Course Contents

Unit I : Introduction **(07 Hours)**

Introduction to Database Management Systems, Purpose of Database Systems, Database-System Applications, View of Data, Database Languages, Database System Structure, Data Models, Database Design and ER Model : Entity, Attributes, Relationships, Constraints, Keys, Design Process, Entity Relationship Model, ER Diagram, Design Issues, Extended E-R Features, converting E-R & EER diagram into tables. **(Refer Chapter 1)**

Unit II : SQL AND PL/SQL **(07 Hours)**

SQL : Characteristics and advantages, SQL Data Types and Literals, DDL, DML, DCL, TCL, SQL Operators, Tables : Creating, Modifying, Deleting, Views: Creating, Dropping, Updating using Views, Indexes, SQL DML Queries : SELECT Query and clauses, Set Operations, Predicates and Joins, Set membership, Tuple Variables, Set comparison, Ordering of Tuples, Aggregate Functions, Nested Queries, Database Modification using SQL Insert, Update and Delete Queries. **PL/SQL :** concept of Stored Procedures & Functions, Cursors, Triggers, Assertions, roles and privileges , Embedded SQL, Dynamic SQL. **(Refer Chapter 2)**

Unit III : Relational Database Design **(08 Hours)**

Relational Model : Basic concepts, Attributes and Domains, CODD's Rules, Relational Integrity: Domain, Referential Integrities, Enterprise Constraints, Database Design : Features of Good Relational Designs, Normalization, Atomic Domains and First Normal Form, Decomposition using Functional Dependencies, Algorithms for Decomposition, 2NF, 3NF, BCNF, Modeling Temporal Data. **(Refer Chapter 3)**

Unit IV : Database Transactions and Query Processing **(08 Hours)**

Basic concept of a Transaction, Transaction Management, Properties of Transactions, Concept of Schedule, Serial Schedule, Serializability : Conflict and View, Cascaded Aborts, Recoverable and Non-recoverable Schedules, Concurrency Control : Need, Locking Methods, Deadlocks, Time-stamping Methods, Recovery methods : Shadow-Paging and Log-Based Recovery, Checkpoints, Query Processing, Query Optimization, Performance Tuning. **(Refer Chapter 4)**

Unit V : Parallel and Distributed Databases **(07 Hours)**

Introduction to Database Architectures: Multi-user DBMS Architectures, Case study- Oracle Architecture. **Parallel Databases :** Speedup and Scale up, Architectures of Parallel Databases.

Distributed Databases : Architecture of Distributed Databases, Distributed Database Design, Distributed Data Storage, Distributed Transaction : Basics, Failure modes, Commit Protocols, Concurrency Control in Distributed Database. **(Refer Chapter 5)**

Unit VI : NoSQL Database **(08 Hours)**

Introduction to NoSQL Database, Types and examples of NoSQL Database - Key value store, document store, graph, Performance, Structured verses unstructured data, Distributed Database Model, CAP theorem and BASE Properties, Comparative study of SQL and NoSQL, NoSQL Data Models, Case Study-unstructured data from social media. Introduction to Big Data, HADOOP : HDFS, MapReduce. **(Refer Chapter 6)**

UNIT I

Chapter 1 : Introduction to DBMS 1-1 to 1-36

Syllabus : Introduction to Database Management Systems, Purpose of Database Systems, Database-System Applications, View of Data, Database Languages, Database System Structure, Data Models, Database Design and ER Model : Entity, Attributes, Relationships, Constraints, Keys, Design Process, Entity Relationship Model, ER Diagram, Design Issues, Extended E-R Features, Converting ER & EER diagram into tables.

UNIT II

Chapter 2 : SQL and PL/SQL 2-1 to 2-46

Syllabus : **SQL** : Characteristics and advantages, SQL Data Types and Literals, DDL, DML, DCL, TCL, SQL Operators, Tables : Creating, Modifying, Deleting, Views : Creating, Dropping, Updating using Views, Indexes, SQL DML Queries: SELECT Query and clauses, Set Operations, Predicates and Joins, Set membership, Tuple Variables, Set comparison, Ordering of Tuples, Aggregate Functions, Nested Queries, Database Modification using SQL Insert, Update and Delete Queries.

PL/SQL: Concept of Stored Procedures & Functions, Cursors, Triggers, Assertions, Roles and privileges, Embedded SQL, Dynamic SQL.

UNIT III

Chapter 3 : Relational Database Design 3-1 to 3-24

Syllabus : Relational Model : Basic concepts, Attributes and Domains, Codd's Rules. Relational Integrity: Domain, Referential Integrities, Enterprise Constraints. Database Design: Features of Good Relational Designs. Normalization, Atomic Domains and First Normal Form, Decomposition using Functional Dependencies, Algorithms for Decomposition, 2NF, 3NF, BCNF. Modeling Temporal Data.

UNIT IV

Chapter 4 : Database Transactions and Query Processing 4-1 to 4-24

Syllabus : Basic concept of a transaction, Transaction Management, Properties of Transaction, Concept of Schedule, Serial schedule, Serializability : Conflict and View, Cascaded Aborts, Recoverable and Non-recoverable schedules, Concurrency Control : Need, Locking Methods, Deadlocks, Time-Stamping Methods, Recovery Methods : Shadow-paging and Log-Based Recovery, Checkpoints, Query Processing, Query Optimization, Performance Tuning.

UNIT V

Syllabus : Introduction to Database architectures : Multi-user DBMS architectures, Case Study- Oracle Architecture, Parallel Databases : Speedup and Scaleup, Architectures of Parallel Databases. Distributed Databases : Architecture of Distributed Databases, Distributed Database Design, Distributed Data Storage Distributed Transaction : Basics, Failure modes, Commit Protocols, Concurrency Control In Distributed Database.

UNIT VI

Syllabus : Introduction to NoSQL Database, Types and examples of NoSQL Database- Key value store, document store, graph, Performance, Structured verses unstructured data, Distributed Database Model, CAP theorem and BASE Properties, Comparative study of SQL and NoSQL, NoSQL Data Models, Case Study - unstructured data from social media, ntroduction to Big Data, Hadoop : HDFS, MapReduce.

❑❑❑

UNIT I

CHAPTER 1

Introduction to DBMS

Syllabus

- Introduction to Database Management Systems
- Purpose of Database Systems
- Database-System Applications
- View of Data
- Database Languages
- Database System Structure
- Data Models
- Database Design and ER Model : Entity, Attributes, Relationships, Constraints, Keys, Design Process, Entity Relationship Model
- ER Diagram
- Design Issues
- Extended E-R Features
- Converting ER & EER diagram into tables

Syllabus Topic : Introduction to Database Management Systems

1.1 Introduction to Database Management Systems

- **Data :** Data is the information which has been translated into a form that is more convenient to process or move.
- **Database :** The collection of related data is termed as Database which is organized in such a way that it can be easily retrieved and managed.
- **Database Management System**
  - A Database Management System (DBMS) is system software which manages the data. It can perform various tasks like creation, retrieval, insertion, modification and deletion of data to manage it in a systematic way as per requirement.
  - Database systems are designed to manage large amount of data by providing security from accidental crash of system and unauthorized access. DBMS provides convenient and efficient environment which used to handle the data.

Syllabus Topic : Purpose of Database Systems

1.2 Purpose of Database Systems

- Programming languages like Java, .Net are used to develop customized software's. Every software or application has its data to be stored permanently.
- Programming languages cannot store data permanently. For this purpose we have to use the Database Management System. The DBMS plays a significant role in storing and managing data.
- In an application we store data in DBMS and for operations like insertion, modification or deletion we write code in programming languages i.e. software is usually created with the help of both Programming language and Database.
- When the application is executed on client side, the client or user interacts with interface of application which is created in programming language.

- The database always remains backside and do not come in front of the user. Hence the database is known as backend while programming language is termed as frontend.
- To understand the purpose or need of database system, we need to study the previous option to store data which is called as File Processing System.

1.2.1 File Processing System

- In our day to day life, number of times we need to store data in such way that it should be easily accessible whenever required.
- The data may be of bank transaction details, daily expenses, employee details, product details etc. Before computers such data was stored with the help of papers.
- After invention of computers, it becomes easy to store data with the help of files. In the early days, database applications were built on top of file systems.
- Traditional File Processing System is a computer based system in which all the information is stored in various computer files.
- It stores data in a systematic way that the different departments of an organization can store their data in set of files which helps to manage and differentiate the data.
- Initially the Traditional File Processing System seems to be useful but as the requirement of data processing and the size of data increases, the drawbacks of this system comes in picture.

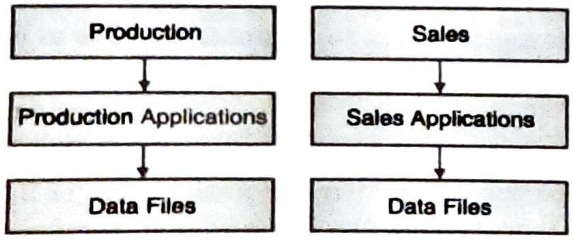


Fig. 1.2.1

1.2.2 Drawbacks of Traditional File Processing Systems

Consider an example of IT company database system where the data of all the employees is stored. In a professional IT firm the tasks are always done as teamwork. Different teams are assigned for different projects. Hence the data is stored in different files so as to differentiate it.

1. Data Redundancy

- Sometimes as per requirement same data may be stored in multiple files. Consider an employee having record in both Employee and Team files. The name and address of employee is stored in both of these files.
- Means the data get duplicated. If such data increases, it leads to higher storage and access cost. This duplication of data in various files is termed as data redundancy.
- In traditional file system, it is very difficult to avoid this data redundancy.

2. Data Inconsistency

- When data is to be updated the data redundancy may lead to data inconsistency. Data inconsistency occurs when data is not updated in all the files simultaneously.
- For example if the designation of employee get changed, then the respective changes should be made in both Employee and Team file. If for some reason, it is not done, then it leads to data inconsistency.
- Because for the sample employee, we may get different information which may create problems in the processing of data.

3. Limited Data Sharing

- It is difficult to share data in traditional file system. Each application has its own private files and users have little choice to share the data outside their own applications.
- To share data, we have to write complex programs.

4. Difficulty in Accessing Data

- The need of data access varies time to time. Means different types of information is needed at different situations.
- For example just consider that we want to retrieve the data of employees who do not have taken any leave throughout the quarter. In such case we have two options.
- We can access the data by manual method or we have to write an application program to

retrieve such customized data. Both the options are not convenient as both of them leads to wastage of time.

- If we do it, then also it may be possible that after some time we may require data with some another filter criteria. The data retrieval for customized information becomes difficult because the conventional files system does not provide any efficient and convenient way to retrieve the data.

5. Data Dependence

- In the files, data is stored in some specific format tab, semicolon or comma. If the format of any of the file is changed, then we have to make changes in program which processes the file.
- But sometimes there may be many programs related with the same file. In such case changes in all such programs should be done. Missing changes in single program may lead to failure of whole application.

6. Poor Data Control

- The Traditional File System does not have centralized data control; the data is de-centralized or distributed. In this system the same field may have different names in files of different departments of an organization.
- This situation may lead to different meaning of same data field in different context or same meaning for different fields. This causes poor data control.

7. Problem of Security

- It is very difficult to enforce security checks and access rights in a traditional file system. To the file we can set password protection.
- But what if we have to give access to only few records in the file? For example, in our database system, the project manager should be able to see all the data regarding teams under him.
- The team leader should be able to see data about his specific team. But payment details of one project manager should not be accessible to his team members or any another project manager. In the conventional file processing system, the application programs are added in ad hoc manner (for specific purpose) which makes it difficult to enforce security constraints.

8. Concurrency Problems

- Concurrency means access of same data by multiple users at the same time. This is very important aspect as it leads to increase in performance of a system and faster response. Many advanced systems allow the concurrent access and manipulation of data.
- For example, in our system, consider a record of an employee is accessed and updated by multiple users simultaneously at a time. This may lead to inconsistency of data, if the concurrency is not controlled in a proper manner.
- In another example if multiple transactions are make updations on a same bank account, then it may show incorrect balance, if any other transactions try to access balance amount in between.
- It is very difficult to implement concurrency control mechanism on file processing system, which leads to incorrect or wrong data retrieval.

9. Poor Data Modelling of Real World

- It is difficult for File Processing System to represent the complex data and interfile relationships. This results in poor data modelling properties.
- That means the real world applications are difficult to implement using File Processing System.

10. Data Isolation

- It is difficult to store the entire data in a single file. It is distributed in different files as per the category.
- These files may be in different formats because of which it becomes difficult to write application programs to access the desired data from these files.

11. Integrity Problems

- Every enterprise has its own constraints while maintain data in the files. Suppose in employee files the employee ID must start with 'E'. Such constraints can be added while writing application programs.

- But later on if any new constraints are introduced by the enterprise, and then it becomes difficult to add these constraints again. The File processing system does not provide any functionality to handle this situation.

12. Atomicity Problem

- Failure in a computer system may occur any time. When failure occurs, if any transaction is in its midway then it may lead to some incorrect data updation in the system.
- Consider another example of bank transaction where some amount is transferred from account A to account B. Initially the balance from account A is accessed and debited by Rs. 1000. Then we are going to credit it in account B. But before that system crash occurs which halts the transaction.
- Now this situation leads to incorrect data updation in the balance of account A. In file processing system, it very difficult to handle such situation to maintain the atomicity of database. The purpose of Database Management System is to solve all these problems and give functionality to store and manage data in efficient and convenient way.

1.2.3 Advantages of Database Management System

1. Controlling Data Redundancy

- In File Processing System the different applications has separate files for data storage. In this case, the duplicated copies of the same data are created at many places.
- In DBMS, all the data of an organization is integrated into a single database.
- The data is recorded at only one place in the database and it is not duplicated. For example, the Employee file and the Team file contain several items that are identical.
- When they are converted into database, the data is integrated into a single database so that multiple copies of the same data are reduced to-single copy.
- Controlling the data redundancy helps to save storage space. Similarly, it is useful for retrieving data from database using queries.

2. Data Consistency

- The data consistency is obtained by controlling the data redundancy, If a data item appears only once, any update to its value has to be performed only once and the updated value is immediately available to all users.
- For example if there is change in designation of employee, then the changes are made in single centralized file which is available to all the users.

3. Sharing of Data

- In DBMS, data can be easily shared by different applications. The database administrator manages the data and gives rights to users to access the data.
- Multiple users can be authorized to access the same data simultaneously. The remote users can also share same data.

4. Data Independence

- In DBMS we can completely separate the data structure of database and programs or applications which are used to access the data.
- This is called as data independence. If any changes are made in structure of database then there is no need to make changes in the programs. For example you can modify the size or data type of a data items (fields of a database table) without making any change in application program.

5. Data Control

- The DBMS provides centralized data storage. Hence keeping control on data is very much easy as compared to Traditional File Processing System.
- As data is common for all the application, no possibility of any confusion or complication.

6. Security

- In DBMS the different users can have different levels of access to data based on their roles. In the college database, students will have access to their own data only, while their teachers will have access to data of all the students whom they are teaching.

- Class teacher will be able to see the reports of all the students in that class, but not other classes. The principal will have access to entire data.
- Similarly, in a banking system, individual operator and clerk will have limited access to the data while the bank manager can access the entire data.
- All these levels of security and access are not allowed in file system.

7. Control over Concurrency

- In a computer file-based system, if multiple users are accessing data simultaneously, it is possible that it may lead to some irrelevant data generation. For example, if both users attempt to perform update operation on the same record, then one may overwrite the values recorded by the other.
- Most database management systems have sub-systems to control the concurrency so that transactions are always recorded with accuracy.

8. Data Modelling of Real World

- The DBMS has many functionalities are provided to represent the complex data and interfile relationships.
- This helps to map the database with real world applications.

1.2.4 Disadvantages of Database Management System

1. Increased Costs

- To install Database Systems, we require standard software and hardware. Also to handle the Database System, highly skilled personnel are required.
- The cost of maintaining the software, hardware, and personnel required to operate and manage the database system is more.
- The cost of training, license, and regulation compliance also increases the overall expenses.

2. Complexity

- Sometimes because of higher functionality expectations, the design of Database may become very complex.
- To utilize such database with complete efficiency, all the stakeholders like database designers, developers, database administrators and end-users must understand the functionality.
- Failure in understanding the system can lead to wrong design decisions, because of which serious consequences for an organization may occur.

3. Size

The DBMS becomes extremely large piece of software because of the complexity of functionality occupying large amount of disk space and requiring substantial amounts of memory to run efficiently.

4. Frequent Upgrade/Replacement Cycles

- New functionalities are often added into DBMS by their vendors. These new features often come bundled in new upgraded versions of the same software.
- Sometimes these versions require hardware upgrades which increases expenses. Also work to train database users and administrators to properly use and manage the new features get increased.

5. Higher Impact of a Failure

- The DBMS is placed at centralized location which increases the vulnerability of the system.
- That means the DBMS may get attacked and harmed. Since all users and applications rely on the centralized database, the failure of any component can bring operations to a halt.

6. Performance

- Usually, a File Based system is written for a specific application. Hence, the performance is generally very good.
- While the DBMS is written to be more general, to cater for multiple applications rather than any specific one.
- Because of which some applications may no run as fast as they used to.
- There are number of advantages of DBM over File System.

1.2.5 Difference between File Processing and DBMS

SPPU - Dec. 13, May 14, Dec. 15

University Questions

Q. Explain significant difference between File Processing and DBMS. **(Dec. 2013, 6 Marks)**

Q. Explain the advantages of DBMS over normal file system in detail. **(May 2014, 8 Marks)**

Q. List significant difference between File Processing and DBMS. **(Dec. 2015, 5 Marks)**

Sr. No.	File Processing System	Database Management System
1.	Duplicate data may exist in multiple files which lead to data redundancy.	The data is integrated into a single database which avoids data redundancy.
2.	Data inconsistency occurs when data is not updated in all the files simultaneously.	The data consistency is obtained by controlling the data redundancy
3.	It is difficult to share data in traditional file system.	In DBMS, data can be easily shared by different applications.
4.	In the files, data is stored in specific format. If the format of any of the file is changed, then we have to make changes in program which processes the file.	In DBMS we can completely separate the data structure of database and programs or applications which are used to access the data.
5.	The Traditional File System does not have centralized data control; the data is de-centralized or distributed.	The DBMS provides centralized data storage. Hence keeping control on data is very much easy.
6.	It is very difficult to enforce security checks and access rights in a traditional file system.	In DBMS the different used can have different levels of access to data based on their roles which provides strong security to data.
7.	Concurrency problems means updation of same data by multiple users may generate irrelevant results.	DBMS have sub-systems to control the concurrency.
8.	It is difficult for File Processing System to represent the complex data and interfile relationships. This results in poor data modelling.	The DBMS has many functionalities are provided to represent the complex data and interfile relationships. This helps to map the database with real world applications.

Syllabus Topic : Database-System Applications

1.3 Database-System Applications

For any enterprise, its data is very important which helps to manage the business as well as decide some strategies to survive and grow the business in this competitive world. A Database Management System is a computerized record-keeping system. It works as a container for collection of computerized data files.

The overall purpose of DBMS is to provide functionality to the users to create, store, retrieve and update the information contained in the database as per requirement. Information can be of an individual or an organization.

Databases touch all aspects of our lives. Some of the major **areas of application** are as follows :

- **Telecom :** In Telecom sector database is maintained to keep track of the information about calls made, customer details, network usage etc. Without using database systems it is difficult to maintain such huge amount of data which keeps track of every second.
- **Banking :** In banking sector the data related to transactions of customers is very huge. Such data can be comfortably stored in the Database Management System. The DBMS is also used for storing customer personal information, tracking day to day credit and debit transactions, generating bank statements etc.

- **Industry :** In industry database management plays an important role. In departments like production, sales or account it is very essential to store the data in systematic format. For example in production department the data about manufactured products is stored which is very useful for sales department to keep track of orders.
- **E-commerce :** There are number of online shopping websites such as ebay, Flipkart Amazon, etc. These sites store the information about various products, user addresses, their preferences and credit details. It can be implemented using Database Management System only.
- **Airlines :** In airlines, the data of ticket booking, flight schedule, employee details, and customer details is very huge. Such data can be managed using the Database Management System.
- **Education System :** In education sector, the schools, colleges, private institutes have to store data of students, teachers, exam schedules, accounts etc which can be handles by Database Management System.
- **Railway Reservation System :** Database Management System helps in keeping record of ticket booking, departure and arrival status of train etc.
- **Library Management System :** In the library there are thousands of books which make it very difficult to keep record of all the books in a register. The DBMS can be used to maintain this information about book issue dates, names of the books, authors and availability of the book.
- **Social Media Sites :** The social media websites are used to share our views and connect with our friends. Daily millions of users signed up for these social media accounts like Facebook, Whatsapp, Twitter, and Google plus. The data of these millions of users and their chats can store using Database Management System.

Syllabus Topic : View Of Data

1.4 View of Data

As we have seen a data base system is collection of related data and system software which manages the data. The data is generally stored in a detailed and complex manner. It is important to provide an abstract view of data to the user.

To understand the view of data, first we have to learn the concept of abstraction.

1.4.1 Abstraction

- Abstraction is an important feature of Database Management System. Extracting the important data by ignoring the remaining irrelevant details is known as **abstraction.**
- Database systems are usually made-up of complex data structures as per their requirements.
- To make the user interaction easy with database, the internal irrelevant details can be hidden from users. This process of hiding irrelevant details from user is called data abstraction.
- The complexity of database can be hiding from user by different level of abstraction as follows.

1.4.1.1 Levels of Abstraction SPPU - Dec. 13

University Question

Q. Explain in detail the different levels of abstraction. (Dec. 2013, 4 Marks)

There are three levels of abstraction :

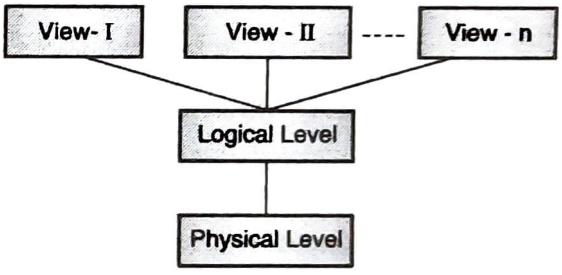


Fig. 1.4.1

1. Physical level

- In data abstraction Physical level is the lowest level. This level describes how the data is actually stored in the physical memory.
- The physical memory may be hard disks, magnetic tapes, etc. In Physical level the methods like hashing are used for organization purpose.
- Developer would know the requirement, size and accessing frequency of the records clearly in this level which makes easy to design this level.

2. Logical level

- This is the next higher level of abstraction which is used to describe what data the database stores,

and what relationships exist in between the data items. The logical level thus describes an entire database in terms of a small number of relatively simple structures.

- Although implementation of the simple structures at the logical level may involve complex physical level structures, the user of the logical level does not need to be aware of this complexity. This is considered as physical data independence.
- Database administrators use the logical level of abstraction to decide what information to keep in a database.

3. View level

SPPU - May 13

University Question

Q. Explain the need of view. (May 2013, 3 Marks)

- It is the highest level of data abstraction. This level describes the user interaction with database system. In the logical level, simple structures are used but still complexity remains because in the large database various type of information is stored.
- Many users are not aware of technical details of the system, and also they need not to access whole information from the database. Hence it is necessary to provide a simple and short interface for such users as per their requirements. Multiple views can be created for same database for multiple users.

  Example : Consider that we are storing information of all the employees of an organization in employee table. At **physical level** these records can be described as blocks of storage (bytes, gigabytes, terabytes etc.) in memory. These details are usually hidden from the developer.
- The records can be described as fields and attributes along with their data types at the **logical level**. The relationship between these fields can be implemented logically. Usually the developer works at this level because they have knowledge of such things about database systems.
- End user interacts with system with the help of GUI and enters the details at the screen at **view level**. User is not aware of how the data is stored and what data is stored; such details are hidden from them.

1.4.2 Schema

- The design of a database is called the schema. To understand schema we can consider an example of a program of an application. A variable or array declared with its structure (data type and/or size) is schema. The changes in schema or not frequent.
- **Types of Schema** : According to the level of abstraction, the database schema is divided into three types : Physical schema, Logical schema and View schema.

  (i) **Physical schema** is the design of a database at physical level, i.e. how the data stored in the blocks of storage is described in this level.

  (ii) **Logical schema** is the design of database at logical level. Developers and database administrators work at this level. Here the data can be described as certain types of data records gets stored in data structures, however the internal details like the implementation of data structure are hidden at this level.

  (iii) **View schema** refers to design of database at view level. This usually describes the end user interaction with database systems. There may be multiple schemas at view level.

1.4.3 Instance

In database changes are quite frequent i.e. insertion, deletion or updation are the frequent operations on database. The data is stored in the database at particular moment is called as instance of the database. In the example of application program, the value of a variable at particular time or situation is called as instance of database schema.

Syllabus Topic : Database Languages

1.5 Database Languages

In general in a database system, the Data Definition Language is used to specify schemas (design) of a database while the data manipulation language is used to fire queries (commands) on database in order to manipulate it. Both are the main pillars of database language like SQL.

With DDL and DML the database system also has the languages like DCL and TCL for different functionalities. The database languages are categorized as DDL, DML, DCL and TCL.

1.5.1 Data Definition Language (DDL)

- The DDL specify the schema of database by set of definitions.
- This language allows the users to define data and their relationship to other types of data. It is used to create data tables, dictionaries, and files within databases.
- The DDL is also used to specify the structure of each table, set of associated values with each attribute, integrity constraints, security and authorization information for all the tables and physical storage structure of all the tables on the disk.
- The data values stored in the database should satisfy some constraints for consistency purpose. For example the salary of an employee should never be negative or the employee id should start with 'E' etc. The Data Definition Languages provides functionality to specify such constraints.

1.5.2 Data Manipulation Language (DML)

- The Data Manipulation Language (DML) is used for accessing and manipulating data in a database. DML provides a set of functionalities to support the basic data manipulation operations on the data stored in the database.
- The DMS is basically of two types.
  1) **Procedural DML** – There is a requirement of user to specify which data is required and how get this data.
  2) **Declarative DML** – Here is also requirement of user to specify which data is required and without specifying how get this data. This is easier to understand and useful than procedural DML.
- A concept of query is used for processing. Query is a statement or command which requests the retrieval of information from the database. The part of DML which involved information retrieval is known as query language. The three levels of abstraction are used in defining structure of data as well as to manipulate the data.

Syllabus Topic : Database System Structure

1.6 Database System Structure

SPPU - Dec. 13

University Question

Q. Explain component and overall structure of DBMS. **(Dec. 2013, 10 Marks)**

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

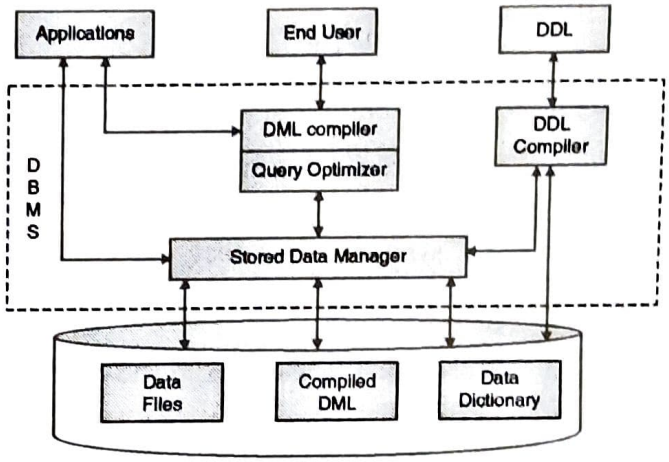


Fig. 1.6.1 : Database System Structure

Structure of DBMS

The Structure of DBMS contains following components :

1. DDL Compiler

- The DDL Compiler converts DDL commands into set of tables containing metadata stored in a data dictionary.
- The metadata information is name of the files, data items, storage details of each file, mapping information and constraints etc.

2. DML Compiler and Query optimizer

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

3. Data Manager

- The Data Manager is the central software component of the DBMS also known as Database Control System.
- The main functions of Data Manager are :
  - It converts the requests received from query optimizer to machine understandable form. It makes actual request inside the database.
  - Controls DBMS information access that is stored on disk.
  - It controls handling buffers in main memory.
  - It enforces constraints to maintain consistency and integrity of the data.
  - It synchronizes the simultaneous operations performed by the concurrent users.
  - It also controls the backup and recovery operations.

4. Data Dictionary

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

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

Importance of Data Dictionary

- Data Dictionary is necessary in the databases due to following reasons:
- It improves the control of DBA over the information system and user's understanding for the use of the system.
- It helps in documenting the database design process by storing documentation of the result of every design phase and design decisions.
- It helps in searching the views on the database definitions of those views.
- It provides great assistance in producing a report of which data elements (i.e. data values) are used in all the programs.

5. Data File

It contains the data portion of the database i.e. it has the real data stored in it. It can be stored as magnetic disks, magnetic tapes or optical disks.

6. Compiled DML

- The DML complier converts the high level Queries into low level file access commands known as compiled DML.
- Some of the processed DML statements (insert, update, delete) are stored in it so that if there is similar requests, the data can be reused.

7. End Users

They are the real users of the database. They can be developers, designers, administrator or the actual users of the database.

Syllabus Topic : Data Models

1.7 Data Models

Basic Concept of Data Models

The process of analysis of data object and their

relationships to other data objects is known as data modeling. It is the conceptual representation of data in database. It is the first step in database designing. Data models define how data is connected to each other and how they are processed and stored inside the system. A data model provides a way to describe the design of a database at the physical, logical and view levels.

1.7.1 Types of Data Models SPPU - May 13

University Question

Q. Explain various Data Models used in DBMS. (May 2013, 10 Marks)

There are different types of data models :

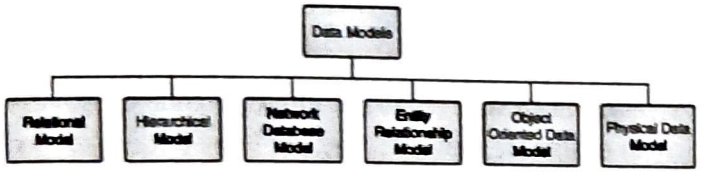


Fig. 1.7.1

1.7.1.1 Relational Model

- The relational model is developed by E.F. Codd. Relational database is a type of record-based relations.
- Relational database is an attempt to simplify the data structure by making use of tables. Tables are used to represent the data and their relationships. Table is a collection of rows and columns. Tables are also known as relations.
- Records are known as tuples and fields are known as attributes.
- The relational model is called as record based model because the database is structured in fixed format records of different types. A record is consists of fields or attributes.
- In the relational model, every record must have a unique identification or key based on the data.
- In following table Stud_ID is the key through which we can identify the record uniquely in the relation. Relational data model is the most widely used record-based data model.

Key

Tuple →

Stud_ID	Stud_Name	DOB
101	Prajakta	03/03/1995
102	Rakesh	13/01/1996
103	Rahul	16/08/1995

Advantages of Relational Data Model

a) Supports SQL

- For accessing the data in Relational data model we have a language known as Structured Query Language (SQL).
- This language is used to access, insert, update or delete the data from the table. By using relational data model we can execute the complex queries.

b) Flexible

- We can easily manipulate the information which is linked with various tables.
- We can extract the information from different table simultaneously by using this model.

1.7.1.2 Hierarchical Model

- A data model in which the data is organized into a tree structure is known as hierarchical data model.
- Hierarchical data model structure contains parent-child relationship where root of the tree is a parent which then branches into its children. It is another type of record based data model.
- The data is stored in the form of records. These records are connected to one another.
- A record is a collection of fields; each field contains only one value. The hierarchical data models represent data according to its hierarchy.

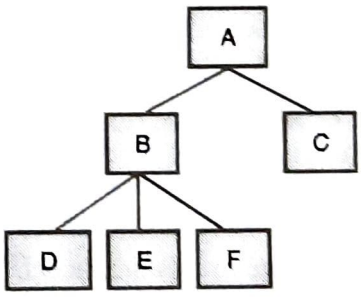


Fig. 1.7.2 : Hierarchical Model

- In hierarchical model there is a business rule. The rule is that, one parent node can have many child nodes but one child nodes can have only one parent node. This type of model is not in use currently.

Advantages of Hierarchical Model

a) Simple to understand : Due to its hierarchical structure it is easy to understand. Most of the time data have hierarchical relationship. Therefore, it is easy to arrange the data in that manner.

b) **Database Integrity :** In hierarchical data model there is always a parent child association between different records on different level. Due to this inherent structure integrity gets maintained.

c) **Efficient :** The performance of this model is very efficient due to its hierarchical structure when database contain large amount of data which has various related records.

1.7.1.3 Network Database Model

- It is extended type of hierarchical data model. This data model is also represented as hierarchical, but any child in the tree can have multiple parents.
- In network data model there is no need of parent child association. There is no downward tree structure.

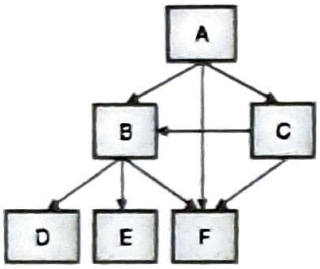


Fig. 1.7.3 : Network Model

- It is the flexible way of representing the objects and their relationship. A network data model allows multiple records linked in the same file.
- Basically, network database model forms a network like structure between the entities.

Advantages of Network Model

a) **Design is simple :** The network model is simple and easy to design and understand. There is no complex structure in network model.

b) **It has capability to handle various relationships :** The network model can handle the one to many and many to many relationships which is useful to develop the database.

c) **Easy to access :** The data access is easy and flexible than the hierarchical data model. In network model there is no any hierarchy in the objects and their relations therefore it is very easy to access the data in network model.

1.7.1.4 Entity Relationship (E-R) Model

- This model describes inter-related things of interest in a specific domain of knowledge. An ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between instances of those entity types. In the next Sections 1.10 and 1.11 we will study this model in detail.

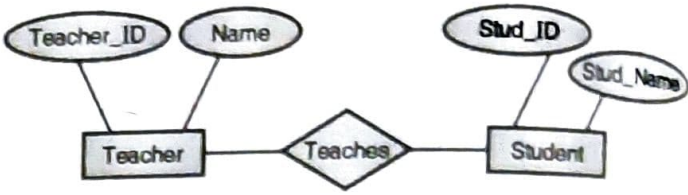


Fig. 1.7.4 : ER Model

Advantages of Entity Relationship Model

a) **Simple Design :** The ER model is simple and easy to design. It shows the logical view of the data and its relationships. This model is easy to understand.

b) **Effective representation :** The presentation of Entity Relationship Model is very simple and effective. The programmer and designer can easily understand the flow of the system by referring the ER Model.

c) **Connected with Relational Model :** The Entity Relationship Model is connected with the relational model. Due to this advantage we can develop a well-structured design.

1.7.1.5 Object Oriented Database Model

- Object oriented data model is nothing but the collection of objects or elements with its methods.
- Now a days all advanced programming languages supports the concepts of Object Oriented Programming.
- This model is based on the Object-oriented paradigm. This paradigm is defined for the database.
- It is extension to E-R model with integration of concepts like object, method and encapsulation.
- Applications of data modelling contain the key concepts of object-oriented.

Advantages of Object- Oriented Data Model

a) Object oriented data Model supports inheritance. Due to this we can reuse attributes and functionalities. It reduces the cost of maintaining data multiple times.

b) Due to inheritance, and encapsulation this model become more flexible.

c) We can bind each class with its attributes and functionality. It helps in representing the real world objects.

d) We can see each object as a real world entity in this model. Hence it is more understandable.

1.7.1.6 Physical Data Model

- The description of data storage in the computer in the form of information is provided by Physical data models.
- Physical data model shows the table structure which includes column name, column data, constraints and relationship between tables. Physical data models define all the components of logical database.

Advantages of Physical Model

a) We can specify the all tables and columns using physical data model.

b) We can use foreign key to join the tables. By using foreign key we can identify the related data from the different tables.

c) By using physical data model we can convert entity into the table.

Syllabus Topic : Database Design and ER Model

1.8 Database Design and ER Model

1.8.1 Database Design

In the creation of database system, the design of database is the most important and initial part. The overall success of the system is completely depends upon the design of database. The design is mainly focused on the security and access of data by the application program. The requirements of user play an important role in database design.

The process of doing database design generally consists of a number of steps which will be carried out by the database designer. Usually, the designer must:

- Determine the data to be stored in the database.
- Determine the relationships between the different data elements.
- Superimpose a logical structure upon the data on the basis of these relationships.

Determine the data to be stored in the database

- Number of times, the database designer is a person with expertise in the area of database design. It is not necessary that he should be expertise in the domain from which is the data is retrieved e.g. financial information, biological information etc.
- Hence it is important that the data to be stored in the database should be determined in cooperation with the domain expertise that has knowledge that which data must be stored within the system.
- This process is a part of requirements analysis and required that the database designer should get the required data from domain expertise.
- The reason is that number of times the domain expertise cannot express exactly what their system requirements for the database are unfamiliar to thinking in terms of the discrete data elements which are needed to be stored.
- Data to be stored can be determined by Requirement Specification. In the life cycle of software it is known as requirement gathering phase.

Determining data relationships

- Once a database designer is aware of the data which is to be stored within the database, they must then determine where dependency is within the data. Sometimes when data is changed you can be changing other data that is not visible.
- For example, in a list of names and addresses, assuming a situation where multiple people can have the same address, but one person cannot have more than one address; the address is dependent upon the name.
- When provided a name and the list the address can be uniquely determined; however, the inverse does not hold - when given an address and the list, a name cannot be uniquely determined because multiple people can reside at an address.
- Because an address is determined by a name, an address is considered dependent on a name.

Logically structuring data

- Once the relationships and dependencies amongst the various pieces of information have been determined, it is possible to arrange the data into a logical structure which can then be mapped into

the storage objects supported by the database management system.

- Now the important part in design is that how to represent things like person, product, place etc. These things can be represented in term of entities. The various entities are related with each other by one or other way.

In database design schema, it is necessary to avoid two major problems.

1. Redundancy

- Sometimes as per requirement same data may be stored in multiple files. Consider an employee having record in both Employee and Team files.
- The name and address of employee is stored in both of these tables. Means the data get duplicated. If such data increases, it leads to higher storage and access cost.
- This duplication of data in various files is termed as data redundancy. Such redundancy can also occur in relational schema also. In the schema of the database, such information may be repeatedly shown.

2. Incompleteness

- An incomplete design is very difficult to model. Suppose in a IT enterprise database system, if a department like R& D is there, to which no employee is still appointed, then it becomes very difficult to represent information about this department.

1.8.2 E-R Model

- The Entity Relationship (E-R) model allows specifications of an enterprise schema and represents the overall logical structure of the database.
- Now a days the database related to real world applications becomes very vast and complex. Representing relations between the different elements of the database becomes difficult.
- ER Model simplifies this task. It is nothing but the design technique for database. It is a graphical technique which helps to understand and organize the complex data which should not depend upon the actual database implementation.
- The real world objects can be easily mapped with entities of E-R model.
- In Entity Relationship Model a graphical representation of a database system is generated. Diagrams are used in this model. These diagrams are known as entity-relationship diagrams, ER diagrams or ERDs.
- Basic concepts of ER Model are as follows :

Entity, attribute, Relationship, constraints and keys.

Syllabus Topic : Entity

1.8.2.1 Entity and Entity Set

- The E-R model consists of entities and relationships between those entities. An entity is nothing but a thing having its own properties. These properties helps to differentiate the object (entity) from other objects.
- An entity is a thing that exists either physically or logically. An entity may be a physical object such as a house or a car, or an entity may be a logical concept like an event such as a house sale or a car service, or a concept such as a customer transaction or order.
- An entity set is a set of entities which share the same properties. In a Company employee is the entity set which has similar properties like Employee_ID, emp_name, salary etc.
- There is difference between an entity and an entity-type. An entity-type is a category. An entity, strictly speaking, is an instance of a given entity-type. There are usually many instances of an entity-type
- There are two types of entities in Database management system.

1. Strong Entity or Regular Entity

- If an entity having it's own key attribute specified then it is a strong entity. Key attribute is used to indentify that entity uniquely among set of entities in entity-set.
- **Example :** In a parent/child relationship, a parent is considered as a strong entity.
- Strong entity is denoted by a single rectangle.

- The relation between two strong entities is denoted by a single diamond simply called relationship.

2. Weak Entity

- The entity which does not have any key attribute is known as weak entity. The weak entity has a partial discriminator key. Weak entity depends on the strong entity for its existence. Weak entity is denoted with the double rectangle.
- **Example :** In a parent/child relationship, a child is considered as a weak entity which is completely depends upon the strong entity 'parent'.

Comparison of an Object is OOP and Entity In E-R Model SPPU - May 13

University Question

Q. How does the concept of an object in the object oriented model differ from the concept of an entity in the E-R model ?
(May 2013, 8 Marks)

Sr. No.	Entity	Object
1.	An entity is a thing that exists either physically or logically. An entity may be a physical object such as a house or a car (they exist physically), an event such as a house sale or a car service, or a concept such as a customer transaction or order (they exist logically as a concept)	Object in Object oriented programming is nothing but anything which has its own properties. E.g. flower is an object having properties like color, fragrance etc.
2.	Even if data in two entity instances is the same, we don't deem them as equivalent.	Objects don't have an identifier field and if two objects have the same set of attributes we can treat them interchangeably.
3.	Entities live in continuum that means they have a history of what happened to them and how they changed during their lifetime.	Objects, at the same time, have a zero lifespan. We create and destroy them with ease.
4.	Entities are mutable. That means we can change them.	Objects should be immutable that means we cannot change them rather we construct a new object based on the existing object.

Syllabus Topic : Attributes

1.8.2.2 Attribute

- An attribute is a characteristic of an entity. Entities are represented by means of their attributes. All attributes have their own specific values. For example, an employee entity may have Employee_ID, emp_name, salary as attributes.
- In a database management system an attribute is a database component, such as field or column of a table.

Example :

The entity student has attributes like student_id, student_name. In this every attribute has a value. Here 101 is the value for the attribute student_id, Kunal is the value for attribute student_name.

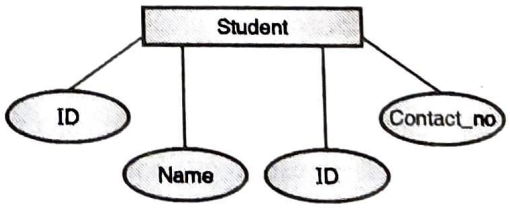


Fig. 1.8.1

There are five different types of attributes in Database Management System :

1. Single-valued Attribute

- A single-valued attribute is the attribute which can hold a single value for the single entity.
- **Example** : In the entity student, student_name is the single-valued attribute since a student have a single value for name attribute.

2. Multi-valued Attribute

- A multi-valued attribute is the attribute which can hold multiple values for the single entity.
- **Example :** In the entity student, the attribute student_contact no could be considered a multi-value attribute since a student could have multiple contact numbers.

3. Simple Attribute

- An attribute whose value cannot be further divided is known as simple attribute. That means it is atomic in nature.
- **Example :** In the entity student, the attribute student_age cannot be divided. Therefore student_age is the simple attribute of student entity.

4. Composite Attribute

- The composite attributes are the attributes which can be further divided into sub parts. These sub parts represent the basic entities with their independent meaning.
- **Example** : In the entity student, student_name is the composite attribute, we can divide this attribute in three different sub parts: First_name, Middle_name and Last_name.

5. Derived Attribute

- The attribute which is not physically exist in database, but its value can be calculated from the other present attributes is known as derived attribute.
- **Example :** In the entity student, we can calculate the average age of students. This average age is not physically present in the database but it can be derived from the attribute student_age;

Syllabus Topic : Relationships

1.8.2.3 Relationships

The association between two different entities is called as relationship. In the real world application, what does one entity do with the other, how do they connect to each other ?

For example : An employee works at a department, a student enrolls for a course. Here, works at and Enrolls are called relationships.

The Degree of Relationships

The degree of relationship refers to number of entities participated in the relation.

1. Unary Relationship

- A unary relationship exists when there is relation between single entity. A unary relationship is also known as recursive relationship in which an entity relates with itself.
- **Example** : A person can be in the relationship with another person, such as :
- A woman who can be someone's mother
- A person that is a someone's child.

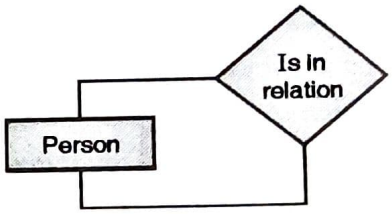


Fig. 1.8.2

2. Binary Relationship

- A binary relationship exist only when there is relation between only two entities. In this case the degree of relation is two.
- **Example :** A teacher teaches student. In this teacher and student are two different entities which are connected with each other via relation Teaches.

Fig. 1.8.3

3. Ternary Relationship

- A ternary relationship exists when there are relations between three entities. In ternary relation the degree of relation is six.
- **Example :** A person can be a student and a person also can be teacher. Here teacher, student and person are three entities which are related to each other.

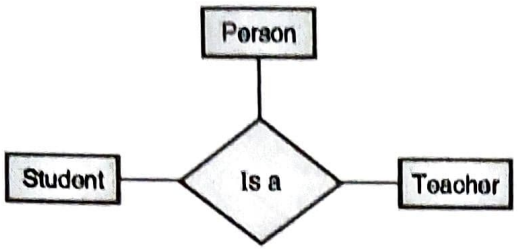


Fig. 1.8.4

4. Quaternary Relationship

- A quaternary relationship exists when there are relations between four entities. In quaternary relation the degree of relation is eight.
- **Example :** The four entities Employee, Management Faculty, Teaching Faculty, and Non-Teaching Faculty are connected with each other via is a relationship.

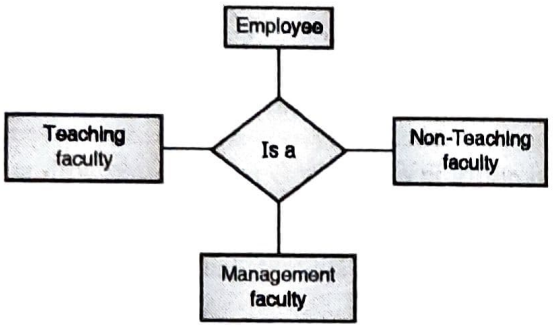


Fig. 1.8.5

Syllabus Topic : Constraints

1.8.2.4 Constraints

There are certain constraints in E-R enterprise schema to which the contents of a database must conform.

Two types of constraints are

1. Mapping cardinalities
2. Participation constraints

1. Mapping Cardinalities SPPU - Dec. 13, May 14

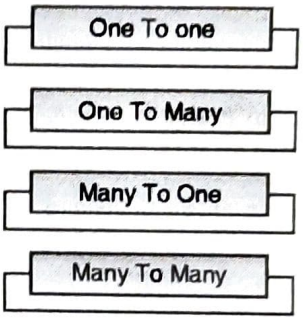
University Questions

Q. What is meant by mapping cardinality? **(Dec. 2013, 2 Marks)**

Q. What is meant by Mapping Cardinality? Explain different types of Cardinalities for a binary relationship with example. **(May 2014, 4 Marks)**

Cardinality in DBMS

- In Database Management System the term 'Cardinality' refers to the uniqueness of data values contained in a column.
- If the column contains a large percentage of totally unique values then it is considered as high cardinality while if the column contains a lot of "repeats" in its data range, it is known as Low cardinality.



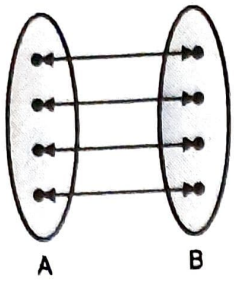
- Sometimes cardinality also refers to the relationships between tables. Cardinality between tables can be one-to-one, many-to-one or many-to-many.
- A relationship where two entities are participating is called a **binary relationship**.

Mapping Cardinalities

- Mapping cardinality expresses the number of entities to which another entity can be associated with in a relationship-set.
- It defines the relationship between two entities via a relationship set. Consider in a binary relationship set R (relation) between entity of set A and set B. For this relationship mapping cardinalities are as Follows :

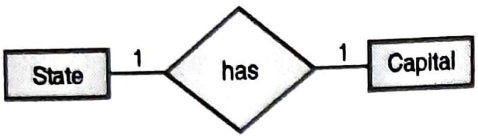
One to One

- An entity of entity-set A can be associated with at most one entity of entity-set B and vice versa that means an entity in entity-set B can be associated with at most one entity of entity-set A.



(a)

Example : In following example one state have only one capital.

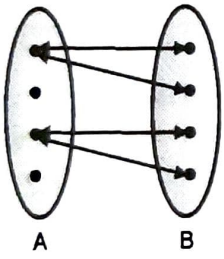


(b)

Fig. 1.8.6

One to Many

- In this type an entity in set A is associated with many other entities in set B.
- But an entity in entity set B can be associated with maximum of one entity in entity set A.



(a)

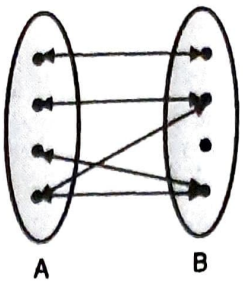
Example : In following example one teacher can teach many students.

(b)

Fig. 1.8.7

Many to One

- In this type an entity in set A is associated with at most one entity in set B. And an entity in set B can be associated with number of entities in set A.



(a)

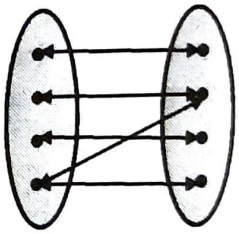
Example : In following example many students can enroll in one school.

(b)

Fig. 1.8.8

Many to Many

- In this type any entity in entity set A is associated with number of entities in entity set B.



(a)

- An entity in entity set B is associated with number of entities in set entity A.
- **Example :** Many students learn many subjects.

(b)

Fig. 1.8.9

The appropriate mapping cardinality for a particular relationship set is depending upon the real world situation to which the relationship set is modeling.

2. Participation Constraints

There are two types of participation constraints.

i. Total participation

ii. Partial participation

i. **Total Participation :** The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R.

ii. **Partial Participation :** The participation of an entity set E in a relationship set R is said to be partial if only some entities in E participates in relationships in R.

- For example, in a college database system, consider teachers are assigned as project guides to every student. In this case every Student entity is related with teacher entity through the relationship "Guide".
- Hence participation of student in the relationship guidance is total. But it is not compulsory that every teacher should guide the students.
- Hence it is possible that not all the teacher entities are related with the student through the relationship "Guide". Here the participation of teacher in the "guide" relationship set is partial.

Syllabus Topic : Keys

1.8.2.5 Keys

SPPU - Dec. 16

University Question

Q. Explain the distinction among the terms primary key, candidate key, and super key. (Dec. 2016, 5 Marks)

The specification of differentiation of entities in a given entity set is very important. The entities can be differentiated in terms of attributes. Here the values of attributes come in picture. These values should be different to identify the attributes uniquely. It is necessary that in an entity set, no two entities should have same values for all the attributes.

In the database schema the notion of key is directly applies to entity sets. The key for an entity in entity set is an attribute or set of attributes which is used to distinguish entities from each other.

Keys are also used to identify relationships uniquely and differentiate these relationships from each other.

There are six types of keys available in DBMS :

1. Primary Key

- Primary key uniquely identify each entity in the entity set. It must have unique values and cannot hold null values. Let, R be a relationship set having entity sets E1,E2,....En. Consider primary key (Ei) denotes the set of attributes that forms the primary key for entity ser Ei. The set of attributes associated with the relationship set R is responsible for composition of primary key for that relationship set.
- **Example :** In Bank database, the account_number entity should be primary key. Because this field cannot be kept NULL as well as no account_number should be repeated.

2. Super Key

- This key is formed by combining more than one attributes for the purpose of uniquely identifying entities.
- **Example :** In student database having attributes Student_reg_id, Student_roll_no, Sutdent_name, Address, Contact_no.
- The Super keys are :
  {Student_reg_id}
  {Student_roll_no}
  { Student_reg_id, Student_roll_no }
  { Student_reg_id, Sutdent_name }
  { Student_roll_no, Sutdent_name }
  {Student_reg_id, Student_roll_no, Sutdent_name }
- It means super key can be any combination of attributes, so that identifying the record becomes easier.

3. Candidate Key

- Candidate key is formed by collection of attributes which hold unique values. A super key without redundant values is known as candidate key. Candidate keys are selected from the set of super keys.
- Candidate key are also known as minimal super key having uniqueness property. The attribute which do not contain duplicate value, may be a candidate key.

2. Participation Constraints

There are two types of participation constraints.

I. Total participation

II. Partial participation

I. **Total Participation :** The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R.

II. **Partial Participation :** The participation of an entity set E in a relationship set R is said to be partial if only some entities in E participates in relationships in R.

- For example, in a college database system, consider teachers are assigned as project guides to every student. In this case every Student entity is related with teacher entity through the relationship "Guide".
- Hence participation of student in the relationship guidance is total. But it is not compulsory that every teacher should guide the students.
- Hence it is possible that not all the teacher entities are related with the student through the relationship "Guide". Here the participation of teacher in the "guide" relationship set is partial.

Syllabus Topic : Keys

1.8.2.5 Keys

SPPU - Dec. 16

University Question

Q. Explain the distinction among the terms primary key, candidate key, and super key.
(Dec. 2016, 5 Marks)

The specification of differentiation of entities in a given entity set is very important. The entities can be differentiated in terms of attributes. Here the values of attributes come in picture. These values should be different to identify the attributes uniquely. It is necessary that in an entity set, no two entities should have same values for all the attributes.

In the database schema the notion of key is directly applies to entity sets. The key for an entity in entity set is an attribute or set of attributes which is used to distinguish entities from each other.

Keys are also used to identify relationships uniquely and differentiate these relationships from each other.

There are six types of keys available in DBMS :

1. Primary Key

- Primary key uniquely identify each entity in the entity set. It must have unique values and cannot hold null values. Let, R be a relationship set having entity sets E1,E2,....En. Consider primary key (Ei) denotes the set of attributes that forms the primary key for entity ser Ei. The set of attributes associated with the relationship set R is responsible for composition of primary key for that relationship set.
- **Example :** In Bank database, the account_number entity should be primary key. Because this field cannot be kept NULL as well as no account_number should be repeated.

2. Super Key

- This key is formed by combining more than one attributes for the purpose of uniquely identifying entities.
- **Example :** In student database having attributes Student_reg_id, Student_roll_no, Sutdent_name, Address, Contact_no.
- The Super keys are :
  {Student_reg_id}
  {Student_roll_no}
  { Student_reg_id, Student_roll_no }
  { Student_reg_id, Sutdent_name }
  { Student_roll_no, Sutdent_name }
  {Student_reg_id, Student_roll_no, Sutdent_name }
- It means super key can be any combination of attributes, so that identifying the record becomes easier.

3. Candidate Key

- Candidate key is formed by collection of attributes which hold unique values. A super key without redundant values is known as candidate key. Candidate keys are selected from the set of super keys.
- Candidate key are also known as minimal super key having uniqueness property. The attribute which do not contain duplicate value, may be a candidate key.

Example : In student database with attributes Student_reg_id, Student_roll_no, Sutdent_name, Address, Contact_no.

- The Candidate keys are :
  {Student_reg_id}
  {Student_roll_no}
  { Student_reg_id, Student_roll_no }
- It means candidate key can be any combination of key attributes, so that identifying the record from the table becomes easier.

4. Alternate Keys

- From all candidate keys, only one key gets selected as primary key, remaining keys are known as alternative or secondary keys.
- **Example :** In student table Student_address, Contact_no, Date_Of_Birth are the alternative keys.

5. Composite Key

- A key which consists of more than one attributes to uniquely identify rows in a table is called composite key. It is also known as compound key.
- **Example :** In student databse having attributes Student_reg_id, Student_roll_no, Sutdent_name, Address, Contact_no.
- The composite keys are :
  { Student_reg_id, Student_roll_no }
  { Student_reg_id, Sutdent_name }
  { Student_roll_no, Sutdent_name }
  { Student_reg_id, Student_roll_no, Sutdent_name }

These sets of keys are used to uniquely identify the record from the table.

Syllabus Topic : Design Process

1.9 Database Design Process

The database design process includes following six stages.

1. Requirement analysis
2. Conceptual database design
3. Choice of the DBMS
4. Logical Database Design
5. Physical design
6. Implementation

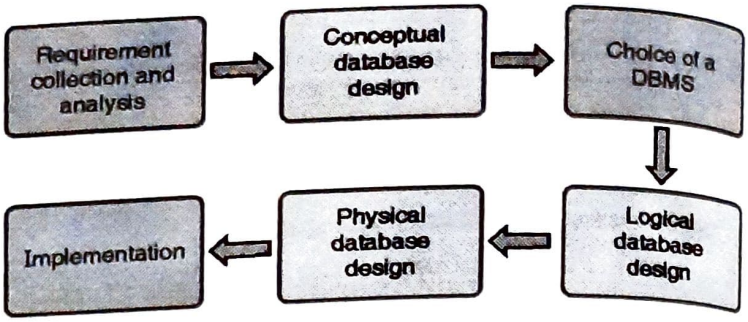


Fig. 1.9.1 : Steps of database design

1. Requirement analysis

- This stage involves the data gathering of requirements from user. Here we discover DATA and OPERATIONS requirements by interaction with the customer.
- The discussion about what current data is and which operations user expect to perform on the database.
- Here we also discuss how data elements are related with each other.

2. Conceptual database design

- The main Purpose of this stage is to produce a conceptual schema of the database using concepts of the high level data model.
- This stage does not include implementation details. It is expected that this design should understood by non-technical users.
- It has detail information about the all the objects of the domain.
- The conceptual database design is independent of the DBMS to be used.
- This design cannot be used directly to implement the database.

3. Choice of the DBMS

- The main purpose of this stage is to select the best suitable database framework for implementing the produced schema.
- The schema includes :
  - Type of DBMS (relational, network, deductive, Object Oriented, ...)
  - User and programmer interfaces
  - Type of query language

- The choice of database is also depends upon technical factors like whether the DBMS has support the required tasks or not.
- The economic factors are also get considered which includes
  - Software and hardware purchase and maintenance.
- Training of staff
- The organisational factors considered are :
  - Platforms supported, availability of vendor services

4. Logical Design

- The main Purpose of logical design is to transform the generic, DBMS independent conceptual schema in the data model of the chosen DBMS. This technique is also called as data model mapping.
- There are two stages of mapping
  1. System independent mapping : In this stage, there is no consideration of characteristics of database i.e. it is totally independent of database system.
  2. Tailoring to DBMS : The same model can be implemented by different DBMS.
- Logical design produces set of DDL statements in the language of the chosen DBMS

5. Physical Design

- The Purpose Physical Design is to select the specific storage structures and access paths for the database files.
- The performances factor is mainly considered.
- Performance of system is based on following factors.
  - **Response time :** The database access time should be less for data items which are frequently used by transactions.
  - **Space utilisation :** Less frequently used data and queries may be archived to save the space.
  - **Transaction throughput :** The number of transactions that can be processed per minute should be more.

6. Implementation

- This is the most important and last stage in design process. In it database is created.
- The DDL statements are compiled and executed.
- In this stage application programs are written with embedded DML statements
- From here operational phase will be initiated.

Syllabus Topic : Entity Relationship Model

1.10 Entity Relationship Model

- In 1976 Entity relationship model was developed. It is useful in conceptual design. It is the high level data model.
- This model defines the data objects and their relationship. It is the popular model in database.
- This model consists of entities and relationships between those entities. An entity is nothing but a thing having its own properties. These properties helps to differentiate the object (entity) from other objects.

Syllabus Topic : ER Diagram

1.11 ER Diagram

The pictorial representation of data using different conventions which state that how these data are related with each other is known as Entity Relationship Diagram. E-R diagrams express the logical structure of database in graphical manner. Special symbols are used to draw an ER-Diagram. Every symbol has its own meaning.

Example of various symbols used in ER Diagram

Following are the symbol / notations used in ER-Diagram :

Symbol	Symbol Name	Symbol Description
Entities		
	Rectangle	**Entity**

Symbol	Symbol Name	Symbol Description
	Double rectangle	**Weak entity**
Attributes		
	Ellipse	**Attribute**
	Ellipse with Under Line	**Key Attribute**
	Double Ellipse	**Multi-valued Attribute**
	Dashed Ellipse	**Derived attribute**
Relationships		
	Diamond	**The relationship**
	Line	**Links attributes to entity sets and entity sets to relationship sets.**
	Double Line	**Represents total participation of an entity in relationship set**

Representations

1. Entity

An **Entity** is any object, place, person or class. In E-R Diagram, an **entity** is represented using rectangles. Consider an example of an Organization. Employee, Manager, Department, Product etc. are considered as entities.

Fig. 1.11.1

Here employee and department are entities.

2. Weak Entity

Weak entity is an entity which depends upon another entity. Weak entity is represented by double rectangle. Subject is the weak entity. Because subject is depends on course.

Fig.1.11.2

3. Attribute

Attributes are nothing but the properties of entity. Here Stud_id, Name and address are attributes of entity Student.

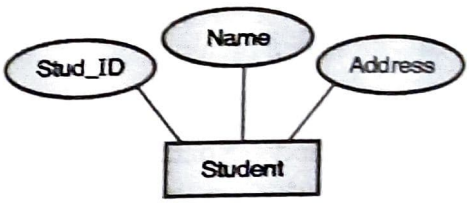


Fig. 1.11.3

4. Key Attribute (Primary key)

To identify attribute uniquely we set the key to the attribute. It is denoted by underline.

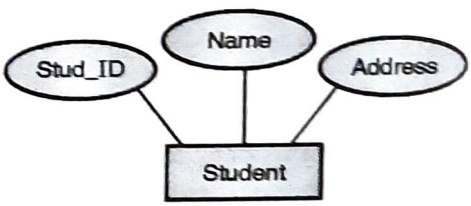


Fig. 1.11.4

5. Multi valued Attribute

The attribute which have multiple values is known as multi valued attribute.

Here Phone No is multi valued attribute as a person can have more than one phone numbers.

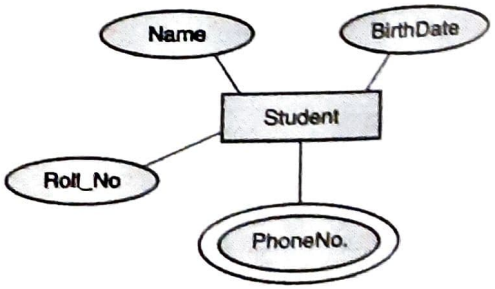


Fig. 1.11.5

6. Derived Attribute

Derived attributes are the attributes that do not exist physically in the database, but their values can be derived from other attributes present in the database.

For example : Age can be derived from data_of_birth.

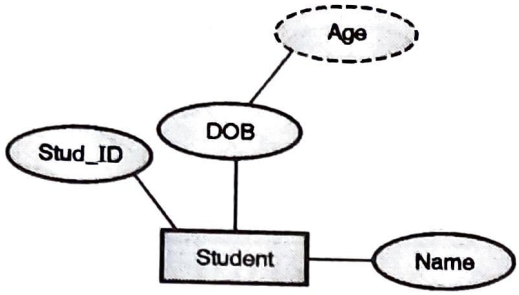


Fig. 1.11.6

7. Relationship

A relationship describes how entities interact with each other. For example, the entity "carpenter" may be related to the entity "table" by the relationship "builds". Relationships are represented by diamond shapes and are labeled using verbs.

Fig. 1.11.7

8. Recursive Relationship

If the same entity participates more than once in a relationship it is known as a recursive relationship. Consider an example where an employee can be a supervisor and be supervised by manager, so there is a recursive relationship.

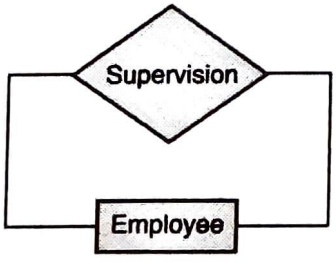


Fig. 1.11.8

Following E-R diagram represent the relationship between two entity sets teacher and student related through binary relationship guide.

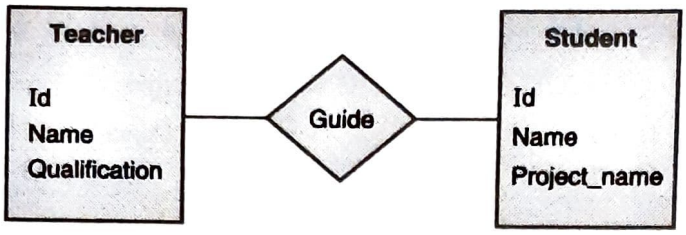


Fig. 1.11.9

The attributes of entity set teacher are
- Id
- Name
- Qualification

The attributes of entity set student are
- Id
- Name
- Project_name

1.11.1 Mapping Cardinality in E-R Diagram

In the two entities Teacher and Student, the relationship *Guide* may be

1. One to One
2. One to Many
3. Many to One
4. Many to Many

1. One to One

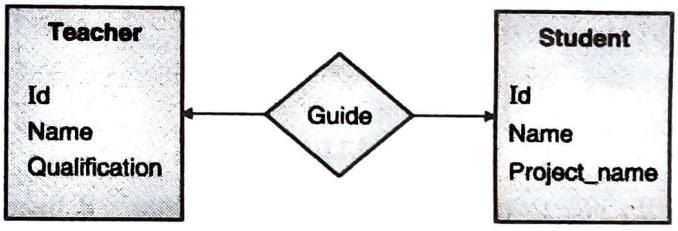


Fig. 1.11.10

In one to one mapping cardinality directed lines from relationship *Guide* are drawn towards both entity sets Teacher and Student. In this, the teacher can guide at most one student and the student can take guidance from at most one teacher.

2. One to Many

In one to many mapping cardinality directed line from relationship *Guide* to entity set Teacher is drawn and undirected line from relationship *Guide* to entity set Student is drawn. In this, the teacher can guide many students but a student can take guidance from at most one teacher.

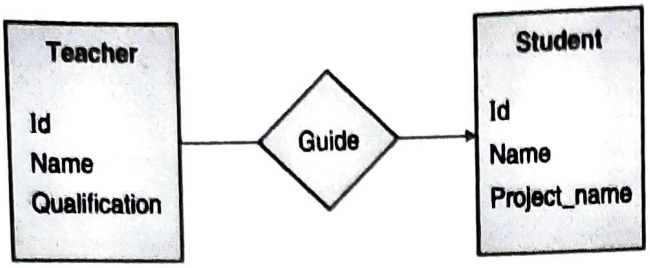


Fig. 1.11.11

3. Many to One

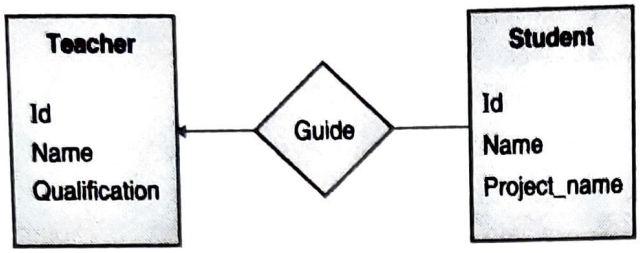


Fig. 1.11.12

In many to one mapping cardinality undirected line from relationship *Guide* to entity set Teacher is drawn and directed line from relationship *Guide* to entity set Student is drawn. In this, the teacher can guide at most one student but a student can take guidance from many teachers.

4. Many to Many

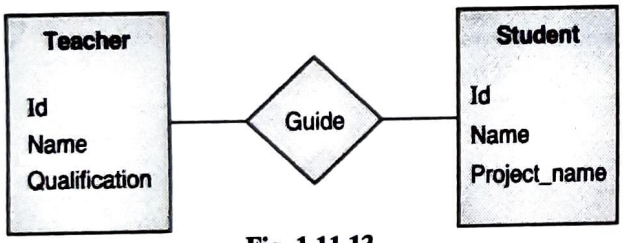


Fig. 1.11.13

In one to one mapping cardinality directed lines from relationship *Guide* are drawn towards both entity sets Teacher and Student. In this, the teacher can guide many students and the student can also take guidance from many teachers.

Points to remember while drawing an ER diagram

1. Initially identify all entities and their relationships with each other in the given database system.
2. No entity should be repeated in a particular diagram.
3. Provide a precise and appropriate name for each entity, attribute, and relationship in the diagram. Try to give user friendly words while naming. The name should also be meaningful, unique and easily understandable
4. Do not set unclear, redundant or unnecessary relationships between entities.
5. Never connect a relationship to another relationship.
6. Using colors helps to make the diagram easily understandable. It helps in differentiation and classification

1.11.2 Examples of ER Diagram

E-R diagram with multi valued and derived attributes

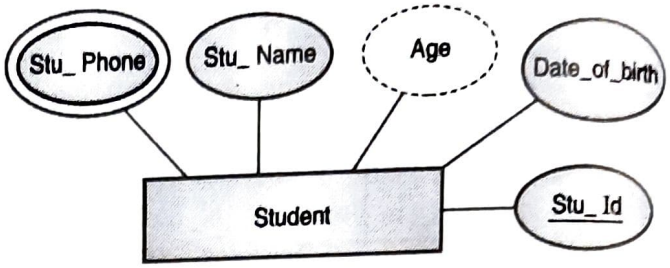


Fig. 1.11.14

Total Participation of an Entity set

If in a relationship set, every entity in entity set has one relationship then it called as total participation of entity set.

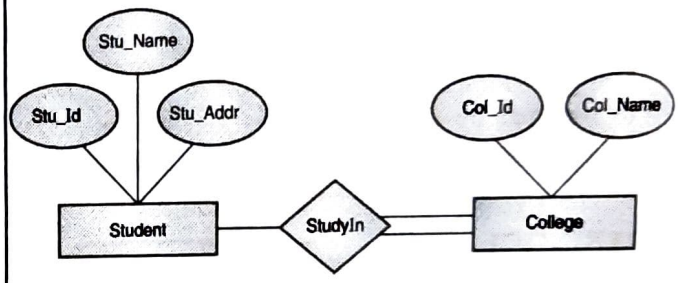


Fig. 1.11.15

In the Fig. 1.11.15, each college must have at least one associated student. A Total participation of an entity set represents that all the entities in the entity set should have minimum one relationship in a relationship set. For example: In the above diagram each college must have at-least one associated Student.

ER diagram for Database of employee

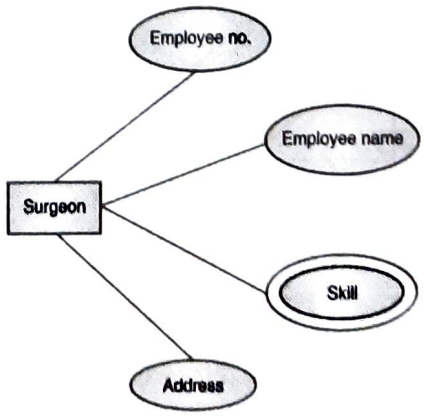


Fig. 1.11.16

Ex. 1.11.1

Following requirements are given for a database of the National Hockey League The NHL has many teams,

Each team has a name, a city, a coach, a captain, and a set of players, Each player belongs to only one team,

Each player has a name, a position (such as left wing or goalie), a skill level, and a set of injury records, A team captain is also a player, A game is played between two teams (referred to as host_team and guest_team) and has a date (such as May 11th, 1999) and a score (such as 4 to 2). Construct an ER diagram for the NHL database.

Soln. :

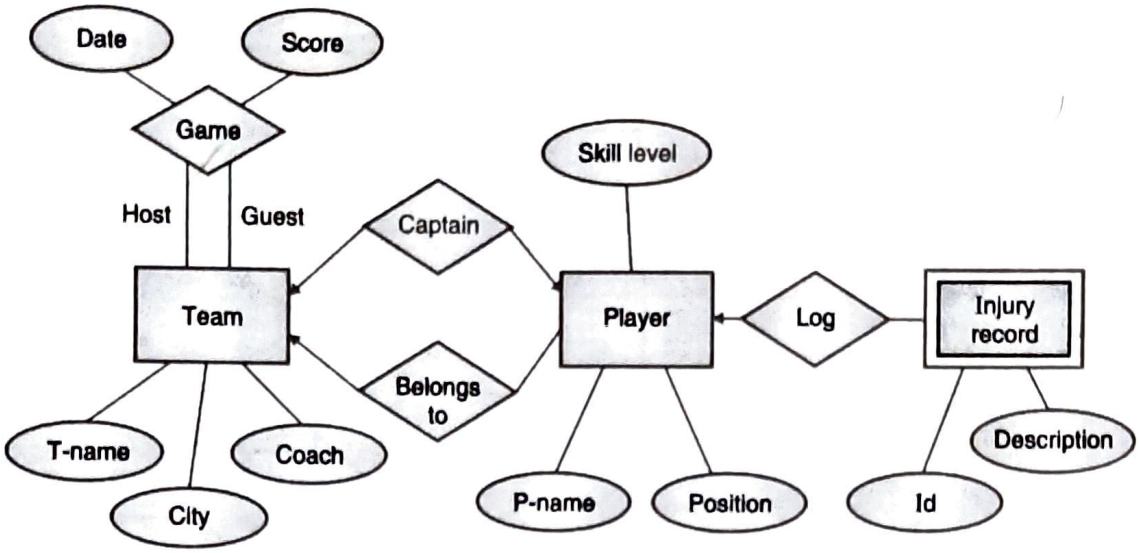


Fig. P. 1.11.1

Ex. 1.11.2 SPPU - Oct. 2016(In sem), 5 Marks

Draw an ER-Diagram for online Book Shop which should consist of entity set, attribute, relationship, mapping cardinality and keys, it will maintain information about all Customers, books, book author, publisher, billing etc.

Soln. :

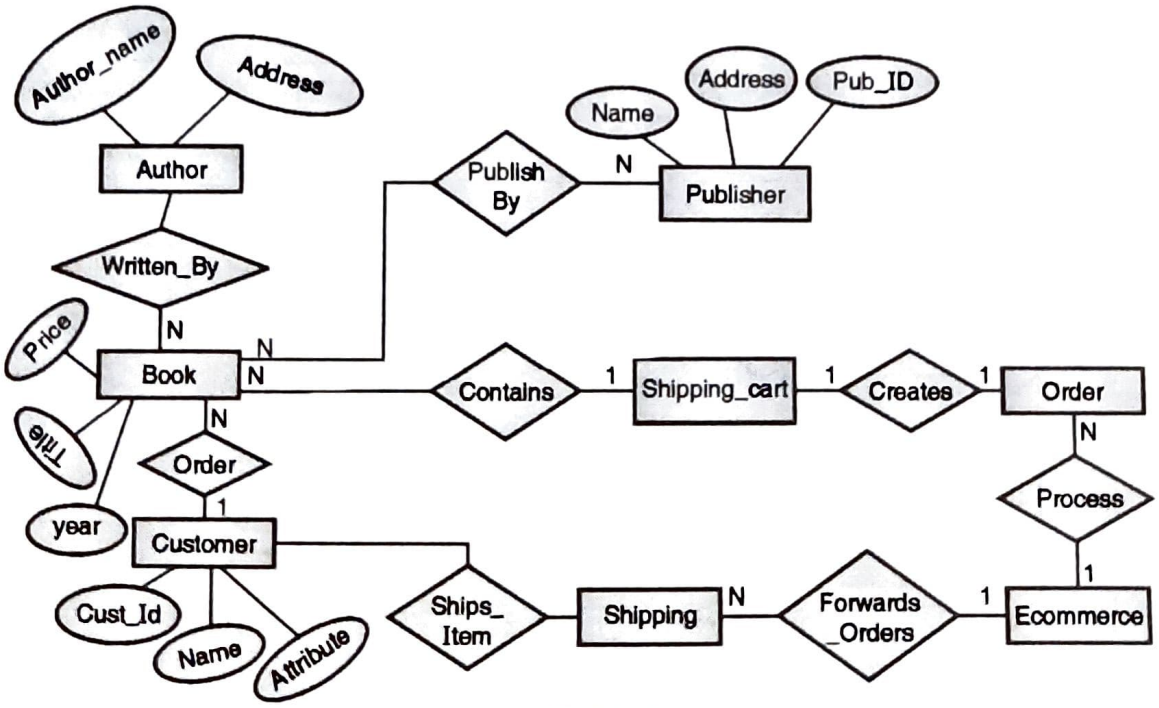


Fig. P. 1.11.2

Ex. 1.11.3

Draw an ER-Diagram for hospital management where patient take treatments from physician and also he/she can claim a medical insurance.

Soln. :

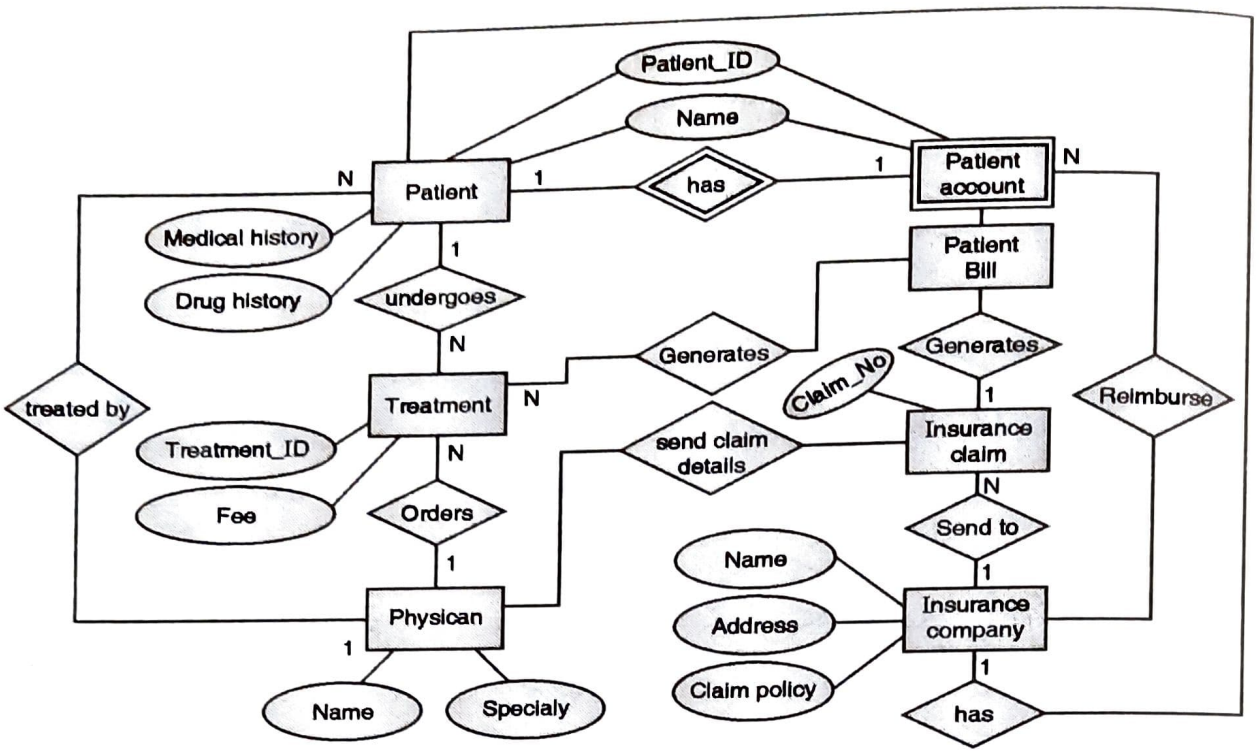


Fig. P. 1.11.3 : ER diagram for Hospital management system

Ex. 1.11.4 :

Draw an ER-Diagram for Online Sales system in which customer can order terms online and pay through credit card.

Soln. :

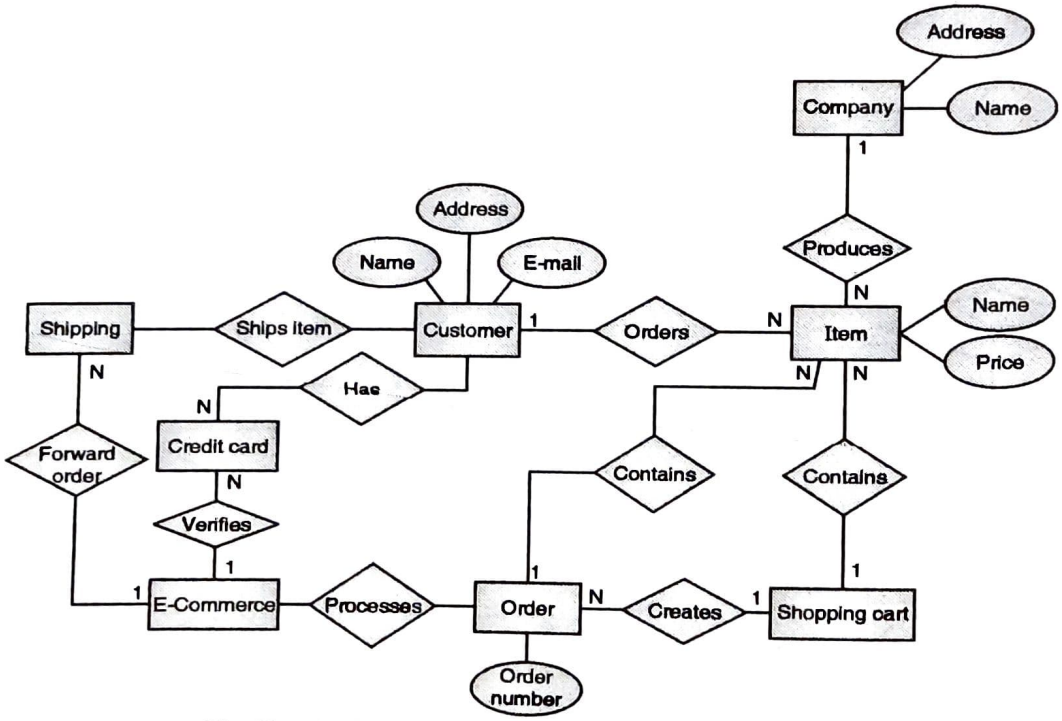


Fig. P. 1.11.5 : ER diagram for Online sales system

Ex. 1.11.5 SPPU - May 2016, 5 Marks

Construct an E-R diagram for a car insurance company that has a set of customers each of whom owns one or more cars. Each car has associated with it zero to any number of recorded accidents.

Soln. :

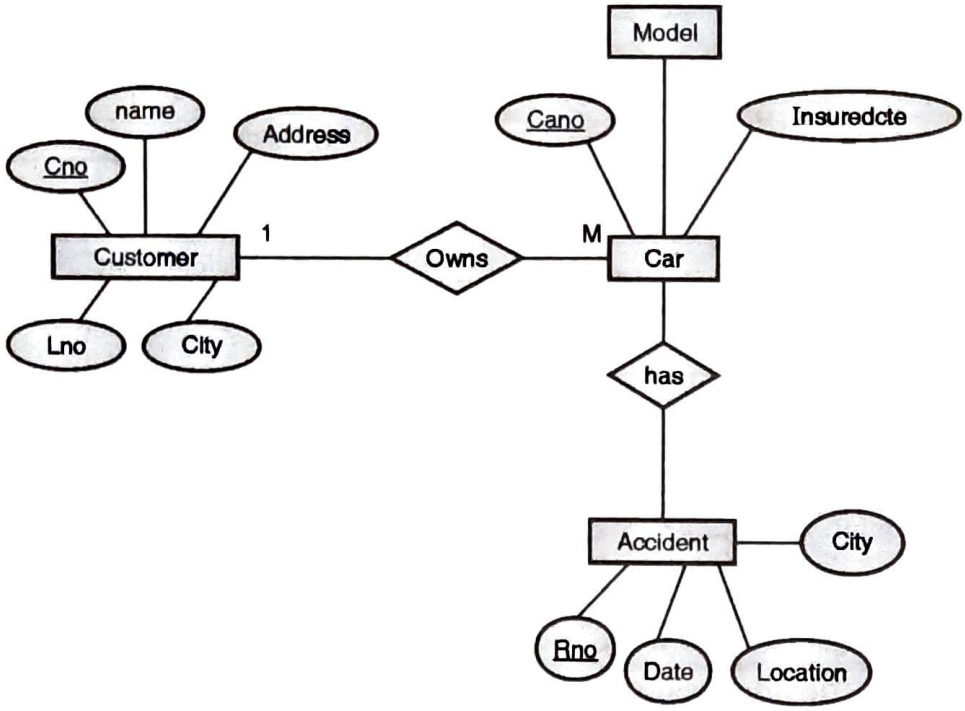


Fig. P.1.11.5

Ex. 1.11.6 :

Following information is maintained for online bookstore

(i) Books(ISBN,price,title,year)

(ii) Author(name,address,url)

(iii) Publisher((name,address,phone,url)

(iv) Customer(name,address,email,phone) (name is discriminating attribute)

(v) Shoppingbasket(basketID)

Construct ER diagram with following constraints :

Each book should have author and publisher. Book may have more than one author. Each customer have a dedicated shopping basket. Books can further be categorized as books, music, cassette or compact disks.

Soln. :

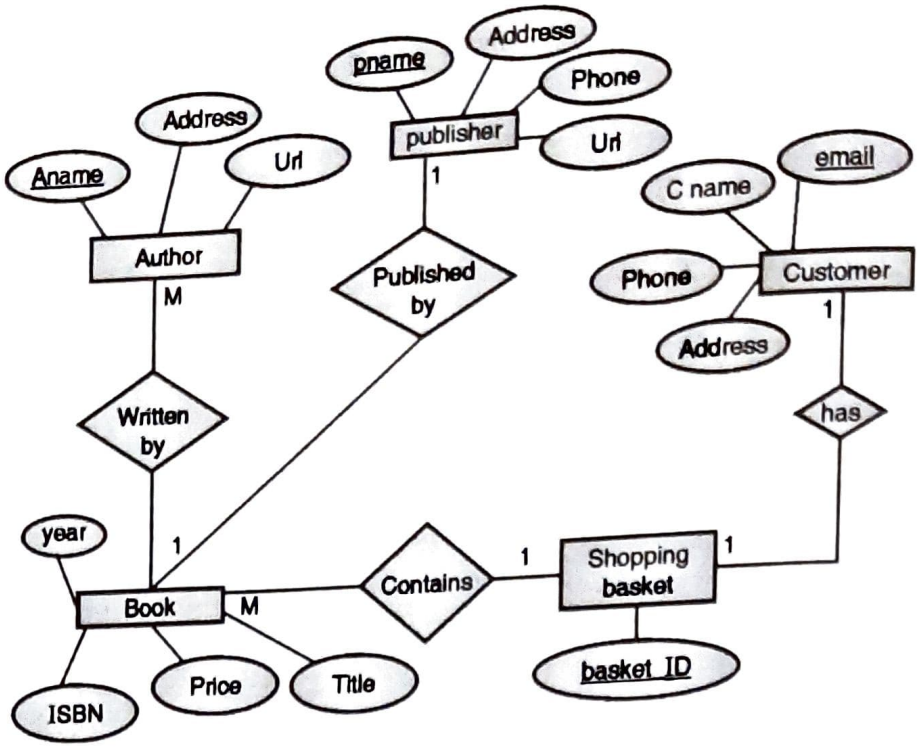


Fig. P.1.11.6

Syllabus Topic : Design Issues

1.12 Design Issues

It is possible to define a set of entities and the relationships among them in a number of different ways. But sometimes it is difficult to decide on the best way to model the application. That means it is hard to find the best option to design. Sometimes it is hard to decide whether something should be represented as an Entity, an attribute or a relationship. These basic issues in the design of an E-R database schema are described as below :

Use of entity sets vs. attributes

- While designing ER-Diagram the question arises as whether a value is represented as a separate entity-set or an attribute?
- The Choice mainly depends on the structure of the enterprise being modeled, and on the semantics associated with the attribute in question.
- Representing a value as a separate entity-set provides a scope to add details later.

For Example

- As shown in Fig. 1.12.1 employee entity has 5 attributes(emp_id, name, phone, city, street).
- An employee has single name but he/she can have multiple phone numbers so it is good to represent phone as a separate entity with two attributes phone_number and location; rather than an attribute. The location stands for office or home. If the phone numbers with different locations are important then we cannot add the phone as attribute, rather we will add it as separate entity having its own attributes.

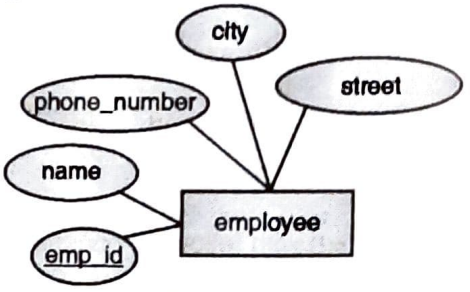


Fig. 1.12.1

- The relationship *emp_phone* can be created between entities employee and phone as follows :

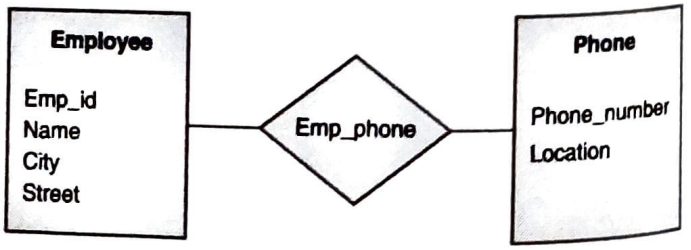


Fig. 1.12.2

- But if the phone number is not so important or detailed concept in the system, then it can be taken as multi-valued attribute.
- Number of times it becomes difficult to set the importance of any element, which makes it complicated to decide whether to make it an attribute or entity.

Use of entity sets vs. relationship sets

- In the situation where ER Diagram is designed to represent loan given to exactly one customer and each loan is given at a particular bank branch.
- We assumed that a loan is modeled as an entity. An alternative is to model a loan as a relationship between customers and branches, with loan-id and amount as descriptive attributes as follows :

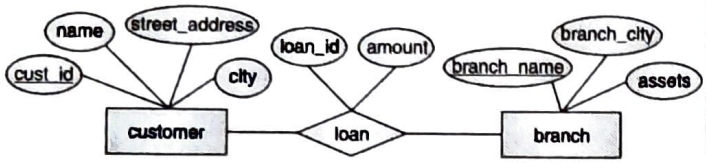


Fig. 1.12.3 : ER diagram for Loan

- Each loan is represented by a relationship between a customer and a branch.
- But in another situation in which several customers hold a joint loan, we must define a separate relationship for each joint-loan-holder.
- So each relationship has the same value for the loan-number attribute and amount attribute. That is more than one customer can hold a loan so the value of loan- id and amount attribute are same for those customers; if for every customer separate relationship is defined then this attribute values are replicated in each relationship.
- This replication causes two problems as :
  1. Waste of storage space
  2. Data in an inconsistent state, if update operation is not well-performed.
- To solve this issue one possible guideline for determining whether to use an entity set or a relation- ship set in ER diagram design, is to choose an entity set to describe an thing and choose a relationship set to describe an action that occurs between entities.
- This approach can also be useful in deciding whether certain attributes may be more appropriately expressed as relationships.

Binary versus n-ary relationship sets

Mostly databases use binary relationships. Some non-binary relationships could actually be better represented by several binary relationships. For example,

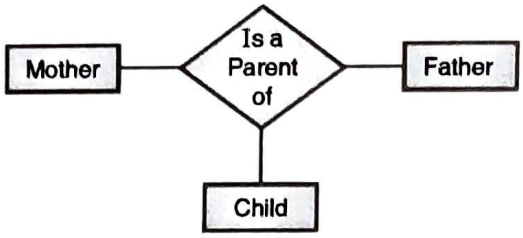


Fig. 1.12.4

Fig. 1.12.4 shows ternary relationship Parent, which relates a Child with his/her Mother and Father. However, it could also be represented by two binary relationships, Mother and Father, relate a Child to his/her Mother and Father separately as follows :

(a)

(b)

Fig. 1.12.5

Advantages of using two relationships Mother and Father allows us to record a child's mother, even if child's father's identity is not known; a null value would be required if the ternary relationship parent is used. This null representation is avoided in binary relationship.

For simplicity, consider the abstract ternary relationship set R, which relates entity sets A, B, and C.

We replace the relationship set R by an entity set E, and create three relationship sets :

- Relationship set RA relates entities E and A.

- Relationship set RB relates entities E and B
- Relationship set RC relates entities E and C

Restricting the E-R model to include only binary relationship sets is not always desirable. It is not possible to translate constraints on the ternary relationship into constraints on the binary relationships.

For example, consider a constraint that says that R is many-to-one relationship from A, B to C; that is, each pair of entities from A and B is associated with at most one in C entity. This constraint cannot be expressed by using cardinality constraints on the relationship sets RA, RB, and RC

Placement of relationship attributes

- Relationship set may have attributes which describe each relation with entities.
- The problem arises where to place this descriptive attribute; whether it is places with one of the entity or it will be placed with the separate relationship-set.

For example

- Consider *loan* relation is related with customer and account entity. If loan relationship has attributes as loan number and amount. We can associate these attributes in the account table or in the loan table.
- The choice of attribute placement is more clear-cut for many-to-many relationship sets. For example: a customer may have one or more number of accounts, and there may be one or more customers for single account.
- To express the date of a specific customer's last account access, then the access date must be an attribute of the depositor relationship set, it should not be one of the participating entities.
- If access-date were an attribute of account, in that case , we could not determine which customer made the most recent access to a joint account.
- When an attribute is determined by the combination of participating entity sets, instead of either entity independently, that attribute must be associated with the many-to-many relationship set.
- In such cases the decision of design: where to place descriptive attributes - as a relationship or entity attribute - should reflect the properties of the business being modeled.
- The cardinality ratio of a relationship can affect the placement of relationship at- tributes. Thus, attributes of one-to-one or one-to-many relationship sets can be associated with one of the participating entity sets, rather than with the relationship set.

Syllabus Topic : Extended ER Features

1.13 Extended ER Features

SPPU - Dec. 13, May 14, Dec. 16

University Questions

Q. Explain the different constraints on specialization/generalization with suitable example. **(Dec. 2013, 4 Marks)**

Q. Explain extended ER features Specialization, Generalization and Aggregation with Example and diagrams. **(May 2014, 8 Marks)**

Q. Explain the concept of specialization & generalization in E-R Model using suitable example. **(Dec. 2016, 5 Marks)**

We can use basic E-R concepts to develop most database features, but to express database deeply some extended features available in ER Model which are known as **Extended ER-Features**. These are as follows :

1.13.1 Specialization

- In an entity set, sometimes further sub grouping is also possible depending upon certain characteristics. For example, a subset may have some attributes which are not shared by other subset in the entity set. In E-R diagram these distinct subsets can be represented by some means.
- Consider an example of entity set person having attributes name, city and street. The entity person may be further classify as follows:
  - o customer
  - o employee
- Both of these types of person are described by a set of attributes that contains all the attributes of entity set person with some additional attributes of their own.
- For example, in the description of customer entities, we may add new attribute like customer-

id, whereas in the description of employee entities we may add new attributes like employee-id and salary.

- This process of creating subgroups within an entity set is called **specialization**. The specialization of entity person helps us to distinguish whether a person is an employee or customer depending upon attributes.
- An entity set may be specialized by more than one distinguishing characters. In the above example, job performed by an employee may be a distinguishing character from customer. The employee may be further divided as permanent or temporary employee depending upon some characteristics.

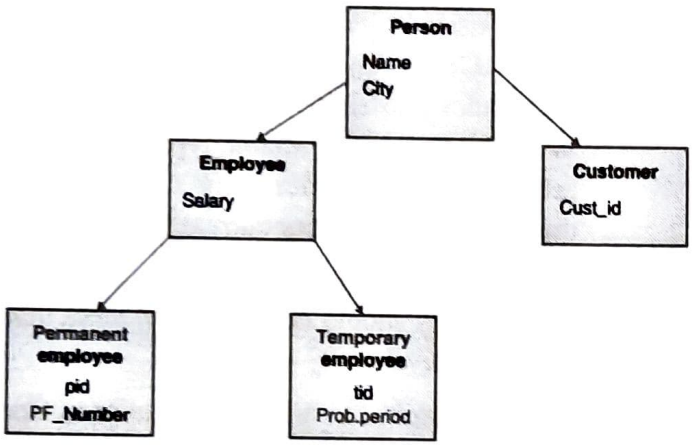


Fig. 1.13.1 : Specialization

1.13.2 Generalization

- In the Fig. 1.13.1, we can observe that the refinement from an initial entity set into subgroups(multiple entity sets) depending upon distinct features shows top-down approch.
- The design process may also proceed into opposite bottom-up approach , in which multiple entity sets are grouped into higher level single entity set depending upon the common characteristics in between these entity sets.
- In out example the entity 'Permananent emplyoee' has following attributes
  - Name
  - City
  - pid
  - Salary
  - Pf_Number
- The entity 'Temporary emplyoee' has following attributes
  - Name
  - City
  - eid
  - Salary
  - Prob_period
- Now if we observe, there are some common attributes between entities 'Permananent emplyoee' and 'Temporary emplyoee'.
- This commonality can be termed as **Generalization** which is containment relationship that exists between higher level entity set and one or more lower level entity set.

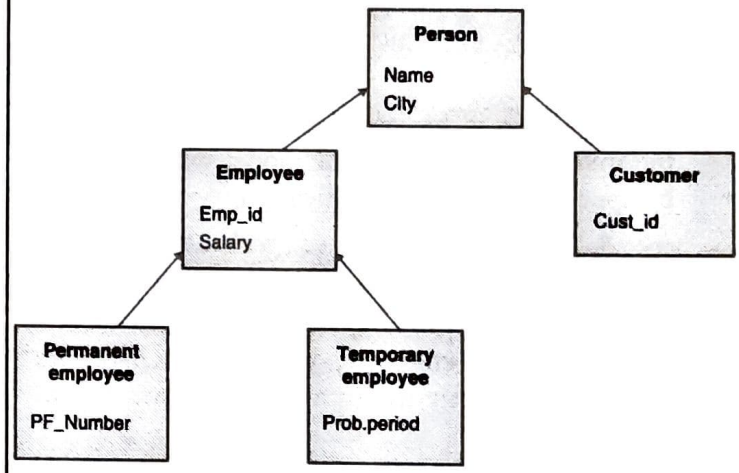


Fig. 1.13.2 : Generalization

- Here the employee is higher level entity set while the entities 'Permananent emplyoee' and 'Temporary emplyoee' are the lower level entity set.
- In this example, the attributes which are conceptually same has different names. e.g. the pid and eid are both nothing but the employee ids of permanent and temporary employees. To create generalization such attributes can be given a common name like emp_id and must be represented in higher level entity *employee*.
- The higher level entity set is also known as Superclass while the lower level entity set also known as subclass.

Attribute Inheritance

- The higher- and lower-level entities created by specialization and generalization has the important property - attribute inheritance
- The attributes of the higher-level entity sets are generally considered to be inherited from the lower-level entity sets.
- In the above example customer and employee both inherit the attributes of person entity set. The description of customer contains name, street, city and additional customer-id attribute. The descriptoin of employee contains name, street, and city and additional employee-id and salary attributes.
- Participation in the relationship sets of higher-level entity is inherted by the lower level entity set.
- Attribute inheritance applies through all tiers of lower-level entity sets. The *employee* and *customer* entity sets can participate in any relationships in which the *person* entity set participates.
- The outcome is basically the same even though the given portion of an E-R model was arrived at by specialization or generalization.
  - A higher-level entity set with attributes and relationships that apply to all of its lower-level entity sets.
  - Lower-level entity sets with unique characters that apply only within a particular lower-level entity set.

Constraints on Generalizations

While designing a database system, the database designer may place certain constraints on a particular generalization.

There are number of such constraints.

(A) One is to detemine which entities can be members of a given lower-level entity set. Such membership may be one of the following :

- **Condition-defined :** In this entity set, it is observed that whether the entity satisfies the specific condition or not.
- Consider an example, assume that the higher-level entity set account contains attribute account-type. Now all account entities are evaluated depending upon the account-type attribute. Only those entities which has account-type as "savings account" are allowed to belong to the lower-level entity set person.
- All entities that has account-type as "current "are included in current account. Here all the lower-level entities are evaluated on the basis of the common attribute as account-type, this type of generalization is called as attribute-defined.
- **User-defined :** This type of lower level entity sets are not constrained by a membership condition. The end user of the database assigns entities to a given entity set. Consider groups are created of students for creating projects. In such case the the teams may be decided by the class teacher.

(B) A second type of constraint check whether entities belong to one or more lower-level entity set within a single generalization.

The lower-level entity sets may be one of the following :

- **Disjoint :** In this generalization , the entity must belong to single lower-level entity set. In the example, the acount_type attribute of account entity satisfies only one condition that it may be saving or current, but not both.
- For a disjointness constraint, it is necessary that an entity should not belong to more than one lower-level entity set.
- **Overlapping :** In this generalizations, One entity may belong to more than one lower-level entity.

Consider an example of IT firm database, where a manager may involve and guide to more than one teams. Hence it appears in more than one lower level team entity sets. Here the generalization is overlapping.

(C) **Completeness :** It specifies whether or not an entity in the higher-level entity set belongs to at least one of the lower-level entity sets within the generalization/ specialization or not.

This constraint may be one of the following :

- **Total generalization or specialization :** Here each higher-level entity must belong to a lower-level entity set.
- **Partial generalization or specialization :** Here higher-level entities may not belong to any lower-level entity set.

1.13.3 Aggregation

In E-R model there is limitation that is it cannot express relationships among relationships. Consider the ternary relationship 'works-on' between a employee, branch, and job. Consider we want to record managers for (employee, branch, job) combinations. An entity set 'manager' is available.

To reprsent this relationship, we can set a quaternary relationships manages between employee, branch, job, and manager. Using the basic E-R modeling constructs, we obtain the following E-R diagram.

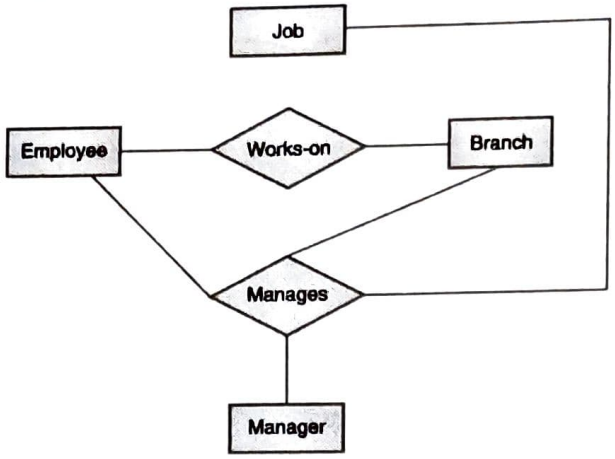


Fig. 1.13.3 : Aggregation

Here the relationship sets 'works-on' and 'manages 'can be combined into one single relationship set. But combining the mis difficult as some employee, branch, job combinations many not have a manager.

Ex. 1.13.1 SPPU - Dec. 2015, 5 Marks

Construct an ER Diagram for a banking Database System. Consider various entities such as Account, Customer, Branch, Loan, Deposit, Borrower, etc. Design Specialization and Generalization EER Feature.

Soln. :

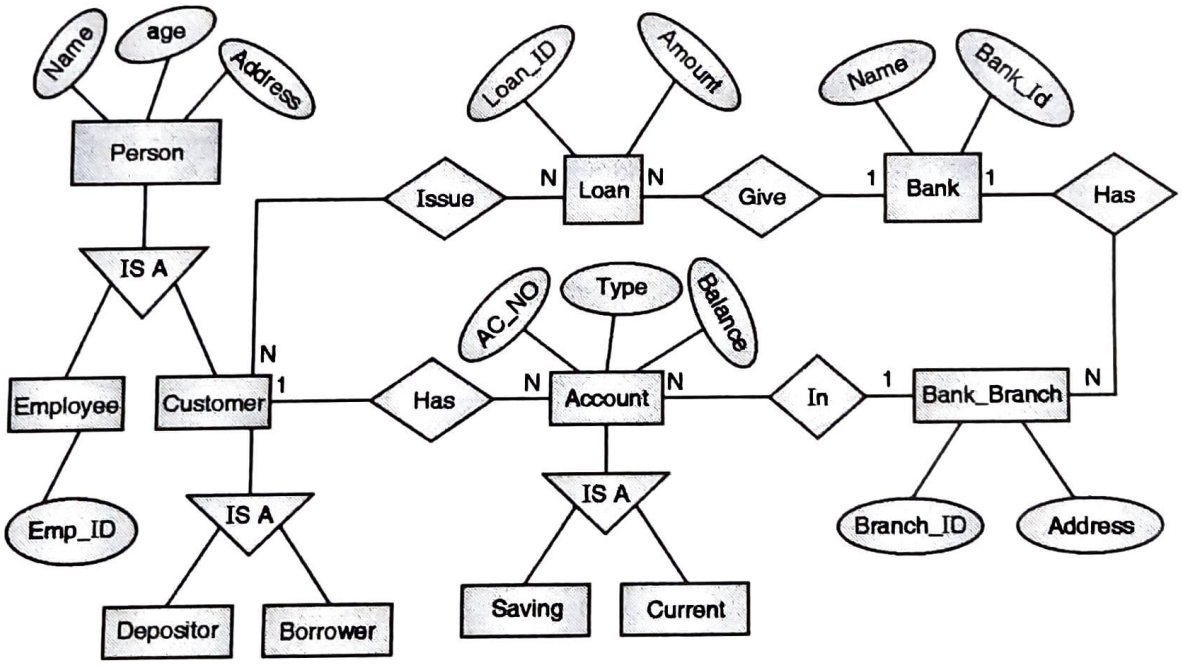


Fig. P. 1.13.1

Ex. 1.13.2 :

Construct an ER Diagram for a Travel Agency. Consider various entities such as travel agency, passenger, Branch, seat, bus, employee, tours etc. Design Specialization and Generalization EER Feature.

Soln. :

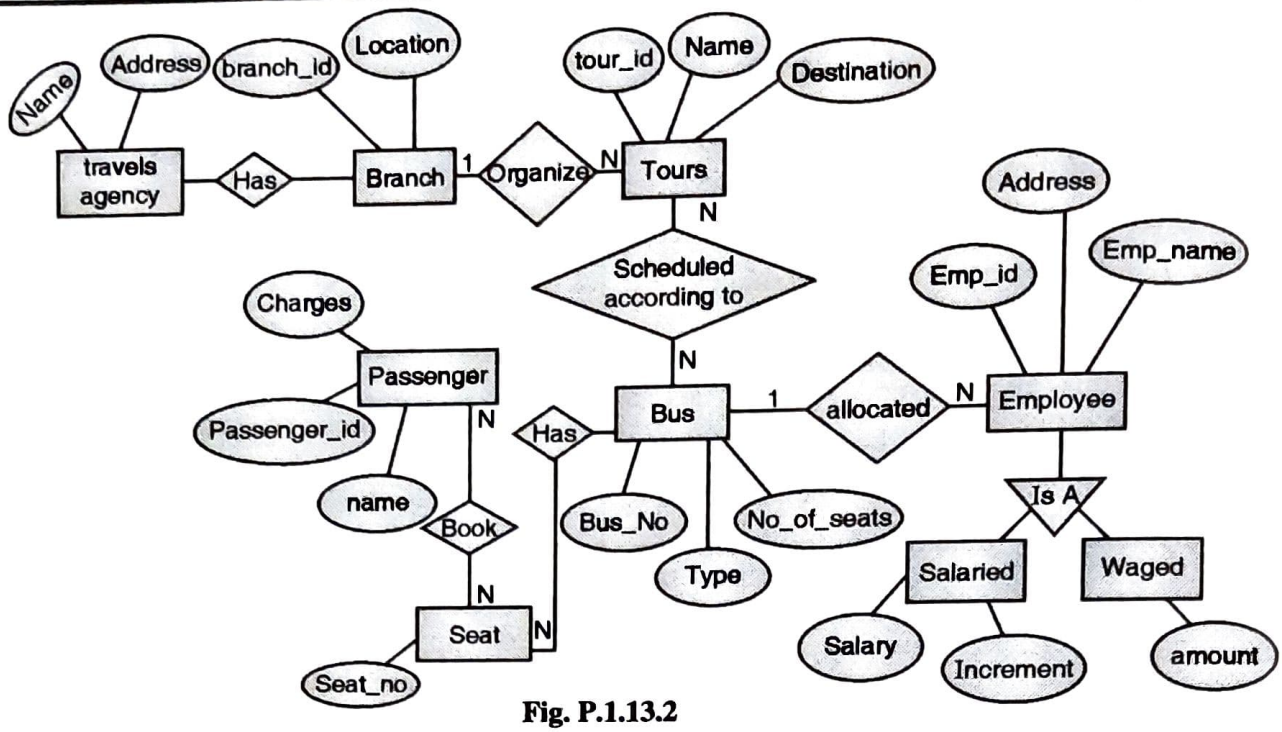


Fig. P.1.13.2

Ex. 1.13.3

For a Library Management System following information is maintained.

Books (Accession_no, Title, Author, Price, Booktype, publisher)

Borrower(Membership_no, Name, Address, Category, max_no_of_books_issued, Accession_no)

Draw E-R Diagram for the above taking into consideration following

Constraints and by making use of at least one Extended ER feature :

i) A book may have more than one author.

ii) There may be more than one copy of a book.

iii) Borrower can be staff or a student. Depending on this category max number of books that can be issued will vary. [i.e. student can ask for max 3 books whereas staff can ask for 10 books]

Soln. :

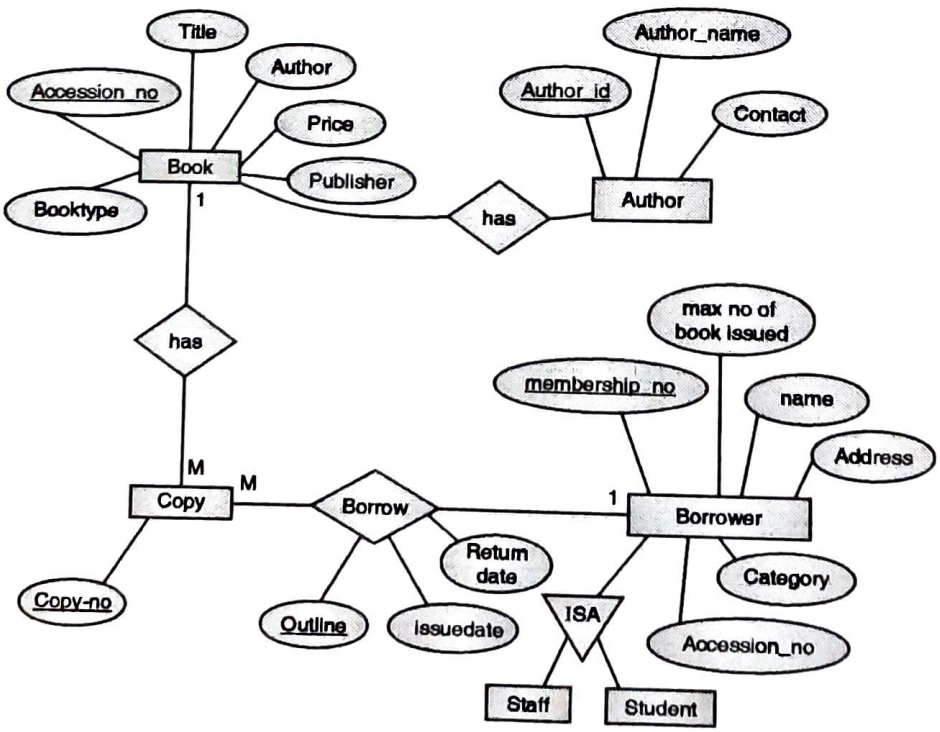


Fig. P.1.13.3 : ER diagram for library management system

Syllabus Topic : Converting ER & EER Diagram into Tables

1.14 Converting ER & EER Diagram into Tables

SPPU - May 13

University Question

Q. Explain with example how E-R diagrams are converted into tables. (May 2013, 6 Marks)

As we know ER Diagram gives us good knowledge of entities and their relations. We can understand various mapping cardinalities from ER-Diagram. Using ER Diagram we can easily create Relational Data Model. It is nothing but the logical view of the database. We can convert these ER Diagrams into the tables. There are various steps involved in conversion of ER diagrams into the table.

An ER diagram consists of :

- Entity sets which are represented by rectangular box.
- Relationship sets which are represented by diamond.

We will convert each entity/relationship set into a tables by using following steps:

Consider example see following ER-Diagram which we are going to convert into table using following steps :

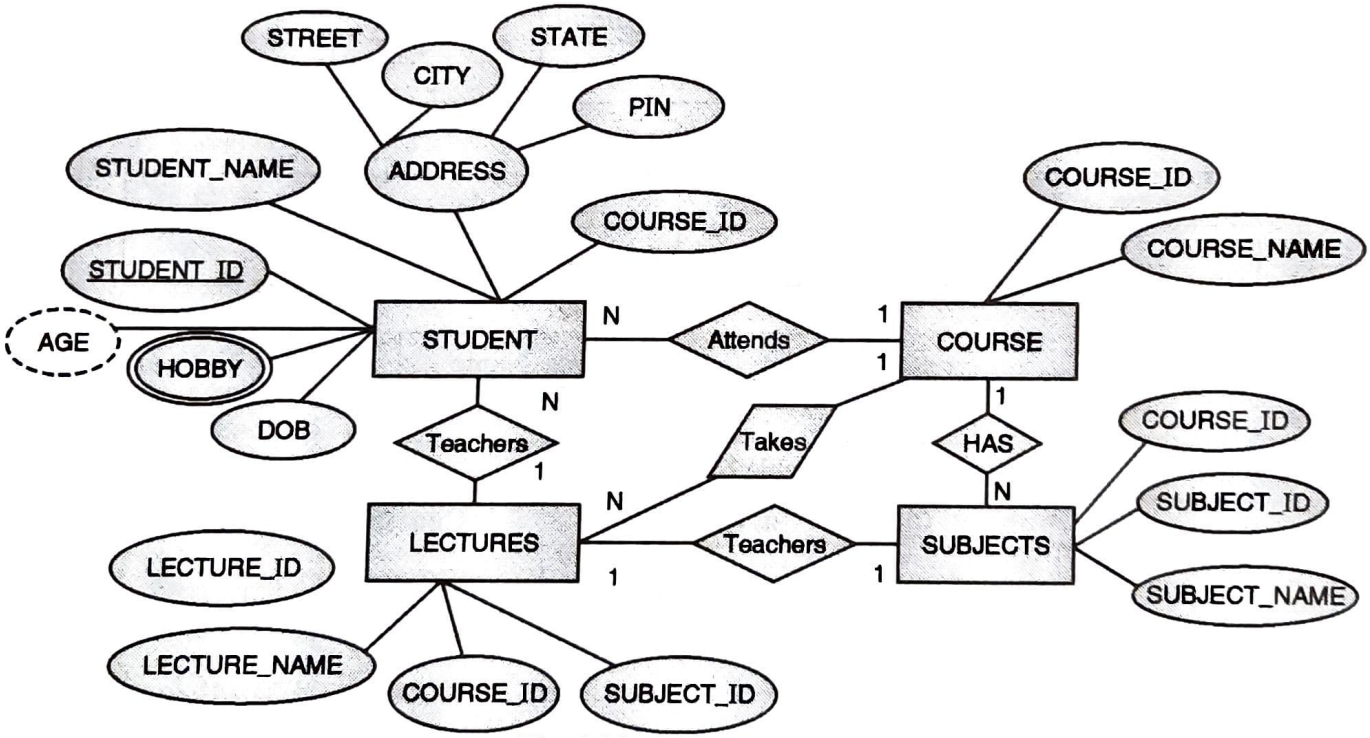


Fig. 1.14.1 : ER diagram

Entity Set ⇒ Table

While converting An ER-Diagram into tables (relational database) first task is to design individual table for each entity in ER diagram. Attributes of entity are treated as fields / columns of tables.

- **Single-valued attribute :** These attributes, whose value is unique, are considered as columns of that table. In the STUDENT Entity, STUDENT_ID, STUDENT_NAME form the columns of STUDENT table. Similarly, LECTURER_ID, LECTURER_NAME form the columns of LECTURER table.
- **Composite Attribute :** Composite attributes are represented in table as individual columns. In above ER diagram, in STUDENT entity ADDRESS is a composite attribute; it is formed by combining STREET, CITY, STATE, and PIN attributes so while converting it into table the table contain columns for STREET, CITY, STATE, and PIN.

- **Multi-valued Attribute :** If any entity has multi-valued attribute then to convert it in table needs to form an individual table for that attribute. In above diagram, in the Student table , hobby attribute is multi-valued. A student can have more than one hobby. So it is difficult to represent multiple values in a single column of STUDENT table. It must be stored separately, so that it is possible to store multiple hobbies. Redundancy in the system should not be created by adding/ removing / deleting hobbies. A separate table STUD_HOBBY with STUDENT_ID and HOBBY as its columns can be created. We can create a composite key using both the columns.
- **Primary key :** In above diagram, STUDENT_ID, LECTURER_ID, COURSE_ID and SUBJECT_ID are the key attributes of the entities. Hence they are the primary keys of respective table.
- **Weak entity :** In above ER diagram no weak entity is present.
- As shown in Fig. 1.14.1 ER-Diagram, it contains 4 entities (STUDENT, COURSE, LECTURER and SUBJECT) which are become independent tables as follows :

Table 1.14.1 : Student

Student_Id	Student_Name	Street	City	State	Pin	Course_Id	Dob

Table 1.14.2 : Stud_Hobby

Student_Id	Hobby

Table 1.14.3 : Lecturer

Lecturer_Id	Lecturer_Name	Course_Id	Subject_Id

Table 1.14.4 : Course

Course_Id	Course_Name

Table 1.14.5 : Subject

Subject_Id	Subject_Name	Course_Id

□□□

CHAPTER 2

UNIT II

SQL and PL/SQL

Syllabus

- **SQL :** Characteristics and advantages
- SQL Data Types and Literals
- DDL, DML, DCL, TCL
- SQL Operators
- Tables : Creating, Modifying, Deleting
- Views : Creating, Dropping, Updating using Views
- Indexes
- SQL DML Queries: SELECT Query and clauses
- Set Operations
- Predicates and Joins
- Set membership
- Tuple Variables
- Set comparison
- Ordering of Tuples,
- Aggregate Functions
- Nested Queries
- Database Modification using SQL Insert, Update and Delete Queries.
- **PL/SQL:** concept of Stored Procedures & Functions
- Cursors, Triggers
- Assertions
- Roles and privileges
- Embedded SQL
- Dynamic SQL

Syllabus Topic : SQL

2.1 SQL - Characteristics and Advantages

- SQL stands for Structured Query Language. SQL is used to communicate with a database. It is the standard language for relational database management systems. SQL statements are used to perform different operations on database like retrieval, insertion, updation and deletion of data.
- SQL is used in various advanced Relational Database Management Systems (RDBMS). Some common RDBMS that use SQL are : MySQL, Oracle, Microsoft Access, Microsoft SQL Server, Sybase, Ingres, etc.
- SQL is developed by IBM as a part of System R project in 1970. Initially it was called as Sequel. SQL was one of the first commercial languages for Edgar F. Codd's relational model. SQL became a standard of the American National Standards Institute (ANSI) in 1986, SQL is a declarative language in which the desired result is given without the specific details about how to accomplish the task.
- The steps required to execute SQL statements are handled transparently by the SQL database. SQL can be characterized as non-procedural because in procedural languages the details of the operations to be specified, such as opening and closing tables, loading and searching indexes, or flushing buffers and writing data to file systems are required which is not necessary in SQL.

Syllabus Topic : Characteristics of SQL

2.1.1 Characteristics of SQL

- SQL is an ANSI and ISO standard computer language for creating and manipulating databases.
- SQL allows the user to create, update, delete, and retrieve data from a database.
- The tokens and syntax of SQL are oriented from English common speech to keep the access barrier as small as possible. Hence it is very simple and easy to learn.
- All the keywords of SQL can be expressed in any combination of upper and lower case characters. It makes no difference whether UPDATE, update, Update, UpDate i.e. the keywords are case insensitive
- SQL is a declarative language, not a procedural one.
- SQL is very powerful language.
- SQL works with database programs like DB2, Oracle, MS Access, Sybase, MS SQL Sever etc.

Syllabus Topic : Advantages of SQL

2.1.2 Advantages of SQL

There are various advantages of SQL

- **High Speed :** SQL Queries can be used to retrieve large amounts of records quickly and efficiently from a database.
- **Portable :** SQL can be run on any platform. Also it can be executed on PCs, laptops, servers and even mobile phones. It runs in local systems, intranet and internet. Databases using SQL can be moved from a device to another without any problems.
- **Well Defined Standards Exist :** SQL databases use long-established standard, which is being adopted by ANSI & ISO. Whereas Non-SQL databases do not adhere to any clear standard.
- **Supports object based programming :** SQL supports various object oriented programming concepts which makes it powerful.
- **Used with all DBMS systems with any vendor :** SQL is used by all the vendors who develop DBMS.
- **No Coding Required :** Using standard SQL it is easier to manage database systems without writing large amount of code.
- **Used for relational databases :** SQL is widely used for number of relational databases.
- **Easy to learn and understand :** SQL mainly consists of English words and hence it is easy to learn and understand the SQL queries.
- **Complete language for a database :** SQL is used to create databases and manage the databases in all aspects.
- **Dynamic database language :** SQL can change the database dynamically at runtime even while the database is being used by users.
- **Can be used as programming and interactive language :** SQL can do both the jobs of being a programming as well as an interactive language at the same time.
- **Client/Server language :** SQL can be used in client server architecture as a mediator between client application and server database.
- **Multiple data views :** We can provide different views(presentations) of contents of a database to different users.
- **Used in internet :** SQL can be used in internet to access the web related data.

Syllabus Topic : SQL Data Types

2.2 SQL Data Types and Literals

2.2.1 SQL Data Types

In SQL, we store the data in tabular format where table (relation) is the combination of rows (tuples) and columns (fields). While creating table we have to assign data types to the columns. These data types are used to decide that which type of data the columns can store.

There are various types of data types in SQL to store different types of data items.

1. CHAR
2. VARCHAR
3. BOOLEAN
4. SMALLINT
5. INTEGER or INT
6. DECIMAL [(p[,s])] or DEC [(p[,s])]
7. NUMERIC [(p,[s])]
8. REAL
9. FLOAT(p)
10. DATE
11. TIME
12. TIMESTAMP
13. CLOB [(length)] or CHARACTER LARGE OBJECT [(length)] or CHAR LARGE OBJECT [(length)]
14. BLOB [(length)] or BINARY LARGE OBJECT [(length)]

(1) CHAR (length)

- The CHAR data type accepts character OR string type of data including Unicode. It is known as fixed length data type. The length of the character string is specified while assigning the data type. For example, CHARACTER(n) where n represents the maximum size of the character string. If size is not specified then the default size will be 1.
- The minimum length of the CHARACTER data type is 1 and maximum length is up to the table page size. Character strings which are larger than the page size of the table can be stored as a Character Large Object (CLOB).
- If value having lower size than the size of CHAR data type is stored in it, then the remaining space is filled with blanks characters. That means it gets wasted.
- If value having greater size than the size of CHAR data type is tried to store, then the extra characters are truncated.

Examples :

CHAR(20) or CHARACTER(20)

'Phoenix', 'INFOTECH'

(2) VARCHAR (length)

- The VARCHAR data type accepts character OR string type of data including Unicode. It is known as variable length data type.
- The length of the character string is specified while assigning the data type which indicates the maximum number of characters it can accept.
- We can assign the length from 1 to the current table page size.
- If value having lower size than the size of VARCHAR data type is stored in it, then the remaining space will get reutilized. That means the memory does not get wasted.
- If you need to store character strings that are longer than the current table page size, the Character Large Object (CLOB) data type should be used.

Examples : VARCHAR(10)

'Phoenix', 'INFOTECH'

(3) BOOLEAN

- The BOOLEAN data type can accepts value either TRUE or FALSE. No need to declare size while declaring the BOOLEAN data type.
- TRUE or FALSE are case insensitive. If you attempt to assign any other value to a BOOLEAN data type, an error gets raised.

Examples : TRUE, true, True, False

(4) SMALLINT

- The SMALLINT data type is used to accept numeric values with default scale as zero. It stores any integer value between the range 2^ -15 and 2^15 -1. Attempting to assign values outside this range causes an error.
- If you assign a numeric value with a precision and scale to a SMALLINT data type, the scale portion truncates, without rounding.

Examples : SMALLINT

-32768, 0, −13.7 (digits to the right of the decimal point are truncated), 32767

(5) INTEGER or INT

- The INTEGER data type is used to accept numeric values with a default scale as zero. It stores any integer value between the range 2^ -31 and 2^31 -1. Attempting to assign values outside this range causes an error.
- If you assign a numeric value with a precision and scale to an INTEGER data type, the scale portion truncates, without rounding.

Examples

– 345, 0, 4532.98 (digits to the right of the decimal point are truncated), 167

(6) DECIMAL [(p[,s])] or DEC [(p[,s])]

- The DECIMAL data type is used to accept floating point values for which you define a precision and a scale in the data type declaration. The precision is a positive integer that represents the total number of digits that the number will contain (precision + scale).
- The scale is a positive integer that represents the number of digits of decimal places which will occur to the right of the decimal point. The scale for a DECIMAL cannot be larger than the precision.
- If you exceed the number of digits expected to the left of the decimal point, an error is thrown. If you exceed the number of expected digits to the right of the decimal point, the extra digits are truncated.

Examples : DECIMAL (10, 3)

98789, 765.123, 10.1234(Final digit is truncated), -987, -897.786, -1234567.1234 (Final digit is truncated)

(7) NUMERIC [(p[,s])]

It is same as of Decimal

(8) FLOAT (p)

The FLOAT data type accepts approximate numeric values, for which you may define a precision up to a maximum of 64. The default precision is 64 if not declared.

Examples : FLOAT(8)

12345678, 1.2, 123.45678, -12345678, -1.2, -123.45678

(9) DATE

- The DATE data type accepts date type of values. No need to assign size while declaring a DATE data type. Date values should be specified in the form: YYYY-MM-DD.
- The value of month must be between 1 and 12, value of day should be between 1 and 31 depending on the month and value of year should be between 0 and 9999. The values should be enclosed in single quotes, preceded by the keyword DATE.

Examples : DATE '1999-01-01'
DATE '2000-2-2'

(10) TIME

- The TIME data type accepts time values. No parameters are required when declaring a TIME data type. The format is : HH:MM:SS. The fractional value can be used to represent nanoseconds.
- The minutes and seconds values must be two digits. Hour values should be between zero 0 and 23, minute values should be between 00 and 59 and second values should be between 00 and 61.999999.
- Values assigned to the TIME data type should be enclosed in single quotes, preceded by keyword TIME.

Examples : TIME '1:10:20'

Syllabus Topic : SQL Literals

2.2.2 SQL Literals

The literal is the constant or fixed data value. For example 'Phoenix', 'Institute' are string literals while 100, 5 are numerical literals and so on. The String, date and time literals are always enclosed in single quotation marks while the numerical literals are without quotation marks. Literal are case sensitive.

SQL Supports following types of literals.

(1) Numeric Literals
(2) Character Literals
(3) String Literals

(4) Date Literals
(5) Time Literals

(1) Numeric Literals

Numeric literals are the sequence of digits proceeded by an optional sign (+ / –) and with an optional decimal point. The Numeric Literals are further classified into two categories :

Integer Literals : These are the whole numbers assigned as data values. An integer can store a maximum of 38 digits of precision.

For example : 100, + 7, – 8 etc.

Number or Floating Point Literals : These are the numbers with decimal point assigned as data values. For example : 10.2, +7.3, -8.2 etc.

(2) Character Literals

Character literal contains single character enclosed in single quotation marks.

For example : 'A', '%', '9', 'z', '('

(3) String Literals

- These are the sequence of characters enclosed in single quotes. In SQL versions up to 7.0, the maximum length of a string literal is 1024. From version 7.1, there is no specific maximum limit on number of characters.
- The character string literals cannot be enclosed in double quotes because it is reserved for delimiting identifiers such as field or table names.

For example : 'Phoenix', 'I' etc.

(4) Date Literals

- These literal are the date type of values in the ANSI date format 'YYYY-MM-DD' or the default date format specified in the application via the 'set date format' operation.
- The date literal is enclosed in single quotation marks.

For Example : '2017-03-18'

(5) Time Literals

- These literal are the Time type of values in the format 'HH:MM' or 'HH:MM:SS'. In addition 'am' or 'pm' can be included at the end for 12-hour time format.
- If the am/pm indicator is excluded, it is assumed that the time is in 24-hour format.
- The date literal is enclosed in single quotation marks.

'11:15', '8:30 am', '06:25:15 pm', '17:00'

Syllabus Topic : DDL, DML, DCL, TCL

2.3 DDL, DML, DCL, TCL

SPPU - May 13, May 14

University Questions

Q. Explain various database Languages. **(May 2013, 8 Marks)**

Q. Write short note on DDL, DML and DCL. **(May 2014, 4 Marks)**

The database languages are categorized as DDL, DML, DCL and TCL.

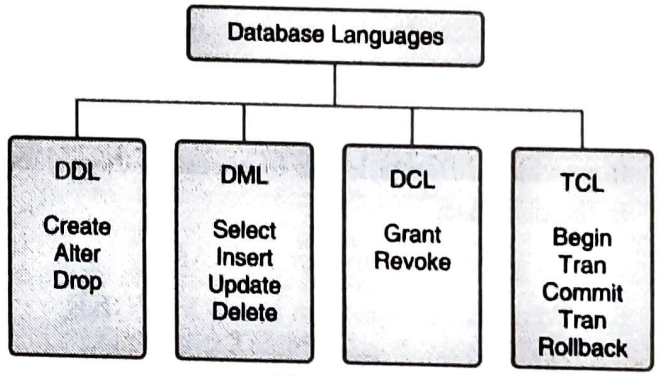


Fig. 2.3.1

2.3.1 Data Definition Language (DDL)

- This language allows the users to define data and their relationship to other types of data. It is used to create data tables, dictionaries, and files within databases.
- The DDL is also used to specify the structure of each table, set of associated values with each attribute, integrity constraints, security and authorization information for all the tables and physical storage structure of all the tables on the disk.
- Let's take SQL for instance to categorize the statements that comes under DDL.
  - To create the database instance – **CREATE**
  - To alter the structure of database – **ALTER**
  - To drop database instances – **DROP**
  - To rename database instances – **RENAME**

2.3.2 Data Manipulation Language (DML)

- The Data Manipulation Language (DML) is used for accessing and manipulating data in a database. DML provides a set of functionalities to support the basic data manipulation operations on the data stored in the database.
- It allows users to access, insert, update, and delete data from the database.
  - To access or read records from table – **SELECT**
  - To insert record into the table – **INSERT**
  - Update the records in table – **UPDATE**
  - Delete the records from the table – **DELETE**

2.3.3 Data Control Language (DCL)

- Data Control Language (DCL) is used to control the user access to the database related elements like tables, views, functions, procedures and packages.
- It provides different levels of access to the objects in the database.
  - To grant access to user – **GRANT**
  - To revoke access from user – **REVOKE**

Grant : GRANT is used to provide the privileges to the users on the database objects. The privileges could be select, delete, update and insert on the tables and views. On the procedures, functions and packages it gives select and execute privileges.

Revoke : REVOKE removes the privileges given on the database objects. All the privileges can be removed at a time or one or more privileges can also be removed from the objects as per requirement.

2.3.4 Transaction Control Language (TCL)

TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.

- **BEGIN Transaction** – opens a transaction
- **COMMIT Transaction** – commits (Save permanently) transactions
- **ROLLBACK Transaction** – ROLLBACK (Cancels. undo) transactions in case of any issue

Syllabus Topic : SQL Operators

2.4 SQL Operators

- An operator is a character or reserved word used in SQL statements to perform different operations like arithmetic or comparison.
- Operators are used to specify conditions in an SQL statement and also used to integrate multiple conditions in SQL statement.

  - Arithmetic operators
  - Comparison operators
  - Logical operators
  - Operators used to negate conditions

2.4.1 Arithmetic Operators

Consider two operands x and y where value of x is 10 while y is 5.

Operators	Descriptions	Examples
+ (Add)	It adds the operands	x + y = 15
- (Subtract)	It subtracts right hand operand from left hand operand	x – y = 5
* (Multiply)	It multiply both operands	x * y = 50
/ (Divide)	It divides left hand operand by right hand operand	x / y = 2
% (Modulo)	It divides left hand operand by right hand operand and returns reminder	x % y = 0

2.4.2 Comparison Operator

Consider two operands x and y where value of **x** is **10** while **y** is **5**.

Operators	Descriptions	Examples
=	Check whether both the operands have same values, if yes condition becomes true.	x=y is not true
>	Check whether left operand is greater than right, if yes condition becomes true	x>y is true
<	Check whether left operand is less than right, if yes condition becomes true	x<y is not true

Operators	Descriptions	Examples
>=	Check whether left operand is greater than or equal to the right operand or not, if yes condition become true	x>=y is true
<=	Check whether left operand is less than or equal to the right operand or not, if yes condition become true	x<=y is not true
!=	Check whether both the operands have same values or not, if not condition become true.	x != y is true
<>	Check whether both the operands have same values or not, if not condition become true.	x<>y is true
!>	Check whether left operand is not greater than the value of right operand	x!>=y is not true
!<	Check whether left operand is not less than the right operand value	x!<=y is true

2.4.3 Logical Operators

Operator	Description
ALL	TRUE if all of a set of comparisons are TRUE.
AND	TRUE if both Boolean expressions are TRUE.
ANY	TRUE if any one of a set of comparisons is TRUE.
BETWEEN	TRUE if the operand is within a range.
EXISTS	TRUE if a sub-query contains any rows.
IN	TRUE if the operand is equal to one of a list of expressions.
LIKE	TRUE if the operand matches a pattern.
NOT	Reverses the value of any other Boolean operator.
OR	TRUE if either Boolean expression is TRUE.
SOME	TRUE if some of a set of comparisons are TRUE.

2.4.4 Bitwise Operators

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise excusive OR

2.4.5 Compound Operators

Operator	Description
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

Syllabus Topic : Tables - Creating, Modifying, Deleting

Note : All the queries are implemented considering MySQL.

2.5 Tables : Creating, Modifying, Deleting

In DBMS the standard format of storing the data is table. Table is also known as relation. It is the combination of rows and columns.

2.5.1 Creating Table

The **CREATE TABLE** statement is used to create table in database.

Syntax

```
CREATE TABLE table_name (
   column1 datatype[size],
   column2 datatype [size],
   column3 datatype[size],
   ....
);
```

In the CREATE TABLE statement the column parameters specify the names of the columns or fields of the table. The data type is the type of data which we want to store in the respective fields. The fields can hold data of different types like char, varchar, number, date etc.

The optional size value can also be mentioned after the data type. This size value indicated the maximum length of data for the field. If size is not given then default value depending upon the data type is assigned.

Example

Student Table

```
CREATE Table student
(roll_no integer(3), stud_name varchar(20),
bdate date , marks integer(3));
```

In the above example, a table student will be created with following attributes. The roll_no field will contain numerical value of maximum digit 3. The stud_name field will contain string value of maximum length 20. The date field will contain date type value of standard date length. For date data type no need to mention size. The marks field will contain numerical value of maximum digit 3.

The structure of the table will be as follows after record insertion.

Table 2.5.1 : Student

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

Employee Table

```
CREATE Table employee
(emp_id integer(3), emp_name varchar(20),
Salary integer(7), department varchar(20));
```

Product Table

```
CREATE Table product
(prod_code integer(3), prod_name varchar(20),
price integer(7), category varchar(20));
```

Customer Table

```
CREATE Table customer
(cust_id integer(3), cust_name varchar(20),
address varchar(20), email_id varchar(20));
```

2.5.1.1 Creating New Table from Existing Table

AS SELECT clause is used to create table from existing table. It can be considered as copy of existing table. While creating such table, we have option whether to take all records, fields from existing table or not. We can copy just structure of the existing table also. The different ways of coping table are as given below :

Consider the existing table student as shown in Table 2.5.1. Creating new table same as of existing table.

Syntax

```
Create table table_name
as select * from existing_table_name;
```

For Example

```
Create table newstudent1
as select * from student ;
```

Output : The newly created table will be

Table 2.5.2 : Newstudent1

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

Creating new table having specific fields but all the records from existing table.

Syntax

```
Create table table_name
as select field_1,field_2... from
existing_table_name;
```

For Example

```
Create table newstudent2
as select roll_no,stud_name from student;
```

Output : The newly created table will be

Table 2.5.3 : Newstudent2

roll_no	stud_name
101	Kunal
102	Jay
103	Radhika
104	Sagar
105	Supriya

Creating new table having specific records but all the fields from existing table.

Syntax

```
Create table table_name
as select * from existing_table_name
where condition
```

For Example

```
Create table newstudent3
as select * from student
where marks > 80;
```

The newly created table will be structure wise same as of existing table, but it will contain records of only those students who got marks above 80.

Output

Table 2.5.4 : Newstudent3

roll_no	stud_name	Bdate	marks
101	Kunal	12-02-2000	90
103	Radhika	05-04-2000	85

Creating new table having no records but all the fields from existing table. That means copying only structure of existing table.

Syntax

```
Create table table_name
 as select * from existing_table_name
where false condition
```

For Example

```
Create table newstudent4
as select * from student
where 1=2;
```

Here 1=2 is the false condition. The newly created table will be structure wise same as of existing table, but it will no records get copied.

Output

Table 2.5.5 : Newstudent4

roll_no	stud_name	bdate	marks

2.5.2 Modifying Table

ALTER TABLE query is used to modify structure of a table. We can add, delete or modify column.

Adding New Column in a Table

```
ALTER TABLE table_name
ADD column_name datatype;
```

For Example : Adding column grade in the table student.

```
ALTER TABLE student
ADD Grade varchar(2);
```

Dropping Column from Table

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

For Example : Deleting column grade from the table student.

```
ALTER TABLE student
DROP column Grade;
```

Modifying Column of a Table

```
ALTER TABLE table_name
MODIFY COLUMN column_name data_type;
```

For Example : Changing the data type and size of column roll_number of student table.

```
ALTER TABLE student
modify column roll_no varchar(4);
```

Here we are changing the.

Deleting all the records from Table

Syntax

```
TRUNCATE TABLE table_name;
```

For Example : Deleting all the records from newstudent1

```
TRUNCATE TABLE newstudent1;
```

2.5.3 Deleting Table

DROP TABLE query is used to delete table permanently from the database.

Syntax

```
drop table table_name;
```

For Example : Deleting the newstudent1 table from the database.

```
Drop table newstudent1;
```

Syllabus Topic : Views - Creating, Dropping, Updating View

2.6 View : Creating, Dropping, Updating View

SPPU - May 15

University Question

Q. Explain view objects in SQL with example. **(May 2015, 3 Marks)**

View : In SQL, a view is a virtual table containing the records of one or more tables based on SQL statement executed. Just like a real table, view contains rows and columns. You can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table. The changes made in a table get automatically reflected in original table and vice versa.

Purpose of View : View is very useful in maintaining the security of database. Consider a base table employee having following data.

Emp_id	emp_name	Salary	Address
E1	Kunal	8000	Camp
E2	Jay	7000	Tilak Road
E3	Radhika	9000	Somwar Peth
E4	Sagar	7800	Warje
E5	Supriya	6700	LS Road

1. Now just consider we want to give this table to any user but don't want to show him salaries of all the employees. In that we can create view from this table which will contain only the part of base table which we wish to show to the user.

   See the following View.

Emp_id	emp_name	Address
E1	Kunal	Camp
E2	Jay	Tilak Road
E3	Radhika	Somwar Peth
E4	Sagar	Warje
E5	Supriya	LS Road

2. Also in multiuser system, it may be possible that more than one user may want to update the data of same table. Consider two users A and B want to update the employee table. In such case we can give views to both these users. These users will make changes in their respective views, and the respective changes are done in the base table automatically.

2.6.1 Creating View

Consider existing table student

Table 2.6.1 : Student

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

1. Creating view having all records and fields from existing table

Syntax

```
CREATE or replace VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

For Example : Creating a view of base table student with same structure and all the records.

```
Create or replace view stud_view1
as select * from student;
```

Output

Table 2.6.2 : stud_view1

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

2. Creating view having specific fields but all the records from existing table

Syntax

```
Create or replace view view_name
as select field_1,field_2...
from existing_table_name;
```

For Example

```
Create or replace view stud_view2
as select roll_no, name from student;
```

Output : The newly created view will be

Table 2.6.3 : stud_view2

roll_no	stud_name
101	Kunal
102	Jay
103	Radhika
104	Sagar
105	Supriya

3. Creating new view having specific records but all the fields from existing table

Syntax

```
Create or replace view view_name
as select * from existing_table_name
where condition;
```

For Example

```
Create or replace view sud_view3
as select * from student
where marks > 80;
```

Output

Table 2.6.4 : stud_view3

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
103	Radhika	05-04-2000	85

2.6.2 Updating View

Update query is used to update the records of view. Updation in view reflects the original table also. Means the same changes will be made in the original table also.

Syntax

```
UPDATE view_name
set field_name = new_value;
where condition;
```

For Example : We are updating marks to 73 of student having roll_no 102.

```
UPDATE stud_view1
set marks = 73
where roll_no=102;
```

In this case marks of roll_no 102 will get updated in both view view1 as well as table student

Output : View - View1

roll_no	stud_name	bdate	marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	**73**
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

Output

Table 2.6.5 : Student

roll_no	stud_name	bdate	Marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	**73**
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

There are some restrictions on the modification with respect to view.

- In case of view containing joins between multiple tables, only insertion and updation in the view is allowed, deletion is not allowed
- Data modification is not allowed in the view which is based on union queries.
- Data modification is not allowed in the view where GROUP BY or DISTINCT statements are used.
- In view the text and image columns can't be modified.

2.6.3 Dropping View

DROP query is used to delete a view.

Syntax

```
DROP view view_name;
```

For Example

```
DROP view stud_view2;
```

Syllabus Topic : Indexes

2.7 Indexes

SPPU - May 15

University Question

Q. Explain Index objects in SQL with example. **(May 2015, 3 Marks)**

- Sometimes the data in the database is very large. For example in the application of State Bank of India, the database related to customers and their

transactions is very large. In such case retrieval of data from such huge database becomes slower.

- Indexes are the special lookup tables which are available to only database search engine for accessing data. Indexes speed up data retrieval effectively.
- An index is a pointer to data in a table. It is similar to the alphabetical index of a book present at the end of book. An index is used to speed up SELECT queries and also WHERE clauses.
- But because of indexes the data input related to INSERT and UPDATE statements get slow down.
- Indexes can be created or dropped with no effect on the data.

Creating Index

CREATE INDEX statement is used to create an index. In this statement we have to mention name of the index, the table and column, and whether the index is in ascending or descending order.

There are different types of indexes.

2.7.1 Single Column Index

This is index is created on a single column of a table.

Syntax

```
CREATE INDEX index_name
ON table_name (column_name)
```

For Example

```
CREATE INDEX ind1
on student(stud_name);
```

2.7.2 Composite Index

Sometimes duplicate records may available in columns. In such case the composite indexing is better option to index the data. This index is created on a multiple columns of a table.

Syntax

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

For Example

```
CREATE INDEX ind2 on student(stud_name , marks);
```

2.7.3 Unique Index

A unique index does not allow any duplicate values to be inserted into the table.

Syntax

```
CREATE UNIQUE INDEX index_name
ON table_name (column1, column2, ...);
```

For Example

```
CREATE UNIQUE INDEX ind3
on student(stud_name);
```

2.7.4 Implicit Index

Implicit indexes are indexes that are automatically created by the database server when an object is created. Such indexes are created for primary key and unique constraints.

Displaying Index : To display index information regarding table following query is used.

Syntax

```
Show index from table_name;
```

For Example

```
Show index from student;
```

Syllabus Topic : SQL DML Queries - Select Query and Clauses

2.8 SQL DML Queries - Select Query and Clauses

2.8.1 SELECT Query

SELECT query is used to retrieve the data from database. SELECT query never make any change in the database. The data returned by the SELECT query is in the form of result sets.

Syntax

```
SELECT column_1, column_1... from table_name;
```

For Example : Consider the table student

Table 2.8.1 : Student

roll_no	stud_name	bdate	Marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68

roll_no	stud_name	bdate	Marks
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

```
SELECT roll_no, stud_name from student;
```

Output

roll_no	stud_name
101	Kunal
102	Jay
103	Radhika
104	Sagar
105	Supriya

The names of columns specify data from which columns we want to display. If we want data from all the columns then no need to mention column names. '*' symbol represent all the columns.

For Example

```
SELECT * from student;
```

Output

roll_no	stud_name	bdate	Marks
101	Kunal	12-02-2000	90
102	Jay	07-08-1999	68
103	Radhika	05-04-2000	85
104	Sagar	13-02-2000	70
105	Supriya	11-08-1999	72

With SELECT statement different clauses can be used to display the data as per our requirements.

2.8.2 WHERE Clause

WHERE clause is used to specify condition in SELECT statement while fetching records from the database. The WHERE clause filters the data to be retrieved. The records satisfying the condition given by where clause are retrieved.

Syntax

```
SELECT column_1,columns_2... from table_name
where condition;
```

For Example

```
select * from student where marks > 80;
```

Output

roll_no	stud_name	Bdate	marks
101	Kunal	12-02-2000	90
103	Radhika	05-04-2000	85

```
Select * from student where ename = 'Kunal';
```

Output

roll_no	stud_name	Bdate	marks
101	Kunal	12-02-2000	90

2.8.3 DISTINCT Clause

This clause is used to avoid selection of duplicate rows. Consider there are duplicate values in JOB column of emp table

Table 2.8.2 : Emp

Eno	Ename	Job	Sal
101	Susheel	Clerk	12000
102	Ajay	Manager	18000
103	Dinesh	Clerk	10000
104	Bharati	Manager	17000
105	Prajakta	Salesman	13000

Syntax

```
SELECT distinct( column_name) from table_name;
```

For Example

```
select distinct(job) from emp;
```

Output

Job
Clerk
Manager
Salesman

2.8.4 GROUP BY Clause

The GROUP BY clause is used in collaboration with the SELECT statement. It helps to arrange similar data into groups. It is also used with SQL functions to group the result from one or more tables.

Table 2.8.3 : Emp1

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

1. Display sum of salaries department wise.

```
Select deptno,sum(sal) from emp group by deptno;
```

Output

DEPTNO	SUM(SAL)
30	9400
20	10875
10	8750

2. Display maximum salaries groping on the basis of job

```
Select job,max(sal) from emp group by job;
```

Output

JOB	MAX(SAL)
CLERK	1300
SALESMAN	1600
PRESIDENT	5000
MANAGER	2975
ANALYST	3000

In general we use WHERE clause to give some condition to filter the data. But WHERE clause is not allowed in collaboration with GROUP BY clause. HAVING clause is used with GROUP by clause to specify condition.

2.8.5 HAVING Clause

1. Display sum of salaries of department 10

```
Select deptno,sum(sal) from emp group by deptno
having deptno = 10;
```

Output

DEPTNO	SUM(SAL)
10	8750

2. Display sum of salaries of department 10 and 20

```
Select deptno,sum(sal) from emp group by deptno
having deptno in (10,20);
```

Output

DEPTNO	SUM(SAL)
20	10875
10	8750

Syllabus Topic : Database Modification using SQL Insert, Update and Delete Queries

2.9 Database Modification using SQL Insert, Update and Delete Queries

2.9.1 Insert

This query comes under the category Data Definition Language. After creation of table the insert command is used to insert one or more records in the table.

Insert query has different forms
(Consider Table 2.8.2 Emp)

(1) Inserting values in all columns

Syntax

```
Insert into table_name
values (value1, value2....)
```

For example

```
Insert into emp
values(106,'Rajesh','Clerk',12000);
```

(2) Inserting values in specific columns

Sometimes we may not have all values to insert into table. In such case the syntax will be

```
Insert into table_name(column1,column2...)
values(val1,val2...);
```

For example, consider we do not have value for salary while inserting a new record in emp table. Then the query will be

```
insert into emp(Eno,Ename,Job)
values(107,'Ankur','Salesman');
```

(3) Inserting records from existing table into new table

We can also take records from existing table to add into new table using as select clause. Consider new table emp1 in which we will add records from Table 2.9.1. Here we can mention condition using where clause to take specific records.

```
Insert into emp1
Select eno,ename,job,sal from emp
Where sal > 15000;
```

Output

Table 2.9.1 : Emp1

Eno	Ename	Job	Sal
102	Dinesh	Manager	18000
104	Bharati	Manager	17000

2.9.2 Update

Sometimes changes to the database become necessary. To make changes in the database 'update' command is used. Updations can be done in single or multiple columns based on the given condition. The update command consists of 'set' clause and an optional 'where' clause'

Syntax

```
Update table_name set column_name = new_value
[where condition]
```

For Example

```
Update emp set sal = 15000;
```

This query will make salary of all the employees to 15000.

Output

Eno	Ename	Job	Sal
101	Susheel	Clerk	15000
102	Dinesh	Manager	15000
103	Dinesh	Clerk	15000
104	Bharati	Manager	15000
105	Prajakta	Salesman	15000

'WHERE' clause is used to make changes in specific records.

For Example

```
Update emp set sal = 20000 where job = 'Manager';
```

This query will change salary of mangers to 20000.

Output

Eno	Ename	Job	Sal
101	Susheel	Clerk	15000
102	Dinesh	Manager	20000
103	Dinesh	Clerk	15000
104	Bharati	Manager	20000
105	Prajakta	Salesman	15000

2.9.3 Delete

As per requirement, the records from existing table can be removed using delete command. Delete command can have 'WHERE' clause optionally.

Syntax

```
Delete from table_name;
```

For Example

```
Delete from emp;
```

This query will delete all the records from Table 2.9.1.

```
Delete from emp where Eno = 103;
```

This query will delete the record of employee with employee number 103 from emp table.

Syllabus Topic : Set Operations

2.10 Set Operations

- Set operations are supported by SQL to be performed on table data. For these operations special operators knows as Set Operators are used. Set operators are used to join the results of multiple SELECT statements.
- These operators help to get meaningful results from data, under different specific conditions. Queries which contain set operators are called compound queries.
- The different Set Operators are as follows

(i) Union (ii) Union All
(iii) Intersect (iv) Minus

Consider following two tables Emp and Dept

Table 2.10.1 : Emp

Empno	Ename	Job	DeptNo	Salary
101	Rahul	Manager	10	17000
102	Vinay	Clerk	20	12000
103	Kunal	Manager	30	18000
104	Rajesh	Salesman	20	13000
105	Kushal	Clerk	10	11000

Table 2.10.2 : Dept

DeptNo	DeptName	Loc
10	Sales	Mumbai
20	Production	Pune
30	Accounts	Nasik
40	Research	Bangalore

2.10.1 Union

The union operator returns all distinct rows selected by either query

Syntax

```
Select column_name from table_1
Union
Select column_name from table_2
```

For Example

```
Select DeptNo from emp
union
select Deptno from dept
```

Output

10
20
30
40

2.10.2 Union All

The Union All operator returns all rows selected by either query including duplicates.

Syntax

```
Select column_name from table_1
Union all
Select column_name from table_2
```

For Example

```
Select DeptNo from emp
Union all
select Deptno from dept
```

Output

10
20
30
20
10
10
20
30
40

2.10.3 Intersect

The intersect operator returns only those rows which are common to both the queries

Syntax

```
Select column_name from table_1
intersect
Select column_name from table_2
```

For Example

```
Select DeptNo from emp
intersect
select Deptno from dept
```

Output

10
20
30

2.10.4 Minus

Minus operator displays the rows which are present in the first query but absent in the second query, with no duplicates and data is arranged in ascending order by default.

Syntax

```
Select column_name from table_1
minus
Select column_name from table_2
```

For Example

```
Select DeptNo from dept
intersect
select Deptno from emp
```

Output

40

Syllabus Topic : Predicates and Joins

2.11 Predicates and Joins

2.11.1 Predicates

Predicate is an expression that evaluates to TRUE, FALSE or UNKNOWN. Predicates are used in the search condition of WHERE and HAVING clauses, the join conditions of FROM clauses and other constructs where value in Boolean format is expected.

Consider the table

Table 2.11.1 : Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

SQL provides various types of predicates.

2.11.1.1 Comparison Predicate

Comparison predicate is the combination of two expressions separated by a comparison operator. There are six types of comparison operators: =, >, <, >=, <=, <>. The data of NUMERIC type is compared with respect to their algebraic values. The data of CHARACTER STRING type is compared with respect to their alphabetic order.

= Equal to Predicate

```
Select * from emp
where ename = 'KING'
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10

> Greater Than Predicate

```
Select * from emp
where sal > 3000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10

< Less Than Predicate

```
Select * from emp
where sal < 3000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

>= Greater Than Equal To Predicate

```
Select * from emp
where sal >= 3000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20

<= Less Than Equal To Predicate

```
Select * from emp
 where sal <= 3000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

<> Not Equal To Predicate

```
Select * from emp
where sal <> 3000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

Combination of Predicates can be used with AND operator

```
Select * from emp
where sal >=3000 and sal <=5000;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20

2.11.1.2 Between Predicate

Between predicate is used to specify certain range of values. The AND keyword is used in this predicate.

Syntax

```
test_expression [ NOT ] BETWEEN begin_expression AND end_expression
```

Example

```
Select * from emp
where comm between 300 and 500;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30

In "Between" predicate the NOT keyword can also be used

```
Select * from emp
 where comm not between 300 and 500;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30

2.11.1.3 In Predicate

IN predicate particularly determines whether the value of expression given to test matches any value in specified the list.

Just consider that we want display records of employees from depno 10 and 20.

Then the query will be

```
Select * from emp
where deptno in(10,20)
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7934	MILLER	CLERK	7782	01/23/1982	1300		10

2.11.1.4 Like Predicate

Like operator determines whether a specific character string matches the given pattern or not. In the pattern we can use regular characters and wildcard characters. In this pattern matching, it is necessary that regular characters must exactly match the characters specified in the character string. However, for the wildcard characters arbitrary fragment matching of the character string is done. The use of wildcard characters makes the LIKE operator more flexible.

Example : Display records of employee whose names starts with letter 'J'

Query

```
select * from emp
where ename like 'J%';
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7900	JAMES	CLERK	7698	12/03/1981	950		30

Example : Display records of employee whose names ends with letter 'N'

Query

```
select * from emp
where ename like '%N';
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30

Example : Display records of employee whose names contains 'L' as second character.

Query

```
select * from emp
 where ename like '_L%';
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30

Example : Display records of employee whose names contains character 'A' anywhere;

Query

```
select * from emp
where ename like '%A%';
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30

2.11.1.5 IS [NOT] NULL

When values for some attributes are not available then, NULL value is assigned. To display records having NULL value, IS NULL predicate is used.

Example : Display records of employees who never get any commission.

```
select * from emp
where comm IS NULL;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

The NOT keyword can be used to get values opposite to given condition

Example : Display records of employees who get commission.

```
select * from emp
where comm IS NOT NULL;
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30

2.11.2 Joins

SPPU - Dec. 13, May 14

University Questions

Q. What are different types of joins in SQL? Explain with suitable example. **(Dec. 2013, 6 Marks)**

Q. Explain different types of joins with example. **(May 2014, 8 Marks)**

- A JOIN is a means for combining columns from one (self-table) or more tables by using values common to each.
- There are following types of Joins.

1. INNER
2. OUTER
3. LEFT OUTER
4. RIGHT OUTER
5. FULL OUTER
6. SELF

To understand joins consider following two tables.

Table 2.11.2 : Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

Table 2.11.3 : Dept

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

- In the database having large amount of data, it is not possible to store the entire data in a single table. The related data may be stored in different tables. In above tables emp and dept the data of employees is stored.
- The EMP table contains the basic information of employee like employee number, name, job(post), salary department number etc. The DEPT table contains the information of same employees about their depar names and locations.
- If we want to display whole information means basic information with department names and locations then we have to combine these two tables in query using joins.
- For using joins, we required a common column between both the tables. In this example the EMP and DEPT tables contains a common column DEPTNO.

2.11.2.1 Inner Join (Equi Join)

The INNER JOIN is used to display records that have matching values in both tables.

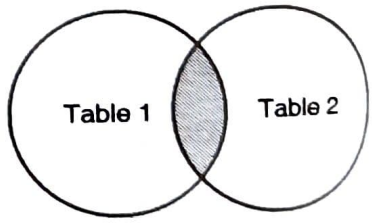


Fig. 2.11.1

Syntax

```
Select column_name_list from table_1
INNER JOIN table_2
ON table_1.column_name = table_2.column_name
```

Example

```
select ename,job,sal,emp.deptno,dept.dname from emp
INNER JOIN dept
on emp.deptno = dept.deptno;
```

Output

ENAME	JOB	SAL	DEPTNO	DNAME
CLARK	MANAGER	2450	10	ACCOUNTING
MILLER	CLERK	1300	10	ACCOUNTING
KING	PRESIDENT	5000	10	ACCOUNTING
FORD	ANALYST	3000	20	RESEARCH
SCOTT	ANALYST	3000	20	RESEARCH
JONES	MANAGER	2975	20	RESEARCH
SMITH	CLERK	800	20	RESEARCH
ADAMS	CLERK	1100	20	RESEARCH
WARD	SALESMAN	1250	30	SALES
MARTIN	SALESMAN	1250	30	SALES
TURNER	SALESMAN	1500	30	SALES
JAMES	CLERK	950	30	SALES
ALLEN	SALESMAN	1600	30	SALES
BLAKE	MANAGER	2850	30	SALES

The INNER JOIN keyword selects all rows from both tables Emp and Dept as long as there is a match between the columns. If there are records in the "Dept" table that do not have matches in "Emp", then such records will not get displayed. In Dept table department OPERATIONS of number 40 is present, but it is not displayed as no matching record is available in table emp.

2.11.2.2 Outer Join

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

(1) Left Outer Join (2) Right Outer Join (3) Full Outer Join

Consider following two tables Stud_data1 and Stud_data2

Table 2.11.4 : Stud_data1

Roll No	NAME
1	Rahul
2	Kunal
3	Jay
4	Vinay
5	Preeti

Table 2.11.5 : Stud_data2

Roll No	Address
1	Mumbai
2	Pune
3	Nasik
7	Bangalore
8	Goa

(1) Left Outer Join

The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. Null values are shown at the place of right table values.

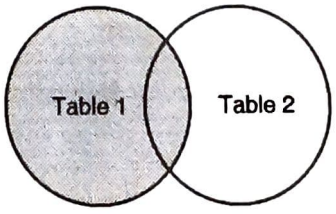


Fig. 2.11.2

Left Outer Join syntax is,

```
SELECT column-name-list
from table-name1
```

```
LEFT OUTER JOIN
table-name2
on table-1.column-name = table-2.column-name;
```

Example

```
SELECT * FROM Stud_data1
LEFT OUTER JOIN Stud_data2 ON
Stud_Data1.rollno = Stud_Data2.Rollno);
```

Output

ID	Name	ID	Address
1	Rahul	1	Mumbai
2	Kunal	2	Pune
3	Jay	3	Nasik
4	Vinay	NULL	NULL
5	Preeti	NULL	NULL

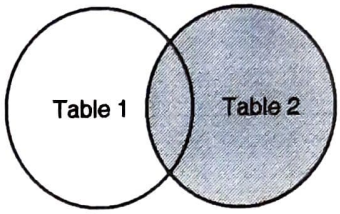


Fig. 2.11.3

(2) Right Outer Join

Returns all rows from the right table even if there are no matches in the left table. Null values are shown at the place of left table values.

Syntax

```
select column-name-list from table-name1
RIGHT OUTER JOIN table-name2
on table-1.column-name = table-2.column-name;
```

Example

```
SELECT * FROM Stud_Data1
RIGHT OUTER JOIN Stud_Data2
on (Stud_Data1.rollno= Stud_Data2. rollno);
```

Output

ID	Name	ID	Address
1	Rahul	1	Mumbai
2	Kunal	2	Pune
3	Jay	3	Nasik
NULL	NULL	7	Bangalore
NULL	NULL	8	Goa

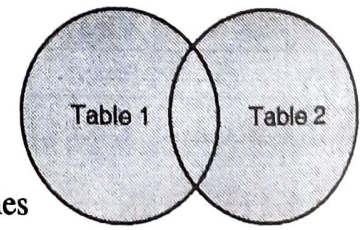


Fig. 2.11.4

(3) Full Outer Join

The full outer join returns a result with the **matching data** of both the tables and then remaining rows of both **left** table and then the **right** table.

Syntax

```
select column-name-list from table-name1
FULL OUTER JOIN table-name2
on table-1.column-name = table-2.column-name;
```

Example

```
SELECT * FROM Stud_Data1
FULL OUTER JOIN Stud_Data2
on (Stud_Data1.rollno= Stud_Data2. rollno);
```

Output

ID	Name	ID	Address
1	Rahul	1	Mumbai
2	Kunal	2	Pune
3	Jay	3	Nasik
4	Vinay	NULL	NULL
5	Preeti	NULL	NULL
NULL	NULL	7	Bangalore
NULL	NULL	8	Goa

2.11.2.3 SELF Join

- Self join is used to join a table to it-self as if the table were two tables. Virtual copies of the table are considered. Consider the table Emp

Table 2.11.6 : Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

- Here the MGR number indicates the name of MANAGER of particular employee. For Example MGR of MILLER is 7782 which is EMPNO of CLARK. That mean CLARK is manager of MILLER.
- Now we want to display list of employees with their manager names. In this case we have to join this emp table to itself. We will consider two copies of emp table, emp A and emp B. From emp A we will retrieve employee names while from emp B we will get manager names;

Query

```
select A.ename "Employee" ,B.ename "Manager"  from emp A, emp B where A.mgr = B.empno;
```

Output

Employee	Manager
FORD	JONES
SCOTT	JONES
ALLEN	BLAKE
JAMES	BLAKE
TURNER	BLAKE
MARTIN	BLAKE
WARD	BLAKE
MILLER	CLARK
ADAMS	SCOTT
BLAKE	KING
CLARK	KING
JONES	KING
SMITH	FORD

Ex. 2.11.1 SPPU - Dec. 2014, 3 Marks

Consider Following Relational Tables :
Instructor (ID, name, dept name)
Student (ID, dept_name,tot_cred)
Takes (ID, course-id, sec-id, semester, year)
Course (course –id, title, dept-name, credits)
Dept (Dept-id, dept -name)
Solve following queries using SQL
Design above relation using SQL DDL statement, primary key and foreign key.

Soln. :

```
Create table instructor(id varchar(5), name varchar(10), dept_name varchar(10), primary key(id));
Create table student(id varchar(5),dept_name varchar(10), tot_cred int(5),foreign key(id) references
instructor(id), foreign key(dept_name) references instructor(dept_name));

Create table takes(id varchar(5),course_id varchar(10), sec_id varchar(10), sem int(2), year int(2) , foreign
key(id) references instructor(id));

Create table course(course_id varchar(5),title varchar(10), dept_name varchar(10), credits int(4), foreign
key(course_id) references takes(course_id));
Create table dept(dept_id varchar(5),dept_name varchar(10),foreign key(dept_name) references
instructor(dept_name));
```

Ex. 2.11.2 SPPU - Oct. 2016(In sem), 5 Marks

Consider the relational database
Supplier (Sid, Sname, address)
Parts (Pid, Pname, color)
Catalog (sid, pid, cost)

Write SQL queries for the following requirements:

i) Find name of all parts whose color is green.

ii) Find names of suppliers who supply some red parts.

iii) Find names of all parts whose cost is more than Rs.25.

Soln. : Consider the following 3 tables

```
mysql> select *from supplier;
+-----+-------+---------+
| sid | sname | address |
+-----+-------+---------+
| s1  | abc   | Pune    |
| s2  | pqr   | Mumbai  |
| s3  | xyz   | Nasik   |
+-----+-------+---------+
3 rows in set (0.08 sec)

mysql> select *from parts;
+-----+-------+-------+
| pid | pname | color |
+-----+-------+-------+
| p1  | prod1 | red   |
| p2  | prod2 | green |
| p3  | prod3 | blue  |
+-----+-------+-------+
3 rows in set (0.05 sec)
```

```
mysql> select *from catalog;
+------+------+------+
| sid  | pid  | cost |
+------+------+------+
| s1   | p1   |   30 |
| s2   | p3   |   10 |
| s3   | p2   |   40 |
+------+------+------+
3 rows in set (0.00 sec)

mysql> _
```

i) Find name of all parts whose color is green.

```
Select * from parts where color = green;
```

Output

```
mysql> select* from  parts where color='green';
+-----+-------+-------+
| pid | pname | color |
+-----+-------+-------+
| p2  | prod2 | green |
+-----+-------+-------+
1 row in set (0.02 sec)

mysql>
```

ii) Find names of suppliers who supply some red parts.

```
Select s.sname, p.color from supplier s, parts p, catalog c where s.sid = c.sid and p.pid = c.pid and p.color ='red';
```

Output

```
c:\wamp\bin\mysql\mysql5.5.24\bin\mysql.exe
mysql> Select s.sname, p.color from supplier s,
 c.sid and p.pid = c.pid and p.color ='red';
+-------+-------+
| sname | color |
+-------+-------+
| abc   | red   |
+-------+-------+
1 row in set (0.02 sec)

mysql>
```

iii) Find names of all parts whose cost is more than Rs.25

```
select p.pname, c.cost from parts p, catalog c where p.pid = c.pid and c.cost > 25;
```

```
c:\wamp\bin\mysql\mysql5.5.24\bin\mysql.exe
mysql> select p.pname, c.cost from parts p, catalog c
    -> where p.pid = c.pid and c.cost > 25;
+-------+------+
| pname | cost |
+-------+------+
| prod1 |   30 |
| prod2 |   40 |
+-------+------+
2 rows in set (0.05 sec)

mysql>
```

Ex. 2.11.3 SPPU - Dec. 2015, 3 Marks

Consider Following Relational Tables :

Person (pname, street, city)

works_for (pname, cname, salary)

Company (cname, city)

Manages (pname, mname)

Solve following quries using SQL

Find the street and city of all employees who work for the "Idea", live in Pune, and earn more than Rs. 3000.

Soln. : Consider the following tables.

```
c:\wamp\bin\mysql\mysql5.5.24\bin\mysql.exe
mysql> select*from person;
+-------+------------+--------+
| pname | street     | city   |
+-------+------------+--------+
| abc   | Tilak Road | Pune   |
| pqr   | MG Road    | Mumbai |
| XY    | CR Road    | Nasik  |
+-------+------------+--------+
3 rows in set (0.01 sec)

mysql>
```

```
mysql> select *from company;
+----------+--------+
| cname    | city   |
+----------+--------+
| Airtel   | Mumbai |
| Idea     | Pune   |
| Reliance | Nasik  |
+----------+--------+
3 rows in set (0.00 sec)

mysql>
```

```
mysql> select *from works_for;
+-------+----------+--------+
| pname | cname    | salary |
+-------+----------+--------+
| xy    | Idea     |   4000 |
| abc   | Airtel   |   5000 |
| pqr   | Reliance |   6000 |
+-------+----------+--------+
3 rows in set (0.00 sec)

mysql> _
```

i) Find the street and city of all employees who work for the "Idea", live in Pune, and earn more than Rs. 3000.

```
Select p.street,p.city from person p,works_for w where w.cname='Idea' and p.city='Pune' and w.salary>3000;
```

Output

```
mysql> Select p.street,p.city from person p,works_for
    -> w where w.cname='Idea' and p.city='Pune' and
    -> w.salary>3000;
+------------+------+
| street     | city |
+------------+------+
| Tilak Road | Pune |
+------------+------+
1 row in set (0.00 sec)
```

Ex. 2.11.4 SPPU - Dec. 2013, 4 Marks

Consider Following Relational Tables:

Student (Roll-no, name, address)

Subject (Sub_code,Sub-name)

Marks (Roll-no, Sub-code, marks)

Solve following queries using SQL

(i) Find out average marks of each student, along with the name of student.

(ii) Find how many students have failed in the subject "DBMS".

Soln. :

```
Create table student(roll_no int(3) , name varchar(10),address varchar(20), primary key(roll_no));
Create table subject(sub_code varchar(10),sub_name varchar(10), primary key(sub_code));
```

Create table marks(roll_no int(3) , sub_code varchar(10),marks int(3), foreign key(roll_no) references student(roll_no), foreign key(sub_code) references subject(sub_code));

Consider following tables

```
mysql> select *from manages;
+-------+-------+
| pname | mname |
+-------+-------+
| xy    | aaa   |
| pqr   | bbb   |
| abc   | ccc   |
+-------+-------+
3 rows in set (0.00 sec)

mysql>
```

```
mysql> select *from subject;
+----------+----------+
| sub_code | sub_name |
+----------+----------+
| c        | C Prog   |
| jv       | Java     |
| or       | Oracle   |
+----------+----------+
3 rows in set (0.00 sec)
```

```
mysql> select *from marks;
+---------+----------+-------+
| roll_no | sub_code | marks |
+---------+----------+-------+
|       1 | or       |    90 |
|       2 | c        |    80 |
|       3 | jv       |    78 |
+---------+----------+-------+
3 rows in set (0.00 sec)
```

(i) Find out average marks of each student, along with the name of student.

Select st.name,su.sub_code, m.marks from student st,subject su,marks m where st.roll_no = m.roll_no and su.sub_code=m. sub_code;

Output

```
+------+----------+-------+
| name | sub_code | marks |
+------+----------+-------+
| abc  | or       |    90 |
| pqr  | c        |    80 |
| xyz  | jv       |    78 |
+------+----------+-------+
3 rows in set (0.02 sec)

mysql>
```

(ii) Find how many students have failed in the subject "Java".

```
Select st.name, su.sub_name, m.marks from student st,subject su,marks m where st.roll_no = m.roll_no and
su.sub_code=m. sub_code and  m.marks > 40 and su.sub_name='Java';
```

Output : Empty Set as no student is there who failed in Java.

Syllabus Topic : Tuple variables

2.12 Tuple Variables

A Relation R can be listed number of times as per the requirements. In this situation we need a way to refer to each occurrence of R. SQL allows us to define, for each occurrence of R in the FROM clause with the help of an "alias". This alias is known as **tuple variable.** When the R is used in the FROM clause, it is followed by the keyword AS which is optional and the name of the tuple variable.

We will see the example which we have already studied in SELF JOIN

Consider the table emp

Table 2.12.1 : Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

Here the MGR number indicates the name of MANAGER of particular employee.

For Example MGR of MILLER is 7782 which is EMPNO of CLARK. That mean CLARK is manager of MILLER.

Now we want to display list of employees with their manager names. In this case we have to join this emp table to itself.

We will consider two copies of emp table, emp A and emp B. From emp A we will retrieve employee names while from emp B we will get manager names;

Query

```
select A.ename,B.ename from emp A, emp B where A.mgr = B.empno;
```

Output

Employee	Manager
FORD	JONES
SCOTT	JONES
ALLEN	BLAKE
JAMES	BLAKE
TURNER	BLAKE
MARTIN	BLAKE
WARD	BLAKE
MILLER	CLARK
ADAMS	SCOTT
BLAKE	KING
CLARK	KING
JONES	KING
SMITH	FORD

Here A and B are **tuple variables.**

Syllabus Topic : Ordering of Tuples

2.13 Ordering of Tuples

To arrange the displayed rows in ascending or descending order on given field(column) , Order By Clause is used.

Syntax

```
Select * from table_name order by col1,col2..[desc]
```

For Example : We need to display the employee information as per their names in ascending order, the query will be

```
Select * from emp order by Ename;
```

Output

Eno	Ename	Job	Sal
102	Ajay	Manager	18000
104	Bharati	Manager	17000
103	Dinesh	Clerk	10000
105	Prajakta	Salesman	13000
101	Susheel	Clerk	12000

Now to display same information in descending order on job the query will be

```
Select * from emp order by Ename desc;
```

Output

Eno	Ename	Job	Sal
101	Susheel	Clerk	12000
105	Prajakta	Salesman	13000
103	Dinesh	Clerk	10000
104	Bharati	Manager	17000
102	Ajay	Manager	18000

Sometimes same records may available in field given for sorting criteria. In such case we can give names of more than one columns for sorting purpose. If data in first column is same, in such case the data of second column can be taken into consideration for sorting. Consider following table

Eno	Ename	Job	Sal
101	Susheel	Clerk	12000
102	Dinesh	Manager	18000
103	Dinesh	Clerk	10000
104	Bharati	Manager	17000
105	Prajakta	Salesman	13000

Here names of two employees is same 'Dinesh'. Now sort the data we can mention sorting fields as ename and job.

```
Select * from emp order by ename,job;
```

Output

Eno	Ename	Job	Sal
104	Bharati	Manager	17000
103	Dinesh	Clerk	10000
102	Dinesh	Manager	18000
105	Prajakta	Salesman	13000
101	Susheel	Clerk	12000

Syllabus Topic : Aggregate Functions

2.14 Aggregate Functions

Aggregate functions perform a calculation on a set of values and return a single value. Usually these functions ignore NULL values(except for COUNT).

There are different types of aggregate functions :

(1) Min (2) Max (3) Sum
(4) Avg (5) Count

Consider the table emp

(1) **Min() :** This function returns smallest value from specified column of the table.

Query

```
Select min(sal) from emp;
```

Output

```
800
```

(2) **Max() :** This function returns greatest value from specified column of the table.

Query

```
Select max(sal) from emp;
```

Output

```
5000
```

(3) **Sum() :** This function returns sum of all the values of specified column of the table.

Query

```
Select sum(sal) from emp;
```

Output

```
29025
```

(4) **Avg() :** This function returns average of all the values of specified column of the table.

Query

```
Select avg(sal) from emp;
```

Output

```
2073.21
```

(5) **Count() :** This function returns total number of values of specified column of the table.

Query

```
Select count(ename) from emp;
```

Output

```
14
```

Syllabus Topic : Nested Queries

2.15 Nested Queries

Writing a query inside another query is known as nested query or subquery. The inner query gets executed first, then the output on inner query is given as input to outer query. Consider the previous emp table

Example : To display records of employees working in SMITH's department

```
Select * from emp where deptno =
(select deptno from emp where ename = 'SMITH');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7876	ADAMS	CLERK	7788	01/12/1983	1100		20

Example : To display records of employees whose salary is more than the salary of FORD

```
Select * from emp where sal >
(select sal from emp where ename = 'FORD');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10

Example : To display records of employees who are senior to JONES

```
Select * from emp where hiredate <
(select hiredate from emp where ename = 'JONES');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30

Syllabus Topic : Set Membership

2.16 Set Membership

It is used to check if value of expression is matching a set of values produced by a subquery or not. There are two keywords used for SET membership – IN and NOT IN. IN is connective test for set of membership while the NOT IN is connective test for absence of SET membership.

Example – In keyword

Display records of employees working in SMITH's department

```
Select * from emp where deptno in
(select deptno from emp where ename = 'SMITH');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7566	JONES	MANAGER	7839	04/02/1981	2975		20

Example – Not In keyword

Display records of employees who are not working in SMITH's department

```
Select * from emp where deptno not in
(select deptno from emp where ename = 'SMITH');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7934	MILLER	CLERK	7782	01/23/1982	1300		10
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7839	KING	PRESIDENT		11/17/1981	5000		10
7900	JAMES	CLERK	7698	12/03/1981	950		30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30

Syllabus Topic : Set Comparison

2.17 Set Comparison

In nested queries, comparison operators are used with WHERE clause to specify the condition to filter the data to be display. Following are the various comparison operators

Comparison Operator	Description
=	Equal
<>	Not Equal
>	Greater Than
<	Less Than
>=	Greater Than or Equal
<=	Less Than or Equal

Example : Consider following table emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

< operator in nested query

Example : Display list of employees having salary less than ADAMS.

```
select * from emp where sal <
(select sal from emp where ename = 'ADAMS');
```

Output

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/1980	800		20
7900	JAMES	CLERK	7698	12/03/1981	950		30

Syllabus Topic : PL/SQL

PL/SQL stands for Procedure Language / Structure Query Language. It is the combination of SQL along with the procedural features of programming languages.

PL/SQL includes procedural language elements such as conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variables of those types, and triggers. It can handle exceptions (runtime errors). Arrays are supported involving the use of PL/SQL collections. It has included features associated with object-orientation. One can create PL/SQL units such as procedures, functions, packages, types, and triggers, which are stored in the database for reuse by applications.

Syllabus Topic : Concept of Stored Procedures and Functions

2.18 Concept of Stored Procedures and Functions

2.18.1 Stored Procedures SPPU - May 13

University Question

Q. Explain Stored Procedures. **(May 2013, 4 Marks)**

Stored procedure is a group of SQL statements which can be executed repeatedly. It allow for variable declarations, flow control and other useful programming techniques. Parameters are the important part of procedures. The parameters make the stored procedure more flexible and useful.

Consider the table

```
mysql> select *from  student;
+-----+-------+------------+-------+
| rno | name  | dob        | class |
+-----+-------+------------+-------+
|   2 | aa    | 1990-11-21 | NULL  |
|   3 | priya | 1989-10-30 | NULL  |
|   4 | asd   | 2000-02-02 | NULL  |
|   5 | ggg   | 0000-00-00 | NULL  |
+-----+-------+------------+-------+
4 rows in set (0.00 sec)

mysql>
```

Syntax

```
DELIMITER //
CREATE procedure procedure _name
([parameter(s)])
STATEMENTS
DELIMITER //
```

IN Parameter : Accepts value when procedure get called.

Example : Create a procedure which should accept rollno as parameter and display the record.

```
DELIMITER //
CREATE PROCEDURE display(IN r integer (3))
 BEGIN
 SELECT * FROM students WHERE rno = r;
 END //
DELIMITER ;
```

Now the procedure can be called as

```
CALL display(3);
```

Output

```
+-----+-------+------------+-------+
| rno | name  | dob        | class |
+-----+-------+------------+-------+
|   3 | priya | 1989-10-30 | NULL  |
+-----+-------+------------+-------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.06 sec)
```

Out Parameter : The value of an OUT parameter can be changed inside the stored procedure. The changed value is passed back to the calling program. The initial value of OUT parameter cannot be accessed by the procedure.

Example

```
DELIMITER $$
CREATE PROCEDURE display1(IN r INT,OUT nm
VARCHAR(25))
BEGIN
select name into nm from students where rno = r;
END $$
DELIMITER ;
```

Now the procedure can be called as

```
CALL display1(3 ,@n);
```

Here n is the OUT parameter which stores the name of student having rno 3;

Then execute the command

```
Select @n;
```

Output

```
+-------+
| @n    |
+-------+
| priya |
+-------+
1 row in set (0.00 sec)

mysql>
```

2.18.2 Stored Functions SPPU - May 13

University Question

Q. Explain Stored functions. **(May 2013, 3 Marks)**

Stored Function is same as of stored procedure means it is a group of SQL statements which can be executed repeatedly. It allow for variable declarations, flow control and other useful programming techniques.

Just difference is that function can return value.

Syntax :

```
CREATE FUNCTION function_name ([parameter(s)])
   RETURNS data type
DETERMINISTIC
STATEMENTS
```

Create a function to return record of student having rollno 3.

```
DELIMITER |
CREATE FUNCTION anualsal (sal int)
  RETURNS int(7)
   DETERMINISTIC
    BEGIN
     DECLARE asal int(7);
Set asal = sal * 12;
    RETURN asal;
    END|
```

Then execute the command

Select anualsal(5000) from dual;

Output

```
mysql> Select anualsal(5000) from dual;
+----------------+
| anualsal(5000) |
+----------------+
|          60000 |
+----------------+
1 row in set (0.06 sec)
```

Syllabus Topic : Cursor, Trigger

2.19 Cursor, Trigger, Assertions

2.19.1 Cursor

SPPU - May 13, Dec. 13

University Questions

Q. What is cursor? Explain various types of Cursor? **(May 2013, 8 Marks)**

Q. What is cursor? Explain explicit cursor in PL/SQL with suitable example? **(Dec. 2013, 6 Marks)**

- Cursor is used to traverse in the database to access the records one by one. For example if we want to calculate the average of marks of all the students, then we can retrieve marks of every student and add in the total_marks. Cursor is just like loop concept used to traverse to every row and manipulate data.
- An area of memory(Context) is allocated for the processing of SQL statements. The context area contains information necessary to complete the processing, including the number of rows processed by the statement, a pointer to the parsed representation of the statement.
- Cursor is a handle or pointer to the context area.
- There are two types of cursors
  1. Implicit Cursor
  2. Explicit cursor

2.19.1.1 Implicit Cursor

- When there is no explicit cursor, the implicit cursors are created automatically whenever an SQL statement is executed. Programmers cannot control the implicit cursors and the information in it.
- When Data Manipulation statements like insert, update or delete are executed, an implicit cursor is automatically associated with these statements. In case of insert statement the data is hold by cursor while for delete and update statements, cursor identifies the rows that would be affected.
- Consider following example in which increment of 10% is given to all the employees. Here an implicit cursor is created to identify the set of rows in the table which would be affected by the update.

```
UPDATE employee
  SET salary = salary * 0.1;
```

2.19.1.2 Explicit Cursor

An explicit cursor is the one in which the cursor name is explicitly assigned to the select statement. Processing an explicit cursor involves following three steps.

(1) **Open** : The Open statement executes the select statement. Positions the cursor at the first row.

(2) **Fetch** : The Fetch statement retrieves the current row and advances the cursor to the next row for processing.

(3) **Close** : After processing of last row the cursor is disabled with the help of Close statement.

Example : Consider the table city

```
mysql> select *from city;
+-----+-----------+----------+
| id  | city_name | state_id |
+-----+-----------+----------+
| 101 | Pune      |        1 |
| 102 | Mumbai    |        1 |
| 103 | Nasik     |        1 |
+-----+-----------+----------+
```

```
DELIMITER //
CREATE PROCEDURE cname(
  IN in_state_id INT
)
BEGIN
  DECLARE record_not_found INTEGER DEFAULT 0;
  DECLARE mycity_name VARCHAR(255)
DEFAULT "";
  DECLARE mysql_cursor CURSOR FOR
     SELECT city_name FROM city
     WHERE state_id = in_state_id;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET
record_not_found = 1;
  OPEN mysql_cursor;
   cities_loop: LOOP
   FETCH mysql_cursor INTO mycity_name;
    IF record_not_found THEN
      LEAVE cities_loop;
    END IF;
   SELECT mycity_name;
  END LOOP cities_loop;
  CLOSE mysql_cursor;
END //
DELIMITER ;
```

To call the procedure give query

```
CALL cname(1);
1 is the state id.
```

2.19.2 Trigger

SPPU - May 13

University Question

Q. Explain Triggers. **(May 2013, 4 Marks)**

A trigger is a set of actions which get executed automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a particular table.

Purpose of trigger

- To generate data automatically
- Validate input data
- Replicate data to different files to achieve data consistency
- Write to other files for audit trail purposes

Syntax

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
trigger_body
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
```

Consider the table employee

```
c:\wamp\bin\mysql\mysql5.5.24\bin\mysql.exe
+-----+---------+------+
| eno | ename   | sal  |
+-----+---------+------+
|   1 | Rajesh  | 9000 |
|   2 | Sudhir  | 8000 |
|   3 | Radhika | 7000 |
+-----+---------+------+
3 rows in set (0.00 sec)
```

Example

```
delimiter //
CREATE TRIGGER upd_check BEFORE UPDATE ON
employee
FOR EACH ROW
BEGIN
  IF NEW.sal < 5000 THEN
  SET NEW.sal = 5000;
  ELSEIF NEW.sal > 20000 THEN
  SET NEW.sal = 20000;
  END IF;
END;//
```

Now using update query if salary is updated to less than 5000 then it will be automatically set 5000 and if it is updated to more than 20000 then it will be automatically set 20000.

Syllabus Topic : Assertion

2.19.3 Assertion

Since SQL-92, assertions have been part of the SQL standard. An assertion is an expression used to define the constraint on database that must be always true.

Difference between Assertions and check constraints is :

Unlike check constraints they are not defined on table or column level instead of that they are defined on schema level. Assertions are not defined within create table or alter table statements like check constraint. To implement some constraint at the database level such as **cross-row constraints, multi-table check constraints,** SQL assertions can be used.

To declare Cross-row constraint triggers can be used but it is error-prone, so better way use assertions on database to serve the same purpose. If an assertion is declared on some tables, then all the constraints specified by assertion are followed by the database transaction which leads to modification on those tables.

Syntax for creating assertion is as follows :

```
CREATE ASSERTION Assertion_Name CHECK
(Condition);
```

The condition is in the form of an SQL query.

Example : Bank application has many tables in their database to store data. Schema of two of them are given below :

```
Account(acc_no, branch_name, balance);
Depositor(customer_name, acc_no);
```

Create an assertion which allows customers to have maximum two accounts in a bank application.

```
CREATE ASSERSTION Num_of_Acconts CHECK
(
(SELECT COUNT(*) FROM Account, Depositor
WHERE Account.acc_no=Depositer.acc_no) =< 2
);
```

2.19.4 Assertion Vs Triggers

Sr. No.	Assertion	Triggers
1.	Assertion is used to specify restrictions on database, it doesn't used to modify the data.	Triggers are used to give restrictions as well as to modify the data.
2.	All assertions can be implemented as triggers.	All triggers can't be implemented as assertions.
3.	Assertions are not linked to specific tables in the database and also not linked to specific events.	Triggers are linked to specific tables and specific events.

Syllabus Topic : Roles and Privileges

2.20 Roles and Privileges

2.20.1 Roles

- This is a group of privileges that will be assigned to users : Creating a Role
- CREATE ROLE 'admin'; You can also create more than one role at once
- CREATE ROLE 'dba', 'developer', 'readonly';

2.20.2 Privileges

- **Privileges** defines the access rights to database users on database objects (like functions, procedures, or tables). They also define rights to run a SQL statement, or PL/SQL Package.

Creating New User

- There are different ways to create users with custom permissions.

  Creating new user within the MySQL shell:

  CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';

- In this situation the newly created user has no permissions to do anything with the databases. Even if the new user try to login with the given password, he will not be able to reach the MySQL shell.

- Hence the most important initial step is to provide the user with access to the information they will need.

  GRANT ALL PRIVILEGES ON *. * TO 'newuser'@'localhost';

- The * symbol indicates all the databases and tables. Means user get rights like read, edit, execute and perform all tasks on the database.

Grant Different User Permissions

The following list shows other common possible permissions that users can get.

- ALL PRIVILEGES - allows a MySQL user all access to a designated database.
- CREATE - allows user to create new tables or databases
- DROP - allows user to them to delete tables or databases.
- DELETE - allows user to delete rows from tables.
- INSERT - allows user to insert rows into tables.
- SELECT - allows user to use the Select command to read through databases.
- UPDATE - allow user to update table rows.
- GRANT OPTION - allows user to grant privileges.
- REVOKE - removes granted privileges.

Granting privileges

```
GRANT [type of permission] ON [database name].[table name] TO '[username]'@'localhost';
```

REVOKE privileges

```
REVOKE [type of permission] ON [database name].[table name] FROM '[username]'@'localhost';
```

Deleting user

```
DROP USER 'demo'@'localhost';
```

Syllabus Topic : Embedded SQL

2.21 Embedded SQL

- The method of combining of general purpose programming languages and database language like SQL is called as **embedded SQL.**
- The SQL is good for defining database structure and defining short queries but for some application it is required to mix SQL with programming language.
- In other words, SQL defines WHAT is required but it can't define HOW to meet this requirement.
- SQL is used to express queries but all the queries can't be expressed with SQL alone, so there is a need of combining database language with general purpose programming language to express such queries.

Some concepts in Embedded SQL

- **Host language :** These are programming languages (such as C, C++, COBOL, Java, etc) in which SQL statements are embedded.
- **SQL Pre-Compiler :** A pre-compiler is used to process embedded SQL statements. It is used to translate SQL statements into DBMS library calls which can be implemented into host language.
- The following code is a simple embedded SQL program, written in C

The program accepts order number from user, retrieves the customer number, salesperson, and status of the order, and displays the data on the screen.

```
int main()
{
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
    int Order_ID;
    int Cust_ID;
    char Sales_Person[10]
    char Order_Status[6]
  EXEC SQL END DECLARE SECTION;

  /* Set up error processing */
  EXEC SQL WHENEVER SQLERROR GOTO qry_error;
  EXEC SQL WHENEVER NOT FOUND GOTO
invalid_number;

  /* Prompt the user for order number */
  printf ("Enter order number: ");
  scanf_s("%d", &Order_ID);

  /* Execute the SQL query */
  EXEC SQL SELECT Cust_ID, Sales_Person,
Order_status
    FROM Orders
    WHERE Order_ID = :Order_ID
    INTO :Cust_ID, :Sales_Person, :Order_Status;
```

```
  /* Display the results */
  printf ("Customer number:  %d\n", Cust_ID);
  printf ("Salesperson: %s\n", Sales_Person);
  printf ("Status: %s\n", Order_Status);
  exit();

qry_error:
  printf ("SQL error: %ld\n", sqlca->sqlcode);
  exit();

invalid_number:
  printf ("Invalid order number.\n");
  exit();
}
```

- **Host Variables :** These variables are declared in section starting with BEGIN DECLARE SECTION and ending with END DECLARE SECTION keywords. These variables are prefix by (:) in an embedded SQL statement. The precompiler distinguish between host variables and database objects(like column and tables) using (:).
- **Data Types :** Usually the data types of DBMS and a host language are different. This affects the host variable because they are used by both by host language statements and embedded SQL statements. When there is no similar type in host language that corresponds to a DBMS data type, the DBMS automatically converts the data.
- **Error Handling :** Run times errors are reported by DBMS to the application program using SQL Communications Area, or SQLCA. In the above program the statement INCLUDE SQLCA includes the SQLCA structure in program. It is used when errors are processed by the programs which are returned by the DBMS.
- **WHENEVER...GOTO statement :** It tells the pre-compiler to generate error-handling code when an error occurs.

2.21.1 Advantages of Embedded SQL

1) In embedded SQL, SQL is used to handle the database and the host language handles the variables and other input and output functions.
2) In the development cycle tasks like parsing and optimizing are done which taked high overheads. It increases efficiency of CPU resources.
3) Portability is achieved. The application be precompiled on one machine and can be moved another machine for further processing.
4) There is no need that programmer should understand the complex structure of DBMS. He has create only the embedded SQL source program code.

Syllabus Topic : Dynamic SQL

2.22 Dynamic SQL

SPPU - Dec. 13

University Question

Q. Write a short note on Dynamic SQL. **(Dec. 2013, 6 Marks)**

Although static SQL works well in many situations, in some applications the data access cannot be determined in advance.

Dynamic SQL refers to SQL code that is generated in an application or from the system tables and then executed against the database to manipulate the data at run time. The SQL code is not stored in the application but rather it is collected depending upon the input given by user.

The dynamic SQL helps to develop powerful applications which allows us to create database objects and manipulate them based on the input provided by the user.

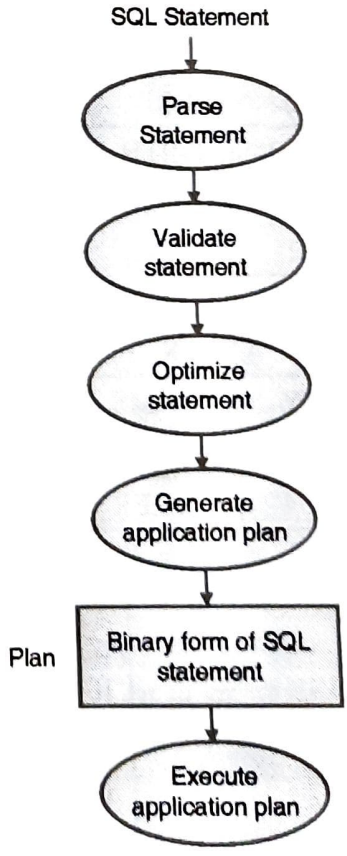


Fig. 2.22.1

In dynamic SQL the column_name or Table_name are not known at compile time so the DBMS can't able to prepare the SQL statement for execution in advance.

In this method, when the program is executed, the SQL statement get the table_name or column_name from user and this statement is given to the DBMS for further execution.

In the Fig. 2.22.1 we can observe the general execution process of dynamic sql.

An Example of Dynamic SQL statement

```
mysql> PREPARE stmt FROM
    -> 'select count(*)
    -> from information_schema.schemata
    -> where schema_name = ? or schema_name = ?'
;
Query OK, 0 rows affected (0.00 sec)
Statement prepared
mysql> EXECUTE stmt
    -> USING @schema1,@schema2
```

Output

```
+----------+
| count(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)

mysql> DEALLOCATE PREPARE stmt;
```

SQL *PLUS Assignments & Solved Queries

Table 2.22.1 : Emp

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT		11/17/1981	5000		10
7698	BLAKE	MANAGER	7839	05/01/1981	2850		30
7782	CLARK	MANAGER	7839	06/09/1981	2450		10
7566	JONES	MANAGER	7839	04/02/1981	2975		20
7788	SCOTT	ANALYST	7566	12/09/1982	3000		20
7902	FORD	ANALYST	7566	12/03/1981	3000		20
7369	SMITH	CLERK	7902	12/17/1980	800		20
7499	ALLEN	SALESMAN	7698	02/20/1981	1600	300	30
7521	WARD	SALESMAN	7698	02/22/1981	1250	500	30
7654	MARTIN	SALESMAN	7698	09/28/1981	1250	1400	30
7844	TURNER	SALESMAN	7698	09/08/1981	1500	0	30
7876	ADAMS	CLERK	7788	01/12/1983	1100		20
7900	JAMES	CLERK	7698	12/03/1981	950		30
7934	MILLER	CLERK	7782	01/23/1982	1300		10

Table 2.22.2 : Dept

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

1. Display the name and jobs of the employees who are clerk.

```
Select Ename, Job From Emp Where Job='Clerk';
```

2. Display the name and dept numbers of the employees whose job is an 'Analyst' or a 'Clerk'.

```
Select Deptno, Ename, Job From Emp Where Job In('ANALYST','CLERK');
```

3. Give the List of Employees sorted On Name.

```
Select * From Emp Order By Ename;
```

4. Display different kinds of jobs available.

```
Select Distinct Job From Emp;
```

5. Find the locations at which various Employees are situated.

```
Select Emp.Ename, Dept.Location From Emp, Dept Where Emp.Deptno= Dept.Deptno ;
```

6. Display the name, sal of the Employees whose dept locations is 'Chicago'.

```
Select Emp.Deptno, Ename, Sal From Emp Where Emp.Deptno= (Select Deptno From Dept Where
Location='Chicago') ;
```

7. Display the names of employees whose dept is 30.

```
Select Deptno, Ename From Emp Where Deptno= 30 ;
Select all Employees whose name is 5 letters long.
Select * From Emp, Where Length(Ename)=5 Order by Deptno;
Select all Employees whose name ends with R.
Select * From Emp, Where Ename like '%R';
```

8. Display the names of employees working in sales or research dept.

```
Select Ename, Deptno From Emp Where Deptno In (Select Deptno From Dept where Dname In ('SALES',
'RESEARCH'));
```

9. Display the dept no and dept name whose dept no > 20.

```
Select Deptno, Dname From Dept Where Deptno>20;
```

10. Find out dept in which maximum employees works.

```
Select Deptno, count(*) From Emp Group by Deptno Having Count(*)=(Select MAX(COUNT(*) From Emp Group
by Deptno);
```

11. Display the job, dept no, name of the employees whose name start with 'B' or 'M'.

```
Select Deptno, Ename, Job From Emp Where Ename Like 'B%' Or Ename like 'M%';
```

12. Find out the difference between maximum sal in dept 10 and minimum sal earn by a person in dept 30.

```
Select A.Sal-B.Sal From Emp A, Emp B Where A.Sal=(Select max(Sal) From Emp  where deptno=10) And
B.Sal=(Select min(Sal) From Emp  where deptno =30);
```

13. Find out the difference in dollers between the average earning of dept 20 and 30.

```
Select avg(A.Sal)-avg(B.Sal) From Emp A, Emp B group by A.Deptno,B.Deptno having avg(A.Sal)=(Select
avg(Sal) From Emp  where Deptno=20) And avg(B.Sal)=(Select avg(Sal) From Emp  where Deptno=30);
```

14. Display names, Sal and commission for Employees whose commission is more than 5% of sal.

```
Select Ename, Sal, Comm From Emp where Comm > Sal*0.05;
```

15. Find out the employees whose sal is < the average sal of dept 20.

```
Select * From Emp where Sal <(select avg(Sal) From Emp where Deptno=20);
```

16. Display information about people who is having the maximum no of people reporting to them.

```
Select * From Emp where Empno= (select MGR from Emp Group by MGR Having Count(MGR)=(select
Max(Count(MGR)) From Emp Group By MGR));
```

17. Give the query to concatenate Empno, name and manager from Emp table.

```
Select Empno||Ename||MGR From Emp;
```

18. List the employees who are hired earliest and latest.

```
Select * From Emp where HireDate=(Select Max(HireDate) From Emp) Or HireDate=(select min(HireDate) From
Emp);
```

19. List the names who are reporting to 'Blake'.

```
Select * From Emp where MGR=(Select Empno From Emp Where Ename ='Blake');
```

20. List all the employees hired in the month of December.

```
Select * From Emp where To_char(HireDate,'Month')='DEC';
```

21. List all the employees hired after 1980.

```
Select * From Emp where To_char(HireDate,'YYYY')>1980;
Pramote 'JAMES; to 'MANAGER' with increment of 1000
Update emp set job = manager and sal = sal + 1000 where ename = 'JONES';
```

22. Write sql command to find average annual salary per job in each department

```
Select deptno,avg(sal*12) from emp group by deptno;
```

23. Display department numbers of the departments who has more than one clerks.

```
Select deptno,count(*) from emp where job = 'CLERK' group by deptno having count(*)>1;
```

❑❑❑

CHAPTER 3

UNIT III

Relational Database Design

Syllabus

- Relational Model : Basic concepts, Attributes and Domains, Codd's Rules.
- Relational Integrity: Domain, Referential Integrities, Enterprise Constraints.
- Database Design: Features of Good Relational Designs.
- Normalization, Atomic Domains and First Normal Form, Decomposition using Functional Dependencies, Algorithms for Decomposition, 2NF, 3NF, BCNF.
- Modeling Temporal Data.

Syllabus Topic : Relational Model

3.1 Relational Model

3.1.1 Introduction

- The **Relational model** stores data in the form of tables. This concept is introduced by Dr. E.F. Codd, a researcher of IBM. The relational model is the first choice of commercial data processing applications for storing the data.
- Relational model is most famous because of its simplest structure as compare to other database models like network or hierarchical model.
- The relational model is now considered as the primary model for databases.
- The relational data model is the simple model having all the properties and capabilities required to process data.
- Most of the modern Database Management Systems (DBMS) are relational.
- The relational model consists of three major components :
  1. The set of relations and set of domains that defines the way data can be represented (data structure).
  2. Integrity rules that define the procedure to protect the data (data integrity).
  3. The operations that can be performed on data (data manipulation).
- In relational database model, the data is stored in the different tables. Theses tables are interlinked with each other with the help of common fields (columns) in between them.

History of relational model

- Edgar Codd introduced the relational model as general model of data.
- Codd proposed the relational model for IBM.
- But Codd has no idea that his influential work would become the basis of relational databases.
- The relational model is later maintained and developed by Chris Date and Hugh Darwen.
- Date and Darwen shows that the relational model contains some desired object oriented features in the third Mainfesto.

Definition of Relational Model

- A relational database is collectively combination of data structures, storage and retrieval operations and integrity constraints.
- In such a database the data and relations between them are organized into tables.
- Table is the collection of records.

3.1.2 Characteristics of Relational Database

The characteristics of Relational database systems are as follows :

- This model is called as Relational Model by Dr. Codd because the data is stored in the tables which are having relationships in between them.
- The whole data of the system is represented as systematic arrangement of data into rows and columns, called as relation or table.
- A table is also form in two-dimensional structure.
- At any given row/column position means in every cell in the relation there is one and only one value which is known as scalar value.
- Column represents attribute, and each column has a distinct name.
- All values entered in the columns are of the same data format.
- Implements the concept of closure means all operations are performed on an entire relation and result is an entire relation.
- It supports the operations like data definition, data manipulation and transaction management.

3.1.3 Advantages of Relational Model

1. **Ease of use :** The system which is managed in the form of tables consisting of rows and columns is much easier to understand.
2. **Flexibility :** The information from multiple tables can be retrieved easily at a time by joining the tables. Also changes can be done easily by using different operators.
3. **Security :** The different users can have different levels of access to data based on their roles. In the college database, students will have access to their own data only, while their teachers will have access to data of all the students to whom they are teaching. Class teacher will be able to see the reports of all the students in that class, but not other classes. The principal will have access to entire data.
4. **Data Independence :** In Relational system we can completely separate the data structure of database and programs or applications which are used to access the data. This is called as data independence. If any changes are made in structure of database then there is no need to make changes in the programs. For example you can modify the size or data type of a data items (fields of a database table) without making any change in application.
5. **Data Manipulation Language :** In the relational database approach it is easy to respond query by means of a language like SQL based on relational algebra and relational calculus. For data organized in other structure the query language either becomes complex or extremely limited in its capabilities.

Syllabus Topic : Basic Concepts, Attributes and Domains

3.1.4 Basic Concepts of Relational Model

Tables

- Relational model refers table as its basic unit.
- In relational model related data is collectively carry in the structured format in the database system.
- It is combination of rows and columns, where records are represented by rows and attributes are represented by columns. In relational data model, relations are saved in the format of Tables. The relations among entities is stored in this format.

Tuple

- Tuple is a single row of the table. It contains a whole record of a particular data item. For example in Student table the entire information about a particular student like his roll number, name, marks etc. are included in the tuple.
- A tuple is an ordered set of attribute values.

Attribute

- It is column of a table. For example in a table Student, there may be different attributes like roll_number, stud_name, marks etc.
- Every attribute is attached with specific data type which decides that which type of values can be inserted in those attributes.
- Attributes are also known as fields.

Domain

- Every attribute has some pre-defined value scope, known as attribute domain.
- This attribute domain decides which types of values are permitted in the attribute.
- The type of value contains where string or numerical or date type. The domain also decides some integrity constraints for values to be inserted

in the database. For example a emp_name attribute can have only string type of value while the salary attribute can have only numeric values with specified range.

- Domain restrict the data to be inserted with specific constraints, hence it is called as domain constraint.

Properties of relational database model

- Data is presented in the relational database model like it is the collection of relations.
- Each relation is considered as table.
- Columns are attributes that belong to the entity modelled by the table.
- Each row (Tuple) represents a single entity.
- Every table has a set of attributes collectively called as a "key" which uniquely identifies each row.

Syllabus Topic : Codd's Rules

3.2 Codd's Rules SPPU - May 14

University Question

Q. Explain in detail CODD's Rules ?
(May 2014, 8 Marks)

Codd designed these rules to define what is required from the relational database management system.

All these thirteen rules are followed by very few commercial products. Even the oracle follows eight and half rules out of thirteen.

Codd's rule are

Rule 0 : Foundation Rule

Foundation rule states that the system must be capable to manage their database systems through their relational capabilities. All other twelve rules are derived from this foundation rule.

Remaining twelve rule are as follows :

Rule 1 : Information Representation

- All the information in the database must be stored in standard form of tables.
- Table is considered as best format to store and manage the data.

Rule 2 : Systematic treatment of null values

- In a database the NULL values must be given a systematic and uniform treatment. In number of cases we may have to set null values on place of data. For Example, data is missing, data is not known, or data is not applicable.
- Null Value is different from empty character or string, string of a blank character, and it is also different from zero value or any number.

Example

EMPLOYEE			
EMP_ID	**NAME**	**DEPARTMENT**	**PH_NO**
E101	Naren	Testing	9656865232
E102	Atharv	NULL	9421589654
E103	Sarang	Testing	NULL

In relation PH_NO of Sarang is NULL.

Rule 3 : The guaranteed access rule

- In a data base every single data element must be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value).
- There should not be requirement of any other entity to access the data.

Rule 4 : Active online catalog

- The description of the structure of entire database must be stored in an online catalog, known as **data dictionary**.
- Data dictionary can be accessed by authorized users. Metadata should be maintained for all the data in the database.
- The system must support an online, inline, relational catalog that is accessible to authorized users by means of their regular query language.
- That is nothing but, just like ordinary data the database description is represented at the logical level, so that is possible for authorized users to apply the same language of regular data to its interrogation.

Rule 5 : The comprehensive data sub language rule

Database should not be directly accessible. It should always be accessible by using some strong query language. This rule illustrates that the system should support at least one relational language. The language -

a) has a linear syntax

b) can be used both interactively and within application programs,

c) supports data definition operations (including view definitions), data manipulation operations (update as well as retrieval), security and integrity constraints, and transaction management operations (begin, commit, and rollback).

Rule 6 : View updating rule

- Views are the virtual tables created using queries to show the partial or complete view of the table.
- All views that are theoretically updatable must also be updatable by the system
- The rule states that we should be able to make changes in views.

Rule 7 : High-level Insert, Update, and Delete Rule

- High Level means multiple rows from multiple columns are affected by the single query.
- This rule states that a database must support insertion, updation, and deletion.
- This must not be limited for a single row, that is, it must also support union, minus and intersection operations to yield sets of data records.

For example,

- Suppose employees got 5% hike in a year. Then their salary has to be updated to reflect the new salary. Since this is the annual hike given to the employees, this increment is applicable for all the employees.
- Hence, the query should not be written for updating the salary one by one for thousands of employee. A single query should be strong enough to update the entire employee's salary at a time.

Rule 8 : Physical data Independence

- Changes in the physical level (i.e. format of data storage or container of data like array or linked list) must not require any change to an application.

Rule 9 : Logical data Independence

- The user's view (application) should not be dependent upon logical data in a database.
- That means changes in logical data must not affect the applications which uses it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application.

Rule 10 : Integrity Independence

- The integrity constraints must be specified independently from application programs and stored in the catalog.
- We should be able to make changes in integrity constraints independently without the need of any change in the application.

Rule 11 : Distribution Independence

- The distribution of database at various locations should not be visible to end user.
- Users should always get the impression that the data is located at one site only.
- That means even though the data is distributed, it should not affect the speed of access and performance of data compared to centralized database.

Rule 12 : The non-subversion rule

- If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language.
- Means anyhow those integrity rules and constraints must be followed, we cannot violet them by using any back door option.

Syllabus Topic : Relational Integrity

3.3 Relational Integrity

SPPU - May 13, Dec. 13

University Question

Q. How Data Integrity Problem is handled with DBMS? **(May 2013, Dec. 2013, 2 Marks)**

- Mainly security and integrity of a database is the most important factors in judging the success of system.

- Integrity constraint is a mechanism to prevent invalid data entry into table to maintain the data consistency.
- Constraints are used to enforce limits to the range of data or type of data that can be inserted/updated/deleted from a table.
- The whole purpose of constraints is to maintain the **data integrity** during the various transactions like update/delete/insert on a table.
- There are different types of constraints

1. Domain Integrity Constraints
2. Entity Integrity Constraints
3. Referential Integrity Constraints
4. Enterprise Constraints

Syllabus Topic : Relational Integrity - Domain

3.3.1 Domain Integrity Constraints

- The domain constraints are considered as the most basic form of integrity constraints. To the attributes a domain of permitted values is associated.
- For attribute it defines the default value, the range value or specific value.
- The domain integrity constraints are easy to test when data is entered.
- The domain integrity constraints check that whether the attribute having proper and right value in the database or not.
- Domain integrity means it is the collection of valid set of values for an attribute.

Constraints

1. Not Null
2. Unique
3. Default
4. Check

Data Type

- A domain is the set of all unique values which are permitted for an attribute.
- Domain constraints are user defined data type. As we say that domain is the set of unique values, the column for which Domain Constraint has set, contains same type of data, based on its data type.
- The column does not accept values of any other data type.

Not allowed as it is integer attribute.

Emp_Id	Name	Salary
8001	Rahul	15000
8002	Seema	15000
A	Devika	14000

3.3.1.1 NOT NULL

- By setting the NOT NULL constraint we can assure that a column does not hold a NULL value.
- When for a specific column, no value is provided while inserting a record into a table, by default it takes NULL value.
- NULL constraint is applied to avoid insertion of any null value in the specific column.

Example

Consider table student having 'name' field with NOT NULL constraint.

Roll_No	Name
1	Rahul
2	Seema
3	

Not allowed. Because we set name as not null constraint.

3.3.1.2 UNIQUE

- UNIQUE Constraint as the name suggests, it can take only unique values in a column or set of columns. It keeps uniqueness of the table.
- When a column has a unique constraint then that particular column cannot have duplicate values in it.
- **Example :** We can set the UNIQUE Constraint for emp_id column in employee table, each employee should have different emp_id which means this column cannot have duplicate values.

Not allowed. Because Emp_id has unique constraint

Emp_id	Name	Salary
8001	Rahul	15000
8002	Seema	15000
8002	Devika	14000

3.3.1.3 DEFAULT

When a user does not provide a value to the column while inserting the records in the table, the DEFAULT constraint provides a default value to that column.

Sr. No.	Primary Key	Foreign Key
1.	Primary key uniquely identify a record in the table.	Foreign key is a field in the table that is primary key in another table.
2.	Primary Key can't accept null values.	Foreign key can accept null values.
3.	By default, Primary key is clustered index and data in the database table is physically organized in the sequence of clustered index.	Foreign key do not automatically create an index, clustered or non-clustered. You can manually create an index on foreign key.
4.	We can have only one Primary key in a table.	We can have more than one foreign key in a table.

On Delete Cascade

If a record in the parent table is deleted then the corresponding records in the child table will automatically deleted. This is called as on delete cascade.

Example

- Consider Training Institute Database having two tables Course_details and Student.
- There is a condition that the students can register for courses which are available in institute currently and not for the courses which are not offered at the moment. To specify this rule foreign key constrain is used.

Table 3.3.1 : COURSE_DETAILS (Master / Parent Table)

COURSE_CODE	COURSE_NAME	FEES
Or	Oracle	5000
Jv	Java	4000
Cp	C Programming	3000

Table 3.3.2 : STUDENT(Transaction / child Table)

STUD_ID	NAME	COURSE_CODE
S101	Kunal	Jv
S102	Radhika	Or
S013	Kiran	Cp

- In both the tables, the field COURSE_CODE is common. In Course details COURSE_CODE is referred as primary key and in STUDENT table it is referred as foreign key.
- Now if we try to delete any record from master table COURSE_DETAILS, then it will show error and force us to delete all corresponding records from child table student.
- But in case of **on delete cascade** rather than showing error, all the corresponding records from child table STUDENT will get automatically deleted when record from parent table COURSE_DETAILS is deleted.

Syllabus Topic : Relational Integrity - Enterprise Constraints

3.3.4 Enterprise Constraints

- Enterprise Constraints are also referred as Semantic constraints. They are additional rules specified by users or database administrators.
- These rules are depending upon the requirements and constraints of the business for which the database system is being maintained.
- For Example, In College System a class can have a maximum of 30 students. A teacher can teach a maximum of 4 classes a semester.
- In Corporate System an employee cannot take a part in more than 5 projects. Salary of an employee cannot exceed the salary of the employee's manager etc.
- Some other examples of business rules are :
  - A class can have a maximum of 35 students.
  - A course can be taught many times, but by only one instructor.
  - Not all teachers teach classes, etc.

Syllabus Topic : Database Design

3.4 Database Design

- Database design is very important aspect because properly designed database provides you with access to up-to-date, accurate information.
- Correct design is essential to achieve goals while working with a database. Investing the time required to learn the principles of good design makes sense.

- It is very important to design the database in such a manner that it should meet all our requirements and should be able to accommodate any change easily.
- Mainly a good, effective database design helps the development team to reduce the costs and time taken for the overall development.
- By creating a good data model and following the correct process, helps the development team to understand user requirements clearly and accurately.
- While designing any database we have to go step by step.
- The basic elements of the design process are :

1. Defining the problem or objective
2. Researching the current database
3. Designing the data structures
4. Constructing database relationships
5. Implementing rules and constraints
6. Creating database views and reports
7. Implementing the design

1. Defining the problem or objective

This is the first and important step of database design in which we will address the problem or objective of the database. Here the nature of data to be stored is decided.

2. Researching the current database

In some cases, there may be some database already in exist. Such database may be in any format like written on papers, spreadsheets, word files etc. The existing database is always useful for the end user. This existing database helps to determine the essential data structure of the database.

3. Designing the data structures

A database system is always a set of data tables, hence the next step in the design process is to identify and describe those data structures. The tables in the database represents some distinct subject or any physical object. By determining the subjects and objects in the system we can create list of tables to design. Then attributes of each table are decided.

4. Constructing database relationships

After creation of data structures, we have to establish relationships between the databases. It is necessary that each table should have a unique key which can identify the individual records in that table.

5. Implementing rules and constraints

Now as per the database and business requirements, some rules are set to restrict the data of insertion and modification so as to maintain data consistency. These rules refine the data and avoid any invalid data insertion or modification.

6. Creating database views and reports

Up till now the database design has completed, so we have to create views and reports to convert the data into useful information. This is usually for the end user. It helps to provide a simple structure of only required data to user by hiding the complexity and unnecessary data.

7. Implementing the design in software

This is the last and final step in which the create design is implemented through the software on the computer.

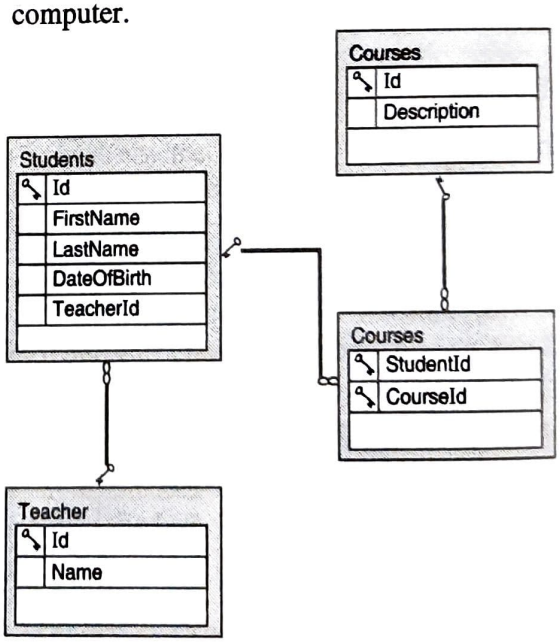


Fig. 3.4.1

Fig. 3.4.1 is simple example for database designing. Fig. 3.4.1 contains four tables students, courses, teachers, StudentCourses. It also shows the relationship between them. Here key sign shows the primary key.

Syllabus Topic : Features of Good Relational Design

3.4.1 Features of Good Relational Designs

- It should be able to distribute the information into different tables to reduce redundancy of data.
- It should provide easy access to the data available in multiple tables.
- It should take care of accuracy and integrity of information.
- It should provide functionalities for data processing and reporting needs.
- Data entry, updates and deletions should be efficient.
- Data retrieval, summarization and reporting should also be efficient.
- The database design must be self-documenting since much of the information is stored in the database rather than in the application.
- To understand the features of a good relational design we will see an example.

Example

- The relational design of Banking
  - branch : (brnch_name, brnch_city, assets) customer : (cust_id, cust_name, cust_street, cust_city)
  - loan : (loan_num, amt)
  - account : (acc_num, balance)
  - employee : (emp_id. emp_name, telephone_num, start_date)
  - dependent_name : (emp_id, dname)
  - account_branch : (acc_num, brnch_name)
  - loan_branch : (loan_num, brnch_name)
  - borrower : (cust_id, loan_num)
  - depositor : (cust_id, acc_num)
  - cust_banker : (cust_id, emp_id, type)
  - works_for : (worker_emp_id, manager_emp_id)
  - payment : (loan_num, payment_num, payment_date, payment_amt)
  - savings_account : (acc_num, interest_rate)
  - checking_account : (acc_num, overdraft_amount)
- Consider we want to get information from borrower and loan, then after combining these two relations -

  bor_loan = (cust_id, loan_num, amt)
- In the result, repetition of information is possible.
- Result is possible repetition of information. Observe **Ln101** in Fig. 3.4.2.

(A) Combine Schemas

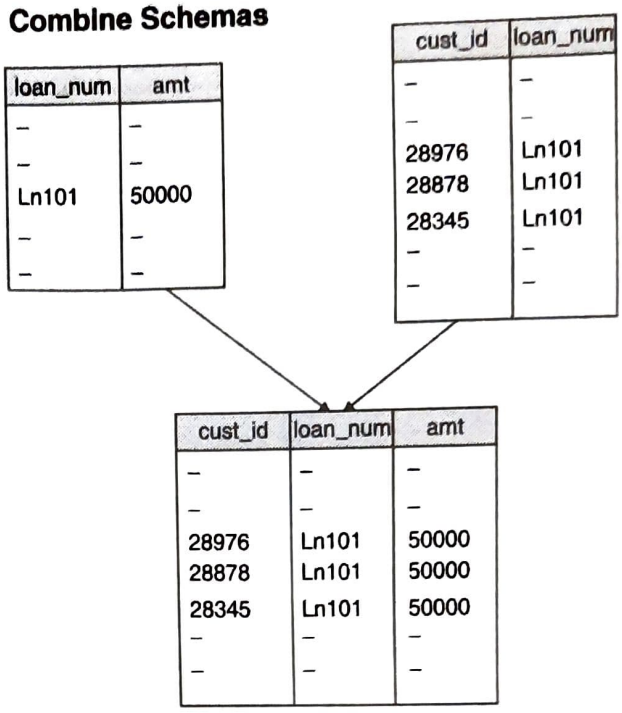


Fig. 3.4.2

1. Combining schema without repetition

Consider we want to get information from loan_branch and loan, then after combining these two relations -

loan_amt_br = (loan_num, amt, brnch_name)

Here no repetition of data will occur.

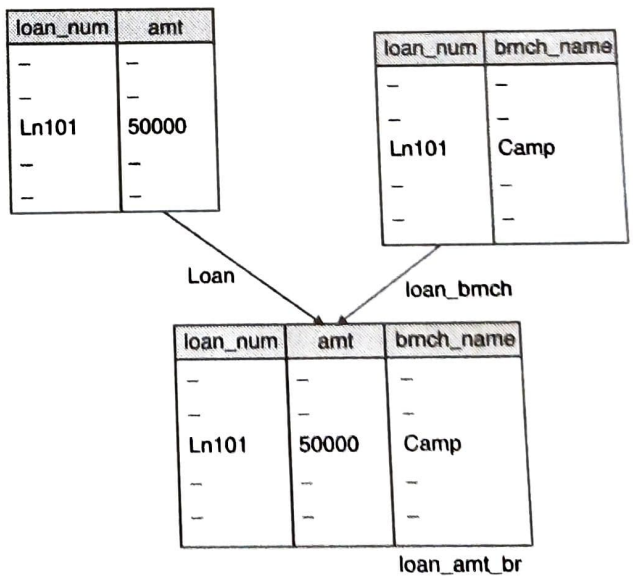


Fig. 3.4.3

B) Smaller Schemas

Emp_id	Emp_name	Telephone_num	Start_date
–	–	–	–
–	–	–	–
E-781	Rahul	788-6755	2014-03-04
E-792	Rahul	988-5677	2016-01-08
–	–	–	–

Emp_id	Emp_name
–	–
–	–
E-781	Rahul
E-792	Rahul
–	–

Emp_name	Telephone_num	Start_date
–	–	–
–	–	–
Rahul	788-6755	2014-03-04
Rahul	988-5677	2016-01-08
–	–	–

Emp_id	Emp_name	Telephone_num	Start_date
–	–	–	–
–	–	–	–
E-781	Rahul	788-6755	2014-03-04
E-781	Rahul	788-6755	2014-03-04
E-792	Rahul	988-5677	2016-01-08
E-792	Rahul	988-5677	2016-01-08
–	–	–	–

Fig. 3.4.4

- Consider that we have created bor_loan first. Now we want to split (decompose) it in borrower and loan.
- Decide a rule for a schema (loan_num, amt), loan_num would be a candidate key.
- Denote as a functional dependency : amount→loan_number
- In bor_loan relation the loan_num is not a candidate key hence the amount of a loan may have to be repeated. This indicates that we have to decompose bor_loan.
- It is not necessary that always all decompositions are good.
- Suppose we decompose employee into

  employee1 = (emp_id, emp_name)

  employee2 = (emp_name, telephone_num, start_date)

See the Fig. 3.4.4 which indicates how we lose information. We cannot reconstruct the original employee relation. This is called as lossy decomposition.

Syllabus Topic : Normalization

3.5 Normalization

SPPU - Dec. 14, Dec. 15, May 16

University Questions

Q. Define Normalization. Explain three normal forms with suitable example? **(Dec. 2014, 5 Marks)**

Q. Define Normalization. **(Dec. 2015, 2 Marks)**

Q. Define Database normalization. Explain any two normal forms with suitable example ? **(May 2016, 5 Marks)**

- Dr. Edgar F. Codd proposes normalization as an integral part of a relational model. **Normalization** is a database design technique which is used to organize the tables in such manner that it should reduce redundancy and dependency of data.
- **Normalization** is a database design technique which is used to organize the tables in such manner that it should reduce redundancy and dependency of data.

- It divides larger tables to smaller tables and links these smaller tables using their relationships.
- Normalization is implemented by following some formal rules either by a process of synthesis or decomposition. Synthesis is used to create database design in normalized form based on a known set of dependencies. Decomposition generally done on existing database which is not normalized. The relation of this database is further divided into multiple relations to improve the design.
- Normalization is the multi step process. It creates smaller tables from larger table.

Types of Normalization

1) **First normal form (1NF) :** Having unique values, no repeating groups.
2) **Second Normal form (2NF) :** Having unique values, no repeating groups, no partial dependency.
3) **Third Normal form (3NF) :** Same like second normal form and having transitive dependency.
4) **Boyce-Codd Normal form (BCNF) :** It is more developed version then 3NF.
5) **Fourth normal form (4NF) :** No multi-valued dependency.

3.5.1 Need of Normalization

- In database management process without Normalization, it is not easy to manage database operations like insertion, deletion, and modification without facing data loss problem.
- If database is not normalized, then Insertion, Updation and Deletion Anomalies occur frequently. To understand need of normalization, first we should learn, what are **Anomalies** ?

3.5.1.1 Anomalies

Anomalies are inconvenient or error-prone situation arising when we process the tables. There are three types of anomalies :

A) Insert Anomaly

- An **Insert Anomaly** occurs when it is not possible to insert certain attributes into the database without the availability of other attributes.
- For example, In College Database System, it is not possible to add entry of any new course unless any student enrolled for it.

Table 3.5.1 : STUDENT_INFO

STUD_ID	COURSE_ID	STU_NAME	ADDRESS	COURSE
S101	C1	Kunal	Camp	VB
S101	C2	Kunal	Camp	Java
S102	C3	Radhika	Sahkar Nagar	Oracle
S103	C4	Kiran	Tilak Road	C++
S104	C5	Jay	Pune Station	DS
S105	C3	Pooja	Sadashiv Peth	Oracle

- Here no student has enrolled for the course Dot Net, hence no entry for course Dot Net.

B) Update Anomaly

- An **Update Anomaly** occurs when there is requirement of changes in multiple records of an entity, but all records not get updated.
- For Example : Address of Kunal get changed.

Table 3.5.2 : STUDENT_INFO

STUD_ID	COURSE_ID	STU_NAME	ADDRESS	COURSE
S101	C1	Kunal	Tilak Road	VB
S101	C2	Kunal	Camp	Java
S102	C3	Radhika	Sahkar Nagar	Oracle
S103	C4	Kiran	Tilak Road	C++
S104	C5	Jay	Pune Station	DS
S105	C3	Pooja	Sadashiv Peth	Oracle

C) Delete Anomaly

- A **Delete Anomaly** is opposite to insert anomaly. This anomaly occurs when data of some attributes get lost because of deletion of data of other attributes in the same record.
- For Example : Just consider the only one entry for course Oracle of student Jay is deleted, then there will be no record related to course 'DS'.

STUD_ID	COURSE_ID	STU_NAME	ADDRESS	COURSE
S101	C1	Kunal	Tilak Road	VB
S101	C2	Kunal	Camp	Java
S102	C3	Radhika	Sahkar Nagar	Oracle
S103	C4	Kiran	Tilak Road	C++
S105	C3	Pooja	Sadashiv Peth	Oracle

- To overcome these anomalies we need to normalize the data in database.

Syllabus Topic : Atomic Domains and First Normal Form

3.6 First Normal Form (1NF)

- A **domain** is the set of all unique values which is permitted for an attribute. **Atomic** means that cannot be divided further.
- First Normal Form defines that all the attributes in a relation must have atomic (not further divisible) and single values.
- Consider the table Course _details

Table 3.6.1 : Course_details

Category_id	Category_name	Languages
C_PRG	Programming	C, VB, Java
C_SCR	Scripting	JavaScript, PHP, HTML

- Table 3.6.1 is **not in 1NF** as the rule says "each attribute of a table must have atomic (single) values", the attribute 'Languages' contain multiple values which violets the rule of 1NF. To convert this data into First Normal Form, we have to rearrange it in the table.

Category_id	Category_name	Languages
C_PRG	Programming	C
C_PRG	Programming	VB
C_PRG	Programming	Java
C_SCR	Scripting	JavaScript
C_SCR	Scripting	PHP
C_SCR	Scripting	HTML

- Now each attribute contain single values. Hence the database design is in First Normal Form.

Syllabus Topic : Decomposition using Functional Dependencies

3.7 Decomposition using Functional Dependency

3.7.1 Functional Dependency

The attributes of a relation is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table. This is called as **functional dependency**.

If attribute A of a relation uniquely identifies the attribute B of same relation then it can represented as $A \rightarrow B$ which means attribute B is functionally dependent on attribute A.

Formal definition of functional dependency

Let, R be a relation schema with attributes A, B i.e. R include A, B.

Given a relation R,

A set of attributes B in R is said to functionally dependent on another set of attributes A in R,

i.e. $A \rightarrow B$

Left-hand side of FD that is A is the determinant set of attributes and right-hand side of FD means B is the dependent or determined set of attributes. Usually primary key of relation is determinant in FD. Thus, given a row and the values of the attributes in set A, one can determine the corresponding value of the B attribute.

Table 3.7.1 : STUDENT

STUDENT_ID	STUDENT_NAME	ADDRESS	COURSE
1001	Akash	Nagpur	Oracle
1002	Anuja	Pune	Java
1003	Smita	Jalgaon	Php

From STUDENT_ID attribute it is possible to identify particular students' name. Name of the student is functionally dependent on STUDENT_ID and not vice versa; because more than two students may have same names and using STUDENT_ID we can determine the name of student but using STUDENT_NAME we can't determine the appropriate STUDENT_ID.

So we can say that,

STUDENT_ID $\rightarrow$ STUDENT_NAME

i.e. STUDENT_NAME is functionally dependent on STUDENT_ID

Similarly,

STUDENT_ID → all the attributes(i.e. ADDRESS, COURSE)

So we can say that all the remaining attributes are functionally dependent on STUDENT_ID attribute.

It is possible that an attribute of a relation can be identified by set of another attributes which are of the same relation. This can be implies as follows:

STUDENT_ID, STUDENT_NAME → ADDRESS

STUDENT_ID, STUDENT_NAME → COURSE

In above situation, usually composite primary key of relation is determinant in FD.

3.7.1.1 Types of Functional Dependencies

A) Single-valued Functional Dependency
B) Multi-valued Functional Dependency
C) Fully Functional Dependency
D) Partial Functional Dependency
E) Transitive Functional Dependency
F) Trivial Functional Dependency
G) Non Trivial Functional Dependency

A) Single-valued Functional Dependency

- Database is a collection of related information in which information depends on another information.
- The information is either single-valued or multi-valued. For example, the name of the person or his / her date of birth is single valued facts. But the Contact number of a person is a multi-valued attribute.
- A single valued functional dependency is when A is the primary key of an relation (e.g. STUDENT_ID) and B is some single valued attribute of the relation (e.g. STUDENT_NAME). Then, A → B must always hold by the relation.

STUDENT

STUDENT_ID	STUDENT_NAME	ADDRESS	DOB	COURSE
1001	Akash	Nagpur	11-2-1980	Oracle
1002	Anuja	Pune	2-10-1982	Java
1003	Anuja	Jalgaon	23-10-1981	PHP
1004	Amruta	Nanded	20-9-1981	DBMS

STUDENT_ID → STUDENT_NAME

1002 Anuja

For every STUDENT ID there should be unique name (A → B)

B) Multi-valued Functional Dependency

SPPU - May 14

University Question

Q. Write short note on : Multivalued Dependency. **(May 2014, 4 Marks)**

- A **multi-valued dependency** exists when a relation R has at least 3 attributes (like A, B and C) and for value of A there is a well defined set of values of B and a well defined set of values of C. However, the set of values of B is independent of set C and vice versa.

- For Example :

Table 3.7.2 : Published_Book

Book_Id	Auther_Name	Price
B_001	Jasmin	1000
B_002	Jasmin	980
B_003	Smita	1500
B_004	Smita	2000

- As shown in Table 3.7.2, name of particular author is determined by Book_Id attribute.

  {Book_Id} ->>{Author_Name}

- Also it is possible to determine the Price of book by Book_Id.

  {Book_Id} ->> {Price}

- But it is difficult to determine the price of book using author name or vice versa. So this dependency is called as multi-valued functional dependency or simply multi-valued dependency.

C) Fully Functional Dependency

Fully Functional Dependency occurs only in a relation with composite primary key. Fully functional dependency occurs when one or more non key attributes are depending on all parts of the composite primary key.

Table 3.7.3 : Stud_Courses

Stud_Id	Stud_Name	Contact_No	Course_Id	Course_Name	Credit_Hours	Grade
1001	Jasmin	9950043321	C_310	Database Management System	3	A
1001	Jasmin	9950043321	C_420	Operating System	3	B
1002	Smita	9800678120	C_310	Database Management System	3	B
1002	Smita	9800678120	C_420	Operating System	3	B
1002	Smita	9800678120	C_422	Computer NetWork	3	C
1003	Rahul	8878712356	C_412	Software Engineering	3	A

Here the composite primary key is Stud_Id + Course_Id

- Attribute Grade is fully functionally dependent on the primary key (Stud_Id, Course_Id) because both parts of the primary keys are needed to determine Grade.
- On the other hand both Stud_Name, and Contact_No attributes are not fully functionally dependent on the primary key, because only a part of the primary key namely Stud_Id is needed to determine both Stud_Name and Contact_No.
- Also attributes Credit-Hours and Course-Name are not fully functionally dependent on the primary key because only Course-Id is needed to determine their values.

D) Partial Functional Dependency

- **Partial Functional Dependency** occurs only in a relation with composite primary key. Partial functional dependency occurs when one or more non key attributes are depending on a part of the primary key.

- In partial functional dependency the determinant (Left-hand side of FD) consists of key attributes, but not the entire primary key and the determined (Right-hand side of FD) consist of non-key attributes.

For example,

Here the composite primary key is Stud_Id + Course_Id

Stud_Id	Stud_Name	Contact_No	Course_Id	Course_Name	Credit_Hours	Grade
1001	Jasmin	9950043321	C_310	Database Management System	3	A
1001	Jasmin	9950043321	C_420	Operating System	3	B
1002	Smita	9800678120	C_310	Database Management System	3	B
1002	Smita	9800678120	C_420	Operating System	3	B
1002	Smita	9800678120	C_422	Computer NetWork	3	C
1003	Rahul	8878712356	C_412	Software Engineering	3	A

- To determine name of the student or contact number of the student we can use only Stud_Id attribute which is part of the primary key.

{Stud_Id} → {Stud_Name, Conatct_No}

E) Transitive Functional Dependency

SPPU - May 13, Dec. 13, May 15

University Questions

Q. Describe the concept of transitive dependency. **(May 2013, Dec. 2013, 3 Marks)**

Q. Define Transitive dependency. **(May 2015, 2 Marks)**

- Functional dependency is said to be **transitive** if it is indirectly form by using two functional dependencies.
- For example in relation R (A, B, C), Functional dependency F includes:

A → B and B → C both the FDs are hold by relation, and if B → A doesn't hold

- Then we can say that A → C also holds by the relation.
- This type of dependency is called as **transitive dependency.**

Example

If Student_Courses relation holds following FDs:

{Course _Id} → {Student_Name}

{Student_Name} → {Student_Age}

Then we can say that the relation also holds following dependency.

{Course_Id} → {Student_Age}

F) Trivial Functional Dependency

- The dependency of an attribute on a set of attributes is known as trivial functional dependency if the set of attributes contains the attribute itself.
- In this dependency the Right-Hand Side of FD is a subset of Left-Hand Side of the FD. Functional dependency of the form A → B is trivial if B is a subset of A.
- The functional dependencies {A, B} → A is trivial.
- In a relation, if attribute A is depend on a set of attributes AB, and also the attribute A is subset of AB then this functional dependency is said to be trivial.

For example,

Stud_Courses

Stud_Id	Stud_Name
1001	Jasmin
1002	Smita
1003	Rahul

- {Stud_Id, Stud_Name} → {Stud_Id} is a trivial functional dependency as {Stud_Id} is subset of {Stud_Id, Stud_Name}. By knowing the values of id and name of the student it is possible to find values of student_id uniquely.
- The functional dependencies :

Stud_Id → Stud_Id and Stud_Name → Stude_Name

is also referred as trivial dependencies.

G) Non Trivial Functional Dependency

- It is inverse to trivial dependency. Functional Dependency is said to be **non trivial** if none of the attributes of Right-Hand Side of FDs are part of the Left-Hand Side attributes.
- Functional dependency of the form A → B is non-trivial if values in B are not present in A i.e. (B is not a subset of A).
- Non-Trivial Functional Dependency can be categorized into:

(i) Complete Non Trivial Functional Dependency

Functional Dependency is **completely non trivial** if none of the Right-Hand Side attributes of FD are part of the Left Hand Side attributes of the FD.

For example,

{Stud_ID} → {Stud_Name}

Values of Stud_Name attribute are totally different from values of Stud_Id in a relation.

(ii) Semi Non Trivial Functional Dependency

A Functional Dependency is **semi non trivial** if at least one of the Right Hand Side attribute of FD is not part of the Left Hand Side attributes of FD.

For example,

{Stud_ID, Student_Name} → {Stud_Name, Contact_No}

Values of Contact_No attribute are totally different from values of {Stud_Id, Stud_Name} in a relation.

3.7.1.2 Closure of Functional Dependency

1. The set of all functional dependencies logically implied by F are known as the closure of set F.
2. The closure of F is denoted by F^+.
3. To compute F^+, we can use some rules of inference like Armstrong's Axioms.

3.7.1.3 Inference Rules for Functional Dependencies

- The **inference rule** is an assertion which is used to derive new FDs from Previously defined basic FDs. A set of new functional dependencies is derived by applying assertions on previously defined basic functional dependency set.
- It is problematic to represent sets of all FDs for a relation initially.
- To make it easy, first define only basic FDs. Then apply set of inference rules on those FDs to derive new set of FDs.
- The most basic inference rule is Armstrongs's Axioms. There are other rules such as union, decomposition and pseudo transitivity which are derived from Axioms.

A) Armstrong's Axioms

SPPU - May 14

University Question

Q. State & prove Armstrong's Axioms rules for functional dependencies ?

(May 2014, 8 Marks)

Armstrong's axioms were developed by William W. Armstrong. These are a set of axioms (or inference rules) which are used to infer (conclude) all the FDs on a relational database.

Consider a relation R having three sets of attributes X, Y, Z. Then the Armstrong's axioms are :

- **Reflexivity rule :** If in a relation R values of Y attribute set are the subset of values in X attribute set, then FD X → Y can hold by the relation R. This type of dependency also called as trivial dependency.
- **Augmentation rule :** If FD X → Y is hold by relation R, then FD XZ → YZ also hold by relation.
- **Transitivity rule :** If Functional dependencies X → Y and Y → Z are hold by relation R then the FD X → Z also followed by relation R. This type of dependency also called as transitive dependency.

Properties of Armstrong's axioms

- The Armstrong's Axioms are **sound** :
  - When basic functional dependencies F on a relation R are given, then the FDs generated by using Armstrong's axioms will hold by R.
  - In other words applying Armstrong axioms on set of FDs which are previously defined will not generate invalid FDs.

- The Armstrong's Axioms are **complete** : When basic functional dependencies F on a relation R are given, then each and every valid Functional dependency on relation R can be found by applying only Armstrong axioms on set of FDs which are previously defined.

B) Union or Additivity

If in relation R having three sets of attributes X, Y, Z, and the set of functional dependencies : $X \rightarrow Y$ and $X \rightarrow Z$ are hold by relation R, then the functional dependency $X \rightarrow YZ$ also hold by relation R. This rule is called as **union**.

C) Decomposition or Projectivity

If in relation R having three sets of attributes X, Y, Z and the functional dependency $X \rightarrow YZ$ is hold by relation R then the following sets of Functional dependencies also hold by relation

R: $X \rightarrow Y$ and $X \rightarrow Z$

This inference rule is called as **decomposition rule**.

D) Pseudo transitivity

If in relation R having four sets of attributes W, X, Y, Z and the functional dependencies $X \rightarrow Y$ and $YW \rightarrow Z$ are hold by relation R, then the relation also hold following functional dependency $XW \rightarrow Z$. This inference rule known **pseudo transitivity**.

Syllabus Topic : Algorithm for Decomposition

3.7.2 Decomposition SPPU - Dec. 13, May 14

University Questions

Q. What is decomposition ? **(Dec. 2013, 2 Marks)**

Q. What do you mean by "Decomposition"? What are the desirable properties of it? How can we implement them ? **(May 2014, 8 Marks)**

One solution to eliminate the redundancy and also the update and deletion anomalies in database is breaking a relation into two or more relations. This process is called as **decomposition**.

Let, R be a relation schema.

A set of relation schemas {R1, R2,..., Rn} is a decomposition of R if

$$R = R1 \cup R2 \cup .....\cup Rn$$

For example,

Let us know how to break the relation into more than one relation.

Consider CLASSINFO relation having COURSE_ID, COURSE_NAME, STUD_NAME, SUB_ID, and SUB_NAME.

Table 3.7.4 : CLASSINFO

COURSE_ID	COURSE_NAME	STUD_NAME	SUB_ID	SUB_NAME
C_101	F. E.	Nishant	S_112	C
C_102	S. E.	Prashant	S_212	Java
C_102	S. E.	Prashant	S_222	DBMS
C_103	T. E	Sanvi	S_311	Ad. Java
C_101	F. E	Sanvi	S_115	CPP

This relation can be decomposed into two relations as follows :

i) COURSE_ STUDENT(COURSE_ID, COURSE_NAME, STUD_NAME)

ii) SUBJECT_ STUDENT(SUB_ID, SUB_NAME, STUD_NAME)

Table 3.7.5 : COURSE_STUDENT

COURSE_ID	COURSE_NAME	STUD_NAME
C_101	F. E.	Nishant
C_102	S. E.	Prashant
C_103	T. E	Sanvi
C_101	F. E	Sanvi

Table 3.7.7 : SUBJECT_STUDENT

SUB_ID	SUB_NAME	STUD_NAME
S_112	C	Nishant
S_212	Java	Prashant
S_222	DBMS	Prashant
S_311	VB	Sanvi
S_115	CPP	Sanvi

Now try to join these two relations COURSE_STUDENT and SUBJECT_STUDENT as follows :

Table 3.7.6 : COURSE_STUDENT ⋈ SUBJECT_STUDENT

COURSE_ID	COURSE_NAME	STUD_NAME	SUB_ID	SUB_NAME
C_101	F. E.	Nishant	S_112	C
C_102	S. E.	Prashant	S_212	Java

COURSE_ID	COURSE_NAME	STUD_NAME	SUB_ID	SUB_NAME
C_102	S. E.	Prashant	S_222	DBMS
C_103	T. E	Sanvi	S_311	VB
C_103	T. E	Sanvi	S_115	CPP
C_101	F. E	Sanvi	S_115	CPP
C_101	F. E	Sanvi	S_311	VB

Here two additional rows are displayed this states that the bad design of decomposition may leads to information loss. So, to ensure that the database has good design after decomposing; the decomposition process is done through some criteria which is defined by desirable properties of decomposition.

3.7.2.1 Desirable Properties of Decompositions

While decomposing the relation there must be some properties to hold so that the database remains in consistent state. There are several properties which are required to hold while decomposing the database schema.

There are two properties of decomposition :

i) Lossless join Decomposition
ii) Dependency Preservation Decomposition

i) Lossless decompositions

- When a relation is decomposed into several relations it is essential that the decomposition does not leads to any loss of the information.
- If there is the loss of the information, then it is called lossy decomposition or **lossy-join decomposition**. Decomposition is called as lossless decomposition if there is no loss of information in the decomposition of the relation into the desirable smaller relations.
- A decomposition {R1, R2,..., Rn} of a relation R is called a **lossless decomposition** for R if the natural join of R1, R2,..., Rn produces exactly the relation R.
- Decomposition is lossless if we can recover to original relation from combining smaller decomposed relations as follows :

  Thus, R` = R

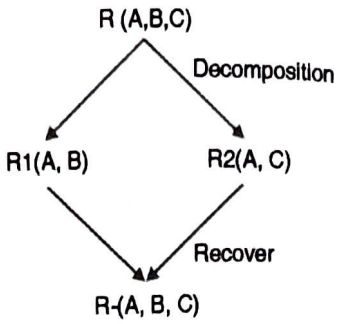


Fig. 3.7.1

Lossless decomposition property

- The lossless join decomposition is defined as, if R is a relation schema and F is a set of functional dependencies on R. R1 and R2 form decomposition of R. If at least one of following functional dependency is hold by F^+ then the decomposition is called as **lossless-join decomposition**.

F :

- $R1 \cap R2 \rightarrow R1$, that is all attributes common to both R1 and R2 functionally determine ALL the attributes in R1 **OR**

  $R1 \cap R2 \rightarrow R2$, that is all attributes common to both R1 and R2 functionally determine ALL the attributes in R2
- In other words, if $R1 \cap R2$ forms a superkey of either R1 or R2. The decomposition of R is a lossless decomposition.

ii) Dependency preservation decomposition

- To ensure that the functional dependencies hold by relation R should also preserved by the smaller relations (R1, R2, ..., Rn), the dependency preservation property of decomposition is used.
- A decomposition of the relation R into smaller relations R1, R2, ..., Rn is dependency preserving if all the Functional dependencies within R are followed in relations R1, R2, ..., Rn.

Dependency preservation decomposition property

- **Dependency preservation decomposition** is defined as, If R is a relation schema and F is a set of functional dependencies on R. R1, R2, ..., Rn form decomposition of R. If

$(F1 \cup F2 \cup \ldots \cup Fn)^+ = F^+$

Where,

F1, F2, ... Fn are the functional dependencies of relation R1, R2, ..., Rn.

F is a functional dependency of relation R

$(F1 \cup F2 \cup \ldots \cup Fn)^+$ is a closure of union of all sets of functional dependencies of R1, R2, ...Rn.

F^+ is a closure of set of functional dependency of relation R.

Syllabus Topic : 2NF

3.8 Second Normal Form (2NF)

SPPU - Dec. 15, May 16, Oct. 16

University Questions

Q. Explain 2nd Normal form with suitable example. **(Dec. 2015, 3 Marks)**

Q. Explain Second Form with suitable example. **(May 2016, Oct. 2016(In sem), 3 Marks)**

- To understand the second normal form fist we should get concepts of Prime and Non-prime attributes.
- **Prime attribute :** An attribute, which is a part of the prime-key is known as a prime attribute.
- **Non-prime attribute :** An attribute, which is not a part of the prime-key is said to be a non-prime attribute.
- As per the rule of **Second Normal Form**, every non-prime attribute must be dependent upon the prime attribute.
- If we follow second normal form, then every non-prime attribute should be fully functionally dependent on prime key attribute.
- Consider Table 3.8.1.

Table 3.8.1 : STUDENT_PROJECT

STUD_ID	PROJ_ID	STU_NAME	PROJ_NAME
S101	P1	Kunal	Banking System
S102	P2	Radhika	Library Management
S103	P3	Kiran	Speech to Text Converter
S104	P4	Jay	ATM

- In the above example the prime key attributes are STUD_ID and PROJ_ID. Here the non key attributes like STU_NAME and PROJ_NAME are dependent upon one of the prime key attributes.
- Means STUD_NAME is depend upon STUD_ID while the PROJ_NAME depends upon PROJ_ID. This is called as partial dependency.
- As per the rule of Second Normal Form the non key attributes should be dependent upon all the key attributes which is not followed.
- To convert the data in Second Normal Form we have to split the table in two different tables as follows.

STUDENT

STUD_ID	STU_NAME	PROJ_NAME
S101	Kunal	Banking System
S102	Radhika	Library Management
S103	Kiran	Speech to Text Converter
S104	Jay	ATM

- In the table STUDENT all the non prime attributes like STU_NAME and PROJ_NAME are completely depends upon the prime attribute STUD_ID

PROJECT

PROJ_ID	PROJ_NAME
P1	Banking System
P2	Library Management
P3	Speech to Text Converter
P4	ATM

- In the table PROJECT the non prime attributes PROJ_NAME is completely depends upon the prime attribute PROJ_ID

Syllabus Topic : 3NF

3.9 Third Normal Form (3NF)

SPPU - May 13, Dec. 13, May 15, May 16, Oct. 16

University Questions

Q. Explain how this concept is used to define 3 NF. **(May 2013, Dec. 2013, 5 Marks)**

Q. Explain third normal form with suitable example.
(May 2015, May 2016, Oct. 2016(In sem), 3 Marks)

A database design is said to be in **3NF** if both the following conditions are satisfied by it

- Initially the design must be in 2NF.
- No non-prime attribute should be transitively dependent on prime key attribute.
- For any **non-trivial** functional dependency, $X \rightarrow A$

  Either X is a superkey or A is prime attribute.
- Consider the Table 3.9.1.

Table 3.9.1 : STUDENT_DETAILS

STUD_ID	STU_NAME	CITY	ZIP
S101	Kunal	Pune	411037
S102	Radhika	Nasik	422001
S103	Kiran	Mumbai	400016
S104	Jay	Nagpur	440001

- In Table 3.9.1 the STUD_ID is the only one primary key. Here the data of city can be retrieved through either STUD_ID or ZIP code. But the CITY is not superkey as well as nor the CITY is prime attribute.
- Here the CITY is depends upon ZIP and ZIP is depends upon STUD_ID
- Stud_id → zip → city.
- This relationship is known as **transitive dependency**.
- We have to remove this transitive dependency by implementing Third Normal Form. For this purpose we have to split the STUDENT_DETAILS table in two different tables say STUDENT_DATA and CITY.

STUDENT_DATA

STUD_ID	STU_NAME	CITY
S101	Kunal	Banking System
S102	Radhika	Library Management
S103	Kiran	Speech to Text Converter
S104	Jay	ATM

CITY

ZIP	CITY
411037	Pune
422001	Nasik
400016	Mumbai
440001	Nagpur

- Now the database design is converted into Third Normal Form. Here the basically design is already in Second Normal Form and No non-prime attribute is transitively dependent on prime key attribute.

Syllabus Topic : BCNF

3.10 Boyce-Codd Normal Form (BCNF)

SPPU - May 14

University Question
Q. Define BCNF. (May 2014, 2 Marks)

A database design is said to be in BCNF if both the following conditions are satisfied by it

- Initially the table must be in 3NF.
- For any non-trivial functional dependency $X \rightarrow A$, X must be a super-key.

In above database design, Stud_id is the super-key in the relation Student_Data and Zip is the super-key in the relation City

Stu_ID → Stu_name, City

And Zip → City

Which confirms that both the relations are in BCNF.

3.10.1 Difference between 3NF and BCNF

SPPU - May 14

University Question
Q. Differentiate between BCNF & 3NF. How it is stronger than 3NF? (May 2014, 6 Marks)

Sr. No.	3NF	BCNF
1.	It stands for Third Normal Form.	It stands for Boyce-Codd Normal Form.
2.	A database design should be already in 2NF to convert in 3NF.	A database design should be already in 3NF to convert in BCNF.

Sr. No.	3NF	BCNF
3.	Rule : For any non-trivial functional dependency, X → A Either X is a superkey or A is prime attribute	Rule : For any non-trivial functional dependency, X → A X must be a super-key.
4.	3NF is weaker than BCNF.	BCNF is stronger than 3NF.
5.	3NF cannot catch all the anomalies.	BCNF was developed to capture those anomalies that could not be captured by 3NF.
6.	More redundancy.	Less redundancy.
7.	Computational time is more.	Computational time is less.

The above difference clearly indicates that **BCNF is stronger than 3NF.**

3.10.2 BCNF Decomposition Algorithm

Input : A relation R and functional dependencies F which are hold on R.

Output : A decomposition of R into a set of relations (R1, R2, R3,...., Rn), all of which are in BCNF.

1. If (R is in BCNF) then
   Return R.
2. Else If (there are BCNF violation, let's consider one FD as X→Y) then
3. Begin
4. Compute X^+
5. Decompose R into (R1 and R2) Select $R1=X^+$, and let R2 have attributes X and those attributes of R which are not in X^+
6. Compute the sets of FDs S1 and S2 for R1 and R2, respectively.
7. End
8. Repeat steps from 1 to 5 for R1 and R2 until all the relations (Ri) are in BCNF.
9. Return the union of the result of these compositions.

Example

As follows : If a relation R (A, B, C, D, E) and Functional dependency set F contains

```
F:=
{
   A→BC
   C-→DE
}
```

Step 1 : Compute closure of F

```
F+:=
{
   A+→ABCDE
   B+→B
   C+→CDE
   D+→D
   E+→E
}
```

So it concludes that the candidate key is A for relation R

Step 2 to 4 : check whether the FD is in BCNF if not decompose it.

Again consider the functional dependency set F :

First FD : A→BC, has candidate key A at the LHS of FD so it is in BCNF

Second FD : C→DE, doesn't contain LHS as candidate key so it violates the rule of BCNF. So let's decompose it to make the FD in BCNF, into R1 and R2 where R1 contains attributes contained in FD that is in BCNF and R2 contains attribute which are in FDs that violates BCNF rule.

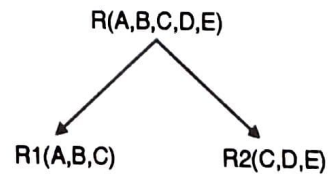


Fig. 3.10.1

Now R1 has set (ABC) which contain attribute A as a key according to the A^+ closure. And R2 has a set (CDE) which also contain attribute C as key According to the C^+ closure. At the end both decomposed relations (R1 and R2) are in BCNF form.

3.11 Fourth Normal Form

SPPU - May 13

University Question

Q. Explain why 4NF is more desirable than BCNF. Rewrite the definition of 4NF and BCNF using the notions of domain constrains and general constraints?

(May 2013, 8 Marks)

In the Fourth Normal Form,

- Initially the design must be in 3NF.
- It should not contain any multivalued dependencies.

Consider Table 3.11.1. It contains the data about Subject, Lecturer and Book_Name referred by the lecturers.

Table 3.11.1

Subject	Lecturer	Book_Name
English	Prof. Jitesh	English Book 1
English	Prof. Meetali	English Book 2
Science	Prof. Patil	Science Book 1
Maths	Prof. Vinay	Maths Book 1

- Now this design satisfies the rules of 3NF but two attributes Lecturer and Book_Name are independent entities.
- The subject English can teach by either Prof. Jitesh or Prof. Meetali. And also student have two choices for books of English. They can either refer English Book 1 or English Book 2.
- Hence the relationship will be

  SUBJECT → LECTURER

  SUBJECT → BOOK_NAME
- The relationship shows multi-valued dependency of attribute SUBJECT.
- If we need to select both lecturer and books recommended for any of the subject, then it will show the combination of lecturer, books, which implies lecturer who recommends which book. This is not correct.
- To remove this dependency the table should be split into two tables as below :

Lecturer

Subject	Lecturer
English	Prof. Jitesh
English	Prof. Meetali
Science	Prof. Patil
Maths	Prof. Vinay

Book

Subject	Book_Name
English	English Book 1
English	English Book 2
Science	Science Book 1
Maths	Maths Book 1

- Now in this design, if we want to retrieve names of lecturer or books recommended, then two independent queries can be executed. This eliminates the multi-valued dependency of attribute "Subject". Hence the design is in 4NF.

3.11.1 Difference between 4NF and BCNF

Sr. No.	4NF	BCNF
1.	It stands for Fourth Normal Form.	It stands for Boyce-Codd Normal Form.
2.	A database design should be already in 3NF and BCNF to convert in 4NF.	A database design should be already in 3NF to convert in BCNF.
3.	No multi-valued dependencies exist in the tables.	Multi-valued dependencies exist in the tables.
4.	The 4NF is more desirable than BCNF because it avoid repetition of data.	The BCNF is less desirable than BCNF because it does not avoid repetition of data.
5.	The decomposition in 4NF does not lead to loss of information when lossless join decomposition is used.	This is little bit difficult in BCNF.
6.	Redundancy is less.	Redundancy is more.

Syllabus Topic : Modeling Temporal Data

3.12 Modeling Temporal Data

- In conventional databases system, the time period is not attached to the data. The data is considered as valid for now. They do not keep track of past or future database states.
- **Temporal data** is the data to which time period is attached to it. This time period indicates that when that data is valid or stored in the database. By attaching a time period to the data, it becomes possible to store different states of the database.
- To implement the temporal database first we have to timestamp the data. This allows the distinction of different database states.
- One approach is that we can timestamp the entities with periods and another approach is that we can timestamp the property values of the entities.
- Tuples or rows are timestamped in the relational data model, while objects and/or attribute values may be timestamped.in object-oriented model.
- How and what time period are stored in databases? In general two different notions of time which are relevant for temporal databases are stored in database. First is called the valid time, the second is called as the transaction time.
- Valid time is nothing but the time period in which a fact is true with respect to the real world. Transaction time is the time period in which a fact is stored in the database.
- These two time periods do not have to be the same for a single fact. Consider that we are storing data about cricket matches played in 2016. Then the valid time of these facts is somewhere between Jan 01 2016 and Dec. 31 2016, where as the transaction time starts when we insert the facts into the database, for example, March 23 2017.
- Assume we would like to store data about our employees with respect to the real world. Then, the format of table could be:

Table 3.12.1 : EMP_DETAILS

Empid	Name	Department	Salary	Valid_time_start	Valid_time_end
101	Rohit	Production	11000	2013	2015
101	Rohit	Sales	13000	2015	2016
102	Vinay	Research	13000	2011	2015
103	Sunil	Accounts	10500	2014	INF

- Table 3.12.1 we have stored the history of the employees with respect to the real world.
- The above valid-time table stores the history of the employees with respect to the real world. The attributes VALIDTIMESTART and VALIDTIMEEND actually represent a time interval which is closed at its lower and open at its upper bound.
- Thus, we see that during the time period [2013 – 2015], employee Rohit was working in the Production department, having a salary of 11000. Then he changed to the sales department with salary raise to 13000.
- The upper bound INF indicates that the tuple is valid until further notice. Storing information about past states is possible now.
- We see that Vinay was employed from 2011 until 2015. In this case, in non-temporal table, this information was (physically) deleted when Vinay left the company.

3.12.1 Different Forms of Temporal Databases

- There are forms of temporal databases which depend upon the two different notions of time - valid time and transaction time.
- **Historical database** stores data with respect to valid time.
- **Rollback database** stores data with respect to transaction time.
- **Bitemporal database** stores data with respect to both valid time and transaction time.
- Usually in the commercial DBMS only a single state of the real world is stored, in general the most recent state. Such databases are known as **snapshot databases**.
- In the context of valid time and transaction time, the snapshot database is denoted in the Fig. 3.12.1.

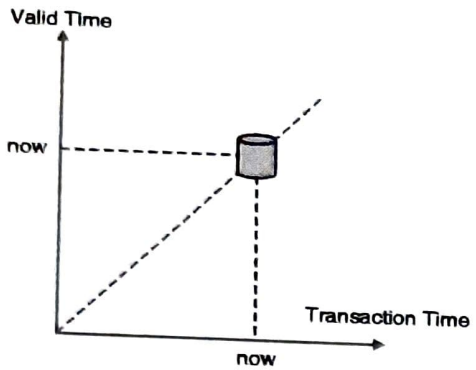


Fig. 3.12.1

- In case of bitemporal database the history of data related to both valid time and transaction time is stored in the database. TimeDB is an example of temporal database.
- In TimeDB DBMS the table may be in following formats :
  - Snapshot table - stores only current data
  - Valid-time table - stores when the data is valid with respect to the real world
  - Transaction-time table - stores when the data was recorded in the database
  - Bitemporal table - stores both valid time and transaction time
- In the SQL extended version it is also specified that which kind of table is needed when the table is created. It is also possible to make changes in existing table which is known as schema versioning.
- The temporal queries, temporal constraints and temporal modification statements are also supported by it.
- Fig. 3.12.2 specifies the different states of bitemporal database.

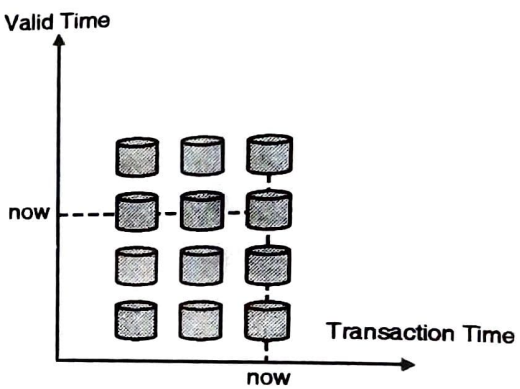


Fig. 3.12.2

❑❑❑