information that can be computed from the other data items stored within the database.

Normalized databases have a design that reflects the true dependencies between tracked quantities, allowing quick updates to data with little risk of introducing inconsistencies. There are formal methods for quantifying "how normalized" a relational database is, and these classifications are called **Normal Forms** (or **NF**).

What is a Normal Form?

Forms are designed to logically address potential problems such as inconsistencies and redundancy in information stored in the database.

A database is said to be in one of the Normal Forms, if it satisfies the rules required by that Form as well as previous; it also will not suffer from any of the problems addressed by the Form.

Types of Normal Forms

Several normal forms have been identified, the most important and widely used of which are:

- First normal form (1NF)
- Second normal form (2NF)
- Third normal form (3NF)
- Boyce-Codd normal form (BCNF)
- Fourth normal form (4NF)
- Fifth Normal Form (5NF)

A form is said to be in its particular form only if it satisfies the previous Normal form.

First Normal Form (1NF)

A Relation is in 1NF, if every row contains exactly one value for each attribute.

Let us understand this with an example.

Consider a table 'Faculty' which has information about the faculty, subjects and, the number of hours allotted to each subject they teach.

Faculty:

Faculty code	Faculty Name	Date of Birth	Subject	Hours
100	Smith	17/07/64	Java	16
			PL/SQL	8
			Linux	8
101	Jones	24/12/72	Java	16
			Forms	8
			Reports	12

102	Fred	03/02/80	SQL	10
			Linux	8
			Java	16
103	Robert	28/11/66	SQL	10
			PL/SQL	8
			Forms	8

Anomalies: -

The above table does not have any atomic values in the 'Subject' column. Hence, it is called un-normalized table. Inserting, Updating and deletion would be a problem is such table.

Hence it has to be normalized.

For the above table to be in first normal form, each row should have atomic values. Hence let us re-construct the data in the table. A 'S.No' column is included in the table to uniquely identity each row.

NO	**Faculty code**	**Faculty Name**	**Date of Birth**	**Subject**	**Hours**
1	100	Smith	17/07/64	Java	16
2	100	Smith	17/07/64	PL/SQL	8
3	100	Smith	17/07/64	Linux	8
4	101	Jones	24/12/72	Java	16
5	101	Jones	24/12/72	Forms	8
6	101	Jones	24/12/72	Reports	12
7	102	Fred	03/02/80	SQL	10
8	102	Fred	03/02/80	Linux	8
9	102	Fred	03/02/80	Java	16
10	103	Robert	28/11/66	SQL	10
11	103	Robert	28/11/66	PL/SQL	8
12	103	Robert	28/11/66	Forms	8

This table shows the same data as the previous table but we have eliminated the repeating groups.
Hence the table is now said to be in **First Normal form (1NF)**. But we have introduced Redundancy into the table now. This can be eliminated using Second Normal Form (2NF).

Functional Dependencies (FDs)

Functional dependency determines the set of values of the attribute based on another attribute.

It is denoted by

A -> B i.e., B is functionally dependent on A
Or

A determines **B**.

Functional Dependencies can be of two types:

- **Full Functional Dependency**
- **Partial Functional Dependency**

Full Functional Dependency:

A Functional Dependency A -> B is a full functional dependency if removal of any attribute x from A means that the dependency does not hold any more.

{Empno, Project_no} -> HOURS

Full functional dependency:

Empno ->hours and Project_no ->Hours

In the above example, *Hours* is fully functional dependent on both *Empno* and Project_no.

Why? The reason is:

The number of hours spent on the project by a particular employee cannot be determined with the project number (project_no) alone. It needs the employee number (empno) as well.

Partial Dependency:

An FD A -> B is a partial dependency if there is some attribute x Є A (x subset of A) , that can be removed from A and the dependency will still hold.

{Empno, Project_no } -> Ename

Partial dependency:
Empno -> Ename holds.

In the above example, Ename is partially dependent on {Empno, Project_no} Reason being, employee name (ename) can be determined using the employee id (empno) alone even if project_no is removed from the relation.

For a table to be in 2nd Normal form, there should be no partial dependencies.

A relation is in 2NF, if it is in 1NF and every non-key attribute is fully functionally dependent on the primary key of the relation.

2NF prohibits partial dependencies.

The steps for converting a database to 2NF are as follows:

- Find and remove attributes that are related to only a part of the key.
- Group the removed items in another table.
- Assign the new table a key that consists of that part of the old composite key.

If a relation is not in 2NF, it can be further normalized into a number of 2NF relations.

Let us consider the table we obtained after first normalization.

SNO	**Faculty code**	**Faculty Name**	**Date of Birth**	**Subject**	**Hours**
1	100	Smith	17/07/64	Java	16
2	100	Smith	17/07/64	PL/SQL	8
3	100	Smith	17/07/64	Linux	8
4	101	Jones	24/12/72	Java	16
5	101	Jones	24/12/72	Forms	8
6	101	Jones	24/12/72	Reports	12
7	102	Fred	03/02/80	SQL	10
8	102	Fred	03/02/80	Linux	8
9	102	Fred	03/02/80	Java	16
10	103	Robert	28/11/66	SQL	10
11	103	Robert	28/11/66	PL/SQL	8
12	103	Robert	28/11/66	Forms	8

While eliminating the repeating groups, we have introduced redundancy into table. Faculty Code, Name and date of Birth are repeated since the same faculty is multi skilled.
To eliminate this, let us split the table into 2 parts; one with the non-repeating groups and the other for repeating groups.

Faculty:

Faculty code	Faculty Name	Date of Birth
100	Smith	17/07/64
101	Jones	24/12/72
102	Fred	03/02/80
103	Robert	28/11/66

Faculty_code $\longrightarrow$ Faculty_name, Date_of_Birth

The other table is those with repeating groups.

Subject:

SNO	Faculty code	Subject	Hours
1	100	Java	16
2	100	PL/SQL	8
3	100	Linux	8
4	101	Java	16
5	101	Forms	8
6	101	Reports	12
7	102	SQL	10
8	102	Linux	8
9	102	Java	16
10	103	SQL	10
11	103	PL/SQL	8
12	103	Forms	8

Faculty Code is the only key to identify the faculty name and the date of birth.

Hence, Faculty code is the primary key in the first table and foreign key in the second table.

Faculty code is repeated in the Subject table. Hence, we have to take into account the 'SNO' to form a composite key in Subject table. Now, SNO +Faculty code can unique identity each row in this table.

Anomalies in 2nd NF:

The situation could lead to the following problems:

- **Insertion:** Inserting the records of various Faculty teaching same subject would result the redundancy of hours information.
- **Updation:** For a subject, the number of hours allotted to a subject is repeated several times. Hence, if the number of hours has to be changed, this change will have to be recorded in every instance of that subject. Any omissions will lead to inconsistencies.

the subject is lost.

This Subject table should therefore be further decomposed without any loss of information as:

SNO	Faculty code	Subject

Subject	Hours

Transitive Dependency

Transitive dependencies arise:

- When one non-key attribute is functionally dependent on another non-key attribute.
- FD: non-key attribute -> non-key attribute
- And when there is redundancy in database.

Third Normal Form

A relation is in 3NF, if it is in 2NF and no non-key attribute of the relation is transitively dependent on the primary key.

3NF prohibits **transitive dependencies.**

In order to remove the anomalies that arose in Second Normal Form and to remove transitive dependencies, if any, we have to perform third normalization.

Now let us see how to normalize the second table obtained after 2NF.

Subject:

SNO	**Faculty code**	**Subject**	**Hours**
1	100	Java	16
2	100	PL/SQL	8
3	100	Linux	8
4	101	Java	16
5	101	Forms	8
6	101	Reports	12
7	102	SQL	10
8	102	Linux	8
9	102	Java	16
10	103	SQL	10
11	103	PL/SQL	8
12	103	Forms	8

In this table, **hours** depend on the **subject** and **subject** depends on the **Faculty code** and **SNO**. But, hours is neither dependent on the faculty code nor the **SNO**. Hence, there

If a faculty code is deleted, due to transitive dependency, information regarding the **subject** and **hours** allotted to it will be lost.

For a table to be in 3rd Normal form, transitive dependencies must be eliminated.

So, we need to decompose the table further to normalize it.

Fac_Sub:

SNO	**Faculty code**	**Subject**
1	100	Java
2	100	PL/SQL
3	100	Linux
4	101	Java
5	101	Forms
6	101	Reports
7	102	SQL
8	102	Linux
9	102	Java
10	103	SQL
11	103	PL/SQL
12	103	Forms

Sub_Hrs:

Subject	**Hours**
Java	16
PL/SQL	8
Linux	8
Forms	8
Reports	12
SQL	10

After decomposing the ‘Subject’ table we now have ‘Fac_Sub’ and ‘Sub_Hrs’ table respectively. By doing so, the following anomalies are addressed in the table.

Insertion: - No redundancy of data for subject and hours while inserting the records.

Updation: - Subject and hours are stored in the separate table. So updation becomes much easier as there is no repetitiveness of data.

Deletion: - Even if the faculty leaves the organization, the hours allotted to a particular subject can be still retrieved from the Sub_Hrs table.

Boyce–Codd Normal Form (BCNF)

The intention of Boyce-Codd Normal Form (BCNF) is that - 3NF does not satisfactorily handle the case of a relation processing two or more composite or overlapping candidate keys.

A relation R is said to be in BCNF, if and only if every determinant is a candidate key.

In most cases, third normal form is the sufficient level of decomposition. But some case requires the design to be further formalized upto the level of 4th as well as 5th. These are based on the concept of MultiValued Dependency. Let us have a idea about it now.

Multivalued Dependency:

Multivalued dependency defined by $X \twoheadrightarrow$ **Y** is said to hold for a relation R(X,Y,Z) if for a given set of values for X, there is a set of associated values for set of values of attribute Y, and X values depend only on X values and have no dependence on the set of attributes Z.

Fourth Normal Form (4NF)

A relation is said to be in fourth normal form if each table contains no more than one multi-valued dependency per key attribute.

Seminar	Faculty	Topic
DBP-1	Brown	Database Principles
DAT-2	Brown	Database Advanced Techniques
DBP-1	Brown	Data Modeling Techniques
DBP-1	Robert	Database Principles
DBP-1	Robert	Data Modeling Techniques
DAT-2	Maria	Database Advanced Techniques

In the above example, same topic is being taught in a seminar by more than 1 faculty. And Each Faculty takes up different topics in the same seminar. Hence, Topic names are being repeated several times. This is an example of multivalued dependency. For a table to be in fourth Normal Form, multivalued dependency must be avoided.

To eliminate multivalued dependency, split the table such that there is no multivalued dependency.

Seminar	Faculty
DBP-1	Brown
DAT-2	Brown
DBP-1	Robert
DAT-2	Maria

Seminar	Topic
DBP-1	Database Principles
DAT-2	Database Advanced Techniques
DBP-1	Data Modeling Techniques

Fifth Normal Form

A relation is said to be in 5NF if and only if it is in 4NF and every join dependency in it is implied by the candidate keys.

Fifth normal form deals with cases where information can be reconstructed from smaller pieces of information that can be maintained with less redundancy. It emphasizes on lossless decomposition.

Consider the following example:

Faculty	Seminar	Location
Brown	DBP-1	New York
Brown	DAT-2	Chicago
Robert	DBP-1	Chicago

If we were to add the seminar DAT-2 to New York, we would have to add a line to the table for each instructor located in New York.

The table would look like as shown below adding the above information:

Faculty	Seminar	Location
Brown	DBP-1	New York
Brown	DAT-2	Chicago
Robert	DBP-1	Chicago
Brown	DAT-2	New York
Robert	DAT-2	New York

From the above table, we observe that there is a redundancy of data stored for Brown's information. So to eliminate this redundancy, we have to do a 'Non-Loss decomposition' of the table.

Consider the following decomposition of the above table into fifth normal form:

Faculty	Seminar
Brown	DBP-1
Brown	DAT-2
Robert	DBP-1
Robert	DAT-2

Seminar	Location
DBP-1	New York
DAT-2	Chicago
DBP-1	Chicago
DAT-2	New York

Faculty	Location
Brown	New York
Brown	Chicago
Robert	Chicago
Robert	New York

Generally, table is in fifth normal form when its information content cannot be reconstructed from several smaller tables, i.e., from tables having fewer fields than the original table, each table having different keys.
In the normalized form, the fact that 'Brown' traveling to 'New York' is recorded only once, whereas, in the unnormalized form it may be repeated many times.

An attempt has been made to explain Normal forms in a simple yet understandable manner.

Some redundancies are unavoidable. One should take care while normalizing a table so that data integrity is not compromised for removing redundancies.

DOMAIN/KEY NORMAL FORM (DKNF)

DEFINITION:

Domain/key normal form (**DKNF**) is a normal form used in database normalization which requires that the database contains no constraints other than domain constraints and key constraints.

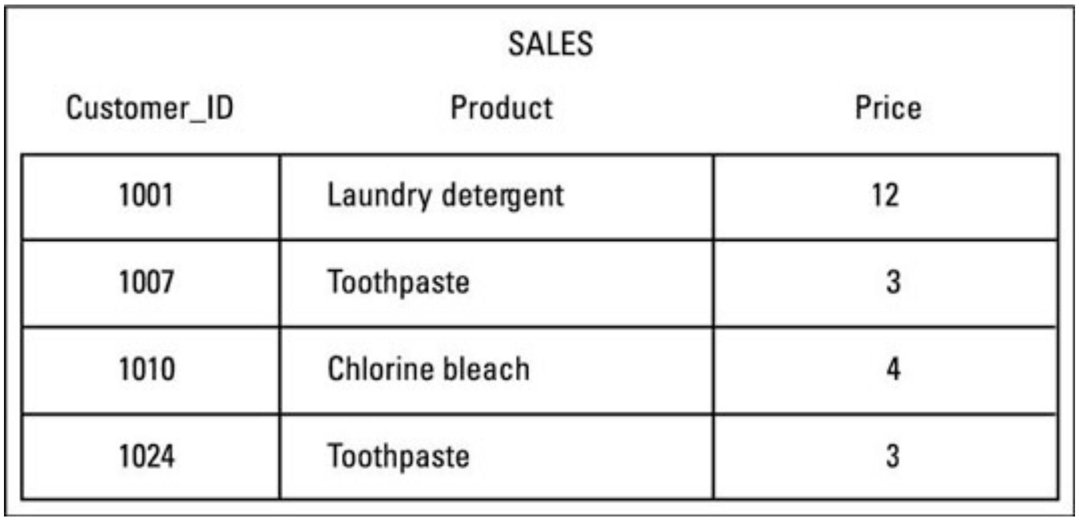
- A *domain* is the set of permitted values of an attribute.
- A domain constraint specifies the permissible values for a given attribute
- A *key* is a unique identifier of a row in a table.

The domain/key normal form is achieved when every constraint on the relation is a logical consequence of the definition of keys and domains, and enforcing key and domain restraints and conditions causes all constraints to be met. Thus, it avoids all non-temporal anomalies.

REASON TO DO:

- The reason to use domain/key normal form is to avoid having general constraints in the database that are not clear domain or key constraints.
- Most databases can easily test domain and key constraints on attributes.
- General constraints however would normally require special database programming in the form of stored procedures that are expensive to maintain and expensive for the database to execute.
- Therefore general constraints are split into domain and key constraints.

Look at this database, which is in 1NF, to see what you must do to put that database in DK/NF.



SALES

Customer_ID	Product	Price
1001	Laundry detergent	12
1007	Toothpaste	3
1010	Chlorine bleach	4
1024	Toothpaste	3

Table: SALES (Customer_ID, Product, Price)

Key: Customer_ID

CONSTRAINTS:

- **Customer_ID** determines Product
- **Product** determines Price
- **Customer_ID** must be an integer > 1000

To enforce Constraint 3 (that Customer_ID must be an integer greater than 1000), you can simply define the domain for Customer_ID to incorporate this constraint. That makes the constraint a logical consequence of the domain of the CustomerID column.

Product depends on Customer_ID, andCustomer_ID is a key, so you have no problem with **Constraint 1**, which is a logical consequence of the definition of the key.

Constraint 2 *is* a problem. Price depends on (is a logical consequence of) Product, and Productisn't a key. The solution is to divide the SALES table into two tables. One table uses Customer_ID as a key, and the other uses Product as a key. The database, besides being in 3NF, is also in DK/NF.

<u>Customer</u>	**Product**
1001	Laundry detergent
1007	Tooth paste
1010	Chlorine bleach
1024	Tooth paste

Product	**Price**
Laundry detergent	12
Tooth paste	3
Chlorine bleach	4
Tooth paste	3

CONCLUSION:

- Design your databases so they're in DK/NF if possible. If you can do that, enforcing key and domain restrictions causes all constraints to be met,
- Modification anomalies aren't possible.
- If a database's structure is designed to prevent you from putting it into DK/NF, then you have to build the constraints into the application program that uses the database. The database itself doesn't guarantee that the constraints will be met.

COMPONENTS OF DBMS

Database Users

Users are differentiated by the way they expect to interact with the system

- Application programmers
- Sophisticated users
- Naïve users
- Database Administrator
- Specialized users etc,.

Application programmers:

Professionals who write application programs and using these application programs they interact with the database system

Sophisticated users :

These user interact with the database system without writing programs, But they submit queries to retrieve the information

Specialized users:

Who write specialized database applications to interact with the database system.

Naïve users:

Interacts with the database system by invoking some application programs that have been written previously by application programmers

Eg : people accessing database over the web

Database Administrator:

Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.

- Schema definition
- Access method definition
- Schema and physical organization modification
- Granting user authority to access the database
- Monitoring performance

Storage Manager

The Storage Manager include these following components/modules

- Authorization Manager
- Transaction Manager
- File Manager
- Buffer Manager

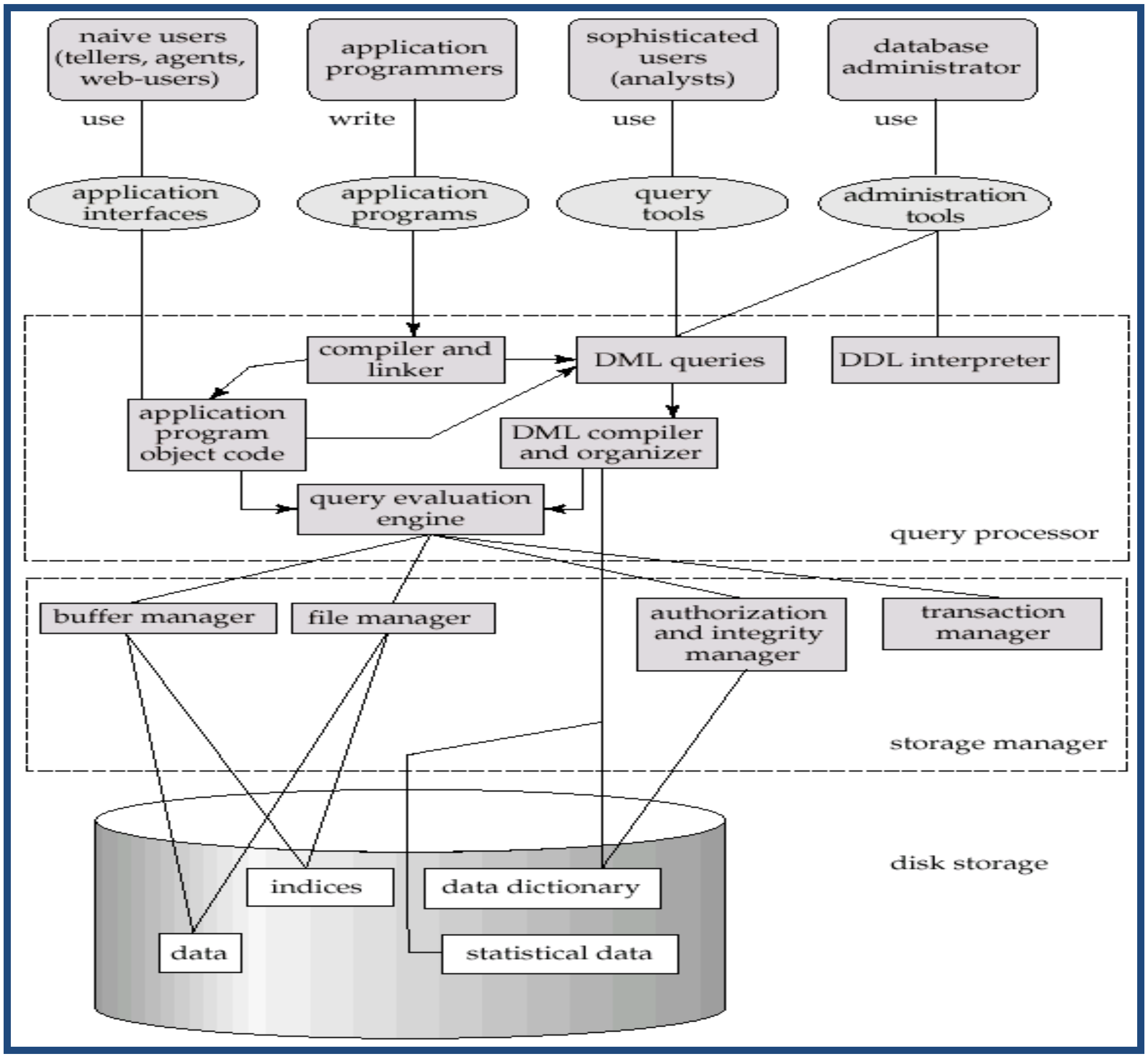
- Storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- **The storage manager is responsible to the following tasks:**
  - interaction with the file manager
  - efficient storing, retrieving and updating of data

Authorization Manager

- Checks whether the user is an authorized person or not
- Test the satisfaction of integrity constraints

Transaction Manager

- Responsible for concurrent transaction execution It ensures that the database remains in a consistent state despite of the system failure



Query Processor

- It is also a collection of components and used to interprets the queries which is submitted by the user.
- The query processor includes these following components
    - ✓ DDL interpreter
    - ✓ DML Compiler
    - ✓ Query evaluation engine
- DDL interpreter
    - ✓ Interprets DDL statements and records the definition in the data dictionary
- DML Compiler
    - ✓ Translate the DML Statements into an evaluation plan that contain low level instructions and the query evaluation engine can understand these instruction
    - ✓ Query can be translated into many alternative evaluation plans
    - ✓ Query optimization –picks up the lowest cost evaluation plan from the alternatives

<u>PURPOSE OF DATABASE SYSTEM</u>

- In the early days, ***File-Processing system*** is used to store records. It uses various files for storing the records.
- Drawbacks of using file systems to store data:
    - ❖ **Data redundancy and inconsistency**
        - ✓ Multiple file formats, duplication of information in different files
    - ❖ **Difficulty in accessing data**
        - ✓ Need to write a new program to carry out each new task
    - ❖ **Data isolation** — multiple files and formats
    - ❖ **Integrity problems**
        - ✓ Hard to add new constraints or change existing ones
    - ❖ **Atomicity problem**
        - ✓ Failures may leave database in an inconsistent state with partial updates carried out
        - ✓ E.g. transfer of funds from one account to another should either complete or not happen at all
    - ❖ **Concurrent access anomalies**
        - ✓ Concurrent accessed needed for performance
    - ❖ **Security problems**
- Database systems offer solutions to all the above problems