**United Group of Institutions**

**Computer Science & Engineering Department**

**Database Management System (KCS-501)**

**Year/Semester- 3rd /5th**

**Session- 2020-21**

**Unit-1 [Notes]**



**Database Management System**

**Unit-I**

**An overview of database management system**

Data: By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people.

Database: A database is a collection of related data. This is a collection of related data with an implicit meaning and hence is a database ; for example, we may consider the collection of words that make up this page of text to be related data and hence to constitute a database.

Properties of Database: A database has the following implicit properties:

• A database represents some aspect of the real world, sometimes called the miniworld or the universe of discourse (DoD). Changes to the miniworld are reflected in the database.

• A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.

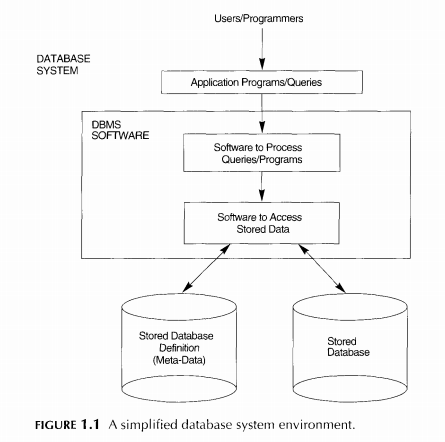
• A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

In other words, a database has some source from which data is derived, some degree of interaction with events in the real world, and an audience that is actively interested in the contents of the database.

Database Management System: A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is hence a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications. Defining a database involves specifying the data types, structures, and constraints for the data to be stored in the database. Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS. Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data. Sharing a database allows multiple users and programs to access the database concurrently.

Other important functions provided by the DBMS include protecting the database and maintaining it over a long period of time. Protection includes both system protection against hardware or software malfunction (or crashes), and security protection against unauthorized or malicious access. A typical large database may have a life cycle of many years.

Database System: We call the database and DBMS software together a database system.



Database System Vs File System

1. Self-Describing Nature of a Database System: A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs.

2. Insulation between Programs and Data, and Data Abstraction: In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access this file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property program-data independence.

3. Support of Multiple Views of the Data: A database typically has many users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. Some users may not need to be aware of whether the data they refer to is stored.

4. Sharing of Data and Multiuser Transaction Processing: A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.

5. Controlling Redundancy: In traditional software development utilizing file processing, every user group maintains its own files for handling its data-processing applications. This redundancy in storing the same data multiple times leads to several problems.

In the database approach, the views of different user groups are integrated during database design. Ideally, we should have a database design that stores each logical data item-such as a student's name or birth date-in only one place in the database.

Database System Concepts and Architecture

**Data Models:** A data model-a collection of concepts that can be used to describe the structure of a database-provides the necessary means to achieve this abstraction. By structure of a database, we mean the data types, relationships, and constraints that should hold for the data. Most data models also include a set of basic operations for specifying retrievals and updates on the database.

Categories of Data Models: Many data models have been proposed, which we can categorize according to the types of concepts they use to describe the database structure.

1. Hierarchical data model: The data is stored hierarchically using a downward tree. This model uses pointers to navigate between stored data. It was the first DBMS model.

Advantages: 1.Simplicity

2.Data security

3.Data Integrity

4.Efficiency

2. Network Model: Conference on data system languages(CODASYL) has defined the network model. The network model was created to represent complex data relationship more effectively to improve database performance. In the network model the user perceives the network database as a collection of records .A record may have more than one parents in contrast to hierarchical model. A relationship is called a set. Each set is composed of at least two record types, an owner record and a member record. A set represents a 1:M relationship between the owner and the member.

3. Entity Relationship model: The basic object that the ER model represents is an entity, which is a "thing" in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or r it may be an object with a conceptual existence (for example, a company, a job, or a university course). Each entity has attributes-the particular properties that describe it. For example, an employee entity may be described by the employee's name, age, address, salary, and job. A particular entity will have a value for each of its attributes. The attribute values that describe each entity become a major part of the data stored in the database.

4: Relational data model: The relational model represents the database as a collection of relations. Informally, each relation resembles a table of values or, to some extent, a "flat" file. When a relation is thought of as a table of values, each row in the table represents a collection of related data values. In the formal relational model terminology, a row is called a tuple, a column header is called an attribute, and the table is called a relation.

Schemas, Instances, and Database State

**Schema:** In any data model, it is important to distinguish between the description of the database and the database itself. The description of a database is called the database schema, which is specified during database design and is not expected to change frequently.

Instance: The actual data in a database may change quite frequently. The data in the database at a particular moment in time is called a database sta. It is also called the current set of occurrences or instances in the database.

Data Independence: The structure of data files stored in the DBMS catalog separated from the access programs. We call this property program-data independence. In DBMS data independence is achieved through three schema architecture which can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

1. Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs.

2. Physical data independence is the capacity to change the internal schema without having to change the conceptual schema.

The three-schema architecture can make it easier to achieve true data independence, both physical and logical.

The Three-Schema Architecture: The goal of the three-schema architecture is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels:

1. The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

2. The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.

3. The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group.

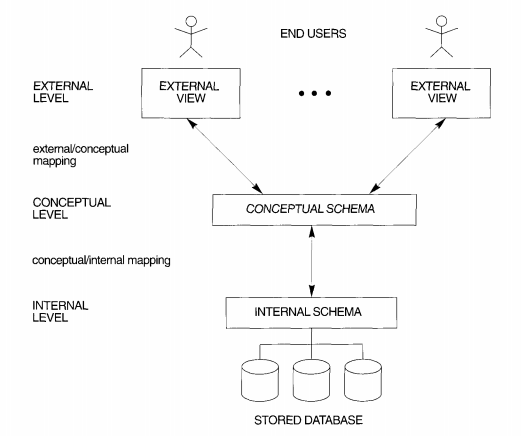


Fig: the three schema architecture

DATABASE LANGUAGES AND INTERFACES

The DBMS must provide e appropriate languages and interfaces for each category of users.

DBMS Languages:

1.Data definition language(DDL): Data definition language is used to specify both conceptual and internal schema where there is no strict separation between levels. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog. In DBMSs where a clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only.

Data Manipulation Language (DML): Once the database schemas arc compiled and the database is populated with data, users must have some means to manipulate the database. Typical manipulations include retrieval, insertion, deletion, and modification of the data. The DBMS provides a set of operations or a language called the data manipulation language (OML) for these purposes.

**SQL**

**Characteristics of SQL**

1. SQL is easy to learn.
2. SQL is used to access data from relational database management systems.
3. SQL can execute queries against the database.
4. SQL is used to describe the data.
5. SQL is used to define the data in the database and manipulate it when needed.
6. SQL is used to create and drop the database and table.
7. SQL is used to create view, stored procedure, function in a database.
8. SQL allows users to set permissions on tables, procedures and views.

**Advantages of SQL**

There are the following advantages of SQL:

High speed-Using the SQL queries, the user can quickly and efficiently retrieve large amount of records from a database.

No coding needed-In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

Well defined standards-Long established are used by the SQL databases that is being used by ISO and ANSI.

Portability-SQL can be used in laptop, PCs, server and even some of mobile phones.

Interactive language-SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

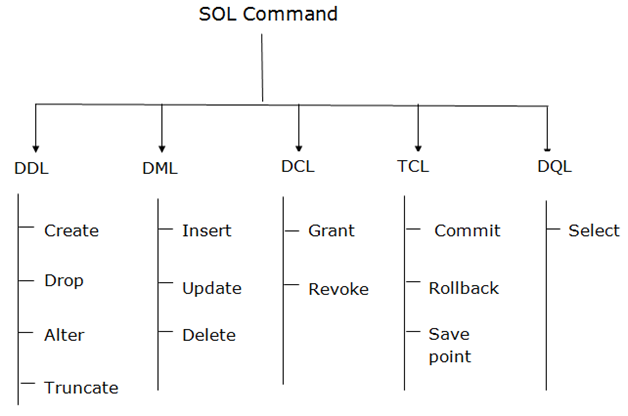
Multiple data view-Using the SQL language, the users can make different views of database structure.

|  |
| --- |
|  |

# SQL command

* SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions and queries of data.
* SQL can perform various tasks like: create table, add data to tables, drop the table, modify the table, set permission for users.

## Types of SQL Command:

### 1. Data definition language (DDL)

* DDL changes the structure of the table like: creating a table, deleting a table, altering a table, etc.
* All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

* CREATE
* ALTER
* DROP
* TRUNCATE

**a. CREATE** It is used to create new table in the database.

**Syntax:**

CREATE TABLE TABLE\_NAME (COLUMN\_NAME DATATYPES[,....]);

**Example:**

CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

**b. DROP:** It is used to delete both the structure and record stored in the table.

**Syntax**

DROP TABLE ;

**Example**

DROP TABLE EMPLOYEE;

**c. ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

**Syntax:**

To add new column in table

1. ALTER TABLE table\_name ADD column\_name COLUMN-definition;

To modify existing column in table

1. ALTER TABLE MODIFY(COLUMN DEFINITION....);

**EXAMPLE**

1. ALTER TABLE STU\_DETAILS ADD(ADDRESS VARCHAR2(20));
2. ALTER TABLE STU\_DETAILS MODIFY (NAME VARCHAR2(20));

**d. TRUNCATE:** it is used to delete all the rows from the table and free the space containing the table.

**Syntax:** TRUNCATE TABLE table\_name;

**Example:** TRUNCATE TABLE EMPLOYEE;

### 2. Data Manipulation Language

* DML commands are used to modify the database. It is responsible for all form of changes in the database.
* The command of DML are not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

* INSERT
* UPDATE
* DELETE

**a. INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.

**Syntax:**

1. INSERT INTO TABLE\_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, .... valueN);

Or

1. INSERT INTO TABLE\_NAME
2. VALUES (value1, value2, value3, .... valueN);

**For example:**

1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");

**b. UPDATE:** This command is used to update or modify the value of column in the table.

**Syntax:**

1. UPDATE table\_name SET [column\_name1= value1,...column\_nameN = valueN] [WHERE CONDITION]

**For example:**

1. UPDATE students
2. SET User\_Name = 'Sonoo'
3. WHERE Student\_Id = '3'

**c. DELETE:** It is used to remove one or more row from a table.

**Syntax:**

1. DELETE FROM table\_name [WHERE condition];

**For example:**

1. DELETE FROM javatpoint
2. WHERE Author="Sonoo";

### 3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

* Grant
* Revoke

**a. Grant:** It is used to give user access privileges to a database.

**Example**

1. GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_USER, ANOTHER\_USER;

**b. Revoke:** It is used to take back permissions from the user.

**Example**

1. REVOKE SELECT, UPDATE ON MY\_TABLE FROM USER1, USER2;

### 4. Transaction Control Language

TCL commands can only use with DML commands like: INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

* COMMIT
* ROLLBACK
* SAVEPOINT

**a. Commit:** Commit command is used to save all the transactions to the database.

**Syntax:**

1. COMMIT;

**Example:**

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;

**b. Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.

**Syntax:**

1. ROLLBACK;

**Example:**

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;

**c. SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax:**

1. SAVEPOINT SAVEPOINT\_NAME;

### 5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

* SELECT

**a. SELECT:** This is same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

**Syntax:**

1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

**For example:**

1. SELECT emp\_name
2. FROM employee
3. WHERE age > 20;

Example: Case Study

CREATE TABLE Students ( sid CHAR(20), name CHAR(30), login CHAR(20), age INTEGER, gpa REAL )

Tuples are inserted using the INSERT command. We can insert a single tuple into the Students table as follows:

INSERT INTO Students (sid, name, login, age, gpa) VALUES (53688, ‘Smith’, ‘smith@ee’, 18, 3.2)

We can delete tuples using the DELETE command. We can delete all Students tuples with name equal to Smith using the command:

DELETE FROM Students S WHERE S.name = ‘Smith’

We can modify the column values in an existing row using the UPDATE command. For example, we can increment the age and decrement the gpa of the student with sid 53688:

UPDATE Students S SET S.age = S.age + 1, S.gpa = S.gpa - 1 WHERE S.sid = 53688

Integrity Constraints Over Relations

An integrity constraint (IC) is a condition that is speciﬁed on a database schema, and restricts the data that can bstored in an instance of the database. If a database instance satisﬁes all the integrity constraints speciﬁed on the database schema, it is a legal instance. A DBMS enforces integrity constraints, in that it permits only legal instances to be stored in the database.

**Key Constraints**

Consider the Students relation and the constraint that no two students have the same student id. This IC is an example of a key constraint. A key constraint is a statement that a certain minimal subset of the ﬁelds of a relation is a unique identiﬁer for a tuple. A set of ﬁelds that uniquely identiﬁes a tuple according to a key constraint is called a **candidate key** for the relation; we often abbreviate this to just key.In the case of the Students relation, the (set of ﬁelds containing just the) sid ﬁeld is a candidate key.

There are two parts to the deﬁnition of (candidate) key:

1. Two distinct tuples in a legal instance (an instance that satisﬁes all ICs, including the key constraint) cannot have identical values in all the ﬁelds of a key.
2. No subset of the set of ﬁelds in a key is a unique identiﬁer for a tuple.

The ﬁrst part of the deﬁnition means that in any legal instance, the values in the key ﬁelds uniquely identify a tuple in the instance

The second part of the deﬁnition means, for example, that the set of ﬁelds {sid, name} is not a key for Students, because this set properly contains the key {sid}.Theset {sid, name} is an example of a **superkey**, which is a set of ﬁelds that contains a key.

Out of all the available candidate keys, a database designer can identify a **primary** key. Intuitively, a tuple can be referred to from elsewhere in the database by storing the values of its primary key ﬁelds. For example, we can refer to a Students tuple by storing its sid value.

**Specifying Key Constraints in SQL**

CREATE TABLE Students ( sid CHAR(20), name CHAR(30), login CHAR(20), age INTEGER, gpa REAL, UNIQUE (name, age), CONSTRAINT StudentsKey PRIMARY KEY

(sid) )

This deﬁnition says that sid is the primary key and that the combination of name and age is also a key. The deﬁnition of the primary key also illustrates how we can name a constraint by preceding it with CONSTRAINT constraint-name. If the constraint is violated, the constraint name is returned and can be used to identify the error.

**Foreign Key Constraints**

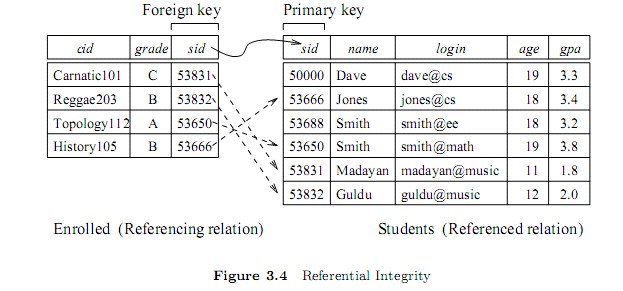
Sometimes the information stored in a relation is linked to the information stored in another relation. If one of the relations is modiﬁed, the other must be checked, and perhaps modiﬁed, to keep the data consistent. An IC involving both relations must be speciﬁed if a DBMS is to make such checks. The most common IC involving two relations is a foreign key constraint.

Suppose that in addition to Students, we have a second relation:

Enrolled(sid: string, cid: string, grade: string)

To ensure that only bonaﬁde students can enroll in courses, any value that appears in the sid ﬁeld of an instance of the Enrolled relation should also appear in the sid ﬁeld of some tuple in the

Students relation. The sid ﬁeld of Enrolled is called a **foreign key** and **refers** to Students. The foreign key in the referencing relation (Enrolled, in our example) must match the primary key of the referenced relation (Students), i.e., it must have the same number of columns and compatible data types, although the column names can be diﬀerent.



**Specifying Foreign Key Constraints in SQL**

CREATE TABLE Enrolled ( sid CHAR(20), cid CHAR(20), grade CHAR(10), PRIMARY KEY

(sid, cid), FOREIGN KEY (sid) REFERENCES Students )

**Enforcing Integrity Constraints**

Consider the instance S1 of Students shown in Figure 3.1. The following insertion violates the primary key constraint because there is already a tuple with the sid 53688, and it will be rejected by the DBMS:

INSERT INTO Students (sid, name, login, age, gpa) VALUES (53688, ‘Mike’, ‘mike@ee’, 17, 3.4)

The following insertion violates the constraint that the primary key cannot contain null:

INSERT INTO Students (sid, name, login, age, gpa) VALUES (null, ‘Mike’, ‘mike@ee’, 17, 3.4)

**Querying Relational Data**

A **relational database query** is a question about the data, and the answer consists of a new relation containing the result. For example, we might want to ﬁnd all students younger than 18 or all students enrolled in Reggae203.

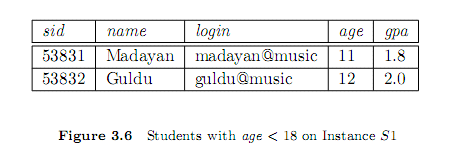
A **query language** is a specialized language for writing queries.

SQL is the most popular commercial query language for a relational DBMS. Consider the instance of the Students relation shown in Figure 3.1. We can retrieve rows corresponding to students who are younger than 18 with the following SQL query:

SELECT \* FROM Students S WHERE S.age < 18

The symbol \* means that we retain all ﬁelds of selected tuples in the result. The condition S.age

< 18 in the WHERE clause speciﬁes that we want to select only tuples in which the age ﬁeld has a value less than 18. This query evaluates to the relation shown in Figure 3.6.



Overall database structure

Figure below illustrates the typical DBMS components. The database and the DBMS catalog are usually stored on disk. Access to the disk is controlled primarily by the operating system (OS), which schedules disk input/output. A higher-level stored data manager module of the DBMS controls access to DBMS information that is stored on disk, whether it is part of the database or the catalog. The dotted lines and circles marked A, B,C, D, and E in Figure illustrate accesses that are under the control of this stored data manager. The stored data manager may use basic os services for carrying out low level data transfer between the disk and computer main storage, but it controls other aspects of data transfer, such as handling buffers in main memory. Once the data is in main memory buffers, it can be processed by other DBMS modules, as well as by application programs. Some DBMSs have their own buffer manager module, while others use the os for handling the buffering of disk pages.

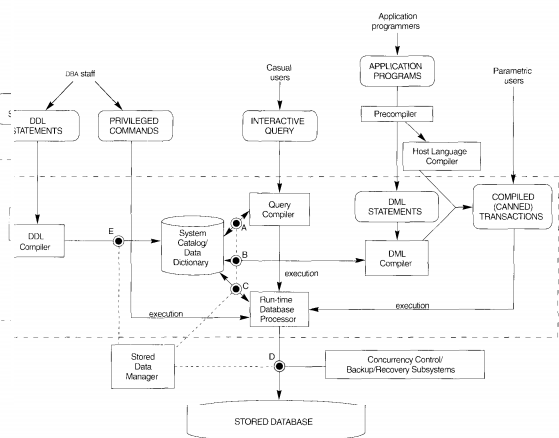


Fig: components of DBMS

Data Modeling Using the Entity-Relationship Model

**ER model concepts**

Entities and Attributes: The basic object that the ER model represents is an entity, which is a "thing" in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for example, a company, a job, or a university course).

Each entity has attributes-the particular properties that describe it. For example, an employee entity may be described by the employee's name, age, address, salary, and job.

Types of attribute

Composite versus Simple (Atomic) Attributes. Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent . For example, the Address attribute of the employee entity can be subdivided into Street Address, City, State.

Attributes that are not divisible are called simple or atomic attribute.

Single-Valued versus Multivalued Attributes. Most attributes have a single value for a particular entity; such attributes are called single-valued. For example, Age is a single-valued attribute of a person. . In some cases an attribute can have a set of values for the same entity-for example, a Colors attribute for a car, or a College Degrees attribute for a person.

Stored versus Derived Attributes. In some cases, two (or more) attribute values are related-for example, the Age and BirthDate attributes of a person. For a particular person entity, the value of Age can be determined from the current (today's) date and the value of that person's BirthDate. The Age attribute is hence called a derived attribute and is said to be derivable from the BirthDate attribute, which is called a stored attribute. Some attribute values can be derived from related entities; for example, an attribute NumberOfEmployees of a department entity can be derived by counting the number of employees related to (working for) that department.

Entity Types, Entity Sets, Keys, and Value Sets

Entity Type: An entity type defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes. For example employ entity type, company entity type.

Entity set: The collection of all entities of a particular entity type in the database at any point in time is called an entity set; the entity set is usually referred to using the same name as the entity type. For example, EMPLOYEE refers to both a type of entity as well as the current set of all employee entities in the database.

Key Attributes of an Entity Type: An entity type usually has an attribute whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute, and its values can be used to identify each entity uniquely. For example, the Name attribute is a key of the COMPANY entity type because no two companies are allowed to have the same name.

Weak Entity and Strong Entity: An entity type may also have no key, in which case it is called a weak entity type . An entity type that has key attribute is called strong entity type.

Value Sets (Domains) of Attributes: Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity. Value sets are typically specified using the basic data types available in most programming languages, such as integer, string, boolean, float, enumerated type, subrange, and so on. Additional data types to represent date, time, and other concepts are also employed.

RELATIONSHIP TYPES, RELATIONSHIP SETS, ROLES, AND STRUCTURAL CONSTRAINTS

There are several implicit relationships among the various entity types. In fact, whenever an attribute of one entity type refers to another entity type, some relationship exists. For example, the attribute Manager of DEPARTMENT refers to an employee who manages the department; the attribute ControllingDepartment of PROJECT refers to the department that controls the project; the attribute Supervisor of EMPLOYEE refers to another employee (the one who supervises this employee); the attribute Department of EMPLOYEE refers to the department for which the employee works; and so on. In the ER model, these references should not be represented as attributes but as relationships.

Degree of a Relationship Type:

The degree of a relationship type is the number of participating entity types. Hence, the WORKSFOR relationship is of degree two. A relationship type of degree two is called binary, and one of degree three is called ternary. . Relationships can generally be of any degree, but the ones most common are binary relationships.

Constraints on Relationship Types: Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. These constraints are determined from the miniworld situation that the relationships represent. For example, if the company has a rule that each employee must work for exactly one department, then we would like to describe this constraint in the schema. We can distinguish two main types of relationship constraints: **cardinality ratio and participation.**

Cardinality Ratios for Binary Relationships: The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in. Cardinality ratio may be one to one, one to many, many to one, many to many. For example, in the WORKS\_FOR binary relationship type, DEPARTMENT: EMPLOYEE is of cardinality ratio one to many meaning that each department can be related to (that is, employs) any number of employees but an employee can be related to (work for) only one department.

An example of a one to one binary relationship is MANAGES which relates a department entity to the employee who manages that department. This represents the miniworld constraints that-at any point in time-an employee can manage only one department and a department has only one manager.

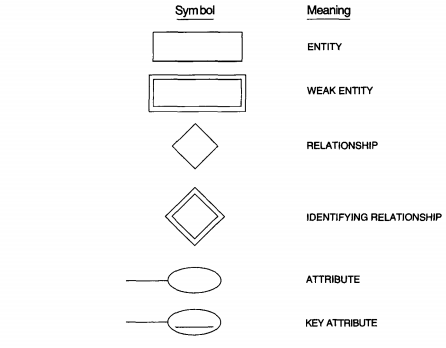
Participation Constraints: The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in, and is sometimes called the minimum cardinality constraint. There are two types of participation constraints**- total and** **partial.**

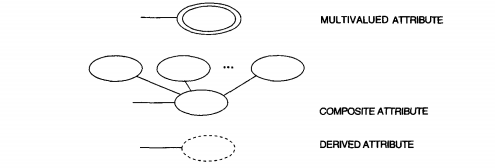
Total Participation: If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS\_ FOR relationship instance Thus, the participation of EMPLOYEE in WORKS\_FOR is called total participation, meaning that every entity in "the total set" of employee entities must be related to a department entity via WORKS\_FOR.

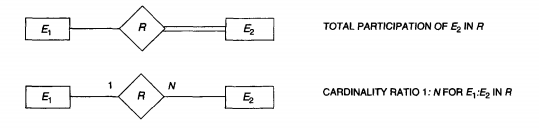
Partial Participation: we do not expect every employee to manage a department, so the participation of EMPLOYEE in the MANAGES relationship type is partial, meaning that some or "part of the set of" employee entities are related to some department entity via MANAGES, but not necessarily all.

Identifying Relationship: Entity types that do not have key attributes of their own are called **weak entity types**. In contrast, strong entity types that do have a key attribute-which include all the examples we discussed so far-are called strong entity types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other entity type the identifying or owner entity type, and we call the relationship type that relates a weak entity type to its owner the identifying relationship of the weak entity type. A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship, because a weak entity cannot be identified without an owner entity type.

For example Consider the entity type DEPENDENT, related to EMPLOYEE, which is used to keep track of the dependents of each employee via a one to many relationship. The attributes of DEPENDENT are Name (the first name of the dependent), BirthDate, Sex, and Relationship (to the employee). Two dependents of two distinct employees may, by chance, have the same values for Name, BirthDate, Sex, and Relationship, but they are still distinct entities. They are identified as distinct entities only after determining the particular employee entity to which each dependent is related. Each employee entity is said to own the dependent entities that are related to it.

Notation for ER Diagram





Concept of Key

A key is a set of attributes whose values uniquely identify an entity in the set. For example consider the entity set **student with attributes name, roll no and address.** To identify individual student entity the set of attributes is called key.

Super Key: A super key is an attribute or set of attributes used to identify the records uniquely in a relation. Consider a relation **PERSON**

|  |  |  |
| --- | --- | --- |
| **ID** | **NAME** | **AGE** |
| 1 | AB | 17 |
| 2 | XY | 17 |
| 3 | PQ | 18 |

In the above example ID is a super key since ID is unique for each person. Similarly (ID,AGE) and (ID,NAME) are also super key since their combinations is also unique for each person.

**Candidate Key:** Minimal set of super key is called candidate key that is candidate key is minimal set of attributes. In the above example (ID) is the minimal set containing only one attribute therefore ID is the candidate key.

**Primary Key:** A relation can have more than one candidate key and one of them can be chosen as a primary key. Primary key cannot be duplicate. If we add additional attributes to a primary key the resulting combination would still uniquely identify an entity in the entity set and set becomes super key.

**Alternate key:** A candidate key which is not the primary key is called alternate key.

**Foreign Key:** Sometimes information stored in a relation is linked to the information stored in another relation. If one of the relation is modified the other must be checked and perhaps modified to keep the data consistent. **A foreign key is an attribute whose values are drawn from the primary key values in the related table. F**oreign key is the same attribute in one relation and that same attribute is primary key in another relation. For example consider the relation EMPLOYEE

|  |  |  |
| --- | --- | --- |
| ID | NAME | DEPTNO |
| 1 | ABC | 14 |
| 2 | XYZ | 14 |
| 3 | PQR | 12 |

ID is the primary key.

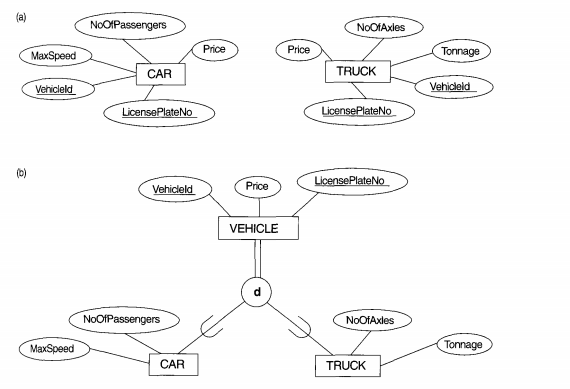
The other relation is DEPARTMENT

|  |  |
| --- | --- |
| DEPTNAME | DEPTNO |
| CS | 12 |
| IT | 14 |

DEPTNO is primary key in DEPARTMENT relation but DEPTNO is foreign key in EMPLOYEE relation. Both the relations are linked through primary key and foreign key and also information stored in both relations are linked.

Composite Key: It is combination of two or more columns in a table that can be used to uniquely identify each row in a table but individual columns can not uniquely identify.

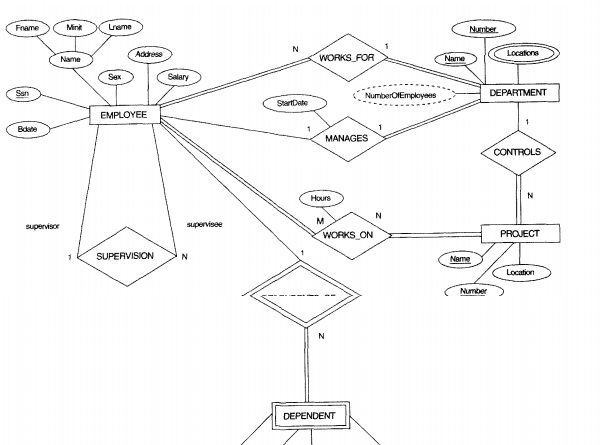
Generalization: We use the term generalization to refer to the process of defining a generalized entity type from the given entity types. For example, consider the entity types CAR and TRUCK .Because they have several common attributes, they can be generalized into the entity type VEHICLE

.

Aggregation: One limitation of the ER diagram is that they do not allow representation of relationship among relationship. In such a case the relationship along with its entities aggregated to form an aggregate entity which can be used for expressing the required relationships. Aggregation is an abstraction through which relationships are treated as higher level entities.

Aggregation allows us to indicate that a relationship set participates in another relationship set.

Reduction of an ER diagram to tables: A database represented by ER diagram can be represented by a collection of tables. For step by step procedure we consider an ER diagram of a company database which is shown below:



After identifying strong entity set, weak entity set, simple attribute, composite attribute, multivalued attribute, derived attribute, relationships, identifying relationships following steps are followed:

**Step:1 Strong Entity Set:** For each regular (strong) entity type E in the ER schema, create a table R that includes all the simple attributes of E. Include only the simple component attributes of a composite attribute. Choose one of the key attributes of E as primary key for R. If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of database.

In our example, we create the tables EMPLOYEE, DEPARTMENT, and PROJECT to correspond to the strong entity types EMPLOYEE, DEPARTMENT, and PROJ ECT.





Step 2: Weak Entity Set: For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R. In addition, include as foreign key attributes of R the primary key attributes) of the relations) that correspond to the owner entity types): this takes care of the identifying relationship type of W The primary key of R is the combination of the primary keys) of the owners) and the partial key of the weak entity type W, if any.

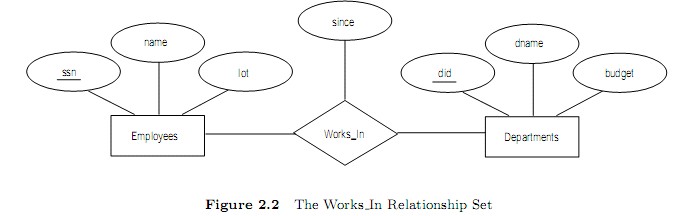
If there is a weak entity type E2 whose owner is also a weak entity type E1, then E] should be mapped before E2 to determine its primary key first.

In our example, we create the relation DEPENDENT in this step to correspond to the weak entity type DEPENDENT. We include the primary key SSN of the EMPLOYEE relation-which corresponds to the owner entity type-as a foreign key attribute of DEPENDENT; we renamed it ESSN, although this is not necessary. The primary key of the DEPENDENT relation is the combination {ESSN, DEPENDENT\_NAME} because DEPENDENT\_NAME is the partial key of DEPENDENT.



Step 3:Relationship Set: Let R be a relationship set, let a, b, c, d be the set of attributes formed by union of the primary keys of each of the entity sets participating in R, and let the descriptive attributes(if any) of R be p,q, r, s. We represent this relationship set by a table called with one column for each attribute of the set:

{a,b,c,d,p,q,r,s}



In our example relationship set are WORKS-FOR, WORKS-ON, MANAGES and CONTROLS. They are represented in tables as follows:

1. If the relationship is many to one and participation of one entity is total then no table is required for relationship and table for relationship is merged with table of total participating entity. In WORKS-FOR relationship both entity EMPLOYEE and DEPARTMENT participate totally therefore table for WORKS-FOR can be merged with any entity.

2. Participation of DEPARTMENT entity in MANAGES relationship is total therefore table for manages will be merged in table for DEPARTMENT with one column more for descriptive attribute ‘startdate’ of MANAGES relationship.

3. Similarly above rules is applied for CONTROLS and WORKS-ON relationships.

**Tabular Representation of Identifying relationship Set:** Table corresponding to a relationship set linking a weak entity set to its identifying strong entity set does not need to be represented in tabular form.

In our example identifying relationship set DEPENDENTS-OF represented by double diamond between weak entity set DEPENDENT and strong entity set EMPLOYEE does not need to be present in tabular representation of ER diagram.

**Composite Attributes:** Composite attributes have no column in the table but for each component of composite attribute must have a separate column in the table of corresponding entity set.

In our example ER diagram of **company** the attribute **name** of EMPLOYEE entity is composite hence there will be no column for name attribute but there will be separate column for components FNAME,MINIT and LNAME as shown in tabular representation of EMPLOYEE above.

**Multivalued Attribute:** For a multivalued attribute M we create a table T with column C that corresponds to M and columns corresponding to primary key of entity set of which M is an attribute.

In our example attribute ‘location’ of entity set DEPERATMENT is multivalued. We do not create a column for location attribute in the table of DEPARTMENT rather we create a separate table LOCATION with columns ‘dlocation’ and other column for primary key attribute of DEPERATMENT that is for ’Name’ and ‘Number’.