
15CS33T- DATABASE MANAGEMENT SYSTEMS

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

UNIT-1: Databases and Database Users

1.1 Introduction

1.2: Characteristics of the Database Approach

1.3: Actors on the Scene

1.4: Workers Behind the Scene

1.5: Advantages of DBMS Approach

1.6: A Brief History of Database Applications

1.7: When Not to Use a DBMS

Database System Concepts and Architecture

1.8 Data Models, Schemas, and Instances

1.9 Three-Schema Architecture and Data Independence

1.10 Database Languages and Interfaces

1.11 The Database System Environment

1.14 Centralized and Client /Server Architectures for DBMSs

1.15 Classification of database Management System.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

DATABASES AND DATABASE USERS

Databases and database systems are required in everyday life, in our daily routine. We interact with database one or the other way, for example we do bus or train reservation and funds transfer. Nowadays purchasing items from a supermarket involves an automatic update of the database that keeps the record of supermarket items.

1.1 Introduction

Data means known facts that can be recorded and have implicit meaning. The collection of related data with an implicit meaning is known as database.

For example, consider the personal details of Students maintained in an institution.

A database has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the miniworld or the universe of discourse (UoD). Changes to the miniworld are reflected in the database.
- a database is a logically coherent collection of data with some inherent meaning
- a database is designed, built, and populated with data for a specific purpose.

A database may be generated and maintained manually or it may be computerized.

For example,

- A library card catalog is a database that may be created and maintained manually.
- A computerized database may be created and maintained either by a group of application programs written specifically for that task or by a database management system.

A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database. The DBMS is a general-purpose software system that facilitates the processes of **defining, constructing, manipulating, and sharing databases** among various users and applications

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Defining: defining a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data**.

Constructing: constructing the database is the process of storing the data on some storage medium that is controlled by the DBMS.

Manipulating : manipulating a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.

Sharing: sharing a database allows multiple users and programs to access the database simultaneously.

DBMS provides important functions like **protecting** the database and **maintaining** it over a long period of time

- **Protection** includes system protection against hardware or software malfunction (or crashes) and security protection against unauthorized access.
- The DBMS must be able to **maintain** the database system by allowing the system to evolve as requirements change over time.

The database and DBMS software are together known as a **database system**.

Figure 1.1 illustrates a simplified database system environment.

An **application program** accesses the database by sending queries or requests for data to the DBMS.

A **query** causes some data to be retrieved; a transaction may cause some data to be read and some data to be written into the database.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

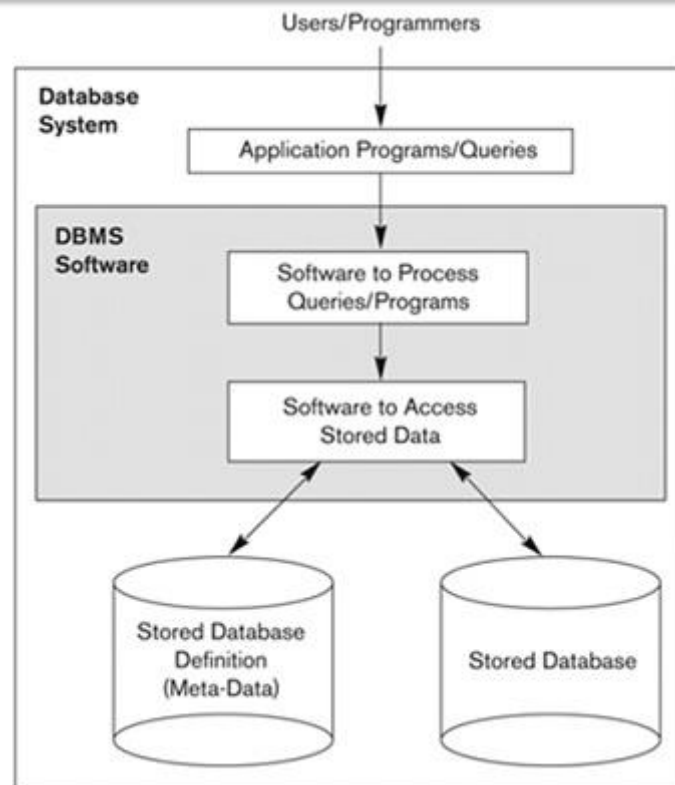


Figure 1.1 A simplified database system environment.

1.2 Characteristics of the Database Approach

In the database approach, a single repository maintains data that is defined once and then accessed by various users. The main characteristics of the database approach versus the file-processing approach are the following:

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing

1) Self-describing nature of a database system

In database approach, the database system contains not only the database but also a complete definition or description of the database structure and constraints.

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

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

The information stored in the catalog which describes the structure of the primary database is called **meta-data**.

2) Insulation between programs and data, and data abstraction

In file processing approach, any changes to the structure of a file may require changing all programs that access that file since the structure of data files is embedded in the application programs

In database approach it does not require such changes because structure of data files is stored in the DBMS catalog separately from the access programs. This property is called as **program-data independence**.

In some database systems user application programs can operate on the data by invoking the operations through their names and arguments, without knowing how the operations are implemented. This may be termed as **program-operation independence**.

The characteristic that allows program-data independence and program-operation independence is called **data abstraction**.

3) Support of multiple views of the data

A database has many users; each of them may require a different purpose or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored.

4) Sharing of data and multiuser transaction processing

A multiuser DBMS must allow multiple users to access the database at the same time. The DBMS must include **concurrency control software** to ensure that several users trying to update the same data in a controlled manner so that the result of the updates is correct.

For example, when several reservation agents try to reserve a seat on an airline flight, the DBMS should ensure that each seat can be accessed by only one agent at a time for reserving to a passenger. These types of applications are generally called **online transaction processing (OLTP) applications**.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Differences between file approach and database approach

| File Approach | Database Approach |
|---|--|
| Each user defines and implements the files needed for a specific software application | The database approach is a single repository maintains data that is defined once and then accessed by various users. |
| Redundancy or duplication of data Suppose both users are interested in students data, each user maintains separate files and separate programs to manipulate these files. | Controls redundancy A multiuser DBMS, allow multiple users to access the database at the same time |
| Data Dependence- Any changes to the structure of a file may require changing all programs that access that file. | Program-data independence- any changes to the structure of a data file does not require changing the programs that access that data file. |
| Provides Data inflexibility –program data dependency and data isolation limits the flexibility | Provides Data flexibility because of program data independence |
| Incompatible file formats- structure of the file is embedded in the application program, so structures are dependent on the application programming language | Compatibility in file formats – structure of the file stored in the catalog and all application programs access it. |
| Security of Data is low | Provides high Data security like backup and recovery |

1.3 Actors on the Scene

In many organizations different people are involved in the design, use, and maintenance of a database with users. The people who involve in the day-to-day use of a large database are called as the actors on the scene. Actors on the scene include Database Administrators, Database Designers, and End Users.

1.3.1 Database Administrators

A database administrator's responsibilities include the following tasks:

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- To oversee and manage the resources - In organization where many people use the same resources.
- In a database environment, the primary resource is the database itself. Administering these resources is the responsibility of the database administrator (DBA).
- The DBA is responsible for authorizing access to the database, coordinating and monitoring its use.
- The DBA is accountable for problems such as security breaches and poor system response time.
- Installing and upgrading the database server and application tools.
- Planning for backup and recovery of database information.
- Backing up and restoring databases.
- Generating various reports by querying from database as per need

1.3.2 Database Designers

Database designer's responsibilities include the following tasks:

- The database designers have to identify the data to be stored in the database and choose the appropriate structures to represent and store this data.
- Database designers are responsible to communicate with all database users in order to understand their requirements.
- Database designers are responsible to create a database design that meets user requirements.
- Database designers interact with each potential group of users and develop views of the database that meet the data and processing requirements of these groups. Each view is then analysed and integrated with the views of other user groups. The final database design must be capable of supporting the requirements of all user groups.

1.3.3 End Users

End users are the persons who involves in access to the database for querying, updating, and generating reports.

There are several categories of end users:

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

- Casual end users.
- Naive or parametric end users.
- Sophisticated end users.
- Sophisticated end users.

■ **Casual end users:** these end users occasionally access the database, but they may need different information each time.

Example: *middle- or high-level managers or other occasional browsers.*

■ **Naive or parametric end users:** these users constantly performs querying and updating the database, using standard types of queries and updates—called canned transactions—that have been carefully programmed and tested.

Example: a) **Bank tellers** check account balances and post withdrawals and deposits. b) **Reservation agents** for airlines, hotels, and car rental companies check availability for a given request and make reservations.

■ **Sophisticated end users:** these end users thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.

Example: *engineers, scientists, and business analysts*

■ **Standalone users:** these end users maintain personal databases by using ready-made program packages.

An example is *the user of a tax package that stores a variety of personal financial data for tax purposes.*

1.3.4 System Analysts and Application Programmers (Software Engineers)

System analysts determine the requirements of end users and develop specifications for canned transactions that meet these requirements.

Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

1.4 Workers behind the Scene

People who work to maintain the database system environment but who are not actively interested in the database contents as part of their daily job are known as workers behind the scene.

There are several categories of workers behind the scene as follows:

■ DBMS system designers and implementers:

These persons are responsible for Design and implement the DBMS modules and interfaces as a software package.

A DBMS is a very complex software system that consists of many components, or modules, controlling concurrency, and handling data recovery and security.

The DBMS must interface with other system software such as the operating system and compilers for various programming languages.

■ Tool developers:

These persons are responsible for Design and implement tool to the software packages for database system design, and improved performance.

Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, and etc.

■ Operators and maintenance personnel (system administration personnel):

These persons are responsible for the actual running and maintenance of the hardware and software environment for the database system.

1.5 Advantages of Using the DBMS Approach

- Controlling Redundancy
- Restricting Unauthorized Access
- Providing Persistent Storage for Program Objects
- Providing Storage Structures and Search Techniques for Efficient Query Processing
- Providing Backup and Recovery

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- Providing Multiple User Interfaces
- Representing Complex Relationships among Data
- Enforcing Integrity Constraints
- Permitting Inferencing and Actions Using Rules

1. Controlling Redundancy

In file processing, every user group maintains its own files for handling its data-processing applications.

For example: consider the COLLEGE here, two groups of user were each group independently keeps files on students. The **accounting office** keeps data on registration and related billing information, whereas the **registration office** keeps track of student courses and grades.

This redundancy in storing the same data multiple times leads to several problems.

First, there is the need to perform a single logical update multiple times, such as entering new student record. This leads to duplication of effort.

Second, storage space is wasted when the same data is stored repeatedly, and this problem may be serious for large databases.

Third, files that represent the same data may become inconsistent. This may happen because an update is applied to some of the files but not to others.

In database approach, a single repository maintains data that is defined once and then accessed by various users which controls redundancy.

2. Restricting Unauthorized Access

When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database.

For example, financial data is often considered confidential and only authorized persons are allowed to access such data. Hence, the type of access operation— retrieval or update—must also be controlled.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

A DBMS should provide a security and authorization subsystem, which the DBA uses to create accounts and to specify account restrictions. Then, the DBMS should enforce these restrictions automatically.

3. Providing Persistent Storage for Program Objects

The persistent storage of program objects and data structures is an important function of database systems. Object-oriented database systems typically offer data structure compatibility with one or more object-oriented programming languages.

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

Database systems must provide capabilities for efficiently executing queries and updates. Because the database is typically stored on disk, the DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records. Indexes are used for this purpose.

The query processing and optimization module of the DBMS is responsible for choosing an efficient query execution plan for each query.

4. Providing Backup and Recovery

A DBMS must provide facilities for recovering from hardware or software failures. The backup and recovery subsystem of the DBMS is responsible for recovery.

For example, if the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing.

5. Providing Multiple User Interfaces

Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces.

These include query languages for casual users, programming language interfaces for application programmers, forms and command codes for parametric users, and menu-driven interfaces and natural language interfaces for standalone users.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

6. Representing Complex Relationships among Data

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

7. Enforcing Integrity Constraints

Most database applications have certain integrity constraints that must hold for the data. A DBMS should provide capabilities for defining and enforcing these constraints.

The simplest type of integrity constraint involves specifying a data type for each data item.

8. Permitting Inference and Actions

Using Rules some database systems provide capabilities for defining deduction rules for inference new information from the stored database facts. Such systems are called deductive database systems.

A trigger is a form of a rule activated by updates to the table, which results in performing some additional operations to some other tables, sending messages, and so on.

More involved procedures to enforce rules are popularly called stored procedures.

Additional Implications of Using the Database Approach

This section discusses some additional implications of using the database approach that can benefit most organizations.

Potential for Enforcing Standards.

The database approach permits the DBA to define and enforce standards among database users in a large organization.

This provides communication and cooperation among various departments, projects, and users within the organization.

Standards can be defined for names and formats of data elements, display formats, report structures, terminology, and so on.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Reduced Application Development Time.

A prime feature of the database approach is that developing a new application takes very little time.

Designing and implementing a large multiuser database may take more time than writing a single specialized file application.

However, once a database is up and running, less time is required to create new applications.

Flexibility

It may be necessary to change the structure of a database as requirements change.

A DBMS allow certain types of changes to the structure of the database without affecting the stored data and the existing application programs.

Availability of Up-to-Date Information

A DBMS makes the database available to all users. As soon as one user's update is applied to the database, all other users can immediately see this update.

This availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases.

Economies of Scale

The DBMS approach permits consolidation of data and applications, thus reducing the amount of wasteful overlap between activities of data-processing personnel in different projects or departments as well as redundancies among applications.

This enables the whole organization to invest in more powerful processors, storage devices, or communication gear, rather than having each department purchase its own (lower performance) equipment. This reduces overall costs of operation and management.

1.6 A Brief History of Database Applications

- Early Database Applications:
 - The Hierarchical and Network Models were introduced in mid 1960s and dominated during the seventies.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- A bulk of the worldwide database processing still occurs using these models.
- Relational Model based Systems:
 - Relational model was originally introduced in 1970, was heavily researched and experimented with in IBM Research and several universities.

Object-oriented and emerging applications:

- Object-Oriented Database Management Systems (OODBMSs) were introduced in late 1980s and early 1990s to cater to the need of complex data processing in CAD and other applications.
- Their use has not taken off much.

Many relational DBMSs have incorporated object database concepts, leading to a new category called object-relational DBMSs (ORDBMSs)

Extended relational systems add further capabilities (e.g. for multimedia data, XML, and other data types)

- Relational DBMS Products emerged in the 1980s
- Data on the Web and E-commerce Applications:
 - Web contains data in HTML (Hypertext markup language) with links among pages.
 - This has given rise to a new set of applications and E-commerce is using new standards like XML (eXtended Markup Language).
 - Script programming languages such as PHP and JavaScript allow generation of dynamic Web pages that are partially generated from a database
- New functionality is being added to DBMSs in the following areas:
 - Scientific Applications
 - XML (eXtensible Markup Language)
 - Image Storage and Management
 - Audio and Video data management
 - Data Warehousing and Data Mining
 - Spatial data management
 - Time Series and Historical Data Management

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- The above gives rise to new research and development in incorporating new data types, complex data structures, new operations and storage and indexing schemes in database systems.
- Also allow database updates through Web pages

1.7 When not to use a DBMS

In spite of the advantages of using a DBMS, there are a few situations in which a DBMS may involve unnecessary overhead costs that would not be incurred in traditional file processing.

The overhead costs of using a DBMS are due to the following:

- High initial investment in hardware, software, and training
- The generality that a DBMS provides for defining and processing data
- Overhead for providing security, concurrency control, recovery, and integrity functions

Therefore, it may be more desirable to use regular files under the following circumstances:

- Simple, well-defined database applications that are not expected to change at all
- Stringent, real-time requirements for some application programs that may not be met because of DBMS overhead
- Embedded systems with limited storage capacity, where a general-purpose DBMS would not fit
- No multiple-user access to data

1.8 Data Models, Schemas, and Instances

A **data model** is a collection of concepts that can be used to describe the structure of a database. The structure of a database means the data types, relationships, and constraints that apply to the data.

1.8.1 Categories of Data Models

Based on the types of concepts they use to describe the database structure data models can be categorized as follows:

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- **High-level or conceptual data models:** Conceptual data models use concepts such as entities, attributes, and relationships.

An **entity** represents a real-world object or concept, such as an employee or a project from the mini world that is described in the database.

An **attribute** represents some property of interest that further describes an entity, such as the employee's name or salary.

A **relationship** among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project.

- **Low-level or physical data models** which provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks. An **access path** is a structure that makes the search for particular database records efficient
- **Representational (or implementation) data models**, which provide concepts that, may be easily understood by end users. Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.

1.8.2 Schemas, Instances, and Database State

In any data model, it is important to distinguish between the *description* of the database and the database itself.

The description of a database is called the **database schema**, which is specified during database design and is not expected to change frequently. A displayed schema is called a **schema diagram**. Figure 1.2 shows a schema diagram which displays the structure of each record type but not the actual instances of records. We call each object in the schema—such as STUDENT or COURSE as a **schema construct**.

Examples for Schema:

STUDENT

| | | | |
|-------|------|----------|-----------|
| REGNO | NAME | COURSENO | SECTIONID |
|-------|------|----------|-----------|

COURSE

| | | | |
|----------|------------|-------------|------------|
| COURSENO | COURSENAME | CREDITHOURS | DEPARTMENT |
|----------|------------|-------------|------------|

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

SECTION

| | | | | |
|-----------|----------|----------|------|---------|
| SECTIONID | COURSENO | SEMESTER | YEAR | FACULTY |
|-----------|----------|----------|------|---------|

GRADEREPORT

| | | |
|-------|----------|-------|
| REGNO | SECTIOID | GRADE |
|-------|----------|-------|

Fig 1.2 Schema diagram

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

Example:

STUDENT instance contains one row of data for the STUDENT schema.

| REGNO | NAME | COURSENO | SECTIONID |
|------------|--------|----------|-----------|
| 175cs13005 | Arpith | 15cs34T | A |

COURSE instance contains two rows of data for the COURSE schema.

| COURSENO | COURSENAME | CREDITHOURS | DEPARTMENT |
|----------|-----------------------|-------------|------------------|
| 15cs31T | Computer organisation | 52 | Computer science |
| 15cs34T | DBMS | 52 | Computer science |

The DBMS stores the descriptions of the schema constructs and constraints—also called the **meta-data**—in the DBMS catalog so that DBMS software can refer to the schema whenever it needs.

The schema is sometimes called the **intension**, and a database state is called an **extension** of the schema.

1.9 Three-Schema Architecture and Data Independence

The four important characteristics of the database approach as discussed earlier are

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- (1) Use of a catalog to store the database description (schema) so as to make it self-describing
- (2) Insulation of programs and data (program-data and program-operation independence)
- (3) Support of multiple user views.

The **three-schema architecture** was proposed to achieve and visualize these characteristics.

1.9.1 The Three-Schema Architecture

The goal of the three-schema architecture, illustrated in Figure 1.3 is to separate the user applications from the physical database. In this architecture, schemas can be defined at the following three levels:

1. The **internal level** has an **internal schema**, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The **conceptual level** has a **conceptual schema**, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. A representational data model is used to describe the conceptual schema when a database system is implemented.
3. The **external or view level** includes a number of **external schemas** or **user views**. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. An external schema is described in a high-level data model.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

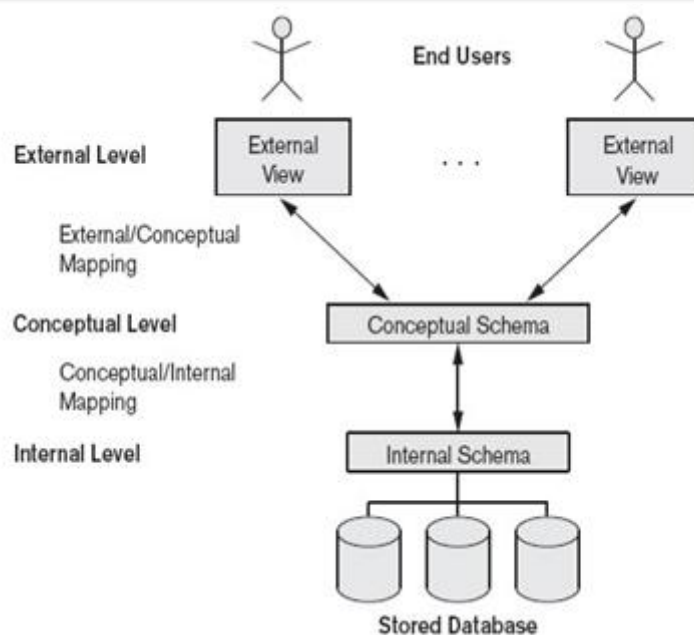


Figure 1.3: The three-schema architecture.

The stored data *actually* exists is at the physical level only. In a DBMS based on the three-schema architecture, each user group refers to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is database retrieval, the data extracted from the stored database must be reformatted to match the user's external view. The processes of transforming requests and results between levels are called **mappings**.

1.9.2 Data Independence

Data independence can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level. We can define two types of data independence:

- **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand, to change constraints, or to reduce the database.
- **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Changes to the internal schema may be needed

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

because some physical files were reorganized to improve the performance of retrieval or update.

1.10 Database Languages and Interfaces

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

1.10.1 DBMS Languages

Once the design of a database is completed and a DBMS is chosen to implement the database, the first step is to specify conceptual and internal schemas for the database and any mappings between the two. DBMS Languages includes the following:

- **Data definition language (DDL)** is used by the DBA and by database designers to define both schemas. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog. The DDL is used to specify the conceptual schema only.
- **Storage Definition Language (SDL)**, is used to specify the internal schema. This permit the DBA staff to control indexing choices and mapping of data to storage
- **View Definition Language (VDL)**, to specify user views and their mappings to the conceptual schema.
- **Data Manipulation Language (DML)**: Once the database schemas are compiled and the database is populated with data, users must have some means to manipulate the database. Typical manipulations include retrieval, insertion, deletion, and modification of the data. The DBMS provides a set of operations or a language called the **Data manipulation language (DML)**.

There are two main types of DMLs.

- **High-level** or **nonprocedural** DML
- **Lowlevel** or **procedural** DML

A **Low-level** or **Procedural** DML:

- This DML *must* be embedded in a general-purpose programming language.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- This type of DML typically retrieves individual records or objects from the database and processes each separately.
- Therefore, it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records.
- Low-level DMLs are also called **record-at-a-time** DMLs.

High-level or Nonprocedural DML

- DML allow high-level DML statements to be embedded in a general-purpose programming language.
- DML statements must be identified within the program so that they can be extracted by a precompiler and processed by the DBMS.
- Highlevel DMLs, such as SQL, can specify and retrieve many records in a single DML statement; therefore, they are called **set-at-a-time** or **set-oriented** DMLs

A query in a high-level DML often specifies *which* data to retrieve rather than *how* to retrieve it; therefore, such languages are also called **declarative**.

Whenever DML commands, whether high level or low level, are embedded in a general-purpose programming language, that language is called the **host language** and the DML is called the **data sublanguage**.

A high-level DML used in a standalone interactive manner is called a **query language**.

1.10.2 DBMS Interfaces

The **user-friendly** interface is used for interacting with the database. User-friendly interfaces provided by a DBMS may include the following:

- Menu-Based Interfaces for Web Clients or Browsing
- Forms-Based Interfaces
- Graphical User Interfaces
- Natural Language Interfaces
- Speech Input and Output

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- Interfaces for Parametric Users
- Interfaces for DBA

- **Menu-Based Interfaces for Web Clients or Browsing**
 - ❖ These interfaces present the user with lists of options called **menus**.
 - ❖ Menus need not to memorize the specific commands and syntax of a query language; rather, the query is composed step-by step by picking options from a menu that is displayed by the system. Pull-down menus are a very popular technique in **Web-based user interfaces**

- **Forms-Based Interfaces**
 - ❖ A forms-based interface displays a form to each user. Users can fill out all of the **form** entries to insert new data, or they can fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries.
 - ❖ Forms are usually designed and programmed for naive users as interfaces to canned transactions.

- **Graphical User Interfaces**
 - ❖ A GUI typically displays a schema to the user in diagrammatic form. The user then can specify a query by manipulating the diagram. GUIs utilize both menus and forms.
 - ❖ Most GUIs use a **pointing device**, such as a mouse, to select certain parts of the displayed schema diagram.

- **Natural Language Interfaces**
 - ❖ These interfaces accept requests written in English or some other language and attempt to *understand* them.
 - ❖ A natural language interface usually has its own *schema*, which is similar to the database conceptual schema, as well as a dictionary of important words. The natural language interface refers to the words in its schema, as well as to the set of standard words in its dictionary, to interpret the request.

- **Speech Input and Output**

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- ❖ Applications such as inquiries for telephone directory, flight arrival/departure, and credit card account information are allowing speech for input and output to enable customers to access this information.
 - ❖ The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech takes place.
- **Interfaces for Parametric Users**
- ❖ Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. For example, a teller is able to use single function keys to invoke routine and repetitive transactions such as account deposits or withdrawals, or balance inquiries.
 - ❖ Systems analysts and programmers design and implement a special interface for each known class of naive users.. This allows the parametric user to proceed with a minimal number of keystrokes.
- **Interfaces for the DBA**
- ❖ Most database systems contain privileged commands that can be used only by the DBA staff.
 - ❖ These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

1.11 The DBMS System Environment

1.11.1 DBMS Components Modules

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

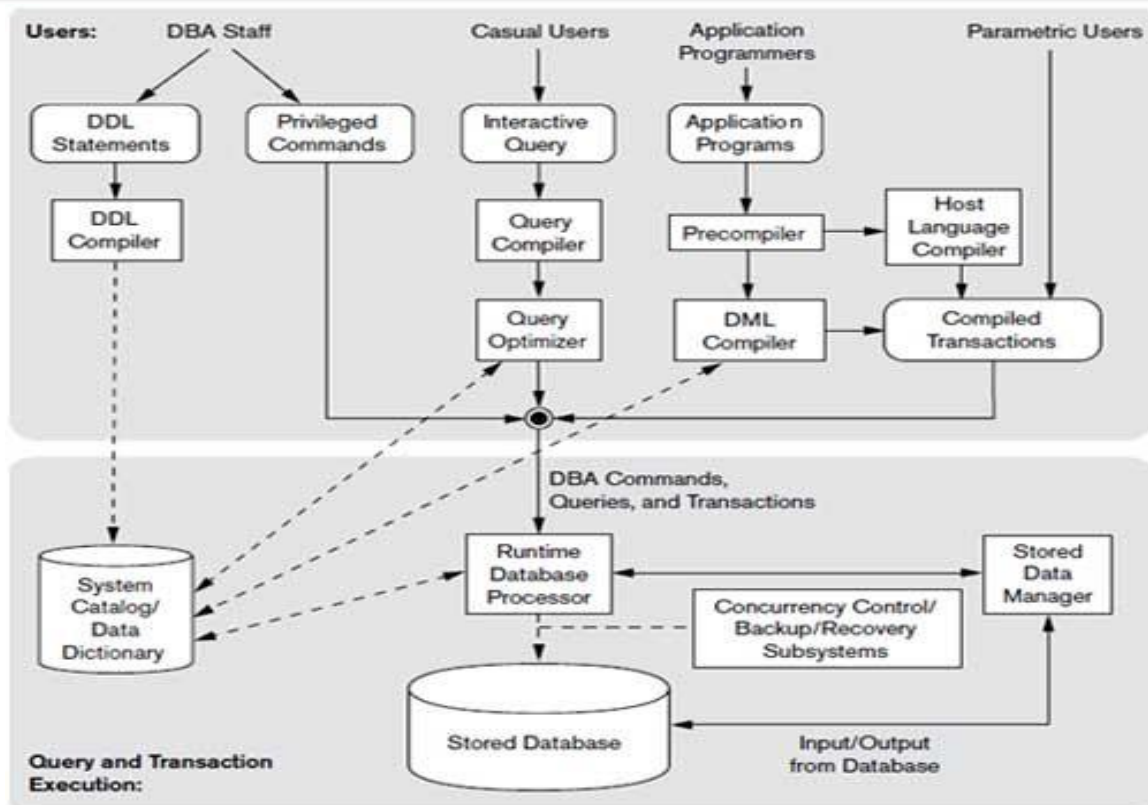


Figure 1.4 DBMS Components and their interactions

Figure 1.4 illustrates the typical DBMS components in a simplified form. The figure is divided into two parts.

- The top part of the figure refers to the various users of the database environment and their interfaces.
- The lower part shows the internals of the DBMS responsible for storage of data and processing of transactions.

DBMS Components modules are:

- Stored data manager
- DDL compiler
- Interactive query interface
 - Query compiler
 - Query optimizer
- Precompiler

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- Runtime database processor
- System catalog
- Concurrency control system
- Backup and recovery system

Stored data manager:

- Storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible for efficient storing, retrieving and updating of data.
- The dotted lines and circle shows access that are under the control of this stored data manager.

DDL Compiler:

- The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.

Interactive Query Interface:

- This interface may be used to generate the interactive query automatically. These queries are parsed and validated for correctness of the query syntax by a **query compiler** that compiles them into an internal form.
- This internal query is subjected to query optimization. The **query optimizer** is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of correct algorithms and indexes during execution.

Pre-compiler:

- The **pre-compiler** extracts DML commands from an application program written in a host programming language.
- These commands are sent to the DML compiler for compilation into object code for database access. The rest of the program is sent to the host language compiler.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor

Runtime database processor:

- Runtime Database Processor executes
- ✓ The privileged commands,
- ✓ The executable query plans, and
- ✓ The canned transactions with runtime parameters.

Concurrency Control System:

- **Concurrency control (CC)** is a process to ensure that data is updated correctly and appropriately when multiple transactions are concurrently executed in **DBMS**

Backup and Recovery System.

- A database backup operation is performed by backup system, when a problem that damages the database, all committed data is recovered by recovery subsystem

1.11.2 Database System Utilities

The DBMS have **database utilities** that help the DBA to manage the database system. Common utilities have the following types of functions:

- Loading
- Backup
- Database storage Reorganization
- Performance Monitoring

■ Loading

- A loading utility is used to load existing data files into the database.
- The current (source) format of the data file and the desired (target) database file structure are specified to the utility, which then automatically reformats the data and stores it in the database. Such tools are also called **conversion tools**

■ Backup

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- A backup utility creates a backup copy of the database, by dumping the entire database onto tape or other mass storage medium.
- The backup copy can be used to restore the database in case of catastrophic disk failure

■ Database storage reorganization

- This utility can be used to reorganize a set of database files into different file organizations, and create new access paths to improve performance.

■ Performance monitoring

- Monitors database usage and provide statistics to the DBA.
- The DBA uses the statistics in making decision such as whether or not to reorganize files or whether to add or drop indexes to improve performance.

1.11.3 Tools, Application Environments, and Communications Facilities

Tools

- CASE tools are used in the design phase of database systems.
- Another tool that can be useful in large organizations is an expanded **data dictionary system**.
- The data dictionary stores other information, such as design decisions, usage standards, application program descriptions, and user information. Such a system is also called an **information repository**.
- This information can be accessed *directly* by users or the DBA when needed.

Application development environments

- The systems provide an environment for developing database applications and include facilities that help in many facets of database systems, including database design, GUI development, querying and updating, and application program development.

Communications software

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- The DBMS also needs to interface with **communications software**, whose function is to allow users at locations remote from the database system site to access the database through computer terminals, workstations, or personal computers.
- These are connected to the database site through data communications hardware such as Internet Router.
- The integrated DBMS and data communications system is called a **DB/DC** system.

1.12 Centralized and Client/Server Architectures for DBMSs

1.12.1 Centralized DBMSs Architecture

Centralized DBMS combines everything into single system including- DBMS software, hardware, application programs and user interface processing software.

User can connect through a remote terminal – however, all processing is done at centralized site.

Figure 1.5 illustrates the physical components in a centralized architecture. Gradually, DBMS systems started to exploit the available processing power at the user side, which led to client/server DBMS architectures.

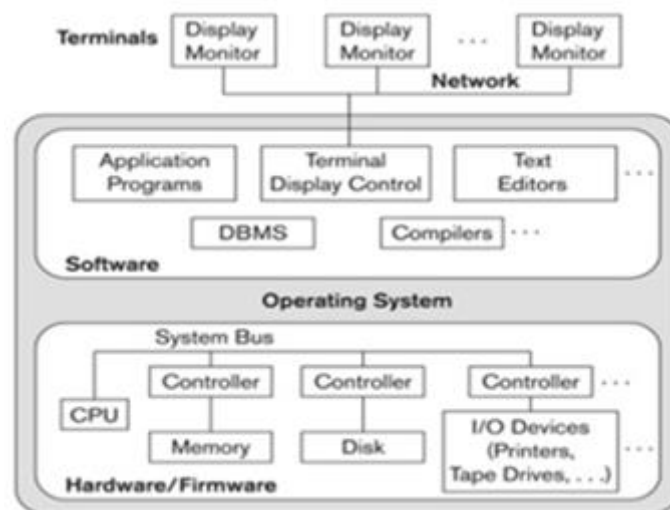


Figure 1.5 Physical components in a centralized architecture

1.11.2 Basic Client/Server Architectures

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

The **client/server architecture** was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, data base servers, Web servers, e-mail servers, and other software and equipment are connected via a network.

In Figure 1.6 illustrates client/server architecture at the logical level:

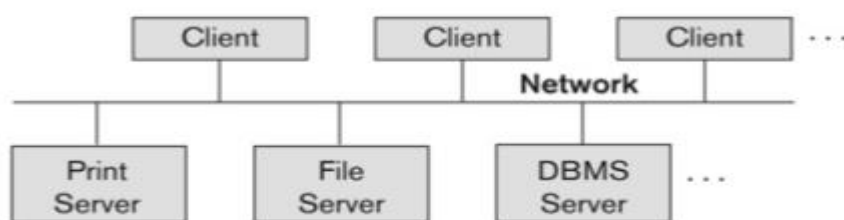


Figure 1.6 Logical two-tier architecture

The **client machines** provide the user with the appropriate interfaces to utilize these servers, as well as with local processing power to run local applications.

A **server** is a system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access.

The **file server** that maintains the files of the client machines, another machine can be designated as a **printer server** by being connected to various printers; all print requests by the clients are forwarded to this machine

Web servers or **e-mail servers** also fall into the specialized server category. The resources provided by specialized servers can be accessed by many client machines.

1.11.3 Two-Tier Client/Server Architectures for DBMSs.

In two tier architecture user interface programs and application programs can run on the client side. When DBMS access is required, the program establishes a connection to the DBMS server.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

A standard called **Open Database Connectivity (ODBC)** provides an **application programming interface (API)**.

Figure 1.7 is a simplified diagram that shows the physical Architecture. Some machines would be client sites only.

A client program can actually connect to several RDBMSs and send query and transaction requests using the ODBC API, which are then processed at the server sites. Any query results are sent back to the client program, which can process and display the results as needed.

A related standard for the Java programming language, called **JDBC**, has also been defined. This allows Java client programs to DBMS server through a standard interface.

The architectures described here are called **two-tier architectures** because the software Components are distributed over two systems: client and server.

The advantages of this architecture are its simplicity and seamless compatibility with existing systems.

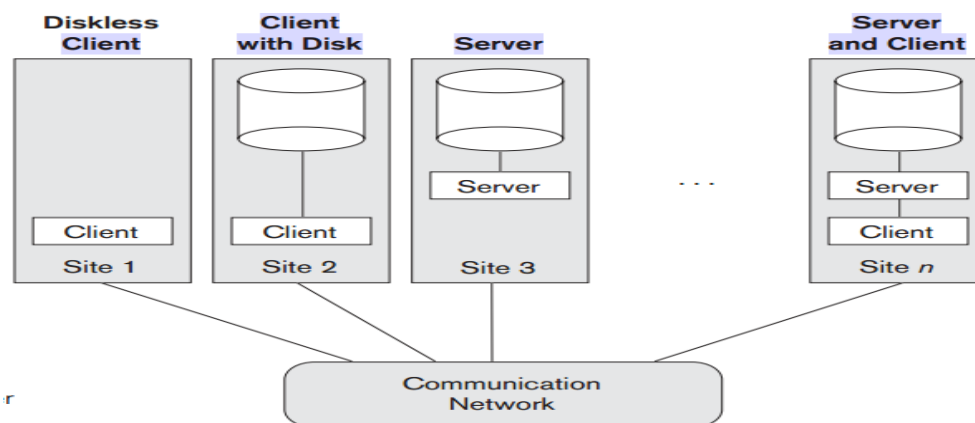


Figure 1.7 Physical two-tier client/server architecture.

1.11.4 Three-Tier and n-Tier Architectures for Web Applications

Many Web applications use an architecture called the **three-tier architecture**, which adds an intermediate layer between the client and the database server, as illustrated in Figure 1.8.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

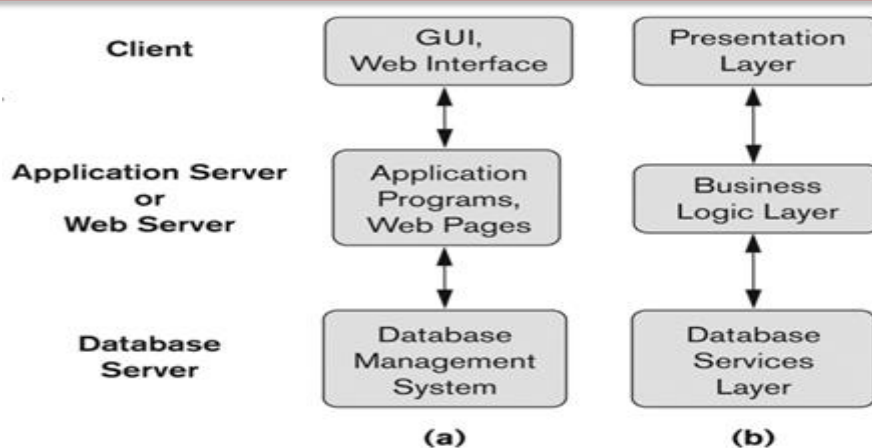


Figure 1.8 Logical three-tier architecture

This intermediate layer or **middle tier** is called the **application server** or the **Web server**, depending on the application. This server plays an intermediary role by running application programs and storing business rules that are used to access data from the database server. It can also improve database security by checking a client's credentials before forwarding a request to the database server.

Clients contain GUI interfaces and some additional application-specific business rules. The intermediate server accepts requests from the client, processes the request and sends database queries and commands to the database server, and then acts as a conduit for passing (partially) processed data from the database server to the clients, where it may be presented to users in GUI format. Thus, the *user interface*, *application rules*, and *data access* act as the three tiers

1.12 Classification of DBMS

The following criteria are normally used to classify DBMSs.

- **Data model**
 - **Number of users**
 - **Number of sites**
 - **Cost**
- ❖ Based on the *Data model* being used the DBMS is classified as follows:
- **Hierarchical data model**
 - Hierarchical data model represents data as hierarchical tree structure.

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

- There is no standard language for the hierarchical model; most hierarchical DBMS have record-at-time languages
- Legacy applications still run on database systems based on the **hierarchical** and **network data models**.
- **Relational data model**
 - Stores data in the form of a table.
 - A relation database is a data driven not design driven
 - Maintaining consistency among all applications is easier
 - It supports power query language SQL.
- **Object-Oriented model**
 - It is based on the collection of objects
 - This database manages objects for multimedia applications and manages data with complex relationships that are difficult to model and process in a RDBMS.
- **Object-relational DBMS**
 - Relational DBMS have been extending their model to incorporate object database concepts and other capabilities
- ❖ Based on number of *users*, DBMS can be classified as follows:
 - **Single-user systems** support only one user at a time and are mostly used with PCs.
 - **Multiuser systems** which include the majority of DBMSs, support concurrent multiple users.
- ❖ Based on *number of Sites* DBMS can be classified as follows:
 - **Centralized DBMS**
 - A DBMS is **centralized** if the data is stored at a single computer site.
 - A centralized DBMS can support multiple users, but the DBMS and the database reside totally at a single computer site.
 - **Distributed DBMS**
 - A **distributed** DBMS (DDBMS) can have the actual database and DBMS software distributed over many sites, connected by a computer network.
- ❖ Based on *the cost* DBMS can be classified as follows:
 - The DBMS package cost between \$10000 to 100000
 - Single user work with micro computers cost between \$100 and \$3000
 - Other few elaborate packages cost more than \$10000.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

UNIT -2 : Data Modeling Using the Entity-Relationship (ER) Model

- 2.1 Using High-Level Conceptual Data Models for Database Design
- 2.2 An example Database Application
- 2.3 Entity Types, Entity Sets, attributes and keys,
- 2.4 Relation Types, Relationship Sets, roles and structural constraints
- 2.5 Weak Entity Types
- 2.6 Refining the ER Design for the Company Database
- 2.7 ER Diagrams, naming, conventions and design issues
- 2.8 Relationship Types of Degree Higher Than Two.

This chapter will mainly focus on the traditional approach for database structures and constraints during database design and present the modelling concepts of the **Entity-Relationship (ER) model**, which is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of database applications, and many database design tools employ its concepts. We describe the basic data-structuring concepts and constraints of the ER model and discuss their use in the design of conceptual schemas for database applications. We also present the diagrammatic notation associated with the ER model, known as ER diagrams.

2.1 Using High-Level Conceptual Data Models for Database Design

The database design using high-level conceptual data models consists of different phases as shown in the figure.2.1.

Requirements collection and analysis

The database designers interview database users to understand and document their **data requirements**. These requirements should be specified in as detailed and complete.

In parallel with specifying the data requirements, it is useful to specify the data requirements of the users.

Functional requirements consists of the user defined operations that will be applied to the database, including both retrieval and updates

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Conceptual design

Once all the requirements have been collected and analyzed, the next step is to create a conceptual schema for the database using high level conceptual data model. This step is called conceptual

Conceptual schema includes detailed description of the entity types, relationships and constraints

Logical design

After the conceptual design, the next step in database design is the actual implementation of the database, using a commercial DBMS.

A simplified diagram to illustrate the main phases of database design.

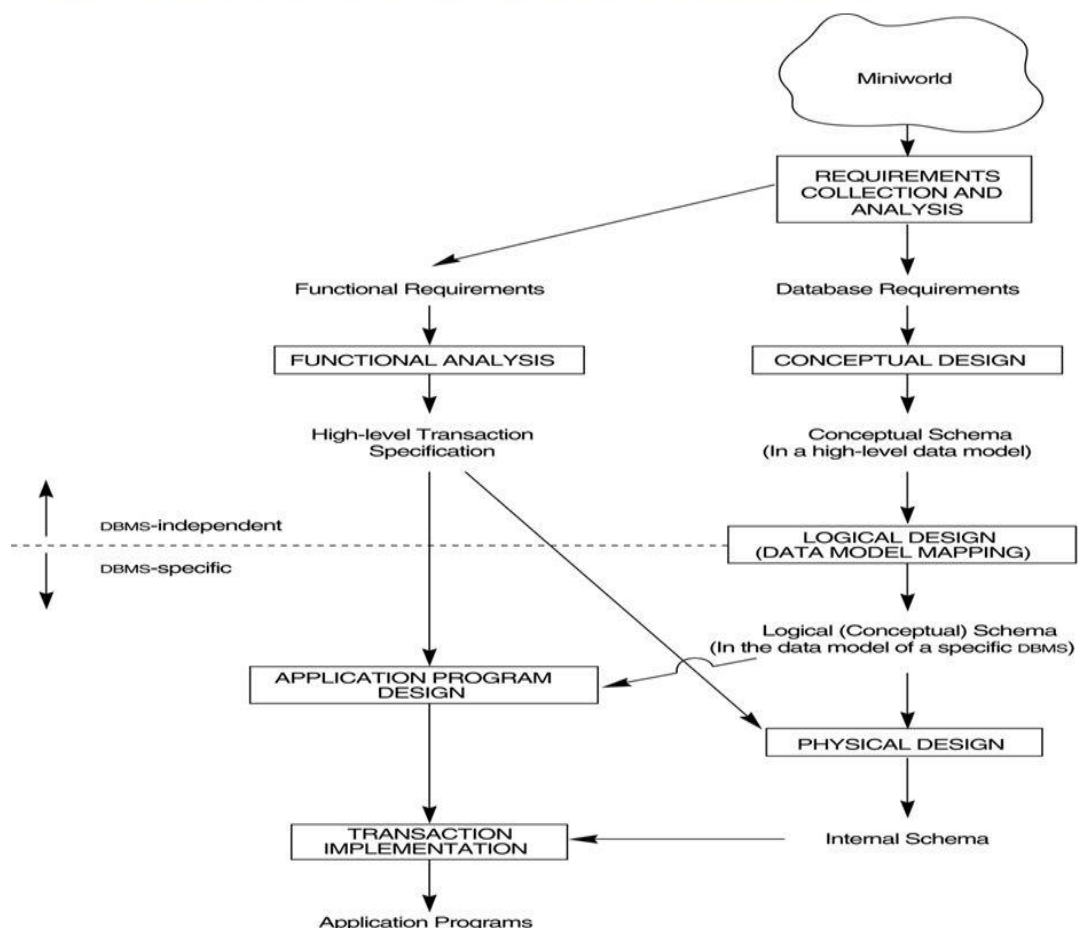


Figure 2.1 a simplified overview of the database design process

The conceptual schema is transformed from the high-level data model into the implementation data model using DBMS.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

This step is called **logical design** or **data model mapping**; its result is a database schema in the implementation data model of the DBMS.

Physical Design

The last step is the **physical design** phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified.

In parallel with these activities, application programs are designed and implemented as database transactions.

2.2 An example Database Application

The schema for database application can be displayed by means of the graphical notation known as **ER diagrams**.

For example,

A sample database application, called COMPANY, which serves to illustrate the basic ER model concepts.

- The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
- A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
- We store each employee's name, Social Security number, address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).
- We want to keep track of the dependents of each employee for insurance purposes. We keep each dependent's first name, sex, birth date, and relationship to the employee.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

2.3 Entity Types, Entity Sets, Attributes, and Keys

The ER model describes data as *entities*, *relationships*, and *attributes*.

2.3.1 Entities and Attributes

Entities: The basic object that the ER model represents is an **entity**, which is a *thing* in the real world with an independent existence. An entity may be an object with a physical existence (for example, EMPLOYEE entity, STUDENT entity and etc).

Attributes: The properties that describes an entity, each entity has **attribute**

For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job`.

Several types of attributes occur in the ER model:

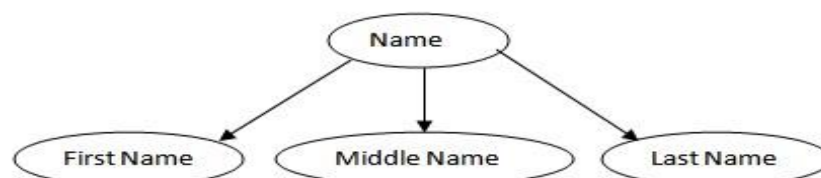
- Simple versus composite
 - Single valued versus multi-valued
 - Stored versus derived
 - Null Attribute
 - Complex attribute
-
- **Composite versus Simple (Atomic) Attributes.**

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

Example: Age of EMPLOYEE Entity.

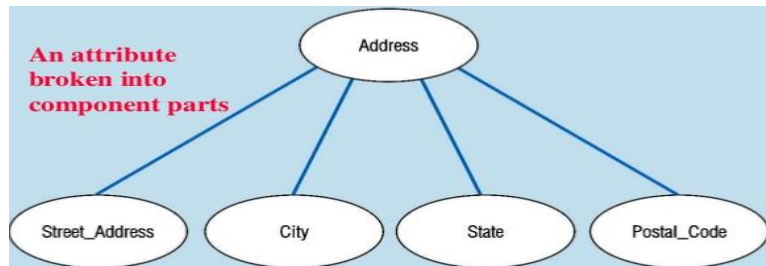
Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.

For example, The Name attribute of the EMPLOYEE entity can be divided into FirstName, Middle Name and LastName.



Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

The Address attribute of the EMPLOYEE entity can be subdivided into Street_address, City, State, and Zip.



- **Single-Valued versus Multivalued Attributes.**

The attributes have a single value for a particular entity. Such attributes are called **single-valued**. For example: Age is a single-valued attribute of a person

The attributes can have set of values for the same entity is called multivalued attribute .
for example : College_degrees attribute for a person. One person may not have a college degree, another person may have one, and a third person may have two or more degrees; therefore, different people can have different *numbers* of *values* for the College_degrees attribute. Such attributes are called **multivalued attribute**.

- **Stored versus Derived Attributes**

For an EMPLOYEE entity, the value of Age can be determined from the current (today's) date and the value of that person's Birth_date attribute.

The Age attribute is hence called a **derived attribute** and is said to be **derivable from** the Birth_date attribute, which is called a **stored attribute**.

- **NULL Attribute**

The Attribute may not have an applicable value for it.

For example, Email attribute, some people may have email id. For those who do not have email id, then the value will be NULL for an Email attribute.

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

The *unknown* category of NULL can be further classified into two cases.

- The first case arises when it is known that the attribute value exists but is *missing* for example, if the Height attribute of a person is listed as NULL.
- The second case arises when it is *not known* whether the attribute value exists or not. for example, if the Email attribute of a person is NULL.

- **Complex Attributes.**

Composite and multivalued attributes can be nested arbitrarily. We can represent composite attribute between parentheses () and separating the components with commas, and by displaying multivalued attributes between braces { }. Such attributes are called **complex attributes**.

For example, if a person can have more than one residence and each residence can have a single address and multiple phones, an attribute Address_phone for a person is specified below. Both Phone and Address are themselves composite attributes.

{Address_phone({Phone(Area_code,Phone_number)},Address(Street_address(Number, Street,Apartment_number),City,State,Zip))}

2.3.2 Entity Types, Entity Sets, Keys, and Value Sets

An **entity type** defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes.

Example: EMPLOYEE Entity type with Name, age, salary so on attributes

The collection of all entities of a particular entity type in the database at any point in time is called an **entity set**; the entity set is usually referred to using the same name as the entity type.

For example, EMPLOYEE refers to both a type of entity as well as the current set of all employee entities in the database.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

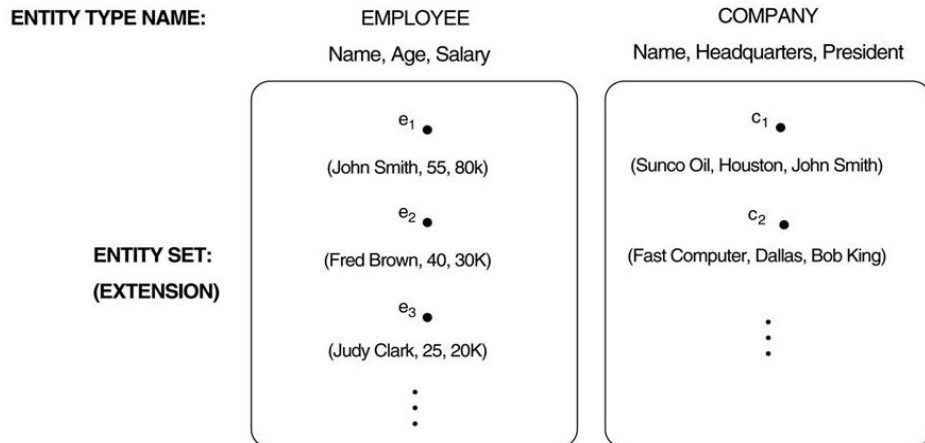


FIGURE 2.2 Two entity types, *EMPLOYEE* and *COMPANY*, and some member entities of each.

An entity type describes the **schema or intension** for a set of entities that share the same structure.

The collection of entities of a particular entity type is grouped into an entity set, which is also called the **extension** of the entity type.

Key Attributes of an Entity Type

The constraint on the entities of an entity type is the **key** or **uniqueness constraint** on attributes. A **key attribute** value can be used to identify each entity uniquely.

For Example: In *EMPLOYEE* entity type SSN attribute is a key attribute whose value is used to uniquely identify an employee in the company.

Some entity types may have *more than one* key attribute. Combination of values for key attributes should be unique to identify.

An entity type may also have *no key*, in which case it is called a **weak entity type**

Value Sets (Domains) of Attributes.

Each simple attribute of an entity type is associated with a **value set** (or **domain** of values), which specifies the set of values that may be assigned to that attribute for each individual entity.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

For Example: The range of ages allowed for employees is between 16 and 70, we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70.

2.3.3 Initial Conceptual Design of the COMPANY Database

The entity types for the COMPANY database, based on the Company requirements.

1 An entity type DEPARTMENT with attributes Name, Number, Locations, Manager, and Manager_start_date. Locations is the only multivalued attribute. We can specify that both Name and Number are (separate) key attributes because each was specified to be unique.

2. An entity type PROJECT with attributes Name, Number, Location, and Controlling_department. Both Name and Number are (separate) key attributes.

3. An entity type EMPLOYEE with attributes Name, Ssn, Sex, Address, Salary, Birth_date, Department, and Supervisor.

4. An entity type DEPENDENT with attributes Employee, Dependent_name, Sex, Birth_date, and Relationship (to the employee).

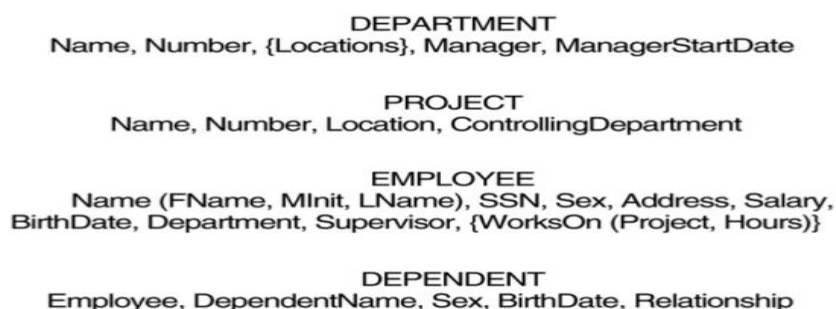


Fig2.3: Preliminary design of entity types for the COMPANY database.

2.4 Relationship Types, Relationship Sets, Roles, and Structural Constraints

Whenever an attribute of one entity type refers to another entity type, some relationship exists. For example,

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- 1 The attribute Manager of DEPARTMENT refers to an employee who manages the department.
- 2 The attribute Controlling_department of PROJECT refers to the department that controls the project;
- 3 The attribute Supervisor of EMPLOYEE refers to another employee.
- 4 The attribute Department of EMPLOYEE refers to the department for which the employee works and so on.

In the ER model, these references should not be represented as attributes but as **relationships**.

2.4.1 Relationship Types, Sets, and Instances

An attribute of one entity refers to another entity. Represent such references as relationships not attributes.

A **relationship type** R among n entity types E_1, E_2, \dots, E_n . Defines a set of associations among entities from these entity types

Relationship instance (ri)

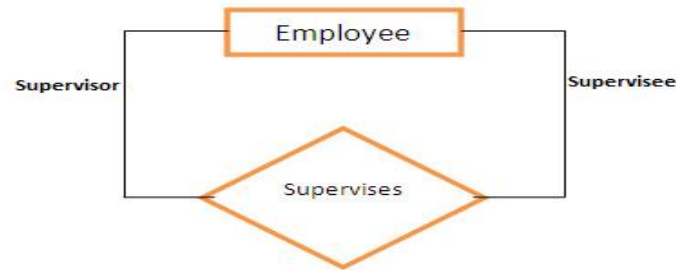
- Each ri associates n individual entities (e_1, e_2, \dots, e_n)
- Each entity e_j in ri is a member of entity set E_j
- Relationships uniquely identified by keys of participating entities

2.4.2 Relationship Degree, Role Names, and Recursive Relationships Degree of a Relationship Type.

The **degree** of a relationship type is the number of participating entity types.

Unary Relationship exists when an association is maintained with a single entity

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus



Binary Relationship: a relationship type of degree two is called as **binary relationship**

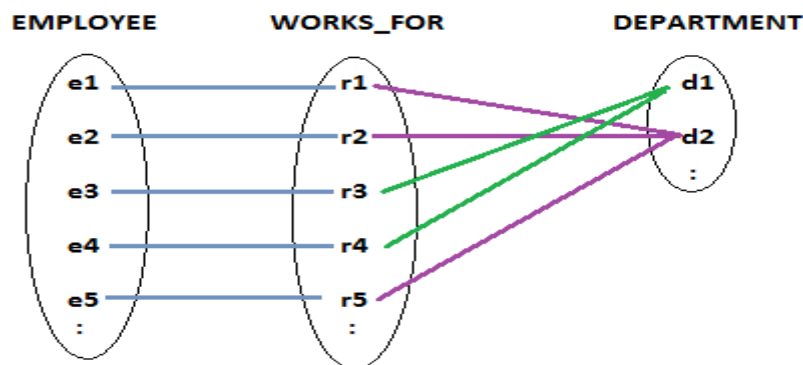


FIGURE 2.4 Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Ternary Relationship: Relationship of degree three is called **ternary**.

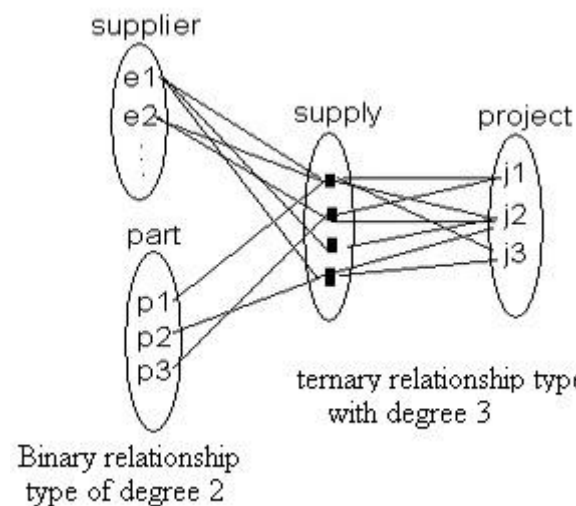


FIGURE 2.5 some relationship instances in the SUPPLY ternary relationship set.

Role Names and Recursive Relationships.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and helps to explain what the relationship means.

For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of *employee* or *worker* and DEPARTMENT plays the role of *department* or *employer*.

Role names are important when same entity type participates more than once in a relationship type in *different roles*. In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships**.

Figure below shows an example. The SUPERVISION relationship type relates an employee to a supervisor, the EMPLOYEE entity type *participates twice* in SUPERVISION: once in the role of *supervisor* (or *boss*), and once in the role of *supervisee* (or *subordinate*).

As shown in figure 2.6 the lines marked '1' represent the supervisor role, and lines marked '2' represent the supervisee role;

- e_1 supervises e_2 and e_3 ,
- e_4 supervises e_6 and e_7 , and
- e_5 supervises e_1 and e_4 .

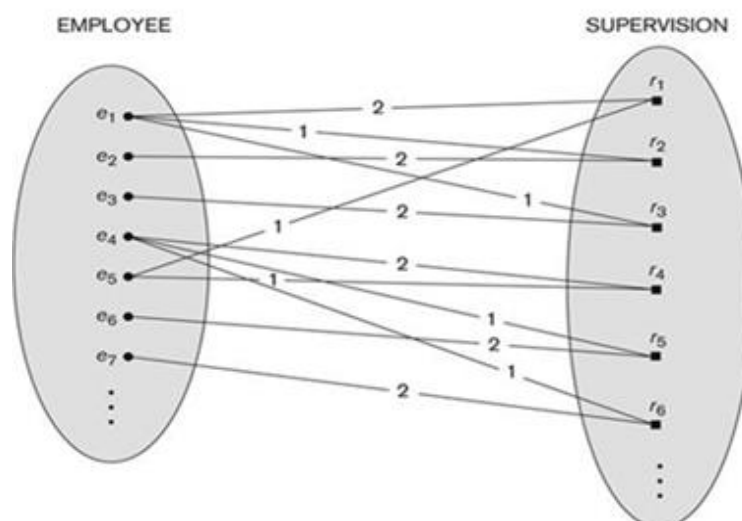


FIGURE 2.6 A recursive relationship *SUPERVISION* between *EMPLOYEE* in the supervisor role (1) and *EMPLOYEE* in the subordinate role (2).

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

2.4.3 Constraints on Binary Relationship Types

The two main types of binary relationship constraints:

1 *Cardinality ratio*

2 *Participation*.

Cardinality Ratios for Binary Relationships.

The **cardinality ratio** for a binary relationship specifies the *maximum* number of relationship instances that an entity can participate in

- One to One (1:1)
- One to Many (1:M)
- Many to one (M:1)
- Many to Many (M:M)

One to One (1:1) Relationships

An entity in A is associated with at most one entity in B

Example: an Employee can manage only one department and that department has only one manager.

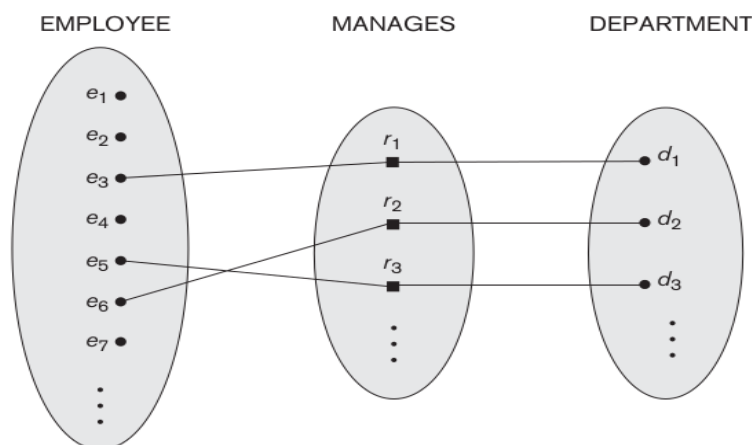


FIGURE 2.7 A 1:1 relationship, MANAGES.

One to Many(1: N)/ Many to One (N:1)

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

An entity in A is associated with any number of entities in B is 1:N. An entity in B however can be associated with atmost one entity in A.

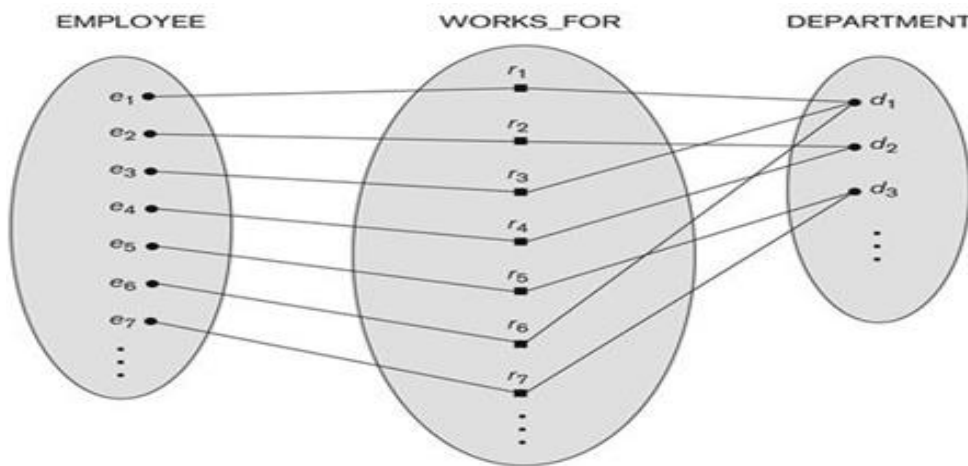


FIGURE 2.8 A 1: N/ N: 1 relationship, WORKS_ON.

An employee can work for at most one department and a department can have many employees.

Many to Many (M:M)

An entity in A is associated with any number of entities in B is 1:N. An entity in B however can be associated with any number of entities in A.

Example : In relationship type WORKS_ON an employee can work on many projects and a project can be handled by many employees

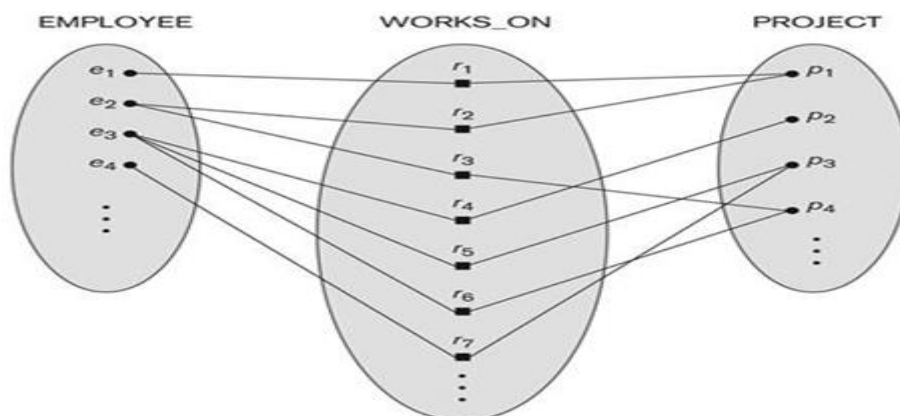


FIGURE 2.9 An M: N relationship, WORKS_ON.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

2.4.4 Participation Constraints and Existence Dependencies

The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.

There are two types of participation constraints

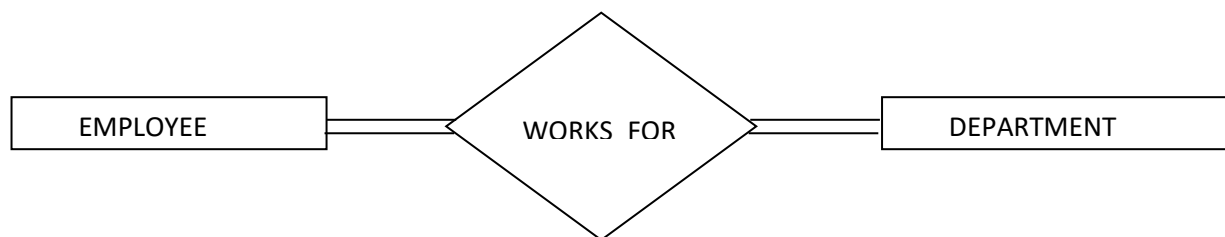
- Total Participation or existence dependency
- Partial Participation
-

Total Participation: If every entity participates in atleast one relationship instance in R

Example: *Every* employee must work for a department, and then an employee entity can exist only if it participates in at least one WORKS_FOR relationship instance.

Thus, the participation of EMPLOYEE in WORKS_FOR is called **total participation**

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



Partial Participation—if some entities in Entity participate in relationship R is called total participation

Example: Not all employee manages the department, so the participation of EMPLOYEE in the MANAGES relationship type is **partial**.



Structural constraints

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

ER notation for structural constraints involves associating a pair of integer numbers (min, max) with each participation of an entity type E in a relationship type R, where $0 \leq \min \leq \max$, and $\max \geq 1$.

Figure 2.12 displays the COMPANY database schema using the (min, max) notation.

(1,N) Structural constraint in WORKS_FOR relationship between EMPLOYEE and DEPARTMENT shows that minimum one employee should work for department and maximum many employees may work for department.

2.4.5 Attributes of Relationship Types

Relationship types can also have attributes, similar to those of entity types.

For example, to record the number of hours per week that an employee works on a particular project, we can include an attribute Hours for the WORKS_ON relationship type

Another example is to include the date on which a manager started managing a department via an attribute Start_date for the MANAGES relationship type.

2.5 Weak Entity Types

Entity types that do not have key attributes of their own are called **weak entity types**.

Regular entity types that have a key attribute are called **strong entity types**.

Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other Entity type the **Identifying** or **Owner entity type**.

The relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

A weak entity type always has a ***total participation constraint*** with respect to its identifying relationship because a weak entity cannot be identified without an owner entity. However, not every existence dependency results in a weak entity type.

A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are *related to the same owner entity*.

2.6 Refining the ER design for the company database

Refining the COMPANY database by adding cardinality ratio and participation constraint of each relationship type are determined from the requirements, we specify the following relationship types:

1. **MANAGES**, a 1:1 relationship type between EMPLOYEE and DEPARTMENT. EMPLOYEE participation is partial. DEPARTMENT participation is not clear from the requirements. We question the users, who say that a department must have a manager at all times, which implies total participation. The attribute Start_Date is assigned to this relationship type.
2. **WORKS_FOR**, a 1:N relationship type between DEPARTMENT and EMPLOYEE. Both participations are total.
3. **CONTROLS**, a 1:N relationship type between DEPARTMENT and PROJECT. The participation of PROJECT is total, whereas that of DEPARTMENT is determined to be partial, after consultation with the users indicates that some departments may control no projects.
4. **SUPERVISION**, a 1:N relationship type between EMPLOYEE (in the supervisor role) and EMPLOYEE (in the supervisee role). Both participations are determined to be partial, after the users indicate that not every employee is a supervisor and not every employee has a supervisor.
5. **WORKS_ON**, determined to be an M:N relationship type with attribute Hours, after the users indicate that a project can have several employees working on it. Both participations are determined to be total.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

6. **DEPENDENTS_OF**, a 1:N relationship type between EMPLOYEE and DEPENDENT, which is also the identifying relationship for the weak entity type DEPENDENT. The participation of EMPLOYEE is partial, whereas that of DEPENDENT is total.

2.7 ER Diagrams, naming, conventions and design issues

2.7.1 Proper Naming of Schema Constructs

When designing a database schema, the choice of names for entity types, attributes, relationship types, and (particularly) roles is not always straightforward.

We choose to use *singular names* for entity types, rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type.

In our ER diagrams, we will use the convention that entity type and relationship type names are uppercase letters, attribute names have their initial letter capitalized, and role names are lowercase letters.

The database requirements, the *nouns* appearing in the narrative tend to give rise to entity type names, and the *verbs* tend to indicate names of relationship types.

Attribute names generally arise from additional nouns that describe the nouns corresponding to entity types.

2.7.2 Design Choices for ER Conceptual Design

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

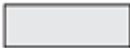








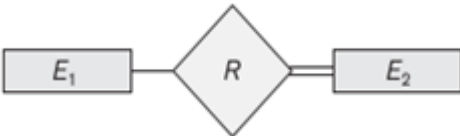

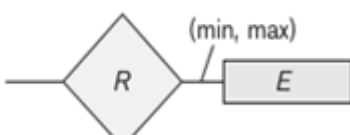
| Symbol | Meaning |
|---|---|
|  | Entity |
|  | Weak Entity |
|  | Relationship |
|  | Identifying Relationship |
|  | Attribute |
|  | Key Attribute |
|  | Multivalued Attribute |
|  | Composite Attribute |
|  | Derived Attribute |
|  | Total Participation of E_2 in R |
|  | Cardinality Ratio 1 : N for $E_1:E_2$ in R |
|  | Structural Constraint (min, max) on Participation of E in R |

FIGURE 2.10 Summary of the notation for ER diagrams.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

2.7.3 Alternative Notations for ER Diagrams

There are many alternative diagrammatic notations for displaying ER diagrams.

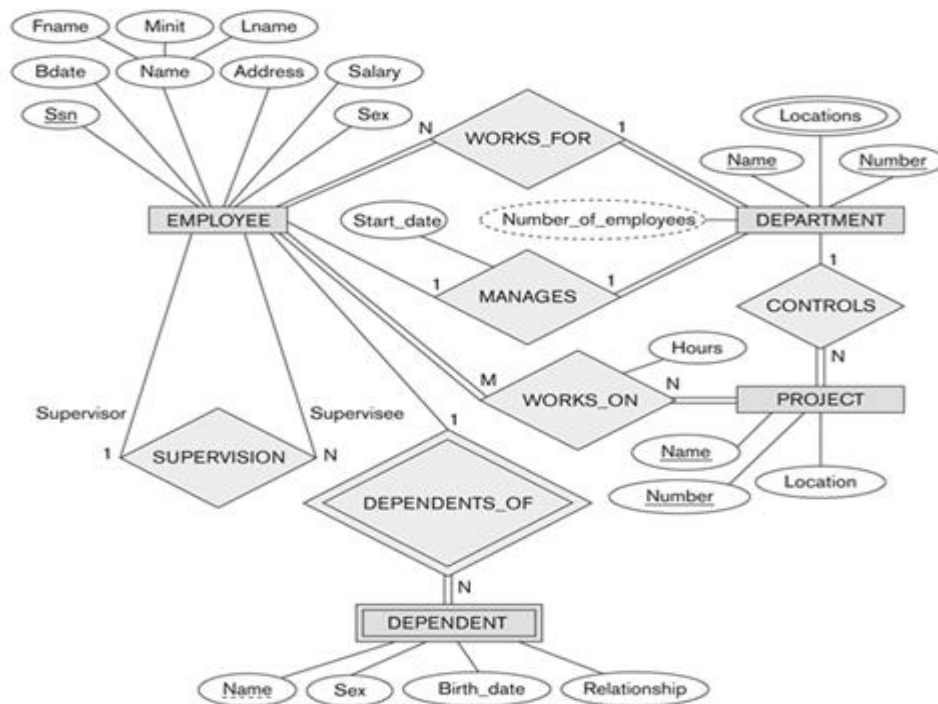
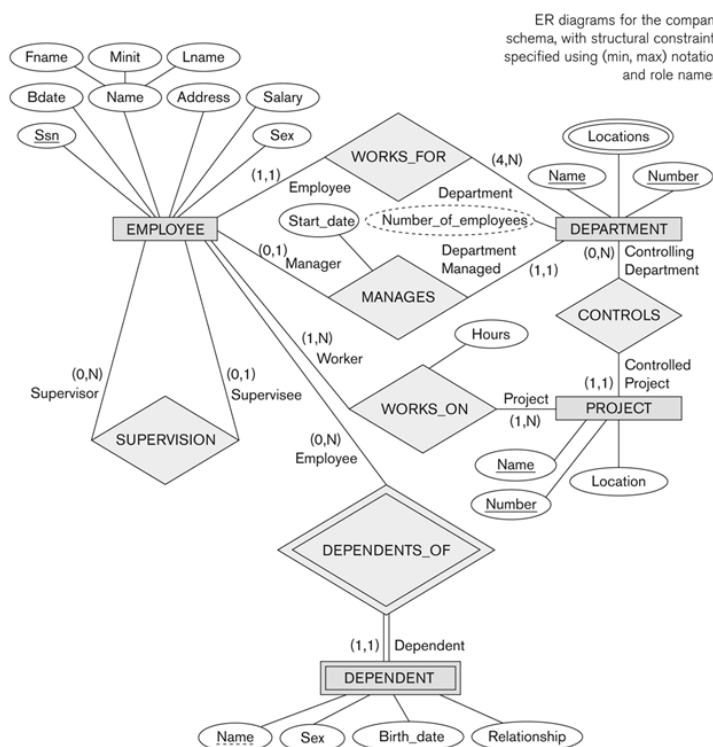


FIGURE 2.11 An ER schema diagram for the COMPANY database

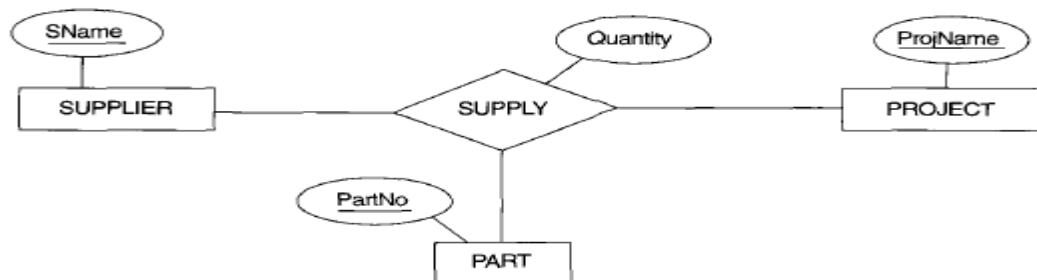


Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

FIGURE 2.12 ER diagrams for the COMPANY schema, with structural constraints specified using (min, max) notation.

2.8 Relationship Types of Degree Higher Than Two.

Three entities taking part in relationship is called **ternary relationship (degree higher than two)**. Example: In the below diagram SUPPLY is a set of relationship instances (s, j, p) , where s is a SUPPLIER who is currently supplying a PART-, p to a PROJECT.



Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

UNIT-3 : The Relational Data Model and Relational Database Constraints

3.1 Relational Model concepts

3.2 Relational Model Constraints and relational database schemas

3.3 Update Operation, Transaction and Dealing with constraints violations

3.1 Relational Model Concepts

The relational model represents the database as a collection of *relations*.

When a relation is thought of as a **table** of values, each row in the table represents a collection of related data values. A row represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help to interpret the meaning of the values in each row.

For example, In a STUDENT each row represents facts about a particular student entity. The column names—Name, Student_number, Class, and Major—specify how to interpret the data values in each row, based on the column each value is in. All values in a column are of the same data type.

A row is called a *tuple*, a column header is called an *attribute*, and the table is called a *relation*.

The data type describing the types of values that can appear in each column is represented by a *domain* of possible values.

3.1.1 Domains, Attributes, Tuples, and Relations

A **domain** D is a set of atomic values. By **atomic** we mean that each value in the domain is indivisible. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn. It is also useful to specify a name for the domain, to help in interpreting its values. Some examples of domains follow:

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- `Social_security_numbers(SSN)`. The set of valid nine-digit Social Security numbers. (This is a unique identifier assigned to each person in the United States for employment, tax, and benefits purposes.)
- `Names`: The set of character strings that represent names of persons.
- `Employee_ages`. Possible ages of employees in a company; each must be an integer value between 15 and 80.
- `Academic_department_names`. The set of academic department names in a university, such as Computer Science, Economics, and Physics.
- `Academic_department_codes`. The set of academic department codes, such as 'CS', 'ECON', and 'PHYS'.

The preceding are called *logical* definitions of domains. A **data type** or **format** is also specified for each domain.

For example, the data type for `Employee_ages` is an integer number between 15 and 80. For a domain is thus given a name, data type, and format. Additional information for interpreting the values of a domain can also be given; for example, a numeric domain such as `Person_weights` should have the units of measurement, such as pounds or kilograms.

A **relation schema** R , denoted by $R(A_1, A_2, \dots, A_n)$, is made up of a relation name R and a list of attributes, A_1, A_2, \dots, A_n . Each **attribute** A_i is the name of a role played by some domain D in the relation schema R . D is called the **domain** of A_i and is denoted by $\text{dom}(A_i)$.

A relation schema is used to *describe* a relation R . The **degree** of a relation is the number of attributes n of its relation schema.

A relation of degree seven, which stores information about university students, would contain seven attributes describing each student. as follows:

STUDENT(Name, Ssn, Home_phone, Address, Office_phone, Age, Gpa)

Using the data type of each attribute, the definition is sometimes written as:

STUDENT(Name: string, Ssn: string, Home_phone: string, Address: string, Office_phone: string, Age: integer, Gpa: real)

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

A **relation** (or **relation state**) r of the relation schema $R(A_1, A_2, \dots, A_n)$, also denoted by $r(R)$, is a set of n -tuples $r = \{t_1, t_2, \dots, t_m\}$. Each **n -tuple** t is an ordered list of n values $t = \langle v_1, v_2, \dots, v_n \rangle$, where each value v_i , $1 \leq i \leq n$, is an element of $\text{dom}(A_i)$ or is a special NULL value. The i th value in tuple t , which corresponds to the attribute A_i , is referred to as $t[A_i]$ or $t.A_i$.

The terms **relation intension** for the schema R and **relation extension** for a relation state $r(R)$ are also commonly used.

Figure 3.1 shows an example of a STUDENT relation, which corresponds to the STUDENT schema just specified. Each tuple in the relation represents a particular student entity

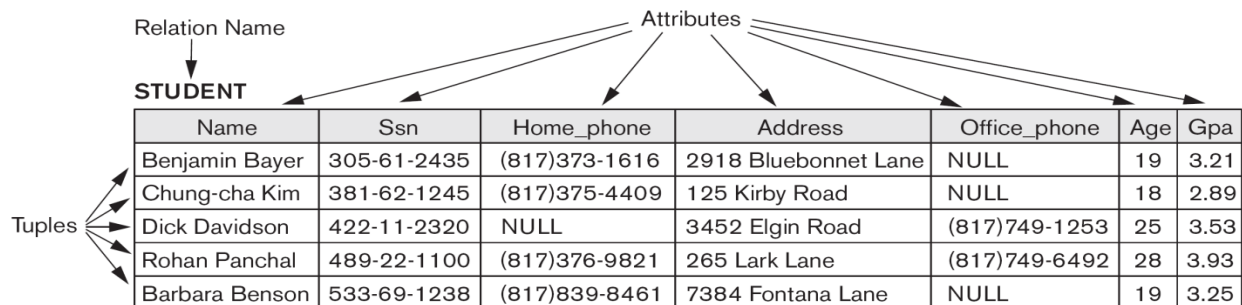


FIGURE 3.1 The attributes and tuples of a relation STUDENT.

Cartesian Product

The **Cartesian Product** is an operator which works on two sets. It is sometimes called the CROSS PRODUCT or CROSS JOIN. It combines the tuples of one relation with all the tuples of the other relation.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Student (enrIno, Name) and Branch (branch_name)

| EnrIno | Name |
|--------|-------|
| 020256 | Amit |
| 567722 | Sumit |
| 234511 | Ravi |

| branchname |
|------------|
| CS |
| IT |

R=student X Branch

R(EnrIno,name,branchname)

R=Student X branch

| EnrIno | Name | Branchname |
|--------|-------|------------|
| 020256 | Amit | CS |
| 020256 | Amit | IT |
| 567722 | Sumit | CS |
| 567722 | Sumit | IT |
| 234511 | Ravi | CS |
| 234511 | Ravi | IT |

3.1.2 Characteristics of Relations

The earlier definition of relations implies certain characteristics that make a relation different from a file or a table. We now discuss some of these characteristics.

Ordering of Tuples in a Relation.

A relation is defined as a *set* of tuples. Tuples in a relation do not have any particular order. However, in a file, records are physically stored on disk so there always is an order among the records. This ordering indicates first, second, *i*th, and last records in the file. Similarly, when we display a relation as a table, the rows are displayed in a certain order.

Tuple ordering is not part of a relation definition because a relation attempts to represent facts at a logical or abstract level.

Ordering of Values within a Tuple and an Alternative Definition of a Relation.

A relation, can have *n*-tuple is an *ordered list* of *n* values, so the ordering of values in a tuple and hence of attributes in a relation schema—is important.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

The order of attributes and their values is *not* that important as long as the correspondence between attributes and values is maintained.

An **alternative definition** of a relation can be given, making the ordering of values in a tuple *unnecessary*. In this definition, a relation schema $R = \{A_1, A_2, \dots, A_n\}$ is a *set* of attributes, and a relation state $r(R)$ is a finite set of mappings $r = \{t_1, t_2, \dots, t_m\}$, where each tuple t_i is a **mapping** from R to D , and D is the **union** (denoted by \cup) of the attribute domains; that is, $D = \text{dom}(A_1) \cup \text{dom}(A_2) \cup \dots \cup \text{dom}(A_n)$.

Values and NULLs in the Tuples

Each value in a tuple is an **atomic** value that is not divisible into components. Hence, composite and multivalued attributes are not allowed.

Multivalued attributes must be represented by separate relations, and composite attributes are represented only by their simple component attributes in the basic relational model.

An important concept is that of NULL values, which are used to represent the values of attributes that may be unknown or may not apply to a tuple. A special value, called NULL, is used in these cases.

We can have several meanings for NULL values, such as *value unknown*, *value exists but is not available*, or *attribute does not apply* to this tuple (also known as *value undefined*).

Interpretation (Meaning) of a Relation.

The relation schema can be interpreted as a declaration or a type of **assertion**.

For example, the schema of the STUDENT relation of. In general, a student entity has a Name, Ssn, Home_phone, Address, Office_phone, Age, and Gpa.

Each tuple in the relation can then be interpreted as a **fact** or a particular instance of the assertion.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

For example, the first tuple in Figure 3.1 asserts the fact that there is a STUDENT whose Name is Ramu, Ssn is 305-61-2435, Age is 19, and so on.

Notice that some relations may represent facts about *entities*, whereas other relations may represent facts about *relationships*.

An alternative interpretation of a relation schema is as a **predicate**; in this case, the values in each tuple are interpreted as values that *satisfy* the predicate.

For example, the predicate STUDENT (Name, Ssn, ...) is true for the five tuples in relation STUDENT.

3.2 Relational Model Constraints and Relational Database Schemas

The state of the whole database will correspond to the states of all its relations at a particular point in time. There are generally many restrictions or **constraints** on the actual values in a database state.

Constraints on databases can generally be divided into three main categories:

1. Constraints those are inherent in the data model. We call these **inherent model-based constraints** or **implicit constraints**.
2. Constraints that can be directly expressed in schemas of the data model, typically by specifying them in the DDL. We call these **schema-based constraints** or **explicit constraints**.
3. Constraints that *cannot* be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs. We call these **application-based** or **semantic constraints** or **business rules**.

The **schema-based constraints** include

- Domain constraints,
- Key constraints and constraints on NULLs,
- Entity integrity constraints, and

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- Referential integrity constraints.

3.2.1 Domain Constraints

Domain constraints specify that within each tuple, the value of each attribute A must be an atomic value from the domain $\text{dom}(A)$.

The data types associated with domains typically include standard numeric data types for integers (such as short integer, integer, and long integer) and real numbers (float and double precision float). Characters, Booleans, fixed-length strings, and variable-length strings are also available, as are date, time, timestamp, and money, or other special data types.

Other possible domains may be described by a subrange of values from a data type or as an enumerated data type in which all possible values are explicitly listed.

3.2.2 Key Constraints and Constraints on NULL Values

A *relation* is defined as a *set of tuples* and all elements of a set are distinct; hence, all tuples in a relation must also be distinct.

This means that no two tuples can have the same combination of values for *all* their attributes.

A relation schema R with the property that no two tuples in any relation state r of R should have the same combination of values for these attributes.

Suppose that we denote one such subset of attributes by SK; then for any two *distinct* tuples t_1 and t_2 in a relation state r of R , we have the constraint that:

$$t_1[\text{SK}] \neq t_2[\text{SK}]$$

Any such set of attributes SK is called a **superkey** of the relation schema R . A superkey SK specifies a *uniqueness constraint* that no two distinct tuples in any state r of R can have the same value for SK. Every relation has at least one default superkey—the set of all its attributes.

A superkey can have redundant attributes, however, a *key*, which has no redundancy.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

A **key** K of a relation schema R is a superkey of R with the additional property that removing any attribute A from K leaves a set of attributes K_- that is not a superkey of R anymore.

Hence, a key satisfies two properties:

1. Two distinct tuples in any state of the relation cannot have identical values for (all) the attributes in the key. This first property also applies to a superkey.
2. It is a *minimal superkey*—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint in condition 1 hold.

The attribute set {Ssn} is a key of STUDENT because no two student tuples can have the same value for Ssn. Any set of attributes that includes Ssn—for example, {Ssn, Name, Age}—is a superkey. However, the superkey {Ssn, Name, Age} is not a key of STUDENT because removing Name or Age or both from the set still leaves us with a superkey.

A relation schema may have more than one key. In this case, each of the keys is called a **candidate key**. One of the candidate keys is designated to be the **primary key** of the relation and others are called secondary keys.

3.2.3 Entity integrity constraint

The **entity integrity constraint** states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples.

For example, if two or more tuples had NULL for their primary keys, we may not be able to distinguish them.

3.2.4 Referential Integrity Constraints, and Foreign Keys

Key constraints and entity integrity constraints are specified on individual relations. The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

For Example: The attribute Dno of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the Dnumber value of some tuple in the DEPARTMENT relation.

Foreign key

The conditions for a foreign key, specify a referential integrity constraint between the two relation schemas $R1$ and $R2$. A set of attributes FK in relation schema $R1$ is a **foreign key** of $R1$ that **references** relation $R2$ if it satisfies the following rules:

1. The attributes in FK have the same domain(s) as the primary key attributes PK of $R2$; the attributes FK are said to **reference** or **refer to** the relation $R2$.

2. A value of FK in a tuple $t1$ of the current state $r1(R1)$ either occurs as a value of PK for some tuple $t2$ in the current state $r2(R2)$ or is *NULL*.

i.e $t1[FK] = t2[PK]$, and we say that the tuple $t1$ **references** or **refers to** the tuple $t2$.

In this definition, $R1$ is called the **referencing relation** and $R2$ is the **referenced relation**.

If these two conditions hold, a **referential integrity constraint** from $R1$ to $R2$ is said to hold.

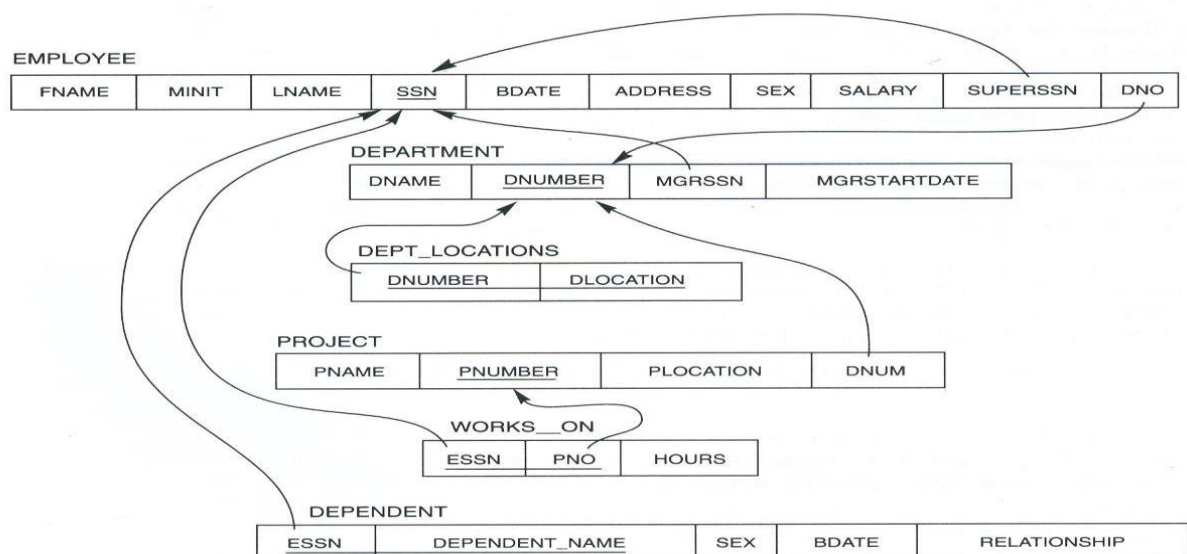


FIGURE 3.2 Referential integrity constraints displayed on the COMPANY relational database schema.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

3.2.5 Other Types of Constraints

A large class of general constraints may have to be specified and enforced on a relational database, sometimes called *semantic integrity constraints*,

Examples of such constraints are the salary of an employee should not exceed the salary of the employee's supervisor.

Some constraints can be specified and enforced within the application programs that update the database, or by using a general-purpose **constraint specification language**. Mechanisms called **triggers** and **assertions** can be used.

Example: The maximum number of hours an employee can work on all projects per week is 56.

Another type of constraint is the *functional dependency* constraint, which establishes a functional relationship among two sets of attributes X and Y . This constraint specifies that the value of X determines a unique value of Y in all states of a relation; it is denoted as a functional dependency $X \rightarrow Y$.

The **state constraints** define the constraints that a *valid state* of the database must satisfy.

Another type of constraint, called **transition constraints**, can be defined to deal with state changes in the database. An example of a transition constraint is: “the salary of an employee can only increase.”

3.2.6 Relational Databases and Relational Database Schemas

A relational database usually contains many relations, with tuples in relations that are related in various ways.

A **relational database schema** S is a set of relation schemas $S = \{R_1, R_2, \dots, R_m\}$ and a set of **integrity constraints** IC . A **relational database state** DB of S is a set of relation states $DB = \{r_1, r_2, \dots, r_m\}$ such that each r_i is a state of R_i and such that the r_i relation states satisfy the integrity constraints specified in IC .

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Figure 3.3 shows a relational database schema that we call $COMPANY = \{EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT\}$.

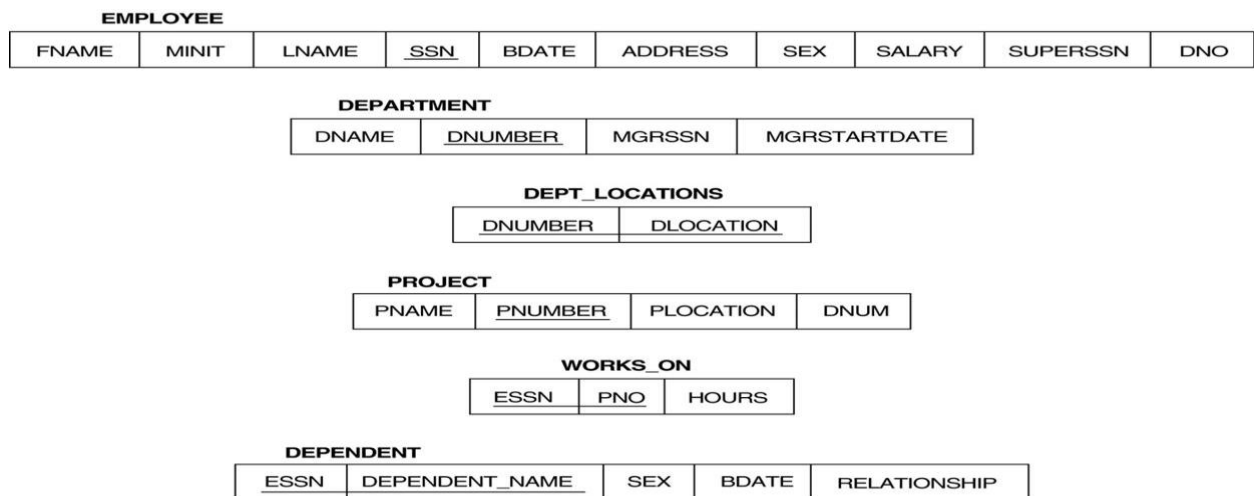


FIGURE 3.3 Schema diagram for the COMPANY relational database schema.

3.3 Update Operations, Transactions and Dealing with Constraint Violations

The operations of the relational model can be categorized into *retrievals* and *updates*.

There are three basic operations database **modification** or **update** operations.

- **Insert,**
- **Delete, and**
- **Update (or Modify).**

Insert is used to insert one or more new tuples in a relation,

Delete is used to delete tuples,

Update (or Modify) is used to change the values of some attributes in existing tuples.

Whenever these operations are applied, the integrity constraints specified on the relational database schema should not be violated.

3.3.1 The Insert Operation

The **Insert** operation provides a list of attribute values for a new tuple t that is to be inserted into a relation R . Insert can violate any of the four types of constraints,

- Domain constraints can be violated if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- Key constraints can be violated if a key value in the new tuple t already exists in another tuple in the relation $r(R)$.
- Entity integrity can be violated when the primary key of the new tuple t is NULL.
- Referential integrity can be violated if the value of any foreign key in t refers to a tuple that does not exist in the referenced relation..

■ Operation:

Insert <‘Cecilia’, ‘F’, ‘Kolonsky’, NULL, ‘1960-04-05’, ‘6357 Windy Lane, Katy, TX’, F, 28000, NULL, 4> into EMPLOYEE.

Result: This insertion violates the entity integrity constraint (NULL for the primary key Ssn), so it is rejected.

■ Operation:

Insert <‘Alicia’, ‘J’, ‘Zelaya’, ‘999887777’, ‘1960-04-05’, ‘6357 Windy Lane, Katy, TX’, F, 28000, 987654321’, 4> into EMPLOYEE.

Result: This insertion violates the key constraint because another tuple with the same Ssn value already exists in the EMPLOYEE relation, and so it is rejected.

■ Operation:

Insert <‘Cecilia’, ‘F’, ‘Kolonsky’, ‘677678989’, ‘1960-04-05’, ‘6357 Windswept, Katy, TX’, F, 28000, ‘987654321’, 7> into EMPLOYEE.

Result: This insertion violates the referential integrity constraint specified on Dno in EMPLOYEE because no corresponding referenced tuple exists in DEPARTMENT with Dnumber = 7.

■ Operation:

Insert <‘Cecilia’, ‘F’, ‘Kolonsky’, ‘677678989’, ‘1960-04-05’, ‘6357 Windy Lane, Katy, TX’, F, 28000, NULL, 4> into EMPLOYEE.

Result: This insertion satisfies all constraints, so it is acceptable.

3.3.2 The Delete Operation

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

The **Delete** operation **can** violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database.

Examples:

■ *Operation:*

Delete the WORKS_ON tuple with Essn = '999887777' and Pno = 10.

Result: This deletion is acceptable and deletes exactly one tuple.

■ *Operation:*

Delete the EMPLOYEE tuple with Ssn = '999887777'

Result: This deletion is not acceptable, because there are tuples in WORKS_ON that refer to this tuple. Hence, if the tuple in EMPLOYEE is deleted, referential integrity violations will result.

■ *Operation:*

Delete the EMPLOYEE tuple with Ssn = '333445555'.

Result: This deletion will result in even worse referential integrity violations, because the tuple involved is referenced by tuples from the EMPLOYEE, DEPARTMENT, WORKS_ON, and DEPENDENT relations.

Several options are available if a deletion operation causes a violation.

The first option, called **restrict**, is to *reject the deletion*.

The second option, called **cascade**, is to *attempt to cascade (or propagate) the deletion* by deleting tuples that reference the tuple that is being deleted.

For example, in operation 2, the DBMS could automatically delete the offending tuples from WORKS_ON with Essn = '999887777'.

A third option, called **set null** or **set default**, is to *modify the referencing attribute values* that cause the violation; each such value is either set to NULL or changed to reference another default valid tuple.

3.3.3 The Update Operation

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

The **Update** (or **Modify**) operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation R . It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified.

Here are some examples.

■ *Operation:*

Update the salary of the EMPLOYEE tuple with Ssn = '999887777' to 28000.

Result: Acceptable.

■ *Operation:*

Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 1.

Result: Acceptable.

■ *Operation:*

Update the Dno of the EMPLOYEE tuple with Ssn = '999887777' to 7.

Result: Unacceptable, because it violates referential integrity.

■ *Operation:*

Update the Ssn of the EMPLOYEE tuple with Ssn = '999887777' to '987654321'.

Result: Unacceptable, because it violates primary key constraint by repeating a value that already exists as a primary key in another tuple; it violates referential integrity constraints because there are other relations that refer to the existing value of Ssn.

Updating an attribute that is neither part of a primary key nor of a foreign key usually causes no problems.

The DBMS need only check to confirm that the new value is of the correct data type and domain. Modifying a primary key value is similar to deleting one tuple and inserting another in its place because we use the primary key to identify tuples.

If a foreign key attribute is modified, the DBMS must make sure that the new value refers to an existing tuple in the referenced relation.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

3.3.4 The Transaction Concept

A transaction is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database.

At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

UNIT – 4 : SQL: Schema Definition, Basic Constraints, and Queries

4.1 SQL Data Definition and Data Types

4.2 Specifying Basic Constraints in SQL

4.3 Schema Change Statements in SQL

4.4 Basic Queries in SQL

4.5 More Complex SQL Queries

4.6 Insert, Delete, and Update Statements in SQL

4.7 Specifying General Constraints as Assertions

4.8 Views (Virtual Tables) in SQL

Introduction:

SQL is a standard language for accessing and manipulating databases.

SQL stands for Structured Query Language was designed and implemented at IBM Research created in late 70's, under the name of SEQUEL

SQL is an ANSI (American National Standards Institute) standard

What Can SQL do?

SQL can execute queries against a database

SQL can retrieve, insert, update, and delete data from a database

- SQL Consists of
 - A Data Definition Language (DDL) for declaring database schemas
The SQL commands for data definition are CREATE, ALTER, and DROP.
 - Data Manipulation Language (DML) for modifying and querying database instances
 - Data Control Language (DCL) used for grant and revoke of permissions by database administrators

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

4.1 SQL DATA DEFINITION AND DATA TYPES

4.1.1 Data Definition, Constraints, and Schema Changes in SQL

- In SQL- relation, tuple, and attribute are called table, row, and columns respectively.

The main SQL commands for data definition is the CREATE statement, which can be used to create schemas, tables (relations), and domains (as well as other constructs such as views, assertions, and triggers)

The basic data types available for attributes include numeric, character string, bit string, Boolean, date, and time.

- **Numeric:** Numeric data types include integer numbers of various sizes (integer or int, and smallint) and floating-point (real) numbers of various precision (float or real, and double precision).
- **Character-String** data types are either fixed length--CHAR(n) or CHARACTER(n), where n is the number of characters-or varying length-VARCHAR(n), where n is the maximum number of characters
- **Bit-String** data types are either of fixed length n-BIT(n)-or varying length-BIT VARYING(n), where n is the maximum number of bits.
- **Boolean** A boolean data type has the traditional values of TRUE or FALSE.
- **DATE** data type has ten positions, and its components are YEAR, MONTH, and DAY in the form YYYY-MM-DD.
- **TIME** data type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form HH:MM:SS.

4.1.2 The CREATE TABLE Command in SQL

The CREATE TABLE command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints. The attributes are specified first, and each attribute is given a name, a data type to specify its domain of values, and any attribute constraints, such as NOT NULL. The key, entity integrity, and referential integrity constraints can be specified within the CREATE TABLE statement. Syntax of create command as follows:

```
CREATE TABLE TABLENAME (COLUMN1 DATATYPE, COLUMN2 DATATYPE .... COLUMNn DATATYPE);
```

Ex: CREATE TABLE EMPLOYEE(FNAME VARCHAR(15), LNAME VARCHAR(15));

Usage of Data types using create command

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

```

CREATE TABLE EMPLOYEE
( FNAME          VARCHAR(15)      NOT NULL ,
  MINIT          CHAR           ,
  LNAME          VARCHAR(15)      NOT NULL ,
  SSN            CHAR(9)         NOT NULL ,
  BDATE          DATE           ,
  ADDRESS        VARCHAR(30)    ,
  SEX            CHAR           ,
  SALARY          DECIMAL(10,2) ,
  SUPERSSN       CHAR(9)        ,
  DNO            INT             NOT NULL ,
  PRIMARY KEY (SSN) ,
  FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN) ,
  FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER) );

CREATE TABLE DEPARTMENT
( DNAME          VARCHAR(15)      NOT NULL ,
  DNUMBER        INT             NOT NULL ,
  MGRSSN         CHAR(9)         NOT NULL ,
  MGRSTARTDATE   DATE           ,
  PRIMARY KEY (DNUMBER) ,
  UNIQUE (DNAME) ,
  FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN) );

CREATE TABLE DEPT_LOCATIONS
( DNUMBER        INT             NOT NULL ,
  DLOCATION        VARCHAR(15)     NOT NULL ,
  PRIMARY KEY (DNUMBER, DLOCATION) ,
  FOREIGN KEY (DNUMBER) REFERENCES DEPARTMENT(DNUMBER) );

CREATE TABLE PROJECT
( PNAME          VARCHAR(15)      NOT NULL ,
  PNUMBER        INT             NOT NULL ,
  PLOCATION        VARCHAR(15)    ,
  DNUM           INT             NOT NULL ,
  PRIMARY KEY (PNUMBER) ,
  UNIQUE (PNAME) ,
  FOREIGN KEY (DNUM) REFERENCES DEPARTMENT(DNUMBER) );

CREATE TABLE WORKS_ON
( ESSN           CHAR(9)         NOT NULL ,
  PNO            INT             NOT NULL ,
  HOURS          DECIMAL(3,1)    NOT NULL ,
  PRIMARY KEY (ESSN, PNO) ,
  FOREIGN KEY (ESSN) REFERENCES EMPLOYEE(SSN) ,
  FOREIGN KEY (PNO) REFERENCES PROJECT(PNUMBER) );

CREATE TABLE DEPENDENT
( ESSN           CHAR(9)         NOT NULL ,
  DEPENDENT_NAME VARCHAR(15)     NOT NULL ,
  SEX            CHAR           ,
  BDATE          DATE           ,
  RELATIONSHIP   VARCHAR(8)     ,
  PRIMARY KEY (ESSN, DEPENDENT_NAME) ,
  FOREIGN KEY (ESSN) REFERENCES EMPLOYEE(SSN) );

```

Figure 4.1: SQL CREATE TABLE data definition statements for defining the COMPANY schema

4.2 Specifying Basic Constraints in SQL

The term data integrity simply means that the data stored in the table is valid.

There are different types of data integrity, often referred to as constraints.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

4.2.1 NOT NULL Constraint

- A NOT NULL constraint means that a data row must have a value for the column specified as NOT NULL.
- A fairly standard practice is to assign each constraint a unique constraint name.
- If constraints are not named, then database system assigns meaningless system-generated names to each constraint.

Example

```
lname VARCHAR2(15) CONSTRAINT nn_emp_last_name NOT NULL,  
fname VARCHAR2(15) CONSTRAINT nn_emp_first_name NOT NULL,
```

4.2.2 PRIMARY KEY Constraint

- Each table must normally contain a column or set of columns that uniquely identifies rows of data that are stored in the table. This column or set of columns is referred to as the primary key.

Ex: ssn CHAR(9) CONSTRAINT pk_employee PRIMARY KEY,

- If a table requires two or more columns in order to identify each row, the primary key is termed a composite primary key (refer figure 4.1 **PRIMARY KEY**(ESSN, PNO) ,in works on table).

4.2.3 CHECK Constraint

- Sometimes the data values stored in a specific column must fall within some acceptable range of values. A CHECK constraint is used to enforce this data limit.

Ex: Salary DECIMAL(10,2)CONSTRAINT ck_emp_salary CHECK(emp_salary<= 4000)

4.2.4 UNIQUE Constraint

- Sometimes it is necessary to enforce uniqueness for a column value that is not primary key column.
- The **UNIQUE** constraint can be used to enforce this rule and database system rejects any rows that violate the constraint.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Assume that each parking space for the organization is numbered and that no two employees can be assigned same parking space.

dname varchar(15) CONSTRAINT un_dept_dname UNIQUE,

4.2.5 Foreign Keys Constraint

- Foreign keys (FKs) are columns in one table that reference primary key (PK) values in another or in the same table.
- Let's relate employee rows to the PK column named dnumber in the department table.
- The value stored to the dno column for any given employee row must match a value stored in the dnumber column in the department table.
- FOREIGN KEY constraints are also referred to as referential integrity constraints, and assist in maintaining database integrity.
- Referential integrity stipulates that values of a foreign key must correspond to values of a primary key in the table that it references.
- Several methods exist to ensure that referential integrity is maintained, ON DELETE SET NULL clause can be used to specify that the value of EMPLOYEE table dno column should be set to null if the referenced department row is deleted in DEPARTMENT table.

Specifying Foreign Key Constraint

```
CREATE TABLE employee (  
  emp_ssn CHAR(9) CONSTRAINT pk_employee PRIMARY KEY,  
  emp_last_name VARCHAR2(25) CONSTRAINT nn_emp_last_name NOT NULL,  
  emp_first_name VARCHAR2(25) CONSTRAINT nn_emp_first_name NOT NULL,  
  emp_date_of_birth DATE,  
  emp_salary NUMBER(7,2) CONSTRAINT ck_emp_salary CHECK (emp_salary <= 85000),  
  emp_parking_space NUMBER(4) CONSTRAINT un_emp_parking_space UNIQUE,  
  emp_gender CHAR(1),
```

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

```
emp_dpt_number NUMBER(2),  
CONSTRAINT fk_emp_dpt FOREIGN KEY (emp_dpt_number)  
REFERENCES department ON DELETE SET NULL  
);
```

4.3 Schema Change Statements in SQL

The commands which can be used to alter a schema by adding or dropping tables, attributes, constraints, and other schema elements are called schema change statements like ALTER, DROP, RENAME.

4.3.1 Dropping a Table

- Employee table can be deleted with the DROP TABLE command.
- This command deletes both the table structure, its data, related constraints, and indexes.

Ex: DROP TABLE employee;

4.3.2 Renaming a Table

- A table can be renamed with the RENAME command.
- This command does not affect table structure or data; it simply gives the current table a new name.

Ex: RENAME employee TO worker;

4.3.3 Alter TABLE Command

- Modifying existing tables to either add new columns or alter existing columns can be accomplished with the ALTER TABLE MODIFY and ALTER TABLE ADD commands.
- The current data type of the emp_parking_space column is NUMBER(4). A very large organization may have in excess of 9,999 employees.
- The ALTER TABLE command can be used to modify the emp_parking_space column to enable the allocation of upto 99,999 parking spaces.

ALTER TABLE employee MODIFY (emp_parking_space NUMBER(5));

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- If a column modification will result in a column that is smaller than was originally specified, Database system will return an error message if data rows exist such that their data will not fit into the new specified column size.
- The ALTER TABLE command can be used to add a new column to the employee table.
- Suppose that an organization recognizes the need to email_id of employees, an email_id column can be added to the employee table.

```
ALTER TABLE employee ADD (email_id VARCHAR(50));
```

4.4 Basic Queries in SQL

4.4.1 The SELECT-FROM-WHERE Structure

SQL has one basic statement for retrieving information from a database: the SELECT statement.

The syntax of this command is:

```
SELECT <attribute list> FROM <table list> WHERE <Condition>;
```

Query 1: Retrieve the birthday and address of the employee(s) whose name is 'Ram S V'

```
SELECT BDATE, ADDRESS
```

```
FROM EMPLOYEE
```

```
WHERE FNAME = 'Ram ' AND MINIT = 'S ' AND LNAME = 'V ' ;
```

Query 2: Retrieve the name and address of all employee who work for the 'Research' Dept.

```
SELECT FNAME, LNAME, ADDRESS
```

```
FROM EMPLOYEE, DEPARTMENT
```

```
WHERE DNAME = 'Research' AND DNUMBER = DNO;
```

Query 3: For every project located in 'Bangalore', list the project number, the controlling department number, and the department manager's last name, address, and birthdate.

```
SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
```

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

FROM PROJECT, DEPARTMENT, EMPLOYEE

WHERE DNUM=DNUMBER AND MGRSSN=SSN AND PLOCATION = 'Bangalore';

4.4.2 Dealing with Ambiguous Attribute Names and Renaming (Aliasing)

Ambiguity is in the case where attributes are same name need to qualify the attribute using DOT separator

e.g., WHERE DEPARTMENT.DNUMBER=EMPLOYEE.DNUMBER;

More Ambiguity in the case of queries that refer to the same relation twice .

Query 4: For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
```

```
FROM EMPLOYEE AS E, EMPLOYEE AS S
```

```
WHERE E.SUPERSSN=S.SSN;
```

4.4.3 Unspecified WHERE Clause and Use of Asterisk (*)

A missing WHERE clause indicates no conditions, which means all tuples are selected in case of two or more tables, then all possible tuple combinations are selected.

Query 5: Select all EMPLOYEE SSNs , and all combinations of EMPLOYEE SSN and DEPARTMENT DNAME.

```
SELECT SSN, DNAME
```

```
FROM EMPLOYEE, DEPARTMENT;
```

Retrieve all employees working for Dept. 5

```
SELECT * FROM EMPLOYEE WHERE DNO=5;
```

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

4.4.4 Tables as Sets in SQL

SQL treats table as a **multiset**, which means duplicate tuples are OK, SQL does not delete duplicate so to eliminate duplicate, have to use DISTINCT.

Query 6: Retrieve the salary of every employee.

```
SELECT ALL SALARY FROM EMPLOYEE;
```

Query 7: Retrieve all distinct salary values.

```
SELECT DISTINCT SALARY FROM EMPLOYEE;
```

4.4.5 Substring Comparisons, Arithmetic Operations, and Ordering

SQL allows comparison conditions on only parts of a character string, using the LIKE comparison operator for string pattern matching.

Partial strings are specified using following reserved characters:

- % replaces an arbitrary number of zero or more characters,
- Underscore _ replaces a single character.
- || concatenate operation for strings
- Partial strings are specified by using '
- If an underscore or '%' is needed as a literal character in the string, the character should be preceded by an escape character '\', which is specified after the string using the keyword ESCAPE.

For example, 'AB_CD\%EF' ESCAPE '\' represents the literal string.

Query 8: Retrieve all employees whose address is in Bangalore , Karnataka.

```
SELECT FNAME, LNAME  
FROM EMPLOYEE  
WHERE ADDRESS LIKE '% Bangalore, Karnataka %';
```

Query 9: In order to list all employee who were born during 1960s we have the followings:

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

```
SELECT FNAME, LNAME  
FROM EMPLOYEE  
WHERE BDATE LIKE ' __6____';
```

SQL also supports addition, subtraction, multiplication and division (denoted by +, -, *, and /, respectively) on numeric values or attributes with numeric domains.

Query 10: Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

```
SELECT FNAME, LNAME, 1.1*SALARY  
FROM EMPLOYEE, WORKS_ON, PROJECT  
WHERE SSN=ESSN AND PNO=PNUMBER AND PNAME='ProductX';
```

Query 11: Retrieve all employees in department number 5 whose salary between \$30000 and \$40000.

```
SELECT *  
FROM EMPLOYEE  
WHERE (SALARY BETWEEN 30000 AND 40000) AND DNO=5;
```

4.4.6 Ordering Of Tuples

Ordering the tuples in the result of a query is possible using **ORDER BY** clause.

Query12 : Retrieve all employees working for project ordered by dname,lname and fname.

```
SELECT DNAME, LNAME, FNAME, PNAME  
FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT  
WHERE DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER
```

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

ORDER BY DNAME, LNAME, FNAME;

The default order is in **ascending order**, but user can specify ORDER BY DNAME DESC, LNAME ASC, FNAME, ASC;

```
SELECT DNAME, LNAME, FNAME, PNAME
FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
WHERE DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER
ORDER BY FNAME DESC;
```

4.5 More Complex SQL Queries

Complex SQL queries can be formulated by composing nested SELECT/FROM/WHERE clauses within the WHERE clause of another query

Query 13 : Make a list of Project numbers for projects that involve an employee whose last name is 'Sharma', either as a worker or as a manger of the department that controls the project .

```
SELECT DISTINCT PNUMBER
FROM PROJECT
WHERE PNUMBER IN (SELECT PNUMBER
                   FROM PROJECT, DEPARTMENT, EMPLOYEE
                   WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME='sharma' OR PNUMBER IN
                   (SELECT PNO
                    FROM WORKS_ON, EMPLOYEE
                    WHERE ESSN=SSN AND LNAME='Sharma '))
```

4.5.1 IN operator and set of union compatible tuples

The comparison operator **IN** compares a value **v** with a set (or multiset) of values **V** and evaluates to TRUE if **v** is one of the elements in **V**.

Query14: Retrieve the social security numbers of all employees who work on the same (project, hours) combination on some project that employee 'Ram Sharma' (whose SSN =

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

'123456789') works on.

```
SELECT DISTINCT ESSN  
FROM WORKS_ON  
WHERE (PNO, HOURS) IN (SELECT PNO, HOURS FROM WORKS_ON WHERE SSN='123456789');
```

4.5.2 The keyword ALL

In addition to the IN operator, a number of other comparison operators can be used to compare a single value **v** to a set of multiset **V**. **ALL V** returns TRUE if **v** is greater than all the value in the set

Query 15: Retrieve the name of employees whose salary is greater than the salary of all the employees in department number 5.

```
SELECT LNAME, FNAME  
FROM EMPLOYEE  
WHERE SALARY > ALL (SELECT SALARY  
                     FROM EMPLOYEE  
                     WHERE DNO=5);
```

4.5.3 Ambiguity in nested query can be resolved as follows.

Since the same name can be used for two (or more) attributes in different relations it leads to ambiguity if both relations are referred in same query. To prevent ambiguity, prefix the relation name to the attribute name and separating the two by a period (.).

Query16: Retrieve the name of employees and their dependents.

```
SELECT E.FNAME, E.LNAME  
FROM EMPLOYEE AS E  
WHERE E.SSN IN (SELECT ESSN  
               FROM DEPENDENT  
               WHERE ESSN=E.SSN AND E.FNAME=DEPENDENTNAME AND  
               SEX=E.SEX);
```

4.5.4 Correlated Nested Query

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

The two queries are said to be correlated if WHERE clause of a nested query references some attributes of a relation declared in the outer query. The result of a correlated nested query is different for each tuple (or combination of tuples) of the relation(s) of outer query.

In general, any nested query involving the = or comparison operator IN can always be rewritten as a single block query

Query 17: For each EMPLOYEE tuple, evaluate the nested query, which retrieves the ESSN values for all DEPENDENT tuples with the same sex and name as that EMPLOYEE tuple; if the SSN value of the EMPLOYEE tuple is in the result of the nested query, then select that EMPLOYEE tuple.

```
SELECT E.FNAME, E.LNAME  
FROM EMPLOYEE E, DEPENDENT D  
WHERE E.SSN=D.ESSN AND E.SEX=D.SEX AND E.FNAME =D.DEPENDENTNAME;
```

Query 18: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT E.FNAME, E.LNAME  
FROM EMPLOYEE AS E  
WHERE E.SSN IN (SELECT ESSN  
FROM DEPENDENT  
WHERE ESSN=E.SSN AND E.FNAME=DEPENDENTNAME)
```

In Query 17 and 18 the nested query has a different result for each tuple in the outer query.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

The **CONTAINS** operator compares two sets of values , and returns TRUE if one set contains all values in the other set.

Query 19: Retrieve the name of each employee who works on all the projects controlled by department number 5.

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE ( (SELECT PNO
        FROM WORKS_ON
        WHERE SSN=ESSN) CONTAINS (SELECT PNUMBER
        FROM PROJECT WHERE DNUM=5));
```

In Query 19 the second nested query, which is not correlated with the outer query, retrieves the project numbers of all projects controlled by department 5. The first nested query, which is correlated, retrieves the project numbers on which the employee works, which is different for each employee tuple because of the correlation.

4.5.5 THE EXISTS AND UNIQUE FUNCTIONS IN SQL

EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not We can formulate Query 19 in an alternative form that uses EXISTS as Query 19b below.

Query 19b: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE E
WHERE EXISTS (SELECT *
              FROM DEPENDENT
              WHERE E.SSN=ESSN AND SEX=E.SEX AND
              E.FNAME=DEPENDENTNAME);
```

Query 20: Retrieve the names of employees who have no dependents.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

```
SELECT FNAME, LNAME
FROM EMPLOYEE
WHERE NOT EXISTS (SELECT *
                  FROM DEPENDENT
                  WHERE SSN=ESSN);
```

In Query 20, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If none exist, the EMPLOYEE tuple is selected

4.5.6 EXPLICIT SETS AND NULLS IN SQL

It is also possible to use an explicit (enumerated) set of values in the WHERE clause rather than a nested query .

Query 21: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

```
SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE PNO IN (1,2,3);
```

Null Value

SQL has various rules for dealing with NULL values. NULL is used to represent a missing value, but that it usually has one of three different interpretations-

- **Unknown value** (exists but is not known): A particular person has a date of birth but it is not known, so it is represented by NULL in the database.
- **Unavailable or withheld value** (exists but is purposely withheld): A person has a home phone but does not want it to be listed, so it is withheld and represented as NULL in the database.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- **Not applicable attribute** (undefined for this tuple): An attribute LastCollegeDegree would be NULL for a person who has no college degrees, because it does not apply to that person.

SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate.

Query 22: Retrieve the names of all employees who do not have supervisors

```
SELECT FNAME, LNAME  
FROM EMPLOYEE  
WHERE SUPERSSN IS NULL;
```

(**Note:** If a join condition is specified, tuples with NULL values for the join attributes are not included in the result).

4.5.7 Join operation on tables in SQL

A **sql join** is a **means** for combining columns from one (self-table) or more tables by using values common to each.

Query 23: Retrieves the name and address of every employee who works for the 'Research' department using join operation.

```
SELECT FNAME, LNAME, ADDRESS  
FROM (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)  
WHERE DNAME='Research';
```

The FROM clause in Query 23 contains a single joined table. The attributes of such a table are all the attributes of the first table, EMPLOYEE, followed by all the attributes of the second table, DEPARTMENT.

The concept of a join on table also allows the user to specify different types of join as follows,

- NATURAL JOIN
- OUTER JOIN

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- INNER JOIN

A **Natural Join** is a form of Inner Join where the join is implicitly across all columns of matching names on both sides of the join.

The DEPARTMENT relation is renamed as DEPT and its attributes are renamed as DNAME, DNO (to match the name of the desired join attribute DNO in EMPLOYEE), MSSN, and MSDATE. The implied join condition for this NATURAL JOIN is EMPLOYEE.DNO = DEPT.DNO, because this is the only pair of attributes with the same name after renaming.

Query 24:

```
SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE NATURAL JOIN (DEPARTMENT AS DEPT (DNAME, DNO,
MSSN, MSDATE)))
WHERE DNAME='Research;
```

The **default** type of join in a joined table is an **inner join**, where a tuple is included in the result only if a matching tuple exists in the other relation.

OUTER JOIN : Return rows when there is at least one match in both tables .It is of three types as follows

- **LEFT OUTER JOIN**: Return all rows from the left table, even if there are no matches in the right table

In Query 4, only employees that have a supervisor are included in the result; an EMPLOYEE tuple whose value for SUPERSSN is NULL is excluded .If the user requires that all employees be included, an OUTER JOIN must be used explicitly specified

Query 25:

```
SELECT E.LNAME AS EMPLOYEE_NAME, S.LNAME AS SUPERVISOR_NAME
FROM (EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S ON E.SUPERSSN=S.SSN);
```

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

A **right outer join** (or **right join**) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those rows that have no match in B.

For example, this allows us to find each employee and his or her department, but still show departments that have no employees.

Query 26:

```
SELECT *  
FROM employee RIGHT OUTER JOIN department ON  
employee.Dno = department.Dnumber;
```

A **FULL OUTER JOIN** combines the effect of applying both left and right outer joins. Where rows in the FULL OUTER JOINed tables do not match, the result set will have NULL values for every column of the table that lacks a matching row. For those rows that do match, a single row will be produced in the result set (containing columns populated from both tables).

For example, this allows us to see each employee who is in a department and each department that has an employee, but also see each employee who is not part of a department and each department which doesn't have an employee.

Query 27:

```
SELECT *  
FROM employee FULL OUTER JOIN department ON employee.Dno = department.Dnumber;
```

INNER JOIN

An **Inner join** requires each row in the two joined tables to have matching rows. Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied by matching non-NULL values, column values for each matched pair of rows of A and B are combined into a result row.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

The "explicit join notation" uses the JOIN keyword, optionally preceded by the INNER keyword, to specify the table to join, and the ON keyword to specify the predicates for the join, as in the following example:

Query 28:

```
SELECT employee.LastName, employee.DepartmentID, department.DepartmentName
FROM employee INNER JOIN department ON
employee.Dno = department.Dnumber;
```

Query 29: The following example is equivalent to the previous one, but this time using implicit join notation

```
SELECT *
FROM employee, department
WHERE employee.Dno = department.Dnumber;
```

An **EQUI-JOIN** is a specific type of comparator-based join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equi-join. The query shown above has already provided an example of an equi-join:

Query 30:

```
SELECT *
FROM employee JOIN department ON employee.DNO = department.Dnumber;
```

4.5.7 Aggregate Functions in SQL

An **aggregate function** is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning. Built-in aggregate functions include Average(), Count(), Maximum(), Minimum(), Sum().

The **COUNT** function returns the number of tuples or values as specified in a query.

Retrieve the total number of employees in the company (Query 31) and the number of employees in the 'Research' Department (Query 32).

Query 31:

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

```
SELECT COUNT (*) FROM EMPLOYEE;
```

Query 32:

```
SELECT COUNT (*)  
FROM EMPLOYEE,DEPARTMENT  
WHERE DNO=DNUMBER AND DNAME='Research';
```

The functions **SUM**, **MAX**, **MIN**, and **AVG** are applied to a set or multiset of numeric values and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values. The functions **MAX** and **MIN** can also be used with attributes that have nonnumeric domains if the domain values have a total ordering among one another

Query 33: Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary.

```
SELECT SUM (SALARY), MAX (SALARY), MIN (SALARY),AVG (SALARY)  
FROM EMPLOYEE;
```

4.5.8 Grouping: The GROUP BY and HAVING Clauses

The **GROUP BY** statement is used in conjunction with the aggregate functions to group the result-set by one or more columns and also to eliminate duplicate rows from a result set. The **WHERE** clause is applied before the **GROUP BY** clause.

Query 34: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
SELECT DNO, COUNT (*), AVG (SALARY)  
FROM EMPLOYEE  
GROUP BY DNO;
```

In Query 34, the **EMPLOYEE** tuples are partitioned into groups-each group having the same value for the grouping attribute **DNO**. The **COUNT** and **AVG** functions are applied to each such group of tuples. Notice that the **SELECT** clause includes only the grouping attribute and the functions to be applied on each group of tuples.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

If NULLs exist in the grouping attribute, then a separate group is created for all tuples with a NULL value in the grouping attribute. For example, if the EMPLOYEE table had some tuples that had NULL for the grouping attribute DNO, there would be a separate group for those tuples in the result of Query 34

SQL provides a **HAVING** clause, which can appear in conjunction with a GROUP BY clause, for this purpose.

HAVING provides a condition on the group of tuples associated with each value of the grouping attributes. Only the groups that satisfy the condition are retrieved in the result of the query.

Query 35: For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

```
SELECT PNUMBER, PNAME, COUNT (*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT (*) > 2;
```

Note that, while selection conditions in the WHERE clause limit the tuples to which functions are applied, the HAVING clause serves to choose whole groups.

4.6 INSERT, DELETE, AND UPDATE STATEMENTS IN SQL

In SQL, three commands can be used to modify the database: **INSERT**, **DELETE**, and **UPDATE**.

INSERT: An SQL **INSERT** statement adds one or more records to any single table in a relational database. Insert statements have the following form:

```
INSERT INTO table (column1 [, column2, column3 ... ]) VALUES (value1 [, value2, value3 ... ]);
```

The number of columns and values must be the same. If a column is not specified, the default value for the column is used. The values specified (or implied) by the INSERT statement must

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

satisfy all the applicable constraints (such as primary keys, CHECK constraints, and NOT NULL constraints).

Example:

```
INSERT INTO phone_book (name, number) VALUES ('Ram Sharma', '9880994441');
```

Shorthand may also be used, taking advantage of the **order** of the columns when the table was created. It is not required to **specify all columns** in the table since any other columns will take their default value or remain null:

```
INSERT INTO table VALUES (value1, [value2, ... ])
```

Example for inserting data into 2 columns in the phone_book table and ignoring any other columns which may be after the first 2 in the table.

```
INSERT INTO phone_book VALUES ('Ram Sharma', '9880994441');
```

DELETE : The **DELETE** statement removes one or more records from a table. A subset may be defined for deletion using a condition, otherwise all records are removed

The DELETE statement follows the syntax:

```
DELETE FROM table_name [WHERE condition];
```

Any rows that match the WHERE condition will be removed from the table. If the WHERE clause is omitted, all rows in the table are removed. The DELETE statement does not return any rows; that is, it will not generate a result set.

Executing a DELETE statement can cause triggers to run that can cause deletes in other tables. For example, if two tables are linked by a foreign key and rows in the referenced table are deleted, then it is common that rows in the referencing table would also have to be deleted to maintain referential integrity.

Examples

1) Delete rows from table pies where the column flavour equals Lemon Meringue:

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

```
DELETE FROM pies WHERE flavour='Lemon Meringue';
```

- 2) Delete rows in trees, if the value of height is smaller than 80.

```
DELETE FROM trees WHERE height < 80;
```

- 3) Delete all rows from mytable:

```
DELETE FROM mytable;
```

- 4) Delete rows from mytable using a subquery in the where condition:

```
DELETE FROM mytable
WHERE id IN (
            SELECT id
            FROM mytable2 );
```

UPDATE :

An SQL **UPDATE** statement changes the data of one or more records in a table. Either all the rows can be updated, or a subset may be chosen using a condition.

The UPDATE statement has the following form:

```
UPDATE table_name SET column_name = value [, column_name = value ...] [WHERE
condition]
```

For the UPDATE to be successful, on the table or column ,the updated value must not conflict with all the applicable constraints (such as primary keys, unique indexes, CHECK constraints, and NOT NULL constraints).

Examples:

- 1) Change the location and controlling department number of project number 10 to 'Bellary' and 5, respectively.

```
UPDATE PROJECT
SET PLOCATION = ' Bellary ', DNUM = 5
```

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

WHERE PNUMBER=10;

- 2) Give all employees in the 'Research' department a 10 percent raise in salary.

```
UPDATE EMPLOYEE
SET SALARY = SALARY *1.1
WHERE DNO IN (SELECT DNUMBER
              FROM DEPARTMENT
              WHERE DNAME='Research');
```

4.7 SPECIFYING GENERAL CONSTRAINTS AS ASSERTIONS

General constraints are called as ASSERTIONS and these can be defined using the **CREATE ASSERTION** statement of the DDL. Each assertion is given a constraint name and is specified via a condition similar to the WHERE clause of an SQL query.

For example, to specify the constraint that "the salary of an employee must not be greater than the salary of the manager of the department that the employee works for" in SQL, we can write the following assertion:

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS (SELECT *
FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
WHERE E.SALARY>M.SALARY AND
E.DNO=:D.DNUMBER AND
D.MGRSSN=:M.SSN));
```

The constraint name SALARY_CONSTRAINT is followed by the keyword CHECK, which is followed by a condition in parentheses that must hold true on every database state for the assertion to be satisfied.

The constraint name can be used later to refer to the constraint or to modify or drop it. Whenever some tuples in the database cause the condition of an ASSERTION statement to evaluate to FALSE, the constraint is violated.

4.8 Views (Virtual Tables) in SQL

A view in SQL is a single table that is derived from other tables. These other tables could be base tables or previously defined views.

A view does not exist in physical form so it is considered a virtual table, in contrast to base tables, whose tuples are actually stored in the database.

Creation of Views in SQL:

In SQL, the command to create a view is **CREATE VIEW**. The view is given a (virtual) table name (or view name), a list of attribute names, and a query to specify the contents of the view.

Syntax

```
CREATE VIEW "VIEW_NAME" AS "SQL SELECT Statement";
```

Example

```
1) CREATE VIEW WORKS_ON1
AS SELECT FNAME, LNAME, PNAME, HOURS
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE SSN=ESSN AND PNO=PNUMBER;
```

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

```
2): CREATE VIEW DEPTINFO(DEPT_NAME,NO_OF_EMPS,TOTAL_SAL)
AS SELECT DNAME, COUNT (*), SUM (SALARY)
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE DNUMBER=DNO
GROUP BY DNAME;
```

The views created from the above commands

WORKS_ON1

| | | | |
|-------|-------|-------|-------|
| FNAME | LNAME | PNAME | HOURS |
|-------|-------|-------|-------|

DEPT_INFO

| | | |
|-----------|------------|-----------|
| DEPT_NAME | NO_OF_EMPS | TOTAL_SAL |
|-----------|------------|-----------|

If we do not need a view any more, we can use the **DROP VIEW** command to delete it. For example, to drop view WORKS_ON1, we can use the SQL statement:

```
DROP VIEW WORKS_ON1;
```

To **update** the PNAME attribute of 'Ram Sharma' from 'ProductX' to 'ProductY'. This view update is shown below:

```
UPDATE WORKS_ON1
```

```
SET PNAME = 'ProductY'
```

```
WHERE LNAME='Sharma' AND FNAME='Ram' AND PNAME='ProductX';
```

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

UNIT – 5 : Functional Dependencies And Normalization

- 5.1 Informal Design guidelines for relation schemas
- 5.2 Functional dependencies
- 5.3 Normal forms based on primary keys
- 5.4 General Definition of second and third normal forms
- 5.5 Boyce-codd Normal form.

5.1 Informal design guidelines for relation schemas

There are four informal measures of quality for relation schema design as follows:

- Semantics of the relation attributes
- Reducing the redundant values in tuples
- Reducing the null values in tuples
- Disallowing the possibility of generating spurious tuples (fake)

Semantics of the relation attributes

Whenever we group attributes to form a relation we assume that attributes belonging to one relation have certain meaning and proper interpretation associated with them.

The semantics specifies how to interpret the attribute values stored in a tuple of the relation.

Example:-consider company database schema. The various relations considered for this database are

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

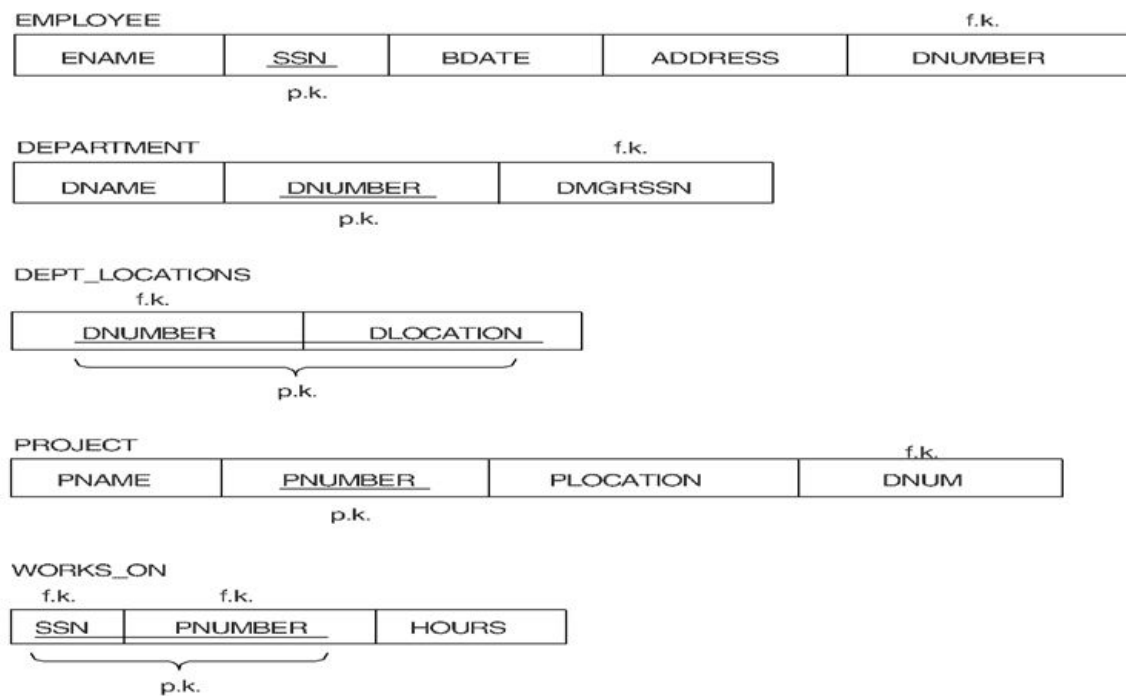


Figure 5.1 Simplified company relational database schema

The meaning of the employee relation is quite simple, each tuple represent an employee. The Dnumber attribute is a foreign key that represent an implicit relationship between EMPLOYEE and DEPARTEMENT relations.

Each department tuple represents a department entity, and each project tuple represents a project entity.

Guideline #1:-

Design a relation schema such that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation.

Redundant information in tuples and update anomalies:-

The goal of a schema design is to reducing redundant values in tuples, save storage space and avoid update anomalies.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

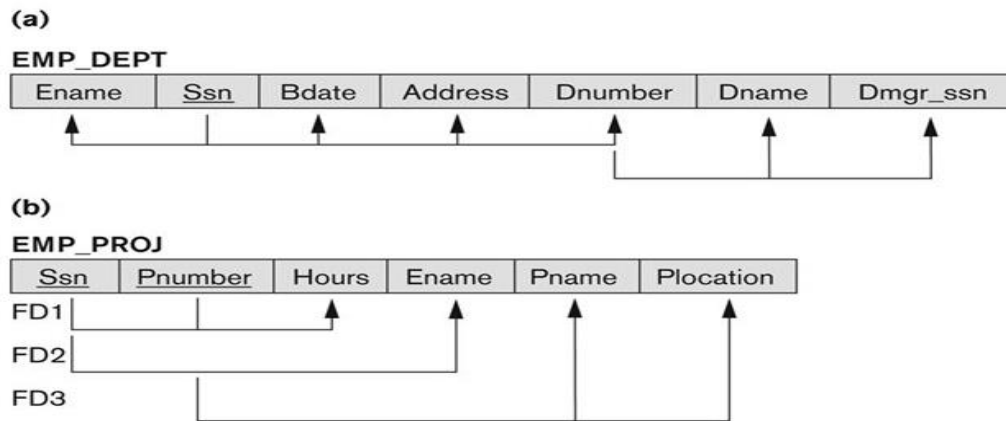


FIGURE 5.2 Two relation schemas suffering from update anomalies.

In figure 5.2(a) EMP_DEPT, the attribute values pertaining to a particular department are repeated for every employee who works for the department.

Another problem with using the relations is the problem of update anomalies.

Update anomalies can be classified into:

- Insertion anomalies
- Deletion anomalies
- Modification anomalies

Insertion anomalies

Insertion anomalies can be differentiated into two types :

- To insert a new employee into EMP_DEPT we must include either the attribute values for the department that the employee works for, or null.
Example: To insert a new tuple for an employee who works in department number 5, the attribute values of department 5 should be entered correctly so that they are consistent with values for department 5 in other tuples in EMP_DEPT.
- It is the difficult to insert a new department that has no employees as yet in EMP_DEPT relation.

To do this place null values in the attributes of EMPLOYEE relation. This cause problem because SSN is the primary key of EMP_DEPT, and each tuple is used to represent an employee entity.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

This problem does not occur when department is entered in the DEPARTMENT relation whether or not any employees works for it, and whenever an employee is assigned to that department, a corresponding tuples is inserted in EMPLOYEE.

Deletion anomalies

- If we are deleting the last employee tuple of a particular department from EMP_DEPT relation the information concerning that department is lost from the database. This problem does not occur in the database when department tuples are stored separately

Modification anomalies

- In EMP_DEPT, if we change the values of one of the attributes of a particular employee, we must update the tuples of the all employees who works in that department; otherwise, the database will become inconsistent.
- If we fail to update some tuples, the same department will be shown to have two different values, which would be wrong

Guideline#2:

Design a base relation schema such that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them and make sure that the programs that update the database will operate correctly.

Null values in tuples

Using null values can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes.

The nulls can have multiple interpretations, such as the following:

- The attribute does not apply to this tuple.
- The attribute value for this tuple is unknown.
- The value is known but absent; that is, it has not been recorded yet.

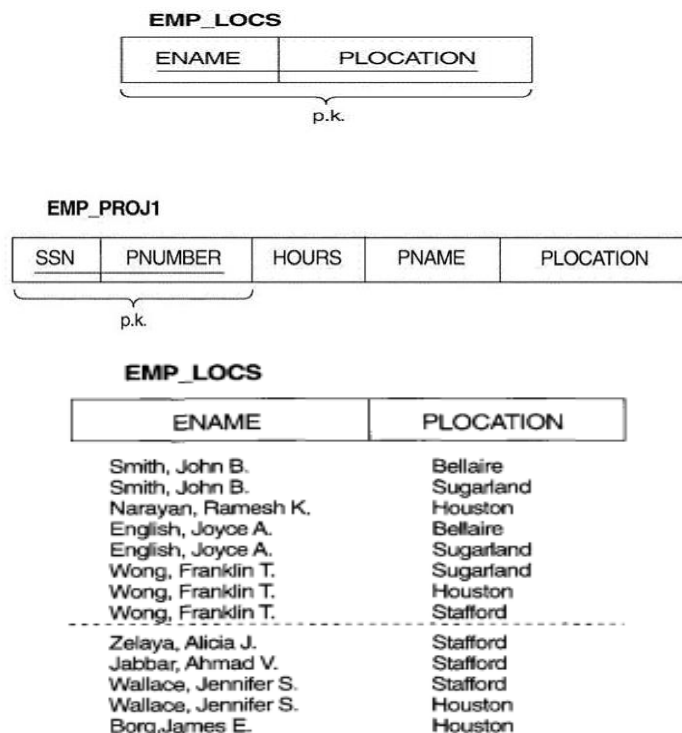
Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Guideline #3: Avoid placing attributes in a base relation whose values may frequently be null and nulls are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

Generation of spurious (fake) tuples

Design relational schema so that they can be joined with equality conditions are:-

- As shown in figure 5.3 we have used EMP_PROJ1 and EMP_LOCS as the base relations instead of EMP_PROJ.
- If we attempt NATURAL JOIN operation on EMP_PROJ1 and EMP_LOCS, the result produces many more tuples than the original set of tuples in EMP_PROJ. Additional tuples that were not in EMP_PROJ are called spurious tuples because they represent wrong information “that is not valid”
- Decomposing EMP_PROJ into EMP_PROJ1 and EMP_LOCS is undesirable because, when we join them back using natural join, we do not get the correct original information.



Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

| EMP_PROJ1 | | | | | |
|-----------|---------|-------|-----------------|-----------|--|
| SSN | PNUMBER | HOURS | PNAME | PLOCATION | |
| 123456789 | 1 | 32.5 | Product X | Bellaire | |
| 123456789 | 2 | 7.5 | Product Y | Sugarland | |
| 666884444 | 3 | 40.0 | Product Z | Houston | |
| 453453453 | 1 | 20.0 | Product X | Bellaire | |
| 453453453 | 2 | 20.0 | Product Y | Sugarland | |
| 333445555 | 2 | 10.0 | Product Y | Sugarland | |
| 333445555 | 3 | 10.0 | Product Z | Houston | |
| 333445555 | 10 | 10.0 | Computerization | Stafford | |
| 333445555 | 20 | 10.0 | Reorganization | Houston | |
| 999887777 | 30 | 30.0 | Newbenefits | Stafford | |
| 999887777 | 10 | 10.0 | Computerization | Stafford | |
| 987987987 | 10 | 35.0 | Computerization | Stafford | |
| 987987987 | 30 | 5.0 | Newbenefits | Stafford | |
| 987654321 | 30 | 20.0 | Newbenefits | Stafford | |
| 987654321 | 20 | 15.0 | Reorganization | Houston | |
| 888665555 | 20 | null | Reorganization | Houston | |

Figure 5.3 The two relation schemas EMP_LOCS and EMP_PROJ1 and the result of projecting the extension of EMP_PROJ from the relations EMP_LOCS and EMP_PROJ1.

This is because Plocation is the attribute that relates EMP_LOCS and EMP_PROJ1 and Plocation is either a primary key or a foreign key in either EMP_PROJ1 or EMP_LOCS.

Guideline #4: Design relation schemas so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated.

5.2 Functional Dependencies

A Functional Dependency (FD) is a constraint between two sets of attributes from the database. FDs are used to specify formal measures of the "goodness" of relational designs. FDs are derived from the real-world constraints on the attributes.

Definition:

- A functional dependency denoted by $fd: X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R .
- The constraint is that for any two tuples t_1 and t_2 in r that have $t_1[X] = t_2[X]$, they must also have $t_1[Y] = t_2[Y]$.

This means that the values of the Y component of a tuple in r depend on, or are determined by the values of the X components.

The values of the X component of a tuples uniquely determine the values of the Y component therefore there is a functional dependency from X to Y , or that Y is functionally dependent on X .

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

If $fd: X \rightarrow Y$ in R , this does not say whether or not $Y \rightarrow X$ in R .

A functional dependency is a property of the semantics or meaning of the attributes, i.e., a property of the relation schema. They must hold on all relation states (extensions) of R or Relation extensions $r(R)$.

A FD: $X \rightarrow Y$ is a **fully functional dependency** if removal of any attribute from X means that the dependency does not hold any more; otherwise, it is a partial functional dependency.

Example:-

a) $fd1: SSN \rightarrow Ename$

b) $fd2: Pnumber \rightarrow \{Pname, Plocation\}$

c) $fd3 : \{SSN, Pnumber\} \rightarrow hours$

The functional dependency specify that

- (a) The value of an employee SSN uniquely determines the employee name (E name).
- (b) The value of a P number uniquely determines the Pname and Plocation.
- (c) A combination of SSN and Pnumber values uniquely determine the no of hours the employee currently works on the project per week.

5.2.2 Inference rules for functional dependencies

The F denotes set of functional dependencies that are specified on relation schema R .

Example: If each department has one manager, so that $dept_no$ uniquely determines mgr_SSN ($dept_no \rightarrow mgr_ssn$) and a manager has a unique phone no called mgr_phone ($mgr_ssn \rightarrow mgr_phno$), then these two dependencies together imply that $dept_no \rightarrow mgr_phno$.

Definition:

- Formally, the set of all dependencies that include F as well as all dependencies that can be inferred from F is called the closure of F ; it is denoted by F^+ .
- Suppose $F = \{SSN \rightarrow \{Ename, Bdate, address, Dnumber\}, Dnumber \rightarrow \{d_name, mgrssn\}\}$

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Some of the additional dependencies that can be inferred from F are the following

$$SSN \rightarrow \{Dname, Mgr_SSN\}$$

$$SSN \rightarrow SSN$$

$$DNumber \rightarrow DNAME$$

The closure F^+ of F is the set of all functional dependencies that can be inferred from F. The notation $F \models X \rightarrow Y$ to denote that the functional dependency $X \rightarrow Y$ is inferred from the set of functional dependencies F.

The following 6 rules IR1 through IR6 are well known inference rules for functional dependencies.

IR1 (reflexive rule): If $X \supseteq Y$, then $X \rightarrow Y$.

IR2 (augmentation rule): $\{X \rightarrow Y\} \models XZ \rightarrow YZ$.

IR3 (transitive rule): $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$.

IR4 (decomposition, or projective, rule): $\{X \rightarrow YZ\} \models X \rightarrow Y$.

IR5 (union, or additive, rule): $\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$.

IR6 (pseudotransitive rule): $\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$.

The reflexive rule (IR1) states that set of attributes always determines itself or any of its subsets because IR1 generates dependencies that are always true, such dependencies are called trivial.

IR2 augmentation rule says that adding the same set of attributes to both the left hand side and right hand side of a dependency results in another valid dependency.

According to IR3 functional dependency is transitive. The decomposition rule IR4 says that we can remove attributes from right hand side of a dependency; applying this rule repeatedly can decompose the FD. $X \rightarrow \{A_1, A_2, \dots, A_n\}$ into the set of dependencies $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$.

The union rule IR5 allows us to do the opposite we can combine a set of dependencies $\{X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n\}$ into a single FD $X \rightarrow \{A_1, A_2, \dots, A_n\}$.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Proof of IR1

Suppose that $X \supseteq Y$ and that two tuples t_1 and t_2 exist in some relation instance r of R such that $t_1[X]=t_2[X]$, then $t_1[Y]=t_2[Y]$ because $X \supseteq Y$; hence $X \rightarrow Y$ must hold in r .

Proof of IR2 (By contradiction)

Assume that $X \rightarrow Y$ holds in relation instance r of R but that $XZ \rightarrow YZ$ does not hold.

Then there must exist two tuples t_1 and t_2 in r such that

1. $t_1[X]=t_2[X]$.
2. $t_1[Y]=t_2[Y]$.
3. $t_1[XZ]=t_2[XZ]$.
4. $t_1[YZ] \neq t_2[YZ]$.

This is not possible because from (1) and (3) we deduce

5. $t_1[Z]=t_2[Z]$ and from (2) and (5) we deduce .

6. $t_1[YZ]=t_2[YZ]$, contradicting (4) we get $t_1[YZ] \neq t_2[YZ]$.

Proof of IR3

Assume that

(1) $X \rightarrow Y$ and (2) $Y \rightarrow Z$ both hold in a relation r .

Then for any two tuples t_1 and t_2 in r such that $t_1[X] = t_2[X]$, we must also have

(3) $t_1[Y]=t_2[Y]$, from assumption (1); hence we must also have

(4) $t_1[Z]=t_2[Z]$ from (3) and assumption (2); hence $X \rightarrow Z$ must hold in r .

We can prove the inference rules IR4 to IR6 and any additional valid inference rules we can prove IR4 through IR6 by using IR1 through IR3 as follows.

Proof of IR4 (using IR1 through IR3):-

1. $X \rightarrow YZ$ (given).
2. $YZ \rightarrow Y$ (using IR1 and knowing $YZ \supseteq Y$).

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

3. $X \rightarrow Y$ (using IR3 on 1 and 2).

Proof of IR5 (using IR1 through IR3)

1. $X \rightarrow Y$ (given).
2. $X \rightarrow Z$ (given).
3. $X \rightarrow XY$ (using IR2 on 1 by augmenting with X; notice that $XX=X$).
4. $XY \rightarrow YZ$ (using IR2 on 2 by augmenting with Y).
5. $X \rightarrow YZ$ (using IR3 on 3 and 4).

Proof of IR6 (using IR1 through IR3)

1. $X \rightarrow Y$ (given).
2. $WY \rightarrow Z$ (given).
3. $WX \rightarrow WY$ (using IR2 on 1 augmenting with W).
4. $WX \rightarrow Z$ (using IR3 on 3 and 2).

Armstrong inference rules IR1 through IR3 are **sound and complete**. By **sound**, we mean that given a set of functional dependencies F specified on a relation schema R , any dependency that we can infer from F by using IR1 through IR3 in every relation state r of R that satisfies the dependencies in F . By **completing**, we can mean that using IR1 through IR3 repeatedly to infer dependencies until no more dependencies can be inferred results in the complete set of all possible dependencies that can be inferred from F . The set of dependencies F^+ , which is called the closure of F , can be determine from F by using only inference rules IR1 through IR3 are known as Armstrong inference rules.

Closure of a Set of Functional Dependencies given a set X of FDs in relation R , the set of all FDs that are implied by X is called the closure of X , and is denoted X^+ .

Algorithms for determining X^+ , the closure of X under F

```

 $X^+ := X$ ;
repeat
    old $X^+ := X^+$ ;
    for each functional dependency  $Y \rightarrow Z$  in  $F$  do
        if  $X^+ \supseteq Y$  then  $X^+ := X^+ \cup Z$ ;
until ( $X^+ = \text{old}X^+$ );

```

Example: The following set F of functional dependencies that should hold on EMP_PROJ;

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

$$F = \{SSN \rightarrow ENAME, \\ PNUMBER \rightarrow \{PNAME, PLOCATION\}, \\ \{SSN, PNUMBER\} \rightarrow HOURS\}$$

Using Algorithm , we calculate the following closure sets with respect to F:

$$\begin{aligned} \{SSN\}^+ &= \{SSN, ENAME\} \\ \{PNUMBER\}^+ &= \{PNUMBER, PNAME, PLOCATION\} \\ \{SSN, PNUMBER\}^+ &= \{SSN, PNUMBER, ENAME, PNAME, PLOCATION, HOURS\} \end{aligned}$$

5.2.3 Equivalence of Sets of Functional Dependencies

Definition:- a set of functional dependencies F is said to cover another set of functional dependencies E if every FD in E is also in F^+ ; that is , if every dependency in E can be inferred from F. We can say that E is covered by F.

(OR)

Definition:-Two sets of functional dependencies E and F are equivalent if $E^+ = F^+$. Hence equivalence means that every FD in E can be inferred from F and every FD in F can be inferred from E; therefore E is equivalent to F if both the conditions E cover F and F covers E hold.

5.2.4 Minimal Sets of Functional Dependencies

A set of functional dependencies F to be minimal if it satisfies the following condition:

- Every dependency in F has a single attributes for its right hand side
- We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y is a proper subset of X , and still have a set of dependencies that is equivalent to F.
- We cannot remove any dependency from F and still have a set of dependencies therefore equivalent to F.

Algorithm: Finding a minimal cover F for a set of functional dependencies E

1. Set $F=E$
2. Replace each functional dependency $X \rightarrow \{A_1, A_2, \dots, A_n\}$ in F by the functional dependencies $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

3. For each functional dependency $X \rightarrow A$ in F for each attributes B that is element of X if $\{ \{F - \{X \rightarrow A\}\} \cup \{X - \{B\} \rightarrow A\} \}$ is equivalent to F , then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in F .
4. For each remaining functional dependency $X \rightarrow A$ in F . If $\{F - \{X \rightarrow A\}\}$ is equivalent to F then remove $X \rightarrow A$ from F .

5.3 Normalization Based On Primary Key

The purpose of normalization ensures the followings:

- The elimination of problems associated with redundant data.
- The identification of various types of update anomalies such as insertion, deletion, and modification anomalies.
- How to recognize the appropriateness or quality of the design of relations.
- The concept of functional dependency, the main tool for measuring the appropriateness of attribute groupings in relations.
- How functional dependencies can be used to group attributes into relations that are in a known normal form.

5.3.1 Normalization of Relations

Normalization is a process of analysing the given relation schemas based on their FDs and primary keys to achieve the desirable properties .i.e.,

1. Minimizing redundancy
2. Minimizing the insertion, deletion and update anomalies

The normalization procedures provides database designer with the following approaches:

- A formal frame work for analyzing relation schemas based on their keys and on the functional dependencies among their attributes.
- A serious of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree.

The process of normalization through decomposition must also confirm the existence of additional properties that the relational would include two properties.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

1. The **lossless join or non-additive join property** which guarantees that the spurious tuples generation problem does not occur with respect to the relation schemas created after decomposition
2. The **Dependency preservation property**, which ensures that each FD is represented in some individual relation resulting after decomposition.

The database designer need not normalize to the highest possible normal form relations may be left in a lower normalization states that the process of storing the join of higher normal form relations as a base relation is known as **de-normalization**

5.3.2 Definition of keys and attributes participating in keys

Definition:

A **super key** of a relation schema $R\{A_1, A_2, \dots, A_n\}$ is a set of attributes $S \subseteq R$ with the property that no two tuples t_1 and t_2 in any legal relation state r of R will have $t_1[S] = t_2[S]$. A key K is a super key with the additional property that removal of any attributes from K will cause K not be a super key any more.

- The difference between a key and a superkey is that a key has to be minimal; that is, if we have a key $K = \{A_1, A_2, \dots, A_k\}$ of R , then $K - \{A_i\}$ $1 \leq i \leq k$ is not a key of R .
- For example $\{SSN\}$ is a key for EMPLOYEE, whereas $\{SSN\}$, $\{SSN, ENAME\}$, $\{SSN, ENAME, BDATE\}$, and any set of attributes that includes SSN are all superkeys.

If a relation schema has more than one key, each is called a **candidate key**. One of the candidate keys is arbitrarily designated to be the primary key, and the others are called secondary keys or alternate keys.

Definition:

An attribute of relation schema R is called a **prime attribute** of R if it is a member of some candidate key of R .

An attribute is called nonprime if it is not a prime attribute-that is, if it is not a member of any candidate key.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

5.3.3 First Normal Form (1 NF)

1NF disallows multi valued attributes, composite attributes, and their combination. It states that the domain of an attributes must include only atomic values and that the values of any attributes in a tuple must be a single value from the domain of the attributes.

- 1NF disallow “relations within relations” or “relations as attribute values within tuples.”
- The only attribute values permitted by 1NF are single **atomic values**.
- Ex:-consider the DEPARTMENT relation schema , whose primary key is DNUMBER

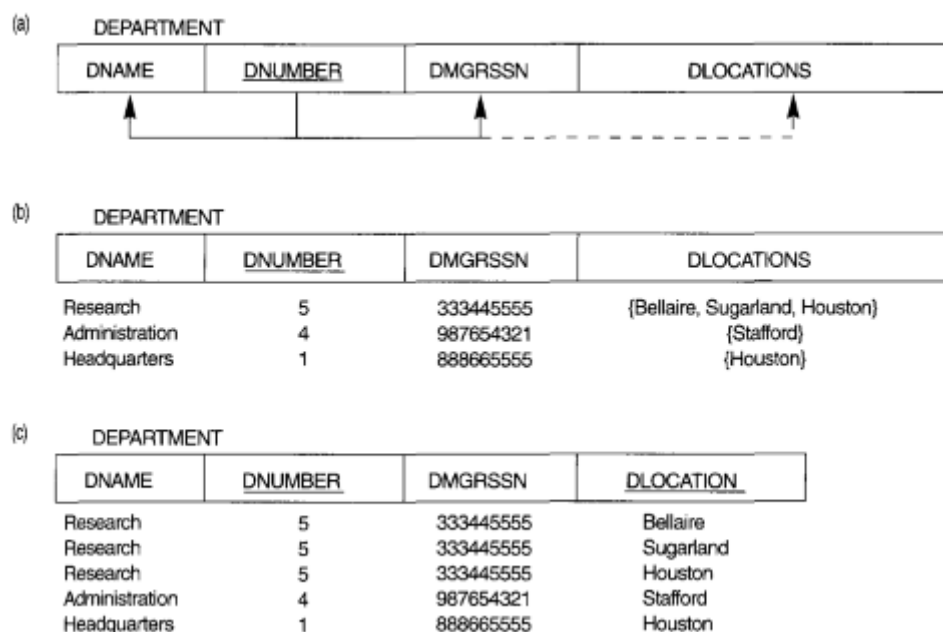


FIGURE 5.4 Normalization into 1NF. (a)A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of same relation with redundancy

- Each department can have any number of locations, which is it does not satisfy the 1NF because DLOCATIONS is not an atomic attribute.

These are the three main techniques to achieve first normal form for such a relation:-

- Remove the attributes DLOCATIONS that violates 1NF and place it in a separate relation DEPT_LOCATION along with the primary key DNUMBER of DEPARTMENT. The primary key of DEPT_LOCATION is the combination {DNUMBER, DLOCATIONS}.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT. The primary key becomes the combination {DNUMBER, DLOCATION} as shown in figure 5.4(c). This solution has the disadvantage of introducing redundancy in the relation.
- If a maximum numbers of values are known for the attribute-for example, if it is known that at most three locations can exist for a department-replace the DLOCATIONS attribute by three atomic attributes: DLOCATION1, DLOCATION2, and DLOCATION3. This solution has the disadvantage of introducing null values if most departments have fewer than three locations.

Among the three solutions above, the first is generally considered best because it does not suffer from redundancy.

- Example: 1NF disallow "relations within relations" or "relations as attribute values within tuples."

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

(a) **EMP_PROJ**

| SSN | ENAME | PROJS | |
|-----|-------|---------|-------|
| | | PNUMBER | HOURS |

(b) **EMP_PROJ**

| SSN | ENAME | PNUMBER | HOURS |
|-----------|----------------------|---------|-------|
| 123456789 | Smith, John B. | 1 | 32.5 |
| | | 2 | 7.5 |
| 666884444 | Narayan, Ramesh K. | 3 | 40.0 |
| 453453453 | Englich, Joyce A. | 1 | 20.0 |
| | | 2 | 20.0 |
| 333445555 | Wong, Franklin T. | 2 | 10.0 |
| | | 3 | 10.0 |
| | | 10 | 10.0 |
| 999887777 | Zelaya, Alicia J. | 20 | 10.0 |
| | | 30 | 30.0 |
| 987987987 | Jabbar, Ahmad V. | 10 | 10.0 |
| | | 30 | 35.0 |
| 987654321 | Wallace, Jennifer S. | 30 | 5.0 |
| | | 20 | 20.0 |
| 888665555 | Borg, James E. | 20 | 15.0 |
| | | 20 | null |

(c) **EMP_PROJ1**

| SSN | ENAME |
|-----|-------|
|-----|-------|

EMP_PROJ2

| SSN | PNUMBER | HOURS |
|-----|---------|-------|
|-----|---------|-------|

FIGURE 5.5 Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a "nested relation" attribute PROJS. (b) Example extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

Each tuple have a relation within it is called nested relations. Figure 5.5(a) shows how the EMP_PROJ could appear if nesting is allowed. Each tuple represent an employee entity and a relation PROJS (PNUMBER, HOURS) within each tuple represent the employee's projects and the hours per week that employee works on project. The schema of the EMP_PROJ relation can be represented as follows:

EMP_PROJ (SSN, ENAME, {PROJS (PNUMBER, HOURS)})

Decomposition of EMP_PROJ into two relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key is as shown in figure 5.5(c).

5.3.4 Second Normal Form (2 NF)

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

Second normal form (2NF) is based on the concept of full functional dependency.

Definition: A relation schema R is said to be in 2NF if it satisfies 1 NF and every nonprime attribute A in R is fully functionally dependent on the primary key of R.

In Figure 5.6, $\{SSN, PNUMBER\} \rightarrow HOURS$ is a full dependency because HOURS is dependent on primary key $\{SSN, PNUMBER\}$. However, $SSN \rightarrow ENAME$ and $PNUMBER \rightarrow PNAME$ dependency is partial because ENAME and PNAME are dependent on part of the primary key i.e SSN and PNUMBER.

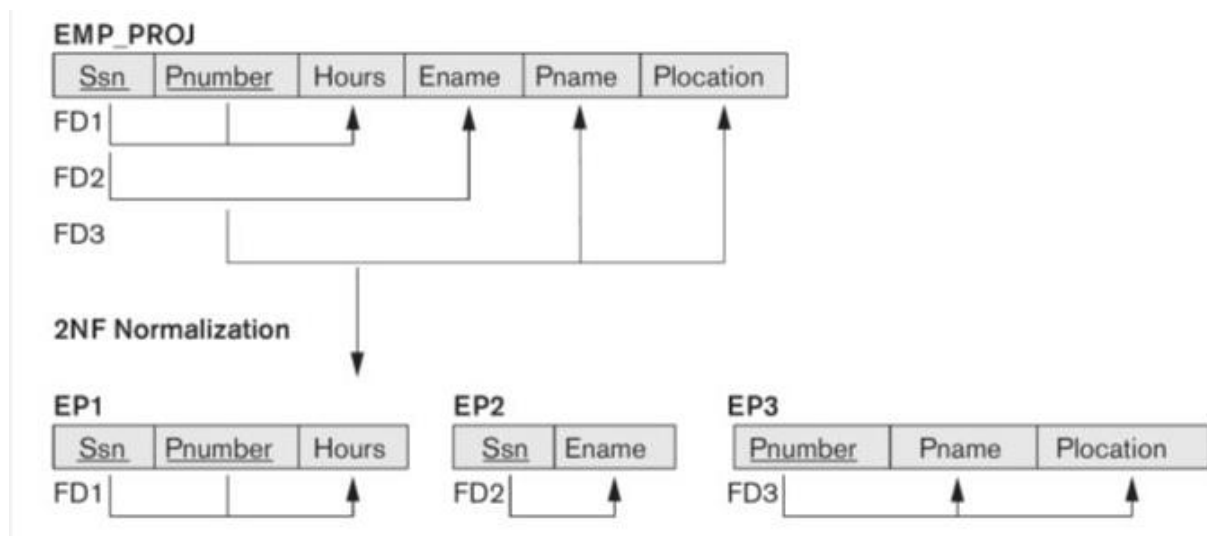


Figure 5.6 Normalizing EMP_PROJ into 2NF relations.

The EMP_PROJ relation in Figure 5.6 is in 1NF but is not in 2NF. The nonprime attribute ENAME violates 2NF because of FD2, as do the nonprime attributes PNAME and PLOCATION because of FD3. The functional dependencies FD2 and FD3 make ENAME, PNAME, and PLOCATION partially dependent on the primary key $\{SSN, PNUMBER\}$ of EMP_PROJ, thus violating the 2NF.

The functional dependencies FD1, FD2, and FD3 in Figure 5.6 hence lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown in Figure 5.6, each of which is in 2NF.

5.3.5 Third Normal Form (3 NF)

Third normal form (3NF) is based on the concept of transitive dependency.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Definition: A relation schema R is said to be in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.

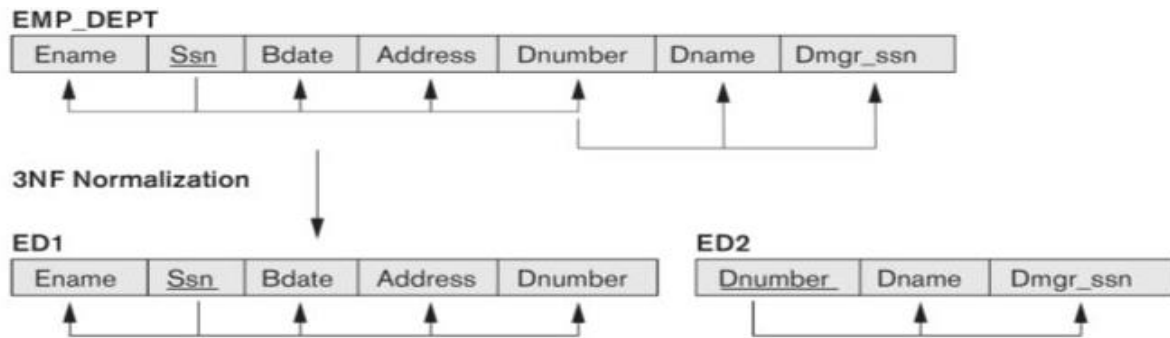


Figure 5.7 Normalizing EMP_DEPT into 3NF relations.

Example: The dependency $SSN \rightarrow DMGRSSN$ is transitive through $DNUMBER$ in relation EMP_DEPT of Figure 5.7 because both the dependencies $SSN \rightarrow DNUMBER$ and $DNUMBER \rightarrow DMGRSSN$ hold and $DNUMBER$ is neither a key itself nor a subset of the key of EMP_DEPT.

However, EMP_DEPT is not in 3NF because of the transitive dependency of DMGRSSN and also DNAME on SSN via DNUMBER. We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2 shown in Figure 5.7. A NATURAL JOIN operation on ED1 and ED2 will recover the original relation EMP_DEPT without generating spurious tuples.

5.4 General Definitions of Second and Third Normal Forms

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

| Normal Form | Test | Remedy (Normalization) |
|--------------|---|--|
| First (1NF) | Relation should have no multivalued attributes or nested relations. | Form new relations for each multivalued attribute or nested relation. |
| Second (2NF) | For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key. | Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it. |
| Third (3NF) | Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key. | Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s). |

TABLE 5.1 summary of Normal forms

5.4.1 General Definition of Second Normal Form

Definition: A relation schema R is in second normal form (2NF) if every nonprime attribute A in R is not partially dependent on any key of R.

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key.

Example: Consider the relation schema LOTS shown in Figure 5.8 (a), which describes land for sale in various counties of a state. Suppose that there are two candidate keys: PROPERTY_ID# and {COUNTY_NAME, LOT#}; that is, lot numbers are unique only within each county, but PROPERTY_ID numbers are unique across counties for the entire state.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Based on the two candidate keys PROPERTY_ID# and {COUNTY_NAME, LOT#}, the functional dependencies FD1 and FD2 of Figure 5.8(a) satisfies 2NF.

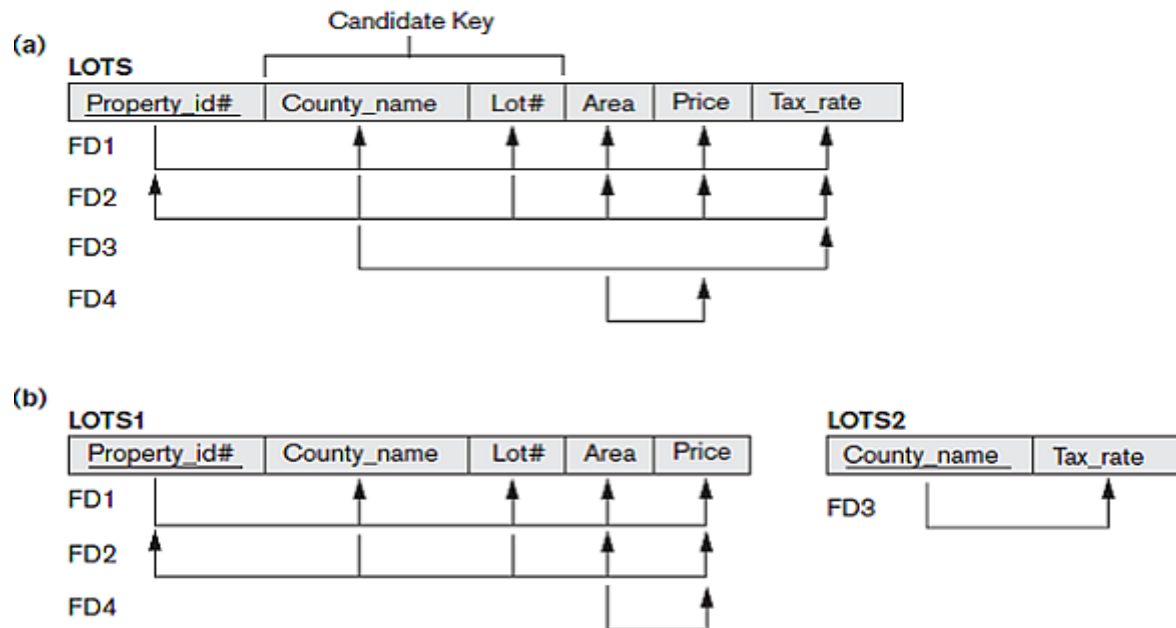


FIGURE 5.8 Normalization into 2NF . (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2

The LOTS relation schema FD3 violates 2NF because TAX_RATE is partially dependent on the candidate key {COUNTY_NAME, LOT#}. To normalize LOTS into 2NF, we decompose it into the two relations LOTS1 and LOTS2, shown in Figure 5.8(b).

Construct LOTS1 by removing the attribute TAX_RATE that violates 2NF from LOTS and placing it with COUNTYNAME into another relation LOTS2. Both LOTS1 and LOTS2 are in 2NF. Notice that FD4 does not violate 2NF and is carried over to LOTS1.

5.4.2 General Definition of Third Normal Form

Definition: A relation schema R is said to be in third normal form (3NF) if it satisfies 2NF and whenever a nontrivial functional dependency $X \rightarrow A$ holds in R, either (a) X is a super key of R, or (b) A is a prime attribute of R.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

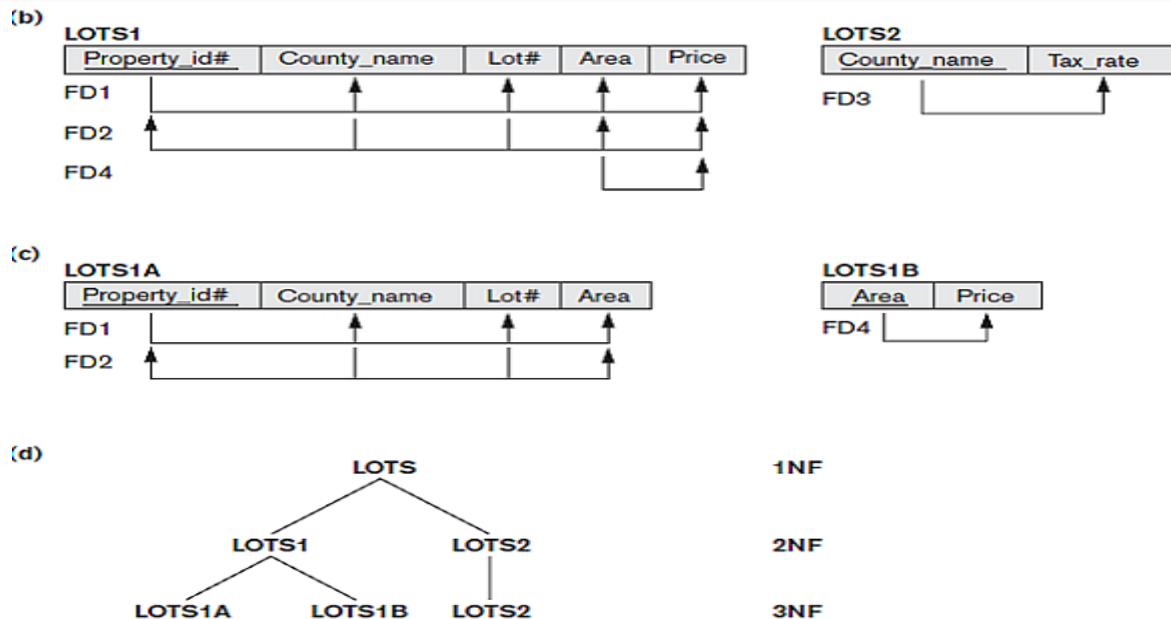


FIGURE 5.9 Normalization into 3NF (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B.
 (d) Summary of the progressive normalization of LOTS.

As shown in figure 5.9 (b) LOTS2 is in 3NF. However, FD4 in LOTS1 violates 3NF because AREA is not a superkey and PRICE is not a prime attribute in LOTS1. To normalize LOTS1 into 3NF, we decompose it into the relation schemas LOTS1A and LOTS1B shown in Figure 5.9(c).

We construct LOTS1A by removing the attribute PRICE that violates 3NF from LOTS1 and placing it with AREA (the left-hand side of FD4 that causes the transitive dependency) into another relation LOTS1B. Both LOTS1A and LOTS1B are in 3NF.

5.5 BOYCE-CODD Normal Form (BCNF)

Boyce-Codd Normal Form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF. That is, every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF.

Definition: A relation schema R is in BCNF if whenever a nontrivial functional dependency $X \rightarrow A$ holds in R, then X is a super key of R.

The only difference between the definitions of BCNF and 3NF is that, in 3NF for $X \rightarrow A$ dependency

a) X is a super key or b) A is a prime attribute of R.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

But for BCNF condition A is a prime attribute of R is absent.

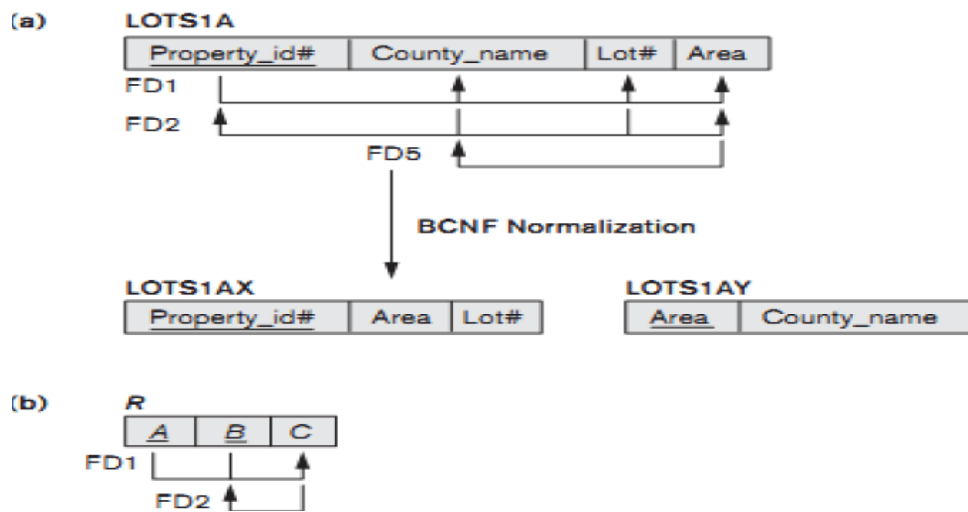


FIGURE 5.10 Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDS; it is in 3NF, but not in BCNF.

As shown in figure 5.10(a) FD5 violates BCNF in LOTS1A because AREA is not a super key of LOTS1A. But FD5 satisfies 3NF in LOTS1A because COUNTY_NAME is a prime attribute (condition (b)), but this condition does not exist in the definition of BCNF.

We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY as shown in Figure 5.10(a).

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

UNIT-6: Fundamentals of Data Base Transaction Processing & NoSQL

Transaction Processing

6.1 Introduction to transaction processing

6.2 Transaction and system concepts

6.3 Desirable properties of transactions

NoSQL –

6.4 Introduction

6.5 Distributed Systems

6.6 Advantages & Disadvantages of Distributed Computing

6.7 Scalability

6.8 What is NoSQL

6.9 Why NoSQL?

6.10 RDBMS vs. NoSQL,

6.11 Brief history of NoSQL

6.12 CAP theorem (Brewer's Theorem)

6.13 NoSQL pros/cons

6.14 NoSQL Categories

6.15 Production deployment

6.1 Introduction to Transaction Processing

A transaction is an atomic unit of work that is either completed in its entirety or not done at all.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

6.1.1 Single-User Versus Multiuser Systems

- A DBMS is single-user id at most one user at a time can use the system, and it is multiuser if many users can use the system—and hence access the database—concurrently.
- Most DBMS are multiuser (e.g., airline reservation system).
- Multiprogramming operating systems allow the computer to execute multiple programs (or processes) at the same time (having one CPU), concurrent execution of processes is actually interleaved.
- If the computer has multiple hardware processors (CPUs), parallel processing of multiple processes is possible

As illustrated in Figure 6.1, which shows two processes A and B executing concurrently in an interleaved fashion. Interleaving keeps the CPU busy when a process requires an input or output (r/o) operation, such as reading a block from disk. The CPU is switched to execute another process rather than remaining idle during r/o time. Interleaving also prevents a long process from delaying other processes.

If the computer system has multiple hardware processors, parallel processing of multiple processes is possible, as illustrated by processes C and D in Figure 6.1

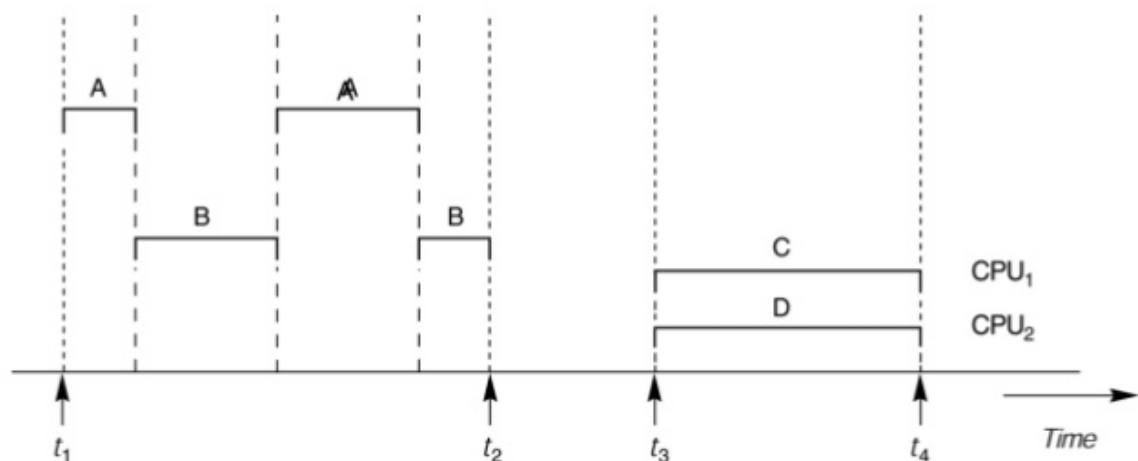


FIGURE 6.1 *Interleaved processing versus parallel processing of concurrent transactions.*

6.1.2 Transactions, Read and Write Operations, and DBMS Buffers

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

A transaction is a logical unit of database processing that includes one or more database access operations like insertion, deletion, modification, or retrieval operations.

The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL.

One way of specifying the transaction boundaries is by specifying explicit begin transaction and end transaction statements in an application program; in this case, all database access operations between the two are considered as forming one transaction.

If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a read-only transaction.

The basic database access operations that a transaction can include are as follows

- `read_item(X)`: Reads a database item named X into a program variable. To simplify our notation, we assume that the program variable is also named X.
- `write_item(X)`: Writes the value of program variable X into the database item named X.

Executing a `read_item(X)` command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the buffer to the program variable named X.

Executing a `write_item(X)` command includes the following steps:

1. Find the address of the disk block that contains item X.
2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
3. Copy item X from the program variable named X into its correct location in the buffer.
4. Store the updated block from the buffer back to disk

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Figure 6.2 shows examples of two very simple transactions. The read-set of a transaction is the set of all items that the transaction reads, and the write-set is the set of all items that the transaction writes. For example, the read-set of T in Figure 6.2 is {X, Y} and its write-set is also {X, Y}.

| (a) | T_1 | (b) | T_2 |
|-----|--|-----|--|
| | $\text{read_item}(X);$ $X := X - N;$ $\text{write_item}(X);$ $\text{read_item}(Y);$ $Y := Y + N;$ $\text{write_item}(Y);$ | | $\text{read_item}(X);$ $X := X + M;$ $\text{write_item}(X);$ |

FIGURE 6.2 Two sample transactions. (a) Transaction T1 . (b) Transaction T2.

6.1.3 Why Concurrency Control Is Needed

Several problems can occur when concurrent transactions execute in an uncontrolled manner.

- The Lost Update Problem
- The Temporary Update (or Dirty Read) Problem
- The Incorrect Summary Problem

Example: In airline reservations database in which a record is stored for each airline flight. Figure 6.2a shows a transaction T 1 that transfers N reservations from one flight i.e database item named X to another flight i.e database item named Y. Figure 6.2b shows a simpler transaction T2 that just reserves M seats on the first flight (X) referenced in transaction T2 .

The Lost Update Problem

This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect.

Suppose that transactions T 1 and T2 are submitted at the same time, and suppose that their operations are interleaved as shown in Figure 6.3., then the final value of item X is incorrect, because T2 reads the value of X before T 1 changes it in the database, and hence the updated value resulting from T 1 is lost.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

For example, if $X = 80$ at the start, $N = 5$ T_1 transfers 5 seat reservations from the flight X to the flight Y , and $M = 4$ T_2 reserves 4 seats on X , the final result should be $X = 79$; but in the interleaving of operations shown in Figure 6.3, it is $X = 84$ because the update in T_1 that removed the five seats from X was lost.

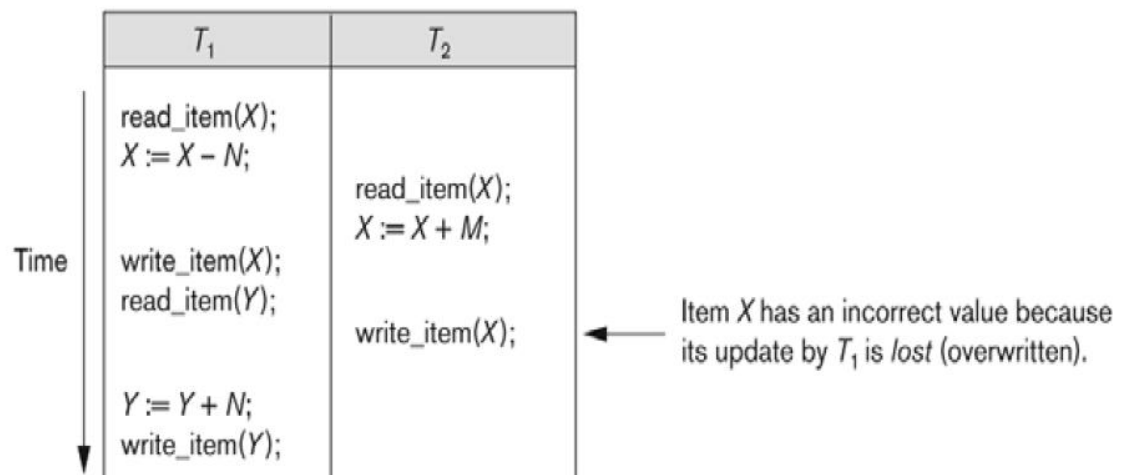


FIGURE 6.3 the lost update problem

The Temporary Update (or Dirty Read) Problem

The problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value.

Figure 6.4 shows an example where T_1 updates item X and then fails before completion, so the system must change X back to its original value. Before doing it, transaction T_2 reads the "temporary" value of X , which will not be recorded permanently in the database because of the failure of T_1 . The value of item X that is read by T_2 is called dirty data, because it has been created by a transaction that has not completed and committed yet; hence, this problem is also known as the dirty read problem.

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

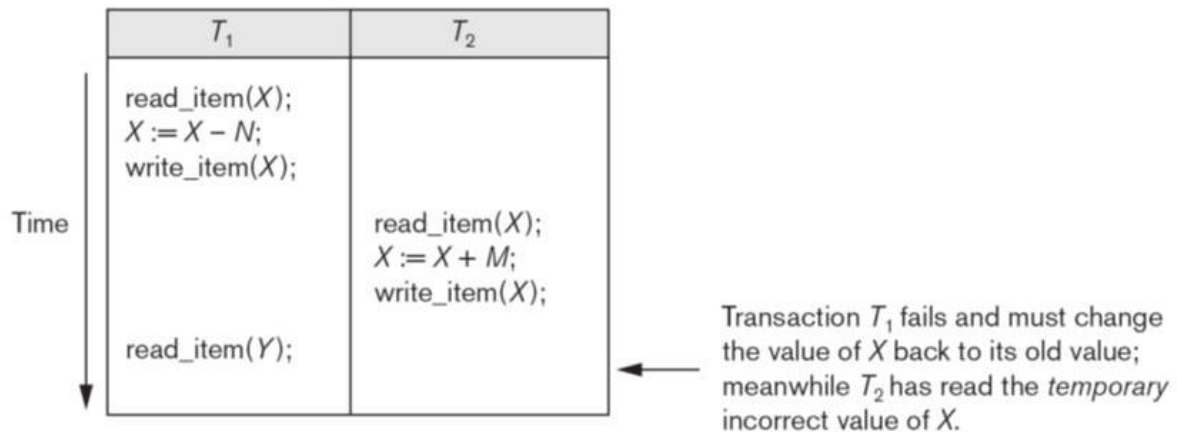
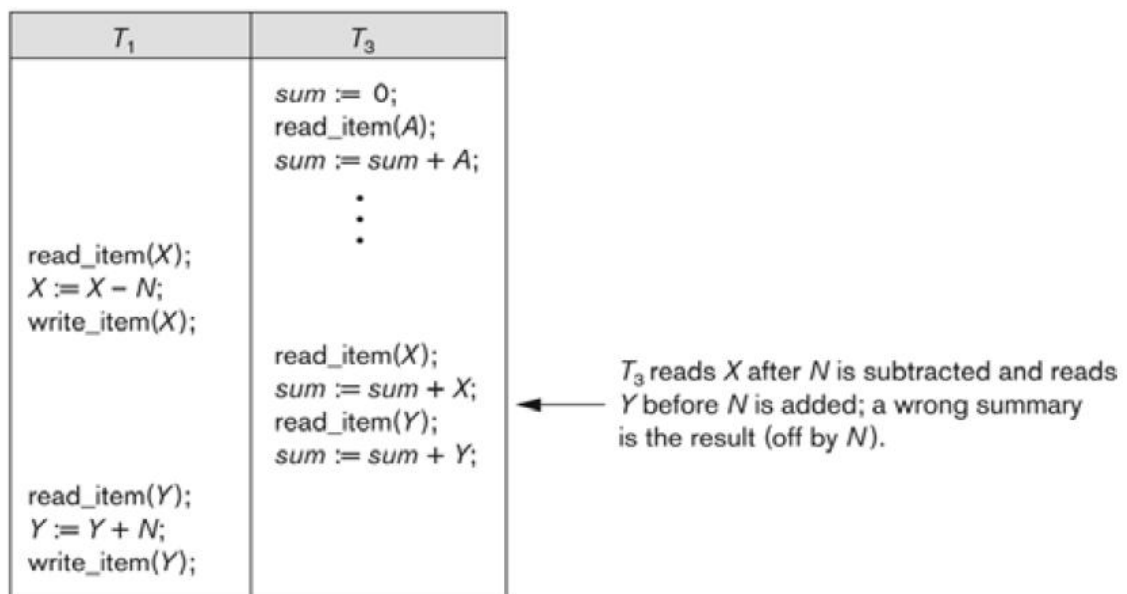


FIGURE 6.4 *The Temporary Update (or Dirty Read) Problem*

The Incorrect Summary Problem

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

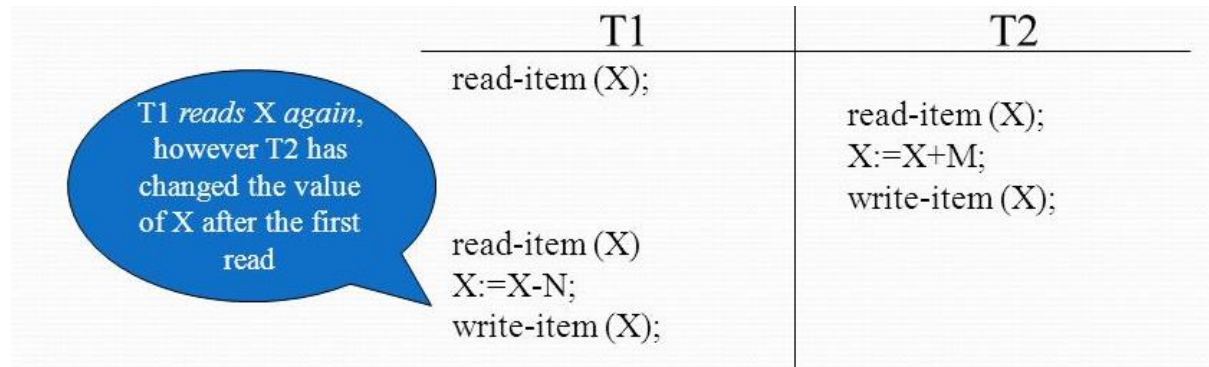
For example, suppose that a transaction T3 is calculating the total number of reservations on all the flights; meanwhile, transaction T1 is executing. If the interleaving of operations shown in Figure 6.5 occurs, the result of T3 will be wrong because T3 reads the value of X after N seats have been subtracted from it but reads the value of Y before those N seats have been added to it.



Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

FIGURE 6.5 The Incorrect Summary Problem

Another problem that may occur is called unrepeatable read, where a transaction T1 reads an item twice and the item is changed by another transaction T2 between the two reads. Hence, T1 receives different values for its two reads of the same item.



6.1.4 Why Recovery Is Needed

Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either

- (1) All the operations in the transaction are completed successfully and their effect is recorded permanently in the database
- (2) The transaction has no effect on the database or on any other transactions, when its operation fails.

Types of Failures

Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

1. **A computer failure (system crash):** A hardware, software, or network error occurs in the computer system during transaction execution. For example, main memory failure.
2. **A transaction or system error:** Some operation in the transaction may cause it to fail, such as integer overflow or division by zero.

Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

3. Local errors or exception conditions detected by the transaction: During transaction execution, certain conditions may occur that necessitate cancellation of the transaction.

For example, data for the transaction may not be found. Insufficient account balance in a banking database may cause a transaction, such as a fund withdrawal, to be cancelled. This exception should be programmed in the transaction itself, and hence would not be considered a failure.

4. Concurrency control enforcement: The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock.

5. Disk failure: Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.

6. Physical problems and catastrophes: This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

Failures of types 1, 2, 3, and 4 are more common than those of types 5 or 6. Whenever a failure of type 1 through 4 occurs, the system must keep sufficient information to recover from the failure. Disk failure or other catastrophic failures of type 5 or 6 do not happen frequently; if they do occur, recovery is a major task.

6.2 Transaction and System Concepts

A transaction is an atomic unit of work that is either completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts (see below). Hence, the recovery manager keeps track of the following operations:

1 BEGIN_TRANSACTION: This marks the beginning of transaction execution.

2 READ or WRITE: These specify read or write operations on the database items that are executed as part of a transaction.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

3 END_TRANSACTION: This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. At this point it may be necessary to check whether the changes made by the transaction can be permanently stored to the database (committed) or whether the transaction has to be aborted because it violates serializability or for some other reason.

4 COMMIT_TRANSACTION: This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.

5 ROLLBACK (or ABORT): This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

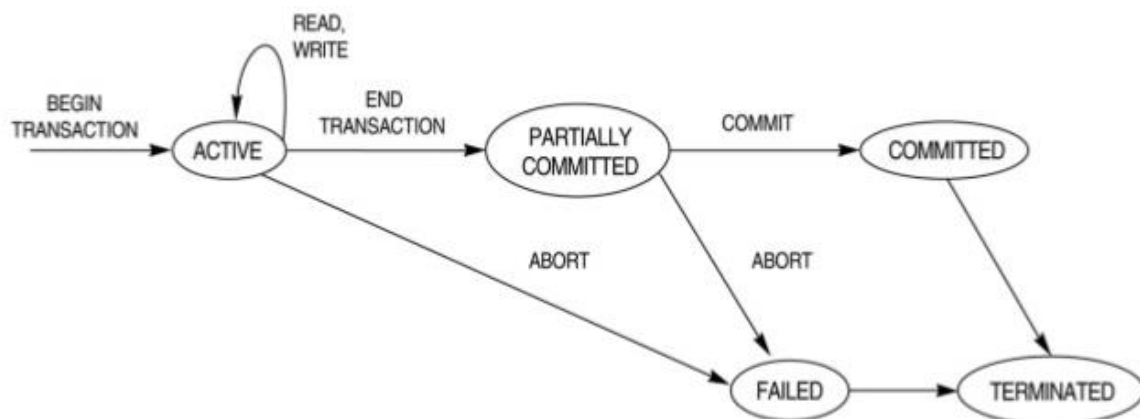


FIGURE 6.6 State transition diagram illustrating the states for transaction execution,

Figure 6.6 shows a state transition diagram that describes how a transaction moves through its execution states. A transaction goes into an **active state** immediately after it starts execution, where it can issue READ and WRITE operations. When the transaction ends, it moves to the **partially committed state**. At this point, some recovery protocols need to ensure that transaction is committed or failed.

Once this check is successful, the transaction is said to have reached its commit point and enters the **committed state**, means that it has concluded its execution successfully and all its changes must be recorded permanently in the database

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

A transaction can go to the **failed state** if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its WRITE operations on the database.

The **terminated state** corresponds to the transaction leaving the system.

6.2.2 The System Log

The system should be able to recover from failures that affect transactions, so the system maintains a log to keep track of all transactions that affect the values of database items.

Log records consists of the following information (T refers to a unique transaction_id)

1. [start_Transaction, T]: Indicates that transaction T has started execution
2. [write_i tem,T,X,old_value,new_value]: Indicates that transaction T has changed the value of database item X from old_value to new_value.
3. [read_i tem,T,X]: Indicates that transaction T has read the value of database item X.
4. [commit,T]: Indicates that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
5. [abort,T]: Indicates that transaction T has been aborted.

6.2.3 Commit Point of a Transaction

A transaction T reaches its commit point when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database have been recorded in the log and in the database. The transaction writes a commit record [commit,T] into the log.

At the time of a system crash, only the log entries that have been written back to disk are considered in the recovery process because the contents of main memory may be lost. Hence, before a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk. This process is called **force-writing** the log file before committing a transaction.

6.3 Desirable Properties of Transactions

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Transactions properties are called the ACID properties. The following are the ACID properties:

- 1. Atomicity:** A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
- 2. Consistency preservation:** A transaction is consistency preserving if its complete execution takes the database from one consistent state to another.
- 3. Isolation:** A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.
- 4. Durability or permanency:** The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

6.4 Introduction to NoSQL

In the computing system (web and business applications), there are enormous data that comes out every day from the web. A large section of these data is handled by Relational database management systems (RDBMS). The idea of relational model came with E.F.Codd's 1970 paper "A relational model of data for large shared data banks" which made data modeling and application programming much easier. Beyond the intended benefits, the relational model is well-suited to client-server programming and today it is predominant technology for storing structured data in web and business applications.

Classical relation database follow the ACID Rules

A database transaction, must be atomic, consistent, isolated and durable. Below we have discussed these four points.

Atomic : A transaction is a logical unit of work which must be either completed with all of its data modifications, or none of them is performed.

Consistent : At the end of the transaction, all data must be left in a consistent state.

Isolated : Modifications of data performed by a transaction must be independent of another transaction. Unless this happens, the outcome of a transaction may be erroneous.

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

Durable : When the transaction is completed, effects of the modifications performed by the transaction must be permanent in the system.

Often these four properties of a transaction is acronym as ACID.

6.5 Distributed Systems

A distributed system consists of multiple computers and software components that communicate through a computer network (a local network or by a wide area network). A distributed system can consist of any number of possible configurations, such as mainframes, workstations, personal computers, and so on. The computers interact with each other and share the resources of the system to achieve a common goal.

6.6 Advantages and Disadvantages of Distributed Computing

Advantages of Distributed Computing

Reliability (fault tolerance) :

The important advantage of distributed computing system is reliability. If some of the machines within the system crash, the rest of the computers remain unaffected and work does not stop.

Scalability:

In distributed computing the system can easily be expanded by adding more machines as needed.

Sharing of Resources:

Shared data is essential to many applications such as banking, reservation system. As data or resources are shared in distributed system, other resources can be also shared (e.g. expensive printers).

Flexibility:

As the system is very flexible, it is very easy to install, implement and debug new services.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

Speed:

A distributed computing system can have more computing power and its speed makes it different than other systems.

Open system:

As it is open system, every service is equally accessible to every client i.e. local or remote.

Performance:

The collection of processors in the system can provide higher performance (and better price/performance ratio) than a centralized computer.

Disadvantages of Distributed Computing**Troubleshooting:**

Troubleshooting and diagnosing problems.

Software:

Less software support is the main disadvantage of distributed computing system.

Networking:

The network infrastructure can create several problems such as transmission problem, overloading, loss of messages.

Security:

Easy access in distributed computing system increases the risk of security and sharing of data generates the problem of data security

6.7 Scalability

In electronics (including hardware, communication and software), scalability is the ability of a system to expand to meet your business needs. For example scaling a web application is all

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

about allowing more people to use your application. You scale a system by upgrading the existing hardware without changing much of the application or by adding extra hardware.

There are two ways of scaling horizontal and vertical scaling :

Vertical scaling

To scale vertically (or scale up) means to add resources within the same logical unit to increase capacity. For example to add CPUs to an existing server, increase memory in the system or expanding storage by adding hard drive.

Horizontal scaling

To scale horizontally (or scale out) means to add more nodes to a system, such as adding a new computer to a distributed software application. In NoSQL system, data store can be much faster as it takes advantage of “scaling out” which means to add more nodes to a system and distribute the load over those nodes.

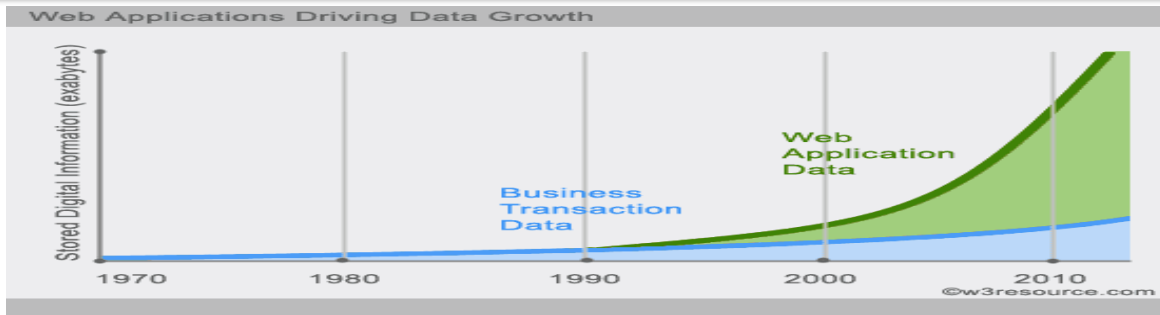
6.8 What is NoSQL ?

NoSQL is a non-relational database management systems, different from traditional relational database management systems in some significant ways. It is designed for distributed data stores where very large scale of data storing needs (for example Google or Facebook which collects terabits of data every day for their users). These type of data storing may not require fixed schema, avoid join operations and typically scale horizontally.

6.9 Why NoSQL ?

In today's time data is becoming easier to access and capture through third parties such as Facebook, Google+ and others. Personal user information, social graphs, geo location data, user-generated content and machine logging data are just a few examples where the data has been increasing exponentially. To avail the above service properly, it is required to process huge amount of data, which SQL databases were never designed. The evolution of NoSql databases is to handle these huge data properly.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus



Example :

Social-network graph :

Each record: UserID1, UserID2

Separate records: UserID, first_name, last_name, age, gender,...

Task: Find all friends of friends of friends of ... friends of a given user.

Wikipedia pages :

Large collection of documents

Combination of structured and unstructured data

Task: Retrieve all pages regarding athletics of Summer Olympic before 1950.

6.10 RDBMS vs NoSQL

RDBMS

- Structured and organized data
- Structured query language (SQL)
- Data and its relationships are stored in separate tables.
- Data Manipulation Language, Data Definition Language
- Tight Consistency
- BASE Transaction

NoSQL

- Stands for Not Only SQL
- No declarative query language
- No predefined schema
- Key-Value pair storage, Column Store, Document Store, Graph databases
- Eventual consistency rather ACID property
- Unstructured and unpredictable data

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- CAP Theorem
- Prioritizes high performance, high availability and scalability

6.11 Brief history of NoSQL

The term NoSQL was coined by Carlo Strozzi in the year 1998. He used this term to name his Open Source, Light Weight, DataBase which did not have an SQL interface.

In the early 2009, when last.fm wanted to organize an event on open-source distributed databases, Eric Evans, a Rackspace employee, reused the term to refer databases which are non-relational, distributed, and does not conform to atomicity, consistency, isolation, durability - four obvious features of traditional relational database systems.

In the same year, the "no:sql(east)" conference held in Atlanta, USA, NoSQL was discussed and debated a lot.

And then, discussion and practice of NoSQL got a momentum, and NoSQL saw an unprecedented growth.

6.12 CAP Theorem (Brewer's Theorem)

You must understand the CAP theorem when you talk about NoSQL databases or in fact when designing any distributed system. CAP theorem states that there are three basic requirements which exist in a special relation when designing applications for a distributed architecture.

Consistency - This means that the data in the database remains consistent after the execution of an operation. For example after an update operation all clients see the same data.

Availability - This means that the system is always on (service guarantee availability), no downtime.

Partition Tolerance - This means that the system continues to function even the communication among the servers is unreliable, i.e. the servers may be partitioned into multiple groups that cannot communicate with one another.

In theoretically it is impossible to fulfill all 3 requirements. CAP provides the basic requirements for a distributed system to follow 2 of the 3 requirements. Therefore all the

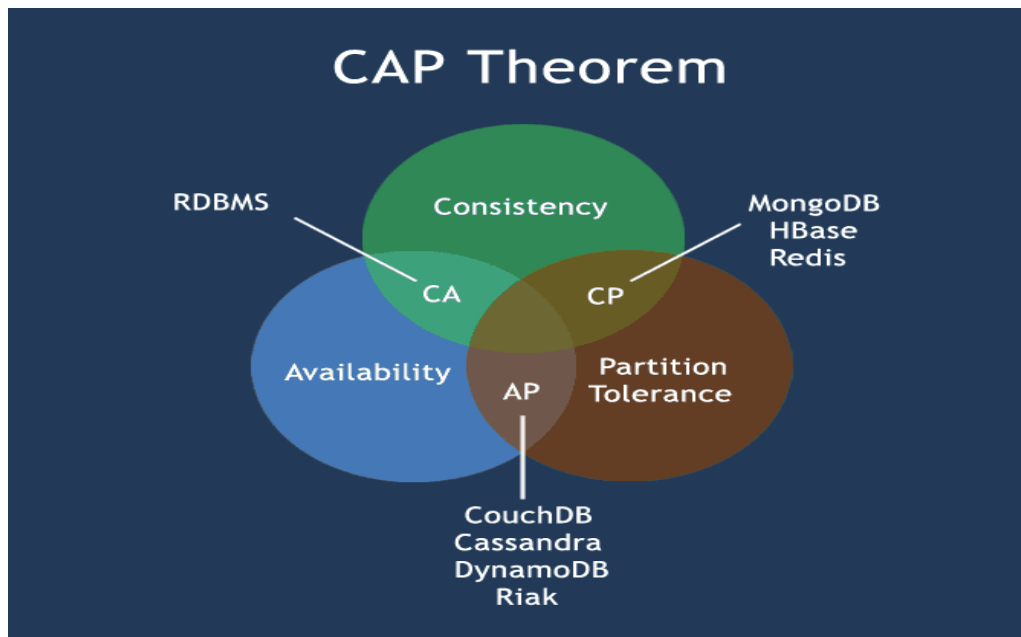
Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

current NoSQL database follow the different combinations of the C, A, P from the CAP theorem. Here is the brief description of three combinations CA, CP, AP:

CA - Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.

CP - Some data may not be accessible, but the rest is still consistent/accurate.

AP - System is still available under partitioning, but some of the data returned may be inaccurate.



6.13 NoSQL pros/cons

Advantages :

- High scalability
- Distributed Computing
- Lower cost
- Schema flexibility, semi-structure data
- No complicated Relationships

Disadvantages

- No standardization
- Limited query capabilities (so far)

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- Eventual consistent is not intuitive to program for

TheBASE

The BASE acronym was defined by Eric Brewer, who is also known for formulating the CAP theorem.

The CAP theorem states that a distributed computer system cannot guarantee all of the following three properties at the same time:

- Consistency
- Availability
- Partition tolerance

A BASE system gives up on consistency.

- Basically Available indicates that the system does guarantee availability, in terms of the CAP theorem.
- Soft state indicates that the state of the system may change over time, even without input. This is because of the eventual consistency model.
- Eventual consistency indicates that the system will become consistent over time, given that the system doesn't receive input during that time.

ACID vs BASE

| ACID | BASE |
|-------------|----------------------|
| Atomic | Basically Available |
| Consistency | Soft state |
| Isolation | Eventual consistency |
| Durable | |

6.14 NoSQL Categories

There are four general types (most common categories) of NoSQL databases. Each of these categories has its own specific attributes and limitations. There is not a single solution which

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

is better than all the others; however there are some databases that are better to solve specific problems. To clarify the NoSQL databases, let's discuss the most common categories:

- Key-value stores
- Column-oriented
- Graph
- Document oriented

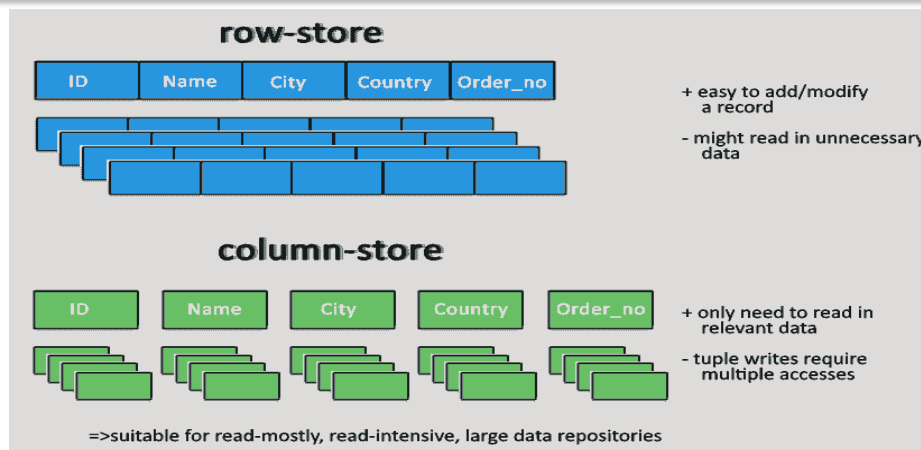
Key-value stores:

- Key-value stores are most basic types of NoSQL databases.
- Designed to handle huge amounts of data.
- Based on Amazon's Dynamo paper.
- Key value stores allow developer to store schema-less data.
- In the key-value storage, database stores data as hash table where each key is unique and the value can be string, JSON, BLOB (basic large object) etc.
- A key may be strings, hashes, lists, sets, sorted sets and values are stored against these keys.
- For example a key-value pair might consist of a key like "Name" that is associated with a value like "Robin".
- Key-Value stores can be used as collections, dictionaries, associative arrays etc.
- Key-Value stores follows the 'Availability' and 'Partition' aspects of CAP theorem.
- Key-Values stores would work well for shopping cart contents, or individual values like color schemes, a landing page URI, or a default account number.

Example of Key-value store DataBase : Redis, Dynamo, Riak. etc.

Pictorial Presentation :

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus



Column-oriented databases:

- Column-oriented databases primarily work on columns and every column is treated individually.
- Values of a single column are stored contiguously.
- Column stores data in column specific files.
- In Column stores, query processors work on columns too.
- All data within each column datafile have the same type which makes it ideal for compression.
- Column stores can improve the performance of queries as it can access specific column data.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- High performance on aggregation queries (e.g. COUNT, SUM, AVG, MIN, MAX).
- Works on data warehouses and business intelligence, customer relationship management (CRM), Library card catalogs etc.

Example of Column-oriented databases : BigTable, Cassandra, SimpleDB etc.

Pictorial Presentation :

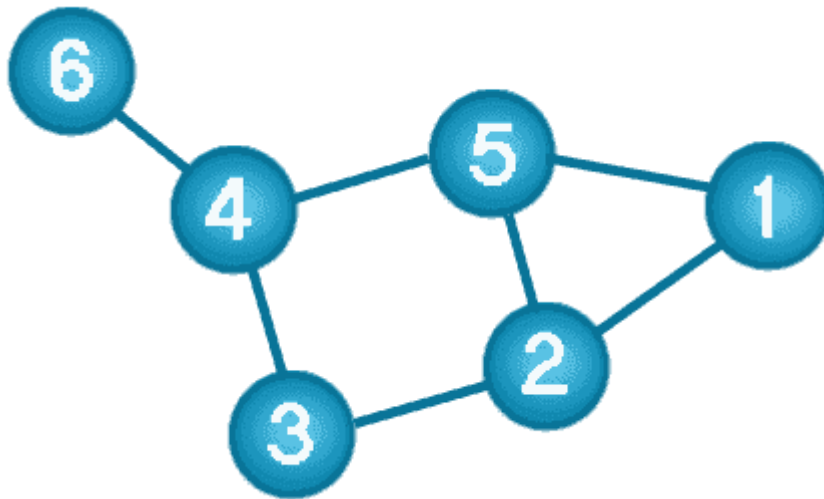


Graph databases:

A graph data structure consists of a finite (and possibly mutable) set of ordered pairs, called edges or arcs, of certain entities called nodes or vertices.

Following picture presents a labeled graph of 6 vertices and 7 edges.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus



What is a Graph Database?

- A graph database stores data in a graph.
- It is capable of elegantly representing any kind of data in a highly accessible way.
- A graph database is a collection of nodes and edges
- Each node represents an entity (such as a student or business) and each edge represents a connection or relationship between two nodes.
- Every node and edge is defined by a unique identifier.
- Each node knows its adjacent nodes.
- As the number of nodes increases, the cost of a local step (or hop) remains the same.
- Index for lookups.

Here is a comparison between the classic relational model and the graph model :

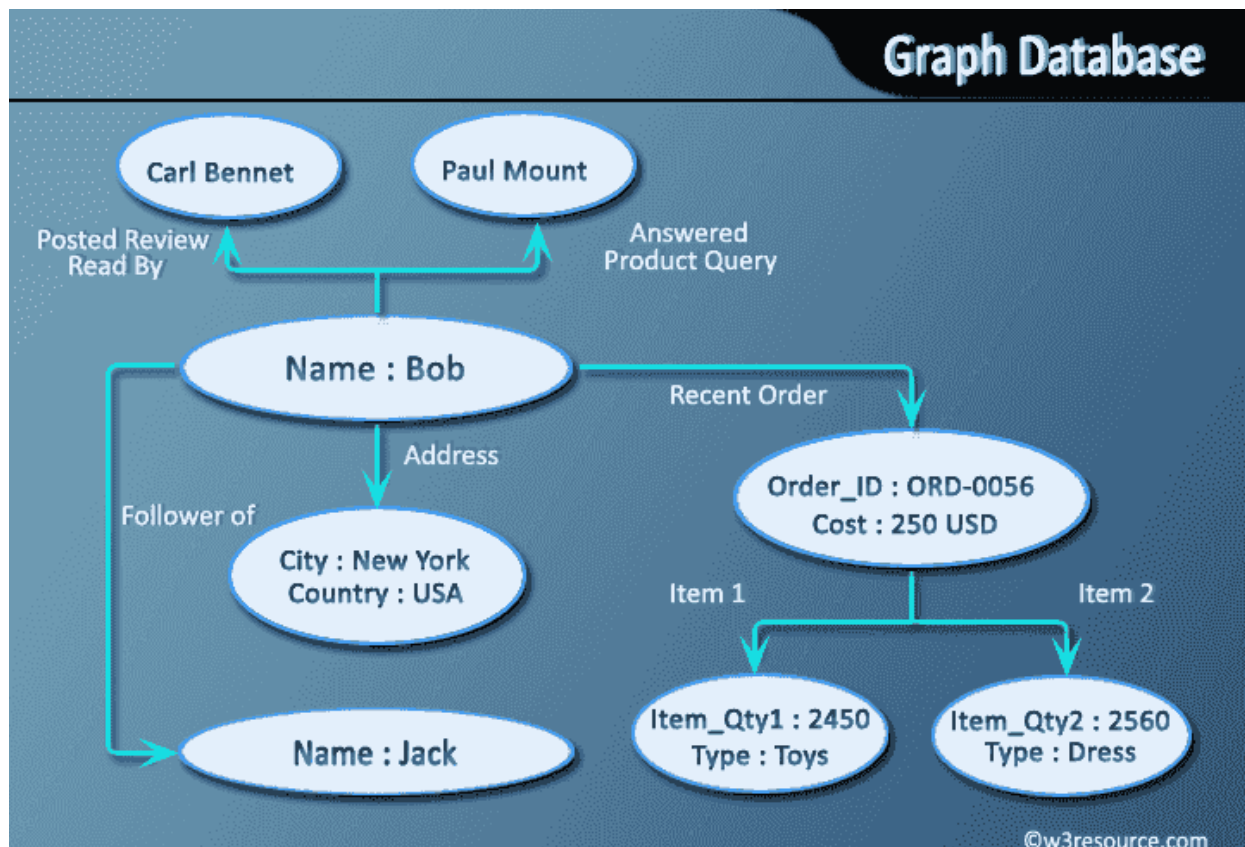
| Relational model | Graph model |
|------------------|------------------------|
| Tables | Vertices and Edges set |
| Rows | Vertices |
| Columns | Key/value pairs |

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

| | |
|------------------|-------------|
| Relational model | Graph model |
| Joins | Edges |

Example of Graph databases : OrientDB, Neo4J, Titan.etc.

Pictorial Presentation :



Document Oriented databases:

- A collection of documents
- Data in this model is stored inside documents.
- A document is a key value collection where the key allows access to its value.
- Documents are not typically forced to have a schema and therefore are flexible and easy to change.
- Documents are stored into collections in order to group different kinds of data.

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

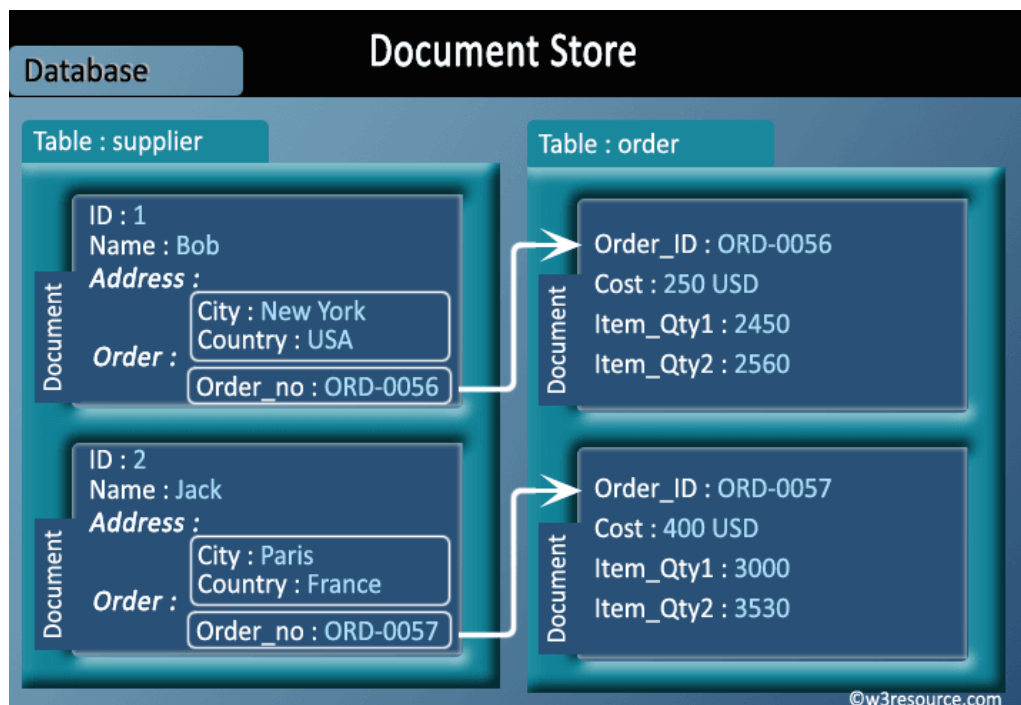
- Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.

Here is a comparison between the classic relational model and the document model:

| Relational model | Document model |
|------------------|-----------------|
| Tables | Collections |
| Rows | Documents |
| Columns | Key/value pairs |
| Joins | not available |

Example of Document Oriented databases : MongoDB, CouchDB etc.

Pictorial Presentation :



6.15 Production deployment

There are large number of companies using NoSQL. To name a few :

- Google

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus

- Facebook
- Mozilla
- Adobe
- Foursquare
- LinkedIn
- Digg
- McGraw-Hill Education
- Vermont Public Radio

Note: This is only Basic Information for students. Please refer “Reference Books” prescribed as per syllabus