**MODULE 1**

A database-management system (DBMS**)** is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information

and providing mechanisms for the manipulation of information. In addition, the

database system must ensure the safety of the information stored, despite system

crashes or attempts at unauthorized access. If data are to be shared among several

users, the system must avoid possible anomalous results. Because information is so important in most organizations, computer scientists have developed a large body of concepts and techniques for managing data.

**PURPOSE OF DATABASE SYSTEM**

* **File System Versus DBMS**
* **Concurrent access Anomalies**
* **Data redundancy**
* **Difficulty in accessing data**
* **Data isolation**
* **Atomicity Problem**
* **Security Problem**

**VIEW OF DATA**

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an abstractview of the data. That is, the system hides certain details of how the data are stored and maintained.

**DATA ABSTRACTION**

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users’ interactions with the system:

**Levels of Abstraction**

**External view**

**External view**

**External view**

**physical**

**conceptual**

**Physical level**: The lowest level of abstraction describes *how* the data are actually

stored. The physical level describes complex low-level data structures

in detail.

**Logical level:** The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as physical data independence. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

**View level:** The highest level of abstraction describes only part of the entire

database.

**DATA MODEL**

A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints. A data model provides a way to describe the design of a database at the physical, logical, and view levels.

The data models can be classified into four different categories:

**Relational Model**:The relational model uses a collection of tables to represent

both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as **relations**.

**Entity-Relationship Model**. The entity-relationship (E-R) data model uses a collection of basic objects, called *entities*, and*relationships* among these objects.

An entity is a “thing” or “object” in the real world that is distinguishable from other objects.

**Object-Based Data Model:**Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology.This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. The object-relational data model combines features of the object-oriented data model and relational data model.

**Semistructured Data Model**:The semistructured data model permits the specification of data where individual data items of the same type may have different sets of attributes. The **Extensible Markup Language (XML)** is widely used to represent semistructured data.

**INSTANCES AND SCHEMAS**

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database. The overall design of the database is called the database **schema**. Schemas are changed infrequently, if at all.The concept of database schemas and instances can be understood by analogy to a program written in a programming language.Adatabase schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an *instance* of a database schema.

Database systems have several schemas, partitioned according to the levels

of abstraction. The physical schema describes the database design at the physical

level, while the logical schema describes the database design at the logical level.

A database may also have several schemas at the view level, sometimes called

subschemas, that describe different views of the database.

**DATABASE LANGUAGES**

**DDL :** To specify the db schema

**DML :** To express database queries and updates.

A data-manipulation language (DML) is a language that enables users to access

or manipulate data as organized by the appropriate data model. The types of access are:

• Retrieval of information stored in the database

• Insertion of new information into the database

• Deletion of information from the database

• Modification of information stored in the database

**DMLs are classified into two**:

**Procedural DMLs** : require a user to specify what data are needed and how to get those data.

**Declarative DMLs** : require a user to specify what data are needed without specifying how to get those data.

**DDL COMMANDS**

* Used to specify the database schema by a set of definitions called a DDL.
* Used to specify the storage structure
* Facility to specify constraints

**Constraints:**

Domain Constraints: A domain of possible values must be associated with every attribute.

Eg; integer types, character types etc.

**Referential Integrity:**It ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in other relation.

**Eg:** the dept\_name in a course record must appear in the dept\_name attribute of some record of the department relation.

**Assertions**:An assertion is a condition that the database must always satisfy.

**Authorization:**Differentiating users based on

**Read authorization:** allows only reading, but not modification

**Insert authorization:** insertion of new data, no modification

**Update:** allows updation, not deletion

**Delete**: deletion of data

**DATABASE USERS**

The people who work with databases include

* Database users,
* System analysts,
* Application programmers, and
* Database administrators (DBA).

**Database users**: Database users are those who interact with the database inorder to query and update the database, and generate reports.

Database users are further classified into:

* Naive users
* Sophisticated users
* Specialized users

**Naive users:**The users who maintain the database by invoking some already written application programs.

* For example, super market s/w user.
* end user.

**Sophisticated users:**The users, who are interact DBMS without writing any application programs.

* For example writing SQL
* Eg: users in DBMS LAb

**Specialized users:** The users who write specialized database programs, Specialized users write applications such as computer-aided design systems, knowledge-base, expert systems that store data having complex data types.

**Eg: -** users who write artificial intelligence programs like prolog, Lisp, etc,.. (eg: - nested queries.)

**System analysts**: determine the requirements of the database users (especially naive users) to create a solution for their business need, and focus on non-technical and technical aspects.

**The non-technical:**aspects involve defining system requirements, interaction between business users and technical staff, etc.

**Technical:** aspects involve developing the specification for user interface (application programs).

**Application programmers:** are the computer professionals who implement the specifications given by the system analysts, and develop application programs For the naive users.

**Eg:-** PHP programmers, Java programmers…

Develop packages that facilitate data access for end users.

[**Database administrator (DBA)**](http://my.safaribooksonline.com/9788131731925/gloss01) is a person who has central control over both data and application programs.

Some of the responsibilities of DBA are:

**1. Schema Definition:**The DBA creates the original db schema by executing a set of DDL statements.

**2. Schema and Physical Organization Modification:**The DBA carries out the changes to the schema and physical organization to reflect the changing needs of the organization.

**3. Storage and Access –method modification:** Creates the appropriate storage structure and access methods by writing a set of definitions which is translated by the data storage and DDL compiler.

**4. Granting of authorization for data access:** Different types of authorization, the DBA can regulate which parts of the database, various users can access.

**5. Routine Maintenance:**

* Backing up the database periodically.
* Ensuring enough space is available
* Monitoring the performance

**ENTITY RELATIONSHIP MODEL**

The entity-relationship (E-R) data model uses a collection of basic objects, called *entities*, and *relationships* among these objects. An entity is a “thing” or “object” in the real world that is distinguishable from other objects.

**The E-R Model constitutes:**

* Entity
* Entity Sets
* Attributes/Properties

**ER Diagram (Entity-Relationship Diagram)**

In 1976 proposed by Peter Chen ER diagram is detailed, logical representation of the data for an organization or business area. Three principal components of ER Diagram:

* **Entity set:** Collection of entities of same type that shares a common properties or characteristics or attributes.
* **Attribute:** Entities are described in a database by a set of attributes.
* **Relationship:** A relationship is an association among several entities

**ENTITY**

A thing or object in the real world , i.e., distinguishable from or other objects. An Entity can be a person, place, object or concept in the real world .Object with physical existence : person, car, book, etc .Object with conceptual existence: university, course, loan etc

**Eg;** each student in a student information DB

**ENTITY SET**

Collection of entities of same type that shares a common properties or characteristics or attributes.

**Eg :**Student: Name, Phone no etc.

Book : entity collection of book 🡪 entity set

**ATTRIBUTE**

Is the property or characteristics of the entity

**Eg:**

Employee

Employee is an entity with following attributes

Name, Age, Sex.

**Types of Attributes**

* Simple Attribute
* Composite Attribute
* Single valued Attribute
* Multi valued Attribute
* Derived Attribute
* Null Attribute

**1.Simple Attribute**

It cannot be divided into sub parts

Eg; student rollno

**2. Composite Attributes**

Can be divided into subparts or other attributes.

Eg;Student Name can be further divided into

* Firstname
* Lastname
* Middlename

**3. Single valued Attribute**

This will be having single values for a particular attribute.

Eg; Age, Id etc.

**4. Multivalued Attribute**

This will be having multiple values for a particular attribute.

Eg: Student->Phno

**5.Derived Attribute**

That are not stored directly, but can be derived from the stored attribute

**Eg:**

For an employee, emp\_year attribute is an example, because year of service can be calculated from the current date and emp\_joined date

The value of age can be derived from the attribute DOB

**6. Null Attribute**

Some case a particular entity may not have a value for an attribute

**Eg;**

Apartment number attribute of an address applies only to address that are in apartments, not to other residential address

**Stored Attribute**

Attributes which are already stored in the DB and from which values of another attribute is derived.

Eg: Date of Birth

**Key attribute of an entity set**

An attribute whose value is unique for each individual entity in the entity set

**Eg:** Rollno number is Unique for each students in the class

**STRONG & WEAK ENTITY TYPES**

**Strong Entity**

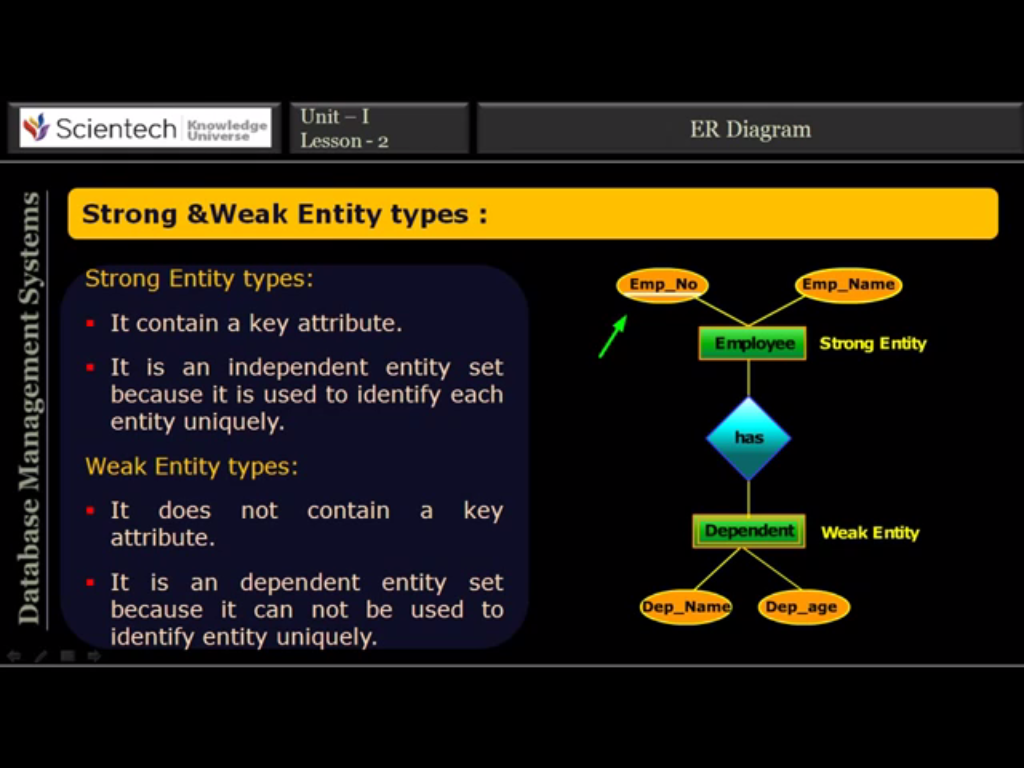
It contains a key attribute.It is an independent entity set. It is used to identify entity uniquely

**Weak entity**  
It does not contain a key attribute.It is a dependent entity set. It cannot be used to identify entity uniquely

**Eg**

Employee 🡪 strong entity type, because it contains Emp\_no as a key attribute

Dependent🡪 weak entity type, because it has no key attribute



**RELATIONSHIP AND RELATIONSHIP-SETS**

**Relationship**

It is an association among several entities.

**Eg**:Anjana works in MCA department

Anjana-entity

Works in-relationship

MCA department-entity

**Relationship Set**

* Collection of similar relations.

**Degree of a relationship set:**

* It is the no. of entity sets that participate in a relation set.
* Relationship that involves 2 entity sets are called binary (degree-2).
* Relationship that involves 3 entity sets are called ternary(degree-3).

**Relationship cardinality (Mapping Cardinalities)**

Express the No of entity to which another entity can be associated via a relationship.

Based on the cardinality, binary relationship can be classified into

* One-to-one
* One-to-Many
* Many-to-one
* Many-to many

**One-to-One**

* Each entity in a relationship will have exactly one relationship
* Eg: one college can have at most one principal and one principal can assigned to only one college.

**One-to-Many**

* An entity in A is associated with any number of entities in B & an entity in B is associated with at most one entity in A.
* **Eg:**One department can appoint any number of faculty members but one faculty member is assigned to only one department.

**Many-to-One**

* An entity in A is associated with at most one entity in B, and an entity in B is associated with any number of entities in A
* Eg:Course has many students but a student can be opt for only one course

**Many-to-Many**

* Entities of both sides of the relationship can have many related entities on the other side
* **Eg**:One faculty member can be assigned to teach many courses and one course may be taught by many facility members.

**Participation Constraints**

Two types:

**Total participation**

When all the entities from an entity set E participate in at least one relationship in R is called total participation. It is represented by double arrow

**Partial participation**

When it is not necessary for all entities in entity set E participate in a relationship R is called partial participation.

**KEYS**

Key is defined as the column or attribute of the database table, used to identify one or more instances of the table.

**Superkey :** A superkey is a set of one or more attributes that can uniquely identify an entity in the entity set.

**Eg:** rollno for the entity set student will uniquely identify a student. So rollno is a superkey. The combination of studentname and rollno is also a superkey for the entity set student.

**Candidate key:**It is a set of one or more fields that can uniquely identify an entity in the entity set. There can be multiple candidate keys in one table.Each candidate key can work as primary key.Only one candidate key can work as primary key.

**Eg:**In student table id, rollno, regno are candidate keys since all these 3 fields can be worked as primary keys.

**Alternate key or Secondary key**

It is a key that can be used as a primary key. Basically it is any candidate key which is currently not selected as the primary key.

**Eg**: In the student table if id is taken as primary key, then rollno and regno will be alternate keys.

**Composite keys**

It is the combination of more than one fields or columns of a table.

**EXTENDED E-R FEATURE**

Subclasses, Super classes & Inheritance.Sub groupings of entities that are meaningful.

**Ex:** EMPLOYEE may be further grouped into

SECRETARY, ENGINEER, MANAGER, TECHNICIAN, and so on.

Set of entities in each subgroups is a subset of the EMPLOYEE entity set.

Each is called a subclass of EMPLOYEE

EMPLOYEE is the superclass for each of these subclasses

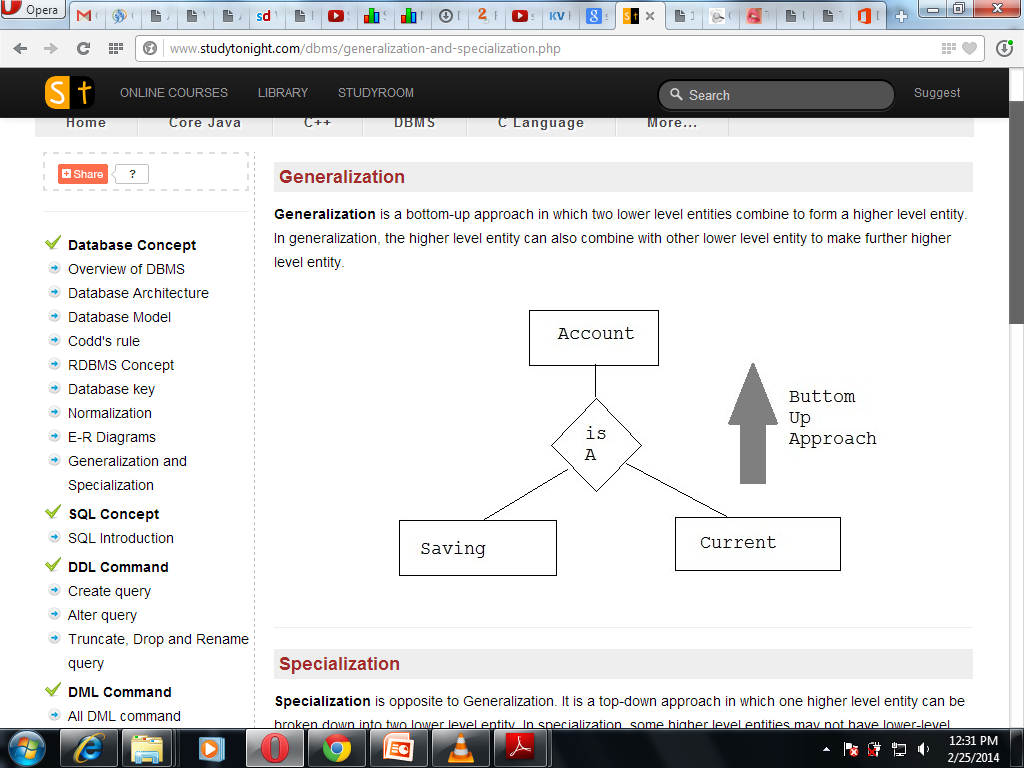
Subclass entity inherits all attributes and relationships of superclass

**Generalization**

**Bottom-up approach**

2 lower level entities combined to form a higher level entity

The higher level entity can also combine with other entity to make further higher level entity

****

**Generalization**

several entity types which have some common features into a single super class.Process of defining a generalized entity type from the given entity types.

**Specialization**

The process of designating subgroupings within an entity is called specialization.

eg;

The person allows to distinguish among person entities according to whether they correspond to employees or students: in general a person can be employee,student etc..

**Specialization**

Opposite to generalization

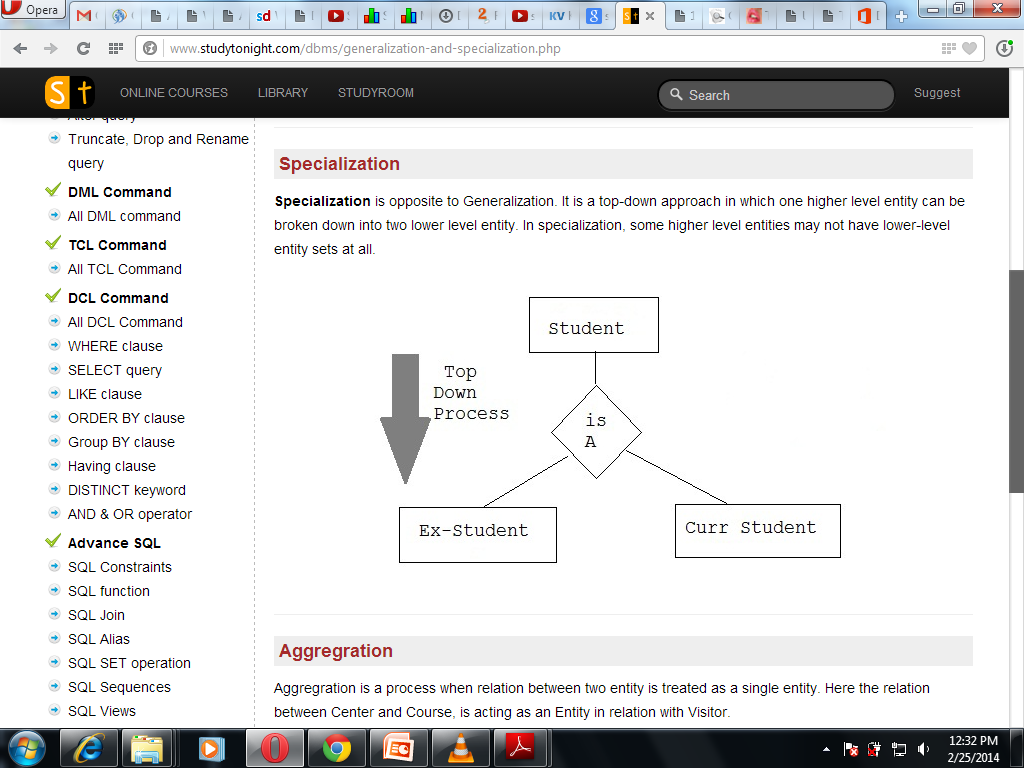
**Top-down approach**

Higher level entity can be broken down into 2 lower level entity

*Represented by a triangle*

*Labeled ISA*

Eg: Curr-Student is a Student



**Aggregation**

The E-R model cannot express relationships among relationships.Consider a DB with information about employees who work on a particular project and use a number of machines doing that work.

Relationship *work* and *uses* could be combined into a single set. we treat the relationship *work* and the entity sets *employee* and *project* as a higher-level entity set called *work*.

Process when relation between 2 entity is treated as a single entity.

Eg Relation between Center & Course is acting as an entity relation with visitor.

**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.

**Eg**: student and employee inherit the attributes of person.A lower-level entity set also inherits participation in the relationship sets in which its higher-level entity participates.

**Eg;** Suppose the person entity participates in a relationship person\_dept with department.then the student, employee,instructor and secretary entity sets implicitly participate in the person\_dept relationship with department.

**CONSTRAINTS ON GENERALIZATION**

One type of constraint involves determining which entities can be members of a given lower-level entity set. Such membership may be one of the following:

**Condition-defined :-** In this, lower-level entity sets, membership is evaluated on the basis of whether or not an entity satisfies an explicit condition or predicate.

**User-defined:-**User-defined lower-level entity sets are not constrained by a membership condition; rather, the database user assigns entities to a given entity set.

A second type of constraint relates to whether or not entities may belong to more than one lower-level entity set within a single generalization. The lower-level entity sets may be one of the following:

**Disjoint:-**A disjointness constraint requires that an entity belong to no more than one lower-level entity set.

In our example, an account entity can satisfy only one condition for the account-type attribute; an entity can be either a savings account or a checking account, but cannot be both.

**Overlapping:-**In overlapping generalizations, the same entity may belong to more than one lower-level entity set within a single generalization.

**Completeness constraint on a generalization** :it specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization. This constraint may be one of the following:

**Total generalization or specialization.** Each higher-level entity must belong to a lower-level entity set.

**Partial generalization or specialization.** Some higher-level entities may not belong to any lower-level entity set.