OVERVIEW OF DBMS

What is Data?

Data is nothing but facts and statistics stored or free flowing over a network, generally it's raw and unprocessed. For example: When you visit any website, they might store you IP address, that is data, in return they might add a cookie in your browser, marking you that you visited the website, that is data, your name, it's data, your age, it's data.

Data becomes **information** when it is processed, turning it into something meaningful. Like, based on the cookie data saved on user's browser, if a website can analyse that generally men of age 20-25 visit us more, that is information, derived from the data collected.

What is a Database?

A **Database** is a collection of related data organised in a way that data can be easily accessed, managed and updated. Database can be software based or hardware based, with one sole purpose, storing data.

The *Database* is an organized collection of structured data to make it easily accessible, manageable and update. In simple words, you can say, a database is a place where the data is stored. The best analogy is the library. The library contains a huge collection of books of different genres, here the library is database and books are the data.

During early computer days, data was collected and stored on tapes, which were mostly write-only, which means once data is stored on it, it can never be read again. They were slow and bulky, and soon computer scientists realised that they needed a better solution to this problem.

Larry Ellison, the co-founder of **Oracle** was amongst the first few, who realised the need for a software based Database Management System.

What is DBMS?

DBMS stands for Database Management System**. We can break it like this DBMS= Database + Management System. Database is a collection of data and Management System is a set of programs to store and retrieve those data. Based on this we can** define DBMS **like this:**

A **DBMS** is a software that allows creation, definition and manipulation of database, allowing users to store, process and analyse data easily. DBMS provides us with an interface or a tool, to perform various operations like creating database, storing data in it, updating data, creating tables in the database and a lot more.

DBMS also provides protection and security to the databases. It also maintains data consistency in case of multiple users.

Here are some examples of popular DBMS used these days:

- MySql
- Oracle
- SQL Server
- IBM DB2
- PostgreSQL
- Amazon SimpleDB (cloud based) etc.

  What is the need of DBMS? Database systems are basically developed for large amount of data. When dealing with huge amount of data, there are two things that require optimization: **Storage of data** and **retrieval of data**.

- **Storage:** According to the principles of database systems, the data is stored in such a way that it acquires lot less space as the redundant data (duplicate data) has been removed before storage. Let’s take a layman example to understand this:

In a banking system, suppose a customer is having two accounts, one is saving account and another is salary account. Let's say bank stores saving account data at one place (these places are called tables we will learn them later) and salary account data at another place, in that case if the customer information such as customer name, address etc. are stored at both places then this is just a wastage of storage (redundancy/ duplication of data), to organize the data in a better way the information should be stored at one place and both the accounts should be linked to that information somehow. The same thing we achieve in DBMS.

- **Fast Retrieval of data**: Along with storing the data in an optimized and systematic manner, it is also important that we retrieve the data quickly when needed. Database systems ensure that the data is retrieved as quickly as possible.

Characteristics of Database Management System

A database management system has following characteristics:

1. **Data stored into Tables:** Data is never directly stored into the database. Data is stored into tables, created inside the database. DBMS also allows to have relationships between tables which makes the data more meaningful and connected. You can easily understand what type of data is stored where by looking at all the tables created in a database.
2. **Reduced Redundancy:** In the modern world hard drives are very cheap, but earlier when hard drives were too expensive, unnecessary repetition of data in database was a big problem. But DBMS follows **Normalisation** which divides the data in such a way that repetition is minimum.
3. **Data Consistency:** On Live data, i.e. data that is being continuosly updated and added, maintaining the consistency of data can become a challenge. But DBMS handles it all by itself.

4. **Support Multiple user and Concurrent Access:** DBMS allows multiple users to work on it(update, insert, delete data) at the same time and still manages to maintain the data consistency.
5. **Query Language:** DBMS provides users with a simple Query language, using which data can be easily fetched, inserted, deleted and updated in a database.
6. **Security:** The DBMS also takes care of the security of data, protecting the data from un-authorised access. In a typical DBMS, we can create user accounts with different access permissions, using which we can easily secure our data by restricting user access.
7. DBMS supports **transactions**, which allows us to better handle and manage data integrity in real world applications where multi-threading is extensively used.

Advantages of DBMS

- Segregation of applicaion program.
- Minimal data duplicacy or data redundancy.
- Easy retrieval of data using the Query Language.
- Reduced development time and maintainance need.
- With Cloud Datacenters, we now have Database Management Systems capable of storing almost infinite data.
- Seamless integration into the application programming languages which makes it very easier to add a database to almost any application or website.

Disadvantages of DBMS

- It's Complexity
- Except MySQL, which is open source, licensed DBMSs are generally costly.
- They are large in size.

DBMS vs. File System

There are following differences between DBMS and File system:

DBMS	File System
DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	File system is a collection of data. In this system, the user has to write the procedures for managing the database.
DBMS gives an abstract view of data that hides the details.	File system provides the detail of the data representation and storage of data.
DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure.	File system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will lost.
DBMS provides a good protection mechanism.	It is very difficult to protect a file under the file system.
DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	File system can't efficiently store and retrieve the data.

DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access has many problems like redirecting the file while other deleting some information or updating some information.

The database management system can be divided into five major components, they are:

1. Hardware
2. Software
3. Data
4. Procedures
5. Database Access Language

Let's have a simple diagram to see how they all fit together to form a database management system.

DBMS Components: Hardware

When we say Hardware, we mean computer, hard disks, I/O channels for data, and any other physical component involved before any data is successfully stored into the memory.

When we run Oracle or MySQL on our personal computer, then our computer's Hard Disk, our Keyboard using which we type in all the commands, our computer's RAM, ROM all become a part of the DBMS hardware.

DBMS Components: Software

This is the main component, as this is the program which controls everything. The DBMS software is more like a wrapper around the physical database, which provides us with an easy-to-use interface to store, access and update data.

The DBMS software is capable of understanding the Database Access Language and intrepret it into actual database commands to execute them on the DB.

DBMS Components: Data

Data is that resource, for which DBMS was designed. The motive behind the creation of DBMS was to store and utilise data.

In a typical Database, the user saved Data is present and meta data is stored.

Metadata is data about the data. This is information stored by the DBMS to better understand the data stored in it.

For example: When I store my Name in a database, the DBMS will store when the name was stored in the database, what is the size of the name, is it stored as related data to some other data, or is it independent, all this information is metadata.

DBMS Components: Procedures

Procedures refer to general instructions to use a database management system. This includes procedures to setup and install a DBMS, To login and logout of DBMS software, to manage databases, to take backups, generating reports etc.

DBMS Components: Database Access Language

Database Access Language is a simple language designed to write commands to access, insert, update and delete data stored in any database.

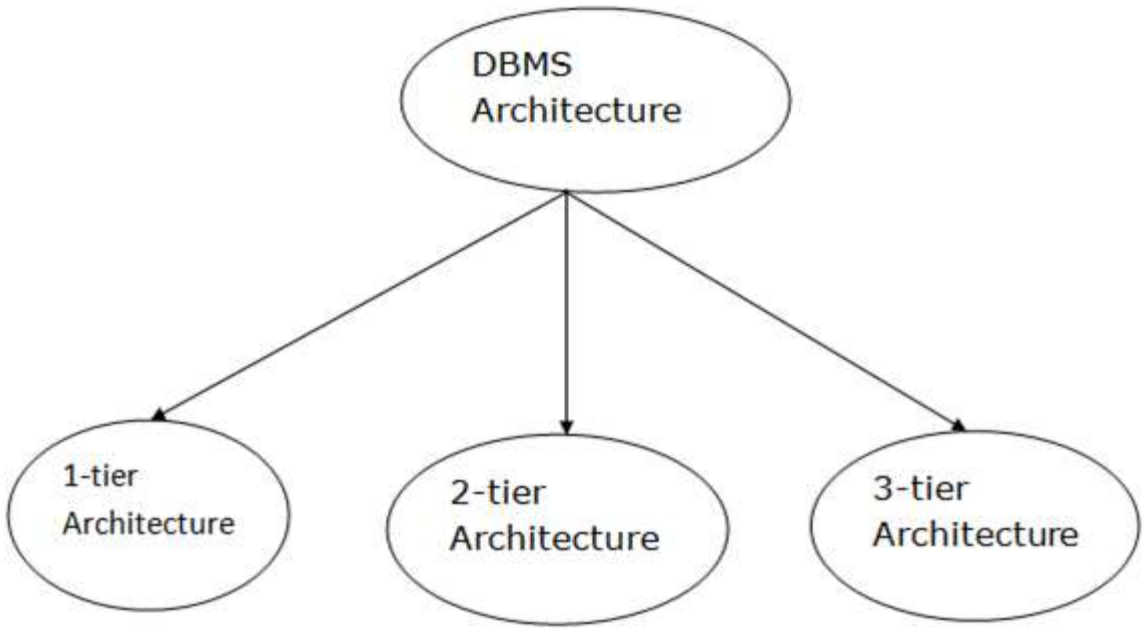
A user can write commands in the Database Access Language and submit it to the DBMS for execution, which is then translated and executed by the DBMS.

User can create new databases, tables, insert data, fetch stored data, update data and delete the data using the access language.

DBMS Architecture

- The DBMS design depends upon its architecture. The basic client/server architecture is used to deal with a large number of PCs, web servers, database servers and other components that are connected with networks.
- The client/server architecture consists of many PCs and a workstation which are connected via the network.
- DBMS architecture depends upon how users are connected to the database to get their request done.

Types of DBMS Architecture



Database architecture can be seen as a single tier or multi-tier. But logically, database architecture is of two types like: **2-tier architecture** and **3-tier architecture**.

1-Tier Architecture

In this architecture, the database is directly available to the user. It means the user can directly sit on the DBMS and uses it.

Any changes done here will directly be done on the database itself. It doesn't provide a handy tool for end users.

The 1-Tier architecture is used for development of the local application, where programmers can directly communicate with the database for the quick response.

2-Tier Architecture

- The 2-Tier architecture is same as basic client-server. In the two-tier architecture, applications on the client end can directly communicate with the database at the server side. For this interaction, API's like: **ODBC**, **JDBC** are used.

- The user interfaces and application programs are run on the client-side.
- The server side is responsible to provide the functionalities like: query processing and transaction management.
- To communicate with the DBMS, client-side application establishes a connection with the server side.

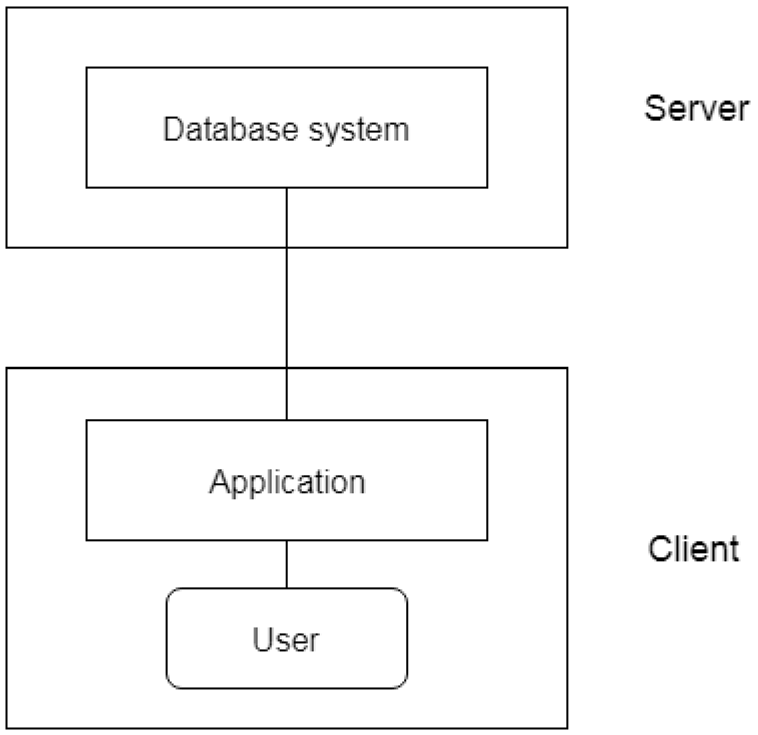


Fig: 2-tier Architecture

3-Tier Architecture

- The 3-Tier architecture contains another layer between the client and server. In this architecture, client can't directly communicate with the server.
- The application on the client-end interacts with an application server which further communicates with the database system.
- End user has no idea about the existence of the database beyond the application server. The database also has no idea about any other user beyond the application.
- The 3-Tier architecture is used in case of large web application.

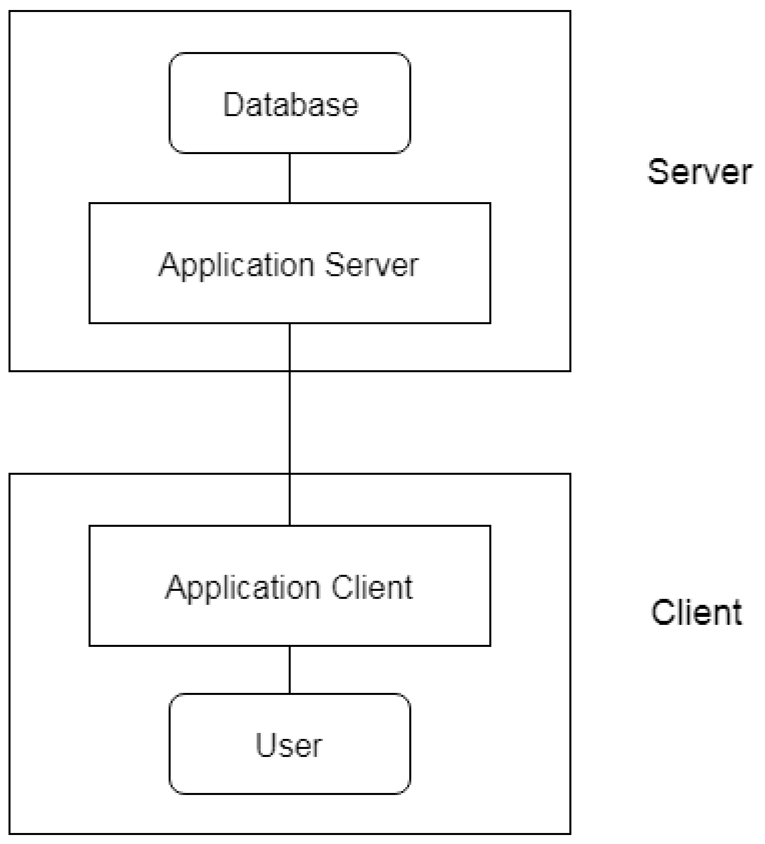


Fig: 3-tier Architecture

Data Abstraction in DBMS

Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction.

We have three levels of abstraction:
Physical level:
This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

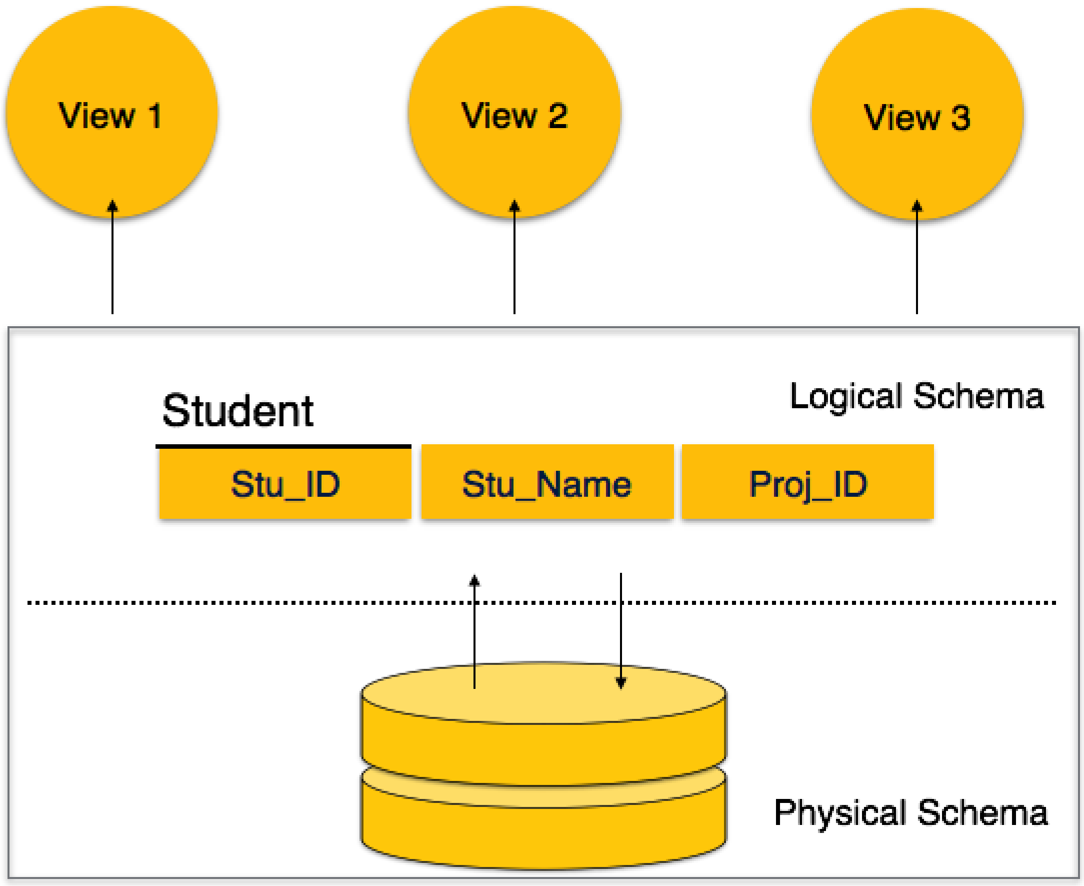
Logical level: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

View level: Highest level of data abstraction. This level describes the user interaction with database system.

Database Schema

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.



A database schema can be divided broadly into two categories −

- **Physical Database Schema** − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
- **Logical Database Schema** − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

Database Instance

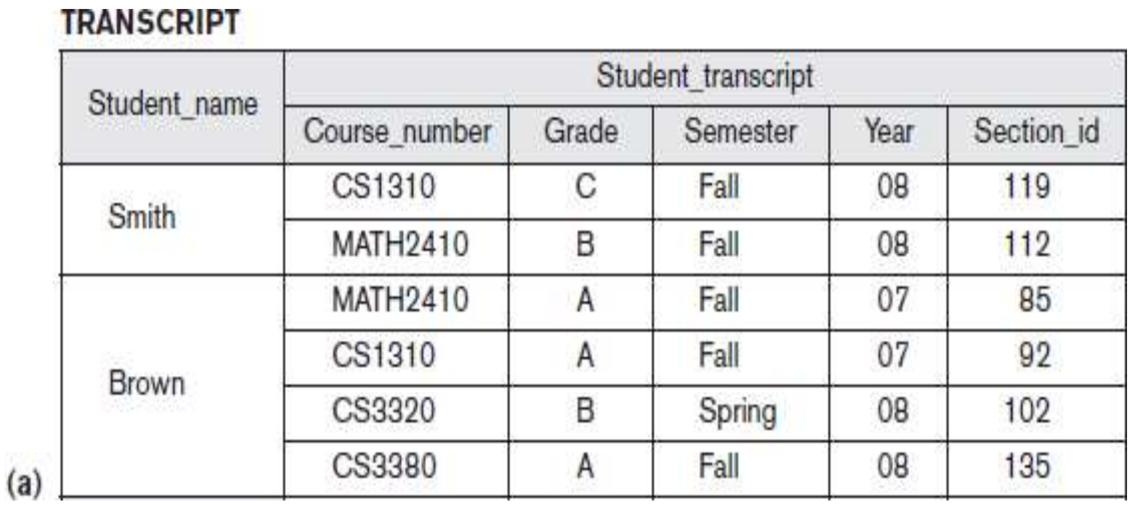
It is important that we distinguish these two terms individually. Database schema is the skeleton of database. It is designed when the database doesn't exist at all. Once the database is operational, it is very difficult to make any changes to it. A database schema does not contain any data or information.

A database instance is a state of operational database with data at any given time. It contains a snapshot of the database. Database instances tend to change with time. A DBMS ensures that its every instance (state) is in a valid state, by diligently following all the validations, constraints, and conditions that the database designers have imposed.

Data Independence

The three-schema architecture can be used to further explain the concept of **data independence**, 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. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item). In the last case, external schemas that refer only to the remaining data should not be affected. For example, the external schema of Figure 1.5(a) should not be affected by changing the GRADE_REPORT file (or record type) shown in Figure 1.2 into the one shown in Figure 1.6(a). Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence. After the conceptual schema undergoes a logical reorganization, application programs that reference the external schema constructs must work as before. Changes to constraints can be applied to the conceptual schema without affecting the external schemas or application programs.



(a)

TRANSCRIPT

Student_name	Student_transcript				
	Course_number	Grade	Semester	Year	Section_id
Smith	CS1310	C	Fall	08	119
	MATH2410	B	Fall	08	112
Brown	MATH2410	A	Fall	07	85
	CS1310	A	Fall	07	92
	CS3320	B	Spring	08	102
	CS3380	A	Fall	08	135

Fig 1.5(a)

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

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

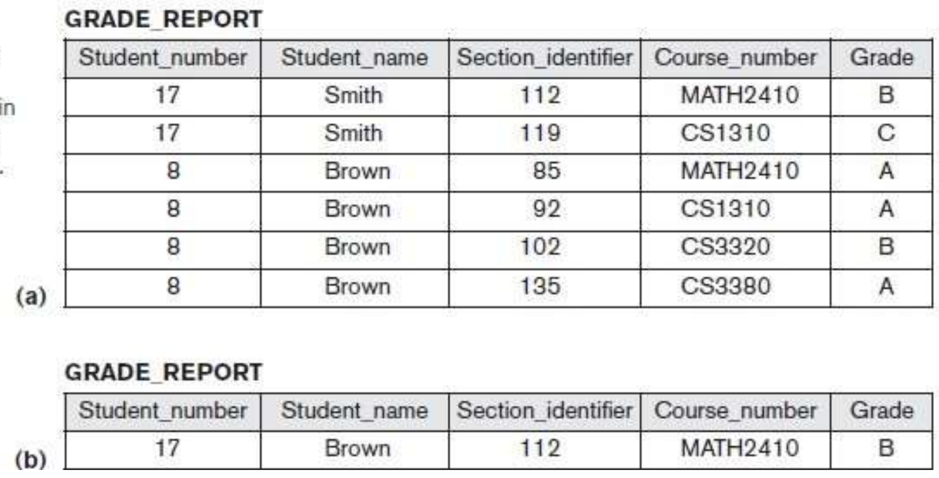


Figure 1.6
Redundant storage of Student_name and Course_name in GRADE_REPORT.
(a) Consistent data.
(b) Inconsistent record.

GRADE_REPORT

(a)

Student_number	Student_name	Section_identifier	Course_number	Grade
17	Smith	112	MATH2410	B
17	Smith	119	CS1310	C
8	Brown	85	MATH2410	A
8	Brown	92	CS1310	A
8	Brown	102	CS3320	B
8	Brown	135	CS3380	A

GRADE_REPORT

(b)

Student_number	Student_name	Section_identifier	Course_number	Grade
17	Brown	112	MATH2410	B

2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema. For example, providing an access path to improve retrieval speed of section records (Figure 1.2) by semester and year should not require a query such as *list all sections offered in fall 2008* to be changed, although the query would be executed more efficiently by the DBMS by utilizing the new access path.
Generally, physical data independence exists in most databases and file environments where physical details such as the exact location of data on disk, and hardware details of storage encoding, placement, compression, splitting, merging of records, and so on are hidden from the user. Applications remain unaware of these details. On the other hand, logical data independence is harder to achieve because it allows structural and constraint changes without affecting application programs—a much stricter requirement.
Whenever we have a multiple-level DBMS, its catalog must be expanded to include information on how to map requests and data among the various levels. The DBMS uses additional software to accomplish these mappings by referring to the mapping information in the catalog. 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. Hence, application programs referring to the higher-level schema need not be changed.
The three-schema architecture can make it easier to achieve true data independence, both physical and logical. However, the two levels of mappings create an overhead during compilation or execution of a query or program, leading to inefficiencies in the DBMS. Because of this, few DBMSs have implemented the full three schema architecture.

The DBMS must provide appropriate languages and interfaces for each category of users. In this section we discuss the types of languages and interfaces provided by a DBMS and the user categories targeted by each interface.

1. DBMS Languages

Once the design of a database is completed and a DBMS is chosen to implement the database, the first step is to specify conceptual and internal schemas for the database and any mappings between the two. In many DBMSs where no strict separation of levels is maintained, one language, called the **data definition language** (**DDL**), is used by the DBA and by database designers to define both schemas. 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. Another language, the **storage definition language** (**SDL**), is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages. In most relational DBMSs today, there *is no specific language* that performs the role of SDL. Instead, the internal schema is specified by a combination of functions, parameters, and specifications related to storage. These permit the DBA staff to control indexing choices and mapping of data to storage. For a true three-schema architecture, we would need a third language, the **view definition language** (**VDL**), to specify user views and their mappings to the conceptual schema, but in most DBMSs *the DDL is used to define both conceptual and external schemas*. In relational DBMSs, SQL is used in the role of VDL to define user or application **views** as results of predefined queries.

Once the database schemas are 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** (**DML**) for these purposes.

In current DBMSs, the preceding types of languages are usually *not considered distinct languages*; rather, a comprehensive integrated language is used that includes constructs for conceptual schema definition, view definition, and data manipulation. Storage definition is typically kept separate, since it is used for defining physical storage structures to fine-tune the performance of the database system, which is usually done by the DBA staff. A typical example of a comprehensive database language is the 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. The SDL was a component in early versions of SQL but has been removed from the language to keep it at the conceptual and external levels only.

There are two main types of DMLs. A **high-level** or **nonprocedural** DML can be used on its own to specify complex database operations concisely. Many DBMSs allow high-level DML statements either to be entered interactively from a display monitor or terminal or to be embedded in a general-purpose programming language. In the latter case, DML statements must be identified within the program so that they can be extracted by a pre compiler and processed by the DBMS. A **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 because of this property. DL/1, a DML designed for the hierarchical model, is a low-level DML that uses commands such as GET UNIQUE, GET NEXT, or GET NEXT WITHIN PARENT to navigate from record to record within a hierarchy of records in the database. High-level DMLs, such as SQL, can specify and retrieve many records in a single DML statement; therefore, they are called **set-at-a-time** or **set-oriented** DMLs. A query in a high-level DML often specifies *which* data to retrieve rather than *how* to retrieve it; therefore, such languages are also called **declarative**.

Whenever DML commands, whether high level or low level, are embedded in a general-purpose programming language, that language is called the **host language** and the DML is called the **data sublanguage**. On the other hand, a high-level DML used in a standalone interactive manner is called a **query language**. In general, both retrieval and update commands of a high-level DML may be used interactively and are hence considered part of the query language.

Casual end users typically use a high-level query language to specify their requests, whereas programmers use the DML in its embedded form. For naive and parametric users, there usually are **user-friendly interfaces** for interacting with the data-base; these can also be used by casual users or others who do not want to learn the details of a high-level query language.

Interfaces in DBMS

A database management system (DBMS) interface is a user interface which allows for the ability to input queries to a database without using the query language itself.

User-friendly interfaces provide by DBMS may include the following:

1. **Menu-Based Interfaces for Web Clients or Browsing** –

   These interfaces present the user with lists of options (called menus) that lead the user through the formation of a request. Basic advantage of using menus is that they removes the tension of remembering specific commands and syntax of any query language, rather than query is basically composed step by step by collecting or picking options from a menu that is basically shown by the system. Pull-down menus are a very popular technique in Web based interfaces. They are also often used in browsing interface which allow a user to look through the contents of a database in an exploratory and unstructured manner.

2. **Forms-Based Interfaces** –
   A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert a new data, or they can fill out only certain entries, in which case the DBMS will redeem same type of data for other remaining entries. This type of forms are usually designed or created and programmed for the users that have no expertise in operating system. Many DBMSs have forms specification languages which are special languages that help specify such forms.
   Example: SQL* Forms is a form-based language that specifies queries using a form designed in conjunction with the relational database schema.

3. **Graphical User Interface –**
   A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUI's utilize both menus and forms. Most GUIs use a pointing device such as mouse, to pick certain part of the displayed schema diagram.

4. **Natural language Interfaces –**
   These interfaces accept request written in English or some other language and attempt to understand them. A Natural language interface has its own schema, which is similar to the database conceptual schema as well as a dictionary of important words.
   The natural language interface refers to the words in its schema as well as to the set of standard words in a dictionary to interpret the request. If the interpretation is successful, the interface generates a high-level query corresponding to the natural language and submits it to the DBMS for processing, otherwise a dialogue is started with the user to clarify any provided condition or request. The main disadvantage with this is that the capabilities of this type of interfaces are not that much advance.

5. **Speech Input and Output –**

   There is an limited use of speech say it for a query or an answer to a question or being a result of a request it is becoming commonplace Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure, and bank account information are allowed speech for input and output to enable ordinary folks to access this information.
   The Speech input is detected using a predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech take place.

6. **Interfaces for DBA –**

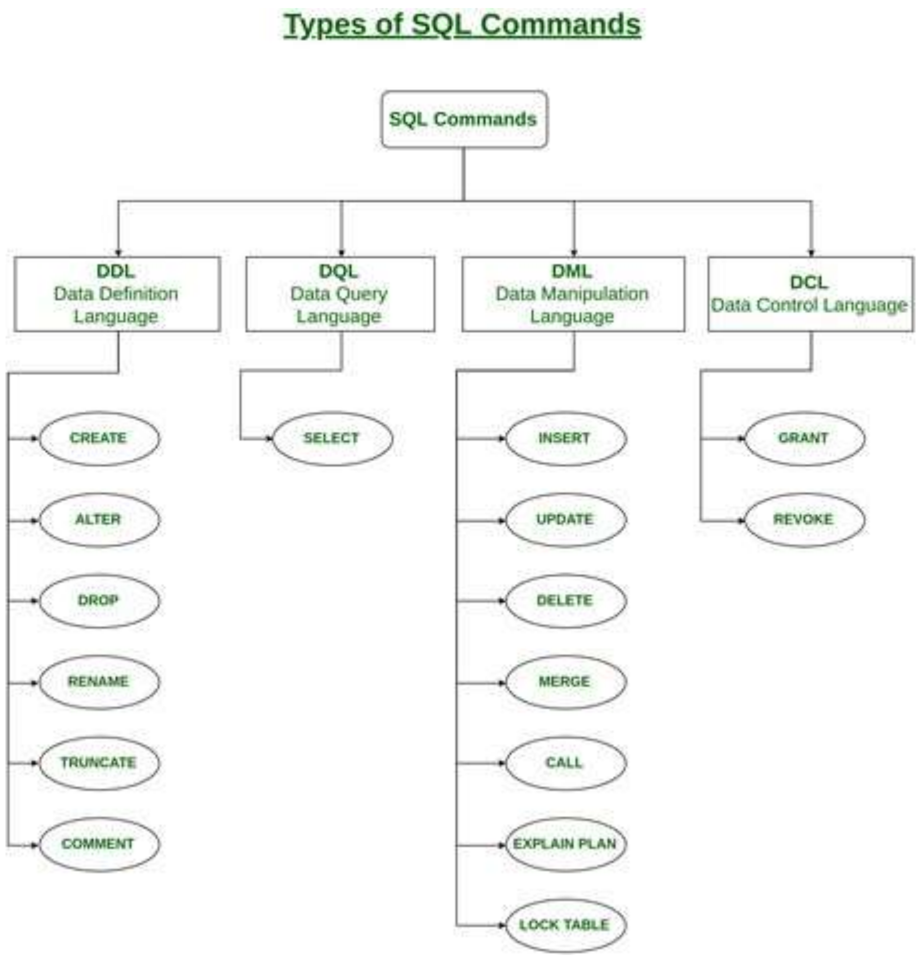
   Most database system contains privileged commands that can be used only by the DBA's staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, reorganizing the storage structures of a databases.

SQL | DDL, DQL, DML, DCL and TCL Commands

Structured Query Language (SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, and Insert etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as:

1. DDL – Data Definition Language
2. DQL – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language



Though many resources claim there to be another category of SQL clauses **TCL – Transaction Control Language**. So we will see in detail about TCL as well.

1. **DDL (Data Definition Language) :** DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.
   Examples of DDL commands:
   - **CREATE** – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
   - **DROP** – is used to delete objects from the database.
   - **ALTER**-is used to alter the structure of the database.
   - **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed.
   - **COMMENT** –is used to add comments to the data dictionary.
   - **RENAME** –is used to rename an object existing in the database.

2. **DQL (Data Query Language) :**
   DML statements are used for performing queries on the data within schema objects. The purpose of DQL Command is to get some schema relation based on the query passed to it.

   Example of DQL:
   - **SELECT** – is used to retrieve data from the a database.

3. **DML (Data Manipulation Language) :** The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.
   Examples of DML:
   - **INSERT** – is used to insert data into a table.
   - **UPDATE** – is used to update existing data within a table.
   - **DELETE** – is used to delete records from a database table.
   - **MERGE** – is used to make changes in one table based on values matched from anther. It can be used to combine insert, update, and delete operations into one statement.
   - **CALL** - statement calls a procedure created by the CREATE PROCEDURE or DECLARE PROCEDURE statement.
   - **EXPLAIN PLAN - command** displays the **execution plan** chosen by the Oracle optimizer for SELECT, UPDATE, INSERT, and DELETE statements.
   - **LOCK TABLE - command** is used to **lock** the **table** named sql_name in either EXCLUSIVE or SHARE mode. In EXCLUSIVE mode, the data of the **table** cannot be read or modified by another transaction. In SHARE mode, the data of the **table** can be read by concurrent transactions but modifications are still prohibited.

4. **DCL (Data Control Language):** DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.
   Examples of DCL commands:
   - **GRANT**-gives user's access privileges to database.

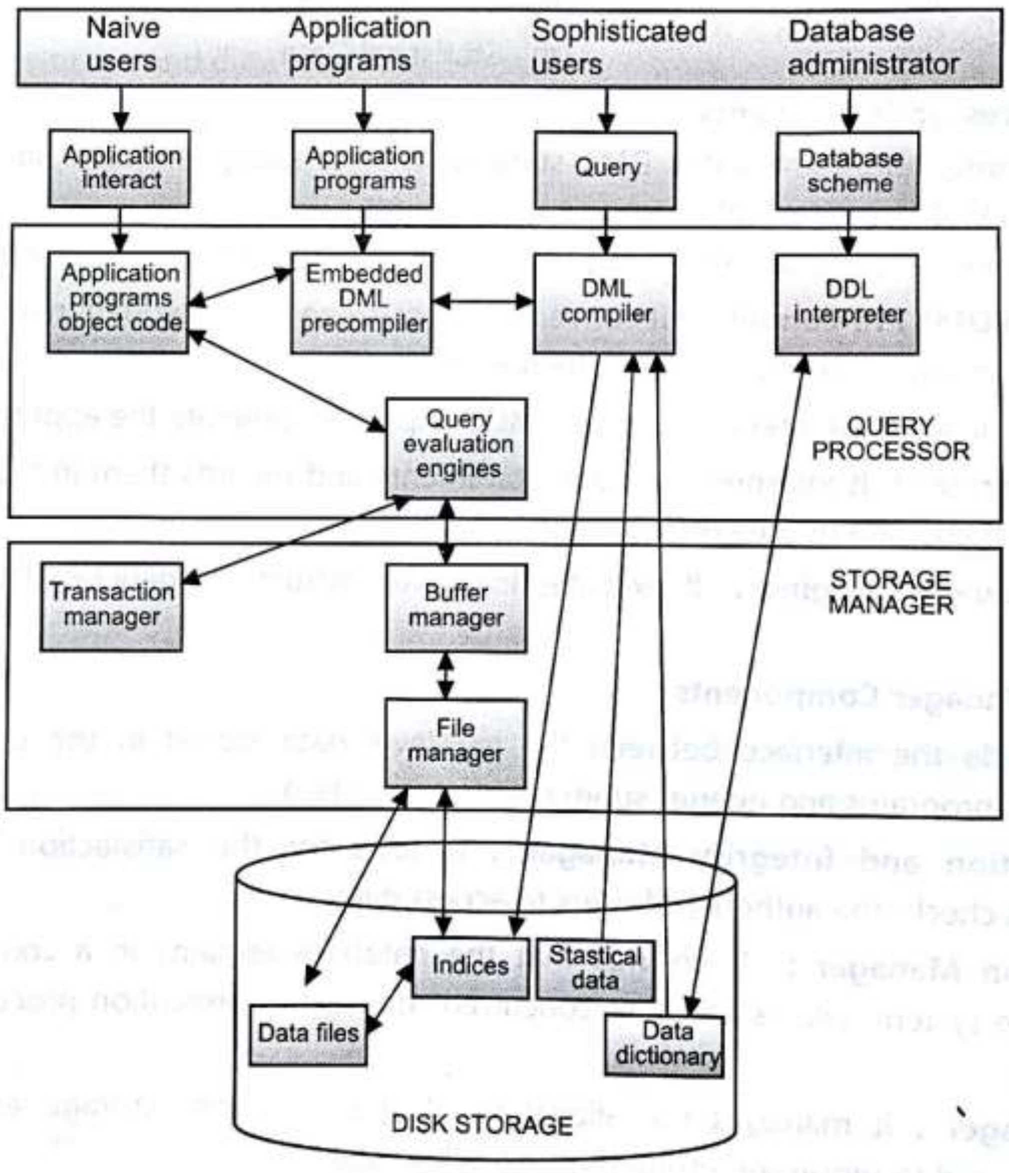
- **REVOKE**-withdraw user's access privileges given by using the GRANT command.

5. **TCL (transaction Control Language):** TCL commands deals with the transaction within the database.
   Examples of TCL commands:
   - **COMMIT**– commits a Transaction.
   - **ROLLBACK**– rollbacks a transaction in case of any error occurs.
   - **SAVEPOINT**–sets a save point within a transaction.
   - **SET TRANSACTION**–specify characteristics for the transaction.

OVERALL DBMS STRUCTURE

In this topic, we will cover the Structure of Database Management System (DBMS). DBMS is responsible to store huge amounts of data and is capable of handling multiple requests from users simultaneously, it should be arranged properly. One can imagine a database as a brain! How is the structure of the brain? Bit sophisticated and each part of the brain is responsible for some specific tasks. Similarly, Database is also designed. A database is partitioned in modules that deal with each of the responsibilities of the overall system.

At a very high level, a database is considered as shown in the below diagram. Let us see them in detail below.



System structure

Components of DBMS are broadly classified as follows:

1. Query Processor :
    - (a) DML Compiler
    - (b) Embedded DML pre-compiler
    - (c) DDL Interpreter
    - (d) Query Evaluation Engine

2. Storage Manager :
    - (a) Authorization and Integrity Manager
    - (b) Transaction Manager
    - (c) File Manager
    - (d) Buffer Manager

3. Data Structure :
    (a) Data Files
    (b) Data Dictionary
    (c) Indices
    (d) Statistical Data

1. Query Processor Components:

   • DML Pre-compiler: It translates DML statements in a query language into low level instructions that query evaluation engine understands. It also attempts to transform user's request into an equivalent but more efficient form.

   • Embedded DML Pre-compiler: It converts DML statements embedded in an application program to normal procedure calls in the host language. The Pre-compiler must interact with the DML compiler to generate the appropriate code.

   • DDL Interpreter: It interprets the DDL statements and records them in a set of tables containing meta data or data dictionary.

   • Query Evaluation Engine: It executes low-level instructions generated by the DML compiler.

2. Storage Manager Components:

   They provide the interface between the low-level data stored in the database and application programs and queries submitted to the system.

   • Authorization and Integrity Manager: It tests for the satisfaction of integrity constraints checks the authority of users to access data.

   • Transaction Manager : It ensures that the database remains in a consistent state despite the system failures and that concurrent transaction execution proceeds without conflicting.

   • File Manager : It manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

   • Buffer Manager : It is responsible for fetching data from disk storage into main memory and deciding what data to cache in memory.

3. Data Structures:

   Following data structures are required as a part of the physical system implementation.

• Data Files: It stores the database.
• Data Dictionary: It stores meta data (data about data) about the structure of the database.
• Indices: Provide fast access to data items that hold particular values.
• Statistical Data: It stores statistical information about the data in the database. This information is used by query processor to select efficient ways to execute query.

- **Applications: –** It can be considered as a user-friendly web page where the user enters the requests. Here he simply enters the details that he needs and presses buttons to get the data.

- **End User: –** They are the real users of the database. They can be developers, designers, administrators, or the actual users of the database.
- **DDL: –** Data Definition Language (DDL) is a query fired to create database, schema, tables, mappings, etc in the database. These are the commands used to create objects like tables, indexes in the database for the first time. In other words, they create the structure of the database.
- **DDL Compiler: –** This part of the database is responsible for processing the DDL commands. That means this compiler actually breaks down the command into machine-understandable codes. It is also responsible for storing the metadata information like table name, space used by it, number of columns in it, mapping information, etc.
- **DML Compiler: –** When the user inserts, deletes, updates or retrieves the record from the database, he will be sending requests which he understands by pressing some buttons. But for the database to work/understand the request, it should be broken down to object code. This is done by this compiler. One can imagine this as when a person is asked some question, how this is broken down into waves to reach the brain!
- **Query Optimizer: –** When a user fires some requests, he is least bothered how it will be fired on the database. He is not all aware of the database or its way of performance. But whatever be the request, it should be efficient enough to fetch, insert, update, or delete the data from the database. The query optimizer decides the best way to execute the user request which is received from the DML compiler. It is similar to selecting the best nerve to carry the waves to the brain!
- **Stored Data Manager: –** This is also known as Database Control System. It is one of the main central systems of the database. It is responsible for various tasks

    - It converts the requests received from query optimizer to machine-understandable form. It makes actual requests inside the database. It is like fetching the exact part of the brain to answer.
    - It helps to maintain consistency and integrity by applying the constraints. That means it does not allow inserting/updating / deleting any data if it has child entry. Similarly, it does not allow entering any duplicate value into database tables.
    - It controls concurrent access. If there are multiple users accessing the database at the same time, it makes sure, all of them see correct data. It guarantees that there is no data loss or data mismatch happens between the transactions of multiple users.
    - It helps to back up the database and recovers data whenever required. Since it is a huge database and when there is any unexpected exploit of the transaction, and reverting the changes is not easy. It maintains the backup of all data so that it can be recovered.
- **Data Files: –** It has the real data stored in it. It can be stored as magnetic tapes, magnetic disks, or optical disks.
- **Compiled DML: –** Some of the processed DML statements (insert, update, delete) are stored in it so that if there are similar requests, it will be re-used.
- **Data Dictionary: –** It contains all the information about the database. As the name suggests, it is the dictionary of all the data items. It contains a description of all the tables, view, materialized views, constraints, indexes, triggers, etc.

DBMS

Lecture 1: ER Model Basics

What is the ER Model?

ENTITY RELATIONAL (ER) MODEL is a high-level conceptual data model diagram. ER modeling helps you to analyze data requirements systematically to produce a well-designed database. The Entity-Relation model represents real-world entities and the relationship between them. It is considered a best practice to complete ER modeling before implementing your database.

ER modeling helps you to analyze data requirements systematically to produce a well-designed database. So, it is considered a best practice to complete ER modeling before implementing your database.

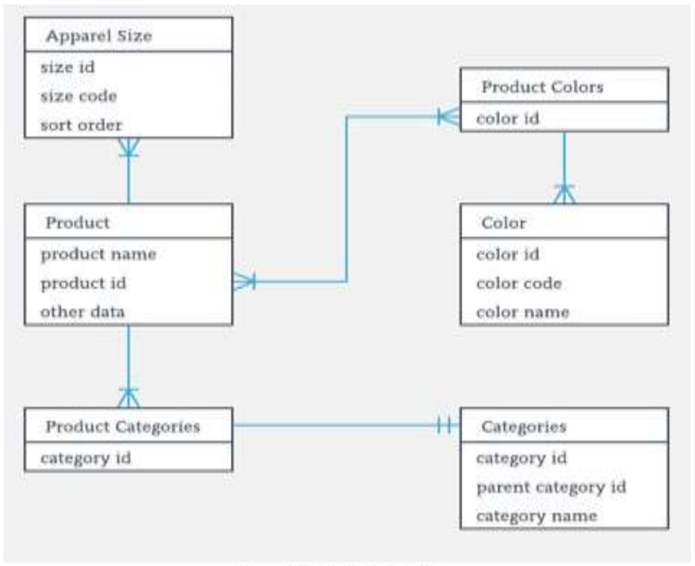
History of ER models

ER diagrams are a visual tool which is helpful to represent the ER model. It was proposed by Peter Chen in 1971 to create a uniform convention which can be used for relational database and network. He aimed to use an ER model as a conceptual modeling approach.

What is ER Diagrams?

ENTITY-RELATIONSHIP DIAGRAM (ERD) displays the relationships of entity set stored in a database. In other words, we can say that ER diagrams help you to explain the logical structure of databases. At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique.

The purpose of ER Diagram is to represent the entity framework infrastructure.



Sample ER Diagram

Facts about ER Diagram Model:

- ER model allows you to draw Database Design
- It is an easy to use graphical tool for modeling data
- Widely used in Database Design
- It is a GUI representation of the logical structure of a Database
- It helps you to identifies the entities which exist in a system and the relationships between those entities

Why use ER Diagrams?

Here, are prime reasons for using the ER Diagram

- ER diagrams can be used by database designers as a blueprint for implementing data in specific software applications
- The database designer gains a better understanding of the information to be contained in the database with the help of ERP diagram
- ERD is allowed you to communicate with the logical structure of the database to users
- Provide a preview of how all your tables should connect, what fields are going to be on each table
- Helps to describe entities, attributes, relationships

- ER diagrams are translatable into relational tables which allows you to build databases quickly

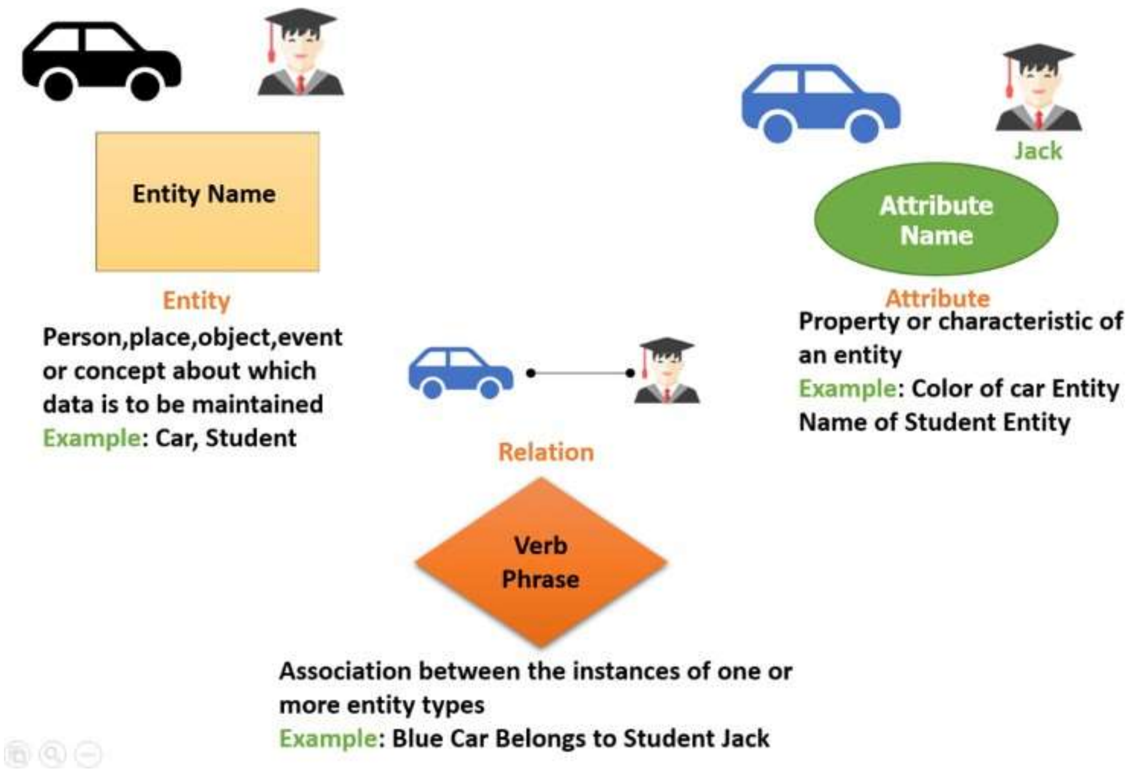
Components of the ER Diagram

This model is based on three basic concepts:

- Entities
- Attributes
- Relationships

Example

For example, in a University database, we might have entities for Students, Courses, and Lecturers. Students entity can have attributes like Rollno, Name, and DeptID. They might have relationships with Courses and Lecturers.


Entity Name
Entity
Person,place,object,event or concept about which data is to be maintained
Example: Car, Student
Jack
Attribute Name
Attribute
Property or characteristic of an entity
Example: Color of car Entity Name of Student Entity
Relation
Verb Phrase
Association between the instances of one or more entity types
Example: Blue Car Belongs to Student Jack

WHAT IS ENTITY?

A real-world thing either living or non-living that is easily recognizable and nonrecognizable. It is anything in the enterprise that is to be represented in our database. It may be a physical thing or simply a fact about the enterprise or an event that happens in the real world.

An entity can be place, person, object, event or a concept, which stores data in the database. The characteristics of entities are must have an attribute, and a unique key. Every entity is made up of some 'attributes' which represent that entity.

Examples of entities:

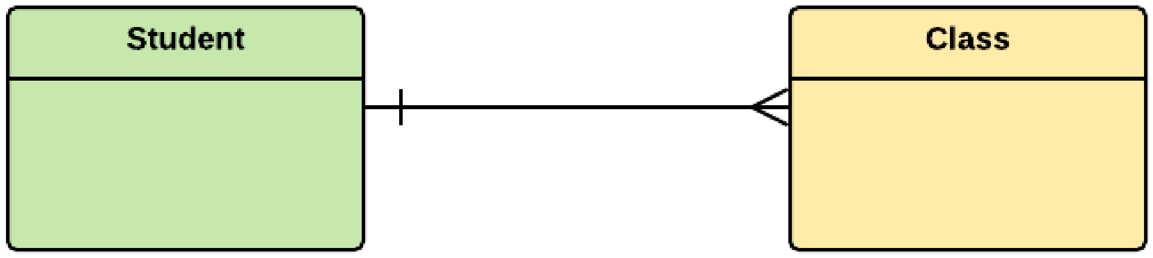
- **Person:** Employee, Student, Patient
- **Place:** Store, Building
- **Object:** Machine, product, and Car
- **Event:** Sale, Registration, Renewal
- **Concept:** Account, Course

Notation of an Entity

Entity set:

Student

An entity set is a group of similar kind of entities. It may contain entities with attribute sharing similar values. Entities are represented by their properties, which also called attributes. All attributes have their separate values. For example, a student entity may have a name, age, class, as attributes.



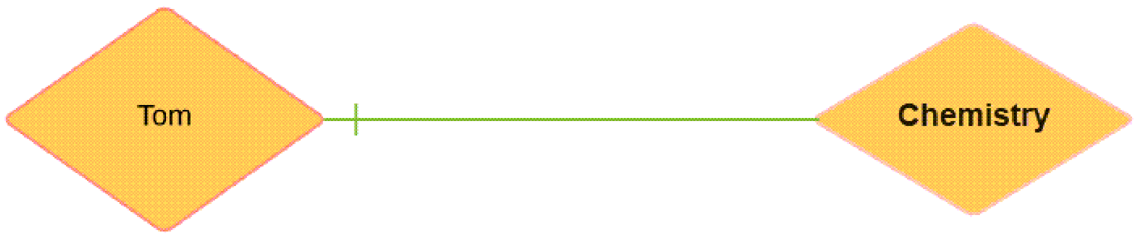
Example of Entities:

A university may have some departments. All these departments employ various lecturers and offer several programs.

Some courses make up each program. Students register in a particular program and enroll in various courses. A lecturer from the specific department takes each course, and each lecturer teaches a various group of students.

Relationship

Relationship is nothing but an association among two or more entities. E.g., Tom works in the Chemistry department.



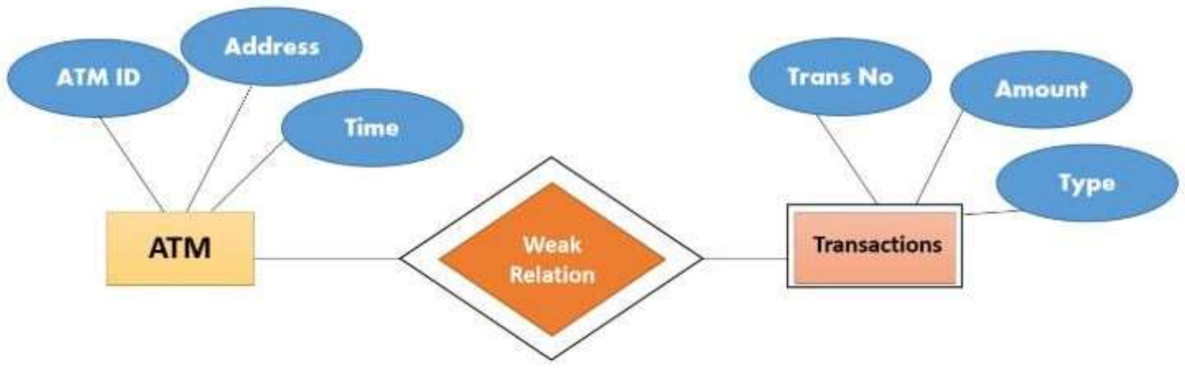
Entities take part in relationships. We can often identify relationships with verbs or verb phrases.

For example:

- You are attending this lecture
- I am giving the lecture
- Just loke entities, we can classify relationships according to relationship-types:
- A student attends a lecture
- A lecturer is giving a lecture.

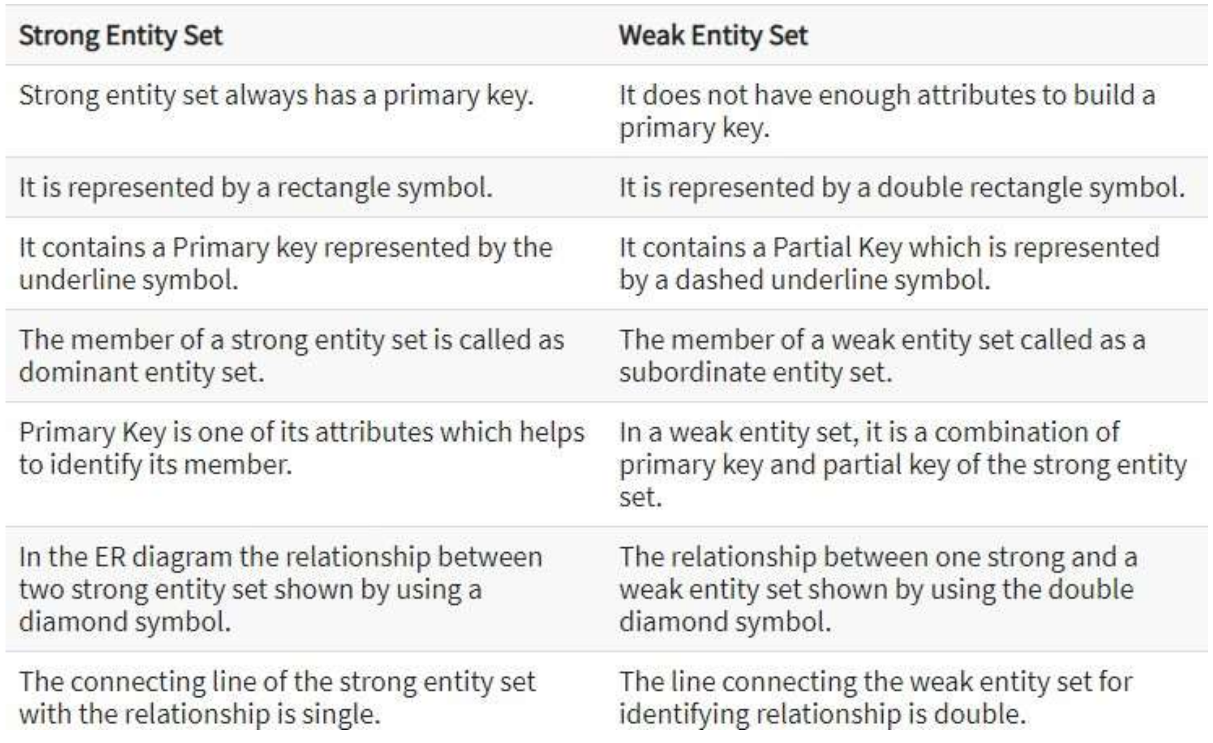
Weak Entities

A weak entity is a type of entity which doesn't have its key attribute. It can be identified uniquely by considering the primary key of another entity. For that, weak entity sets need to have participation.



In above example, "Trans No" is a discriminator within a group of transactions in an ATM.

Let's learn more about a weak entity by comparing it with a Strong Entity:



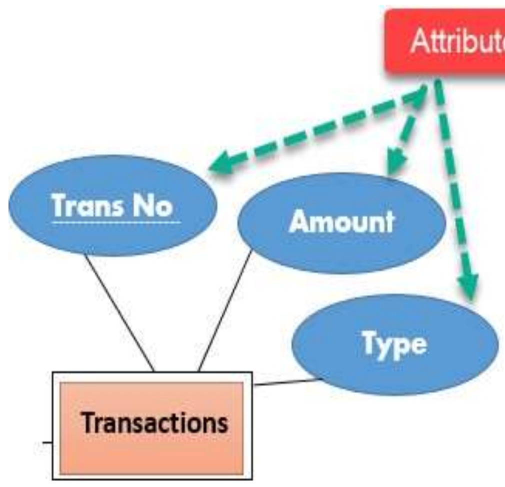
Strong Entity Set	Weak Entity Set
Strong entity set always has a primary key.	It does not have enough attributes to build a primary key.
It is represented by a rectangle symbol.	It is represented by a double rectangle symbol.
It contains a Primary key represented by the underline symbol.	It contains a Partial Key which is represented by a dashed underline symbol.
The member of a strong entity set is called as dominant entity set.	The member of a weak entity set called as a subordinate entity set.
Primary Key is one of its attributes which helps to identify its member.	In a weak entity set, it is a combination of primary key and partial key of the strong entity set.
In the ER diagram the relationship between two strong entity set shown by using a diamond symbol.	The relationship between one strong and a weak entity set shown by using the double diamond symbol.
The connecting line of the strong entity set with the relationship is single.	The line connecting the weak entity set for identifying relationship is double.

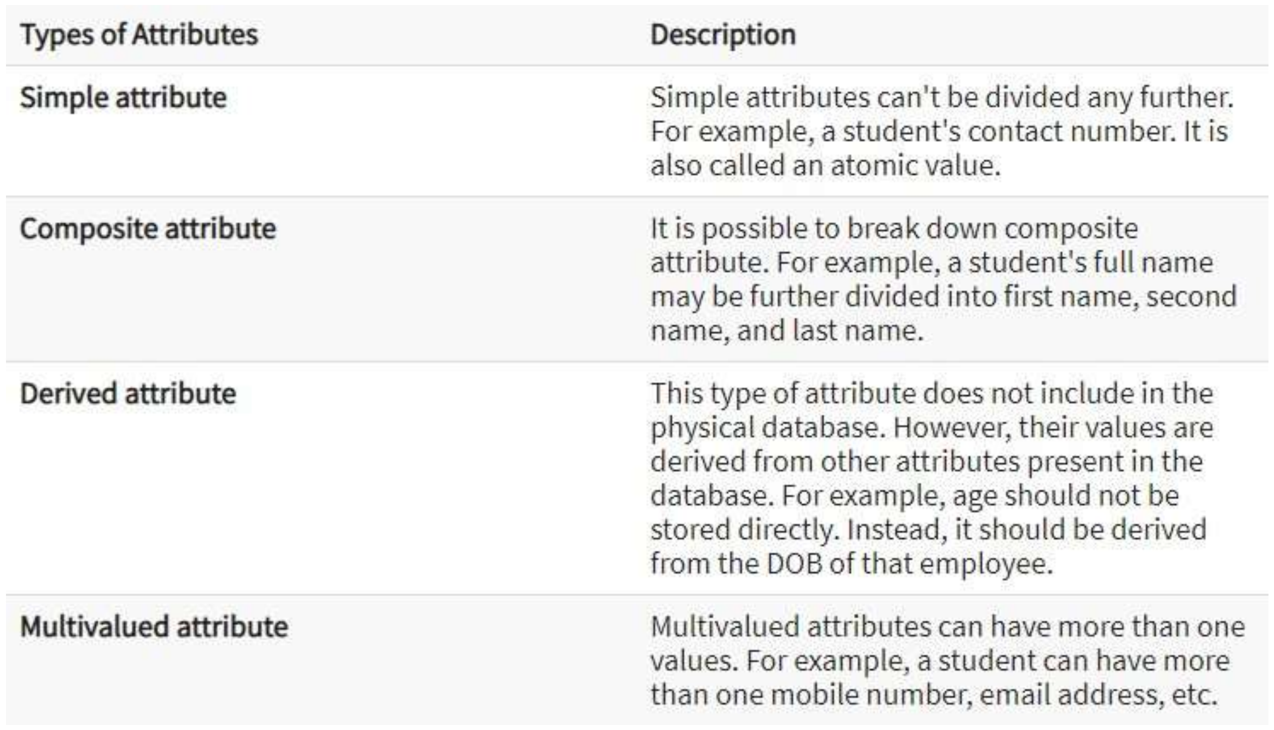
Attributes

It is a single-valued property of either an entity-type or a relationship-type.

For example, a lecture might have attributes: time, date, duration, place, etc.

An attribute is represented by an Ellipse





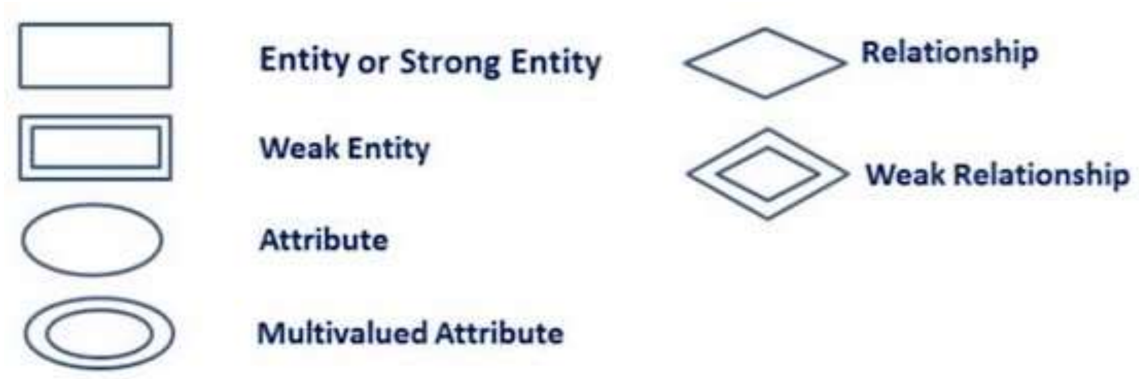
Types of Attributes	Description
Simple attribute	Simple attributes can't be divided any further. For example, a student's contact number. It is also called an atomic value.
Composite attribute	It is possible to break down composite attribute. For example, a student's full name may be further divided into first name, second name, and last name.
Derived attribute	This type of attribute does not include in the physical database. However, their values are derived from other attributes present in the database. For example, age should not be stored directly. Instead, it should be derived from the DOB of that employee.
Multivalued attribute	Multivalued attributes can have more than one values. For example, a student can have more than one mobile number, email address, etc.

ER- Diagram Notations

ER- Diagram is a visual representation of data that describe how data is related to each other.

- **Rectangles:** This symbol represent entity types
- **Ellipses :** Symbol represent attributes
- **Diamonds:** This symbol represents relationship types

- **Lines:** It links attributes to entity types and entity types with other relationship types
- **Primary key:** attributes are underlined
- **Double Ellipses:** Represent multi-valued attributes



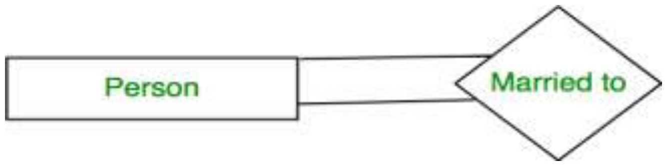
DBMS

Lecture 2: Degree, Cardinality and Participation Constraints in Relation

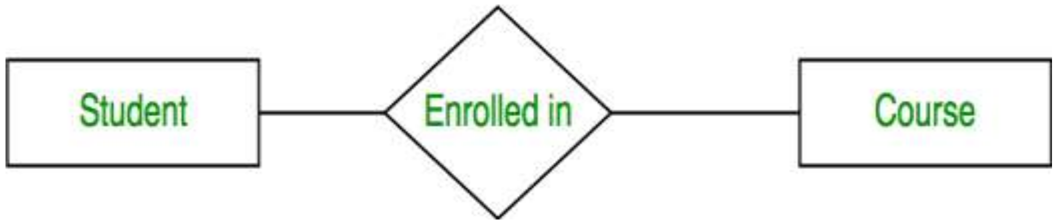
1. Degree Of An Entity Relationship Type:

The number of different entity sets **participating in a relationship** set is called as degree of a relationship set.

1. **Unary Relationship –**
   When there is **only ONE entity set participating in a relation**, the relationship is called as unary relationship. For example, one person is married to only one person.



2. **Binary Relationship –**
   When there are **TWO entities set participating in a relation**, the relationship is called as binary relationship.For example, Student is enrolled in Course.



3. **n-ary Relationship –**
   When there are n entities set participating in a relation, the relationship is called as n-ary relationship.

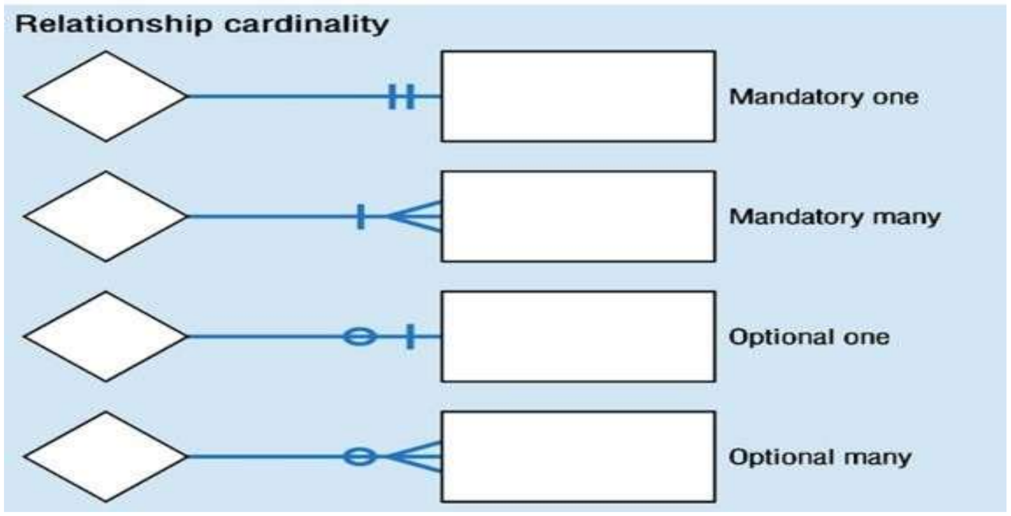
2. Cardinality

Defines the numerical attributes of the relationship between two entities or entity sets. Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

By connectivity we mean how many instances of one entity are associated with how many instances of other entity in a relationship. Cardinality is used to specify such connectivity. The connectivity of a relationship describes the mapping of associated entity instances in the relationship. The values of connectivity are "one" or "many". The cardinality of a relationship is the actual number of related occurrences for each of the two entities.

Different types of cardinal relationships are:

- One-to-One Relationships
- One-to-Many Relationships
- May to One Relationships
- Many-to-Many Relationships

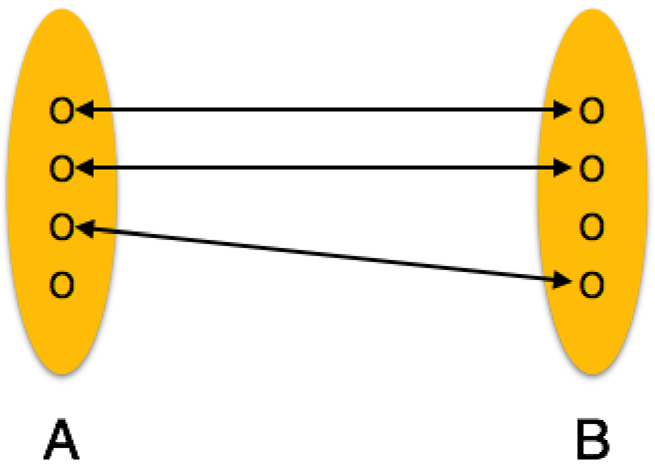


1. One-to-one:

One entity from entity set X can be associated with at most one entity of entity set Y and vice versa.

For example, one Student can have only one college ID at a time.

One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



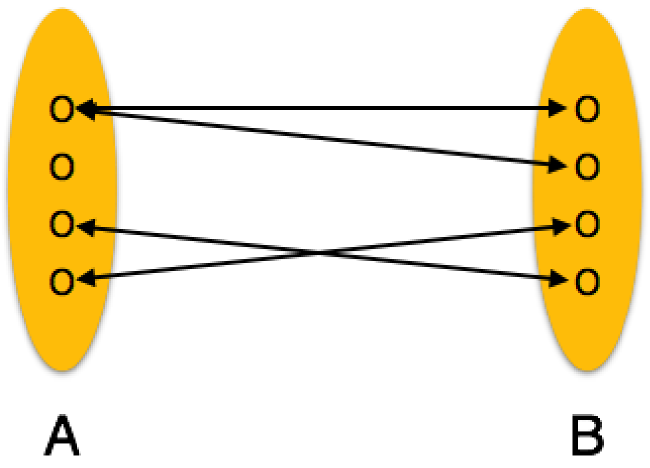
A **one-to-one (1:1)** relationship is when at most one instance of an entity A is associated with one instance of entity B. For example, take the relationship between board members and offices, where each office is held by one member and no member may hold more than one office.

2. One-to-many:

One entity from entity set X can be associated with multiple entities of entity set Y, but an entity from entity set Y can be associated with at least one entity.

For example, one class is consisting of multiple students.

One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



A **one-to-many (1:N)** relationship is when for one instance of entity A, there are zero, one, or many instances of entity B but for one instance of entity B, there is only one instance of entity A. An example of a 1:N relationships is

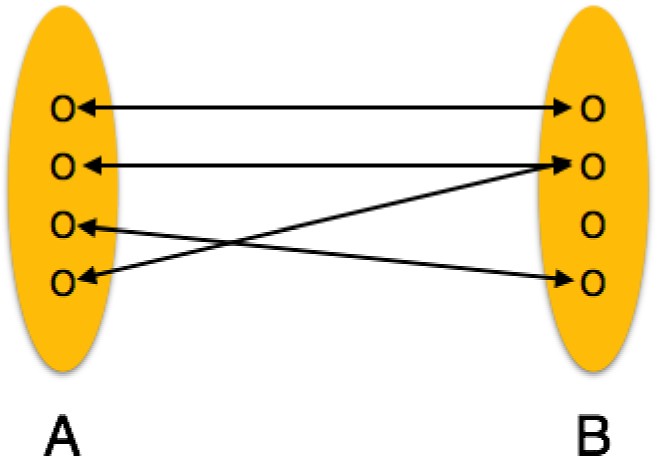
a department has many employees; each employee is assigned to one department.

3. Many to One

More than one entity from entity set X can be associated with at most one entity of entity set Y. However, an entity from entity set Y may or may not be associated with more than one entity from entity set X.

For example, many students belong to the same class.

More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.

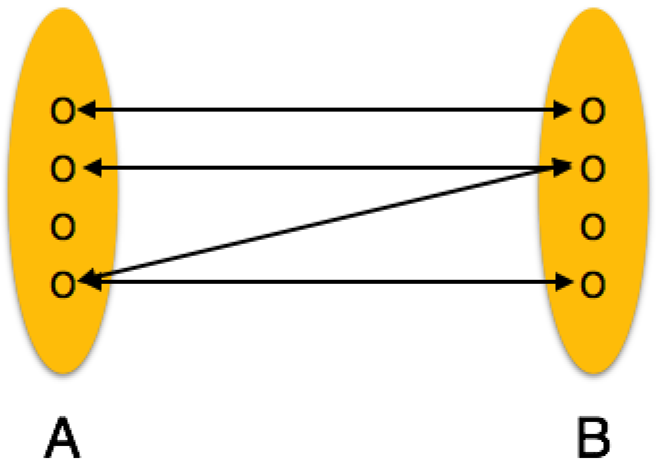


4. Many to Many:

One entity from X can be associated with more than one entity from Y and vice versa.

For example, Students as a group are associated with multiple faculty members, and faculty members can be associated with multiple students.

One entity from A can be associated with more than one entity from B and vice versa.



A **many-to-many (M:N)** relationship, sometimes called non-specific, is when for one instance of entity A, there are zero, one, or many instances of entity B and for one instance of entity B there are zero, one, or many instances of entity A. An example is employees may be assigned to no more than three projects at a time; every project has at least two employees assigned to it.

Here the cardinality of the relationship from employees to projects is three; from projects to employees, the cardinality is two. Therefore, this relationship can be classified as a many-to-many relationship.

If a relationship can have a cardinality of zero, it is an optional relationship. If it must have a cardinality of at least one, the relationship is mandatory. Optional relationships are typically indicated by the conditional tense. For example,

An employee may be assigned to a project.

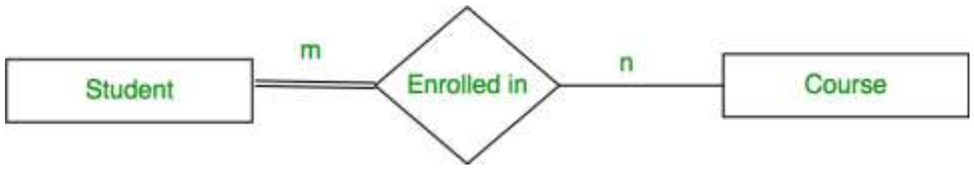
Mandatory relationships, on the other hand, are indicated by words such as must have. For example, a student must register for at least three courses in each semester.

3. Participation Constraint:

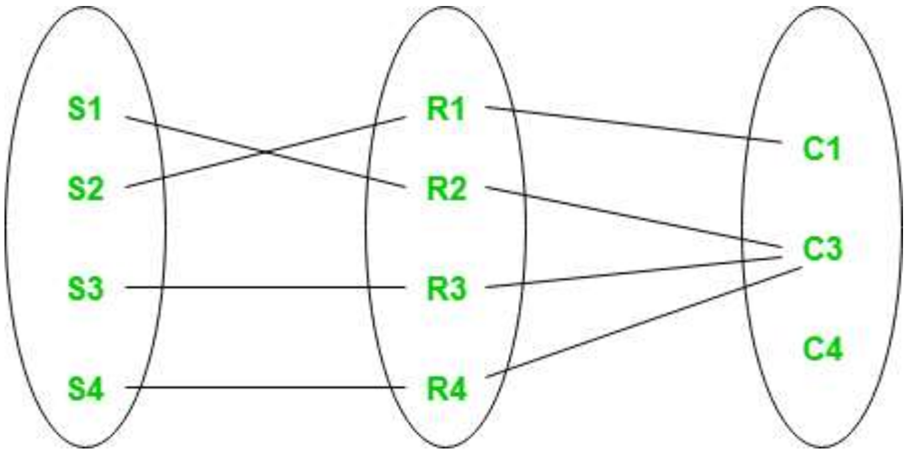
Participation Constraint is applied on the entity participating in the relationship set.

1. **Total Participation –** Each entity in the entity set **must participate** in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.

2. **Partial Participation –** The entity in the entity set **may or may NOT participat**e in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.
   The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.



   Using set, it can be represented as,



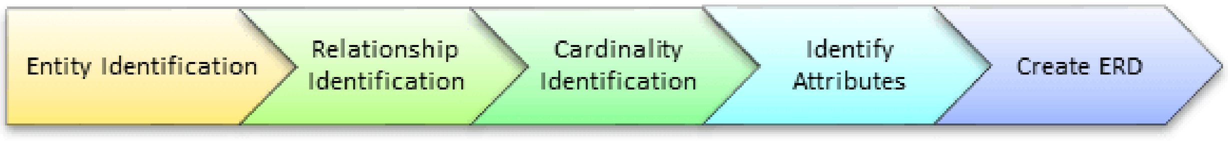
   Every student in Student Entity set is participating in relationship but there exists a course C4 which is not taking part in the relationship.

DBMS

Lecture 3: Model an ER Diagram

Steps to Create an ERD

Following are the steps to create an ERD.



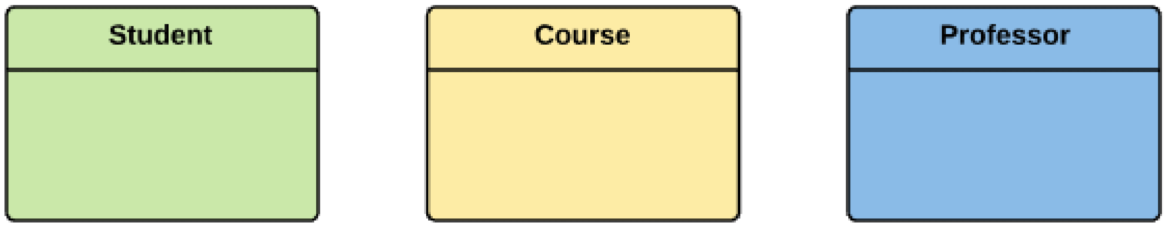
Let's study them with an example:

```
In a university, a Student enrolls in Courses. A student must be assigned to at least
one or more Courses. Each course is taught by a single Professor. To maintain instruc
tion quality, a Professor can deliver only one course
```

Step 1) Entity Identification

We have three entities

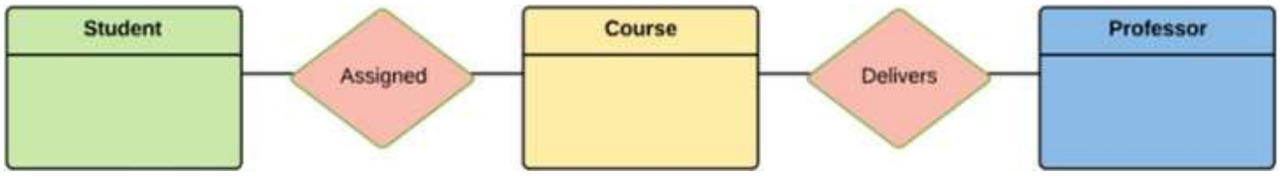
- Student
- Course
- Professor



Step 2) Relationship Identification

We have the following two relationships

- The student is **assigned** a course
- Professor **delivers** a course



Step 3) Cardinality Identification

For them problem statement we know that,

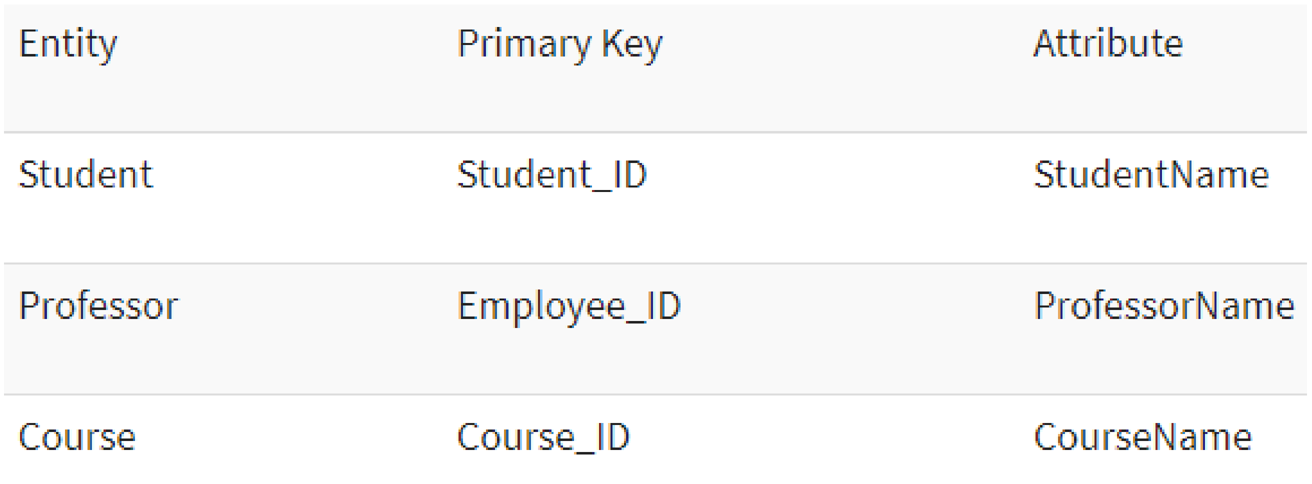
- A student can be assigned **multiple** courses
- A Professor can deliver only **one** course

Step 4) Identify Attributes

You need to study the files, forms, reports, data currently maintained by the organization to identify attributes. You can also conduct interviews with various stakeholders to identify entities. Initially, it's important to identify the attributes without mapping them to a particular entity.

Once, you have a list of Attributes, you need to map them to the identified entities. Ensure an attribute is to be paired with exactly one entity. If you think an attribute should belong to more than one entity, use a modifier to make it unique.

Once the mapping is done, identify the primary Keys. If a unique key is not readily available, create one.



Entity	Primary Key	Attribute
Student	Student_ID	StudentName
Professor	Employee_ID	ProfessorName
Course	Course_ID	CourseName

For Course Entity, attributes could be Duration, Credits, Assignments, etc. For the sake of ease we have considered just one attribute.

Step 5) Create the ERD

Best Practices for Developing Effective ER Diagrams

- Eliminate any redundant entities or relationships
- You need to make sure that all your entities and relationships are properly labeled
- There may be various valid approaches to an ER diagram. You need to make sure that the ER diagram supports all the data you need to store
- You should assure that each entity only appears a single time in the ER diagram
- Name every relationship, entity, and attribute are represented on your diagram
- Never connect relationships to each other
- You should use colors to highlight important portions of the ER diagram

Summary

- The ER model is a high-level data model diagram
- ER diagrams are a visual tool which is helpful to represent the ER model
- Entity relationship diagram displays the relationships of entity set stored in a database

- ER diagrams help you to define terms related to entity relationship modeling
- ER model is based on three basic concepts: Entities, Attributes & Relationships
- An entity can be place, person, object, event or a concept, which stores data in the database
- Relationship is nothing but an association among two or more entities
- A weak entity is a type of entity which doesn't have its key attribute
- It is a single-valued property of either an entity-type or a relationship-type
- It helps you to defines the numerical attributes of the relationship between two entities or entity sets
- ER- Diagram is a visual representation of data that describe how data is related to each other
- While Drawing ER diagram you need to make sure all your entities and relationships are properly labeled.

DBMS
Lecture 4: Keys in Relational Model

Keys

Keys are very important part of Relational database model. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table.

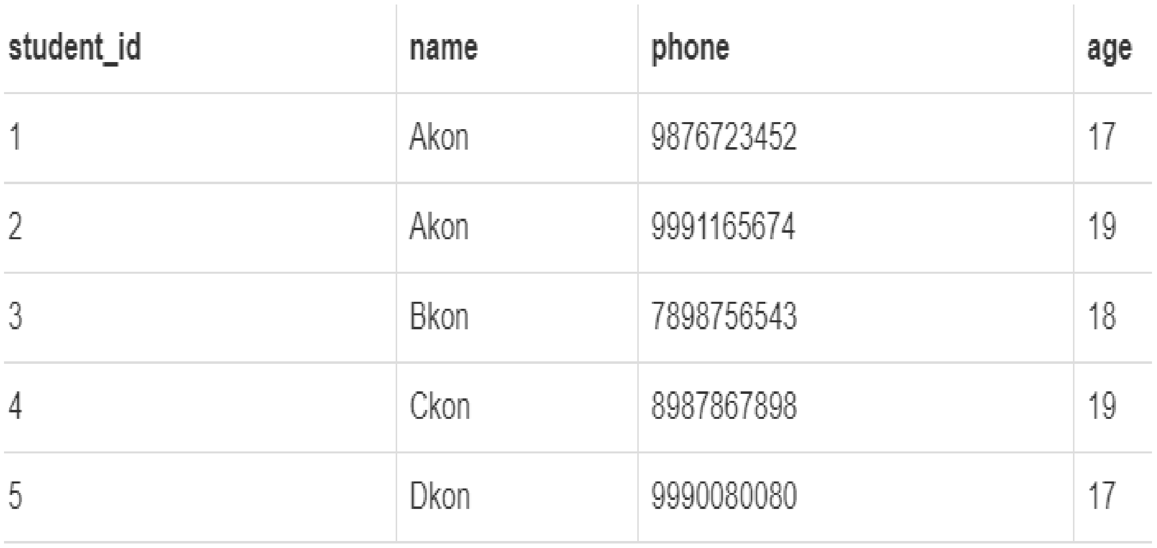
A Key can be a single attribute or a group of attributes, where the combination may act as a key.

Why we need a Key?

In real world applications, number of tables required for storing the data is huge, and the different tables are related to each other as well.

Also, tables store a lot of data in them. Tables generally extends to thousands of records stored in them, unsorted and unorganised.

Now to fetch any particular record from such dataset, you will have to apply some conditions, but what if there is duplicate data present and every time you try to fetch some data by applying certain condition, you get the wrong data. How many trials before you get the right data?



student_id	name	phone	age
1	Akon	9876723452	17
2	Akon	9991165674	19
3	Bkon	7898756543	18
4	Ckon	8987867898	19
5	Dkon	9990080080	17

To avoid all this, **Keys** are defined to easily identify any row of data in a table.

Let's try to understand about all the keys using a simple example.

Let's take a simple **Student** table, with fields `student_id`, `name`, `phone` and `age`.

Super Key

Super Key is defined as a set of attributes within a table that can uniquely identify each record within a table. Super Key is a superset of Candidate key.

In the table defined above super key would include `student_id`, `(student_id, name)`, `phone` etc.

Confused? The first one is pretty simple as `student_id` is unique for every row of data, hence it can be used to identity each row uniquely.

Next comes, `(student_id, name)`, now name of two students can be same, but their `student_id` can't be same hence this combination can also be a key.

Similarly, phone number for every student will be unique, hence again, `phone` can also be a key.

So they all are super keys.

Candidate Key

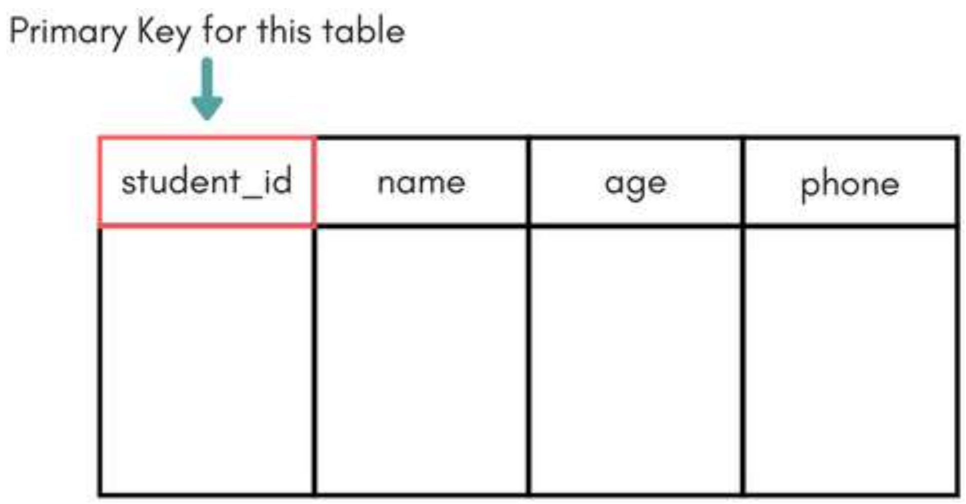
Candidate keys are defined as the minimal set of fields which can uniquely identify each record in a table. It is an attribute or a set of attributes that can act as a Primary Key for a table to uniquely identify each record in that table. There can be more than one candidate key.

In our example, `student_id` and `phone` both are candidate keys for table **Student**.

- A candiate key can never be NULL or empty. And its value should be unique.
- There can be more than one candidate keys for a table.
- A candidate key can be a combination of more than one columns(attributes).

Primary Key

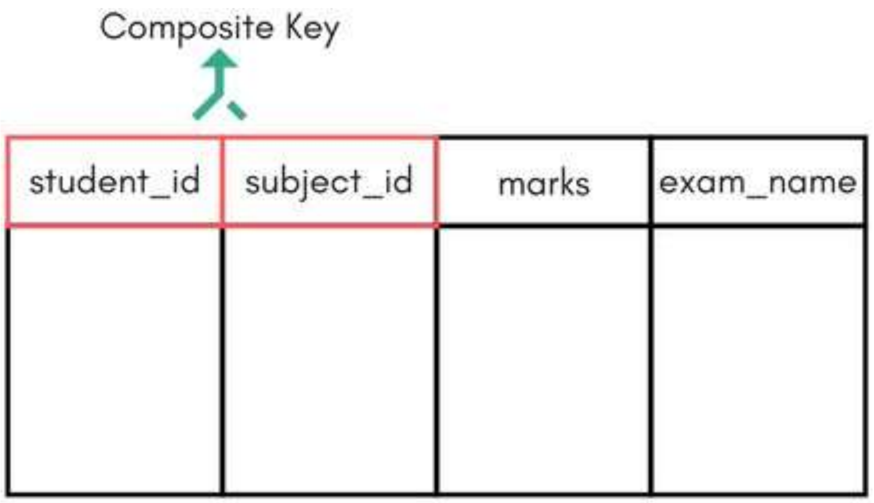
Primary key is a candidate key that is most appropriate to become the main key for any table. It is a key that can uniquely identify each record in a table.



For the table **Student** we can make the `student_id` column as the primary key.

Composite Key

Key that consists of two or more attributes that uniquely identify any record in a table is called **Composite key**. But the attributes which together form the **Composite key** are not a key independentely or individually.



Score Table - To save scores of the student for various subjects.

In the above picture we have a **Score** table which stores the marks scored by a student in a particular subject.

In this table `student_id` and `subject_id` together will form the primary key, hence it is a composite key.

Secondary or Alternative key

The candidate key which are not selected as primary key are known as secondary keys or alternative keys.

Non-key Attributes

Non-key attributes are the attributes or fields of a table, other than **candidate key** attributes/fields in a table.

Non-prime Attributes

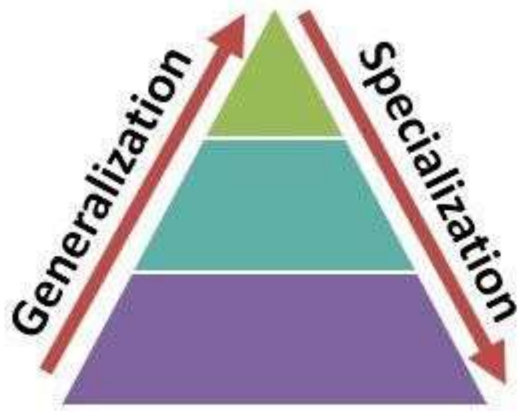
Non-prime Attributes are attributes other than **Primary Key attribute(s).**.

Extended ER Model concepts:

The ER Model has the power of expressing database entities in a conceptual hierarchical manner. As the hierarchy goes up, it generalizes the view of entities, and as we go deep in the hierarchy, it gives us the detail of every entity included.

Going up in this structure is called generalization, where entities are clubbed together to represent a more generalized view. For example, a particular student named Mira can be generalized along with all the students. The entity shall be a student, and further, the student is a person. The reverse is called specialization where a person is a student, and that student is Mira.

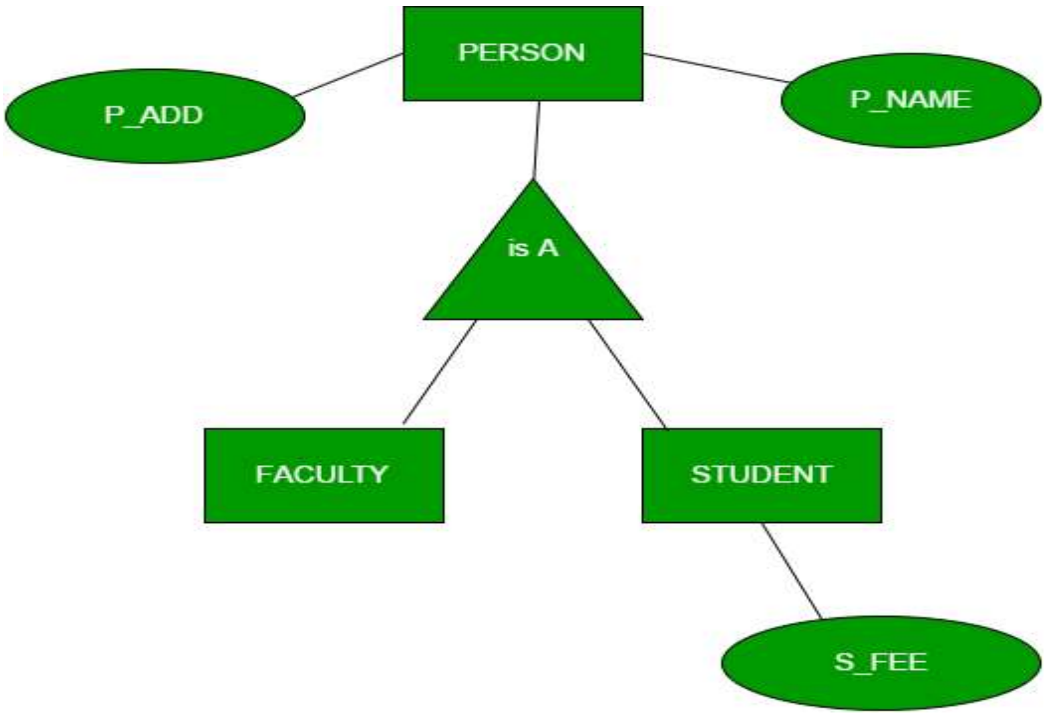
Generalization and Specialization both the terms are more common in Object Oriented Technology, and they are also used in the Database with the same features. Generalization occurs when we ignore the differences and acknowledge the similarities between lower entities or child classes or relations (tables in DBMS) to form a higher entity.



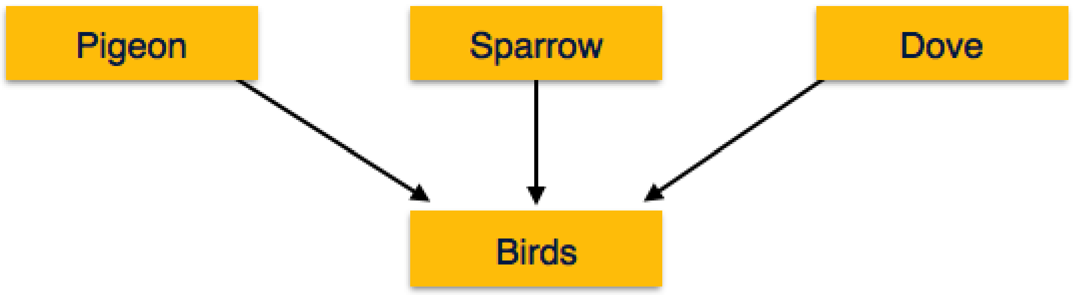
Generalization

- Generalization is like a bottom-up approach in which two or more entities of lower level combine to form a higher level entity if they have some attributes in common.
- In generalization, an entity of a higher level can also combine with the entities of the lower level to form a further higher level entity.
- Generalization is more like subclass and superclass system, but the only difference is the approach. Generalization uses the bottom-up approach.
- In generalization, entities are combined to form a more generalized entity, i.e., subclasses are combined to make a superclass.
- Generalization is the process of extracting common properties from a set of entities and creating a generalized entity from it.

For example, STUDENT and FACULTY can be generalized to a higher level entity called PERSON as shown in figure below. In this case, common attributes like P_NAME, P_ADD become part of higher entity (PERSON) and specialized attributes like S_FEE become part of specialized entity (STUDENT).



The process of generalizing entities, where the generalized entities contain the properties of all the generalized entities, is called generalization. In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For example, pigeon, house sparrow, crow and dove can all be generalized as Birds.

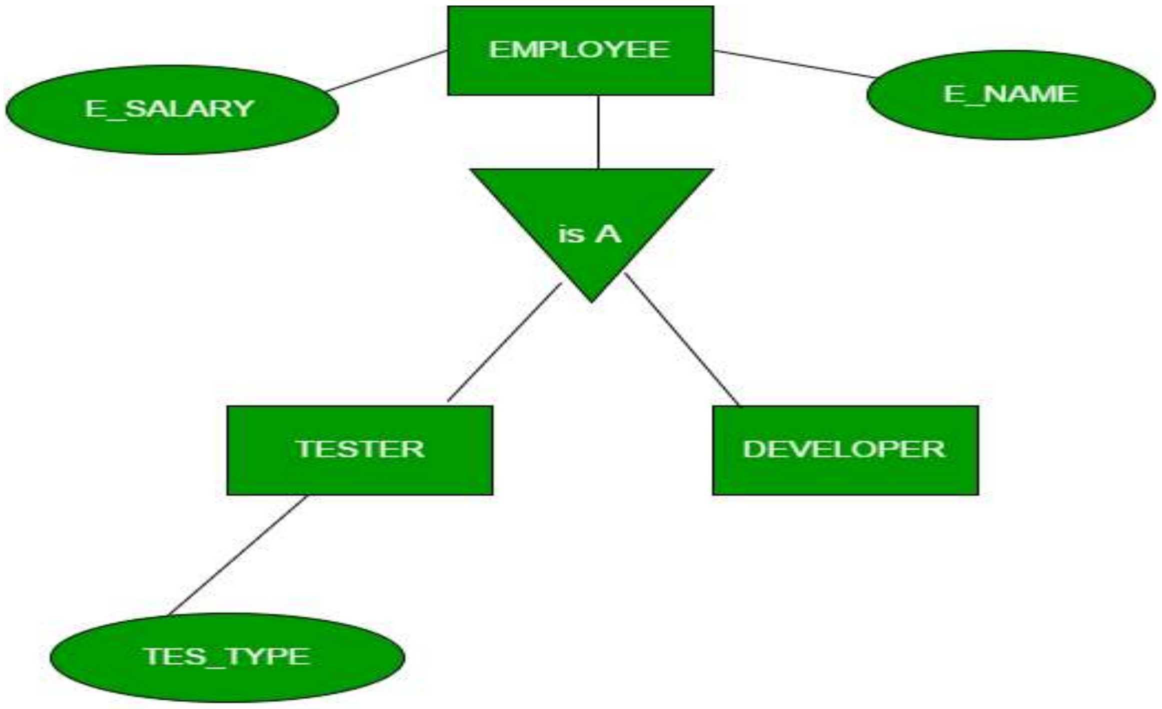


Specialization

- Specialization is a top-down approach, and it is opposite to Generalization. In specialization, one higher level entity can be broken down into two lower level entities.

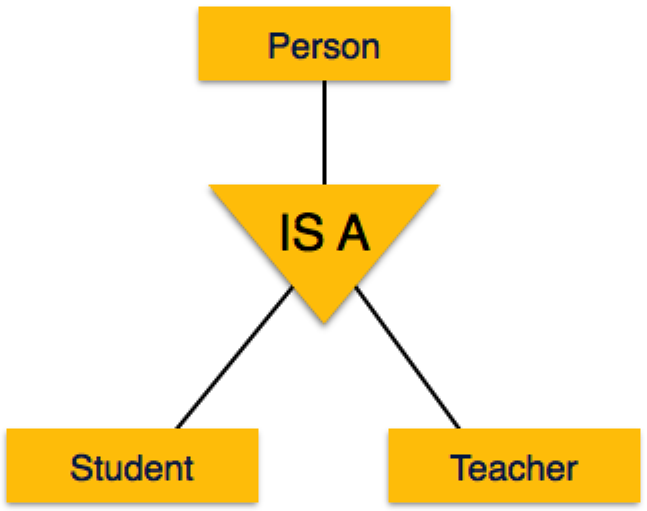
- Specialization is used to identify the subset of an entity set that shares some distinguishing characteristics.
- Normally, the superclass is defined first, the subclass and its related attributes are defined next, and relationship set are then added.
- In specialization, an entity is divided into sub-entities based on their characteristics. It is a top-down approach where higher level entity is specialized into two or more lower level entities.

For example, EMPLOYEE entity in an Employee management system can be specialized into DEVELOPER, TESTER etc. as shown in figure below. In this case, common attributes like E_NAME, E_SALARY etc. become part of higher entity (EMPLOYEE) and specialized attributes like TES_TYPE become part of specialized entity (TESTER).

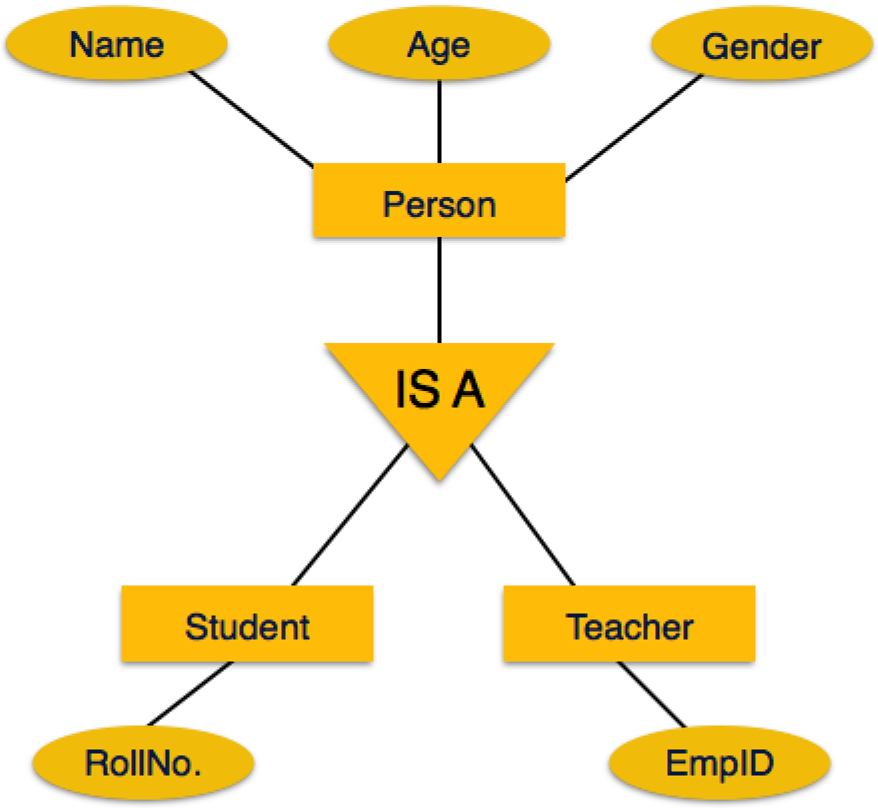


Specialization

Let us consider another example, an entity set 'Person'. A person has name, date of birth, gender, etc. These properties are common in all persons, human beings. But in a company, persons can be identified as employee, employer, customer, or vendor, based on what role they play in the company. Similarly, in a school database, persons can be specialized as teacher, student, or a staff, based on what role they play in school as entities.



Inheritance is an important feature of Generalization and Specialization. It allows lower-level entities to inherit the attributes of higher-level entities.



For example, the attributes of a Person class such as name, age, and gender can be inherited by lower-level entities such as Student or Teacher.

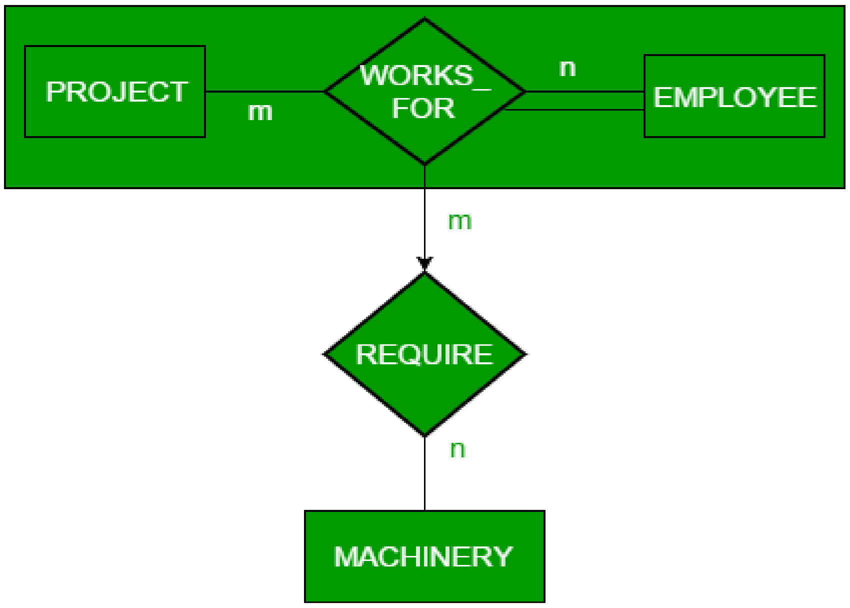
Aggregation:

Aggregration is a process when relation between two entities is treated as a single entity.

In aggregation, the relation between two entities is treated as a single entity. In aggregation, relationship with its corresponding entities is aggregated into a higher level entity.

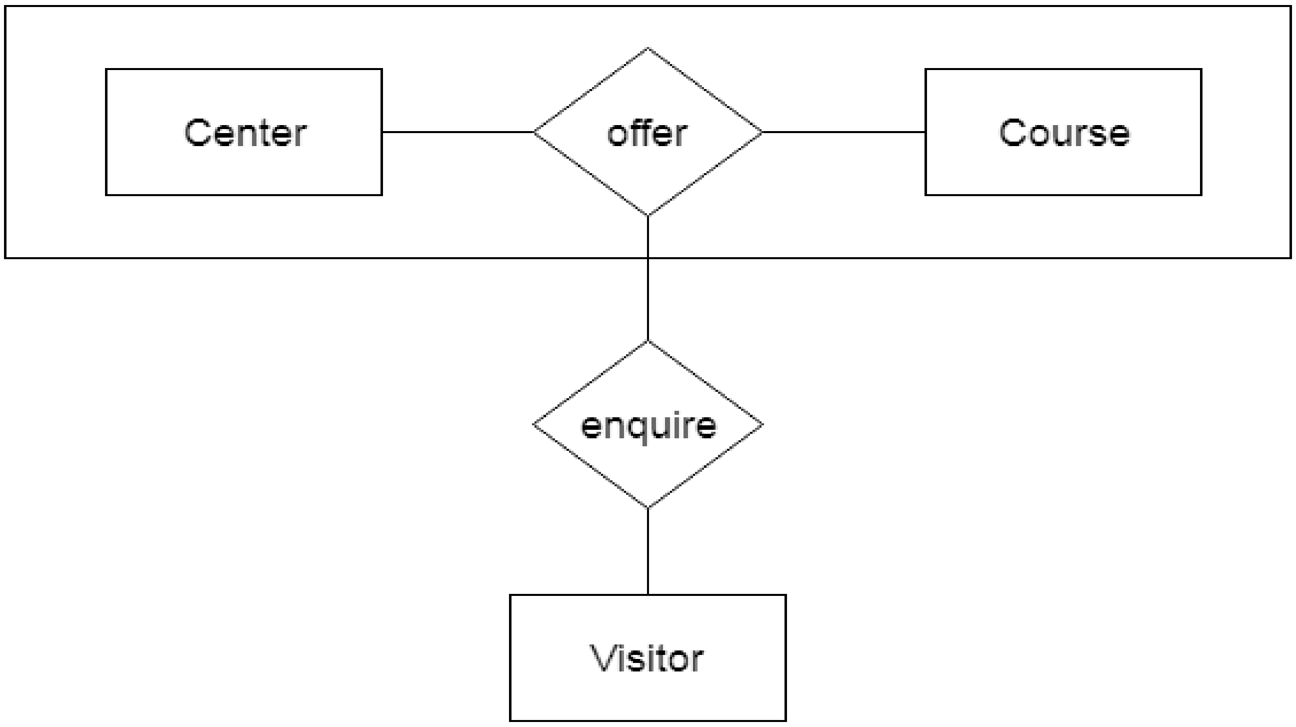
An ER diagram is not capable of representing relationship between an entity and a relationship which may be required in some scenarios. In those cases, a relationship with its corresponding entities is aggregated into a higher level entity.

For Example, Employee working for a project may require some machinery. So, REQUIRE relationship is needed between relationship WORKS_FOR and entity MACHINERY. Using aggregation, WORKS_FOR relationship with its entities EMPLOYEE and PROJECT is aggregated into single entity and relationship REQUIRE is created between aggregated entity and MACHINERY.

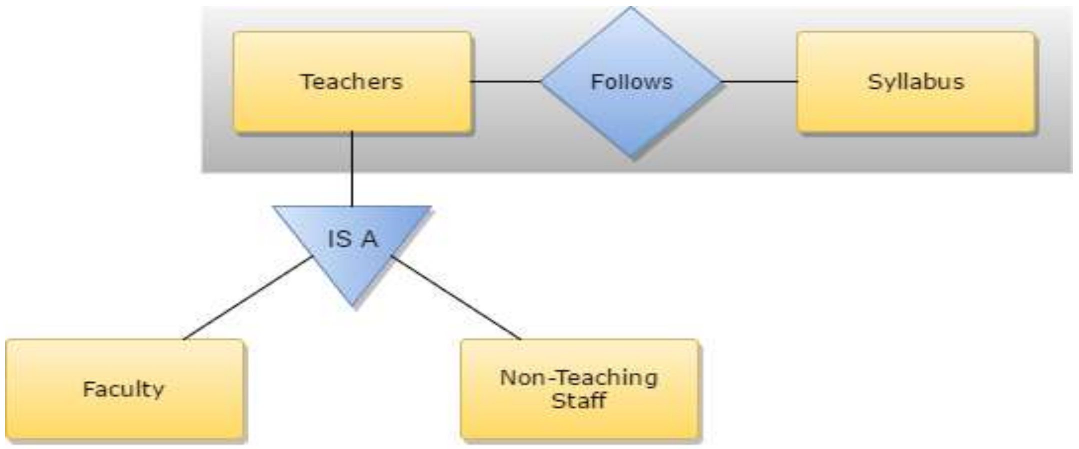


Aggregation

Another example: Center entity offers the Course entity act as a single entity in the relationship which is in a relationship with another entity visitor. In the real world, if a visitor visits a coaching center then he will never enquiry about the Course only or just about the Center instead he will ask the enquiry about both.



A relationship represents a connection between two entity types that are conceptually at the same level. Sometimes you may want to model a 'has-a,' 'is-a' or 'is-part-of' relationship, in which one entity represents a larger entity (the 'whole') that will consist of smaller entities (the 'parts'). This special kind of relationship is termed as an aggregation. Aggregation does not change the meaning of navigation and routing across the relationship between the whole and its parts. An example of aggregation is the 'Teacher' entity following the 'syllabus' entity act as a single entity in the relationship. In simple words, aggregation is a process where the relation between two entities is treated as a single entity.



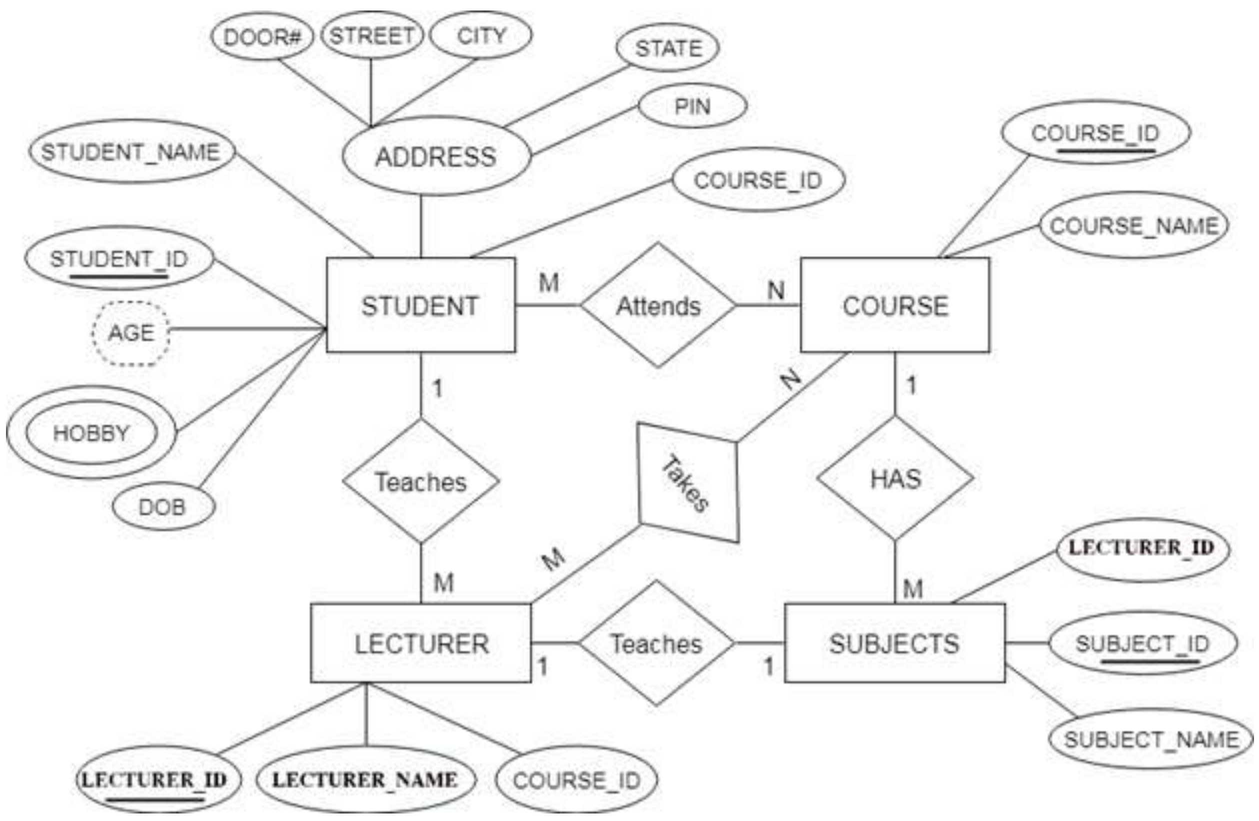
Specialization and Aggregation

Reduction of ER diagram to Tables:

The database can be represented using the notations, and these notations can be reduced to a collection of tables.

In the database, every entity set or relationship set can be represented in tabular form.

Let us consider the ER diagram below:



There are some points for converting the ER diagram to the table:

- **Entity type becomes a table.**

In the given ER diagram, LECTURE, STUDENT, SUBJECT and COURSE forms individual tables.

- **All single-valued attribute becomes a column for the table.**

In the STUDENT entity, STUDENT_NAME and STUDENT_ID form the column of STUDENT table. Similarly, COURSE_NAME and COURSE_ID form the column of COURSE table and so on.

- **A key attribute of the entity type represented by the primary key.**

In the given ER diagram, COURSE_ID, STUDENT_ID, SUBJECT_ID, and LECTURE_ID are the key attribute of the entity.

- **The multivalued attribute is represented by a separate table.**

In the student table, a hobby is a multivalued attribute. So it is not possible to represent multiple values in a single column of STUDENT table. Hence we create a table STUD_HOBBY with column name STUDENT_ID and HOBBY. Using both the column, we create a composite key.

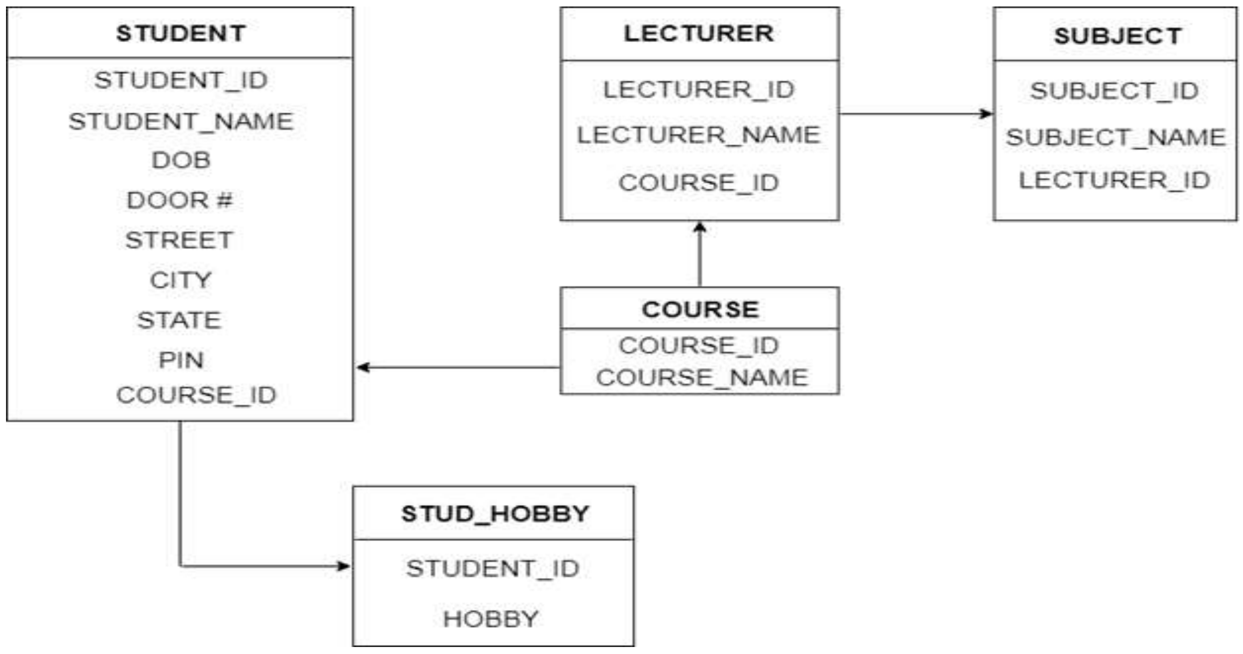
- **Composite attribute represented by components.**

In the given ER diagram, student address is a composite attribute. It contains CITY, PIN, DOOR#, STREET, and STATE. In the STUDENT table, these attributes can merge as an individual column.

- **Derived attributes are not considered in the table.**

In the STUDENT table, Age is the derived attribute. It can be calculated at any point of time by calculating the difference between current date and Date of Birth.

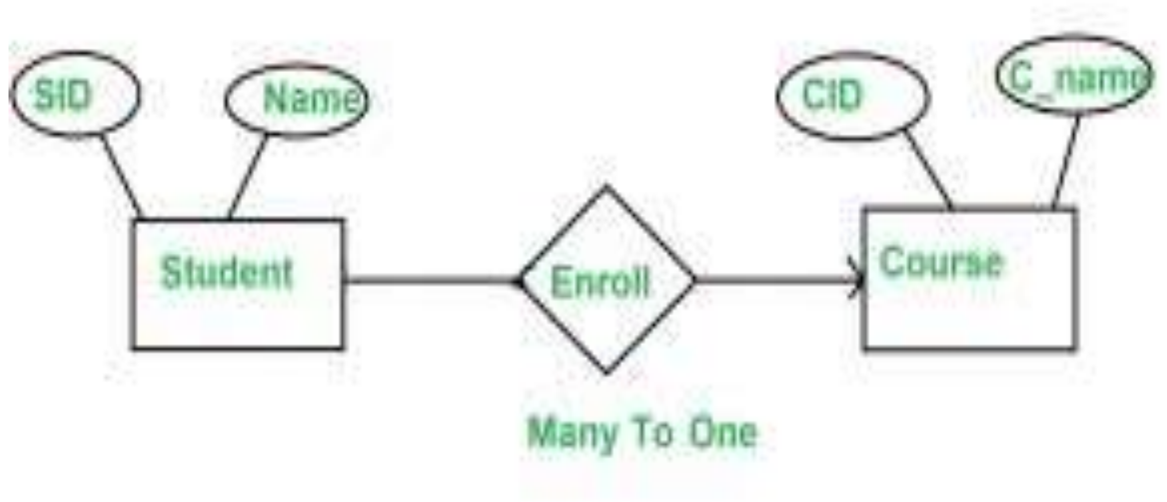
Using these rules, we can convert the ER diagram to tables and columns and assign the mapping between the tables. Table structure for the given ER diagram is as below:



Relationship of higher degree and mapping constraints:

Entity Relationship (ER) Diagram is diagrammatic representation of data in databases, it shows how data is related.

1) When there is One to Many cardinality in ER diagram.
For example, a student can be enrolled only in one course, but a course can be enrolled by many students



For Student(SID, Name), SID is the primary key. For Course (CID, C_name), CID is the primary key

Student	Course
(SID Name)	(CID C_name)
----------------	--------------------
1 A	c1 Z
2 B	c2 Y
3 C	c3 X
4 D	

Enroll

(SID CID)

1	C1
2	C1
3	c3
4	C2

Let us consider the primary key for Enroll SID or CID or combined. We can’t have CID as primary key as you can see in enroll for the same CID we have multiples SID. (SID , CID) can distinguish table uniquely, but it is not minimum. So SID is the primary key for the relation enroll.

For above ER diagram, we considered three tables in database

Student

Enroll

Course

But we can combine Student and Enroll table renamed as Student_enroll.

Student_Enroll

(SID Name   CID)

1	A	c1
2	B	c1
3	C	c3
4	D	c2

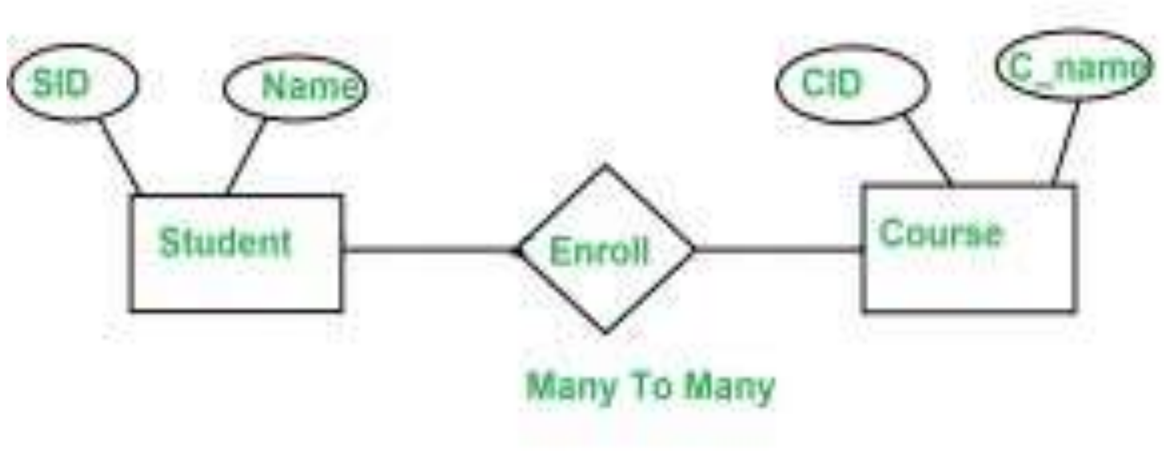
Student and enroll tables are merged now .

So require minimum two DBMS tables for Student_enroll and Course.

Note: In One to Many relationship we can have minimum two tables.

2. When there is Many to Many cardinality in ER Diagram.

Let us consider above example with the change that now student can also enroll more than 1 course.



Student

SID	Name
1	A
2	B
3	C
4	D

Course

CID	C_name
c1	Z
c2	Y
c3	X

Enroll

SID	CID
1	C1

1	C2
2	C1
2	C2
3	c3
4	C2

Now, same question what is the primary key of Enroll relation, if we carefully analyse the Enroll primary key for Enroll
table is (SID , CID).

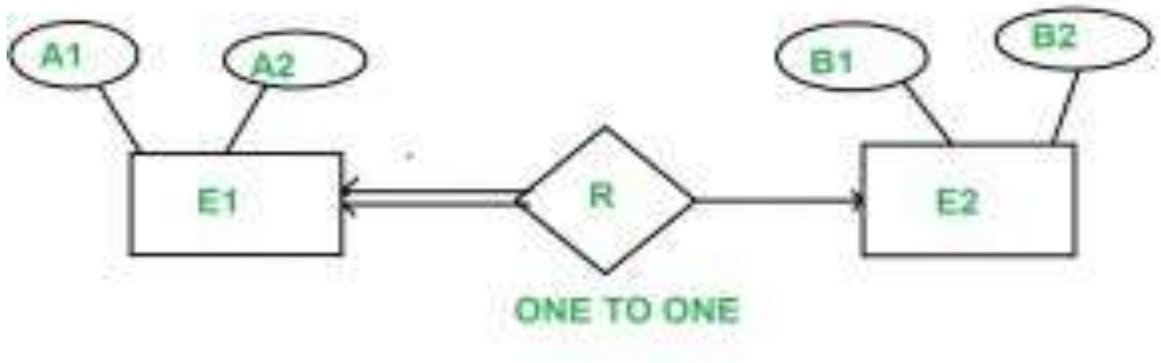
But in this case we can't merge Enroll table with any one of Student and Course. If we try to merge Enroll with any one of the Student and Course it will create redundant data.

Note: Minimum three tables are required in Many to Many relationship.

3. One to One Relationship

There are two possibilities
A) If we have One to One relationship and we have total participation at at-least one end.

For example, consider the below ER diagram.



A1 and B1 are primary keys of E1 and E2 respectively.

In the above Diagram we have total participation at E1 end.

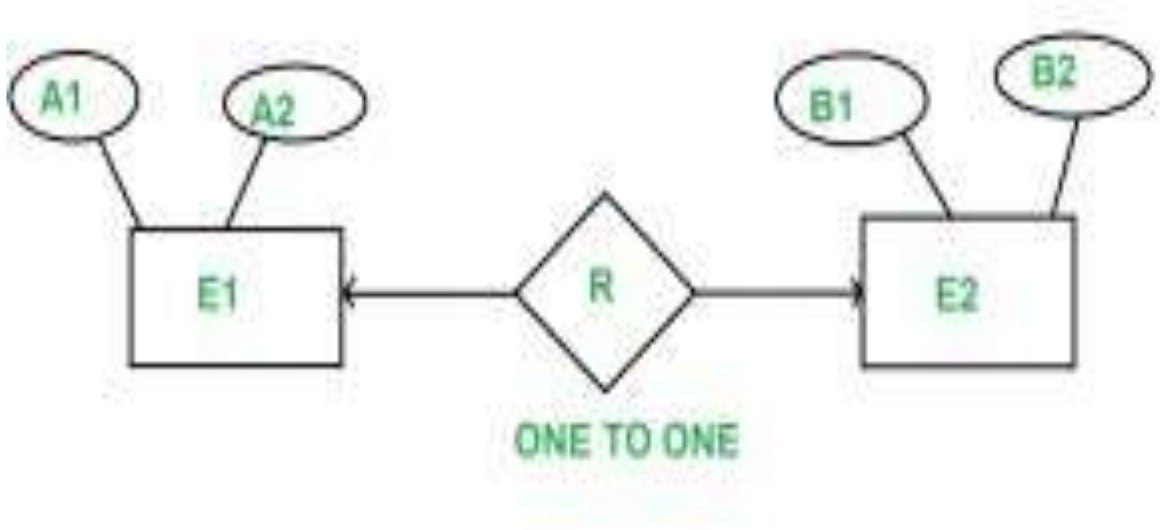
Only a single table is required in this case having primary key of E1 as its primary key.

Since E1 is in total participation, each entry in E1 is related to only one entry in E2, but not all entries in E2 are related to an entry in E1.

The primary key of E1 should be allowed as the primary key of the reduced table, since if the primary key of E2 is used, it might have null values for many of its entries in the reduced table.

Note: Only 1 table required.

B) One to One relationship with no total participation.



A1 and B1 are primary keys of E1 and E2 respectively.

Primary key of R can be A1 or B1, but we can’t still combine all the three table into one. if we do, so some entries in combined table may have NULL entries. So idea of merging all three table into one is not good.

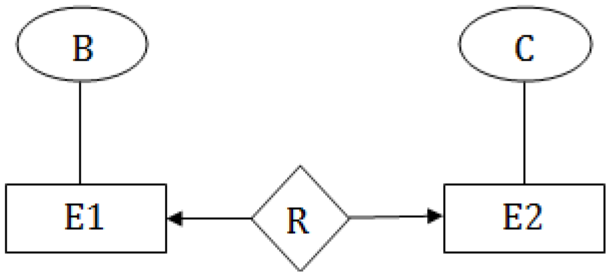
But we can merge R into E1 or E2. So minimum 2 tables are required.

Mapping Constraints:

- A mapping constraint is a data constraint that expresses the number of entities to which another entity can be related via a relationship set.
- It is most useful in describing the relationship sets that involve more than two entity sets.
- For binary relationship set R on an entity set A and B, there are four possible mapping cardinalities. These are as follows:
    1. One to one (1:1)
    2. One to many (1:M)
    3. Many to one (M:1)
    4. Many to many (M:M)

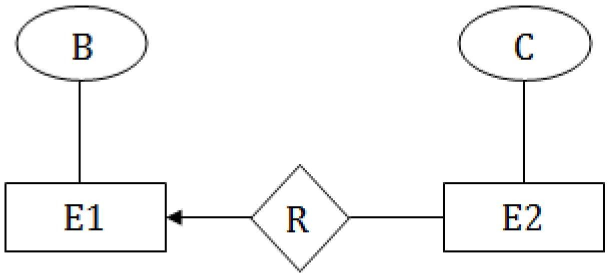
One-to-one

In one-to-one mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with at most one entity in E1.



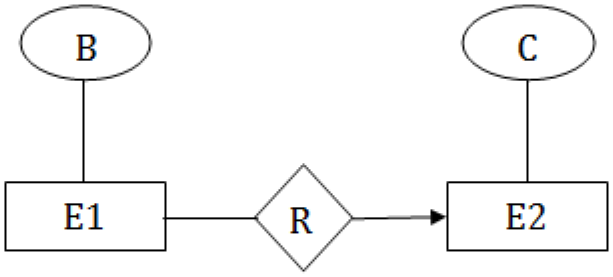
One-to-many

In one-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with at most one entity in E1.



Many-to-one

In one-to-many mapping, an entity in E1 is associated with at most one entity in E2, and an entity in E2 is associated with any number of entities in E1.



Many-to-many

In many-to-many mapping, an entity in E1 is associated with any number of entities in E2, and an entity in E2 is associated with any number of entities in E1.