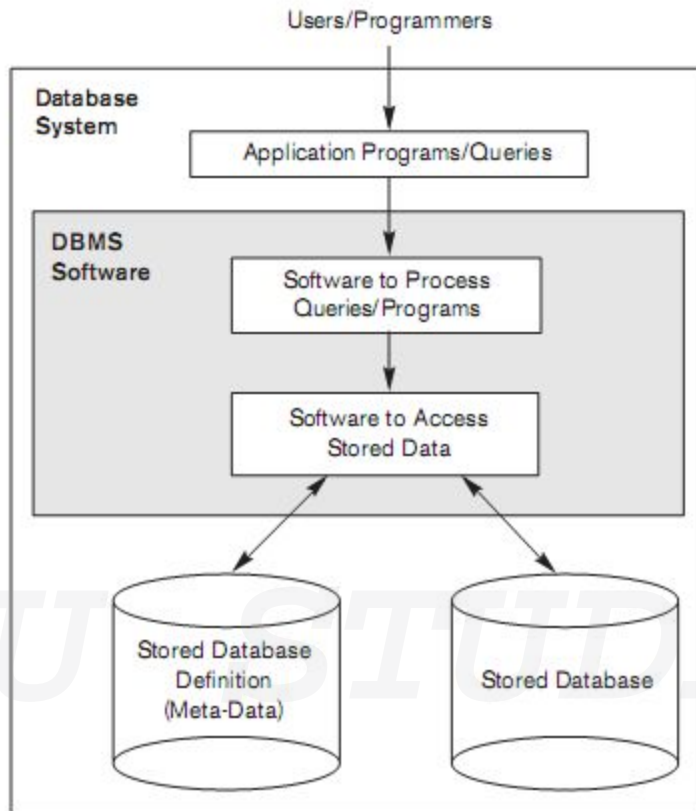


MODULE 1

- A **database** is a collection of related data. **Data** means, known facts that can be recorded and that have implicit meaning.
- A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database. The **DBMS** is 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 of the data to be stored in the data-base.
- **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS.
- **Manipulating** a database includes functions such 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 simultaneously.
- An **application program** accesses the database by sending queries or requests for data to the DBMS. A **query** typically causes some data to be retrieved; a **transaction** may cause some data to be read and some data to be written into the database.
- Other important functions provided by the DBMS include protecting the database and maintaining it over a long period of time.
- **Protection** includes 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, so the DBMS must be able to **maintain** the database system by allowing the system to evolve as requirements change over time.

Database System Environment



EXAMLE STUDENT DB

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Characteristics of the Database Approach

- In traditional **file processing**, each user defines and implements the files needed for a specific software application as part of programming the application.
- Redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.
- In the database approach, a single repository maintains data that is defined once and then accessed by various users
- The main characteristics of the database approach versus the file processing approach are the following:
 1. Self-describing nature of a database system
 2. Insulation between programs and data, and data abstraction
 3. Support of multiple views of the data
 4. Sharing of data and multiuser transaction processing

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.
- This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called **meta-data** and it describes

the structure of the primary database.

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 that file.
- 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. This property is called **program-data independence**
- An **operation** (also called a function or method) is specified in two parts.
 - Interface
 - ✓ The interface (or signature) of an operation includes the operation name and the data types of its arguments (or parameters).
 - Implementation
 - ✓ The implementation (or method) of the operation is specified separately and can be changed without affecting the interface.
- User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This is termed as **program-operation independence**
- The characteristic that allows program-data independence and program operation independence is called **data abstraction**
- A **data model** is a type of data abstraction that is used to provide conceptual representation. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented.

- The **data model** hides storage and implementation details that are not of interest to most database users.
-

3. Support of Multiple Views of the Data

- A database has many users, each user 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.
- A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.

4. Sharing of Data and Multiuser Transaction Processing

- A multiuser DBMS, 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.
- A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.
- A transaction is an executing program or process that includes one or more database accesses, such as reading or updating of database records.
- Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions.
- The DBMS must enforce several transaction properties.

1. Isolation property

- The **isolation** property ensures that each transaction

www.ktustudents.in

[For more study materials>www.ktustudents.in](http://www.ktustudents.in)

appears to execute in isolation from other transactions, even though hundreds of transactions may be executing concurrently.

2. Atomicity property

- The **atomicity** property ensures that either all the database operations in a transaction are executed or none are.

ACTORS ON THE SCENE

- The people whose jobs involve the day-to-day use of a large database are called as the **actors on the scene**

1. Database Administrators
2. Database Designers
3. End Users
4. System Analysts and Application Programmers (Software Engineers)

Database Administrators

- In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the **database administrator (DBA)**
- The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed. The DBA is accountable for problems such as security breaches and poor system response time.

Database Designers

- **Database designers** are responsible for identifying the data to be stored in the data-base and for choosing appropriate structures to represent

www.ktustudents.in

[For more study materials>www.ktustudents.in](http://www.ktustudents.in)

and store this data.

- It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements.
- Database designers typically interact with each potential group of users and develop **views** of the database that meet the data and processing requirements of these groups.
- Each view is then analyzed and integrated with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

End Users

- **End users** are the people whose jobs require access to the database for querying, updating, and generating reports.
- There are several categories of end users:

1.Casual end users

- **Casual end users** occasionally access the database, but they may need different information each time.
- They use a sophisticated database query language to specify their requests and are typically middle or high level managers or other occasional browsers.

2.Naive or parametric end users

- **Naive or parametric end users** make up a sizable portion of database endusers.
- Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates

called **canned transactions** that have been carefully programmed and tested.

- The tasks that such users perform are varied:

3.Sophisticated end users

- **Sophisticated end users** include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.

4.Standalone users

- **Standalone users** maintain personal databases by using ready made program packages that provide easy-to-use menu based or graphics based interfaces.
- An example is the user of a tax package that stores a variety of personal financial data for tax purposes.

System Analysts and Application Programmers (Software Engineers)

- **System analysts** determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements.
- **Application programmers** implement these specifications as programs, then they test, debug, document, and maintain these canned transactions. Such analysts and programmers commonly referred to as

software developers or software engineers

WORKERS BEHIND THE SCENE

➤ The people who work to maintain the database system environment but who are not actively interested in the database contents as part of their daily job are called as the **workers behind the scene**

1. DBMS system designers and implementers
2. Tool developers
3. Operators and maintenance personnel (system administration personnel)

DBMS system designers and implementers

- KTU STUDENTS
- DBMS system designers and implementers design and implement the DBMS modules and interfaces as a software package.
 - A DBMS is a very complex software system that consists of many components, or **modules**, including modules for implementing the catalog, query language processing, interface processing, accessing and buffering data, controlling concurrency, and handling data recovery and security.

Tool developers

- Tool developers design and implement tools , the software packages that facilitate database modeling and design, database system design, and improved performance.
- Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data

www.ktustudents.in

[For more study materials>www.ktustudents.in](http://www.ktustudents.in)

generation.

Operators and maintenance personnel (system administration personnel)

- Operators and maintenance personnel (system administration personnel) are responsible for the actual running and maintenance of the hardware and software environment for the database system.

ADVANTAGES OF USING THE DBMS

1. Controlling Redundancy

- **Redundancy** in storing the same data multiple times leads to several problems.
 - a) Duplication of effort
 - b) Storage space is wasted
 - c) Files that represent the same data may become inconsistent

2. Restricting Unauthorized Access

- A DBMS should provide a **security and authorization subsystem**, which the DBA uses to create accounts and to specify account restrictions.

3. Providing Persistent Storage for Program Objects

- Databases can be used to provide **persistent storage** for program objects and data structures.

4. Providing Storage Structures and Search Techniques for Efficient Query Processing

- Database systems must provide capabilities for *efficiently executing queries and updates*.
- The database is typically stored on disk, the DBMS must provide specialized data structures and search techniques to speed up

disk search for the desired records. Auxiliary files called **indexes** are used for this purpose

5. Providing Backup and Recovery

- A DBMS must provide facilities for recovering from hardware or software failures. The **backup and recovery subsystem** of the DBMS is responsible for recovery.
- For example, if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing.

6. Providing Multiple User Interfaces

- Users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces.
- These include query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users, and menu-driven interfaces and natural language interfaces for standalone users.
- Both forms-style interfaces and menu-driven interfaces are commonly known as **graphical user interfaces (GUIs)**

7. Representing Complex Relationships among Data

- A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.

8. Enforcing Integrity Constraints

- Integrity constraints are used to ensure accuracy and consistency of data in DB.
- A DBMS should provide capabilities for defining and enforcing these constraints. The simplest type of integrity constraint involves

specifying a data type for each data item.

9. Permitting Inferencing and Actions Using Rules

- Some database systems provide capabilities for defining deduction rules for inferencing new information from the stored database facts. Such systems are called **deductive database systems**

10. Additional Implications of Using the Database Approach

- a) Potential for Enforcing Standards.
- b) Reduced Application Development Time
- c) Flexibility.
- d) Availability of Up-to-Date Information
- e) Economies of Scale.

DATABASE SYSTEM CONCEPTS AND ARCHITECTURE

- One fundamental characteristic of the database approach is that it provides some level of data abstraction. **Data abstraction** refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data.
- A **data model** is a collection of concepts that can be used to describe the structure of a database provides the necessary means to achieve this abstraction.

Categories of Data Models

1.High-level or conceptual data models provide concepts that are close to the way many users perceive data.

2.Low-level or physical data models provide concepts that describe the details of how data is stored on the computer storage media.

www.ktustudents.in

[For more study materials>www.ktustudents.in](http://www.ktustudents.in)

3. Representational (or implementation) data models, which provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage.

- Conceptual data models use concepts such as entities, attributes, and relationships.
- An **entity** represents a real-world object or concept, such as an employee or a project from the mini world that is described in the database.
- An **attribute** represents some property of interest that further describes an entity, such as the employee's name or salary.
- A **relationship** among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project.
- Representational or implementation data models are the models used most frequently in traditional commercial DBMSs. They are
 1. relational data model,
 2. network data model
 3. hierarchical data model

Schemas, Instances, and Database State

- The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently.
- A displayed schema is called a **schema diagram**

STUDENT

Name	Student_number	Class	Major
------	----------------	-------	-------

COURSE

Course_name	Course_number	Credit_hours	Department
-------------	---------------	--------------	------------

PREREQUISITE

Course_number	Prerequisite_number
---------------	---------------------

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
--------------------	---------------	----------	------	------------

GRADE_REPORT

Student_number	Section_identifier	Grade
----------------	--------------------	-------

- A schema diagram displays only *some aspects* of a schema, such as the names of record types and data items, and some types of constraints.
- 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 state** or **snapshot**. It is also called the *current* set of **occurrences** or **instances** in the database. Each schema construct has its own *current set* of instances.

The difference between database schema and database state

- When we **define** a new database the corresponding database state is the *empty state* with no data.
- We get the *initial state* of the database when the database is first **populated** or **loaded** with the initial data.
- At any point in time, the database has *acurrent state*
- A **valid state** is, a state that satisfies the structure and constraints specified in the schema.
- The DBMS stores the descriptions of the schema constructs and

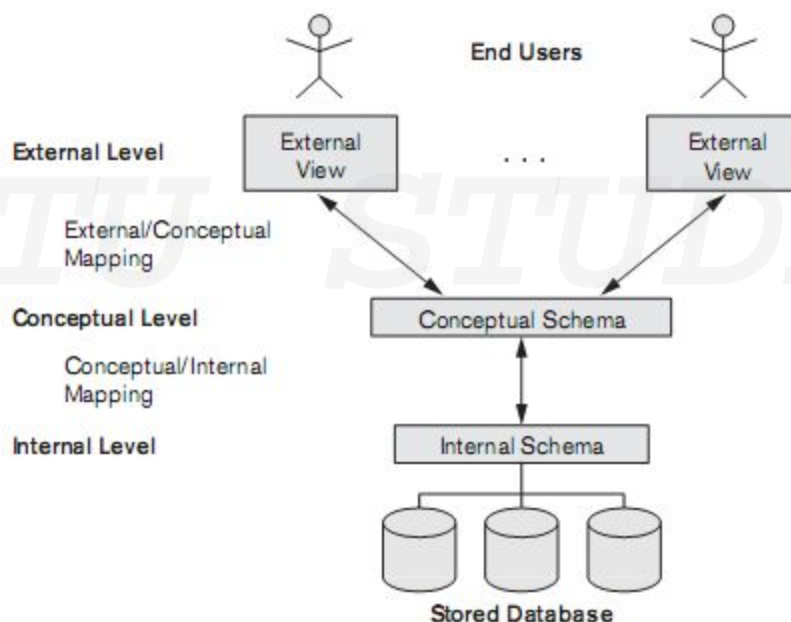
constraints also called the **meta-data** in the DBMS catalog.

- The schema is not supposed to change frequently ,**but** it is not uncommon that changes occasionally need to be applied to the schema as the application requirements change. It is called as **schema evolution**.

Three-Schema Architecture and Data Independence

The Three-Schema Architecture

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



1. Internal level

- 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. Conceptual level

- 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. External or view level

- 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.
- The processes of transforming requests and results between levels are called **mappings**.

DATA INDEPENDENCE

- The capacity to change the schema at one level of a database system without having to change the schema at the next higher level.
- Two types of data independence:

1. Logical data independence

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

2. Physical data independence

Physical data independence is the capacity to change the internal schema

without having to change the conceptual schema.

- Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the *mapping* between the two levels is changed.

Database Languages and Interfaces

DBMS Languages

Data definition language (DDL), is used by the DBA and by database designers to define conceptual and internal schemas schemas.

Storage definition language (SDL), is used to specify the internal schema.

View definition language (VDL), to specify user views and their mappings to the conceptual schema.

Data manipulation language (DML) is used for retrieval, insertion, deletion, and modification of the data.

- SQL relational database language which represents a combination of DDL, VDL, and DML, as well as statements for constraint specification, schema evolution, and other features.
- There are two main types of DMLs.
 1. **high-level** or **nonprocedural** DML can be used on its own to specify complex database operations concisely.
 2. **low-level** or **procedural** DML *must* be embedded in a general-purpose programming language. This type of DML typically retrieves individual records or objects from the database and processes each separately. Therefore, it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records. Low-level DMLs are also called **record-at-a-time** DMLs

DBMS Interfaces

- User-friendly interfaces provided by a DBMS may include the following:
 1. Menu-Based Interfaces for Web Clients or Browsing.
 2. Forms-Based Interfaces.
 3. Graphical User Interfaces (GUI)
 4. Natural Language Interfaces
 5. Speech Input and Output
 6. Interfaces for Parametric Users.

DATA MODELING USING THE ENTITY-RELATIONSHIP (ER) MODEL

- **Entity-Relationship (ER) model**, which is a popular high-level conceptual data model, used for the conceptual design of database applications, and many database design tools employ its concepts.
- The diagrammatic notation associated with the ER model, known as **ER diagrams**.

Entity, Types, Entity Sets, Attributes and Keys

- 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.
- Each entity has **attributes**, the particular properties that describe it.

Composite versus Simple (Atomic) Attributes

- **Composite attributes** can be divided into smaller subparts, which represent more basic attributes with independent meanings.

- Attributes that are not divisible are called **simple** or **atomic attributes**

Single-Valued versus Multivalued Attributes

- 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.

- Attribute that can have different *numbers of values* for same attributes are called **multivalued**

Eg: College_degrees attribute

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 **stored attribute**

NULL Values

- In some cases, a particular entity may not have an applicable value for an attribute.

Complex Attributes

- Composite and multivalued attributes can be nested. }. Such attributes are called **complex attributes**

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.

Entity set

- The collection of all entities of a particular entity type in the data-base 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.
- An entity type describes the **schema** or **intension** for a *set of entities* that share the same structure. The collection of entities of a particular entity type is grouped into an entity set, which is also called the **extension** of the entity type.

Key Attributes of an Entity Type

- An important constraint on the entities of an entity type is the **key** or **uniqueness constraint** on attributes.
- An entity type usually has one or more attributes 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.

Composite key

- It must be *minimal*, that is, all component attributes must be included in the composite attribute to have the uniqueness property.

RELATIONSHIP TYPES, RELATIONSHIP SETS, ROLES, AND STRUCTURAL CONSTRAINTS

www.ktustudents.in

[For more study materials>www.ktustudents.in](http://www.ktustudents.in)

- A **relationship type** R among n entity types E_1, E_2, \dots, E_n defines a set of associations or a **relationship set** among entities from these entity types.
- As for the case of entity types and entity sets, a relationship type and its corresponding relationship set are customarily referred to by the *same name*, R . Mathematically, the relationship set R is a set of **relationship instances** r_i ,
- where each r_i associates n individual entities (e_1, e_2, \dots, e_n) , and each entity e_j in r_i is a member of entity set E_j , $1 \leq j \leq n$.
- Hence, a relationship set is a mathematical relation on E_1, E_2, \dots, E_n ; alternatively, it can be defined as a subset of the Cartesian product of the entity sets $E_1 \times E_2 \times \dots \times E_n$.
- Each of the entity types E_1, E_2, \dots, E_n is said to participate in the relationship type R ; similarly, each of the individual entities e_1, e_2, \dots, e_n is said to **participate** in the relationship instance

$$r_i = (e_1, e_2, \dots, e_n).$$

Degree of a Relationship Type

- The **degree** of a relationship type is the number of participating entity types.
- A relationship type of degree two is called **binary**, and one of degree three is called **ternary**.

Role name

- The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.
- The relationship types in which the *same* entity type participates more than once in a relationship type in *different roles* such relationship

types are called **recursive relationships**

Constraints on Binary Relationship Types

- Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set.
- Two main types of binary 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.
- The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.

Participation Constraints and Existence Dependencies

- 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, they are total and partial

1.Total participation

- Total participation is also called **existence dependency**

2. Partial participation

- **partial**, meaning that *some* or *part* of the set of employee entities are

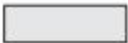










related to some department entity but not necessarily all.

- In ER diagrams, total participation (or existence dependency) is displayed as a *double line* connecting the participating entity type to the relationship,
- partial participation is represented by a *single line*
- The cardinality ratio and participation constraints, are together known as the **structural constraints** of a relationship type.

WEAK ENTITY TYPES

- Entity types that do not have key attributes of their own are called **weak entity types**.
- Weak entity are identified by being related to specific entities from another entity type in combination with one of their attribute values.
- This other entity type is called the **identifying** or **owner entity type**,
- The relationship type that relates a weak entity type to its owner is called 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 can not be identified without an owner entity.
- A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are *related to the same owner entity*.

Design Choices for ER Conceptual Design

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for $E_1:E_2$ in R