BBA(G)

GGS Indraprastha University

BBA(G)108: Data Base Management System

To develop understanding of database management system and abilities to use DBMS packages.

Course Contents:

Unit I

Introduction to Database Systems: File System versus a DBMS, Advantages of a DBMS, Describing and storing data in a DBMS, Queries in a DBMS, Structure of a DBMS, People who deal with database, introduction to Data Models, Architecture of DBMS.

Unit II

Entity Relationship Model: Overview of Database Design, Entities, attributes, and Entity sets, Relationships and Relationship sets, additional features of the ER Model, Conceptual database design with the ER model – Entity versus attribute, entity versus relationship.

Relational model: Introduction to Relational model, foreign key constraints, enforcing integrity constraints, Querying relational data, logical database design: ER to relation, introduction to views, destroying/altering tables and views, Codd rules

Unit III

Schema Refinement & Normal Forms: Introduction to schema refinement, functional dependencies, examples motivation schema refinement, reasoning about functional dependencies, normal forms, decompositions, normalization (Up to 3rd Normal Form)

Unit IV

Concept of Objects: objects, tables, queries, forms, reports, modules.

Database Creation and Manipulation.

SQL Queries: The form of a basic SQL query, Union, intersect, and expect, introduction to nested queries, aggregate operators, Null values.

Text Books:

1. Ramakrishnan, R. and J. Gehrke; *Database Management Systems,* McGrawHill, Company, Higher Education, 2000.

Reference Books:

1. Elmasri, R. and S B Navathe; *Fundamentals of Database Systems,* Addison Wesley, 2000.

2. Date, C. J.; *An Introduction to Database System,* Vol. I & Vol. II, Addison Wesley Publishing Company, 2000.

UNIT I

A ***database*** is a set of data that has a regular structure and that is organized in such a way that a computer can easily find the desired information.

A ***database management system*** **(DBMS)** is software that has been created to allow the efficient use and management of databases, including ensuring that data is consistent and correct and facilitating it's updating.

For small, single user databases, all functions are often managed by a single program; for larger and multi-user databases, multiple programs are usually involved and a client-server architecture is generally employed. The first DBMSs were developed in the 1960s in an attempt to make more effective use of the new direct access storage devices (i.e., hard disk drives) that were becoming available as supplements and eventual replacements for punched cards and magnetic tape. The word *database* is commonly used in a broad sense to refer not only just too structured data but also to the DBMS that is used with it.

Advantages and disadvantages of DBMS

Advantages

- Reduced data redundancy
- Reduced updating errors and increased consistency
- Greater data integrity and independence from applications programs
- Data can be shared
- Improved data security
- Reduced data entry, storage, and retrieval costs
- Facilitated development of new applications program.

Disadvantages

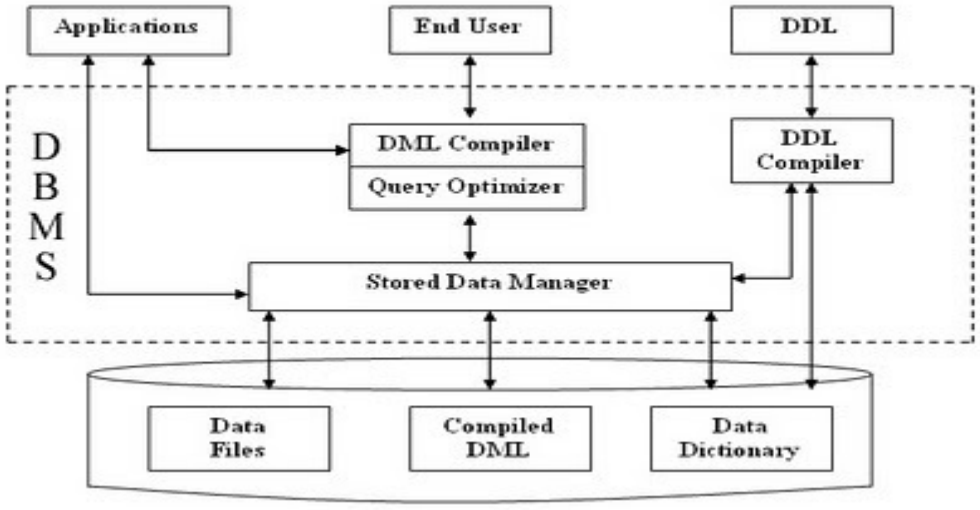
- Database systems are complex, difficult, and time-consuming to design
- Substantial hardware and software start-up costs
- Damage to database affects virtually all applications programs

- Extensive conversion costs in moving form a file-based system to a database system
- Initial training required for all programmers and users
- **Comparison of file management system with database management system**
-

File management system with In C,++ or COBOL	**Database management system with MS Access, SQL & Oracle**
Small system	Large system
Relatively cheap	Relatively expensive
Few files	Many files
Files are in the form of files	Files are in the form of table
Simplex structure	Complex structure
Redundant data	Reduce redundancy
Changes of inconsistency	Consistent data
Isolated data	Data can be shared
Little preliminary design	Vast preliminary design
Integrity left to application programmer	Rigorous inbuilt integrity checking
No security	Rigorous security
Simple, primitive backup	Complex & sophisticated backup
No recovery mode	Fully furnished Recovery mode
Often single user	Multiple user

STRUCTURE OF DBMS

DBMS (Database Management System) acts as an interface between the user and the database. The user requests the DBMS to perform various operations (insert, delete, update and retrieval) on the database. The components of DBMS perform these requested operations on the database and provide necessary data to the users. The various components of DBMS are shown below: -



1. DDL Compiler - Data Description Language compiler processes schema definitions specified in the DDL. It includes metadata information such as the name of the files, data items, storage details of each file, mapping information and constraints etc.

2. DML Compiler and Query optimizer - The DML commands such as insert, update, delete, retrieve from the application program are sent to the DML compiler for compilation into object code for database access. The object code is then optimized in the best way to execute a query by the query optimizer and then send to the data manager.

3. Data Manager - The Data Manager is the central software component of the DBMS also knows as Database Control System.

The Main Functions Of Data Manager Are: –

- Convert operations in user's Queries coming from the application programs or combination of DML Compiler and Query optimizer which is known as Query Processor from user's logical view to physical file system.
- Controls DBMS information access that is stored on disk.
- It also controls handling buffers in main memory.
- It also enforces constraints to maintain consistency and integrity of the data.
- It also synchronizes the simultaneous operations performed by the concurrent users.
- It also controls the backup and recovery operations.

4. Data Dictionary - Data Dictionary is a repository of description of data in the database. It contains information about

- Data - names of the tables, names of attributes of each table, length of attributes, and number of rows in each table.
- Relationships between database transactions and data items referenced by them which is useful in determining which transactions are affected when certain data definitions are changed.
- Constraints on data i.e. range of values permitted.
- Detailed information on physical database design such as storage structure, access paths, files and record sizes.
- Access Authorization - is the Description of database users their responsibilities and their access rights.
- Usage statistics such as frequency of query and transactions..Data dictionary is used to actually control the data integrity, database operation and accuracy. It may be used as an important part of the DBMS.

Importance of Data Dictionary -

Data Dictionary is necessary in the databases due to following reasons:

• It improves the control of DBA over the information system and user's understanding of use of the system.

• It helps in documentation the database design process by storing documentation of the result of every design phase and design decisions.

• It helps in searching the views on the database definitions of those views.

• It provides great assistance in producing a report of which data elements (i.e. data values) are used in all the programs.

• It promotes data independence i.e. by addition or modifications of structures in the database application program are not effected.

5. Data Files - It contains the data portion of the database.

6. Compiled DML - The DML complier converts the high level Queries into low level file access commands known as compiled DML.

7. End Users - end-user as the person who uses an Application.

People who deal with database- Data Base Administrator (DBA), Data manager File manger and Disk Manger

What are the functions of a database administrator?

1. Selection of hardware and software

- Keep up with current technological trends
- Predict future changes
- Emphasis on established off the shelf products

2. Managing data security and privacy

- Protection of data against accidental or intentional loss, destruction, or misuse
- Firewalls
- Establishment of user privileges
- Complicated by use of distributed systems such as internet access and client/ server technology.

How many major threats to database security can you think of?

1. Accidental loss due to human error or software/ hardware error.
2. Theft and fraud that could come from hackers or disgruntled employees.
3. Improper data access to personal or confidential data.
4. Loss of data integrity.
5. Loss of data availability through sabotage, a virus, or a worm.

3. Managing Data Integrity

- Integrity controls protects data from unauthorized use
- Data consistency
- Maintaining data relationship
- Domains- sets allowable values
- Assertions- enforce database conditions

4. Data backup

- We must assume that a database will eventually fail
- Establishment procedures
  - how often should the data be back-up?
  - what data should be backed up more frequently?
  - who is responsible for the back ups?
- Back up facilities
  - automatic dump- facility that produces backup copy of the entire database
  - periodic backup- done on periodic basis such as nightly or weekly
  - cold backup- database is shut down during backup
  - hot backup- a selected portion of the database is shut down and backed up at a given time
  - backups stored in a secure, off-site location

5. Database recovery

- Application of proven strategies for reinstallation of database after crash
- Recovery facilities include backup, data lossed by human error or failure in the hardware or supporting operating system – it is essential to be able to repair the data concerned with a minimum of delay and with as little effect as possible an the rest of the system.

Data manager

The data manger is the central software component of the DBMS. It sis sometimes reffered to as the data base control system. One of the function of the data manger is to convert operations in the user's quries coming directly via the query processor or indirectly via an application program from the user's logical view to a physical file system. The data manger is responsible for interfacing with the file system.

File manger

Responsibility for the structure of the files and manging the file space rest with the file manger. It is also responsibilte for locating the block containing the required record, requesting this block from the disk manager.

Disk manager

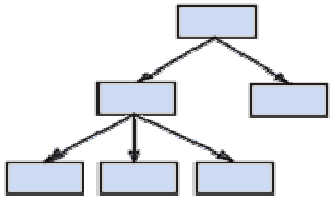
The disk manger is part of the operating system of the host computer and all physical input and output operations are performed by it. The disk manger transfers the block or page requested by file manager.

Data models

1. **Hierarchical data model.**
2. **network data model.**
3. **Relational data model.**

Hierarchical Model

Data is sorted hierarchically, using a downward tree. This model uses pointers to navigate between stored data. It was the first DBMS model.

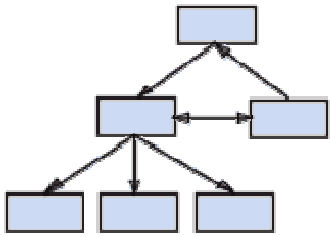


- The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments.
- This structure implies that a record can have repeating information, generally in the child data segments.
- Data in a series of records, which have a set of field values attached to it.
- It collects all the instances of a specific record together as a record type. These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows.
- To create links between these record types, the hierarchical model uses Parent Child Relationships. These are a 1:N mapping between record types.
- This is done by using trees, like set theory used in the relational model, "borrowed" from maths.
- For example, an organization might store information about an employee, such as name, employee number, department, salary.
  The organization might also store information about an employee's children, such as name and date of birth.

The employee and children data forms a hierarchy, where the employee data represents the parent segment and the children data represents the child segment.If an employee has three children, then there would be three child segments associated with one employee segment.In a hierarchical database the parent-child relationship is one to many.This restricts a child segment to having only one parent segment.

Network Data Model

Like the hierarchical model, this model uses pointers toward stored data. However, it does not necessarily use a downward tree structure.



- Resembles hierarchical model
- Difference child can have multiple parents
- Collection of records in 1: M relationships
- Set – Relationship of at least two record types
- Owner – Equivalent to the hierarchical model's parent
- Member – Equivalent to the hierarchical model's child

RELATIONAL DATA MODEL

The relational model (**RDBMS**, *Relational database management system*): The data is stored in two-dimensional tables (rows and columns). The data is manipulated based on the relational theory of mathematics.
A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organised in tables. A table is a collection of records and each record in a table contains the same fields.

UNIT 2

Entity Relationship model

What is it?

The entity-relationship model (or ER model) is a way of graphically representing the logical relationships of entities (or objects) in order to create a database. The ER model was first proposed by Peter Pin-Shan Chen of Massachusetts Institute of Technology (MIT) in the 1970s.

The whole purpose of ER modelling is to create an accurate reflection of the real world in a database. The ER model doesn't actually give us a database description. It gives us an

intermediate step from which it is easy to define a database. Let's look at an example. Suppose you are presented with the following situation and are told to create a database for it:

Every department within our company is in only one division. Each division has more than one department in it. We don't have an upper limit on the number of departments that a division can have. For example, the **New Business Development**---the one managed by Mackenzie---and **Higher Education** departments are both in the **Marketing** division.

Basic Constructs of E-R Modeling

Entity

One of the basic components of ER model is entity. An entity is any distinguishable object about which information is stored. These objects can be person, place, thing, event or a concept. Entities contain descriptive information. **Each entity is distinct.**

An entity may be physical or abstract. A person, a book, car, house, employee etc. are all physical entities whereas a company, job, or a university course, are abstract entities.

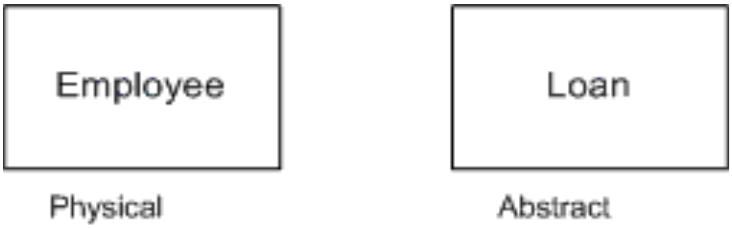


Fig - Physical and Abstract Entity

Another classification of entities can be independent or dependent (strong or weak) entity.

Entities are classified as independent or dependent (in some methodologies, the terms used are strong and weak, respectively). An independent entity is one, which does not rely on another entity for identification. A dependent entity is one that relies on another entity for identification. An independent entity exists on its own whereas dependent entity exists on the existence of some other entity. For example take an organization scenario. Here department is independent entity. Department manager is a dependent entity. It exists for existing depts. There won't be any department manager for which there is no dept.

Some entity types may not have any key attributes of their own. These are called weak entity types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with some of their attribute values. For example, take the license entity. It can't exist unless it is related to a person entity.

Attributes

After you identify an entity, then you describe it in real terms, or through its attributes. Attributes are basically properties of entity. We can use attributes for identifying and expressing entities. For example, Dept entity can have DeptName, DeptId, and DeptManager as its attributes. A car entity can have modelno, brandname, and color as its attributes.

A particular instance of an attribute is a value. For example, "Bhaskar" is one value of the attribute Name. Employee number 8005 uniquely identifies an employee in a company.

Relational Key constraints

A **constraint** is a rule that defines what data is valid for a given field

A relation is a set of tuples. By definition, all elements in a set are distinct hence all tuple in a relation must be distinct.

In relational model, tuples have no identity like object –identification. Tuple identity is totally value-based.

Therefore, we need *key constraint* that is the way of uniquely identify a tuple.

- Candidate key
- Super key
- Primary key
- Foreign key

Candidate key(K)

Candidate key are those attribute of relation, which have the properties of uniqueness and irreducibility.

Let K be asset of attribute of relation R. Then K is acandidate key for R if and only is it posses both of the following properties:

- Uniqueness: No legal value of R ever conatins two distinct touple with the same value of k.
- Irreducibility : No proper subset of k has the uniqueness property.

For example: In Employee relation(E_id and E_name) is unique, then it can be identified candidate key if and only if E_id and E_name individually are not Unique means it can not declare separately unique earlier.

(E_id)=candidate key

Then we declare

(E_id and E_name) = candidate key **(wrong)**

Super key(S)

Super key are those attribute of relation, which have the properties of uniqueness and but not neccessary irreducibility.

Let K be asset of attribute of relation R. Then K is acandidate key for R if and only is it posses both of the following properties:

- Uniqueness: No legal value of R ever conatins two distinct touple with the same value of k.
- Irreducibility : it have proper subset of k has the uniqueness property.

Thus we say that" A superset of a candidate key is super key"

(E_id)=candidate key

Then we declare

(E_id and E_name) = Super key

Primary key

A **primary key** is a field or combination of fields that uniquely identify a record in a table, so that an individual record can be located without confusion.

- Primary keys enforce entity integrity by uniquely
- identifying entity instances.
- Primary key will not allow "Null values" and "Duplicate values

A **constraint** is a rule that defines what data is valid for a given field. So a **primary key constraint** is a rule that says that the **primary key** fields cannot be null and cannot contain duplicate data.

Foreign key

A **foreign key** (sometimes called a referencing key) is a key used to link two tables together. Typically you take the primary key field from one table and insert it into the other table where it becomes a foreign key (it remains a primary key in the original table).

A **foreign key constraint** specifies that the data in a **foreign key** must match the data in the primary key of the linked table.

RELATION AND RELATIONSHIP SETS

A relationship is an association among several entities.

Example:
Hayes depositor A-102

customer entity relationship set account entity

_ A relationship set is a mathematical relation among n _ 2 entities, each taken from entity sets

f(e1, e2, ..., en) *j* e1 *2* E1, e2 *2* E2, ..., en *2* En*g*
where (e1, e2, ..., en) is a relationship.

ADDITIONAL FEATURES OF THE ER MODEL

Advantages and Disadvantages of ER Modeling (Merits and Demerits of ER Modeling)

Advantages

1. ER Modeling is simple and easily understandable. It is represented in business users language and it can be understood by non-technical specialist.
2. Intuitive and helps in Physical Database creation.
3. Can be generalized and specialized based on needs.
4. Can help in database design.
5. Gives a higher level description of the system.

Disadvantages

1. Physical design derived from E-R Model may have some amount of ambiguities or inconsistency.
2. Sometime diagrams may lead to misinterpretations

Conceptual design: (ER Model is used at this stage.)

– A database `schema' in the ER Model can be represented pictorially (ER diagrams).
– Can map an ER diagram into a relational schema.
Schema Refinement: (Normalization) Check relational schema for redundancies and related anomalies. Physical Database Design and Tuning: Consider typical workloads and further refine the database design.

Entity versus attribute

ENTITY:

One of the basic components of ER model is entity. An entity is any distinguishable object about which information is stored. These objects can be person, place, thing, event or a concept. Entities contain descriptive information. **Each entity is distinct.**

Attributes

After you identify an entity, then you describe it in real terms, or through its attributes. Attributes are basically properties of entity. We can use attributes for identifying and expressing entities.

For example, Dept entity can have DeptName, DeptId, and DeptManager as its attributes. A car entity can have modelno, brandname, and color as its attributes.

ENTITY VERSUS RELATIONSHIP.

ENTITY:

One of the basic components of ER model is entity. An entity is any distinguishable object about which information is stored. These objects can be person, place, thing, event or a concept. Entities contain descriptive information. **Each entity is distinct.**

RELATIONSHIP:
A relationship is an association among several entities.

Example:
Hayes depositor A-102

customer entity relationship set account entity

_ A relationship set is a mathematical relation among n _ 2 entities, each taken from entity sets

f(e1, e2, ..., en) *j* e1 *2* E1, e2 *2* E2, ..., en *2* En*g*
where (e1, e2, ..., en) is a relationship.

INTRODUCTION TO RELATIONAL MODEL

Relational database schema defines:

Names of tables in the database, the columns of each table, i.e., the column name and the data types of the column entries, integrity constraints, i.e., conditions that data entered into the tables is required to satisfy. Assume a database maintaining information about a small real-world subset: a company's departments and employees.

SQL FOREIGN KEY Constraint

A FOREIGN KEY in one table points to a PRIMARY KEY in another table.

Let's illustrate the foreign key with an example. Look at the following two tables:

The "Persons" table:

P_Id	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

The "Orders" table:

O_Id	OrderNo	P_Id
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Note that the "P_Id" column in the "Orders" table points to the "P_Id" column in the "Persons" table.

The "P_Id" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "P_Id" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

SQL FOREIGN KEY Constraint on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "P_Id" column when the "Orders" table is created:

MySQL:

```
CREATE TABLE Orders
(
O_Id int NOT NULL,
OrderNo int NOT NULL,
P_Id int,
PRIMARY KEY (O_Id),
FOREIGN KEY (P_Id) REFERENCES Persons(P_Id)
)
```

INTEGRITY CONSTRAINTS

Integrity constraints are used to ensure accuracy and consistency of data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity. There are many types of integrity constraints that play a role in referential integrity.

Entity Integrity

The entity integrity constraint states that no primary key value can be null. This is because the primary key value is used to identify individual tuples in a relation. Having null value for the primary key implies that we cannot identify some tuples.This also specifies that there may not be any duplicate entries in primary key column key word

Referential Integrity

The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation. It is a rule that maintains consistency among the rows of the two relations.

Domain Integrity

The domain integrity states that every element from a relation should respect the type and restrictions of its corresponding attribute. A type can have a variable length which needs to be respected. Restrictions could be the range of values that the element can have, the default value if none is provided, and if the element can be NULL.

User Defined Integrity

A business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business. E.g.: Age>=18 && Age<=60

QUERYING RELATIONAL DATA

When a company asks you to make them a working, functional DBMS which they can work with, there are certain steps to follow. Let us summarize them here:

1. **Gathering information**: This could be a written document that describes the system in question with reasonable amount of details.
2. **Producing ERD**: ERD or Entity Relationship Diagram is a diagrammatic representation of the description we have gathered about the system.
3. **Designing the database**: Out of the ERD we have created, it is very easy to determine the tables, the attributes which the tables must contain and the relationship among these tables.
4. **Normalization**: This is a process of removing different kinds of impurities from the tables we have just created in the above step.

LOGICAL DATABASE DESIGN: ER TO RELATION

Step 1

Let us take a very simple example and we try to reach a fully organized database from it. Let us look at the following simple statement:

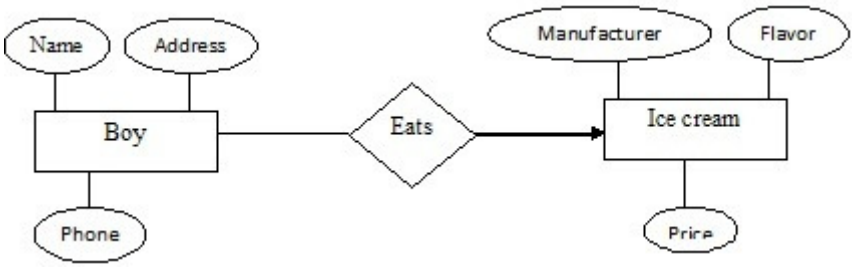
A boy eats an ice cream.

This is a description of a real word activity, and we may consider the above statement as a written document (very short, of course).

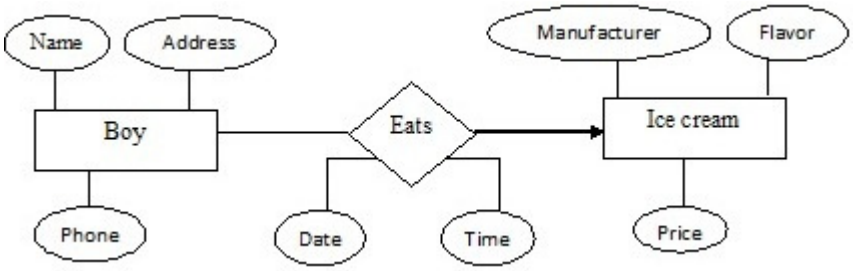
Step 2

Now we have to prepare the ERD. Before doing that we have to process the statement a little. We can see that the sentence contains a subject (**boy**), an object (**ice cream**) and a verb (**eats**) that defines the relationship between the subject and the object. Consider the nouns as entities (**boy** and **ice cream**) and the verb (**eats**) as a relationship. To plot them in the diagram, put the nouns within rectangles and the relationship within a diamond. Also, show the relationship with a directed arrow, starting from the subject entity (**boy**) towards the object entity (**ice cream**).

Well, fine. Up to this point the ERD shows how **boy** and **ice cream** are related. Now, every boy must have a name, address, phone number etc. and every ice cream has a manufacturer, flavor, price etc. Without these the diagram is not complete. These items which we mentioned here are known as attributes, and they must be incorporated in the ERD as connected ovals.



But can only entities have attributes? Certainly not. If we want then the relationship must have their attributes too. These attribute do not inform anything more either about the **boy** or the **ice cream**, but they provide additional information about the relationships between the **boy** and the **ice cream.**



Step 3

We are almost complete now. If you look carefully, we now have defined structures for at least three tables like the following:

Boy

Name	Address	Phone

Ice Cream

Manufacturer	Flavor	Price

Eats

Date	Time

However, this is still not a working database, because by definition, database should be "collection of related tables." To make them connected, the tables must have some common attributes. If we chose the attribute Name of the Boy table to play the role of the common attribute, then the revised structure of the above tables become something like the following.

Boy

Name	Address	Phone

Ice Cream

Manufacturer	Flavor	Price	Name

Eats

Date	Time	Name

This is as complete as it can be. We now have information about the boy, about the ice cream he has eaten and about the date and time when the eating was done.

Cardinality of Relationship

While creating relationship between two entities, we may often need to face the cardinality problem. This simply means that how many entities of the first set are related to how many entities of the second set. Cardinality can be of the following three types.

One-to-One

Only one entity of the first set is related to only one entity of the second set. E.g. **A teacher teaches a student.** Only one teacher is teaching only one student. This can be expressed in the following diagram as:

One-to-Many

Only one entity of the first set is related to multiple entities of the second set. E.g. **A teacher teaches students.** Only one teacher is teaching many students. This can be expressed in the following diagram as:

Many-to-One

Multiple entities of the first set are related to multiple entities of the second set. E.g. **Teachers teach a student.** Many teachers are teaching only one student. This can be expressed in the following diagram as:

Many-to-Many
Multiple entities of the first set is related to multiple entities of the second set. E.g. **Teachers teach students.** In any school or college many teachers are teaching many students. This can be considered as a two way one-to-many relationship. This can be expressed in the following diagram as:

In this discussion we have not included the attributes, but you can understand that they can be used without any problem if we want to.

INTRODUCTION TO VIEWS

In database theory, a **view** is the result set of a *stored* query — or map-and-reduce functions — on the data, which the database users can query just as they would a persistent database collection object. This pre-established query command is kept in the database dictionary. Unlike ordinary *base tables* in a relational database, a view does not form part of the physical schema: as a result set, it is a virtual table computed or collated from data in the database, dynamically when access to that view is requested. Changes applied to the data in a relevant *underlying table* are reflected in the data shown in subsequent invocations of the view.

Views can provide advantages over tables:

- Views can represent a subset of the data contained in a table; consequently, a view can limit the degree of exposure of the underlying tables to the outer world: a given user may have permission to query the view, while denied access to the rest of the base table.
- Views can join and simplify multiple tables into a single virtual table
- Views can act as aggregated tables, where the database engine aggregates data (sum, average etc.) and presents the calculated results as part of the data
- Views can hide the complexity of data; for example a view could appear as Sales2000 or Sales2001, transparently partitioning the actual underlying table
- Views take very little space to store; the database contains only the definition of a view, not a copy of all the data which it presents
- Depending on the SQL engine used, views can provide extra security

DESTROYING/ALTERING TABLES AND VIEWS

The ALTER TABLE statement changes a Base table's definition. The required syntax for the ALTER TABLE statement is:

ALTER TABLE <Table name> <alter table action>

<alter table action> ::=
ADD [COLUMN] <Column definition> |
ALTER [COLUMN] <Column name> SET DEFAULT default value |
ALTER [COLUMN] <Column name> DROP DEFAULT |
ALTER [COLUMN] <Column name> ADD SCOPE <Table name list> |
ALTER [COLUMN] <Column name> DROP SCOPE {RESTRICT | CASCADE} |
DROP [COLUMN] <Column name> {RESTRICT | CASCADE} |
ADD <Table Constraint> |
DROP CONSTRAINT <Constraint name> {RESTRICT | CASCADE}

<Table name list> ::=
(<Table name> [{,<Table name>}...]) |
<Table name>

The <Table name> must identify an existing Base table whose owner is either the current <AuthorizationID> or a Role that the current <AuthorizationID> may use. That is, only the <AuthorizationID> that owns the Table may alter it. ALTER TABLE can be used to change a persistent Base table, a GLOBAL TEMPORARY Base table or a created LOCAL TEMPORARY Base table, but you can't use it to change a declared LOCAL TEMPORARY Base table.

ADD [COLUMN] clause

The effect of ALTER TABLE <Table name> ADD [COLUMN] <Column definition>, e.g.:

```
ALTER TABLE Table_1 ADD COLUMN
  column_1 SMALLINT DEFAULT 150
    CONSTRAINT constraint_1 NOT NULL NOT DEFERRABLE;
```

is that the Table named will increase in size by one Column: the Column defined by the <Column definition>. The <keyword> COLUMN in the ADD [COLUMN] clause is noise and can be omitted. For example, these two SQL statements are equivalent:

```
ALTER TABLE Table_1 ADD COLUMN
  column_1 SMALLINT DEFAULT 150;
```

```
ALTER TABLE Table_1 ADD
  column_1 SMALLINT DEFAULT 150;
```

Adding a new Column to a Table has a four-fold effect:

1. The degree (i.e.: the number of Columns) of the Table is increased by 1; the new Column's ordinal position in the Table is the new degree of the Table.
2. Every <AuthorizationID> that has a SELECT, UPDATE, INSERT, or REFERENCES Privilege on all existing Columns of the Table receives a matching set of Privileges on the new Column. The grantor of the new Privilege(s) is the same as the grantor of the previous Privileges(s) and so is the grantability of the new Privilege(s).
3. The value of the new Column for every existing row of the Table is set to its default value.
4. The Column is added to the Column list of every UPDATE Trigger event for all Triggers that act on the Table. However, adding a new Column to a Table has no effect on any existing View definition or Constraint definition that refers to the Table because implicit <Column name>s in these definitions are replaced by explicit <Column name>s the first time the View or Constraint is evaluated.

CODD'S RULES

Codd's 12 rules are a set of twelve rules proposed by Edgar F. Codd, a pioneer of the relational model for databases, designed to define what is required from a database management system in order for it to be considered *relational*, i.e., an RDBMS. They are sometimes jokingly referred to as "Codd's Twelve Commandments".

Codd produced these rules as part of a personal campaign to prevent his vision of the relational database being diluted, as database vendors scrambled in the early 1980s to repackage existing products with a relational veneer. Rule 12 was particularly designed to counter such a positioning. In fact, the rules are so strict that all popular so-called "relational" DBMSs fail on many of the criteria.[

1. **Information Rule**: **All information in a relational database including table names, column names are represented by values in tables**. Everything within the database exists in tables and is accessed via table access routines. This simple view of data speeds design and learning. User productivity is improved since knowledge of only one language is necessary to access all data such as description of the table and attribute definitions, integrity constraints. Action can be taken when the constraints are violated. Access to data can be restricted. All these information are also stored in tables.

2. **Guaranteed Access Rule**: **Every piece of data in a relational database, can be accessed by using combination of a table name, a primary key value that identifies the row and**

column name which identified a cell. User productivity is improved since there is no need to resort to using physical pointers addresses. Provides data independence. Possible to retrieve each individual piece of data stored in a relational database by specifying the name of the table in which it is stored, the column and primary key which identified the cell in which it is stored.

3. **Systematic Treatment of Nulls Rule**: **The RDBMS handles records that have unknown or inapplicable values in a pre-defined fashion.** The DBMS must allow each field to remain null (or empty). Specifically, it must support a representation of "missing information and inapplicable information" that is systematic, distinct from all regular values (for example, "distinct from zero or any other number", in the case of numeric values), and independent of data type. It is also implied that such representations must be manipulated by the DBMS in a systematic way.

Also, the RDBMS distinguishes between zeros, blanks and nulls in the records hand handles such values in a consistent manner that produces correct answers, comparisons and calculations. Through the set of rules for handling nulls, users can distinguish results of the queries that involve nulls, zeros and blanks. Even though the rule doesn't specify what should be done in the case of nulls it specifies that there should be a consistent policy in the treatment of nulls.

4. **Active On-line catalog based on the relational model**: **The description of a database and in its contents are database tables and therefore can be queried on-line via the data manipulation language**. The system must support an online, inline, relational catalog that is accessible to authorized users by means of their regular query language. That is, users must be able to access the database's structure (catalog) using the same query language that they use to access the database's data. The database administrator's productivity is improved since the changes and additions to the catalog can be done with the same commands that are used to access any other table. All queries and reports can also be done as any other table.

5. **Comprehensive Data Sub-language Rule**: **A RDBMS may support several languages. But at least one of them should allow user to do all of the following: define tables and views, query and update the data, set integrity constraints, set authorizations and define transactions.** A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings and that is comprehensive in supporting all the following items

- Data Definition
- View Definition
- Data Manipulation (Interactive and by program).
- Integrity Constraints
- Authorization.

Every RDBMS should provide a language to allow the user to query the contents of the RDBMS and also manipulate the contents of the RDBMS

User productivity is improved since there is just one approach that can be used for all database operations. In a multi-user environment the user does not have to worry about the data integrity an such things, which will be taken care by the system. Also, only users with proper authorization will be able to access data.

6. **View Updating Rule**: **Any view that is theoretically updateable can be updated using the RDBMS.** Data consistency is ensured since the changes made in the view are transmitted to the base table and vice-versa.

7. **High-Level Insert, Update and Delete**: **The RDBMS supports insertions, updation and deletion at a table level**. The system must support set-at-a-time *insert*, *update*, and *delete* operators. This means that data can be retrieved from a relational database in sets constructed of data from multiple rows and/or multiple tables. This rule states that insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table. The performance is improved since the commands act on a set of records rather than one record at a time.

8. **Physical Data Independence**: **The execution of adhoc requests and application programs is not affected by changes in the physical data access and storage methods. "Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods."** Database administrators can make changes to the physical access and storage method which improve performance and do not require changes in the application programs or requests. Here the user specified what he wants an need not worry about how the data is obtained.

9. **Logical Data Independence**: **Logical changes in tables and views such adding/deleting columns or changing fields lengths need not necessitate modifications in the programs or in the format of adhoc requests. "Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit un-impairment are made to the base tables."** The database can change and grow to reflect changes in reality without requiring the user intervention or changes in the applications. For example, adding attribute or column to the base table should not disrupt the programs or the interactive command that have no use for the new attribute.

10. **Integrity Independence**: **Like table/view definition, integrity constraints are stored in the on-line catalog and can therefore be changed without necessitating changes in the application programs.** Integrity constraints specific to a particular RDB must be definable in

the relational data sub-language and storable in the catalog. At least the Entity integrity and referential integrity must be supported.

"Integrity constraints specific to a particular relational data base must be definable in the relational data sub-language and storable in the catalog, not in the application programs."

If a column only accepts certain values, then it is the RDBMS which enforces these constraints and not the user program, this means that an invalid value can never be entered into this column, whilst if the constraints were enforced via programs there is always a chance that a buggy program might allow incorrect values into the system.

11. **Distribution Independence**: **Application programs and adhoc requests are not affected by change in the distribution of physical data.** Improved systems reliability since application programs will work even if the programs and data are moved in different sites.

> The distribution of portions of the database to various locations should be invisible to users of the database. Existing applications should continue to operate successfully :
>
> 1. when a distributed version of the DBMS is first introduced; and
> 2. when existing distributed data are redistributed around the system

12.**No subversion Rule**: **If the RDBMS has a language that accesses the information of a record at a time, this language should not be used to bypass the integrity constraints.** If the system provides a low-level (record-at-a-time) interface, then that interface cannot be used to subvert the system, for example, bypassing a relational security or integrity constraint. This is necessary for data integrity.

UNIT 3

Introduction to schema refinement:

> INTRODUCTION TO SCHEMA REFINEMENT Problems Caused by Redundancy Storing the same information redundantly, that is, in more than one place within a database, can lead to several problems: Redundant storage: Some information is stored repeatedly. Update anomalies: If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated. Insertion anomalies: It may not be possible to store some information unless some other information is stored as well. Deletion anomalies: It may not be possible to delete some information without losing some other information as well. Consider a relation obtained by translating a variant of the Hourly Emps entity set Ex: Hourly Emps(ssn , name , lot , rating , hourly wages , hours worked) The key for Hourly Emps is ssn . In addition, suppose that the hourly wages attribute is determined by the rating attribute. That is, for a given rating

value, there is only one permissible hourly wages value. This IC is an example of a functional dependency. It leads to possible redundancy in the relation Hourly Emps

What is Normalization?

Normalization is a process designed to remove redundant data from the records stored in the database and thus to reduce the potential for errors in data input.

In the design of a relational database management system (RDBMS), the process of organizing data to minimize redundancy is called normalization. The goal of database normalization is to decompose relations with anomalies in order to produce smaller, well-structured relations. Normalization usually involves dividing large, badly-formed tables into smaller, well-formed tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.

DEPENDENCIES in RDBMS

- Functional Dependency
- Fully Functional Dependency
- Transitive Dependency
- Partial Functional Dependency

Functional dependency (FD)

A Functional Dependency is an association between two attributes of the same relational databse table. One of the attributes is called the determinant and the other attribute is called the determined. For each value of the determinant there is associated one and only one value of the determined.

If A is the determinant and b is the determined then we say that A Functionally determines B and graphically represent this A→B. the symbols A→ can also be expressed as B functionally determined by A.

Fully Functional Dependency

It is based on concept of full functional dependency. A functional dependency X Y is full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.

A functional dependency X → Y is full functional dependency if any attribute A removed from X. It means that the dependency does not hold any more then it is not Fully Functional dependence. Means each attribute is functionally dependent.
Transitive Dependency

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

A transitive dependency can occur only in a relation that has three or more attributes. Let A, B, and C designates three distinct attributes (or distinct collections of attributes) in the relation. Suppose all three of the following conditions hold:

1. A → B
2. It is not the case that B → A
3. B → C
4. Partial Functional Dependency:
5. A partial dependency is a dependency where A is functionally dependant on B (A → B), but there is some attribute on A that can be removed from A and yet the dependency stills holds. For instance if the relation existed
6. StaffNo, sName → branchNo
7. Then you could say that for every StaffNo, sName there is only one value of branchNo, but since there is no relation between branchNo and staffNo the relation is only partial.

First Normal Form

- Eliminate repeating groups in individual tables.
- Create a separate table for each set of related data.
- Identify each set of related data with a primary key.

Do not use multiple fields in a single table to store similar data. For example to track an inventory item that may come from two possible sources an inventory record may contain fields for Vendor Code 1 and Vendor Code 2.
But what happens when you add a third vendor? Adding a field is not the answer; it requires program and table modifications and does not smoothly accommodate a dynamic number of vendors. Instead place all vendor information in a separate table called Vendors then link inventory to vendors with an item number key or vendors to inventory with a vendor code key.

Second Normal Form

- Create separate tables for sets of values that apply to multiple records.
- Relate these tables with a foreign key.

Records should not depend on anything other than a table's primary key (a compound key if necessary). For example consider a customer's address in an accounting system. The address is needed by the Customers table but also by the Orders Shipping Invoices Accounts Receivable and Collections tables. Instead of storing the customer's address as a separate entry in each of these tables store it in one place either in the Customers table or in a separate Addresses table.

Third Normal Form

- Eliminate fields that do not depend on the key.

Values in a record that are not part of that record's key do not belong in the table. In general any time the contents of a group of fields may apply to more than a single record in the table consider placing those fields in a separate table.
For example in an Employee Recruitment table a candidate's university name and address may be included. But you need a complete list of universities for group mailings. If university information is stored in the Candidates table there is no way to list universities with no current candidates. Create a separate Universities table and link it to the Candidates table with a university code key.

Denormalization

De-normalization is the process of attempting to optimize the performance of a database by adding redundant data. It is sometimes necessary because current DBMSs implement the relational model poorly.

A true relational DBMS would allow for a fully normalized database at the logical level, while providing physical storage of data that is tuned for high performance. De-normalization is a technique to move from higher to lower normal forms of database modeling in order to speed up database access.

UNIT 4

What are these objects?

When you create a database, Access offers you *Tables, Queries, Forms, Reports, Macros,* and *Modules*. Here's a quick overview of what these are and when to use them.

 ***Tables.*** All data is stored in tables. When you create a new table, Access asks you define **fields** (column headings), giving each a unique name, and telling Access the **data type**. Use the "Text" type for most data, including numbers that don't need to be added e.g. phone numbers or postal codes. Using Wizards, Access will walk you through the process of creating common tables such as lists of names and addresses. Once you have defined a table's structure, you can enter data. Each new row that you add to the table is called a **record**. To define **relationships** between tables:

- in Access 2007 or later: Database Tools | Relationships,
- in Access 95 — 2003: Tools | Relationships,
- in Access 1 — 2: Edit | Relationships.

 ***Queries.*** Use a query to find or operate on the data in your tables. With a query, you can display the records that match certain **criteria** (e.g. all the members called "Barry"), **sort** the data as you please (e.g. by Surname), and even **combine data** from different tables. You can **edit** the data displayed in a query (in most cases), and the data in the underlying table will change. Special queries can also be defined to make **wholesale changes** to your data, e.g. delete all members whose subscriptions are 2 years overdue, or set a "State" field to "WA" wherever postcode begins with 6.

 ***Forms.*** These are screens for **displaying** data from and **inputting** data into your tables. The basic form has an appearance similar to an index card: it shows only one record at a time, with a

different field on each line. If you want to control how the records are **sorted,** define a query first, and then create a form based on the query. If you have defined a one-to-many relationship between two tables, use the "**Subform**" Wizard to create a form which contains another form. The subform will then display only the records matching the one on the main form.

 ***Reports.*** If forms are for input, then reports are for output. **Anything you plan to print** deserves a report, whether it is a list of names and addresses, a financial summary for a period, or a set of mailing labels. Again the Access Wizards walk you through the process of defining reports.

 ***Pages*** (Access 2000 - 2003). Use pages to enter or display data via Internet. Pages are stored as HTML files, with data read from and written to the database. Michael Kaplan has published a free utility to convert Access forms and reports into Data Access Pages. (Pages were deprecated in Access 2007.)

 ***Macros.*** An Access Macro is a **script** for doing some job. For example, to create a button which opens a report, you could use a macro which fires off the "OpenReport" action. Macros can also be used to **set one field** based on the value of another (the "SetValue" action), to **validate** that certain conditions are met before a record saved (the "CancelEvent" action) etc.

 ***Modules.*** This is where you **write your own functions** and programs if you want to. Everything that can be done in a macro can also be done in a module, but you don't get the Macro interface that prompts you what is needed for each action. Modules are far more powerful, and are essential if you plan to write code for a **multi-user** environment, since macros cannot include error handling. Most serious Access users start out with macros to get a feel for things, but end up using modules almost exclusively. On the other hand, if your needs are simple, you may never need to delve into the depths of Access modules.

SQL DML and DDL

SQL can be divided into two parts: The Data Manipulation Language (DML) and the Data Definition Language (DDL).

The query and update commands form the DML part of SQL:

- **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

The DDL part of SQL permits database tables to be created or deleted. It also define indexes (keys), specify links between tables, and impose constraints between tables. The most important DDL statements in SQL are:

- **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\

AGGREGATE OPERATIN

- **1. SQL SUM() Function -** The SUM() function returns the total sum of a numeric column.

- **SQL SUM() Syntax**

```
SELECT SUM(column_name) FROM table_name
```

. SQL AVG() Function

The AVG() function returns the average value of a numeric column.

SQL AVG() Syntax

```
SELECT AVG(column_name) FROM table_name
```

SQL MIN() Function

The MIN() function returns the smallest value of the selected column.

SQL MIN() Syntax

```
SELECT MIN(column_name) FROM table_name
```

SQL MAX() Function

The MAX() function returns the largest value of the selected column.

SQL MAX() Syntax

```
SELECT MAX(column_name) FROM table_name
```

The UNION, EXCEPT, and INTERSECT operators all operate on multiple result sets to return a single result set:

- The UNION operator combines the output of two query expressions into a single result set. Query expressions are executed independently, and their output is combined into a single result table.
- The EXCEPT operator evaluates the output of two query expressions and returns the difference between the results. The result set contains all rows returned from the first query expression except those rows that are also returned from the second query expression.
- The INTERSECT operator evaluates the output of two query expressions and returns only the rows common to each.

The following figure illustrates these concepts with Venn diagrams, in which the shaded portion indicates the result set.

UNION, EXCEPT, INTERSECT

UNION combines the rows from two or more result sets into a single result set.

EXCEPT evaluates two result sets and returns all rows from the first set that are not also contained in the second set.

INTERSECT computes a result set that contains the common rows from two result sets.

UNION, INTERSECT, and EXCEPT operators can be combined in a single UNION expression.

In statements that include multiple operators, the default order of evaluation (precedence) for these operators is left to right; however, the INTERSECT operator is evaluated before UNION and EXCEPT. The order of evaluation can be modified with parentheses.

NESTED QUERIES

Queries within the query are called the nested query.

Example:

```
SELECT Products.ProductName, Products.UnitPrice
FROM Products
WHERE (((Products.UnitPrice) >
(SELECT AVG([UnitPrice]) From Products)))
ORDER BY Products.UnitPrice DESC;
```

NULL VALUES

Null is a special marker used in SQL to indicate that a data value does not exist in the database. Introduced by the creator of the relational database model, E. F. Codd, SQL Null serves to fulfill the requirement that all true RDBMS support a representation of missing information and/or inapplicable information

References:

Elmasri, R. and S B Navathe; *Fundamentals of Database Systems,* Addison Wesley, 2000.
http://en.wikipedia.org/wiki/Database
http://en.wikipedia.org/wiki/Relational_database_management_system