renaissance
college of commerce & management

B.com II Year

Subject – RELATIONAL DATABASE MANAGEMENT SYSTEM

SYLLABUS

Unit-I	Introduction to different storage system. Comparative study of manual storage, file storage and dbms. Data, database, DBMS, advantages disadvantages of DBMS, Data development process. Data Models : object based, record based, relationship, network, hierarchical) & physical data models, object oriented models.
Unit – II	E-R model: entity, entity set, relationship & their types, mapping, constraints Extended E-R features: generalization, specialization, aggregation, E- R diagram
Unit – III	Introduction to database language: SQL functions limitations of SQL. Components of SQL (DDL, DML, DCL, TCL with syntax, example) Data types of SQL. Introduction to different operators, set operators, aggregate functions.
Unit – IV	Advanced SQl : review of SQL ,Concept of group by, having order by clause, nested query, join & Its types, Different functions of SQL. Numeric, data, data type conversion, character functions, and miscellaneous functions.
Unit – V	Normalization: Introduction to Normalization, Need of Normalization, Normal form. Normalization using partial dependency, using full dependency, fully functional dependency, multivalue dependency, transitive dependency, join dependency.

Unit-I

Introduction to different storage system:-

There are a number of characteristics to distinguish the database approach from the traditional approaches of programming with files.

In traditional file processing, each user defines and implements the files needed for a specific application as part of programming the application.

In database approach, a single repository of data is maintained that is defined once and then is accessed by various users.

The main characteristics of the database approach versus the file processing approach are as follows:-

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Multiple views of the data
- Sharing of data and multiuser transaction processing

Advantages of database approach versus traditional file approach are as follows:

- Controlled redundancy and reduced inconsistency
- Data independence
- Concurrent access with multiple user views
- Data integrity
- Data security
- Backup and recovery
- Easy accessing of data
- Data isolation
- No atomicity problem

Data:-

The raw facts that can be stored or recorded and that have a clear meaning is called data.

Database:-

A collection of data designed to be used by different people is called a database. It is collection of interrelated data stored together with controlled redundancy to serve one or more applications in an optimal fashion. A database system is basically a computer based record keeping system. The collection of data, usually referred to as the database, contains information about one particular enterprise.

Purpose of Database –

The database system should be repository of the data needed for an organization data processing. The data should be accurate, private & protected from damage. It should be organized so that diverse application with different data requirements can employ the data when needed.

Advantages of DBMS –

1) Database reduced the data redundancy to a large extent.
2) Database can control inconsistency to a large extent.
3) Database facilitate sharing of data.
4) Database enforces standards.

5) Database can ensure data security & privacy.
6) Integrity can be maintained through database.
7) Conflicting requirement can be balance through database.

Characteristics of data – The data stored in database should have this characteristics –

1) Shared
2) Persistence
3) Validity / integrity
4) Security
5) Consistency
6) Non-redundancy
7) Data independence.

Database Development Process

Database Development process can be classically thought of as a systematic process which can be divided into following steps –

1) Interpreting the need of an organization.
2) Identifying the business rules
3) Analyzing the business situations and data
4) Designing the database and defining the functional dependencies.

Database Users –

A primary goal of database system is to provide an environment for retrieving information from and storing new information into the database. There are four different types of database users, differentiated by the way that they expect to interact with the system –

1) Application programmer
2) Sophisticated users
3) Specialized users
4) Naïve Users

Database Administrator –

The person who has central control over the system is called database Administrator. The function of DBA include –

1) Schema definition
2) Storage structure and access method definition
3) Schema and physical organization, modification
4) Granting of Authorization for data access.
5) Integrity – Constraints specifications
6) Routine maintenance

Database Management System –

A Database Management System is a collection of program that enable users to store, create, modify & extract information from a database. The DBMS is hence a general propose software system that facilitate the process of defining, constructing & manipulating database for various applications.

The major activities, operations & services provided by DBMS are as follows –

1) Transaction Management
2) Concurrency Control
3) Recovery Management
4) Security Management
5) Language Interface
6) Storage Management
7) Data Catalog Management.

Applications of DBMS – There are different applications of Database Management System as its competitive era the DBMS is used in following areas.

1) Banking
2) Airlines
3) Organization
4) Universities,
5) Credit Card Transactions
6) Tele Communications
7) Finance
8) Sales
9) Human Resources
10) Manufacturing etc.

Data Models – Data models are different models that can be used to design a database. Design a database include describing data, data relationship, data semantics and consistency constraints. Various data models are as follows –

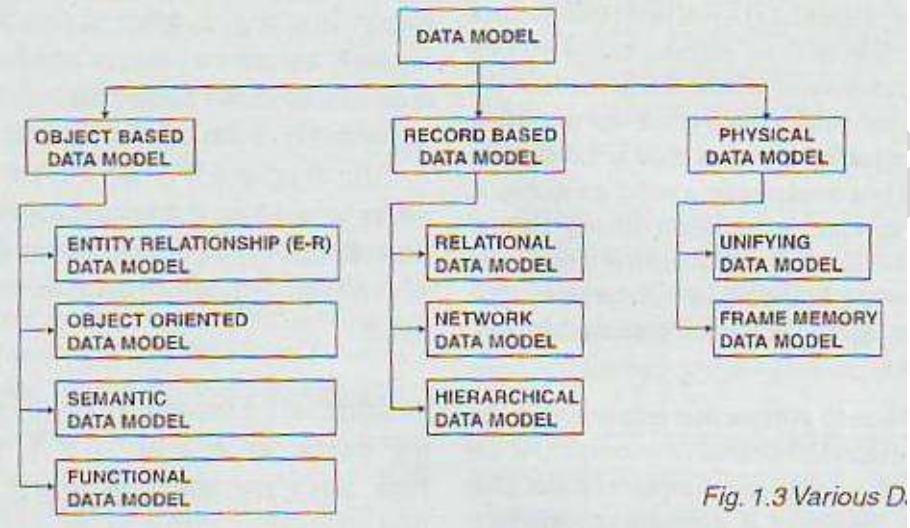
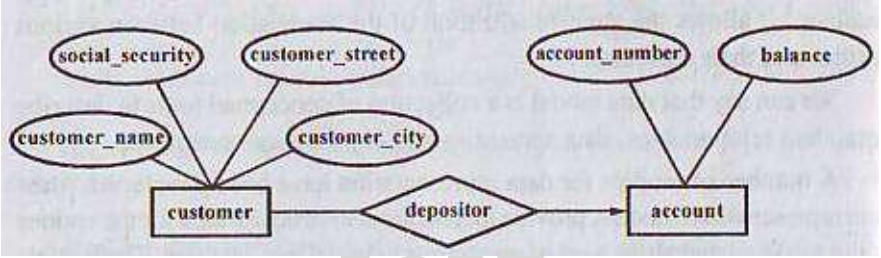


Fig. 1.3 Various D

Object based data Model – Object based logical model are used in describing data at a logical & view levels. They are characterized that they provide fairly flexible structure in capabilities & allow data constraints to be specified explicitly. This model emphasis on the fact that everything is a object having a setoff attributes. There are different data models that utilizes this characteristics –

1) The entity relationship model
2) The object oriented model
3) The semantic data model
4) The functional data model

1) **The entity relationship model** – Entity relationship model moves around three things –
   a) Entity, b) Relationship & c) Attribute.
   ER-Model is based on perception that everything that have physical properties that is entity, every two entities can be distinguish from other. Relationship exists between these entities.



2) **The object oriented model** – Object Oriented Model as name indicates takes everything as object is based on collection of object. Object contains values stored in instance variable within the object. An object also contained bodies of code that operate on the object. These bodies of code are called method.

3) **The semantic data model**

4) **The functional data model**

Limitation of DBMS –

1) High initial investment in Hardware Software & Training
2) Generality that a DBMS provide for defining and processing the data.
3) Overhead for providing security, concurrency control & Integrity function.

Record Based Data Model –

Record based logical model are used in describing data at the logical and view level. In contrast to object based data model they are used both to specify the overall logical structure of the database and to provide a higher level description of implementation. Record based model are so named because the database is structured in fixed format records of several type.

The three most widely used record based data models are –

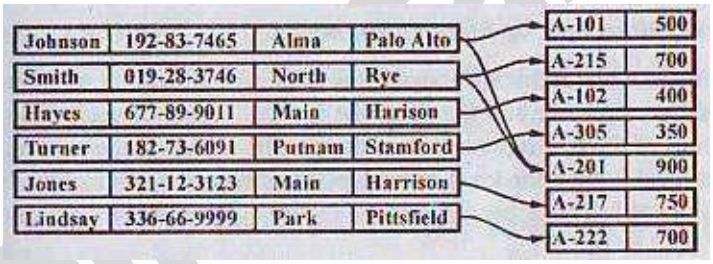
1) Relational Model
2) Network Model
3) Hierarchical Model

1) **Relational Model** – This is most popular among the various record based data model. This model uses a collection of table to represent both data and the relationship among those data.

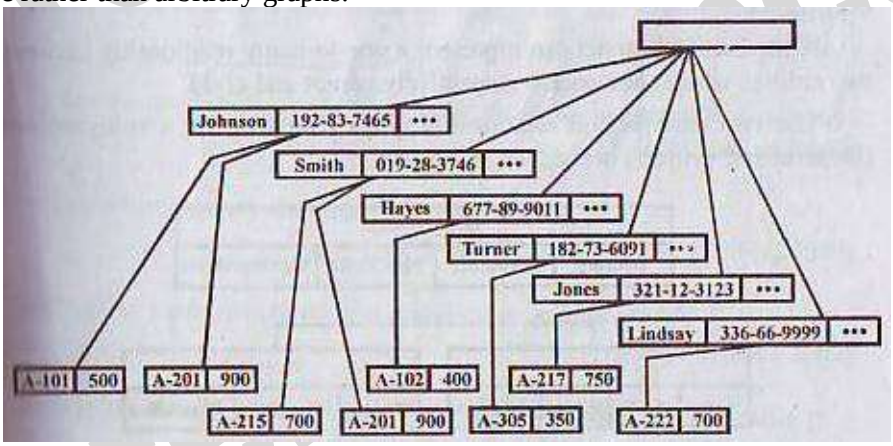
customer_name	social_security	customer_street	customer_city	account_number
Johnson	192-83-7465	Alma	Palo Alto	A-101
Smith	019-28-3746	North	Rye	A-215
Hayes	677-89-9011	Main	Harrison	A-102
Turner	182-73-6091	Putnam	Stamford	A-305
Johnson	192-83-7465	Alma	Palo Alto	A-201
Jones	321-12-3123	Main	Harrison	A-217
Lindsay	336-66-9999	Park	Pittsfield	A-222
Smith	019-28-3746	North	Rye	A-201

account_number	balance
A-101	500
A-215	700
A-102	400
A-305	250
A-201	900
A-217	750
A-222	700

2) **Network Model** – Data in network model are represented by collection of records and relationship among the data are represented by links (Pointer). A Pointer is a physical address which identifies where the next record can be found on the disk.



3) **Hierarchical Model** – It is very similar to network model. In this data model records are organized as collection of tree rather than arbitrary graphs.



Physical Data Model – This model is used to describe data at the lowest level that is to describe to behavior of data at the disk level i.e. the way of data and the data relationship are maintain by storing them on the disk.

This deciding the way the DBMS is going to used secondary storage devices for storing and accessing database.

The widely used data models are –

1) Unifying model
2) Frame memory model

Entity Relationship Data Model
Unit – II

Entity Relationship Data Model was introduced in a key article by Chen (1976) in which he describe the main construct of the ER-Model. – Entities & relationships and their associates attribute. An Entity Relationship Model is a detailed logical representation of the Data for an organization or a business area. An Entity Relationship Model is normally expressed as an Entity Relationship Diagram.

Components of ER-Model:

1) **Entity –** An Entity is a person, place, object, event or concept in the real world i.e. distinguishable from all other objects.
2) **Entity Sets –** An Entity set of Entities of the same type that share the same properties or attributes.
    a. **Strong Entities –** A strong entity set is one that exists independent of other entity sets. A strong entity set that has primary key.
    b. **Weak Entities –** A weak entity is an entity whose existence depends on some other entities. A strong entity set that has no primary key.
3) **Attributes –** An entity can be simply defined as property or characteristics of an entity.
    a. **Simple Attribute –** Simple attributes is an attributes that cannot be broken into smaller subparts.
    b. **Composite Attribute –** Composite Attribute is an attributes that can be broken into smaller subparts.
    c. **Single Valued Attribute –** An attribute is said to be single valued attribute if it can have only one value.
    d. **Multi Value Attribute -** An attribute is said to be single valued attribute if it can have only more than one value.
    e. **Stored Attribute –** An attribute which is already present as an attribute for an entity is a stored attribute.
    f. **Derived Attribute -** An attribute which is derived from stored attribute as it is not present as an attribute for an entity is a derived attribute.
    g. **Null Attribute –** An attribute that can have null value is a null attribute.

Relation (or Table):

The terms Relation & Table can be used interchangeably. Each relation consists of a set of named columns. An attribute is a named column of a relation.

A relation has the following properties:-

1) In any given column of a table, all items are of the same kind whereas items in different columns may not be of the same kind.
2) For a row, each column must have an atomic (indivisible) value and also for a row, a column cannot have more than one value.
3) All rows of a relation are distinct. That is, a relation does not contain two rows which are identical in every column. That is, each row of the relation can be uniquely identified by its contents.
4) The ordering of rows within a relation is immaterial. That is, we cannot retrieve any things by saying that from row number 5, column name is to be accessed. There is no order maintained for rows inside a relation.
5) The columns of a relation are assigned distinct names and the ordering of these columns is immaterial.

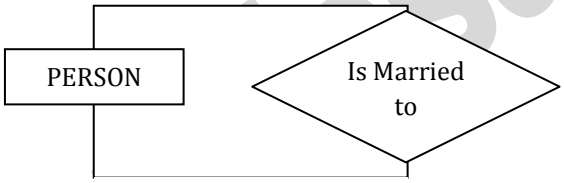
Employee

EmpID	Name	DeptName	Salary
1001	Ravindra Agrawal	Finance	20000
1002	Khelan Nagar	Production	18000
1003	Himanshu Kulkarni	Personnel	25000
1004	Amol Maheshwari	Marketing	30000
1005	Ritesh Singh Chouhan	Advertisement	22000

Relationship Sets:
A relationship is an association among several entities. Relationships are the glue that holds together the various components of an ER Model.
A relationship set is a set of relationships of the same type. For example in a bank, any customer can have any types of loan (Business loan, Personal loan, Home loan) given by the bank. So all the relationship between all the customers and the loan taken by them are together called as relationship set.

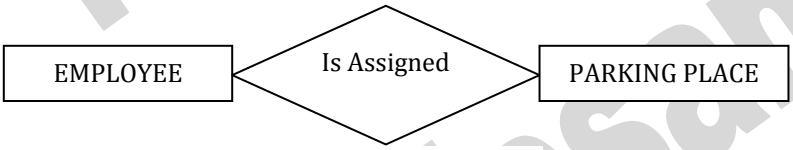
Degree of Relationship:
The degree of a relationship is the number of entity types that participate in that relationship. The three most common relationships in E-R-Model are Unary (degree 1), Binary (degree 2) and Ternary (degree 3).

1) **Unary Relationship**: A unary relationship is a relationship between the instance of a single entity type.

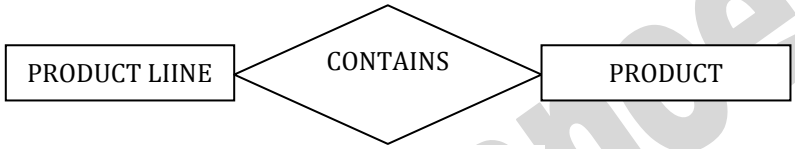


ONE-TO-ONE

2) **Binary Relationship**: A binary relationship is a relationship between the instances of two entity types and is the most common type of relationship encountered in data modeling.



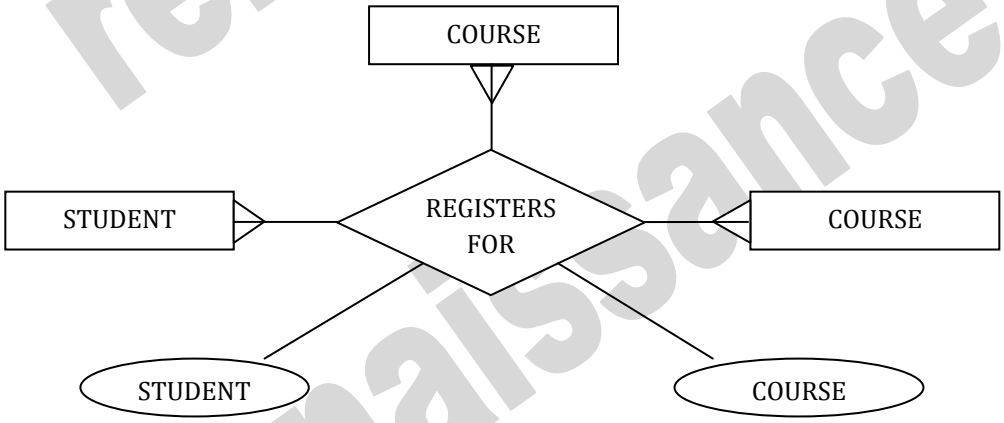
ONE-TO-ONE



ONE-TO-MANY

MANY-TO-MANY

3) **Ternay Relationship**: A ternary relationship is a simultaneous relationship among the instances of three entity types.



Keys:

Keys are attributes or set of attributes used to distinguish one entity from another in an entity set.

1) **Super Key:** A super key is set of one or more attributes that can uniquely identify an entity in an entity set.
2) **Candidate Key:** All the attributes or set of attribute, when can uniquely identify an entity are candidate keys. Only those key can be candidate key whose no proper subset is a superkey.
3) **Primary Key:** The primary key is the term used for the candidates key that is chosen by the database designer as the principal means of identifying an entity.
4) **Alternate Keys:** The alternate key is term used for the candidate keys that are remaining after the primary key has be choosen by database designer.
5) **Foreign Key:** A foreign key is an attribute or set of attribute in a relation of database that serve as the primary key of another relation in the same database.
6) **Composite Key:** A primary key that consists of more than one attribute is called composite key.

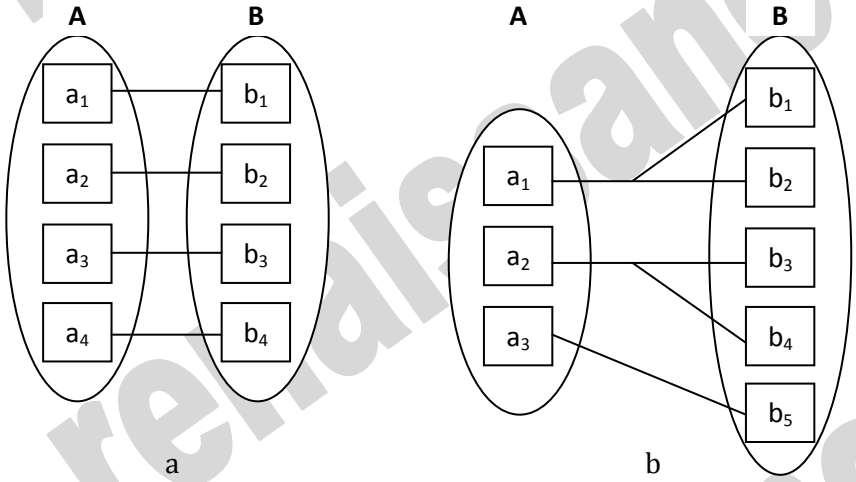
Mapping Constraints:

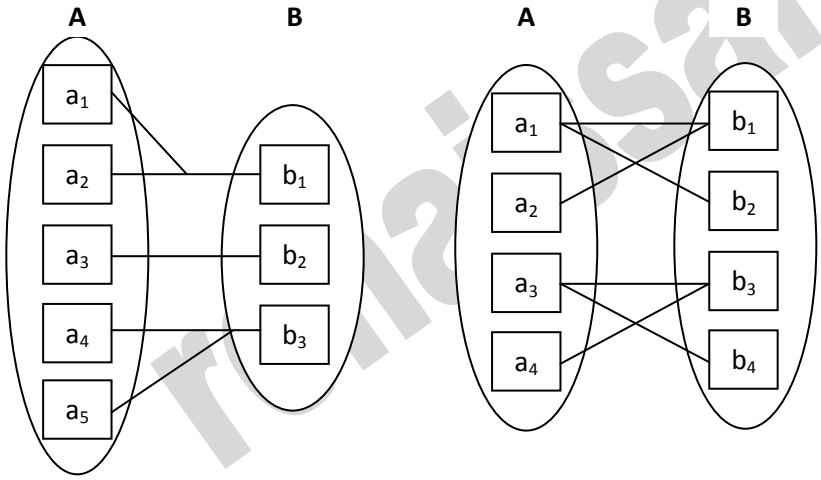
An E-R enterprise schema may define certain constraints to which the contents of a database must conform. This process can be termed as Mapping Constraints.

(1) **Cardinality Constraint**: Cardinality Constraint specifies the number of instances of one entity that can or must be associated with each instance of another entity.

Mapping cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets.

- **One to One:** An entity in A is associated with at most one entity in B and an entity in B is associated with at most one entity in A.
- **One to Many:** An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however can be associated with at most one entity in A.
- **Many to One:** An entity in A is associated with at most one entity in B. An entity in B, however can be associated with any number (zero or more) of entities in A.
- **Many to Many:** An entity in A is associated with any number (zero or more) of entities in B and an entity in B is associated with any number (zero or more) of entities in A.





c d

(2) **Existence Dependencies**: Another important class of constraints is existence dependencies. Specifically, if the existence of entity x depends on the existence of entity y, then x is said to be existence dependent on y. If y is deleted so x has to be deleted. Entity y is said to be **dominant entity** and entity x is said to be **subordinate entity**.

Extended E-R Features:-

Although the basic E-R concepts can model most database features, some aspects of a database may be more aptly expressed by certain extensions to the basic E-R model. The extended E-R features are as follows: -

- specialization,
- generalization,
- higher- and lower-level entity sets,
- attribute inheritance, and
- Aggregation.

Specialization

An entity set may include subgroupings of entities that are distinct in some way from other entities in the set. For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set. The E-R model provides a means for representing these distinctive entity groupings.

Consider an entity set *person*, with attributes *name*, *street*, and *city*. A person may be further classified as one of the following:

· *customer*

· *employee*

Each of these person types is described by a set of attributes that includes all the attributes of entity set *person* plus possibly additional attributes. For example, *customer* entities may be described further by the attribute *customer-id*, whereas *employee* entities may be described further by the attributes *employee-id* and *salary*. The process of designating subgroupings within an entity set is called **specialization**. The specialization of *person* allows us to distinguish among persons according to whether they are employees or customers.

Generalization

The refinement from an initial entity set into successive levels of entity subgroupings represents a top-down design process in which distinctions are made explicit. The design process may also proceed in a bottom-up manner, in which multiple entity sets are synthesized into a higher-level entity set on the basis of common features. The database designer may have first identified a customer entity set with the attributes name, street, city, and customer-id, and an employee entity set with the attributes name, street, city, employee-id, and salary.

There are similarities between the customer entity set and the employee entity set in the sense that they have several attributes in common. This commonality can be expressed by generalization, which is a containment relationship that exists between a higher-level entity set and one or more lower-level

entity sets. In our example, person is the higher-level entity set and customer and employee are lower-level entity sets.

Higher- and lower-level entity sets also may be designated by the terms superclass and subclass, respectively. The person entity set is the superclass of the customer and employee subclasses.

Attribute Inheritance

A crucial property of the higher- and lower-level entities created by specialization and generalization is attribute inheritance. The attributes of the higher-level entity sets are said to be inherited by the lower-level entity sets. For example, customer and employee inherit the attributes of person. Thus, customer is described by its name, street, and city attributes, and additionally a customer-id attribute; employee is described by its name, street, and city attributes, and additionally employee-id and salary attributes. A lower-level entity set (or subclass) also inherits participation in the relationship sets in which its higher-level entity (or superclass) participates. The officer, teller, and secretary entity sets can participate in the works-for relationship set, since the superclass employee participates in the works-for relationship. Attribute inheritance applies

through all tiers of lower-level entity sets. The above entity sets can participate in any relationships in which the person entity set participates.

Whether a given portion of an E-R model was arrived at by specialization or generalization,

the outcome is basically the same:

· A higher-level entity set with attributes and relationships that apply to all of its lower-level entity sets

· Lower-level entity sets with distinctive features that apply only within a particular lower-level entity set

Aggregation

One limitation of the E-R model is that it cannot express relationships among relationships. To illustrate the need for such a construct, consider the ternary relationship works-on, which we saw earlier, between a employee, branch, and job Now, suppose we want to record managers for tasks performed by an employee at a branch; that is, we want to record managers for (employee, branch, job) combinations. Let us assume that there is an entity set manager. One alternative for representing this relationship is to create a quaternary relationship manages between employee, branch, job, and manager. (A quaternary relationship is required—a binary relationship between manager and employee would not permit us to represent which (branch, job) combinations of an employee are managed by which manager.)

It appears that the relationship sets works-on and manages can be combined into one single relationship set. Nevertheless, we should not combine them into a single relationship, since some employee, branch, job combinations many not have amanager. There is redundant information in the resultant figure, however, since every employee, branch, job combination in manages is also in works-on. If the manager were a value rather than an manager entity,we could instead make manager amultivalued attribute of the relationship works-on. But doing so makes it more difficult (logically as well as in execution cost) to find, for example, employee-branch-job triples for which

a manager is responsible. Since the manager is a manager entity, this alternative is ruled out in any case.

The best way to model a situation such as the one just described is to use aggregation. Aggregation is an abstraction through which relationships are treated as higherlevel entities.

E-R Diagram:

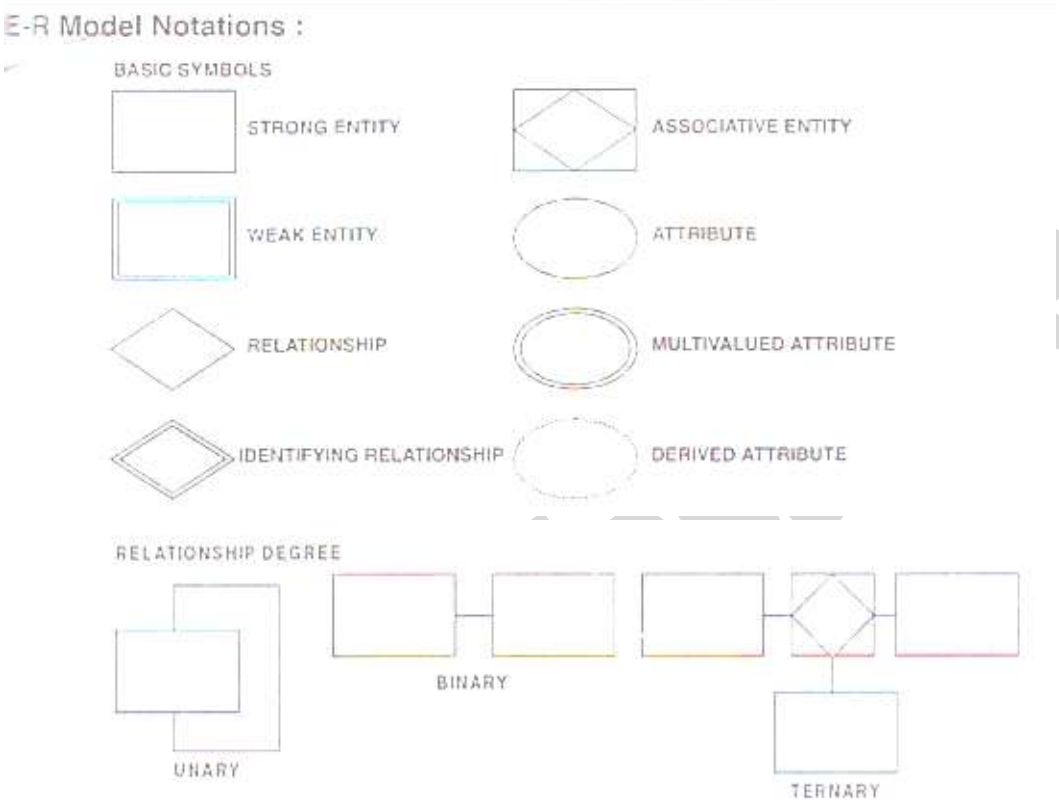
The basic E-R model first introduced during mid 1970s. It has been suitable for modeling most common business problems and has enjoyed widespread use.

The overall logical structure of a database can be expressed graphically by an E-R diagram.

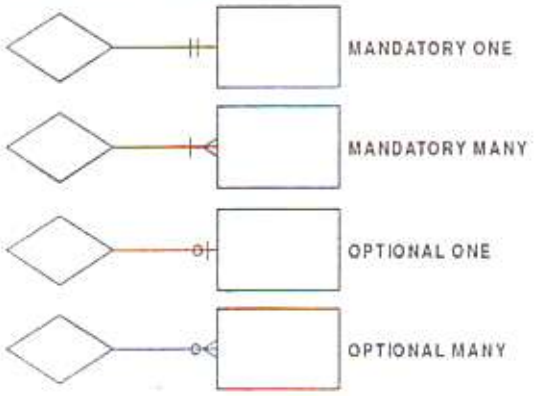
E-R Diagram Conventions:

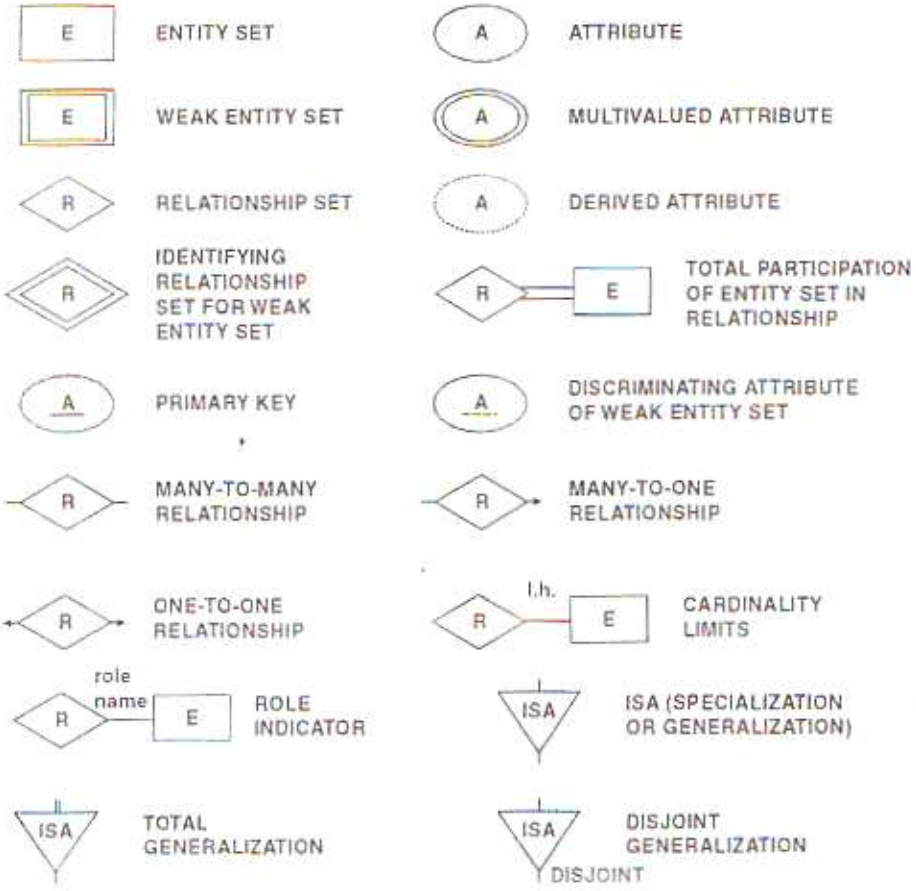
There are conventions for representing the entities and attributes in the E-R diagram.

- The entities are represented by a rectangular box with the name of the entity in the box.
- An attributes is shown as an ellipse attached to a relevant entity by a line and labeled with the attributed name.
- The entity name is written in uppercase where as the attributes name is written in lowercase.
- The primary keys (key attributes) are underlined.
- The attributes are connected using lines to the entities. If the attributes is simple or single valued a single lie is used.
- If the attributes is derived a dotted line is used,
- If it is multi-valued than double lines are used.
- If the attributed is composite, its components attributes are shown as ellipses emanating from the composite attribute.

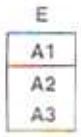


RELATIONSHIP CARDINALITY





ENTITY SET E WITH ATTRIBUTES A1, A2, A3 AND PRIMARY KEY A1



MANY-TO-MANY RELATIONSHIP

ONE-TO-ONE RELATIONSHIP

MANY-TO-ONE RELATIONSHIP

UNIT-III

Introduction to Database Languages

The main objective of a database management system is to allow its users to perform a number of operations on the database such as insert, delete, and retrieve data in abstract terms without knowing about the physical representations of data. To provide the various facilities to different types of users, a DBMS normally provides one or more specialized programming languages called **Database** (or **DBMS**) **Languages**.

There are many popular RDBMS available to work. They are as follows:-

- MySQL
- MS SQL Server
- ORACLE
- MS ACCESS

SQL:-

SQL (Structured Query Language) is a database sublanguage for querying and modifying relational databases. It was developed by IBM Research in the mid 70's and standardized by ANSI in 1986.

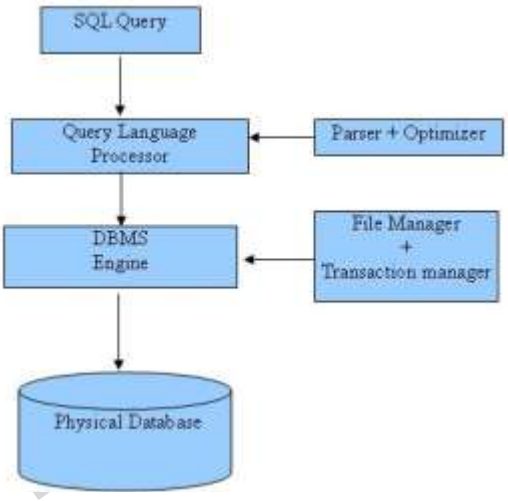
SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database.

SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc.

Characteristics of SQL:-

- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures, and views

SQL Process:



SQL Functions:-
SQL has many built-in functions for performing calculations on data.

SQL Aggregate Functions
SQL aggregate functions return a single value, calculated from values in a column.
The Useful aggregate functions are as follows:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

SQL Scalar functions
SQL scalar functions return a single value, based on the input value.
The Useful scalar functions are as follows:

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

Components of SQL:-
SQL commands are instructions used to communicate with the database to perform specific task that work with data. SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can create tables, add data to tables, or modify data, drop the table, set permissions for users. SQL commands are grouped into four major categories depending on their functionality:

- **Data Definition Language (DDL)** - These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.
- **Data Manipulation Language (DML)** - These SQL commands are used for storing, retrieving, modifying, and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.
- **Transaction Control Language (TCL)** - These SQL commands are used for managing changes affecting the data. These commands are COMMIT, ROLLBACK, and SAVEPOINT.
- **Data Control Language (DCL)** - These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

Some of the Most Important SQL Commands

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database

- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

Some of the Most Important SQL Commands with SQL statement

DML: Data Manipulation Language
SQL-Data Statements -- query and modify tables and columns

- SELECT Statement -- query tables and views in the database
- INSERT Statement -- add rows to tables
- UPDATE Statement -- modify columns in table rows
- DELETE Statement -- remove rows from tables

TCL:- Transaction Control Language
SQL-Transaction Statements -- control transactions

- COMMIT Statement -- commit the current transaction
- ROLLBACK Statement -- roll back the current transaction

DDL:- Data Definition Language
SQL-Schema Statements -- maintain schema (catalog)

- CREATE TABLE Statement -- create tables
- CREATE VIEW Statement -- create views
- DROP TABLE Statement -- drop tables
- DROP VIEW Statement -- drop views
- GRANT Statement -- grant privileges on tables and views to other users
- REVOKE Statement -- revoke privileges on tables and views from other users

The SQL SELECT Statement:-
The SELECT statement is used to select data from a database.
Syntax:
SELECT *column_name,column_name*
FROM *table_name*;
Or
SELECT * FROM *table_name*;

WHERE clause: - It is used to filter records.
SELECT *column_name,column_name*
FROM *table_name*
WHERE *column_name operator value*;

Operators in where clause:-

Operator	Description
=	Equal
<>	Not equal.
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

SQL **AND & OR** Operators:-

- The AND & OR operators are used to filter records based on more than one condition.
- The AND operator displays a record if both the first condition AND the second condition are true.
- The OR operator displays a record if either the first condition OR the second condition is true.

IN

IN operator is used when you know the exact value you want to return for at least one of the columns

SQL ORDER BY Keyword:-

The ORDER BY keyword is used to sort the result-set. The ORDER BY keyword is used to sort the result-set by one or more columns.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword.

Syntax:-

SELECT *column_name,column_name*
FROM *table_name*
ORDER BY *column_name,column_name* ASC|DESC;

SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

The first form does not specify the column names where the data will be inserted, only their values:

INSERT INTO *table_name*
VALUES (*value1*,*value2*,*value3*,...);

The second form specifies both the column names and the values to be inserted:

INSERT INTO *table_name* (*column1*,*column2*,*column3*,...)
VALUES (*value1*,*value2*,*value3*,...);
SQL UPDATE Statement
The UPDATE statement is used to update records in a table.

UPDATE *table_name*
SET *column1=value1*,*column2=value2*,...
WHERE *some_column=some_value*;

SQL DELETE Statement
The DELETE statement is used to delete records in a table.
DELETE FROM *table_name*
WHERE *some_column=some_value*;

SQL CREATE TABLE Statement
The CREATE TABLE statement is used to create a table in a database.
Tables are organized into rows and columns; and each table must have a name.
CREATE TABLE *table_name*
(
column_name1 data_type(*size*),
column_name2 data_type(*size*),
column_name3 data_type(*size*),
....
);
The DROP TABLE Statement
The DROP TABLE statement is used to delete a table.
DROP TABLE table_name;

The ALTER TABLE Statement
The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
To add a column in a table, use the following syntax:
ALTER TABLE table_name
ADD column_name datatype

SQL GRANT Command
SQL GRANT is a command used to provide access or privileges on the database objects to the users.
The Syntax for the GRANT command is:
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];

- *privilege_name* is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.
- *object_name* is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.
- *user_name* is the name of the user to whom an access right is being granted.
- *user_name* is the name of the user to whom an access right is being granted.
- *PUBLIC* is used to grant access rights to all users.
- *ROLES* are a set of privileges grouped together.

- *WITH GRANT OPTION* - allows a user to grant access rights to other users.

SQL REVOKE Command:

The REVOKE command removes user access rights or privileges to the database objects.

The Syntax for the REVOKE command is:

REVOKE privilege_name
ON object_name
FROM {user_name |PUBLIC |role_name}

The COMMIT Command:

The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.

The COMMIT command saves all transactions to the database since the last COMMIT or ROLLBACK command.

Syntax:-

COMMIT;

The ROLLBACK Command:

The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.

The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued.

The syntax for ROLLBACK command is as follows:

ROLLBACK;

SQL Data Types:-

SQL data type is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL.

SQL offers six categories of data types for your use:

- **Exact Numeric Data Types:** int, numeric, bit etc
- **Approximate Numeric Data Types:** Float, real
- **Date and Time Data Types:** Datatime, date, time, smalldatetime
- **Character Strings Data Types:** Char, varchar,varchar(max), text
- **Unicode Character Strings Data Types:** Nchar, nvarchar,ntext
- **Binary Data Types:** Binary,varbinary
- **Misc Data Types**

SQL Operator

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.

Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

Set operators:-

SQL support few of set operators on the SQL tables. They are as follows:-

- ✓ Union
- ✓ Intersect
- ✓ minus

UNIT IV

ORDER BY :-

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

Syntax:-

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

GROUP BY

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

HAVING

The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

```
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
HAVING [ conditions ]
ORDER BY column1, column2
```

Subquery

A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.

A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.
There are a few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery.

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
    (SELECT column_name [, column_name ]
    FROM table1 [, table2 ]
    [WHERE])
```

Join

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.
SQL Join Types:
There are different types of joins available in SQL:

- INNER JOIN: returns rows when there is a match in both tables.
- LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN: returns rows when there is a match in one of the tables.
- SELF JOIN: is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN: returns the Cartesian product of the sets of records from the two or more joined tables.

SQL Functions:-
There are two types of functions in SQL

1) **Single Row Functions:** Single row or Scalar functions return a value for every row that is processed in a query.
2) **Group Functions:** These functions group the rows of data based on the values returned by the query. This is discussed in SQL GROUP Functions. The group functions are used to calculate aggregate values like total or average, which return just one total or one average value after processing a group of rows.

There are four types of single row functions. They are:

- **Numeric Functions:** These are functions that accept numeric input and return numeric values.
- **Character or Text Functions:** These are functions that accept character input and can return both character and number values.
-
- **Date Functions:** These are functions that take values that are of datatype DATE as input and return values of datatype DATE, except for the MONTHS_BETWEEN function, which returns a number.
- **Conversion Functions:** These are functions that help us to convert a value in one form to another form. For Example: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE etc.

Numeric Functions:
Numeric functions are used to perform operations on numbers. They accept numeric values as input and return numeric values as output. Few of the Numeric functions are:
ABS (x)
CEIL (x)
FLOOR (x)
TRUNC (x, y)
ROUND (x, y)

Character or Text Functions:
Character or text functions are used to manipulate text strings. They accept strings or characters as input and can return both character and number values as output.
Few of the character or text functions are as given below:

- LOWER (string_value)
- UPPER (string_value)
- INITCAP (string_value)
- LTRIM (string_value, trim_text)
- RTRIM (string_value, trim_text)
- TRIM (trim_text FROM string_value)
- SUBSTR (string_value, m, n)
- LENGTH (string_value)
- LPAD (string_value, n, pad_value)
- RPAD (string_value, n, pad_value)

Date Functions:
These are functions that take values that are of datatype DATE as input and return values of datatypes DATE, except for the MONTHS_BETWEEN function, which returns a number as output.
Few date functions are as given below.

- ADD_MONTHS (date, n)
- MONTHS_BETWEEN (x1, x2)
- ROUND (x, date_format)
- TRUNC (x, date_format)
- NEXT_DAY (x, week_day)
- LAST_DAY (x)
- SYSDATE
- NEW_TIME (x, zone1, zone2)

Conversion Functions:

These are functions that help us to convert a value in one form to another form. For Ex: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE.

Few of the conversion functions available in SQL are:

- TO_CHAR (x [,y])
- TO_DATE (x [, date_format])
- NVL (x, y)
- DECODE (a, b, c, d, e, default_value)

UNIT V

Functional dependency

In a given table, an attribute *Y* is said to have a functional dependency on a set of attributes *X* (written $X \rightarrow Y$) if and only if each *X* value is associated with precisely one *Y* value.

For example, in an "Employee" table that includes the attributes "Employee ID" and "Employee Date of Birth", the functional dependency {Employee ID} → {Employee Date of Birth} would hold. It follows from the previous two sentences that each {Employee ID} is associated with precisely one {Employee Date of Birth}.

Full functional dependency

An attribute is fully functionally dependent on a set of attributes X if it is:

- functionally dependent on X, and
- not functionally dependent on any proper subset of X. {Employee Address} has a functional dependency on {Employee ID, Skill}, but not a *full* functional dependency, because it is also dependent on {Employee ID}.Even by the removal of {Skill} functional dependency still holds between {Employee Address} and {Employee ID}.

Transitive dependency

A transitive dependency is an indirect functional dependency, one in which $X \rightarrow Z$ only by virtue of $X \rightarrow Y$ and $Y \rightarrow Z$.

Trivial functional dependency

A trivial functional dependency is a functional dependency of an attribute on a superset of itself. {Employee ID, Employee Address} → {Employee Address} is trivial, as is {Employee Address} → {Employee Address}.

Multivalve dependency

A multivalued dependency is a constraint according to which the presence of certain rows in a table implies the presence of certain other rows.

Join dependency

A table *T* is subject to a join dependency if *T* can always be recreated by joining multiple tables each having a subset of the attributes of *T*.

Normal form	Defined by	In	Brief definition	Description
1NF	First normal form	Two versions: E.F. Codd (1970), C.J. Date (2003)	1970and 2003	The domain of each attribute contains only atomic values, and the value of each attribute contains only a single value from that domain.
2NF	Second normal form	E.F. Codd	1971	No non-prime attribute in the table is functionally dependent on a proper subset of any candidate
3NF	Third normal form	Two versions: E.F.	1971and 1982	Every non-prime attribute is non-transitively dependent on every candidate key in the

Normal form	Defined by	In	Brief definition	Description
		Codd (1971), C. Zaniolo (1982)		table. The attributes that do not contribute to the description of the primary key are removed from the table. In other words, no transitive dependency is allowed.
EKNF	Elementary Key Normal Form	C. Zaniolo	1982	Every non-trivial functional dependency in the table is either the dependency of an elementary key attribute or a dependency on a superkey
BCNF	Boyce–Codd normal form	Raymond F. Boyceand E.F. Codd	1974	Every non-trivial functional dependency in the table is a dependency on a superkey
4NF	Fourth normal form	Ronald Fagin	1977	Every non-trivial multivalued dependency in the table is a dependency on a superkey
5NF	Fifth normal form	Ronald Fagin	1979	Every non-trivial join dependency in the table is implied by the superkeys of the table
DKNF	Domain/key normal form	Ronald Fagin	1981	Every constraint on the table is a logical consequence of the table's domain constraints and key constraints
6NF	Sixth normal form	C.J. Date, Hugh Darwen, and Nikos Lorentzos	2002	Table features no non-trivial join dependencies at all (with reference to generalized join operator)

UNF: A table that contains 1/more repeating gps.

1NF: Each cell must have one value & no repeating gps.

2NF: Every non-primary key Attribute is fully functionaly dependent on primary key. i.e. Remove
partially dependency.

3NF: Dependent on primary key. i.e. Remove transitive dependency.

BCNF: Boyce Codd Normal Form: Every determenent is a candidate key.

4NF---->5NF: Higher Normal Form

Modification Anomalies

Once our E-R model has been converted into relations, we may find that some relations are not properly specified. There can be a number of problems:

Deletion Anomaly: Deleting one fact or data point from a relation results in other information being lost.

Insertion Anomaly: Inserting a new fact or tuple into a relation requires we have information from two or more entities – this situation might not be feasible.

Update Anomaly: Updating one fact in a relation requires us to update multiple tuples.

Here is a quick example to illustrate these anomalies: A company has a Purchase Order form:

Normalization Process

- Relations can fall into one or more categories (or classes) called Normal Forms
- Normal Form: A class of relations free from a certain set of modification anomalies.
- Normal forms are given names such as:
  - 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)
  - Domain-Key normal form (DK/NF)
- These forms are cumulative. A relation in Third normal form is also in 2NF and 1NF.
- The Normalization Process for a given relation consists of:
  a. Specify the Key of the relation
  b. Specify the functional dependencies of the relation.
     Sample data (tuples) for the relation can assist with this step.
  c. Apply the definition of each normal form (starting with 1NF).
  d. If a relation fails to meet the definition of a normal form, change the relation (most often by splitting the relation into two new relations) until it meets the definition.
  e. Re-test the modified/new relations to ensure they meet the definitions of each normal form.

In the next set of notes, each of the normal forms will be defined along with an example of the normalization steps.

First Normal Form (1NF)

A relation is in first normal form if it meets the definition of a relation:

- Each attribute (column) value must be a single value only.
- All values for a given attribute (column) must be of the same type.
- Each attribute (column) name must be unique.
- The order of attributes (columns) is insignificant
- No two tuples (rows) in a relation can be identical.
- The order of the tuples (rows) is insignificant.
  - If you have a key defined for the relation, then you can meet the unique row requirement.
  - Example relation in 1NF (note that key attributes are underlined):
  - STOCKS (Company, Symbol, Headquarters, Date, Close_Price)

Company	Symbol	Headquarters	Date	Close Price
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
Microsoft	MSFT	Redmond, WA	09/08/2013	23.93
Microsoft	MSFT	Redmond, WA	09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
Oracle	ORCL	Redwood Shores, CA	09/08/2013	24.14

Oracle	ORCL	Redwood Shores, CA	09/09/2013	24.33

Second Normal Form (2NF)
A relation is in second normal form (2NF) if all of its non-key attributes are dependent on all of the *key*.
Another way to say this: A relation is in second normal form if it is free from partial-key dependencies
Relations that have a single attribute for a key are automatically in 2NF.
This is one reason why we often use artificial identifiers (non-composite keys) as keys.
In the example below, Close Price is dependent on Company, Date
The following example relation *is not* in 2NF:
STOCKS (Company, Symbol, Headquarters, Date, Close_Price)

Company	Symbol	Headquarters	Date	Close Price
Microsoft	MSFT	Redmond, WA	09/07/2013	23.96
Microsoft	MSFT	Redmond, WA	09/08/2013	23.93
Microsoft	MSFT	Redmond, WA	09/09/2013	24.01
Oracle	ORCL	Redwood Shores, CA	09/07/2013	24.27
Oracle	ORCL	Redwood Shores, CA	09/08/2013	24.14
Oracle	ORCL	Redwood Shores, CA	09/09/2013	24.33

Company	Symbol	Headquarters
Microsoft	MSFT	Redmond, WA
Oracle	ORCL	Redwood Shores, CA

FD1: Symbol → Company, Headquarters

STOCK_PRICES relation:

Symbol	Date	Close Price
MSFT	09/07/2013	23.96
MSFT	09/08/2013	23.93
MSFT	09/09/2013	24.01
ORCL	09/07/2013	24.27
ORCL	09/08/2013	24.14
ORCL	09/09/2013	24.33

FD1: Symbol, Date → Close Price

In checking these new relations we can confirm that they meet the definition of 1NF (each one has well defined unique keys) and 2NF (no partial key dependencies).

Third Normal Form (3NF)
A relation is in third normal form (3NF) if it is in second normal form and it contains no *transitive dependencies*.
Consider relation R containing attributes A, B and C. R(A, B, C)
If A → B and B → C then A → C
Transitive Dependency: Three attributes with the above dependencies.
Example: At CUNY:

Course_Code → Course_Number, Section
Course_Number, Section → Classroom, Professor

Consider one of the new relations we created in the STOCKS example for 2nd normal form:

Company	Symbol	Headquarters
Microsoft	MSFT	Redmond, WA
Oracle	ORCL	Redwood Shores, CA

The functional dependencies we can see are:
FD1: Symbol → Company
FD2: Company → Headquarters
so therefore:
Symbol → Headquarters

This is a transitive dependency.
This gives us the following sample data and FD for the new relations

Company	Symbol
Microsoft	MSFT
Oracle	ORCL

FD1: Symbol → Company

Company	Headquarters
Microsoft	Redmond, WA
Oracle	Redwood Shores, CA

FD1: Company → Headquarters

Again, each of these new relations should be checked to ensure they meet the definition of 1NF, 2NF and now 3NF.
Boyce-Codd Normal Form (BCNF)

- A relation is in BCNF if every determinant is a candidate key.
- Recall that not all determinants are keys.
- Those determinants that are keys we initially call *candidate keys*.
- Eventually, we select a single candidate key to be *the key* for the relation.

- Consider the following example:
  - Funds consist of one or more Investment Types.
  - Funds are managed by one or more Managers
  - Investment Types can have one more Managers
  - Managers only manage one type of investment.
- Relation: FUNDS (FundID, InvestmentType, Manager)

FundID	InvestmentType	Manager
99	Common Stock	Smith
99	Municipal Bonds	Jones
33	Common Stock	Green
22	Growth Stocks	Brown
11	Common Stock	Smith

Fourth Normal Form (4NF)
A relation is in fourth normal form if it is in BCNF and it contains no *multivalued dependencies*.
Multivalued Dependency: A type of functional dependency where the determinant can determine more than one value.
More formally, there are 3 criteria:

1. There must be at least 3 attributes in the relation. call them A, B, and C, for example.
2. Given A, one can determine multiple values of B.
3. Given A, one can determine multiple values of C.
4. B and C are independent of one another.

Book example:
Student has one or more majors.
Student participates in one or more activities.

StudentID	Major	Activities
100	CIS	Baseball
100	CIS	Volleyball
100	Accounting	Baseball
100	Accounting	Volleyball
200	Marketing	Swimming

FD1: StudentID →→ Major
FD2: StudentID →→ Activities

Portfolio ID	Stock Fund	Bond Fund
999	Janus Fund	Municipal Bonds
999	Janus Fund	Dreyfus Short-Intermediate Municipal Bond Fund

999	Scudder Global Fund	Municipal Bonds
999	Scudder Global Fund	Dreyfus Short-Intermediate Municipal Bond Fund
888	Kaufmann Fund	T. Rowe Price Emerging Markets Bond Fund

A few characteristics:

- No regular functional dependencies
- All three attributes taken together form the key.
- Latter two attributes are independent of one another.
- Insertion anomaly: Cannot add a stock fund without adding a bond fund (NULL Value). Must always maintain the combinations to preserve the meaning.
  - Stock Fund and Bond Fund form a multivalued dependency on Portfolio ID.
  - PortfolioID →→ Stock Fund
  - PortfolioID →→ Bond Fund

Resolution: Split into two tables with the common key:

Portfolio ID	Stock Fund
999	Janus Fund
999	Scudder Global Fund
888	Kaufmann Fund

Portfolio ID	Bond Fund
999	Municipal Bonds
999	Dreyfus Short-Intermediate Municipal Bond Fund
888	T. Rowe Price Emerging Markets Bond Fund

Fifth Normal Form (5NF)

Also called "Projection Join" Normal form.
There are certain conditions under which after decomposing a relation, it cannot be reassembled back into its original form.
We don't consider these issues here.

Domain Key Normal Form (DK/NF)

A relation is in DK/NF if every *constraint* on the relation is a logical consequence of the definition of *keys*and *domains*.
Constraint: An rule governing static values of an attribute such that we can determine if this constraint is True or False. Examples:

- Functional Dependencies
- Multivalued Dependencies
- Inter-relation rules

- Intra-relation rules

However: Does *Not* include time dependent constraints.

Key: Unique identifier of a tuple.

Domain: The physical (data type, size, NULL values) and semantic (logical) description of what values an attribute can hold.
There is no known algorithm for converting a relation directly into DK/NF.

What is Normalization?
Normalization is the process of efficiently organizing data in a database. There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table). Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

Summary of the Normal Forms
The database community has developed a series of guidelines for ensuring that databases are normalized. These are referred to as normal forms and are numbered from one (the lowest form of normalization, referred to as first normal form or 1NF) through five (fifth normal form or 5NF). In practical applications, you'll often see 1NF, 2NF, and3NF along with the occasional 4NF. Fifth normal form is very rarely seen and won't be discussed in this article.

Before we begin our discussion of the normal forms, it's important to point out that they are guidelines and guidelines only. Occasionally, it becomes necessary to stray from them to meet practical business requirements. However, when variations take place, it's extremely important to evaluate any possible ramifications they could have on your system and account for possible inconsistencies. That said, let's explore the normal forms.

First Normal Form (1NF)
First normal form (1NF) sets the very basic rules for an organized database:
Eliminate duplicative columns from the same table.
Create separate tables for each group of related data and identify each row with a unique column or set of columns (the primary key).

Second Normal Form (2NF)
Second normal form (2NF) further addresses the concept of removing duplicative data:
Meet all the requirements of the first normal form.
Remove subsets of data that apply to multiple rows of a table and place them in separate tables.
Create relationships between these new tables and their predecessors through the use of foreign keys.

Third Normal Form (3NF)
Third normal form (3NF) goes one large step further:
Meet all the requirements of the second normal form.
Remove columns that are not dependent upon the primary key.

Boyce-Codd Normal Form (BCNF or 3.5NF)

The Boyce-Codd Normal Form, also referred to as the "third and half (3.5) normal form", adds one more requirement:

Meet all the requirements of the third normal form.
Every determinant must be a candidate key.

Fourth Normal Form (4NF)

Finally, fourth normal form (4NF) has one additional requirement:

Meet all the requirements of the third normal form.
A relation is in 4NF if it has no multi-valued dependencies.

Remember, these normalization guidelines are cumulative. For a database to be in 2NF, it must first fulfill all the criteria of a 1NF database.