190+30=220

DATABASE MANAGEMENT SYSTEM (Effective from the academic year 2018 -2019) SEMESTER – V			
Course Code	**18CS53**	**CIE Marks**	40
Number of Contact Hours/Week	3:2:0	**SEE Marks**	60
Total Number of Contact Hours	50	**Exam Hours**	03
CREDITS –4			

Course Learning Objectives: This course (18CS53) will enable students to:

- Provide a strong foundation in database concepts, technology, and practice.
- Practice SQL programming through a variety of database problems.
- Demonstrate the use of concurrency and transactions in database
- Design and build database applications for real world problems.

Module 1	**Contact Hours**
Introduction to Databases: Introduction, Characteristics of database approach, Advantages of using the DBMS approach, History of database applications. **Overview of Database Languages and Architectures:** Data Models, Schemas, and Instances. Three schema architecture and data independence, database languages, and interfaces, The Database System environment. **Conceptual Data Modelling using Entities and Relationships:** Entity types, Entity sets, attributes, roles, and structural constraints, Weak entity types, ER diagrams, examples, Specialization and Generalization. **Textbook 1:Ch 1.1 to 1.8, 2.1 to 2.6, 3.1 to 3.10** **RBT: L1, L2, L3**	10
Module 2	
Relational Model: Relational Model Concepts, Relational Model Constraints and relational database schemas, Update operations, transactions, and dealing with constraint violations. **Relational Algebra:** Unary and Binary relational operations, additional relational operations (aggregate, grouping, etc.) Examples of Queries in relational algebra. **Mapping Conceptual Design into a Logical Design:** Relational Database Design using ER-to-Relational mapping. **SQL:** SQL data definition and data types, specifying constraints in SQL, retrieval queries in SQL, INSERT, DELETE, and UPDATE statements in SQL, Additional features of SQL. **Textbook 1: Ch4.1 to 4.5, 5.1 to 5.3, 6.1 to 6.5, 8.1; Textbook 2: 3.5** **RBT: L1, L2, L3**	10
Module 3	
SQL: Advances Queries: More complex SQL retrieval queries, Specifying constraints as assertions and action triggers, Views in SQL, Schema change statements in SQL. **Database Application Development:** Accessing databases from applications, An introduction to JDBC, JDBC classes and interfaces, SQLJ, Stored procedures, Case study: The internet Bookshop. **Internet Applications:** The Three-Tier application architecture, The presentation layer, The Middle Tier **Textbook 1: Ch7.1 to 7.4; Textbook 2: 6.1 to 6.6, 7.5 to 7.7.** **RBT: L1, L2, L3**	10
Module 4	
Normalization: Database Design Theory – Introduction to Normalization using Functional and Multivalued Dependencies: Informal design guidelines for relation schema, Functional Dependencies, Normal Forms based on Primary Keys, Second and Third Normal Forms, Boyce-Codd Normal Form, Multivalued Dependency and Fourth Normal Form, Join Dependencies and Fifth Normal Form. **Normalization Algorithms:** Inference Rules, Equivalence, and Minimal Cover, Properties of Relational Decompositions, Algorithms for	

Designs, Further discussion of Multivalued dependencies and 4NF, Other dependencies and Normal Forms **Textbook 1: Ch14.1 to 14.7, 15.1 to 15.6** **RBT: L1, L2, L3**	10
Designs, Further discussion of Multivalued dependencies and 4NF, Other dependencies and Normal Forms **Textbook 1: Ch14.1 to 14.7, 15.1 to 15.6** **RBT: L1, L2, L3**	
Module 5	
Transaction Processing: Introduction to Transaction Processing, Transaction and System concepts, Desirable properties of Transactions, characterizing schedules based on recoverability, characterizing schedules based on Serializability, Transaction support in SQL. **Concurrency Control in Databases:** Two-phase locking techniques for Concurrency control, Concurrency control based on Timestamp ordering, Multiversion Concurrency control techniques, Validation Concurrency control techniques, Granularity of Data items and Multiple Granularity Locking. **Introduction to Database Recovery Protocols:** Recovery Concepts, NO-UNDO/REDO recovery based on Deferred update, Recovery techniques based on immediate update, Shadow paging, Database backup and recovery from catastrophic failures **Textbook 1: 20.1 to 20.6, 21.1 to 21.7, 22.1 to 22.4, 22.7.** **RBT: L1, L2, L3**	10

Course Outcomes: The student will be able to:

- Identify, analyze and define database objects, enforce integrity constraints on a database using RDBMS.
- Use Structured Query Language (SQL) for database manipulation.
- Design and build simple database systems
- Develop application to interact with databases.

Question Paper Pattern:

- The question paper will have ten questions.
- Each full Question consisting of 20 marks
- There will be 2 full questions (with a maximum of four sub questions) from each module.
- Each full question will have sub questions covering all the topics under a module.
- The students will have to answer 5 full questions, selecting one full question from each module.

Textbooks:

1. Fundamentals of Database Systems, Ramez Elmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson.
2. Database management systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill

Reference Books:

1. Silberschatz Korth and Sudharshan, Database System Concepts, 6th Edition, Mc-GrawHill, 2013.
2. Coronel, Morris, and Rob, Database Principles Fundamentals of Design, Implementation and Management, Cengage Learning 2012.

KARANAM SUNIL KUMAR
Asst. Prof. Dept of CSE
RNSIT
ph: 9972995720

SUB: Database Management System

Module 1

CONTENTS:

Introduction to Databases

1. Introduction
2. Characteristics of database approach
3. Advantages of using DBMS approach
4. History of Database applications

Over view of Database Languages and Architectures

1. Data Models
2. Schema and Instances
3. Three schema architecture and data independence
4. Database Languages and Interfaces
5. The Database system environment

Conceptual Data Modelling using Entities and Relationship

1. Entity types
2. Entity sets
3. Attributes
4. Roles and structural constraints
5. Weak entity types
6. ER Diagram examples
7. Specialization and Generalization

Introduction to database System :

Databases and database technology have a major impact on growing use of Computers

→ Databases play a critical role in almost all areas where Computers are used

examples of areas : Business, Engineering, Medicine, Transportation, Law, Education. and library Science.

Definition of database :

A database is a collection of related data. i.e collection of known facts with implicit meaning.

ex : Student- database

USN	Name	Sem	age

Another definition for database :

The collection of related data with an implicit meaning is a database.

Implicit properties of database:

- A database represents some aspect of the real world
- A database is a logically coherent collection of data with some inherent meaning.
- A database is designed, built and populated with data for Spefic purpose.
- A database can be any size and of varying complexity
- A database can be generated and maintained manually or by machine.

Database Management System :

DBMS is a collection of programs that enables user to create and maintain a database.

OR

DBMS is a general-purpose software system that facilitates the process of defining, constructing, manipulating and sharing a database among various users and application.

- Defining : Specifying the data types, structures and constraints associated with the different attributes for the data to be stored in the database.

Defining for Student database

Name	char(20)
USN	char(10)
Sem	char(2)
age	number

→ Constructing a database is the process of storing the data on some storage medium ie controlled by the DBMS.

Ex: Disk, Drum, Tape.

- Manipulating a database refers to querying the database to retrieve specific data., updating the database to reflect changes in the mini world and generating reports from the data.

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

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

Example : Let us Consider UNIVERSITY database for maintaining information Concerning Students, courses and grades in a university environment. The database is organized as five files, each of which stores data records of the same type. as shown in fig below.

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....	....	.....
....	....	.....
....	....	.....
Prerequisite_number	XXXXNNNN	PREREQUISITE

→ The STUDENT file stores data on each Student.

→ The COURSE file stores data on each Course

→ The SECTION file stores data on each Section of a Course.

→ The GRADE_REPORT file stores the grades that Students receive in the various Sections they have Completed.

→ The PREREQUISITE file stores the prerequisites of each Course.

To define this database, we must Specify the Structure of the records of each file by Specifying the different types of data elements to be Stored in each record. as Shown in the fig below.

STUDENT

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

COURSE

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

SECTION

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

GRADE_REPORT

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

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

STUDENT record includes data to represent - the Student's Name, Student_number, Class and major.

COURSE record includes data to represent - the Course_name, Course_number, Credit_hours and Department.

Grad_Report includes Student_number, Student_identifier and Grade.

Characteristics of the Database Approach. (Traditional File Processing v/s Database approach)

1. Data Redundancy & Inconsistency : In file processing each user defines and implements the file needed for specific applications which are likely to have different formats and the programs may be written in several programming languages. Over and above the same piece of information may be duplicated in several places. (Data Redundancy). This results in wasted storage space and redundant efforts to maintain common data up to date. This may lead to data inconsistency.

2. Self Describing nature of a database System : This is the fundamental characteristic, where the database system contains not only the database, but also the complete definition or description of the database ; This definition is stored in the system's catalog and is called metadata. This contains information such as the structure of each file, type, storage format for each data item and various constraints on the data. catalog is shown in fig 1.3 in the next page.

3. Insulation b/w Programs and Data (Data Abstraction) :

In file ~~prog~~ processing system, structure of data files is embedded in the access programs. So any changes to the structure of a file may require changing all programs that access this file.

In DBMS access programs are written independently of any specific files. This property is called as "program Data Independence"

4. Support of Multiple views of data :

A database typically has many users, each of whom may require a different perspective or view of the database. A multiuser DBMS must provide facilities for defining multiple views. as shown in the fig 1.5.

5. Sharing of Data and Multiuser Transaction processing :

A multiuser DBMS, as its name implies, must allow the multiple users to access the database at the same time. Meanwhile DBMS does Concurrency Control.

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

Figure 1.3
An example of a database catalog for the database in Figure 1.2.

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
....	....	.....
....	....	.....
....	....	.....
Prerequisite_number	XXXXNNNN	PREREQUISITE

Note: Major_type is defined as an enumerated type with all known majors. XXXXNNNN is used to define a type with four alpha characters followed by four digits.

(a)

TRANSCRIPT

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

(b)

COURSE_PREREQUISITES

Course_name	Course_number	Prerequisites
Database	CS3380	CS3320
		MATH2410
Data Structures	CS3320	CS1310

Figure 1.5
Two views derived from the database in Figure 1.2. (a) The TRANSCRIPT view. (b) The COURSE_PREREQUISITES view.

Actors on the Scene : (Database users) DCM

For a small database, one person is good enough to define, Construct, and manipulate the database. Where as for large database, requires many people to be involved. We call them as actors on the Scene.

- Database Administrators (DBAs)
- Database Designers
- End users
- System Analysts and application programmers.

Database Administrators :

In a database environment, the primary resource is the database itself, and the secondry resource is the DBMS and related software.

→ Administering these resources is the responsibility of the database Administrator (DBA).

→ DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed.

→ DBA is accountable for problems such as security breaches and poor system response time.

Duties of Database Administrator :

- → Managing the resources : Databases and DBMS.
- → Schema definition, storage structure and access method definition.
- → Schema and physical organization modification.
- → Granting authorization to users for accessing the database.
- → Specifying integrity constraints.
- → Acting as liaison with users.
- → Monitoring performance and responding to changes in requirements.

Database Designers :

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structure to store the identified data.

→ Responsible to communicate with all prospective database users.

→ Database designers with help of DBA will interact with each potential group of users and develop a view of the databases that meets the data and processing requirements

→ Also responsible to define the content, the structure the constraints and functions or transactions against the database.

End users :

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

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

Naive or Parametric end users : who constantly querying and updating the database using standard types of queries and updates called Canned transactions that have been carefully programmed and tested.

ex : Bank tellers, Reservation clerks. etc.

Sophisticated end users : includes Engineers, Scientists Business analysts and others who thoroughly familiarize themselves with the facilites of the DBMS so as to meet their complex requirements

Stand-Alone users : Who maintain personal databases by using ready made program packages that provide easy to use menu or GUI based interfaces

Ex : Tax Package user, Lawyer, Doctor etc.

System Analysts and application Programmers :

System Analysts determine the requirments of end users, especially naive / parametric uses and develop specifications that meet these requirments.

Application Programmers : implement these specifications as programs (Testing, debug, documentation)

Workers behind the Scene:

The persons are associated with the design, development and operation of the DBMS software and system environment.

→ These persons are typically not interested in the database content itself.

→ These professionals are called as "workers behind the scene", and they include the following categories

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

DBMS system designers and implementers design and implement the DBMS modules and interfaces as a software package

→ A DBMS is a very complex software system that consists of many components or modules, including modules for implementing the catalog, query language processing, interface processing, accessing and buffering data, controlling concurrency and handling data recovery and security.

Tool developers design and implement tools — the software package that facilitate database modeling and design, database design and improved performance

→ Tools are optional packages that are often purchased separately.

→ They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation and test data generation.

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

Advantages of using the DBMS Approach:

The various advantages of using the DBMS approach are listed below.

1 ⟶ Controlling Redundancy
2. ⟶ Restricting unauthorized accers
3. providing Persistent Storage for program objects
4. Providing Storage Structure and Search techniques for efficient query processing.
5. providing Backup and Recovery
6. providing Multiple users Interfaces.
7. Representing Complex Relationships among data
8. Enforcing integrity Constraints.
9. Permitting Interencing and Action using Rules

Controlling Redundancy:

Redundancy means " Storing the same data multiple-times.

→ This leads to several problems

1. Each operation must be done multiple times because same data is stored at multiple times. this leads to duplication of effort
2. Storage Space is wasted when the same data is stored repeatedly.
3. Files that represent the same data may become inconsistent.

By using DBMS concepts like " Data normalization", view etc, Redundancy is Controlled

Restricting Unauthorized Access :

DBMS restricts unauthorized access by providing "a security and authorization subsystem" which the DBA uses to create accounts and to specify account-restrictions. Then DBMS enforce these restrictions automatically.

Providing Persistent Storage for program objects

Database can be used to provide persistent storage for program objects and data structure. object oriented database systems typically offer data structure compatibility with one or more object-oriented programming languages

Providing storage structure and search techniques for Efficient query processing

Database systems must provide capabilities for efficiently executing queries and updates.

→ DBMS provide specialized data structure and search techniques to speed up disk search for the desired records.

→ Auxiliary files are called 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 based on existing storage structure.

Providing Backup and Recovery :

A DBMS must provide facilities for recovering from h/w or s/w failures. The backup and recovery subsystem of the DBMS is responsible for recovery.

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

→ Both forms-style interfaces are menu driven interfaces are commonly known as "graphical user interfaces(GUI).

Representing Complex Relationships among data :

A DBMS have the capacity 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.

Enforcing Integrity Constraints :

Most database applications have certain integrity constraints that must hold the data

→ A DBMS should provide capabilities for defining and enforcing these constraints

→ There are various constraints like " Referential integrity, Uniqueness constraint, and business rules.

Permitting inferencing and actions using rules :

Database systems provide capabilities for defining deduction rules for inferencing new information from the stored database facts, such system are called deductive database system.

ex: rules, triggers, stored procedure and so on.

Brief History of Database Applications :

Historical overview of Database Applications are explained with technologies adopted in the below Sections

1. Early database Applications using Hierarchical and Network Systems.
2. providing Data Abstraction and Application Flexibility with Relational Databases.
3. Object oriented Applications and the Need of More complex databases.
4. Interchanging Data on the web for E-Commerce using XML.
5. Extending Database Capabilities for New Applications

Early Database Applications using Hierarchical and Network Systems.

Many early database applications maintained records in large organizations such as Corporations, Universities, hospitals and banks.

for ex : In a university application, similar information would be kept for each Student, each Course, each grade record and so on.

→ one of the main problems with early database System was the intermixing of Conceptual relationships with the physical storage and placement of records on disk.

→ Another Shortcoming of early System was that they provided only programming language interface This made it time - Consuming and expensive to implement new queries and transactions.

Providing Data Abstraction and Application Flexibility with Relational Databases :

Relational ~~representation of data somewhat resembles~~ Systems were initially targeted to the same applications as earlier systems and provided flexibility to develop new queries quickly and to reorganize the database as requirements changed. Hence data abstraction and program-data independence were improved.

Object-oriented Applications and the Need for more Complex Databases.

The emergence of object-oriented programming languages in the 1980 and the need to store and share complex, structured objects led to the development of object-oriented databases (OODBs)

OODB incorporated paradigms like abstract data types, encapsulation of operations, inheritance and object identity.

Many object-oriented concepts were incroporated into the newer versions of relational DBMS leading to ORDBMS.

Interchanging data on the Web for E-Commerce using XML

A variety of techniques were developed to allow the interchange of data on the Web. Currently extended Markup language (XML) is considered to be the primary standard for interchanging the data among various types of databases and web pages. XML combines concepts from the models used in document systems with database modeling concepts.

Extending Database Capabilities for New Applications :

The success of database systems in traditional applications encouraged developers of other types of applications to attempt to use them.

→ Such applications traditionally used their own specialized file and data structures.

→ Database systems now offer extensions to better support the specialized requirements for some of these applications.

→ The following are some examples of these applications.

1. Scientific
2. Storage and retrieval of images
3. Storage and retrieval of videos
4. Data mining applications
5. Spatial applications
6. Time Series applications

Databases vs Information Retrieval :

The traditional database technology applies to structured and formatted data that arises in routine applications in government, business and industry.

→ Database technology is heavily used in manufacturing, retail, banking, insurance, finance and health care industries.

→ "An area related to database technology is Information Retrieval (IR) which deals with books, manuscripts and various forms of library-based articles.

→ Data is indexed, cataloged and annotated using keywords.

When Not to use a DBMS (Disadvantages of DBMS)

There are a few situations in which a DBMS may involve unnecessary overhead costs. 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.

DBMS Not be used under the following circumstances -

1. Simple, well defined database applications that are not expected to change at-all.
2. Stringent, real time requirments for some application programs that may not be met- because of DBMS overhead.
3. Embedded system with limited storage capacity
4. No multiple user access to data.

Overview of Database languages and Architecture :

Contents :

1. Data Models, Schemas and Instances.
2. Three Schema architecture and data independence
3. Database languages and Interfaces
4. Database System environment.

Data Models :

A Collection of Concepts that can be used to describe the Structure of a database.

Data Model Structure and Constraints :

These are the constructs that are used to define the database Structure, typically include elements as well as group of elements and relationship among such groups.

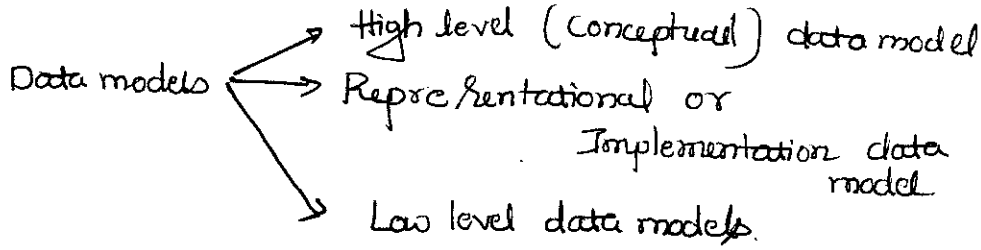
Data Model operations :

These operations are used for Specifying database retrievals and updates by referring to the Constructs of the data model.

- Operations on the data model may include basic model operations (e.g. generic insert, delete, update) and user defined operations (e.g. Compute_Student_gpa, update_inventory).

Categories of Data Models

Catagories of Data Models are represented with following notation.



Conceptual data Models : provide concepts that are close to the way many users perceive data. (either entity based or object-based data models)

→ Uses the concepts such as entity attribute, relationship ⇒ Entity - Relationship model.

→ An entity represents a real world object-
Ex: Employee, Book or project-

→ An attribute represents some property of interest-
Book: Book_id, Title, Author, Cost, publication etc.
Entity

→ Ex: Works-on relationship between an employee and a project-

physical data models :

provide concepts that describe details of how data is stored in the Computer. Includes record formats, record ordering, access paths and file organizations etc.

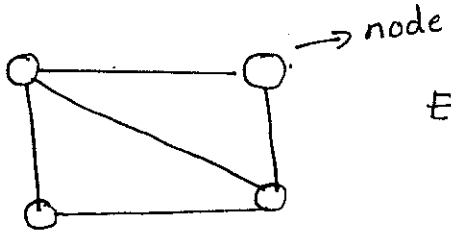
Implementation Data Model (Representational Data Model)

Representational data model includes relational, network and hierarchical data models.

Relational :

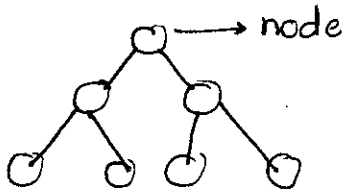
Sid	Sname	age
1	Anil	20
2	Sachin	22
3	Rahul	21

Network data Model :



Each node represents data

Hierarchical data Model : (Tree Structure)



" Node Represents data "

Database Schema (Define database schema & with diagram)

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. a schema diagram is shown in the fig below.

Figure 2.1
Schema diagram for the database in Figure 1.2.

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Database State or Snap Shot :

The data in the database at a particular moment-in-time is called a database state or Snapshot

→ Database state is also called as "Current-set-of occurrences or instances in the database.

Distinction between Database Schema and Database State.

Database Schema	Database State
1. Changes very frequently	1. Changes as and when database is updated
2. Schema is called intension	2. State is also called extension

Note : DBMS Stores the description of Schema in the System catalog.

"Data Stored at System catalog is called "meta data"

Three - Schema Architecture : (Levels of Abstraction in a DBMS)

Three Schema architecture is illustrated in the below fig. is to Separate the user application from the physical

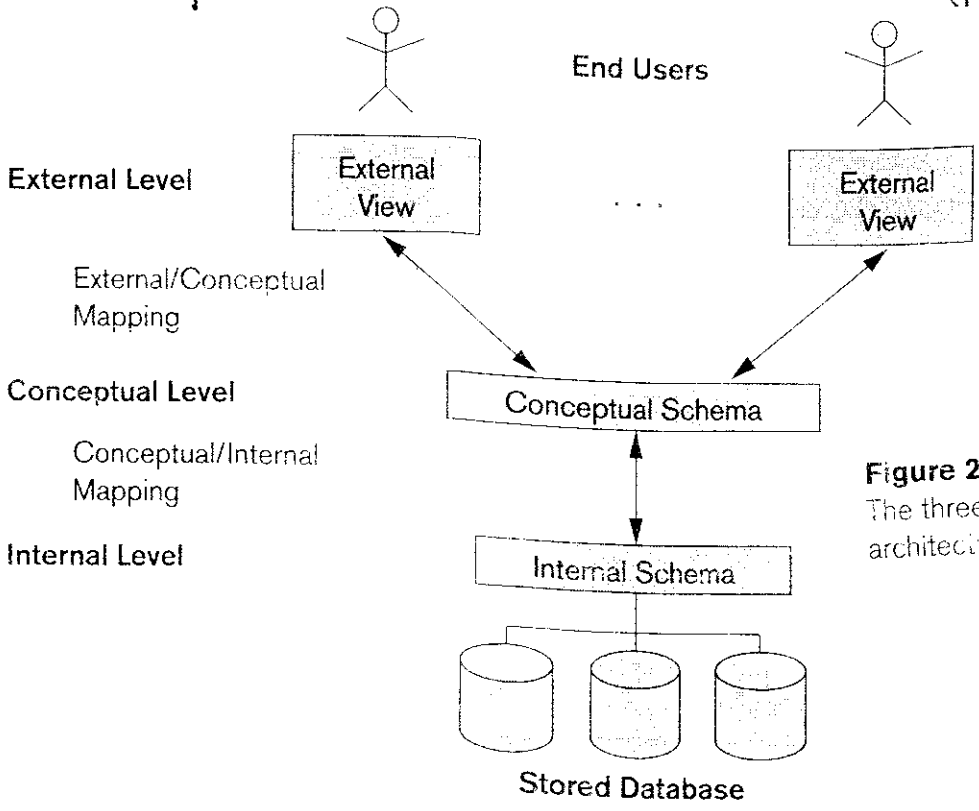


Figure 2.2
The three-schema architecture.

In the above 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. Conceptual level has Conceptual schema, which describes the structure of the whole database for a community of users. The Conceptual schema hides the details of physical storage structure and concentrates on describing entities, data types, relationships, user operations and constraints. This implementation Conceptual schema is often based on a conceptual schema design in a high level data model.

3. External or view level: 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.

Data Independence:

Data independence is the ability / capacity to change the schema at one level of a database system without having to change the schema at the next higher levels.

Logical Data Independence:

The capacity to change the conceptual schema without having to change the external schema and their associated application program.

Note: We may change the conceptual schema to expand or reduce the database

Physical Data Independence :

The capacity to change the internal Schema without having to change the conceptual Schema

Ex: Internal Schema may be changed when certain file Structures are reorganized or new indexes are created to improve database performance

DBMS Languages : Various languages are Supported by DBMS are listed below.

1. DDL (Data Definition Language)
2. DML – Data Manipulation Language
3. DCL – Data control Language
4. SDL – Storage Definition Language } part of DDL
5. VDL – View Definition Language }

Data Definition Language (DDL)

Used by the DBA and database designers to Specify the conceptual Schema of a database.

→ DDL is also used to define internal and external Schemas.

→ Storage Definition Language (SDL) and View Definition language are used to define internal and external Schemas.

→ Supports CREATE, ALTER and DROP Commands.

Data Manipulation Language (DML)

Data Manipulation lanayuage mainly used to insert, delete, update and retrieve the values.

→ There are two types of "Data Manipulation languages"

1 High level or Non-Procedural Languages: These include the relational language SQL. May be used in a Standalone way or maybe embedded in a programming language

2. Low level or Procedural Languages : These must be embedded in a programming language, used to Specify database retrievals and updates

→ DML Supports INSERT, DELETE, UPDATE and SELECT Commands.

Data Control Language (DCL)

Used by the DBA, Supports Commands to control the transactions granting the privileges to other users and revoking of the Privileges.

→ DCL Supports COMMIT, ROLLBACK, GRANT, REVOKE and SAVE TO

DBMS Interfaces : DBMS Interfaces are listed below.

1. Menu Based Interfaces for web clients or Browsing.
2. Forms – Based Interfaces
3. Graphical user Interfaces
4. Natural Language Interfaces
5. Speech Input and Output.
6. Interfaces for parametric users
7. Interfaces for the DBA

Menu based Interfaces for Web Clients or Browsing : The interfaces present the user with list of options (called menus) that lead the user through the formulation of a request. pull down menus are a very popular technique in web-based user interfaces. They are also often used in browsing interfaces.

Forms Based Interfaces : A form based interface displays a form to each user. User can fill out all of the form entries to insert new data, or they can fillout 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 Interface : A GUI typically displays a schema to the user in diagrammatic form. The user can then specify a query by manipulating the diagram. In many case GUI

Natural Language Interfaces : These interfaces accepts request 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 dictonary 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 interupt the request. If the interpretation is successful the interface generates a high level query corresponding to the natural language request and submits it to the DBMS for processing ; otherwise a dialogue is started with the user to clarify the request.

Speach input and output : Limited use of speach as an input query and speach as an answer to a question or result of a request is becoming common place. Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure and credit card account information are allowing speach for i/p and o/p to enable customers to access this information.

Interfaces for Parametric users : Parametric users such as bank tellers, often have a small set of operations that they must perform repeatedly.

→ System analysts and programmers design and implement a special interface for each known class of naive users.

→ Keystroke or touch screen types of interfaces are introduced.

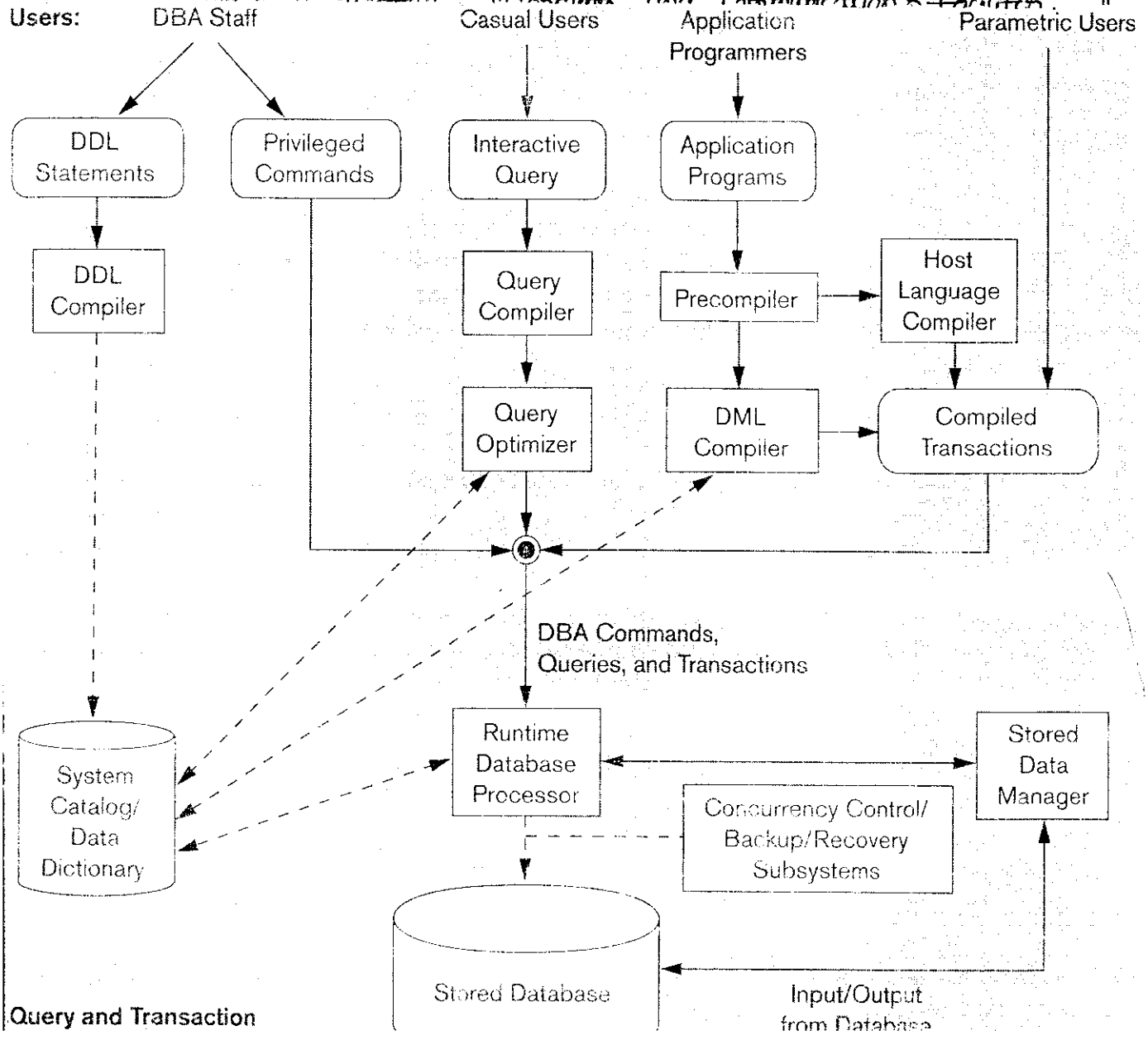
Interfaces for the DBA:

Most database systems contail privileged commands that can be used only by the DBA Staff. These includes Commands for creating accounts, Setting System parameters, granting account authorization, Changing a Schema and recoganizing the Storage Structure of a data base.

Database System environment:

Database System environment is ~~explained~~ divided into following Section.

1. DBMS Component Modules 2. Database s/y utilito 3. Tools, applicaton environment
2. Database System Utilities
3. Tool, application environments and Communication Facilites.



The above fig 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.

→ The database and the DBMS catalog are usually stored on disk

→ Access to the disk is controlled primarily by the operating system (OS).

→ A high level stored data manager of the DBMS controls access to DBMS information i.e. stored on disk.

→ DDL Compiler processes schema definitions specified in the DDL and stores description of the schema in the DBMS catalog with DBMS

→ Casual users interacts through interactive query interface.

→ These queries are parsed, analyzed and validated for correctness of the query syntax.

→ Query Compiler compiles these queries into internal form.

→ Query optimizer is concerned with the rearrangement and possible reordering of operations, elimination of redundancies and use of correct algorithms and indices during execution.

In the lower part of fig, the runtime database processor executes

1. privileged commands 2. the executable query plans
3. canned transactions it works with system dictionary

Client program can access the DBMS running on a separate computer from the computer on which database resides.

Data base Server can manage several Client Programs

Database System utilities:

DBMS have database utilities, that help the DBA in managing the database System. Common utilities have the following types of functions.

Loading: A loading utility is used to load existing data files such as text files or sequential files into the database.

BACKup: A backup utility creates a backup copy of the database usually by dumping the entire database onto tape or other mass storage medium.

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: Such a utility monitors database usage and provides statistics to the DBA.

Tools, Application Environments and Communications Facilites:

A tool that can be quite useful in large organization is an expanded data dictionary.

→ Design decisions, usage standards, application program descriptions and user information is stored at information repository

Application development environments such as powerBuilder (Sybase) or JBuilder (Borland) have been quite popular. These systems provide an environment for devdping database applications.

DBMS also needs to interface with communication software, whose function is to allow users at locations remote from the database system bit to access the database through Computer terminals, workstations or personal Computer.

Conceptual Data Modeling using Entities and Relationships:

This section covers the ~~topics like~~ following topics

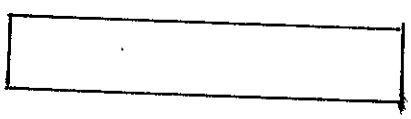
1. Entity
2. Attributes
3. Composite vs Simple Attributes
4. Single valued vs Multivalued attribute
5. Stored vs derived attribute
6. Entity set
7. Entity type
8. Keys
9. Value sets
10. Relationship
11. Relationship type
12. Relationship sets
13. Roles
14. Structural Constraints
15. Weak Entity types
16. Specialization and Generalization
17. Notations of ER diagram.
18. Refining the ER design for the Company Database

Entity: An Entity is a real world object with an independent existance.

An entity may be an object with a physical existance -
ex: person, car, house, employee, student

- Or with a conceptual/logical existance or events
ex: company, department, course

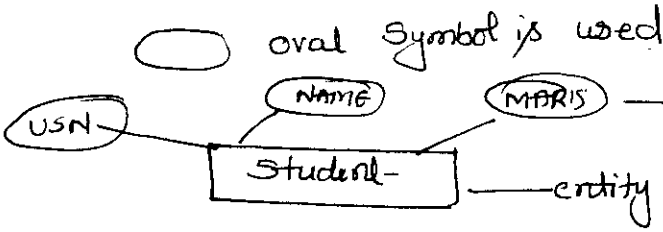
→ Entity is represented by rectangle.



Attribute: Property of an entity is called Attribute

ex1: Employee entity may be described by the employee's name, age address and salary.

ex2: Student entity may be described by the Student's name, USN, marks etc



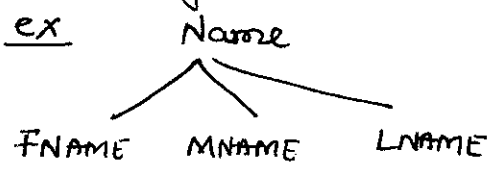
Different types of attributes are listed and explained below.

1. **Simple Attributes**: Attributes that are not divisible are called Simple or automic attributes.

   ex: first name, last name, middle name

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

   ex



3. Single valued Attribute: Most attributes have a single value for a particular entity " Such attributes are called Single valued"

   ex: age, gender

4. Multivalued Attribute: Attribute has more than one value for a particular entity.

   ex1: phone No (one person can have more than one phone connections.

   ex2: degree (one person can have one or more degrees.

Stored Attribute : Attribute doesn't require updation.

ex1. DOB → Date of birth

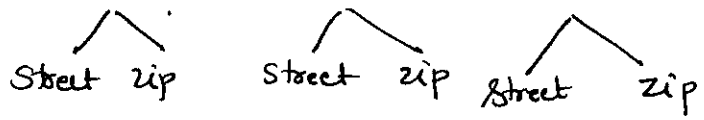
ex2: DOJ → Date of joining

Derived Attribute : The value of the attribute can be derived from other attribute.

ex: Age can be derived from DOB.

Complex Attribute : Collection of both multivalued & Composite attributes.

ex: Address1, Addr2, Address 3



Key Attribute :- Attribute which uniquely identifies an entity in the entity set

ex: usn attribute in the Student entity

ex2: SSN attribute in the employee entity.

Entity, Entity Type and Entity Set :

Entity : An Entity is a real world object with an independent existance.

Consider the following Relation.

Student

USN	SNAME	SMARKS
01	Anil	40
02	Kumar	50
03	Rama	45
04	Amith	60

Entity type

Entity Set

→ Entity instance (value given to a particular entity)

Entity type : Name given to the Set of Common entities.
OR " Refers to the Collection of entity that Share a Common definition

→ For above Relation entity type is Student

Entity Set : Set of entities of Same entity type that Shares the Same attribute. Entity Set is denoted in the figure.

Key Attribute of an Entity type :

It is an important Constraint on the Entities of an entity type is the key or the uniqueness Constraint on attributes.

→ An entity type usually has one or more attribute whose values are unique / distinct from each individual entity in the entity set. Such attribute is called key attribute.

ex : Consider person and Company Relation.

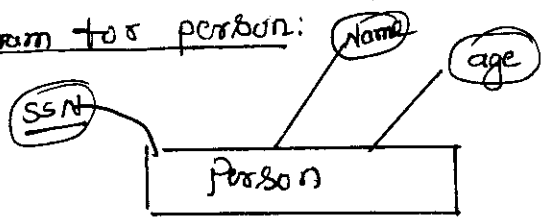
Key attribute

SSN	Name	Age

Key attribute

Comp_name	Location

ER diagram for person:



Value Set (Domain) 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.

Consider the fig below.

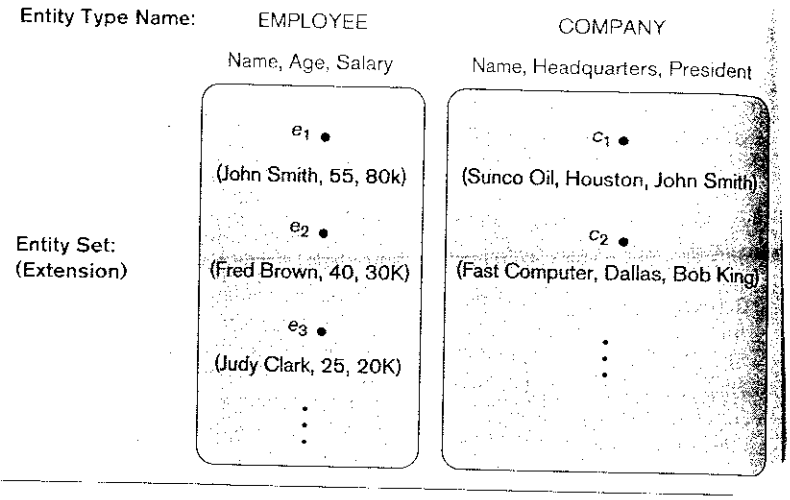


Figure 7.6
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

↳ We can (Database designers) can specify the value set of Age attribute of Employee to be the set of integer numbers between 16 and 70.

↳ We can also specify the value set for the Name attribute to the set of strings of alphabetic characters separated by blank characters and so on.

↳ value sets are not displayed in ER diagram.

↳ Mathematically an attribute A of entity set E whose value set is V can be defined as a function from E to the power set $P(V)$ of V

$$A : E \longrightarrow P(V)$$

Initial Conceptual Design of the COMPANY Database :

Before moving to Conceptual Design, it is necessary to know requirements of Company database. Requirements are listed below.

1. The Company is oraganized into departments. Each department has a unique name, a unique number and particular employee who manages the department.

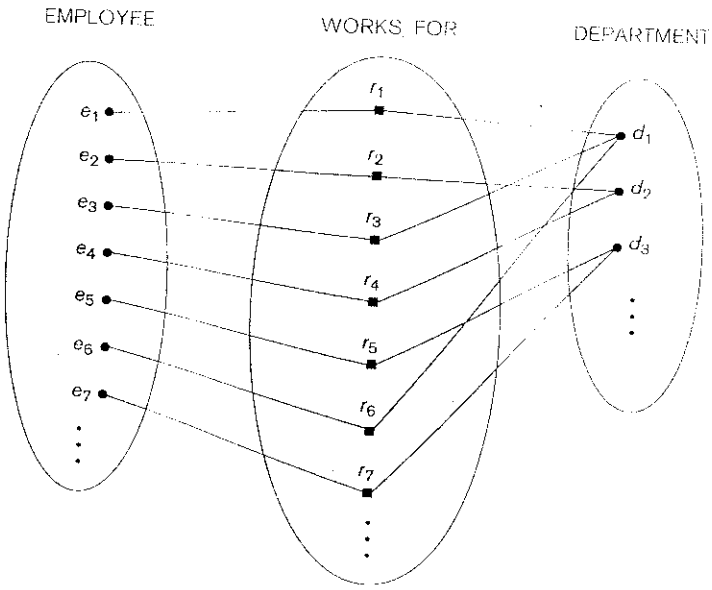


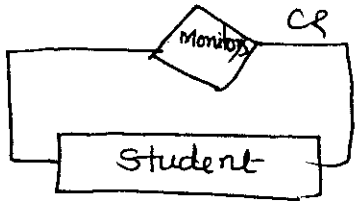
Figure 7.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Degree of a Relationship Type : The degree of a relationship type is the number of participating entities.

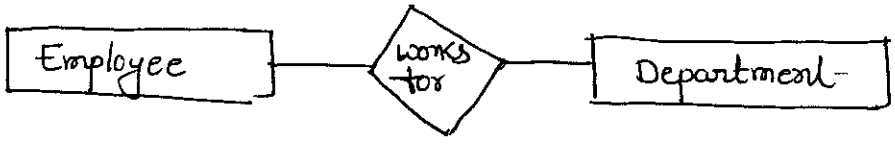
Degrees of a relationship type are named as

1. Unary
2. binary
3. ternary
4. n array.

Unary : Unary relationship exists within one relation.



Binary. In Binary relationship no of participating entities are two.



Ternary relationship : No,. of participating entities will be three.

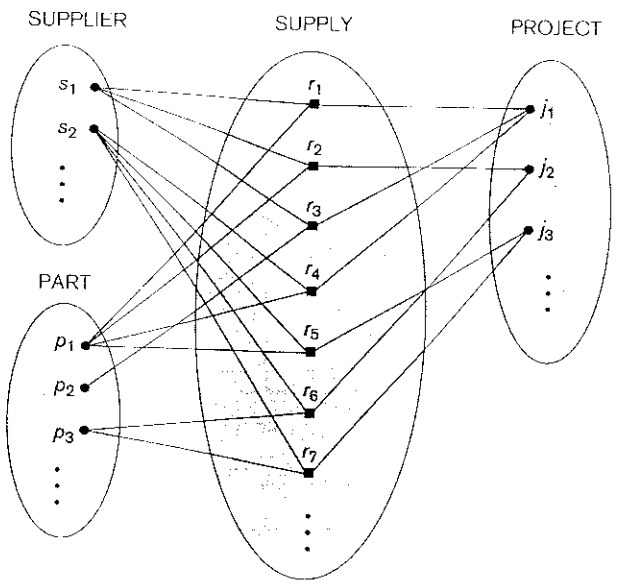


Figure 7.10
Some relationship instances in the SUPPLY ternary relationship set.

Note : Relationship can also have attributes.

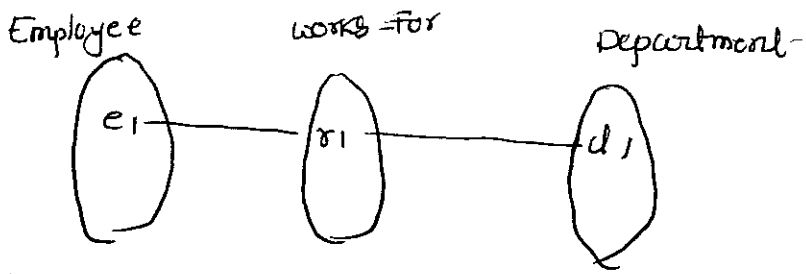
Role Names :

Each Entity type that participates in a relationship type plays a Particular role in the relationship. The role-name signifies the role role that a participating entity from the entity type plays in each relationship instance & helps to explain what the relationship means

for ex : Employee in the WORKS_FOR relationship type Employee plays the role of employee or worker and department plays the role of department or employer.

ex: Employee department

e1 WORKS FOR D1



e1 is an employee works for d1 department.

Recursive Relationships :

Some 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. The following fig shows Recursive Relationships.

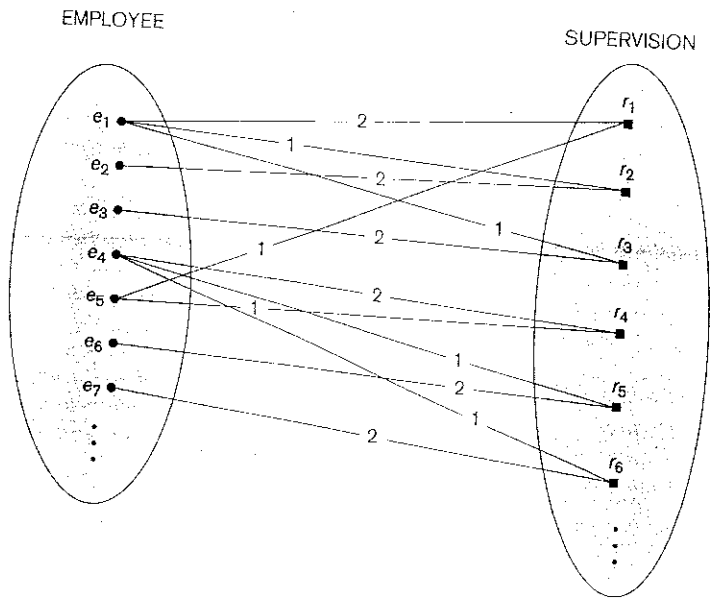


Figure 7.11
A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2).

The Supervision relationship type relates an employee to a supervisor, where both employee and supervisor entities are members of same Employee Set.

Constraints on Binary Relationship Types :

Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set.

Normally company has a rule that each employee must work for exactly one department.

W.R.t to above constraint we can distinguish two main types of binary relationship constraints

1. Cardinality Ratio
2. Participation.

Cardinality Ratio : The cardinality Ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in.

→ The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1 and M:N.

1:1 Cardinality Ratio, binary relationship is MANAGES which relates a department entity to the employee who manages that department.

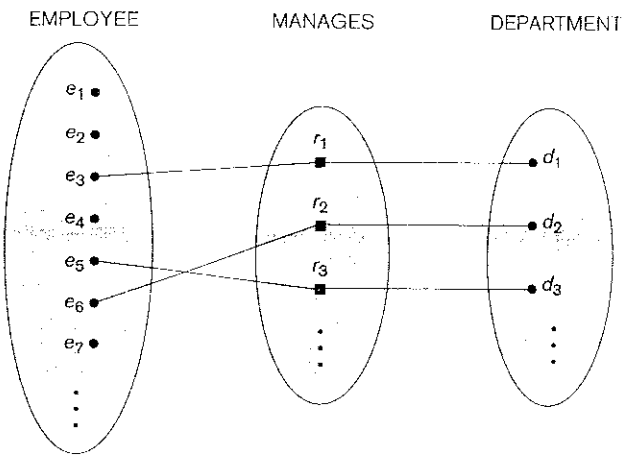
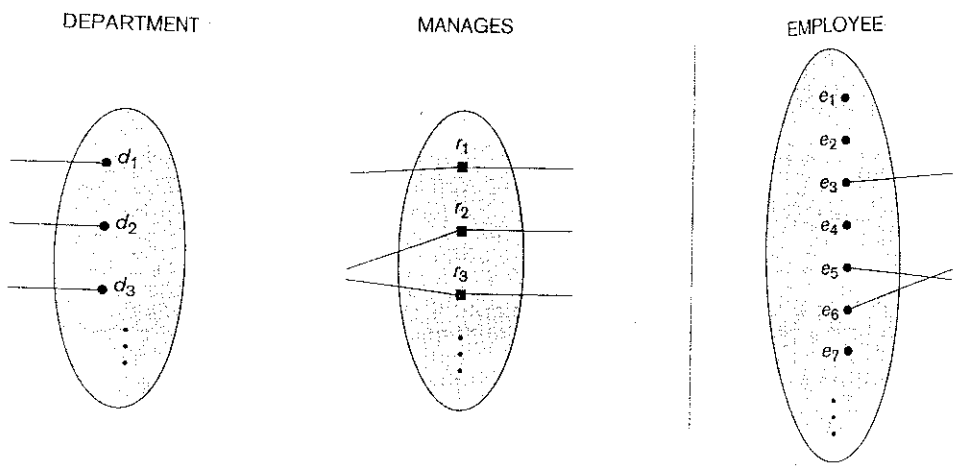
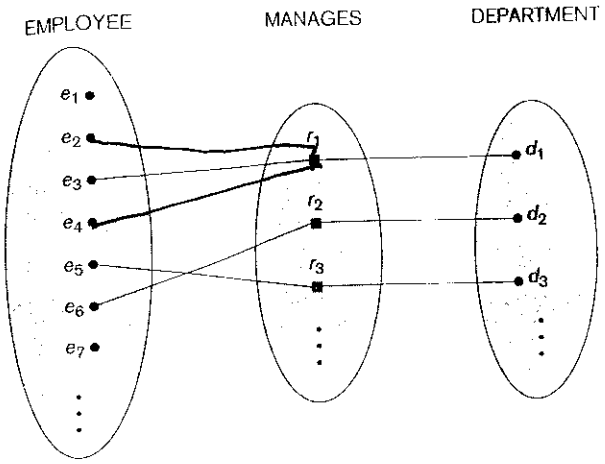


Figure 7.12
A 1:1 relationship, MANAGES.

1:N Cardinality Ratio, binary relationship one department consists of several employees.



N:1 CR N employees works for 1 department.



M:N CR . binary Relationship, M employees works_on N department.

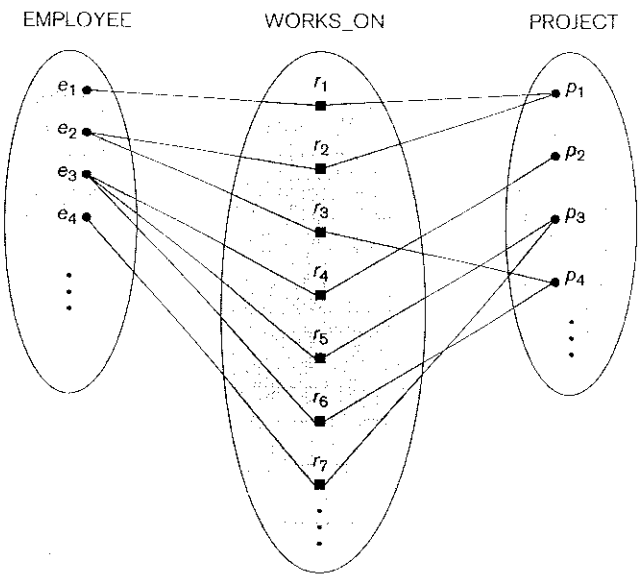


Figure 7.13
An M:N relationship, WORKS_ON.

Participation Constraints and Existance: The participation Constraint specifies whether the existance of an entity depends on its being related to another entity via the relationship type.

⟶ This constraint specifies minimum number of relationship instances that each entity can participate in and is some times called the minimum Cardinality ~~Ratio~~ Constraint.

⟶ There are two types of participation Constraints

1. total participation
2. partial participation

Total participation :

Total participation is also called "existance dependency"

Consider the situation " If a company policy states that every employee must work for a department then an employee entity can exist only if it participates in at least one works_FOR relationship instance

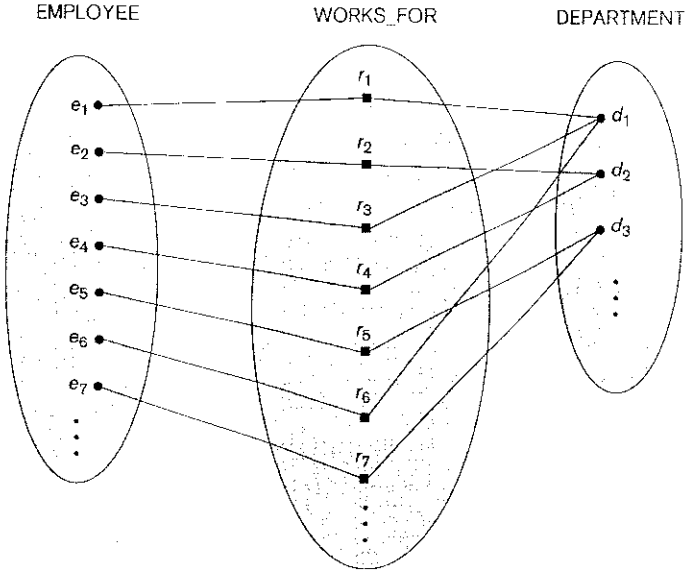


Figure 7.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

fig 7.9

Thus the participation of Employee in works_FOR is called total participation.

Partial participation : Every employee will not manage a department, so the participation of Employee in the MANAGES relationship type is partial, The following fig shows partial participation.

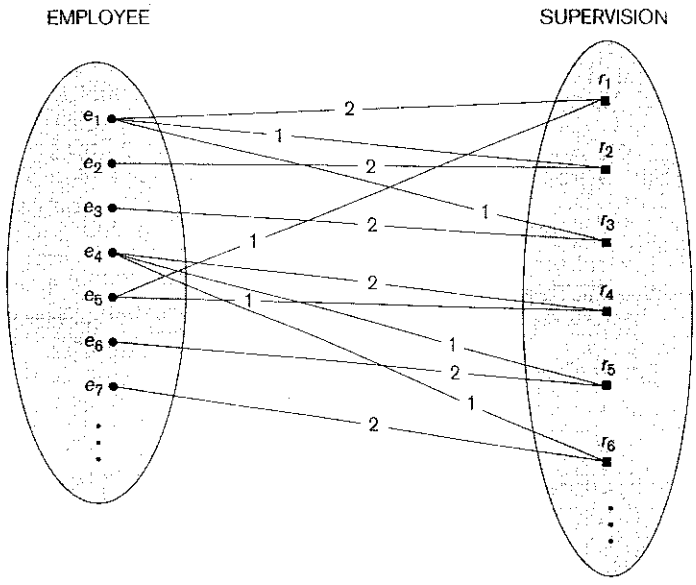
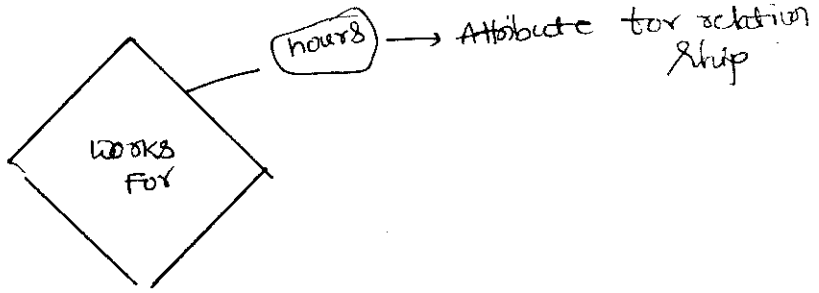


Figure 7.11
A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate* role (2).

Note: Cardinality Ratio and Participation Constraints together called "Structural Constraints" of a relationship type.

Attributes of Relationship types :

Relationship types can also have attributes, similar to those of entity types. For example, to record the numbers of hours per week that an employee works on a particular project. The following fig shows attribute for relationship.



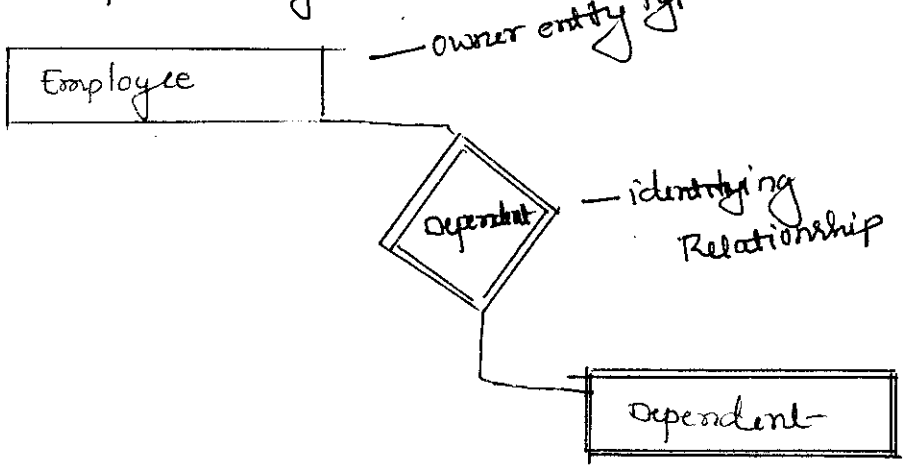
Weak Entity Types :

Entity type that do not have key attributes of their own are called weak entity types. Regular entity types that do have a key attribute — which include all the standards are called a strong entity type.

→ 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. The identifying entity type is called owner entity type.

→ Relationship type that relates a weak entity type to its owner entity type is called "Identifying relationship"

→ The following diagram shows weak entity type and above explained things.



Note: A weak entity type normally has a partial key

ER Diagrams, Naming Conventions :

The ER diagrams and their conventions are given below.

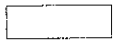
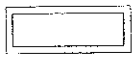
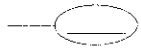
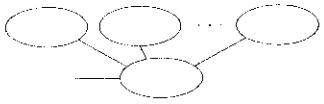
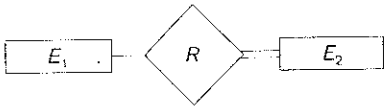
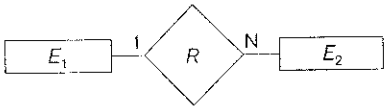
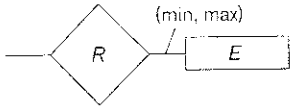
Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for E_1:E_2 in R
	Structural Constraint (min, max) on Participation of E in R

Figure 7.14
Summary of the notation for ER diagrams.

ER Diagram Examples :

Ex1 : ER diagrams for the Company Schema with Structural Schema, with Structural Constraints Specified using (min, max) notation and role name.

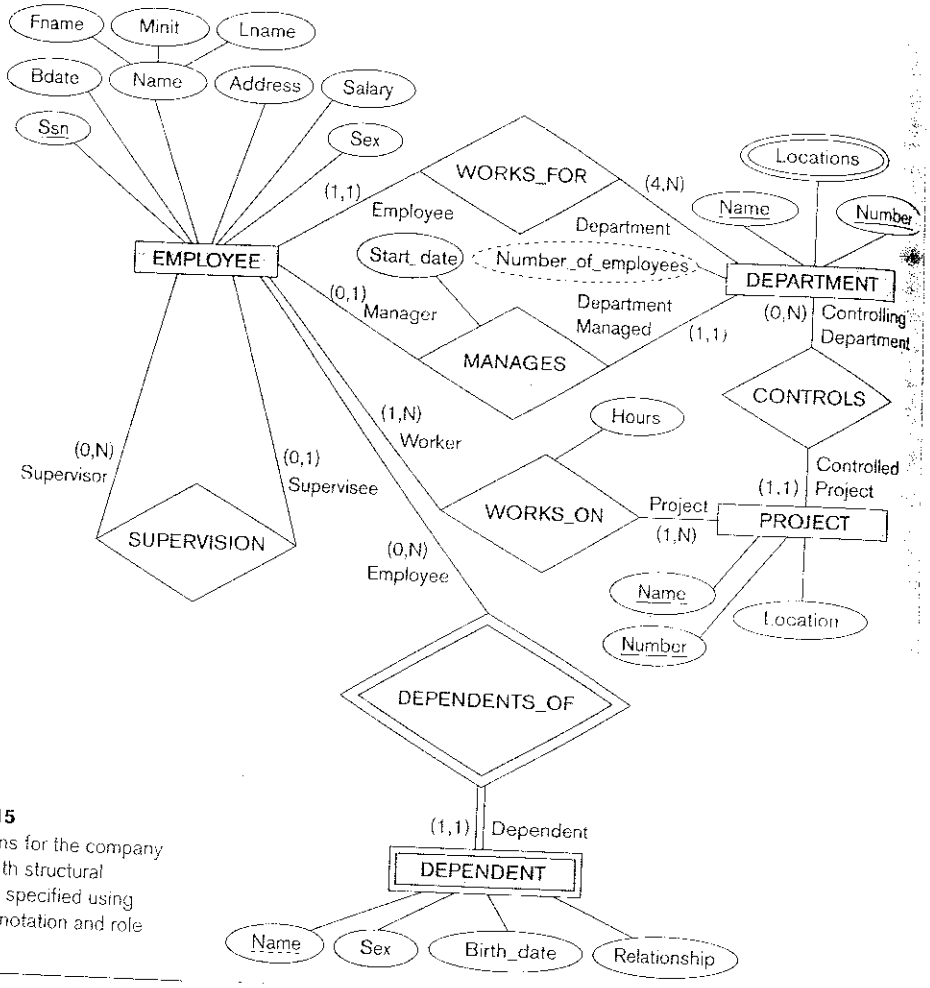
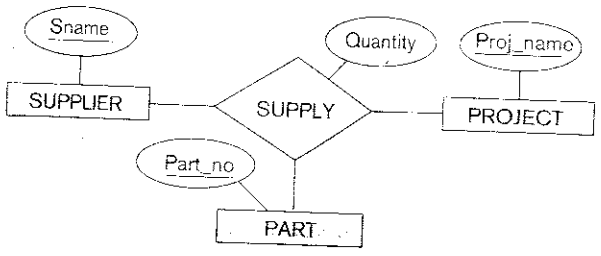


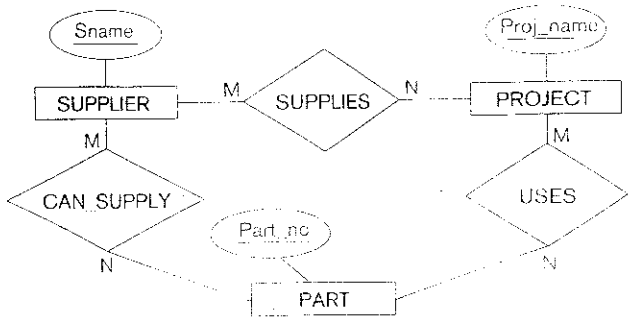
Figure 7.15
ER diagrams for the company schema, with structural constraints specified using (min, max) notation and role names.

Ex2 : The Supply relationship.



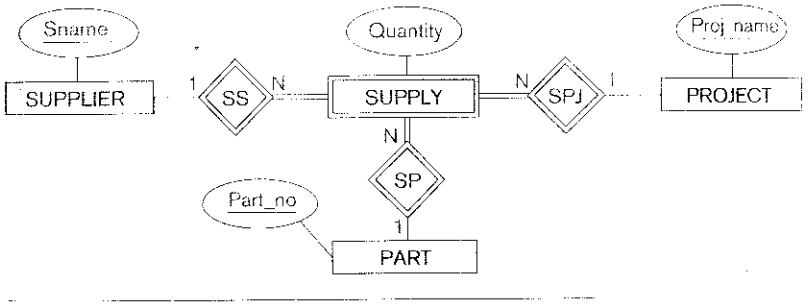
Ex3: Three binary relationships

(b)



(c)

Ex4: Supply Represented as "Weak entity type"



Specialization and Generalization in ERR:

Specialization: Specialization is the process of defining a set of subclass of an entity type, this entity type is called the Superclass of the Specialization.

→ The set of subclass that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass.

→ For example the set of subclass {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the Job type

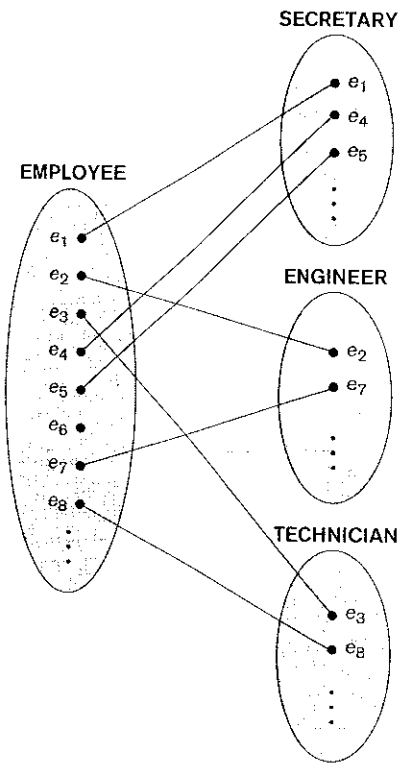


Figure 7.20
Instances of a specialization.

→ Specialization process allows us to do the following

- Define a set of subclasses of an entity type
- Establish additional specific attributes with each subclass
- Establish additional specific relationship types between each subclass and other entity

Generalization :

Generalization is a process, in which differences among several entity type are suppressed, and generalize them into a single superclass.

ex: Consider entity type CAR and TRUCK because they have several common attribute, they can be generalize into the entity type Vehicle as shown in fig below.

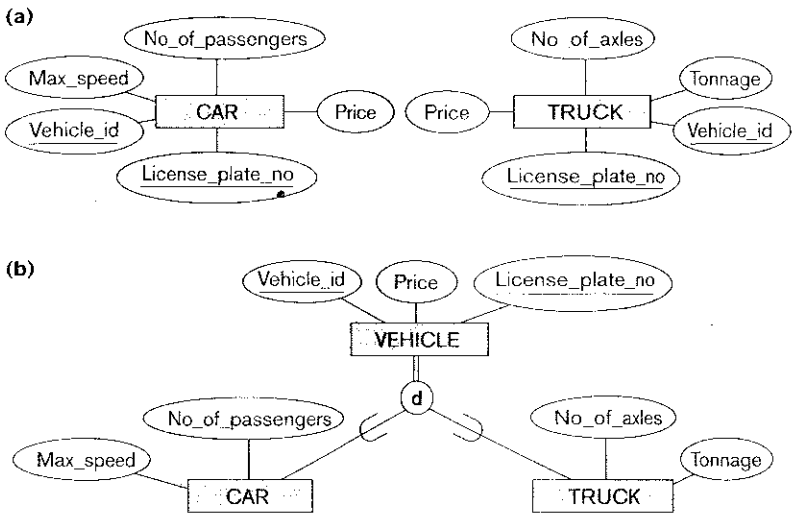


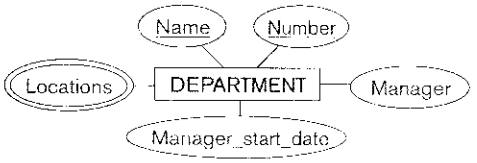
Figure 7.21
Generalization. (a) Two entity types, CAR and TRUCK. (b) Generalizing CAR and TRUCK into the superclass VEHICLE.

2. A department controls a number of projects, each of which has a unique name, a unique number and a single location.

3. ~~This~~ It is necessary to store employee name, Social Security number, address, salary, sex and birthdate. An employee is assigned to one department but may work on several Projects.

4. It is necessary to keep track of the dependents of each employee for insurance purposes.

Conceptual design of company database is given below.



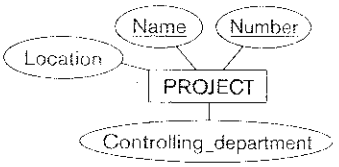
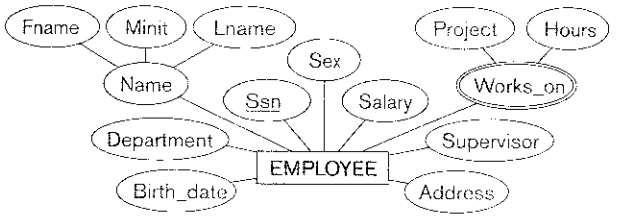
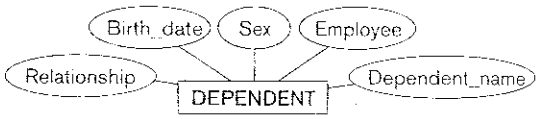


Figure 7.8
Preliminary design of entity types for the COMPANY database. Some of the shown attributes will be refined into relationships.





Data base designers identified ~~four types~~ entity types — one corresponding to each of the ~~four~~ items in the specification.

1. An entity type DEPARTMENT with attributes Name, Number, Locations, Manager, and Manager_start_date. Locations is the only multivalued attribute.

2. Any entity type PROJECT with attribute Name, Number, Location and Controlling-department

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.

Relationship Types, Relationship Sets, Roles and Structural Constraints

This heading covers topics like

1. Relationship types, Sets and instances.
2. Relationship degree
3. Role Names
4. Recursive Relationships
5. Constraints on Binary Relationship Types
6. Cardinality Ratios for Binary Relationships
7. Participation Constraints and Existance dependencies
8. Attributes of Relationship types.

Relationship type: A relationship type R_t among 'n' entities $e_1, e_2, \ldots e_n$ indicates / defines set of associations.

Relationship set: A relationship set R_s is a set of relationship instances.

Relationship instance: where each r_i associates n individual entities $(e_1, e_2 \ldots e_n)$.

The following diagram shows some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Module 2

Relational Model

KARANAM SUNIL KUMAR
Asst. Prof, Dept of CSE
RNSIT
ph: 9972995720
Email: SunilKaranam72@gmail.com

CONTENTS:

1. Relational Model Concepts
2. Relational Model Constraints and relational Model - - Schema
3. Update Operations
4. Dealing with Constraint- violations.

Relational Algebra

1. Unary and Binary relational operations
2. Additional relational operations (Aggregate, grouping etc)
3. Examples of Queries in relational algebra.

Mapping Conceptual Design into a Logical Design:

1. Relational Database Design using ER-to-Relational- - Mapping

SQL:

1. SQL data definition and data types
2. Specifying Constraints in SQL
3. Retrieval queries in SQL
4. Insert, Delete and Update Commands
5. Additional features of SQL

Relational Model Concepts :

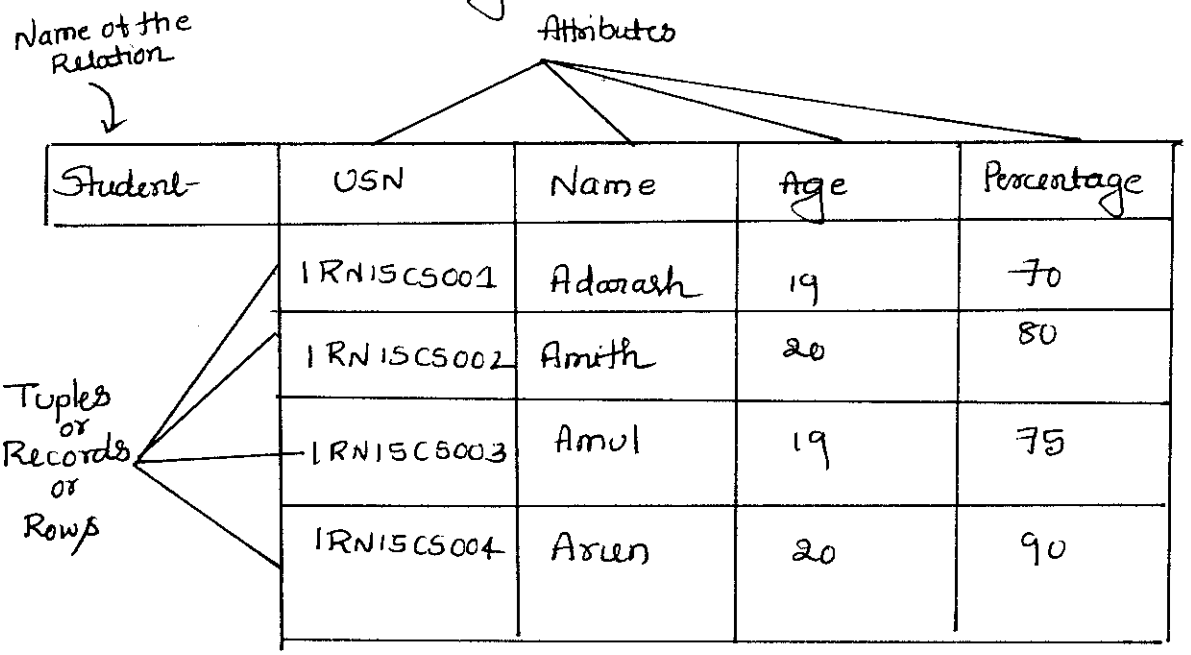
Relational Model is based on the concept of a Relation.

Relational model represents database as a Collection of Relations.

→ Relation is a mathematical concept based on idea of Set.

→ This Model was proposed by Dr E.F Codd. of IBM in 1970

Consider the relation given below :



Student	USN	Name	Age	Percentage
	1RN15CS001	Adarsh	19	70
	1RN15CS002	Amith	20	80
	1RN15CS003	Amul	19	75
	1RN15CS004	Arun	20	90

Domain of a Attribute, Tuple, Relation, Relation State, Relation Schema and Relational database Schema :

Domain : A domain 'D' is a Set of automatic values.

or

It is a Set of permitted values.

Domain of a Attribute : Set of permitted values to attribute.

ex Name : A-Z, a-z (datatype is varchar)

age : 0-9 (datatype is integer)

Tuple : An ordered set of values belongs to a particular entity.

Relation : A Relation may be regarded as a "Set of tuples"

Relation State : Represents set of value types

$$r = \{t_1, t_2, t_3\}$$

Relation Schema :- denoted by $R(A_1, A_2 \ldots A_n)$ is made up of a relation name R and a list of attributes $A_1, A_2 \ldots A_n$.

Relational database Schema : Is represented by

$$S(R_1, R_2, \ldots R_n) \text{ where}$$

$S \longrightarrow$ Schema

$R_1, R_2 \ldots R_n$ list of Relations.

Keys in DBMS :

DBMS Supports various Keys. The Keys are listed below.

1. Primary Key
2. Candidate Key
3. Super Key
4. Foreign Key
5. Alternate Key

Candidate Key : Minimum set of attributes used to uniquely differentiate records of the relation. Consider the relation given below.

Sid	Sname	Marks
S1	A	40
S2	A	40
S3	B	50
S2	B	50

Redundancy

Redundancy

Redundancy : Existance of valve more than once in a attribute

So, In the above relation the candidate Key is

S1 A
S2 A
S3 B
S4 B

∴ <Sid, Sname> is the Candidate Key which participate in candidate Keys are called prime Attributes.

Consider the relation given below.

Sid	Sname	Marks
S1	A	30
S2	B	20
S3	A	20
S4	C	50

In the above Relation Sid Alone acts a Candidate Key

Primary Key : One of the candidate key which follows the following rules.

1. No NULL values are allowed
2. For a Relation, Almost one primary Key is allowed.

Ex: Considers the below given relation.

Cust_id	Bank Name	Cust_ name
1	SBI	A
2	CANARA	B
3	SBI	A
4	vijaya	D

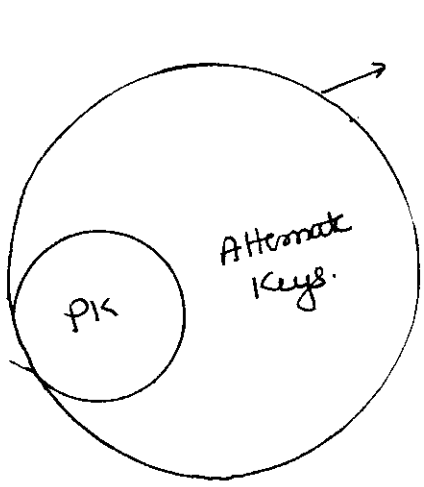
Cust_id is the primary Key

Note: Candidate Key with 1 attribute acts as a "Primary Key".

Alternate Key :

All the candidate key except primary key are known as "alternate key"

→ Venn diagram explains the concept of alternate key in the below section.



Venn diagram to explain the concept of alternate keys

Properties of Candidate Keys

1. NULL values are allowed.
2. Relation may consists of more than one "Alternate Keys"

Consider the Relation given below :

Sid	Bank_Account	Name	Voter_id
1	Acc 1	A	V1
2	Acc 2	B	V2
3	Acc 3	A	V3
4	Acc 4	B	V4
5	Acc 5	A	NULL

In the above Relation :

Primary Key → Sid

Candidate Keys → Sid, Bank_Account, voter_id

Alternate_Keys → Bank_Account, voter_id

Note : Name is not a alternate key because it is not candidate ...

Super Key :⇒ Candidate Key + Zero or More Attributes.

Minimal Super Key is the Candidate Key.

Consider the relation given below.

Emp_id	Salary	Emp_name
1	40,000	A
2	50,000	C
3	40,000	B
4	50,000	A

In the above relation emp_id is the candidate key, also acts as primary key and Super key

Super keys → emp_id

emp_id, Salary

emp_id, emp_name

Emp_id, Salary, emp_name

Foreign Key : It is the attribute of one relation which refers to Primary Key attribute of another relation.

Properties of Foreign Key :

1. Redundancy is allowed
2. Value of the foreign key should be within primary Key values.

Example for Foregin Key with a Relation :

Primary key

Depart	dno	dname
	1	CS
	2	IS

Foreign key

Employee	Emp-id	Emp-name	Emp-sal	Dno
	1	Adarash	15,000	1
	2	Amith	16,000	2
	3	Arun	17,000	1

Relational Data Model Constraints :

The constraints are conditions that must hold on all valid relation instances.

→ Different relational data model Constains are listed below.

1. Domain Constraint-
2. Key Constraint-
3. Entity Integrity Constraint-
4. Referential Integrity Constraint-

Domain Constraint-: When database designers or users are trying to insert a value is not within the range of set of permitted values.

Consider the relation given below.

Sid	Sname	Smarks
1	A	90
2	B	60
3	C	70

Insert into table values (ABC, 'D', 70) violates Domain Constraint ∴ sid domain type is integer, and set of Permitted values 0 — 9

2. Key Constraint :

Key Constraint is denoted by $t_1[SK] \neq t_2[SK]$

Consider the below given relation.

Sid	Sname	Smarks
1	A	90
2	B	60
3	C	70

In $t_1[SK] \neq t_2[SK]$ Notation

↓ tuple ↳ SuperKey

In the above relation Consider Sid as Superkey.

$t_1[SK]$ $t_2[SK]$

$1 \neq 2$ and

$2 \neq 3$

Definition :- A Set of attributes in SK of relation R such that no two tuples in any valid instance of a relation will have same value for SK

---

3. Entity Integrity Constraint :

Primary Key value should not be null.

$t[PK] \neq NULL$

---

4. Referential Integrity : The value of Foreign Key Attribut should be equal to, the value present in primary key Attribute.

→ Referential Integrity Constraint- deales with two relations (R_1 and R_2), if primary key is part of R_1 and foreign key is part of R_2 then. Referential Integrity Constraint is expressed as

$R_2[t[FK]] = R_1[t[PK]]$

Referential Integrity Constraints on Company database is shown in the below figure.

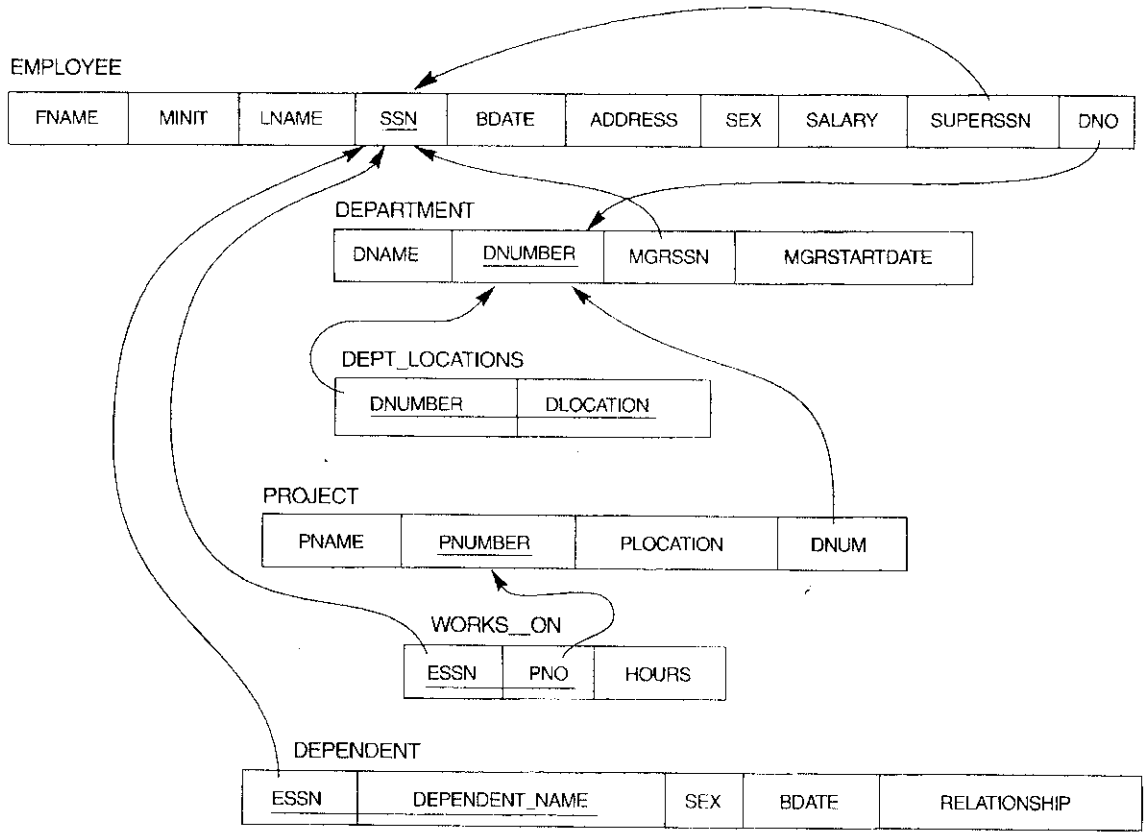


Figure : Referential integrity constraints displayed on the COMPANY relational database schema diagram

Update operations and Dealing with Constraints :

There are 3 basic update operation on relation are

1 Insert : Insert is used to insert a new tuple in a relation.

2 Delete : Delete is used to delete tuples

3 update : Update is used to change values of some attributes in existing tuples.

* Update operations and Dealing with Constraints are explained by Considering below given relations.

Employee

SSN	Name	SAL	DNO
1234	Adarsh	12000	1
1235	Bharath	15000	2

Department —

Dno	Dname
1	CS
2	IS.

Insert operation: Insert operation can violate any constraints (Domain, Key, Entity Integrity and Referential Integrity)

ex:

Insert < KA1321, Arun, 25000, 2 >

violates domain constraint SSN field accepts only integer

Insert < 1238, Rama, 20000, 8 >

violates Referential Integrity constraint-

Insert < NULL, Abhi, 20,000, 1 >

violates Entity Integrity constraint-

Insert < 1234, Adarash, 12000, 1 >

violates Key constraint-

Delete operation:

Delete operation violates "Referential Integrity Constraint" when the tuple being deleted is referenced by the foreign key from other tuple in the data base.

ex 1:

Delete employee with SSN = 1235 is acceptable

ex2: Delete from department where DNO = 1

violates referential Integrity constraint-

Update operation:

Update operation is used to change the values of one or more attributes in a tuple of some Relation R.

ex: 1. Update Salary of employee where SSN - 1231 to 25000 → Acceptable (Not violates any constraint)

2. update DNO employee where SSN = 1234 to 3

violates Referential Integrity constraint-

Ex3: Update DNO of the employee to 'A'

Domain Integrity violates

Ex4: Update SSN to NULL where SSN = 1234

Entity Integrity violates

Relational Algebra:

The basic set of operations for the relational data model is known as the relational Algebra.

→ Relational Algebra is the query language for Relational Data Model.

→ These operations enable the users to specify basic retrieval queries.

→ Relational Algebra operations are broadly divided into two categories.

1. Relational Database operations (select, Project, Join)
2. Set theory operations (Union, Intersection, Set Difference)

Before we move to Relational Algebric operations it is necessary know the following things.

Unary operators are (operations on Single Relation)

1. Select σ
2. project π
3. Rename $\leftarrow$

Binary operators are (operations on two Relations)

1. Cartesian product $\times$
2. Joint $\bowtie$
3. Natural join $*$
4. Set theory operations $\cup, \cap, -$

Relational database and Set theory operations are explained by consider below given Company database.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Select operation :

Select operation is used to select a subset of tuples from a relation that satisfy a selection condition.

→ Select operation is denoted by σ (Sigma Symbol)

→ General format :

$$\sigma_{\text{<Selection Condition>}}(R)$$

R → Relation

→ Selection Condition is a filter keeps only those tuples that satisfy qualifying conditions, others are discarded.

Ex : Retrieve tuples from employee Relation works for deptno = 4.

: $\sigma_{<DNO=4>}(\text{Employee})$

FNAME	MINIT	LNAME	SSN	Bdate	Address	Sex	Salary	Super SSN	DNO
Alicia	J	Zelaya	999887777	1968-01-19	Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	Bellaire, TX	F	43000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	Houston, TX	M	25000	987654321	4

Project operation : This operation selects certain columns from the specified relation and discards other columns π

→ Project operation is denoted by π (pi Symbol)

→ General format : $\pi_{\text{<attribute list>}}(R)$

query 1 : Display Lname, Fname and Salary from Employee

Fname	Lname	Salary
Jhon	Smith	30,000
Franklin	Wong	40,000
Alicia	Zelaya	25,000
Jennifer	Wallace	43,000
Ramesh	Narayan	38,000
Joyce	English	25,000
Ahamad	Jabbar	25,000
James	Borg	55,000

query 2 : Display Sex and Salary From Employee.

Sex	Salary
M	30,000
M	40,000
F	25,000
F	43,000
M	38,000
F	25,000
M	25,000
M	55,000

Sequence of operations :

Relational database operation includes both Select- and project operation.

query : Retrieve names and Salaries of employee who works for dept No 4

$$\pi_{FNAME, MINIT, LNAME, SALARY}(\sigma_{DNO=4}(Employee))$$

FNAME	MINIT	LNAME	SALARY
Alicia	J	Zelaya	25000
Jennifer	S	Wallace	43000
Ahmad	V	Jabbes	25000

query2 : Retrieve Fname and B date of an employee whose Salary is > 25000.

$$\pi_{Fname, Bdate}(\sigma_{Salary > 25000}(Employee))$$

Fname	Bdate
Jhon	1965-01-09
Jennifer	1955-12-08
Ramesh	1962-09-15
James	1935-11-10

Intermediate Relations : Relation which consists of result of intermediate operations.

→ The concept and usage of intermediate relation is illustrated below with an example.

→ query : Retrieve names, salary of all the employee who works for dept 4.

DEP4_EMPS ← Rename $\sigma_{DNo=4}$ (Employee)

↗ Intermediate Relation holds the result of select operation

NAME	MINIT	LNAME	SSN	Bdate	Address	Sex	Salary	Super_SSN	DNO
licia	J	Zelaya	999887777	1968-01-19	Spring, TX	F	25000	987654321	4
ennifer	S	Wallace	987654321	1941-06-20	Bellaire, TX	F	43,000	888665555	4
hmad	V	Jabbar	987987987	1969-03-29	Houston, TX	M	25000	987654321	4

Res ← $\pi_{FNAME, MINIT, LNAME, SALARY}$ (DEP4_EMPS)

FNAME	MINIT	LNAME	SALARY
Alicia	J	Zelaya	25000
Jennifer	S	Wallace	43,000
Ahmad	V	Jabbar	25,000

Set Theory Operations :

Set theory operations on "Relational Algebra" are standard mathematical operations on sets.

→ Various Set Theory operations are listed below.

1. Union : $R \cup S$
2. Intersection $R \cap S$
3. Set Difference $R - S$
4. Cartesian product $R \times S$

The Set theory operations "UNION, INTERSECTION and SET DIFFERENCE" are used to merge the tuples of two relations in various ways.

→ For above mentioned operations "the operand relations $R_1(A_1, A_2 \ldots A_n)$ and $R_2(B_1, B_2, \ldots B_n)$ must have the same number of attributes and the domains of corresponding attributes must be compatible; ie

$dom(A_i) = dom(B_i)$ for $i = 1, 2 \ldots, n$ This Condition is called union compatibility.

UNION operation :

The result of this operation denoted by $R \cup S$, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

→ Example. Set theory operations are explained by using below given relations

Student Relation

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

Instructor Relation

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francies	Johnson
Ramesh	Sash

Example for union operation:

Student ∪ INSTRUCTOR

Fname	Lname
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

Intersection: The result of this operation, denoted by R ∩ S is a relation that includes all tuples that are in both R and S.

Example: Student ∩ INSTRUCTOR

↑ intersection

Fname	Lname
Susan	Yao
Ramesh	Shah

Set Difference:

This operation is also called as MINUS operation. The result of this operation is denoted by R-S or S-R.

→ The operation R-S includes all tuples that are in R but not in S.

→ The operation S-R includes all tuples that are in S but not in R.

Example 1: Student MINUS INSTRUCTOR

Fname	Lname
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

Example 2: INSTRUCTOR – STUDENT

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Note 1: Both UNION AND INTERSECTION are Commutative Operations

$$R \cup S = S \cup R$$

(∪ : UNION on both sides)

Note 2 MINUS operation is not commutative

$$R - S \neq S - R$$

CARTESIAN PRODUCT :

Cartesian product operation is also known as CROSS PRODUCT or ~~Cross~~ Cross Join – which is denoted by ×.

→ Union compatibility of relations is not required for Cross product.

→ This operation is used to combine tuples from two relations in a Combinatorial fashion.

→ The result of $R(A_1, A_2 \ldots A_n)$ × (Cross product) ...→ $S(B_1, B_2, \ldots B_m)$ is a relation Q with degree $n+m$ attributes $Q(A_1, A_2 \ldots A_n, B_1, B_2, \ldots B_m)$ in that order.

→ Hence if R has n_R tuples (denoted as $|R| = n_R$) and S has n_S tuples, then $|R \times S|$ will have $n_R * n_S$ tuples.

Examples for cartesian product is given below:

example1: Consider Employee database

query: Retrieve a list of names of each female employee's dependents.

FEMALE_EMPS ← $\sigma_{Sex = 'F'}$ (EMPLOYEE)

EMPNAMES ← $\pi_{Fname, Lname, Ssn}$ (FEMALE_EMPS)

EMP_DEPENDENTS ← EMPNAME S × DEPENDENTS

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

Join Operation :

The sequence of Cartesian product followed by Select operation or Select condition is used to commonly identify and Select related tuples from two relations.

→ Join operation is denoted by ⋈

General format of Join operation

$$R \bowtie_{\text{<join condition>}} S$$

Note: Every join operation requires a condition

Different join operations are listed below.

1. Equi join
2. Natural join
3. Theta join

Equi join :

The join operation which uses "Comparison operator =". is called Equi join.

→ General Syntax of Equi join is given below :

$$R \bowtie_{\text{attribute of R} = \text{Attribute of S}} S$$

→ Equi join Examples are given below.

Examples : To retrieve the name of the manger of each department-

$$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{MGRSSN} = \text{SSN}} \text{EMPLOYEE}$$

$$\text{RESULT} \leftarrow \pi_{\text{Dname, Lname, Fname}} (\text{DEPT\_MGR})$$

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

Result

Dname	Lname	Fname
Research	Wong	Franklin
Administration	Wallace	Jennifer
Headquarters	James	Borg

Natural join:

Natural join is denoted by *, was created to get rid of the second (superfluous) attribute in an Equi-join condition.

→ Join condition in an Natural join is implicit.

→ Implicit join condition is based on primary key and foreign key.

Note: To apply natural join "primary key attribute name should be same as foreign key attribute name".

→ Natural join avoids superfluous attribute.

→ Example for Natural join is given below.

Consider Department- and Project- department-location from Employee ~~relation~~ database

Proj_DEPT ← Project * DEPT

(a)

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

Natural join operation:

DEPARTMENT_DEPTLOCATIONS ← DEPARTMENT * DEPT_LOCATIONS

(b)

DEPT_LOCS

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Location
Headquarters	1	888665555	1981-06-19	Houston
Administration	4	987654321	1995-01-01	Stafford
Research	5	333445555	1988-05-22	Bellaire
Research	5	333445555	1988-05-22	Sugarland
Research	5	333445555	1988-05-22	Houston

Theta join Operation : Similar to CARTESION PRODUCT followed by SELECT operation with join condition.

Theta join is denoted by

$$R(A_1, A_2 \cdots A_m, B_1, B_2 \cdots B_n) \leftarrow R_1(A_1, A_2 \cdots A_m) \xrightarrow{c}$$

$$\therefore \xrightarrow{c} \bowtie_{cond} R_2(B_1, B_2 \cdots B_n)$$

Where cond = { =, <, >, <=, >=, # }

Note : By default join is theta join

→ Example for Theta join is given below.

General Syntax for Theta join:

$$R \bowtie S$$

attribute value

$=$

$<$

$\geq$

$\therefore$

Ex

DEPT_MGR ← DEPARTMENT ⋈ EMPLOYEE

MGRSSN = SSN

Equi join by default it is Theta join.

Result ← π Dname, Lname, Fname (DEPT-MGR)

DEPT-MGR

Dname	Dnumber	Mgr-SSN	...	Fname	Minit	Lname	SSN	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888 665555	...	James	E	Borg.	888 665555	...

Division Operation :

The Division operation is denoted by ÷ is useful for a special kind of query that sometimes occurs in data-base applications.

→ Division operation is applied to two relations

$$R(y) = R(Z) \div R(X)$$

Where the attributes of R are a subset of the attributes of S, where x, y and Z are attributes.

Relation R(y) consists of all the tuples T(x) in R(z) that appears in the relation R_1 in combination with every tuple from R(x)

Example 1: Find the name of the students who failed in all the subjects. By considering below given relations.

Student -

Name	Sub-failed
A1	DBMS
A1	JAVA
B1	DBMS
C1	JAVA

Subject -

Sub-name
DBMS
JAVA

Res ← Student ÷ Res.

Name
A1

A1 has failed in all the subjects.

Example 2: Retrieve the names of employees who work on all the projects that 'Jhon Smith' works on.

Consider Employee database which is given earlier in the Notes.

" To Solve the above query it is necessary to know project-no where Smith works, as a second Step we need to ~~take~~ Retrieve SSN of all employee works on projects Finally Division operation is applied" this process is given in queries form below.

Q1: SMITH ← σ Fname='John' AND Lname='Smith' (EMPLOYEE)

Fname	MINIT	Lname	SSN	Bdate	Address	Sex	Sal	Super_SSN	DNO
John	B	Smith	123456789	09-JAN-1965	731-Fondren-Houston-TX	M	30000	33344555	5

Q2: SMITH_PNOS ← π_{Pno} (WORKS_ON $\bowtie_{ESSN=SSN}$ SMITH)

SMITH_PNO:

Pno
1
2

Q3: SSN_PNOS ← π_{pno} (WORKS_ON $\bowtie_{ESSN=SSN}$ SMITH)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	?
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

$SSNS(SSN) \leftarrow SSN\_PNOS \div SMITH\_PNOS$

SSNS

Ssn
123456789
453453453

$RESULT \leftarrow \pi_{Fname, Lname}(SSNS * EMPLOYEE)$

Fname	Lname
John	Smith
Joyce	English

Outer join : DBMS Supports another type of join operation called Outerjoin.

- ⟶ The basic operation of outerjoin is Cartesian product
- ⟶ There are 3 types of outer join.
  1. Left-outer join $R ⟕ S$
  2. Right-Outer join $R ⟖ S$
  3. Full Outer join $R ⟗ S$

The outerjoin operations are used to retain either all the tuples of relation R or all tuples of relation S. OR tuples of both the R and S

Left outer join :

This operation keeps every tuple in the left relation. If no match found in S then attributes of relation S is padded with NULL values.

Consider 2 relations T_1 & T_2.

T_1	P	Q	R
	10	a	5
	15	b	8
	25	a	6

T_2	A	B	R
	10	b	6
	25	C	3
	10	b	5

Left outer join :

$T_1 \bowtie T_2$

$T_1.P = T_2.A$

Resultant of left outer join

P	Q	R	A	B	C
10	a	5	10	b	6
10	a	5	10	b	5
15	b	8	NULL	NULL	NULL
25	a	b	25	C	3

Right Outer join :

This operation keeps every tuple of the second relation If match not found in first relation, then first relation attribute values are filled with NULL.

→ Right outer join operation is denoted by

$$R \bowtie_{cond} S$$

Left Relation ↵ cond ↵ Right relation

→ Example :

$$T_1 \bowtie_{T_1.Q = T_2.B} T_2$$

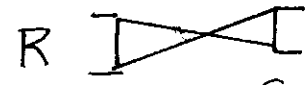
→ Resultant :

P	Q	R	A	B	C
15	b	8	10	b	6
15	b	8	10	b	5
NULL	NULL	NULL	25	C	3

FULL OUTER JOIN :

This operation keeps all the tuples of both the relations. If match not found, attributes will be filled with NULL values.

→ FULL OUTER JOIN operation is denoted by



→ Example:

$T_1.Q = T_2.B$

P	Q	R	A	B	C
10	a	5	NULL	NULL	NULL
15	b	8	10	b	6
15	b	8	10	b	5
25	a	6	NULL	NULL	NULL
NULL	NULL	NULL	25	C	3

Aggregate Functions: In Relational Algebra An Aggregate function is a function where the value of multiple rows are grouped together to form a single value of more significant meaning or measurement.

→ These functions are applied on collection of values from the database.

→ Aggregate Functions are SUM, AVERAGE, MAXIMUM, MINIMUM AND COUNT.

→ Another common type of request involves grouping the tuples in a relation by the value of some of their attributes and then applying an aggregate function independently to each group.

→ The Syntax to use aggregate functions with grouping attribute is given below.

$$\langle \text{grouping Attribute} \rangle \; \mathcal{F} \; \langle \text{function list} \rangle \; (R)$$

<grouping Attribute> → is a list of attributes of the Relation specified in R

<function list> is a list of (<function> <attribute>) pairs.

→ In each such pair, <function> is one of the allowed functions — such as SUM, AVERAGE, MAXIMUM, MINIMUM AND COUNT.

→ <Attribute> is one of the attribute of the relation

Consider the below given Relation:

EMPLOYEE	Eid	Ename	Esal	Dno
	1110	Adarsh	₹5,000	1
	1111	Ankit	50,000	2
	1112	Arun	80,000	1
	1114	Bharath	40,000	2

Ex1: Retrieve the tuples of employee relation group by dno

DNO $\mathcal{F}$ EMPLOYEE

1110	Adarsh	75000	1
1112	Arun	80,000	1
1111	Ankit	50000	2
1114	Bharath	40000	2

Ex2: Count no of Eid group by DNo

DNo $\Im$ COUNT Eid (EMPLOYEE)

DNO	COUNT_EID
1	2
2	2

Ex3: Retrieve the MAx Salary from Employee Relation.

$\Im$ MAX (ESal)

MAX (ESal)
80,000

Ex4: Retrieve the Min Salary from Employee Relation.

$\Im$ MIN (ESal)

MIN (ESal)
40000

Ex5: Retrieve $\overset{no}{\wedge}$ Eid and Total Sal for Employee group by DNo

DNo $\Im$ COUNT Eid, SUM ESal (EMPLOYEE)

DNO	COUNT Eid	SUM_SAL
1	2	1,55,000
2	2	90,000

Ex6 : Retrieve Average Salary from employee.

AVERAGE(ESAL)
61250

$\Im_{AVERAGE\ ESAL}$ (Employee)

ER-to-Relational Mapping Algorithm : ER to Relational Mapping Algorithm consists of 6 steps.

⟶ The steps involved in Mapping are mentioned and explained below.

Step1 : Mapping of Regular Entity Types.

→ For each regular entity type E in the ER schema, create a relation R that includes the simple attributes of E.

→ Choose one of the key attributes of E as the primary key for R.

⟶ If the chosen key of E is composite (candidate), the set of simple attributes that form it will together form the primary key of R.

Example : We create relations EMPLOYEE, DEPARTMENT, ~~in the~~ and PROJECT in the relational schema.

⟶ SSN, DNUMBER and PNUMBER are the Primary Keys for the relation Employee, Department and project respectively.

Step2: Mapping of Weak Entity Types.

- For each weak entity type W in the ER Schema with owner entity type E & include all simple attributes of W as attributes of R.

→ Also include foreign key attributes (of R) for the primary key attributes S that corresponding to the owner entity type(s).

Example: Create the relation DEPENDENT and it is a weak entity type.

→ Include the primary key SSN of the Employee relation as a foreign key attribute of DEPENDENT (renamed to ESSN)

Step3: Mapping of Binary 1:1 Relation Types.

- For each binary 1:1 relationship type R in the ER Schema, identify the relation S and T that corresponding to the entity types participating in R.

1. Foreign Key Approach: Choose one of the relations-S, and include a foreign key in the primary key of T. It is better to choose an entity type with total participation in R in the role of S.

Example: 1:1 relation Manages is mapped by choosing the participating entity type DEPARTMENT to serve in the role of S, because its participation in the MANAGES relationship type is total

2. Merged Relation option: An alternative mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total

3. Cross-reference or relationship relation option: The third alternative is to setup a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.

Step 4: Mapping of Binary 1:N Relationship Types.

For each regular binary 1:N Relationship type R, Identify the relation S that represent the participating entity type at the N-side of the relationship type.

→ Include as foreign key in S the primary key of the relation T that represents the othe entity type participating in R.

Example: 1:N Relationship type WORKS_FOR, CONTROLS and SUPERVISION in the figure.

Step 5: Mapping of Binary M:N Relationship Types.

For each regular binary M:N relationship type R, Create a new relation S to represent R.

→ Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S.

→ Also include any simple attributes of the M:N relationship type as attributes of S.

Example: The M:N relation type WORKS_ON form the ER diagram is mapped by creating a relation WORKS_ON in the relational database schema.

Step6 : Mapping of Multivalued Attributes.

- For each multivalued attribute A, Create a new relation R.

- This relation R will include an attribute Corresponding to A, plus the primary key attribute K as a foreign key in R - of the relation that represents the entity type of relationship type that has A as an attribute.

→ The primary key of R is the Combination of A and K. If the multivalued attribute is Composite we include its Simple components.

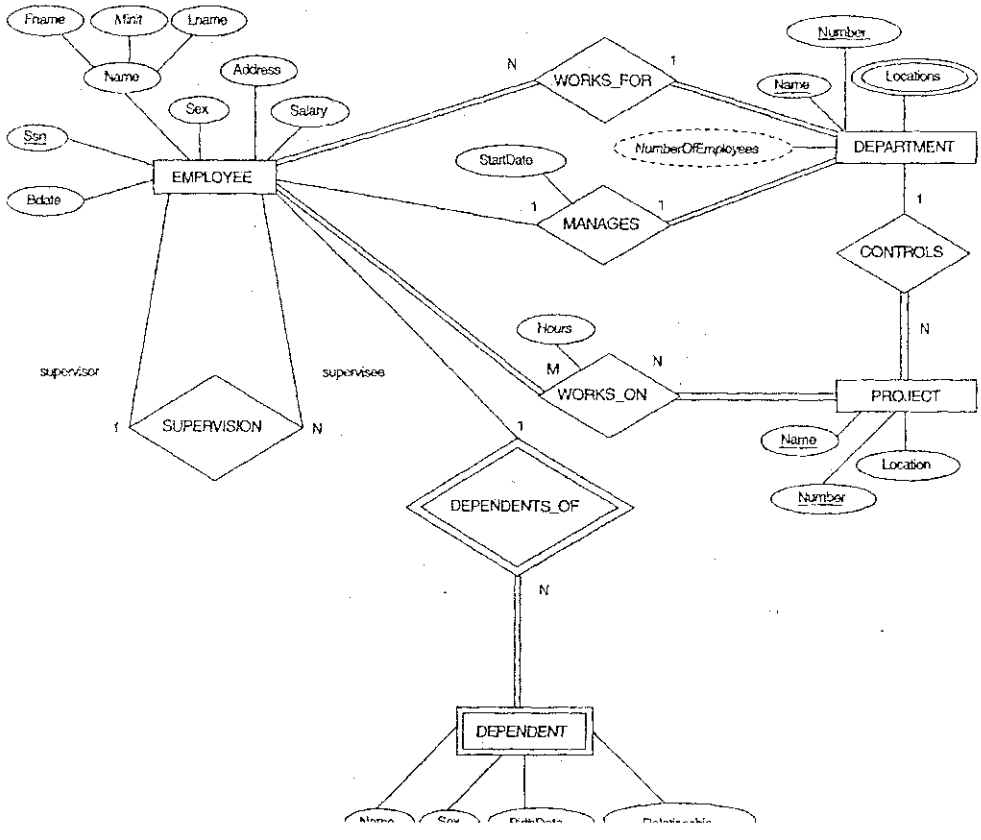
Example : The relation DEPT_LOCATIONS is Created

- The attribute DLocation represents the multivalued attribute LOCATIONS of DEPARTMENT, while DNUMBER as foreign key - represents the primary key of the DEPARTMENT relation.

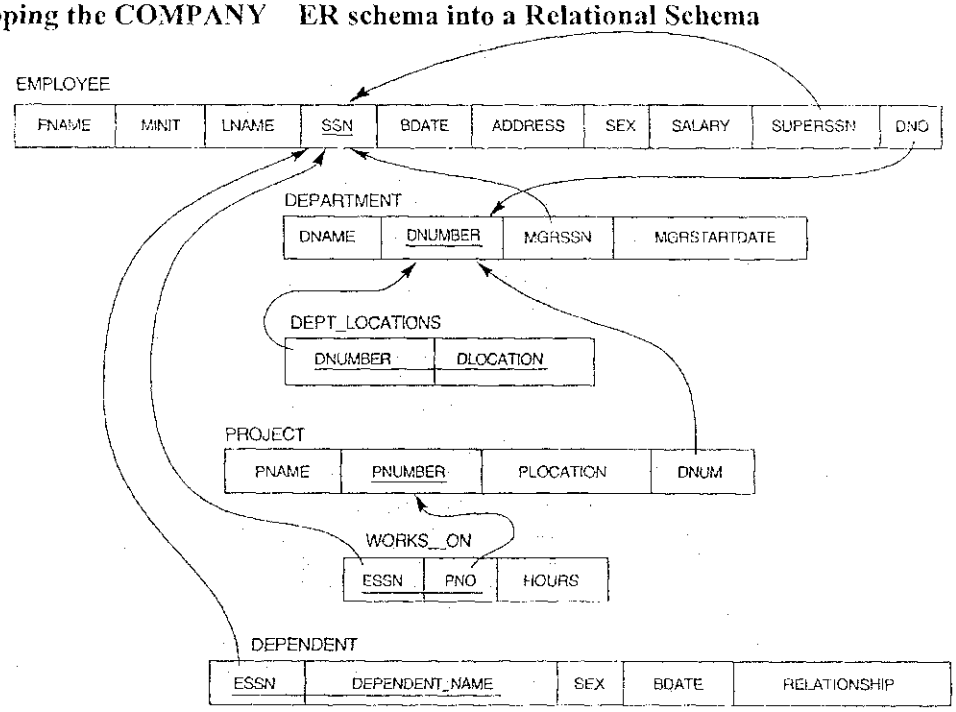
- The primary key of R is the Combination of {DNUMBER, DLOCATION}

Example: ER-to-Relational Mapping

Given: The ER conceptual schema diagram for the COMPANY database



Result of mapping the COMPANY ER schema into a Relational Schema



Solve the below given queries by considering the given schema.

SAILOR	Sid	Sname	Rating	Age

BOATS	Bid	Bname	Color

RESERVES	Sid	Bid	Reserved_date

1. Find the name of the sailor who has reserved boat 1
2. Find the name of the sailors who have reserved green boats.
3. Find the color of the boat reserved by Ramesh.
4. Find the name of the sailors who have reserved red or green boats.
5. Find the Sname of the sailors with age over 20 who have not reserved a red boat

SAILOR	Sid	Sname	Rating	Age
	S123	Krishna	5	22
	S124	Ramesh	7	21

BOATS	Bid	Bname	Color
	1	Titanic	Green
	2	Bismark	Red

RESERVES	Sid	Bid	Date
	S123	1	22/08/2017
	S124	2	25/09/2017

Note: Relation must be constructed based on the data available in the queries.

Q1 : Find the name of the sailor who has reserved boat1.

$$R_1 \leftarrow \pi_{Sid} (\sigma_{bid=1} (Reserves))$$

R1	Sid
	S123

$$Result \leftarrow \pi_{Sname} (Sailor * R1)$$

Result	Sname
	Krishna.

2. Find the name of the sailors who has reserved green boats

$$R_1 \leftarrow \pi_{Bid} (\sigma_{Color = Green} (Boats))$$

R1	Bid
	1

$R_2 \leftarrow \pi_{Sid}\ (R_1 * Reserves)$

R2	Sid
	S123

$Result \leftarrow \pi_{Sname}\ (R_2 * Sailor)$

Result	Sname
	Krishna.

Q3: Find the Color of the boat reserved by Ramesh

$R_1 \leftarrow \pi_{Sid}\ (\sigma_{Sname = Ramesh}\ (Sailors))$

R1	Sid
	S124

$R_2 \leftarrow \pi_{Bid}\ (R_1 * Reserves)$

R2	Bid
	2

$Result \leftarrow \pi_{Color}\ (R_2 * Boats)$

Result	Color
	Red.

Q4 Find the name of the Sailors who have reserved red or green boats.

$$R_1 \leftarrow \pi_{Bid}(\sigma_{Color = red \cup Green}(Boats))$$

R_1	Bid
	1
	2

$$R_2 \leftarrow \pi_{Sid}(R_1 * Reserves)$$

R_2	Sid
	S123
	S124

$$Res \leftarrow \pi_{Sname}(R_2 * Sailor)$$

Result	Sname
	Krishna
	Ramesh

Q5: Find the Sname the Sailors with age over 20 who have not reserved a red boat.

$$R_1 \leftarrow \pi_{Bid}(\sigma_{Color \neq Red}(Boats))$$

R_1	Bid
	1

$R_2 \leftarrow \pi_{sid}(R_1 \ltimes Reserves)$

R_2	sid
	S123

$R_3 \leftarrow R_2 \ltimes Sailor$

R_3	Sid	Sname	Rating	Age
	S123	Krishna	5	22

$Result \leftarrow \pi_{Sname}(\sigma_{Age>20}(R_3))$

Result	Sname
	Krishna.

Solve the below given queries by considering the given Schema.

Student (Student no, Student name)

Work (Student no, Project no)

Project (Project no, Project area.)

Q1: Obtain the Student number and Student-name of all the Students who are working on database projects.

Q2: obtain Student number and Student name of all the Students who are working on both the project having Project no P75 and P81.

3) Obtain Student number and Student name of all the Student who are working on both the project no 68.

(4) Obtain Student number and Student name of all the Students other than the Student with Student no 54. Who work on atleast one project.

~~Consider~~ The below given relations are constructed by considering Partial data present in the queries.

Student

Student_No	Student_Name
54	A
55	B
56	C

Assigned To

Student_No	Project_no
54	P81
55	P75
56	P68

Project:

Project_No	Project_Area
P68	Database
P75	Coding
P81	~~Stati~~ Statistics

1. Obtain the Student number and Student name of all the Students who are working on database projects.

$R_1 \leftarrow \pi_{Project\ no}(\sigma_{project\_area = database}(Project))$

$R_2 \leftarrow \pi_{Stud\_no}(R_1 * Assigned\ To)$

$Res \leftarrow (\pi_{Stud\_no,\ Stud\_name}(R * Student))$

Res	Stud_no	Stud_name
	56	C

2. Obtain Student number and Student name of all the Students who are working on both the project having Project no P75 and P81

$R_1 \leftarrow \pi_{Stud\_no}(\sigma_{Projectno = P75 \cup P81}(Assigned\ To))$

$Res \leftarrow \pi_{Studno,\ Studname}(R_1 * Student)$

Res	Stud-no	Stud-name
	54	A
	55	B

3. Obtain Student number and Student name of all the student who do not work on project no P68

$R_1 \leftarrow \pi_{Stud\_no}(\sigma_{project\_no \neq P68}(Assigned\ To))$

$Res \leftarrow \pi_{Stud\_no,\ Stud\_name}(R_1 * Student)$

$Stud_{no}$	$Stud_{name}$
54	A
55	B

4. Obtain Student numbers and Student name of all Students other than the Student with Student no 54 who work on alleastone project-

$$R_1 \leftarrow \pi_{Stud\_no}(\sigma_{Student\_no \neq 54}\ Assigned\text{-}TO)$$

$$Res \leftarrow \pi_{Student\_no,\ Studentname}(R_1 \bowtie Student)$$

Stude-no	Stud_name
55	B
56	C

Consider the following Schema for a Company database.

EMPLOYEE (Name, SSN, Address, Sex, Salary, DNO (FK), SuperSSN (FK))

DEPARTMENT (Dname, Dnumber, MGRSSN (FK), MGRSTART date)

PROJECT (PName, Pno, plocation, Dno (FK))

WORKS_ON (ESSN (FK), PNO (FK), Hours)

DEPENDENT (ESSN, Dependent_name, Sex, Bdate, Relationship)

I. Retrieve all employees who either work in depart 4 and make over 25000 per year or work in dept 5 & makes 30,000.

$$R_1 \leftarrow \sigma_{DNO=4}(EMPLOYEE)$$

$$R_2 \leftarrow \sigma_{Salary>25000}(R_1)$$

$$R_3 \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$

$$R_4 \leftarrow \sigma_{Salary>30000}(R_3)$$

$$R \leftarrow \pi_{Name, SSN}(R_2 \cup R_4)$$

II. Retrieve the SSN of all employees who either work in dept 5 or directly supervise an employee who works in dept 5.

$$R_2 \leftarrow \pi_{SSN}(\sigma_{DNO=5}(Employee))$$

$$R_3 \leftarrow \pi_{SuperSSN}(\sigma_{DNO=5}(Employee))$$

$$R \leftarrow R_2 \cup R_3$$

iii) Retrieve the name and address of all employee who work for the 'research' department

$$R_1 \leftarrow \pi_{Dnumber}\left(\sigma_{Dname = 'research'}(Department)\right)$$

$$Res \leftarrow \pi_{Name, Address}\left(R_1 \bowtie_{Dnumber = DNO} EMPLOYEE\right)$$

iv) List all the project on which employee Smith working

$$R_1 \leftarrow \sigma_{Name = 'Smith'}(EMPLOYEE)$$

$$R_2 \leftarrow \pi_{SSN}(R_1)$$

$$R_3 \leftarrow R_2 \bowtie_{SSN = ESSN} (WORKS\_ON)$$

$$R_4 \leftarrow \pi_{PNO}(R_3)$$

$$Res \leftarrow \pi_{PName, PNO}(R_4 * PROJECT)$$

** v. Retrieve the names of employees who work on all the projects that 'jhon Smith works on

$$R_1 \leftarrow \pi_{SSN}\left(\sigma_{Name = 'john\ Smith'}(Employee)\right)$$

$$Smith\_PNO \leftarrow \pi_{PNO}\left(R_1 \bowtie_{SSN = ESSN} (WORKS\_ON)\right)$$

$$SSN\_PNO \leftarrow \pi_{ESSN, PNO}(WORKS\_ON)$$

$$SSNS(SSN) \leftarrow SSN\_PNOS \div SMITH\_PNO$$

$$RESULT \leftarrow \pi_{Fname, SSN}(SSNS * EMPLOYEE)$$

Q6. Find the name of the employee who work on all the projects controlled by dept no 5.

$$PRNO \leftarrow \pi_{pno}(\sigma_{DNO=5}(PROJECT))$$

$$SSNS \leftarrow \pi_{ESSN}(PRNO * WORKS\_ON)$$

$$Result \leftarrow \pi_{SSN, name}(SSNS \bowtie_{ESSN=SSN} EMPLOYEE)$$

** Q7 Retrieve the names of employee who have no dependencies

$$TNO\_SSN \leftarrow \pi_{SSN}(EMPLOYEE)$$

$$D\_SSN \leftarrow \pi_{ESSN}(DEPENENT)$$

$$NO\_DEP \leftarrow TNO\_SSN - D\_SSN$$

$$Result \leftarrow \pi_{SSN, Name}(NO\text{-}DEP * EMPLOYEE)$$

SQL: STRUCTURED QUERY LANGUAGE

KARANAM SUNIL KUMAR
Asst. Prof. Dept of CSE
RNSIT
ph: 9972995720
Email id: Sunilkaranam72@gmail.com

SQL:

1. SQL data definition and data types
2. Specifying constrains in SQL
3. Retrieval of Queries in SQl
4. Insert, Delete and Update statements in SQl
5. Additional features of SQL

ADVANCED QUERIES:

1. More complex SQL retrieval queries
2. Specifying constraints as assertions and action triggers
3. Views in SQL
4. Schema change statements in SQl

DATABASE APPLICATION DEVELOPMENT:

1. Accessing database from applications
2. An introduction to JDBC
3. JDBC classes and Interfaces
4. SQLJ stored procedures
5. The Internet Bookshop

INTERNET APPLICATIONS:

1. The three-Tier application Architecture
2. Presentation Layer
3. The Middle Layer

SQL Data definition and data types

SQL is an Abbreviation Structed query language, and Pronounced either See-Kwell or as Separate letters. SQL is a standard query language for requesting information from a database.

→ The original version called SEQUEL was designed by an IBM research Center in 1974 and 1975.

→ Historically, SQL has been the favorite query language for database management systems running on minicomputers and mainframes.

→ Various version of SQL are

— SQL 86 (SQL1) : First

— SQL 92 (SQL2) : major version

— SQL 99 (SQL3) : add triggers, and recursive queries.

→ SQL 2003 : XML, Windows functions.

Data Definition, Constraints and Schema change Statements

Data Definition is SQL takes place through Commands like CREATE, ALTER AND DROP.

→ The main Command for data definition is the CREATE Command, which can be used to Create the following

1. Tables (Relations)
2. Schemas
3. Domains (as well as other constructs such as views, assertions and triggers).

Example for data definition

1. Creation of Schema.

```
CREATE SCHEMA COMPANY AUTHORIZATION 'jSmith'
```

2. Creation of Table

```
CREATE TABLE DEPARTMENT
( DNAME   VARCHAR(15)   NOT NULL
  DNAME   VARCHAR(15)   NOT NULL
  DNO     INTEGER);
```

3. Creation of Views

```
CREATE VIEW DEPT as
Select * from Departmentd;
```

Data Types in SQL :

Various data types of SQL are listed below.

1. Numeric : NUMBER, NUMBER(S,P), INTEGER, INT, FLOAT, DECIMAL
2. Character : CHAR(n), VARCHAR(n), VARCHAR2(n), CHAR VARING(n)
3. Bit String : BLOB, CLOB
4. Boolean : true, false, and null
5. Date and Time : DATE (Date - MON - YEAR) TIME (HH:MM:SS)
   ex: 25 - MAR - 2017
6. User Defined types.

Constraints in SQL :

Constraints are the rules enforced on data columns on tables. These are used to limit the type of data that can go into a table.

→ This ensures the accuracy and reliability of the data in the database.

→ Constraints could be column level or table level.

→ Following are commonly used constraints available in SQL

1. NOT NULL Constraint : Ensures that column cannot have NULL values.

2. DEFAULT Constraint : provides a default value for a column when none is specified.

3. UNIQUE Constraint : Ensures that all values in a column are different.

4. Primary Key Constraint : Uniquly identifies each row in a

5. Foreign Key : Uniquely identifies a row in any another data base

6. CHECK Constraint : The CHECK constraint ensures that all values in a column satisfy certain conditions.

Data Definition : Data definition includes " Create, Alter and drop Commands.

→ In the following section examples given for Create, Alter and drop Command

→ Create Command :

Following section provides Create and Insert Commands for Company database.

Create Command and sample Insert command for company database

Creation of Table Employee:

create table EmpS(Fname varchar(15),Minit char,Lname varchar(15) ,SSN char(9) Primary Key,Bdate Date,Address varchar(30),Sex char,Salary decimal(10,2),SuperSSN char(9),Dno Int);

SQL> desc emps

Name	Null?	Type
FNAME		VARCHAR2(15)
MINIT		CHAR(1)
LNAME		VARCHAR2(15)
SSN	NOT NULL	CHAR(9)
BDATE		DATE
ADDRESS		VARCHAR2(30)
SEX		CHAR(1)
SALARY		NUMBER(10,2)
SUPERSSN		CHAR(9)
DNO		NUMBER(38)

Creation of Table Department:

create table DeptS(Dname varchar(15) Unique,Dnumber Int Primary Key,MgrSSN char(9),Mgr_SDate Date);

SQL> desc depts;

Name	Null?	Type
DNAME		VARCHAR2(15)
DNUMBER	NOT NULL	NUMBER(38)
MGRSSN		CHAR(9)
MGR_SDATE		DATE

Insertion Queries

1)insert into Emps values('&Fname','&Minit','&Lname','&SSN','&Bdate','&Address','&Sex',&Salary,'&SuperSSN',&Dno);

2)insert into DeptS values('&DName',&Dnumber,'&MGRSSN','&MGR_SDATE');

Set SuperSSN as Foreign Key:

alter table EmpS add Foreign Key(SuperSSN) references EmpS(SSN);

Set MgrSSN as Foreign Key to SSN from Employee table:

alter table Depts add Foreign key(MGRSSN) references EmpS(SSN);

Set Dno as foreign key to Dnumber from Department Table:

alter table EmpS add Foreign Key(Dno) references Depts(Dnumber);

Create Table Department Locations:

create table DLocS(Dnumber int,DLocation varchar(15),primary key(Dnumber,Dlocation),foreign key(Dnumber) references Depts(Dnumber));

SQL> desc dlocs

Name	Null?	Type
DNUMBER	NOT NULL	NUMBER(38)
DLOCATION	NOT NULL	VARCHAR2(15)

Create Table Project:

create table Project(Pname varchar(15) Unique,Pnumber int primary key,Plocation varchar(15),DNum int,Foreign key(dnum) references Depts(Dnumber));

SQL> desc project

Name	Null?	Type
PNAME		VARCHAR2(15)
PNUMBER	NOT NULL	NUMBER(38)
PLOCATION		VARCHAR2(15)
DNUM		NUMBER(38)

Create Table Works_On:

create table Works_on(Essn char(9),Pno int,Hours Decimal(3,1),Primary key(Essn,Pno),Foreign key(Essn) references EmpS(SSN),Foreign key(Pno) references Project(Pnumber));

SQL> desc works_on

Name	Null?	Type
ESSN	NOT NULL	CHAR(9)
PNO	NOT NULL	NUMBER(38)
HOURS		NUMBER(3,1)

Create Table Dependant:

create table DepanS(Essn char(9),Dependant_Name varchar(15),Sex char,Bdate Date,Relationship varchar(8),primary key(Essn,Dependant_Name),Foreign key(Essn) references EmpS(Ssn));

SQL> desc depans

Name	Null?	Type
ESSN	NOT NULL	CHAR(9)
DEPENDANT_NAME	NOT NULL	VARCHAR2(15)
SEX		CHAR(1)
BDATE		DATE
RELATIONSHIP		VARCHAR2(8)

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
	1	Houston
	4	Stafford
	5	Bellaire
	5	Sugarland
	5	Houston

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

SQL queries :

Query1 : Retrieve the birth date and address of the employee who name is 'john B Smith".

```
SQL> SELECT BDATE,ADDRESS FROM EMPS WHERE FNAME='john' AND MINIT='B' AND
LNAME='smith';

BDATE     ADDRESS
--------- ------------------------------
09-JAN-65 731 Fondren, Houston, TX
```

query2: Retrieve the name and address of all employees who work for the research department.

```
SQL> SELECT FNAME,LNAME,ADDRESS
 2  FROM EMPS,DEPTS
 3  WHERE DNAME='Research' AND DNUMBER=DNO;

FNAME          LNAME          ADDRESS
-------------- -------------- ----------------------------
john           smith          731 Fondren, Houston, TX
Franklin       wong           638 voss,houston,TX
Ramesh         Narayan        975 Fire Oak,Humble, TX
joyce          English        5631 Rice,Houston, TX
```

query3 : For every project located in Stafford list the project number the controlling department manager's last name address and birth data.

```
SQL> SELECT PNUMBER,DNUM,LNAME,ADDRESS,BDATE
 2  FROM PROJECT,DEPTS,EMPS
 3  WHERE DNO=DNUMBER AND MGRSSN=SSN AND Plocation='Stafford';

 PNUMBER     DNUM LNAME        ADDRESS                      BDATE
---------- ---------- -------------- ---------------------------- ---------
        30          4 wong           638 voss,houston,TX        08-DEC-55
        30          4 wallace        291 Berry,Bellaire,TX      20-JUN-41
        30          4 Borg           450 Stone,Houston, Tx      10-NOV-37
```

Ambiguous Attribute Names, Aliasing, Renaming and Tuple Variables

In SQL, the same name can be used for two (or more) attributes as long as the attributes are in different relations.

→ these attribute Names leads to Ambiguity.

→ In this case, a multiple query refers to two or more attributes with the same name, we must qualify the attribute name with the relation name to prevent Ambiguity.

→ Assume Dno and Lname attributes of the Employee as Dnumber and Name, In the Relation Department there are attributes Named as Dnumber, Name.

→ To prevent the ambiguity we need to prefix relation name with . (dot) before the attribute name.

→ Example.

```
SQL> SELECT FNAME,LNAME,ADDRESS
 2 FROM EMPS,DEPTS
 3 WHERE DEPTS.DNAME='Research' AND DEPTS.DNUMBER=EMPS.DN

FNAME        LNAME        ADDRESS
-------------- -------------- ------------------------
john         smith        731 Fondren, Houston, TX
Franklin     wong         638 voss,houston,TX
Ramesh       Narayan      975 Fire Oak,Humble, TX
joyce        English      5631 Rice,Houston, TX
```

DBMS also allows the Creation of Alias for a relation to avoid Ambiguity.

Example.

SQL> Select FNAME, LNAME, DNO, DNAME From

Employee E, Department D;

↓ Alias ↓ Alias

The ambiguity of attribute name also arises in the case of queries that refers to the same relation twice as shown in the below fig

```
SQL> SELECT E.FNAME,E.LNAME,S.FNAME,S.LNAME
  2  FROM EMPS E,EMPS S
  3  WHERE E.SUPERSSN=S.SSN;

FNAME           LNAME           FNAME           LNAME
--------------- --------------- --------------- -------
joyce           English         Franklin        wong
Ramesh          Narayan         Franklin        wong
john            smith           Franklin        wong
alicia          Zelaya          jennifer        wallace
jennifer        wallace         james           Borg
Franklin        wong            james           Borg
```

Unspecified WHERE clause and use of the Asterisk :

Unspecified where clause indicates no condition on tuple selection, hence all tuples of the relation specified in the from clause qualify and are selected for the query result.

Ex :
query : Select all Employee SSns.

```
SQL> SELECT SSN
  2  FROM EMPS;

SSN
---------
123456789
333445555
453453453
666884444
888665555
987654321
987987987
999887777
```

If more than one relation is specified in the FROM clause and there is no where clause, then the cross product all possible tuple combinations of these relations is selected.

ex: query : Select Sname and dname attributes from employee and department.

```
SQL> SELECT SSN,DNAME
  2  FROM EMPS,DEPTS;

SSN       DNAME
--------- --------------

123456789 Research
333445555 Research
453453453 Research
666884444 Research
888665555 Research
987654321 Research
987987987 Research
999887777 Research
123456789 Administration
333445555 Administration
453453453 Administration

SSN       DNAME
--------- --------------
666884444 Administration
888665555 Administration
987654321 Administration
987987987 Administration
999887777 Administration
123456789 Headquarters
333445555 Headquarters
453453453 Headquarters
666884444 Headquarters
888665555 Headquarters
987654321 Headquarters

SSN       DNAME
--------- --------------
987987987 Headquarters
999887777 Headquarters
```

To retrieve all the attribute values of the selected tuples, we do not have to list the attribute names explicitly in SQL; we just specify an asterisk (*) which stands for all the attribute.

ex query: Select * from employee Where Dno = 5.

```
SQL> SELECT *
  2  FROM EMPS
  3  WHERE DNO=5;

FNAME      M LNAME      SSN       BDATE     ADDRESS                        S  SALARY SUPERSSN
DNO
---------- - ---------- --------- --------- ------------------------------ - ---------- --

john       B smith      123456789 09-JAN-65 731 Fondren, Houston, TX       M      30000 333445555
5
Franklin   T wong       333445555 08-DEC-55 638 voss,houston,TX            M      40000 888665555
5
  mesh     K Narayan    666884444 15-SEP-62 975 Fire Oak,Humble, TX        M      38000 333445555

           A English    453453453 31-JUL-72 5631 Rice,Houston, TX          F      25000 333445555       5
```

query: Select * from employee Where
DNo = 5

```
SQL> SELECT *
  2  FROM EMPS
  3  WHERE DNO=5;

FNAME      M LNAME      SSN       BDATE     ADDRESS                        S  SALARY SUPERSSN
DNO
---------- - ---------- --------- --------- ------------------------------ - ---------- --
john       B smith      123456789 09-JAN-65 731 Fondren, Houston, TX       M      30000 333445555
5
Franklin   T wong       333445555 08-DEC-55 638 voss,houston,TX            M      40000 888665555
5
Ramesh     K Narayan    666884444 15-SEP-62 975 Fire Oak,Humble, TX        M      38000 333445555
5
joyce      A English    453453453 31-JUL-72 5631 Rice,Houston, TX          F      25000 333445555       5

S
```

query: Select * from emps, depts
Where dname = 'Research' AND DNo = DNUMBER

```
SQL> SELECT *
  2  FROM EMPS.DEPTS
  3  WHERE DNAME='Research' AND DNO=DNUMBER;

FNAME      M LNAME      SSN       BDATE     ADDRESS                    S  SALARY SUPERSSN      DNO DNAME
DNUMBER
---------- - ---------- --------- --------- -------------------------- - ---------- --
MGRSSN    MGR_SDATE
--------- ---------
john       B smith      123456789 09-JAN-65 731 Fondren, Houston, TX   M  30000 333445555      5 Res
333445555 22-MAY-88

Franklin   T wong       333445555 08-DEC-55 638 voss,houston,TX        M  40000 888665555      5 Researc
333445555 22-MAY-88

Ramesh     K Narayan    666884444 15-SEP-62 975 Fire Oak,Humble, TX    M  38000 333445555      5 Re
333445555 22-MAY-88


FNAME      M LNAME      SSN       BDATE     ADDRESS                    S  SALARY SUPERSSN      DNO DNAME
DNUMBER
---------- - ---------- --------- --------- -------------------------- - ---------- --
MGRSSN    MGR_SDATE
--------- ---------
joyce      A English    453453453 31-JUL-72 5631 Rice,Houston, TX      F  25000 333445555      5 Resea
333445555 22-MAY-88
```

Query: Retrieve the salary of every employee and all distinct salary values.

Usage of ALL

Query:1

```
SQL> SELECT ALL SALARY
  FROM EMPS;

    SALARY
----------
     30000
     40000
     25000
     43000
     38000
     25000
     25000
     55000
```

query2: Usage of Distinct

```
SQL> SELECT DISTINCT SALA
  2  FROM EMPS;

    SALARY
----------
     38000
     43000
     55000
     30000
     40000
```

Union ; INTERSECTION AND MINUS :

SQL has directly incorporated some of the set theory operations like Union, INTERSECT AND MINUS

① UNION :

ex: Make a list of all project numbers for projects that involve an employee who last name is 'Smith' either as a worker or as a manager of the department that controls the project.

```
SQL> (SELECT DISTINCT PNUMBER
  2  FROM PROJECT,DEPTS,EMPS
  3  WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME='smith')
  4  UNION
  5  (SELECT DISTINCT PNUMBER
  6  FROM PROJECT,WORKS_ON,EMPS
  7  WHERE PNUMBER=PNO AND ESSN=SSN AND LNAME='smith');

 PNUMBER
---------
       1
       2
```

ex 2:

Consider Student and Instructor relation

Student

Fname	Lname
Susan	Yao
Ramesh	Sash
Johnney	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wong
Ernest	Gilbert

Instructor

Fname	Lname
John	Smith
Richardo	Browne
Susan	Yao
Francies	Johnson
Ramesh	Sash

```
SQL> select * from student1 UNION select * from instruct;

FNAME           LNAME
--------------- ---------------
Barbara         jones
Johnney         kohler
Ramesh          sash
amy ford        ford
ernest          gilbert
francis         jphnson
jimmy           wang
john            smith
ramesh          sash
ricardo         browne
susan           yao
```

INTERSECTION

```
SQL> select * from student1 INTERSECT select * from instruct;

FNAME           LNAME
--------------- ---------------
susan           yao
```

MINUS

```
SQL> select * from student1 MINUS select * from instruct;

FNAME           LNAME
--------------- ---------------
Barbara         jones
Johnney         kohler
Ramesh          sash
amy ford        ford
ernest          gilbert
jimmy           wang
SQL> select * from instruct MINUS select * from student1;

FNAME           LNAME
--------------- ---------------
francis         jphnson
john            smith
ramesh          sash
ricardo         browne
```

Substring Pattern Matching and Arithmetic Operators

SQL have some special features to deal with pattern Matching.

→ The first feature allows comparison conditions on only parts of the character string

— LIKE comparison operator is used for string pattern matching

— Exple for LIKE Comparison operator is given below

```
SQL> SELECT FNAME,LNAME
  2  FROM EMPS
  3  WHERE ADDRESS LIKE '%Houston, TX%';

FNAME        LNAME
------------ ------------
john         smith
joyce        English
```

Partial strings are specified using two reserved characters %. replaces an arbitrary number of zero or more characters and the underscore (_) replaces a single character. example is given in the below section.

```
SQL> SELECT FNAME,LNAME
  2  FROM EMPS
  3  WHERE BDATE LIKE '_______5';

FNAME        LNAME
------------ ------------
john         smith
Franklin     wong
```

Another feature allows the use of arithmetic in queries. The standard arithmetic operators for addition(+), subtraction(-), multiplication(*) and division(/) can be applied to numeric values or attributes with numeric domains.

query: Show the resulting salaries if every employee working on the product x project is given a 10% raise.

```
SQL> SELECT FNAME,LNAME,1.1*SALARY AS INCREASED_SAL
  2  FROM EMPS,WORKS_ON,PROJECT
  3  WHERE SSN=ESSN AND PNO=PNUMBER AND PNAME='ProductX';

FNAME        LNAME        INCREASED_SAL
------------ ------------ -------------
john         smith        33000
joyce        English      27500
```

query: Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

```
SQL> SELECT *
  2 FROM EMPS
  3 WHERE (SALARY BETWEEN 30000 AND 40000)AND DNO=5;

FNAME      M LNAME      SSN       BDATE     ADDRESS                  S  SALARY SUPERSSN    DNO
---------- - ---------- --------- --------- ------------------------ - ------ ---------- ---
john       B smith      123456789 09-JAN-65 731 Fondren. Houston, TX M  30000 333445555    5
Franklin   T wong       333445555 08-DEC-55 638 voss.houston,TX      M  40000 888665555    5
Ramesh     K Narayan    666884444 15-SEP-62 975 Fire Oak.Humble. TX  M  38000 333445555    5
```

Ordering of query Results :

SQL allows the user to order the tuples in the result of a query by the values of one or more attributes that appear in the query result by using the ORDER BY Clause.

query : Retrieve a list of employees and the project they are working on ordered by department and within each department, ordered alphabetically by lastname, then firstname.

```
SQL> select D.Dname, E.Lname,E.Fname,P.pname from depts D,Emps E,WORKS_ON W,PROJECT P
  where D.Dnumber=E.Dno AND E.SSN=W.ESSN AND W.PNo=P.Pnumber ORDER BY
D.DNAME,E.LNAME,E.FNAME;

DNAME          LNAME      FNAME      PNAME
-------------- ---------- ---------- --------------
Administration Zelaya     alicia     Computerization
Administration Zelaya     alicia     Newbenefits
Administration jabbar     Ahamad     Computerization
Administration jabbar     Ahamad     Newbenefits
Administration wallace    jennifer   Reorganization
Administration wallace    jennifer   Newbenefits
Headquarters   Borg       james      Reorganization
Research       English    joyce      ProductX
Research       English    joyce      ProductY
Research       Narayan    Ramesh     ProductZ
Research       smith      john       ProductY

DNAME          LNAME      FNAME      PNAME
-------------- ---------- ---------- --------------
Research       smith      john       ProductX
Research       wong       Franklin   ProductZ
Research       wong       Franklin   Computerization
Research       wong       Franklin   ProductY
Research       wong       Franklin   Reorganization
```

INSERT, DELETE and UPDATE Statements in SQL :

In SQL 3 Commands can be used to modify the database INSERT, DELETE and UPDATE

INSERT Command : INSERT command is used to add a single tuple to a relation. We must specify the relation name and a list of values for tuple.

⟶ The values should be listed in the same order in which corresponding attributes were specified in the CREATE Command.

⟶ Examples for insert Command is shown below.

Contents of Employee relation **(emps is name given employee relation)**
SQL> select * from emps;

FNAME	M	LNAME	SSN	BDATE	ADDRESS	S	SALARY	SUPERSSN	DNO
john	B	smith	123456789	09-JAN-65	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	wong	333445555	08-DEC-55	638 voss,houston,TX	M	40000	888665555	5
alicia	J	Zelaya	999887777	19-JAN-68	3321 castle,Spring, TX	F	25000	987654321	4
jennifer	S	wallace	987654321	20-JUN-41	291 Berry,Bellaire,TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	15-SEP-62	975 Fire Oak,Humble, TX	M	38000	333445555	5
joyce	A	English	453453453	31-JUL-72	5631 Rice,Houston, TX	F	25000	333445555	5
Ahamad	V	jabbar	987987987	29-MAR-69	980 Dallas,Houston, Tx	M	25000		4
james	E	Borg	888665555	10-NOV-37	450 Stone,Houston, Tx	M	55000	NULL	1

SQL> insert into Emps values('RICHARD','K','Marini','653298653','30-dec-1962','98 Oak Forest, Katy,
TX','M',37000,'653298653',4);

1 row created.
SQL> select * from emps;

Contents of Employee Relation After Insertion

FNAME	M	LNAME	SSN	BDATE	ADDRESS	S	SALARY	SUPERSSN	DNO
RICHARD	**K**	**Marini**	**653298653**	**30-DEC-62**	**98 Oak Forest, Katy, TX**	**M**	**37000**	**653298653**	**4**
john	B	smith	123456789	09-JAN-65	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	wong	333445555	08-DEC-55	638 voss,houston,TX	M	40000	888665555	5
alicia	J	Zelaya	999887777	19-JAN-68	3321 castle,Spring, TX	F	25000	987654321	4
jennifer	S	wallace	987654321	20-JUN-41	291 Berry,Bellaire,TX	f	43000	888665555	4
Ramesh	K	Narayan	666884444	15-SEP-62	975 Fire Oak,Humble, TX	M	38000	333445555	5
joyce	A	English	453453453	31-JUL-72	5631 Rice,Houston, TX	F	25000	333445555	5
Ahamad	V	jabbar	987987987	29-MAR-69	980 Dallas,Houston, Tx	M	25000		4
james	E	Borg	888665555	10-NOV-37	450 Stone,Houston, Tx	M	55000	NULL	1

DELETE Command :

The DELETE Command removes tuples from a relation. It includes a WHERE clause similar to that used in SQL query; to select the tuples to be selected. Tuples are explicitly deleted from only one table at a time.

→ However the deletion may propagate to tuples in other relations if referential triggered actions are specified in the referential Integrity Constraints of the DDL.

→ Examples. : Employee relation before deletion.

FNAME	M	LNAME	SSN	BDATE	ADDRESS	S	SALARY	SUPERSSN	DNO
RICHARD	K	Marini	653298653	30-DEC-62	98 Oak Forest, Katy, TX	M	37000	653298653	4
Sunil		karanam	999888777						4
john	B	smith	123456789	09-JAN-65	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	wong	333445555	08-DEC-55	638 voss,houston,TX	M	40000	888665555	5
alicia	J	Zelaya	999887777	19-JAN-68	3321 castle,Spring, TX	F	25000	987654321	4
jennifer	S	wallace	987654321	20-JUN-41	291 Berry,Bellaire,TX	f	43000	888665555	4
Ramesh	K	Narayan	666884444	15-SEP-62	975 Fire Oak,Humble, TX	M	38000	333445555	5
joyce	A	English	453453453	31-JUL-72	5631 Rice,Houston, TX	F	25000	333445555	5
Ahamad	V	jabbar	987987987	29-MAR-69	980 Dallas,Houston, Tx	M	25000		4
james	E	Borg	888665555	10-NOV-37	450 Stone,Houston, Tx	M	55000	NULL	1

After delete command

SQL> delete from emps where Lname='Marini';

1 row deleted.

SQL> select * from emps;

FNAME	M	LNAME	SSN	BDATE	ADDRESS	S	SALARY	SUPERSSN	DNO
Sunil		karanam	999888777						4
john	B	smith	123456789	09-JAN-65	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	wong	333445555	08-DEC-55	638 voss,houston,TX	M	40000	888665555	5
alicia	J	Zelaya	999887777	19-JAN-68	3321 castle,Spring, TX	F	25000	987654321	4
jennifer	S	wallace	987654321	20-JUN-41	291 Berry,Bellaire,TX	f	43000	888665555	4
Ramesh	K	Narayan	666884444	15-SEP-62	975 Fire Oak,Humble, TX	M	38000	333445555	5
joyce	A	English	453453453	31-JUL-72	5631 Rice,Houston, TX	F	25000	333445555	5
Ahamad	V	jabbar	987987987	29-MAR-69	980 Dallas,Houston, Tx	M	25000		4
james	E	Borg	888665555	10-NOV-37	450 Stone,Houston, Tx	M	55000	NULL	1

Example 2

After execution of delete command

SQL> delete from emps where ssn='999888777';

1 row deleted.

SQL> select * from emps;

FNAME	M	LNAME	SSN	BDATE	ADDRESS	S	SALARY	SUPERSSN	DNO
john	B	smith	123456789	09-JAN-65	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	wong	333445555	08-DEC-55	638 voss,houston,TX	M	40000	888665555	5
alicia	J	Zelaya	999887777	19-JAN-68	3321 castle,Spring, TX	F	25000	987654321	4
jennifer	S	wallace	987654321	20-JUN-41	291 Berry,Bellaire,TX	f	43000	888665555	4
Ramesh	K	Narayan	666884444	15-SEP-62	975 Fire Oak,Humble, TX	M	38000	333445555	5
joyce	A	English	453453453	31-JUL-72	5631 Rice,Houston, TX	F	25000	333445555	5
Ahamad	V	jabbar	987987987	29-MAR-69	980 Dallas,Houston, Tx	M	25000		4
james	E	Borg	888665555	10-NOV-37	450 Stone,Houston, Tx	M	55000	NULL	1

SQL>delete from emps;

9 rows deleted.

Note: Be careful while executing above query because it deletes all the tuples of the relation.

The UPDATE Command :

The UPDATE Command is used to modify attribute values of one or more selected tuples.

→ Updating a primary key value may propagate to the foreign key values of tuples in other relations if such a referential triggered action is specified in the referential integrity constraints of the DDL.

→ SET Clause in the referential integrity constraints of the DDL

Examples : Contents of project relation before execution

SQL> select * from project;

PNAME	PNUMBER	PLOCATION	DNUM
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

6 rows selected.

SQL> update project set Plocation='Bellaire', Dnum=5 where Pnumber=10;

1 row updated.

SQL> select * from project;

PNAME	PNUMBER	PLOCATION	DNUM
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Bellaire	5
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

SQL> select * from emps;

FNAME	M	LNAME	SSN	BDATE	ADDRESS	S	SALARY	SUPERSSN	DNO
john	B	smith	123456789	09-JAN-65	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	wong	333445555	08-DEC-55	638 voss,houston,TX	M	40000	888665555	5
alicia	J	Zelaya	999887777	19-JAN-68	3321 castle,Spring, TX	F	25000	987654321	4
jennifer	S	wallace	987654321	20-JUN-41	291 Berry,Bellaire,TX	f	43000	888665555	4
Ramesh	K	Narayan	666884444	15-SEP-62	975 Fire Oak,Humble, TX	M	38000	333445555	5
joyce	A	English	453453453	31-JUL-72	5631 Rice,Houston, TX	F	25000	333445555	5
Ahamad	V	jabbar	987987987	29-MAR-69	980 Dallas,Houston, Tx	M	25000		4
james	E	Borg	888665555	10-NOV-37	450 Stone,Houston, Tx	M	55000	NULL	1

SQL> update emps set salary=salary *1.1 where Dno=5;

4 rows updated.

SQL> select * from emps;

FNAME	M	LNAME	SSN	BDATE	ADDRESS	S	SALARY	SUPERSSN	DNO
john	B	smith	123456789	09-JAN-65	731 Fondren, Houston, TX	M	**33000**	333445555	5
Franklin	T	wong	333445555	08-DEC-55	638 voss,houston,TX	M	**44000**	888665555	5
alicia	J	Zelaya	999887777	19-JAN-68	3321 castle,Spring, TX	F	25000	987654321	4
jennifer	S	wallace	987654321	20-JUN-41	291 Berry,Bellaire,TX	f	43000	888665555	4
Ramesh	K	Narayan	666884444	15-SEP-62	975 Fire Oak,Humble, TX	M	**41800**	333445555	5
joyce	A	English	453453453	31-JUL-72	5631 Rice,Houston, TX	F	**27500**	333445555	5
Ahamad	V	jabbar	987987987	29-MAR-69	980 Dallas,Houston, Tx	M	25000		4
james	E	Borg	888665555	10-NOV-37	450 Stone,Houston, Tx	M	55000	NULL	1

Additional Features of SQL :

SQL has a number of features, those are listed below.

1. SQL provides various techniques for specifying complex retrieval queries, including nested queries, aggregate function, grouping ,joined tables and recursive queries.

2. SQL has various techniques for writing programs in various programming languages that include SQL statements to access one or more database.

3. Each commercial RDBMS will have, in addition to the SQL commands, a set of commands for specifying physical database design parameters, file structures for relations and access paths such as indexes

4. SQL has transaction control commands. These are used to specify units of database processing for concurrency control and recovery purposes.

5. SQ lhs language constructs for specifying the granting and revoking of privileges to users. Privileges typically correspond to the right use certain SQL commands to access certain relations. The commands called GRANT and REVOKE is used for this purpose.

6. SQL has language constructs for creating triggers. These are generally referred to as active database techniques.

7. SQL has incorporated many features from object oriented model to have more powerful capabilities.

8. SQL and relational databases can interact with new technologies such as XML.

SQL.

KARANAM SUNIL KUMAR
Dept of CSE
RNSIT
ph: 9972995720

SQL is a standard language for storing, manipulating and retrieving data in databases.

Our SQL tutorial will teach you how to use SQL in: MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems.

```
SELECT * FROM Customers;
```

What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

Different SQL Versions:

Version	Release	Year
13.0	SQL Server 2016	2016

12.0	SQL Server 2014	2014
11.0	SQL Server 2012	2012
10.50	SQL Server 2008 R2	2010

Alter table command:

1. ALTER TABLE *table_name*
   ADD *column_name datatype*;

2. ALTER TABLE Customers
   ADD Email varchar(255);

3. **ALTER TABLE table_name DROP COLUMN column_name;**

To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following SQL:
MySQL / SQL Server / Oracle / MS Access:

4. ALTER TABLE Persons
   ADD PRIMARY KEY (ID);

5. ALTER TABLE Persons
   DROP PRIMARY KEY;

6. ALTER TABLE Customer RENAME COLUMN Address TO Addr;

1. ALTER TABLE Customer CHANGE Address Addr char(50);
2. **ALTER TABLE table_name**
3. **RENAME TO new_table_name;**

7. ALTER TABLE Orders
   ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
8. SELECT CustomerName, City FROM Customers;

SELECT DISTINCT Examples

The following SQL statement selects only the DISTINCT values from the "Country" column in the "Customers" table:

Example

```
SELECT DISTINCT Country FROM Customers;
```

9. SELECT * FROM Customers
   WHERE Country='Mexico';

10. SELECT * FROM Customers
    WHERE CustomerID=1;

Operator	Description
=	Equal
>	Greater than

<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be wr !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a

The SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Example:

```
SELECT * FROM Customers
WHERE Country='Germany' AND City='Berlin';
```

OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Example:

```
SELECT * FROM Customers;
WHERE City='Berlin' OR City='München';
```

Example2:

```
SELECT * FROM Customers
WHERE Country='Germany' OR Country='Spain';
```

NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

```
SELECT * FROM Customers
WHERE NOT Country='Germany';
```

The SQL ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

ORDER BY Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

Example1:

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

Example1:

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

The SQL INSERT INTO Statement

The `INSERT INTO` statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the `INSERT INTO` statement in two ways:

1. Specify both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

SQL NULL values

The SQL **NULL** is the term used to represent a missing value. A NULL value in a table is a value in a field that appears to be blank.

A field with a NULL value is a field with no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

```
ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |          |
|  7 | Muffy    |  24 | Indore    |          |
+----+----------+-----+-----------+----------+
```

```
SELECT  ID, NAME, AGE, ADDRESS, SALARY
   FROM CUSTOMERS
   WHERE SALARY IS NOT NULL;
```

```
SELECT  ID, NAME, AGE, ADDRESS, SALARY
   FROM CUSTOMERS
   WHERE SALARY IS NULL;
```

The SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table

UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1
```

The SQL DELETE Statement

The DELETE statement is used to delete existing records in a table.

DELETE Syntax

```
DELETE FROM table_name WHERE condition;
```

SQL TOP, LIMIT and FETCH FIRST Examples

The following SQL statement selects the first three records from the "Customers" table (for SQL Server/MS Access):

Example

```
SELECT TOP 3 * FROM Customers; //works for access
```

```
SELECT * FROM Customers //works for sql
LIMIT 3;
```

```
SELECT * FROM Customers
FETCH FIRST 3 ROWS ONLY;
```

```
SELECT * FROM Customers
FETCH FIRST 50 PERCENT ROWS ONLY;
```

```
SELECT * FROM Customers
WHERE Country='Germany'
LIMIT 3;
```

The SQL MIN() and MAX() Functions

The `MIN()` function returns the smallest value of the selected column.

The `MAX()` function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

Example

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

The SQL COUNT(), AVG() and SUM() Functions

The `COUNT()` function returns the number of rows that matches a specified criterion.

COUNT() Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

The `AVG()` function returns the average value of a numeric column.

AVG() Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

The `SUM()` function returns the total sum of a numeric column.

SUM() Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

COUNT() Example

The following SQL statement finds the number of products:

Example

```
SELECT COUNT(ProductID)
FROM Products;
```

Try it Yourself »

Note: NULL values are not counted.

AVG() Example

The following SQL statement finds the average price of all products:

Example

```
SELECT AVG(Price)
FROM Products;
```

The SQL LIKE Operator

The `LIKE` operator is used in a `WHERE` clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the `LIKE` operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (_) represents one, single character

Note: MS Access uses an asterisk (*) instead of the percent sign (%

LIKE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

SQL LIKE Examples

The following SQL statement selects all customers with a CustomerName starting with "a":

Example

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

Try it Yourself »

The following SQL statement selects all customers with a CustomerName ending with "a":

Example

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%a';
```

Try it Yourself »

The following SQL statement selects all customers with a CustomerName that have "or" in any position:

Example

```
SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

Try it Yourself »

The following SQL statement selects all customers with a CustomerName that have "r" in the second position:

Example

```
SELECT * FROM Customers
WHERE CustomerName LIKE '_r%';
```

Try it Yourself »

The following SQL statement selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

Example

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a__%';
```

Try it Yourself »

The following SQL statement selects all customers with a ContactName that starts with "a" and ends with "o":

Example

```
SELECT * FROM Customers
WHERE ContactName LIKE 'a%o';
```

SQL Wildcard Characters

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Using the % Wildcard

The following SQL statement selects all customers with a City starting with "ber":

Example

```
SELECT * FROM Customers
WHERE City LIKE 'ber%';
```

Try it Yourself »

The following SQL statement selects all customers with a City containing the pattern "es":

Example

```
SELECT * FROM Customers
WHERE City LIKE '%es%';
```

Try it Yourself »

Using the _ Wildcard

The following SQL statement selects all customers with a City starting with any character, followed by "ondon":

Example

```
SELECT * FROM Customers
WHERE City LIKE '_ondon';
```

Try it Yourself »

The following SQL statement selects all customers with a City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

Example

```
SELECT * FROM Customers
WHERE City LIKE 'L_n_on';
```

Try it Yourself »

The SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

IN Operator Examples

The following SQL statement selects all customers that are located in "Germany", "France" or "UK":

Example

```
SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');
```

Try it Yourself »

The following SQL statement selects all customers that are NOT located in "Germany", "France" or "UK":

Example

```
SELECT * FROM Customers
WHERE Country NOT IN ('Germany', 'France', 'UK');
```

Try it Yourself »

The following SQL statement selects all customers that are from the same countries as the suppliers:

Example

```
SELECT * FROM Customers
WHERE Country IN (SELECT Country FROM Suppliers);
```

SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of that query.

An alias is created with the AS keyword.

Alias Column Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

Alias Table Syntax

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Alias for Columns Examples

The following SQL statement creates two aliases, one for the CustomerID column and one for the CustomerName column:

Example

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

Try it Yourself »

The following SQL statement creates two aliases, one for the CustomerName column and one for the ContactName column. **Note:** It requires double quotation marks or square brackets if the alias name contains spaces:

Example

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;
```

Try it Yourself »

The following SQL statement creates an alias named "Address" that combine four columns (Address, PostalCode, City and Country):

Example

```
SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' +
Country AS Address
FROM Customers;
```

Try it Yourself »

Note: To get the SQL statement above to work in MySQL use the following:

```
SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,',
',Country) AS Address
FROM Customers;
```

Alias for Tables Example

The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn). We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we use aliases to make the SQL shorter):

Example

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
```

The SQL BETWEEN Operator

The `BETWEEN` operator selects values within a given range. The values can be numbers, text, or dates.

The `BETWEEN` operator is inclusive: begin and end values are included.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example:

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

Try it Yourself »

SQL JOIN

A `JOIN` clause is used to combine rows from two or more tables, based on a related column between them.

Let's look at a selection from the "Orders" table:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- `(INNER) JOIN`: Returns records that have matching values in both tables

- `LEFT (OUTER) JOIN`: Returns all records from the left table, and the matched records from the right table
- `RIGHT (OUTER) JOIN`: Returns all records from the right table, and the matched records from the left table
- `FULL (OUTER) JOIN`: Returns all records when there is a match in either left or right table

SQL INNER JOIN Keyword

The `INNER JOIN` keyword selects records that have matching values in both tables.

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

SQL LEFT JOIN Keyword

The `LEFT JOIN` keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

129, 132, ~~134~~, 144, 147, 153, 166, 168, 171, 173, 183, 189, 413, 416.

SQL RIGHT JOIN Keyword

The `RIGHT JOIN` keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

SQL Comments

Any text between -- and the end of the line will be ignored (will not be executed).

--Select * from employee

SQL Self Join

Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

Example:

```
SELECT A.CustomerName AS CustomerName1,
B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

- Every SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in every SELECT statement must also be in the same order

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Example

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

SQL GROUP BY Statement

The `GROUP BY` statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The `GROUP BY` statement is often used with aggregate functions (`COUNT()`, `MAX()`, `MIN()`, `SUM()`, `AVG()`) to group the result-set by one or more columns.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

SQL GROUP BY Examples

The following SQL statement lists the number of customers in each country:

Example

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

Try it Yourself »

Example

```
SELECT COUNT(CustomerID), Country
FROM Customers
```

```
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

The SQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Example:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

SQL EXISTS Operato

The SQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

SQL EXISTS Examples

The following SQL statement returns TRUE and lists the suppliers with a product price less than 20:

Example

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID =
Suppliers.supplierID AND Price < 20);
```

The SQL ANY and ALL Operators

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

The SQL ANY Operator

The ANY operator:

- returns a boolean value as a result
- returns TRUE if ANY of the subquery values meet the condition

ANY means that the condition will be true if the operation is true for any of the values in the range.

ANY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

ALL Syntax With SELECT

```
SELECT ALL column_name(s)
FROM table_name
WHERE condition;
```

ALL Syntax With WHERE or HAVING

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

The SQL SELECT INTO Statement

The `SELECT INTO` statement copies data from one table into a new table.

SELECT INTO Syntax

Copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

Copy only some columns into a new table:

```
SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

SQL SELECT INTO Examples

The following SQL statement creates a backup copy of Customers:

```
SELECT * INTO CustomersBackup2017
FROM Customers;
```

The following SQL statement uses the IN clause to copy the table into a new table in another database:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

The following SQL statement copies only a few columns into a new table:

```
SELECT CustomerName, ContactName INTO CustomersBackup2017
FROM Customers;
```

The following SQL statement copies only the German customers into a new table:

```
SELECT * INTO CustomersGermany
FROM Customers
WHERE Country = 'Germany';
```

The following SQL statement copies data from more than one table into a new table:

```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup2017
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Tip: SELECT INTO can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

```
SELECT * INTO newtable
FROM oldtable
WHERE 1 = 0;
```

SQL | SOME

- Difficulty Level : Easy
- Last Updated : 21 Mar, 2018

SQL | ALL and ANY

SOME operator evaluates the condition between the outer and inner tables and

evaluates to true if the final result returns **any one** row. If not, then it evaluates to false.

- The SOME and ANY comparison conditions are similar to each other and are completely interchangeable.
- SOME must match at least one row in the subquery and must be preceded by comparison operators.

Syntax:

Attention reader! Don't stop learning now. Learn SQL for interviews using SQL Course by GeeksforGeeks.

```
SELECT column_name(s)
FROM table_name
WHERE expression comparison_operator SOME (subquery)
select name

from instructor

where Salary > some(select Salary

from instructor

where dept='Computer Science');
```

SQL | ALL and ANY

- Difficulty Level : Easy
- Last Updated : 09 Feb, 2018

ALL & ANY are logical operators in SQL. They return boolean value as a result.

ALL

Attention reader! Don't stop learning now. Get hold of all the important CS Theory concepts for SDE interviews with the **CS Theory Course** at a student-friendly price and become industry ready.

ALL operator is used to select all tuples of SELECT STATEMENT. It is also used to compare a value to every value in another value set or result from a subquery.

ALL with SELECT Statement:

Syntax:

```
SELECT ALL field_name
```

```
FROM table_name
WHERE condition(s);
```

ALL with WHERE or HAVING Statement:

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name comparison_operator ALL
(SELECT column_name
FROM table_name
WHERE condition(s));
```

Truncate statement

- To truncate Student_details table from student_data database.

```
TRUNCATE TABLE Student_details;
```

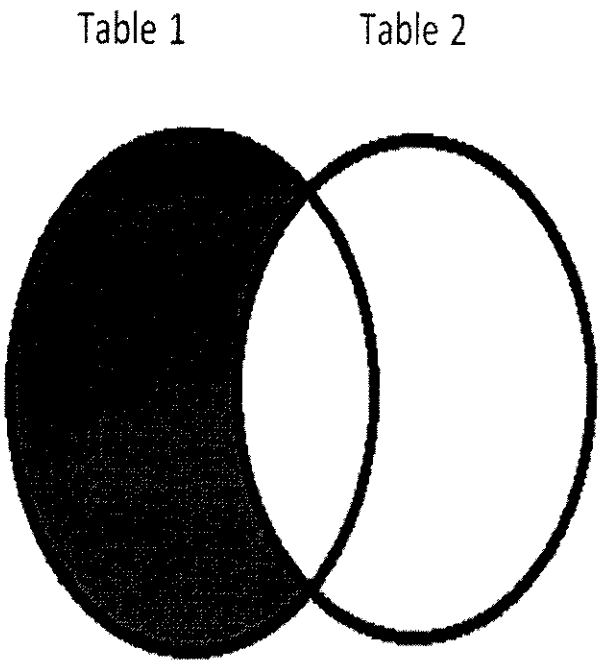
After running the above query Student_details table will be truncated, i.e, the data will be deleted but the structure will remain in the memory for further operations.

SQL | MINUS Operator

- Difficulty Level : Medium
- Last Updated : 09 Nov, 2020

The Minus Operator in SQL is used with two SELECT statements. The MINUS operator is used to subtract the result set obtained by first SELECT query from the result set obtained by second SELECT query. In simple words, we can say that MINUS operator will return only those rows which are unique in only first SELECT query and not those rows which are common to both first and second SELECT queries.

Pictorial Representation:



Basic Syntax:

```
SELECT column1 , column2 , ... columnN
FROM table_name
WHERE condition
MINUS
SELECT column1 , column2 , ... columnN
FROM table_name
WHERE condition;
```

columnN: column1, column2.. are the name of columns of the table.

Important Points:

- The WHERE clause is optional in the above query.
- The number of columns in both SELECT statements must be same.
- The data type of corresponding columns of both SELECT statement must be

```
SELECT NAME, AGE , GRADE
FROM Table1
MINUS
SELECT NAME, AGE, GRADE
FROM Table2
```

SQL | UNIQUE Constraint

- Difficulty Level : Medium
- Last Updated : 06 Sep, 2018

SQL Constraints

Unique constraint in SQL is used to check whether the sub query has duplicate tuples in it's result. It returns a boolean value indicating the presence/absence of duplicate tuples. Unique construct returns true only if the sub query has no duplicate tuples, else it return false.

```
SELECT table.ID
FROM  table
WHERE UNIQUE (SELECT table2.ID
              FROM table2
              WHERE table.ID = table2.ID);
```

More Complex SQL Retrieval Queries :

Comparisons Involving NULL and Three-valued Logic

SQL has various rules for dealing with NULL values. Consider the following examples to illustrate each of the meanings of NULL.

1. Unknown value
2. Unavailable or withheld value
3. Not applicable Attribute.

1 Unknown value : A person's date of birth is not known. So it is represented by NULL in the database.

2. Unavailable or withheld value: A person has a home phone but doesn't want it to be listed. So it is withheld and represented as NULL in the database.

3. Not applicable attribute : An attribute Last College Degree would be NULL for a person who has no college degrees because it doesn't apply to that person.

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

SQL> select Fname,Lname from emps where SUPERSSN IS NULL;

FNAME	LNAME
Ahamad	jabbar

Nested Queries, Tuples and Set/Multiset Comparisons:

Some queries require that existing value in the database be fetched and then used in a comparison condition. Such queries can be conveniently formulated by using nested queries.

```
SQL> select DISTINCT Pnumber from PROJECT where Pnumber IN(Select Pnumber from
project,depts,emps
 where Dnum=Dnumber AND Mgrssn=SSN AND Lname='smith')
OR
Pnumber IN
(Select Pno from WORKS_ON, EMPS where Essn=ssn and Lname='smith');

 PNUMBER
---------
       1
       2
```

SQL allows the use of tuples of values in comparisons by placing them within parentheses.

```
SQL> select DISTINCT Pnumber from PROJECT where Pnumber IN(Select Pnumber from
project,depts,emps
  2  where Dnum=Dnumber AND Mgrssn=SSN AND Lname='smith')
  3  OR
  4  Pnumber IN
  5  (Select Pno from WORKS_ON, EMPS where Essn=ssn and Lname='smith');

 PNUMBER
---------
       1
       2
```

In addition to the IN operator, a number of other comparisons operators can be used to compare a single value v.

→ The = ANY operator returns TRUE if the value v is equal to some value in the set V and is hence equivalent to IN.

→ The two keywords in equal to some value the same effect.

→ Other operators that can be combined with ANY include >, <=, >, <=, and <>.

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

LNAME	FNAME
smith	john
wong	Franklin
Zelaya	alicia
wallace	jennifer
Narayan	Ramesh
English	joyce
jabbar	Ahamad
Borg	james

Correlated Nested queries :

Whenever a condition in the WHERE clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be correlated.

→ Nested query can be understand better by considering the nested query is evaluated once for each tuple in the outer query.

```
SELECT  E.Fname, E.Lname
  From Employee as E, Department as D
  where  E.SSN = D.ESSN AND E.Sex = D.Sex
    AND  E.Fname = D.Dependent_name;
```

EXISTS AND UNIQUE Functions in SQL :

The EXISTS function in SQL is used to check whether result of a correlated nested query is empty or not.

→ The result of exists is a Boolean value TRUE if the nested query result contains at least one tuple or FALSE if the nested query result contains no tuples.

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

no rows selected
```

EXISTS AND NOT EXISTS are typically used in conjunction with a correlated nested query. given above.

query: Retrieve names of employees who have no dependents

```
SQL> SELECT E.FNAME,E.LNAME FROM EMPS E WHERE NOT EXISTS(SELECT * FROM
DEPENDENT D WHERE E.SSN=D.ESS
N AND  E.SEX=D.SEX AND E.FNAME=D.DEPENDANT_NAME);

FNAME           LNAME
--------------- ---------------
john            smith
Franklin         wong
Alice           Zelaya
jennifer        wallace
Ramesh           Narayan
joyce           English
Ahamad           jabbar
james           Borg
```

List the name of manager who have one dependent -

```
SQL> SELECT FNAME,LNAME
 FROM EMPLOYEE
 WHERE EXISTS
 (SELECT * FROM DEPENDENT
 WHERE SSN=ESSN)
 AND
 EXISTS
 (SELECT * FROM DEPARTMENT
 WHERE SSN=MGRSSN);

FNAME           LNAME
--------------- ---------------
Franklin        Wong
Jennifer        Wallace
```

Example 4: Multiway Join.

```
SQL> SELECT PNUMBER,DNUM,LNAME,ADDRESS,BDATE
 FROM ((PROJECT JOIN DEPARTMENT ON DNUM=DNUMBER)
 JOIN EMPLOYEE ON MGRSSN=SSN)
 WHERE PLOCATION='STAFFORD';

no rows selected
```

Aggregate Functions in SQL :

Aggregate functions are used to summarize information from multiple tuples into a single-tuple summary.

→ Grouping is used to create subgroups of tuples before summarization.

→ A number of built-in aggregate function exists: COUNT, SUM, MAX, MIN and AVG.

Query : Find the sum of the salaries of all employee the maximum salary, the maximum salary and the average salary.

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

SUM(SALARY) MAX(SALARY) MIN(SALARY) AVG(SALARY)
----------- ----------- ----------- -----------
```

Query:

Find the sum of the salaries of all employee of the Research department as well as the maximum salary, the minimum salary and the average salary in this department.

```
SQL> SELECT SUM(SALARY),MAX(SALARY),MIN(SALARY),AVG(  LARY)
 FROM (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
 WHERE DNAME='RESEARCH';

SUM(SALARY) MAX(SALARY) MIN(SALARY) AVG(SALARY)
----------- ----------- ----------- -----------
```

Queries : Retrieve the total number of employees in the company and the no of employeees in the research department –

1

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

 COUNT(*)
----------
        8
```

2.

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

 COUNT(*)
----------
        0
```

Query : Count the number of distinct Salary values in the database.

```
SQL> SELECT COUNT(DISTINCT SALARY)
FROM EMPLOYEE;

COUNT(DISTINCTSALARY)
---------------------
                    6
```

Above query eliminates the duplicate values (redundant) However, any tuples with NULL for SALARY will be counted. When Aggregate functions are used, NULL values are discarded.

The preceding example summarize a whole relation, or a selected subset of tuples

They illustrate how the functions are applied to retrieve a summary values and summary tuple from a database

– We can specify a co-related nested query with an aggregate function.

Retrieve the names of each employee who works on all the projects controlled by department no 5.

```
SQL> SELECT LNAME,FNAME
 FROM EMPLOYEE
 WHERE NOT EXISTS
 (SELECT * FROM WORKS_ON B
 WHERE (B.PNO IN (SELECT PNUMBER FROM PROJECT WHERE DNUM=5)
 AND
 NOT EXISTS
 (SELECT * FROM WORKS_ON C WHERE C.ESSN=SSN AND C.PNO=B.PNO)));

no rows selected
```

Explicit Sets and Remaining of Attributes in SQL:

The where clause even allows "explicit set of values"

query: Retrieve the social security numbers of all employee who work on project number 1, 2 & 3.

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

ESSN
---------
333445555
453453453
123456789
666884444
```

query: To retrieve lname of employee as employee name, supervisor name.

```
SQL> SELECT E.LNAME AS EMPLOYEE_NAME, S.LNAME AS SUPERVISOR_NAME
 FROM EMPLOYEE E,EMPLOYEE S
 WHERE E.SUPERSSN=S.SSN;

EMPLOYEE_NAME   SUPERVISOR_NAME
--------------- ---------------
English         Wong
Narayan         Wong
Smith           Wong
Jabbar          Wallace
Zelaya          Wallace
Wallace         Borg
Wong            Borg
```

Joined Tables in SQL and Outer joins :

The Concept of a joined table was incorporated into SQL to permit users to specify a table resulting from a join operation in the FROM Clause of a query.

→ This Construct may be easier to comprehend than mixing together all the Select and join Conditions in the WHERE Clause.

Example 1 : JOIN operation (Equi join)

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

no rows selected
```

Select fname, lname, address from employee join department on Employee.dno = department.dno

Example 2 : Natural join.

```
SQL> SELECT Fname, Lname, Address
     FROM (EMPLOYEE NATURAL JOIN
        (DEPARTMENT AS DEPT (Dname, Dno, Mssn,
                             Msdate))
     Where Dname = 'Research'
```

Select fname, lame, address from employee join dependent on Employee.dno = department.dno

Example 3 : OUTER JOIN :

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

EMPLOYEE_NAME   SUPERVISOR_NAME
--------------- ---------------
English         Wong
Narayan         Wong
Smith           Wong
Jabbar          Wallace
Zelaya          Wallace
Wallace         Borg
Wong            Borg
Borg
```

Select * from works_on left outer join project on works_on.pno = project.pnum ;

```
SQL> SELECT FNAME,LNAME
 FROM EMPLOYEE
 WHERE(SELECT COUNT(*)
 FROM DEPENDENT
 WHERE SSN=ESSN)>=2;

FNAME           LNAME
--------------- ---------------
John            Smith
Franklin        Wong
```

Grouping :

In doing sometimes it require to apply the aggregate functions to subgroups of tuples in the relation, where subgroups are based on some attribute values.

⟶ SQL has Group By clause for this purpose. The Group By clause specifies the grouping attribute which ~~should~~ should also appear in where clause.

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

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

       DNO   COUNT(*) AVG(SALARY)
---------- ---------- -----------
         1          1       55000
         5          4       33250
         4          3       31000
```

Query 25: For each project, retrieve the project number, the Project name, and the number of employee who work on the project-

~~SQL Query is~~ P.T.O.

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

  PNUMBER PNAME          COUNT(*)
--------- --------------- ----------
       20 Reorganization          3
        1 ProductX                2
       10 Computerization         3
       30 Newbenefits             3
        2 ProductY                3
        3 ProductZ                2
```

Having Caluse :

SQL provides Having clause, which can appear in Conjunction with a Group By clause. For this purpose. Having provides a Condition on the Summary information regarding the group of tuples associated with each value of the grouping attribute. Only the group that Satisfy the condition are retrieved in the result of the query.

ex : query : For each project on which more than two employee work, retrieve the project-number, the project-name, and the number of employee who work on the project.

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

  PNUMBER PNAME          COUNT(*)
--------- --------------- ----------
       20 Reorganization          3
       10 Computerization         3
       30 Newbenefits             3
        2 ProductY                3
```

Query 2: To count the total number of employee whose salaries exceed $ 40,000 in each department, but only for departments where more than five employee work. Here the condition (SALARY > 40000) applies only to the COUNT function in the SELECT Clause.

```
SQL> SELECT DNAME,COUNT(*)
FROM DEPARTMENT,EMPLOYEE
WHERE DNUMBER=DNO AND SALARY>40000
GROUP BY DNAME
HAVING COUNT(*)>5;

no rows selected
```

Specifying Constraints as Assertions and Action as Triggers

In this section we introduce two additional features of SQL. the CREATE ASSERTION statement and the CREATE TRIGGER statement.

→ CREATE ASSERTION can be used to specify additional types of constraints that are outside the scope of the built-in relational model constraints.

→ 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.

```
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.Mgr_SSN = 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.

Note: CHECK clause and constraint-condition can also be used to specify constraints on individual attributes and domains.

Triggers in SQL:

Trigger is one of the special feature of SQL. In many cases it is convienient to specify the type of action to be taken when certain events occur and when certain conditions are satisfied.

ex: A manager may want to be informed if an employee's if an employee's travel expenses exceed a certain limit by receiving a message whenever this occurs. This action that the DBMS must take in this case is to send an appropriate message to that user. This condition is used to monitor the database.

Query: To create a trigger in SQL.

```
CREATE TRIGGER SALARY_VIOLATION BEFORE
INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
ON EMPLOYEE
    FOR EACH ROW
        WHEN ( NEW SALARY > ( SELECT SALARY FROM
EMPLOYEE WHERE SSN = NEW.SUPERVISOR_SSN ) )
    . INFORM_SUPERVISOR ( NEW. Supervisor_SSN,
                          NEW. SSn).
```

Views in SQL (virtual Tables)

A view (virtual table) in SQL terminology is a single that is derived from other tables. These other tables can be tables or previously defined views.

→ A view does not necessarily exist in physical form; it is considered to be a virtual table.

Specification of views in SQL:

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

example:

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

View created.
```

query: To view the contents of WORKS-ON1 view.

```
SQL> SELECT * FROM WORKS_ON1;
```

FNAME	LNAME	PNAME	HOURS
John	Smith	ProductY	7.5
John	Smith	ProductX	32.5
Franklin	Wong	Reorganization	10
Franklin	Wong	Computerization	10
Franklin	Wong	ProductZ	10
Franklin	Wong	ProductY	10
Alicia	Zelaya	Computerization	10
Alicia	Zelaya	Newbenefits	30
Jennifer	Wallace	Reorganization	15
Jennifer	Wallace	Newbenefits	20
Ramesh	Narayan	ProductZ	40

FNAME	LNAME	PNAME	HOURS
Joyce	English	ProductY	20
Joyce	English	ProductX	20
Ahmad	Jabbar	Newbenefits	5
Ahmad	Jabbar	Computerization	35
James	Borg	Reorganization	

ex2:

```
SQL> CREATE VIEW DEPT_INFO(DEPT_NAME,NO_OF_EMPS,TOTAL_SAL) AS
 SELECT DNAME,COUNT(*),SUM(SALARY)
 FROM DEPARTMENT,EMPLOYEE
 WHERE DNUMBER=DNO
 GROUP BY DNAME;

View created.


SQL> SELECT * FROM DEPT_INFO;

DEPT_NAME      NO_OF_EMPS  TOTAL_SAL
-------------- ---------- ----------
Research                4     133000
Administration          3      93000
Headquarters            1      55000
```

Like a drop table, SQL also permits ~~ups~~ to drop a view. Syntax to drop a view is given below.

ex drop view viewname;

```
SQL> drop view works_on1;

View dropped.
```

SQL also permits to update the views like table. The example to update the view is given below.

```
SQL> Update Dept_INFO
     SET Total_Sal = 100000
     WHERE Dname = 'Research';
```

Difference Between JDBC and ODBC

ODBC	JDBC
1) ODBC Stands for Open Database Connectivity	1) JDBC Stands for Java Database Connectivity
2) Introduced by Microsoft.	2) Introduced by Sun Micro Systems.
3) We can Use ODBC for any Languages like C, C++, Java, Etc.	3) We can Use JDBC only for Java Language.
4) We can use ODBC only for Windows Platforms.	4) We can use JDBC for any Platform.
5) Mostly ODBC Drivers are developed in Native Languages like C OR C++.	5) Mostly JDBC Drivers are developed in Java.
6) For Java Applications, it is not recommended to use ODBC because Performance will be Down due to Internal Conversions and Application will become Platform Dependent.	6) For Java Applications, it is highly recommended to use JDBC because there is no Performance Problems and Platform Dependency Problems.

SQLJ

Definition

SQLJ is a set of programming extensions that allow a programmer using the Java programming language to embed statements that provide SQL (Structured Query Language) database requests. SQLJ is similar to existing extensions for SQL that are provided for C, Formula Translation, and other programming languages. IBM, Oracle, and several other companies are proposed SQLJ as a standard and as a simpler and easier-to-use alternative to Java Database Connectivity.

The SQLJ specifications are in several parts:

SQLJ: Embedded SQL

(Specifications for embedding SQL statements in Java methods.)

This provides a somewhat more object-oriented approach to the standard way of embedding SQL statements in programs. Part 0 supports static SQL statements in Java. It does not support dynamic SQL statements. Those are handled by JDBC. Part 0 does support mixing embedded static SQL statements with JDBC statements. Part

0 supports the same mapping between Java data types and SQL data types that is defined by JDBC. Also see the SQLJ Execution Environment.

SQLJ: SQL Routines

(Specifications for calling Java static methods as SQL stored procedures and user-defined functions.)

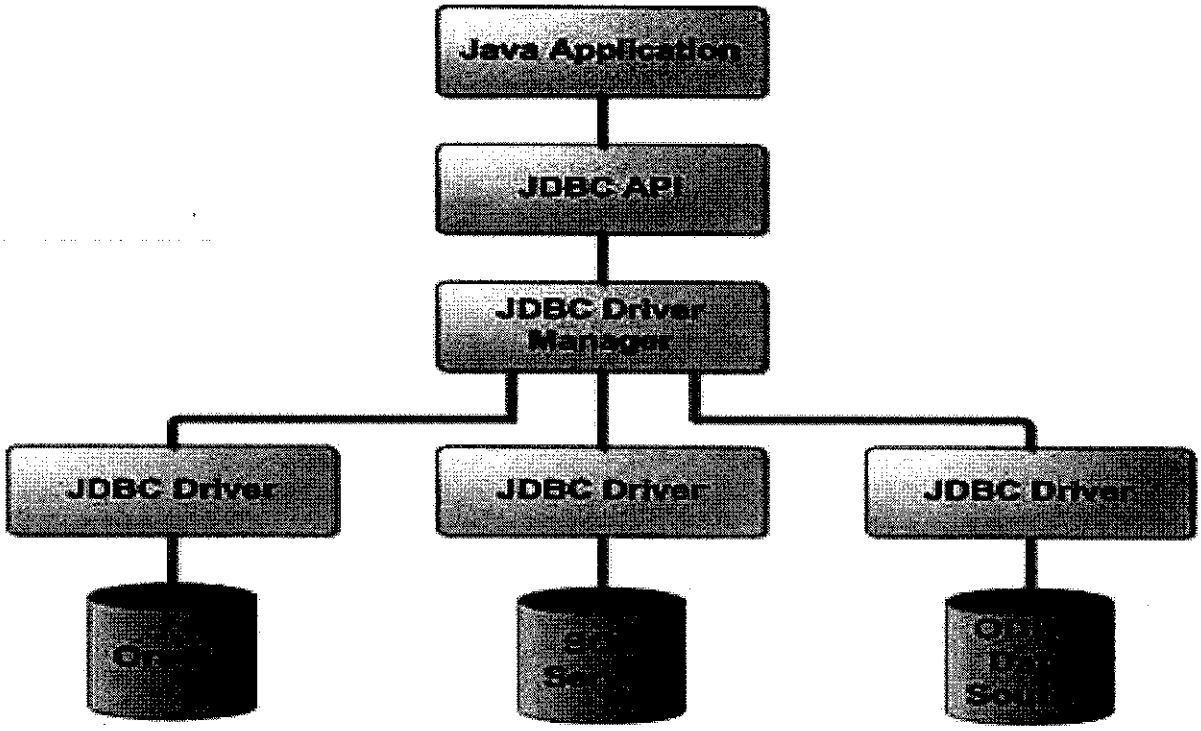
This provides the ability to invoke methods written in Java from SQL code. The Java methods provide the implementation of SQL procedures. To support Part 1, a DBMS must have a Java Virtual Machine associated with it. Part 1 deals only with static methods. For an association between SQL functions and Java methods, each SQL parameter and its corresponding Java parameter must be able to be mapped and the two return types must be able to be mapped.

SQLJ: SQL Types

(Specifications for using Java classes as SQL user-defined data types.)

This defines SQL extensions for using Java classes as data types in SQL. Part 2 allows mapping of SQL:1999 User Defined Types (UDTs) to Java classes. It also allows importing a Java package into your SQL database by defining tables containing columns whose data type are specified to be a Java class. Structured types are associated with classes, attributes with fields, and initializers to constructors. All or part of an SQL type hierarchy can be represented in a Java class hierarchy. It is not necessary to associate the entire SQL type hierarchy to a Java hierarchy. Part 2 adds non-static methods to the static methods in Part1.

ARCHITECTURE OF JDBC



- It consists of 4 main components:
    - Java Application
    - JDBC Driver Manager
    - JDBC Driver
    - Data Source
- JDBC Driver Manager -
    - It loads JDBC drivers and passes JDBC the

function calls from the application to the correct driver

- JDBC Driver Manager –
    - It establishes the connection with the

data source and submits requests and returns request results.

- It translates data, error formats and error codes into the JDBC standard.

- Data Source –
    - It processes commands from the driver and returns the results

Steps to Connect

1. Define connection URL
2. Establish the connection
3. Create the Statement object
4. Execute a Query
5. Process the results
6. Close the connection

1. Class.forName("oracle.jdbc.driver.OracleDriver");

2. Connection con = DriverManager.getConnection
   ("jdbc:oracle:thin:@localhost:1521:xe", "USERNAME", "PWD");

3. Statement stmt = con.createStatement();

4. // SELECT Query

   ResultSet rs = stmt.executeQuery ("select * from MOVIES");

5. while(rs.next()){

   System.out.println(

   rs.getInt(1) + " " + rs.getString(2) + " " + rs.getInt(3) + " " + rs.getString(4) + " " + rs.getInt(5)

   );

}

6. stmt.close();

   con.close();

WHAT IS XML

- eXtensible Markup Language
- designed to store and transport data
- XML can be used in complement with HTML

WHAT IS XSL

- eXtensible StyleSheet Language
- XSL Consists of three parts:
  - XSL Tranformations (XSLT) : Transforming XML documents
  - XML Path Language (X Path) : navigate through parts of an XML document
  - XML Formatting Objects (XSL - FO) : XML vocabulary for specifying formatting semantics (format output of xsl transformation)

HTML vs XML

HTML is used to display the data XML is used to store and transport the data

HTML tags are pre-defined.

They are not case-sensitive.

Closing tags not necessary.

XML atgs are user-defined.

They are case-sensitive.

Closing tags are necessary.

Limited number of tags. They are extensible. User can create as many required.

CSS is used for styling HTML files.

The result can be directly viewed on

a browser.

XSL is used for styling XML files.

Can't be viewed like a html file.

SIMPLE XML EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
 <note>
 <to>Ram</to>
 <from>Sita</from>
 <heading>Information</heading>
 <body>I am in Lanka</body>
 </note>
```

SIMPLE XSL EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/"> // namespace
<html>
 <body >
 <style>
 body
 {
 background-color: lightblue;
 color: navy;
 }
</style>
 <h1>MESSAGE</h1>
 <h5>FROM: <xsl:value-of select="note/from"/></h5>
```

<h5>TO: <xsl:value-of select="note/to"/></h5>

<h3>PURPOSE: <xsl:value-of select="note/heading"/></h3>

<h4>CONTENT: <xsl:value-of select="note/body"/></h4>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

LINKING XML AND XSL

<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="first.xsl"?>

<note>

<to>Ram</to>

<from>Sita</from>

<heading>Information</heading>

<body>I am in Lanka</body>

</note>

WHERE TO EXECUTE : DEMO

XML VIEWER AND EDITOR

https://www.w3schools.com/xml/tryxslt.asp?xmlfile

=cdcatalog&xsltfile=cdcatalog

USES OF XML

- XML Separates Data from Presentation
- XML is Often a Complement to HTML
- XML Separates Data from HTML
- With a few lines of JavaScript code, you can read an XML

file and update the data content of any HTML page.

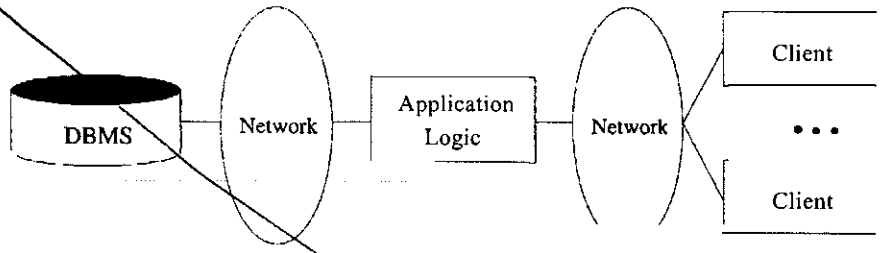


Figure 7.8 A Standard Three-Tier Architecture

(Stored procedures can mitigate this bottleneck.) Fourth, thick-client systems do not scale as the application accesses more and more database systems. Assume there are x different database systems that are accessed by y clients, then there are $x \cdot y$ different connections open at any time, clearly not a scalable solution.

These disadvantages of thick-client systems and the widespread adoption of standard, very thin clients—notably, Web browsers—have led to the widespread use thin-client architectures.

7.5.2 Three-Tier Architectures

The thin-client two-tier architecture essentially separates presentation issues from the rest of the application. The three-tier architecture goes one step further, and also separates application logic from data management:

- Presentation Tier: Users require a natural interface to make requests, provide input, and to see results. The widespread use of the Internet has made Web-based interfaces increasingly popular.
- Middle Tier: The application logic executes here. An enterprise-class application reflects complex business processes, and is coded in a general purpose language such as C++ or Java.
- Data Management Tier: Data-intensive Web applications involve DBMSs, which are the subject of this book.

Figure 7.8 shows a basic three-tier architecture. Different technologies have been developed to enable distribution of the three tiers of an application across multiple hardware platforms and different physical sites. Figure 7.9 shows the technologies relevant to each tier.

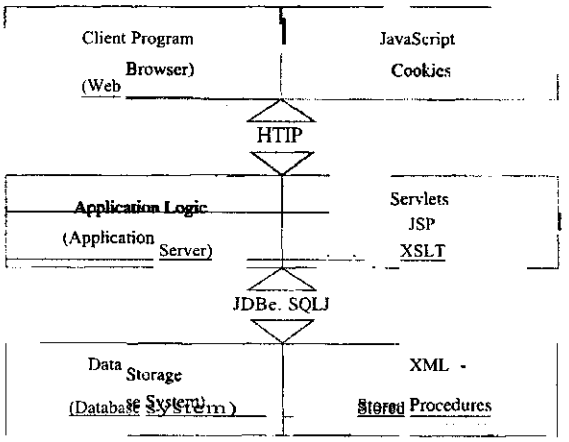


Figure 7.9 Technologies for the Three Tiers

Overview of the Presentation Tier

At the presentation layer, we need to provide forms through which the user can issue requests, and display responses that the middle tier generates. The hypertext markup language (HTML) discussed in Section 7.3 is the basic data presentation language.

It is important that this layer of code be easy to adapt to different display devices and formats; for example, regular desktops versus handheld devices versus cell phones. This adaptivity can be achieved either at the middle tier through generation of different pages for different types of client, or directly at the client through style sheets that specify how the data should be presented. In the latter case, the middle tier is responsible for producing the appropriate data in response to user requests, whereas the presentation layer decides *how* to display that information.

We cover presentation tier technologies, including style sheets, in Section 7.6.

Overview of the Middle Tier

The middle layer runs code that implements the business logic of the application: It controls what data needs to be input before an action can be executed, determines the control flow between multi-action steps, controls access to the database layer, and often assembles dynamically generated HTML pages from database query results.

The middle tier code is responsible for supporting all the different roles involved in the application. For example, in an Internet shopping site implementation, we would like customers to be able to browse the catalog and make purchases, administrators to be able to inspect current inventory, and possibly data analysts to ask summary queries about purchase histories. Each of these roles can require support for several complex actions.

For example, consider the a customer who wants to buy an item (after browsing or searching the site to find it). Before a sale can happen, the customer has to go through a series of steps: She has to add items to her shopping basket, she has to provide her shipping address and credit card number (unless she has an account at the site), and she has to finally confirm the sale with tax and shipping costs added. Controlling the flow among these steps and remembering already executed steps is done at the middle tier of the application. The data carried along during this series of steps might involve database accesses, but usually it is not yet permanent (for example, a shopping basket is not stored in the database until the sale is confirmed).

We cover the middle tier in detail in Section 7.7.

7.5.3 Advantages of the Three-Tier Architecture

The three-tier architecture has the following advantages:

- Heterogeneous Systems: Applications can utilize the strengths of different platforms and different software components at the different tiers. It is easy to modify or replace the code at any tier without affecting the other tiers.
- **Thin** Clients: Clients only need enough computation power for the presentation layer. Typically, clients are Web browsers.
- Integrated Data Access: In many applications, the data must be accessed from several sources. This can be handled transparently at the middle tier, where we can centrally manage connections to all database systems involved.
- Scalabilit,y to Many Clients: Each client is lightweight and all access to the system is through the middle tier. The middle tier can share database connections across clients, and if the middle tier becomes the bottle-neck, we can deploy several servers executing the middle tier code; clients can connect to anyone of these servers, if the logic is designed appropriately. This is illustrated in Figure 7.10, which also shows how the middle tier accesses multiple data sources. Of course, we rely upon the DBMS for each

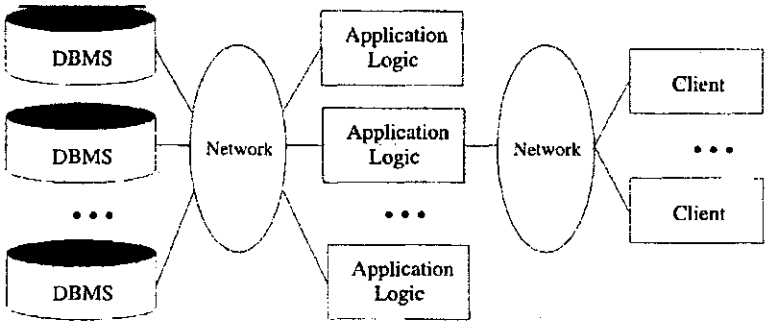


Figure 7.10 Middle-Tier Replication and Access to Multiple Data Sources

data source to be scalable (and this might involve additional parallelization or replication, as discussed in Chapter 22).

- Software Development Benefits: By dividing the application cleanly into parts that address presentation, data access, and business logic, we gain many advantages. The business logic is centralized, and is therefore easy to maintain, debug, and change. Interaction between tiers occurs through well-defined, standardized APIs. Therefore, each application tier can be built out of reusable components that can be individually developed, debugged, and tested.

7.6 THE PRESENTATION LAYER

In this section, we describe technologies for the client side of the three-tier architecture. We discuss HTML forms as a special means of passing arguments from the client to the middle tier (i.e., from the presentation tier to the middle tier) in Section 7.6.1. In Section 7.6.2, we introduce JavaScript, a Java-based scripting language that can be used for light-weight computation in the client tier (e.g., for simple animations). We conclude our discussion of client-side technologies by presenting style sheets in Section 7.6.3. Style sheets are languages that allow us to present the same webpage with different formatting for clients with different presentation capabilities; for example, Web browsers versus cell phones, or even a Netscape browser versus Microsoft's Internet Explorer.

7.6.1 HTML Forms

HTML forms are a common way of communicating data from the client tier to the middle tier. The general format of a form is the following:

```
<FORM ACTION="page.jsp" METHOD="GET" NAME="LoginForm">
```

```
</FORM>
```

A single HTML document can contain more than one form. Inside an HTML form, we can have any HTML tags except another FORM element.

The FORM tag has three important attributes:

- ACTION: Specifies the URI of the page to which the form contents are submitted; if the ACTION attribute is absent, then the URI of the current page is used. In the sample above, the form input would be submited to the page named page.jsp, which should provide logic for processing the input from the form. (We will explain methods for reading form data at the middle tier in Section 7.7.)
- METHOD: The HTTP/1.0 method used to submit the user input from the filled-out form to the webserver. There are two choices, GET and POST; we postpone their discussion to the next section.
- NAME: This attribute gives the form a name. Although not necessary, naming forms is good style. In Section 7.6.2, we discuss how to write client-side programs in JavaScript that refer to forms by name and perform checks on form fields.

Inside HTML forms, the INPUT, SELECT, and TEXTAREA tags are used to specify user input elements; a form can have many elements of each type. The simplest user input element is an INPUT field, a standalone tag with no terminating tag. An example of an INPUT tag is the following:

```
<INPUT TYPE=ltext" NAME="title">
```

The INPUT tag has several attributes. The three most important ones are TYPE, NAME, and VALUE. The TYPE attribute determines the type of the input field. If the TYPE attribute has value `text`, then the field is a text input field. If the TYPE attribute has value password, then the input field is a text field where the entered characters are displayed as stars on the screen. If the TYPE attribute has value `reset`, it is a simple button that resets all input fields within the form to their default values. If the TYPE attribute has value submit, then it is a button that sends the values of the different input fields in the form to the server. Note that `reset` and submit input fields affect the entire form.

The NAME attribute of the INPUT tag specifies the symbolic name for this field and is used to identify the value of this input field when it is sent to the server. NAME has to be set for INPUT tags of all types except submit and `reset`. In the preceding example, we specified `title` as the NAME of the input field.

The VALUE attribute of an input tag can be used for text or password fields to specify the default contents of the field. For submit or reset buttons, VALUE determines the label of the button.

The form in Figure 7.11 shows two text fields, one regular text input field and one password field. It also contains two buttons, a reset button labeled 'Reset Values' and a submit button labeled 'Log on.' Note that the two input fields are named, whereas the reset and submit button have no NAME attributes.

```
<FORM ACTION="page.jsp" METHOD="GET" NAME="LoginForm">
    <INPUT TYPE="text" NAME="username" VALUE="Joe"><P>
    <INPUT TYPE="password" NAME="p&ssword"><P>
    <INPUT TYPE="reset" VALUE="Reset Values"><P>
    <INPUT TYPE="submit" VALUE="Log on">
</FORM>
```

Figure 7.11 HTML Form with Two Text Fields and Two Buttons

HTML forms have other ways of specifying user input, such as the aforementioned TEXTAREA and SELECT tags; we do not discuss them.

Passing Arguments to **Server-Side** Scripts

As mentioned at the beginning of Section 7.6.1, there are two different ways to submit HTML Form data to the webserver. If the method GET is used, then the contents of the form are assembled into a query URI (as discussed next) and sent to the server. If the method POST is used, then the contents of the form are encoded as in the GET method, but the contents are sent in a separate data block instead of appending them directly to the URI. Thus, in the GET method the form contents are directly visible to the user as the constructed URI, whereas in the POST method, the form contents are sent inside the HTTP request message body and are not visible to the user.

Using the GET method gives users the opportunity to bookmark the page with the constructed URI and thus directly jump to it in subsequent sessions; this is not possible with the POST method. The choice of GET versus POST should be determined by the application and its requirements.

Let us look at the encoding of the URI when the GET method is used. The encoded URI has the following form:

```
action?name1=value1&name2=value2&name3=value3
```

The action is the URI specified in the ACTION attribute to the FORM tag, or the current document URI if no ACTION attribute was specified. The 'name=value' pairs are the user inputs from the INPUT fields in the form. For form INPUT fields where the user did not input anything, the name is stil present with an empty value (name=). As a concrete example, consider the password submission form at the end of the previous section. Assume that the user inputs 'John Doe' as username, and 'secret' as password. Then the request URI is:

page.jsp?username=John+Doe&password=secret

The user input from forms can contain general ASCII characters, such as the space character, but URIs have to be single, consecutive strings with no spaces. Therefore, special characters such as spaces, '=', and other unprintable characters are encoded in a special way. To create a URI that has form fields encoded, we perform the following three steps:

1. Convert all special characters in the names and values to '%xyz,' where 'xyz' is the ASCII value of the character in hexadecimal. Special characters include =, &, %, +, and other unprintable characters. Note that we could encode *all* characters by their ASCII value.
2. Convert all space characters to the '+' character.
3. Glue corresponding names and values from an individual HTML INPUT tag together with '=' and then paste name-value pairs from different HTML INPUT tags together using'&' to create a request URI of the form: action?name1=value1&name2=value2&name3=value3

Note that in order to process the input elements from the HTML form at the middle tier, we need the ACTION attribute of the FORM tag to point to a page, script, or program that will process the values of the form fields the user entered. We discuss ways of receiving values from form fields in Sections 7.7.1 and 7.7.3.

7.6.2 JavaScript

JavaScript is a scripting language at the client tier with which we can add programs to webpages that run directly at the client (Le., at the machine running the Web browser). JavaScript is often used for the following types of computation at the client:

- Browser Detection: JavaScript can be used to detect the browser type and load a browser-specific page.
- Form Validation: JavaScript is used to perform simple consistency checks on form fields. For example, a JavaScript program might check whether a

form input that asks for an email address contains the character '@,' or if all required fields have been input by the user.

- **Browser Control:** This includes opening pages in customized windows; examples include the annoying pop-up advertisements that you see at many websites, which are programmed using JavaScript.

JavaScript is usually embedded into an HTML document with a special tag, the SCRIPT tag. The SCRIPT tag has the attribute LANGUAGE, which indicates the language in which the script is written. For JavaScript, we set the language attribute to JavaScript. Another attribute of the SCRIPT tag is the SRC attribute, which specifies an external file with JavaScript code that is automatically embedded into the HTML document. Usually JavaScript source code files use a '.js' extension. The following fragment shows a JavaScript file included in an HTML document:

```
<SCRIPT LANGUAGE=" JavaScript" SRC="validateForm.js"> </SCRIPT>
```

The SCRIPT tag can be placed inside HTML comments so that the JavaScript code is not displayed verbatim in Web browsers that do not recognize the SCRIPT tag. Here is another JavaScipt code example that creates a pop-up box with a welcoming message. We enclose the JavaScipt code inside HTML comments for the reasons just mentioned.

```
<SCRIPT LANGUAGE=" JavaScript" >
<!--
    alert("Welcome to our bookstore");
//-->
</SCRIPT>
```

JavaScript provides two different commenting styles: single-line comments that start with the '//' character, and multi-line comments starting with '/*' and ending with ,*/' characters.[1]

JavaScript has variables that can be numbers, boolean values (true or false), strings, and some other data types that we do not discuss. Global variables have to be declared in advance of their usage with the keyword var, and they can be used anywhere inside the HTML documents. Variables local to a JavaScript function (explained next) need not be declared. Variables do not have a fixed type, but implicitly have the type of the data to which they have been assigned.

[1]Actually, '<!--' also marks the start of a single-line comment, which is why we did not have to mark the HTML starting cormnent '<!--' in the preceding example using JavaScript comment notation. In contrast, the HTML closing comment "-->" has to be commented out in JavaScript as it is interpreted otherwise.

27. XSL also contains XSL Formatting Object, a way of formatting the output of an XSL transformation.

7.7 THE MIDDLE TIER

In this section, we discuss technologies for the middle tier. The first generation of middle-tier applications were stand-alone programs written in a general-purpose programming language such as C, C++, and Perl. Programmers quickly realized that interaction with a stand-alone application was quite costly; the overheads include starting the application every time it is invoked and switching processes between the webserver and the application. Therefore, such interactions do not scale to large numbers of concurrent users. This led to the development of the application server, which provides the run-time environment for several technologies that can be used to program middle-tier application components. Most of today's large-scale websites use an application server to run application code at the middle tier.

Our coverage of technologies for the middle tier mirrors this evolution. We start in Section 7.7.1 with the Common Gateway Interface, a protocol that is used to transmit arguments from HTML forms to application programs running at the middle tier. We introduce application servers in Section 7.7.2. We then describe technologies for writing application logic at the middle tier: Java servlets (Section 7.7.3) and Java Server Pages (Section 7.7.4). Another important functionality is the maintenance of state in the middle tier component of the application as the client component goes through a series of steps to complete a transaction (for example, the purchase of a market basket of items or the reservation of a flight). In Section 7.7.5, we discuss Cookies, one approach to maintaining state.

7.7.1 CGI: The Common Gateway Interface

The Common Gateway Interface connects HTML forms with application programs. It is a protocol that defines how arguments from forms are passed to programs at the server side. We do not go into the details of the actual CGI protocol since libraries enable application programs to get arguments from the HTML fonn; we shortly see an example in a CGI program. Programs that communicate with the webserver via CGI are often called CGI scripts, since many such application programs were written in a scripting language such as Perl.

As an example of a program that interfaces with an HTML form via CGI, consider the sample page shown in Figure 7.14. This webpage contains a form where a user can fill in the name of an author. If the user presses the 'Send

```
<HTML><HEAD><TITLE>The Database Bookstore</TITLE></HEAD>
<BODY>
<FORM ACTION="find_books.cgi" METHOD=POST>
    Type an author name:
    <INPUT TYPE="text" NAME=lauthorName"
           SIZE=30 MAXLENGTH=50>
    <INPUT TYPE="submitil value="Send it">
    <INPUT TYPE=lreset" VALUE="Clear form" >
</FORM>
</BODY></HTML>
```

Figure 7.14 A Sample Web Page Where Form Input Is Sent to a CGI Script

it' button, the Perl script 'findBooks.cgi' shown in Figure 7.14 is executed as a separate process. The CGI protocol defines how the communication between the form and the script is performed. Figure 7.15 illustrates the processes created when using the CGI protocol.

Figure 7.16 shows the example CGI script, written in Perl. We omit error-checking code for simplicity. Perl is an interpreted language that is often used for CGI scripting and many Perl libraries, called **modules,** provide high-level interfaces to the CGI protocol. \Ve use one such library, called the **DBI library,** in our example. The CGI module is a convenient collection of functions for creating CGI scripts. In part 1 of the sample script, we extract the argument of the HTML form that is passed along from the client as follows:

```
$authorName = $dataln- >paramCauthorName');
```

Note that the parameter name authorName was used in the form in Figure 7.14 to name the first input field. Conveniently, the CGI protocol abstracts the actual implementation of how the webpage is returned to the Web browser; the webpage consists simply of the output of our program, and we start assembling the output HTML page in part 2. Everything the script writes in print-statements is part of the dynamically constructed webpage returned to the browser. We finish in part 3 by appending the closing format tags to the resulting page.

7.7.2 Application Servers

Application logic can be enforced through server-side programs that are invoked using the CGI protocol. However, since each page request results in the creation of a new process, this solution does not scale well to a large number of simultaneous requests. This performance problem led to the development of

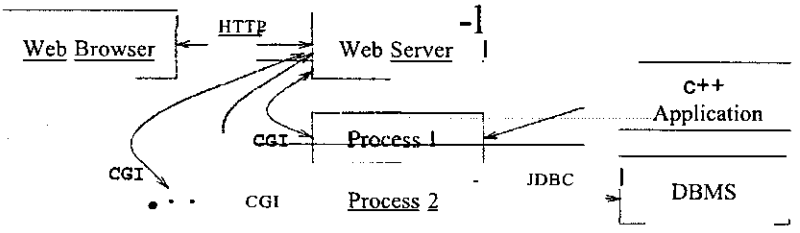


Figure 7.15 Process Structure with eGI Scripts

```
#!/usr/bin/perl
use CGI;

### part 1
$dataIn = new CGI;
$dataIn-¿header();
$authorName = $dataIn-I,param('authorName');

### part 2
print (II<HTML><TITLE>Argument passing test</TITLE> II) ;
print (II The user passed the following argument: II) ;
print (II authorName: ", $authorName);

### part 3
print ("</HTML>");
exit;
```

Figure 7.16 A Simple Perl Script

specialized programs called application servers. An application server maintains a pool of threads or processes and uses these to execute requests. Thus, it avoids the startup cost of creating a new process for each request.

Application servers have evolved into flexible middle-tier packages that provide many functions in addition to eliminating the process-creation overhead. They facilitate concurrent access to several heterogeneous data sources (e.g., by providing JDBC drivers), and provide session management services. Often, business processes involve several steps. Users expect the system to maintain continuity during such a multistep session. Several session identifiers such as cookies, URI extensions, and hidden fields in HTML forms can be used to identify a session. Application servers provide functionality to detect when a session starts and ends and keep track of the sessions of individual users. They

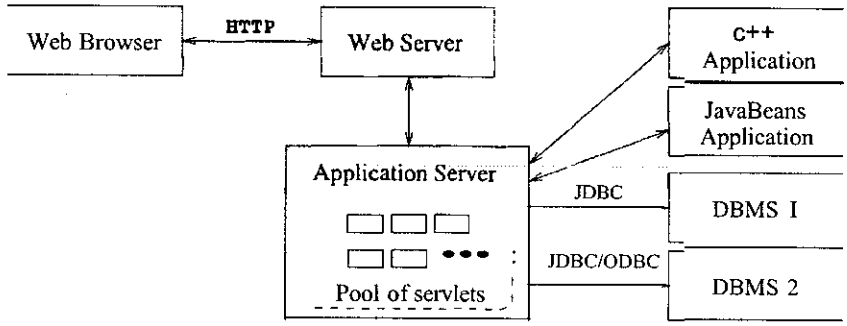


Figure 7.17 Process Structure in the Application Server Architecture

also help to ensure secure database access by supporting a general user-id mechanism. (For more on security, see Chapter 21.)

A possible architecture for a website with an application server is shown in Figure 7.17. The client (a Web browser) interacts with the webserver through the HTTP protocol. The webserver delivers static HTML or XML pages directly to the client. To assemble dynamic pages, the webserver sends a request to the application server. The application server contacts one or more data sources to retrieve necessary data or sends update requests to the data sources. After the interaction with the data sources is completed, the application server assembles the webpage and reports the result to the webserver, which retrieves the page and delivers it to the client.

The execution of business logic at the webserver's site, server-side processing, has become a standard model for implementing more complicated business processes on the Internet. There are many different technologies for server-side processing and we only mention a few in this section; the interested reader is referred to the bibliographic notes at the end of the chapter.

7.7.3 Servlets

Java servlets are pieces of Java code that run on the middle tier, in either webservers or application servers. There are special conventions on how to read the input from the user request and how to write output generated by the servlet. Servlets are truly platform-independent, and so they have become very popular with Web developers.

Since servlets are Java programs, they are very versatile. For example, servlets can build webpages, access databases, and maintain state. Servlets have access

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

pUblic class ServletTemplate extends HttpServlet {
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        // Use 'out' to send content to browser
        out.println("Hello World");
    }
}
```

Figure 7.18 Servlet Template

to all Java APIs, including JDBC. All servlets must implement the Servlet interface. In most cases, servlets extend the specific HttpServlet class for servers that communicate with clients via HTTP. The HttpServlet class provides methods such as doGet and doPost to receive arguments from HTML forms, and it sends its output back to the client via HTTP. Servlets that communicate through other protocols (such as ftp) need to extend the class GenericServlet.

Servlets are compiled Java classes executed and maintained by a servlet container. The servlet container manages the lifespan of individual servlets by creating and destroying them. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by webservers. For such applications, there is a useful library of HTTP-specific servlet classes.

Servlets usually handle requests from HTML forms and maintain state between the client and the server. We discuss how to maintain state in Section 7.7.5. A template of a generic servlet structure is shown in Figure 7.18. This simple servlet just outputs the two words "Hello World," but it shows the general structure of a full-fledged servlet. The request object is used to read HTML form data. The response object is used to specify the HTTP response status code and headers of the HTTP response. The object out is used to compose the content that is returned to the client.

Recall that HTTP sends back the status line, a header, a blank line, and then the context. Right now our servlet just returns plain text. We can extend our servlet by setting the content type to HTML, generating HTML as follows:

```
PrinfWriter out = response.get\Vriter();
String docType =
    "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 " +
    "Transitional//EN"> \n";
out.println(docType +
    "<HTML>\n" +
    "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
    "<BODY>\n" +
    "<H1>Hello WWW</H1>\n" +
    "</BODY></HTML>");
```

What happens during the life of a servlet? Several methods are called at different stages in the development of a servlet. When a requested page is a servlet, the webserver forwards the request to the servlet container, which creates an instance of the servlet if necessary. At servlet creation time, the servlet container calls the `init ()` method, and before deallocating the servlet, the servlet container calls the servlet's `destroyO` method.

When a servlet container calls a servlet because of a requested page, it starts with the `service()` method, whose default behavior is to call one of the following methods based on the HTTP transfer method: `service()` calls `doGet0` for a HTTP GET request, and it calls `doPost()` for a HTTP POST request. This automatic dispatching allows the servlet to perform different tasks on the request data depending on the HTTP transfer method. Usually, we do not override the `service()` method, unless we want to program a servlet that handles both HTTP POST and HTTP GET requests identically.

We conclude our discussion of servlets with an example, shown in Figure 7.19, that illustrates how to pass arguments from an HTML form to a servlet.

7.7.4 JavaServer Pages

In the previous section, we saw how to use Java programs in the middle tier to encode application logic and dynamically generate webpages. If we needed to generate HTML output, we wrote it to the out object. Thus, we can think about servlets as Java code embodying application logic, with embedded HTML for output.

JavaServer pages (JSPs) interchange the roles of output and application logic. JavaServer pages are written in HTML with servlet-like code embedded in special HTlVIL tags. Thus, in comparison to servlets, JavaServer pages are better suited to quickly building interfaces that have some logic inside, whereas servlets are better suited for complex application logic.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class ReadUserName extends HttpServlet {
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType('j textjhtml'j);
        PrintWriter out = response.getWriter();

        out.println("<BODY>\n" +
            "<Hi ALIGN=CENTER> Username: </Hi>\n" +
            "<UL>\n" +
            " <LI>title: "
            + request.getParameter("userid") + "\n" +
            + request.getParameter("password'j) + "\n" +
            1</UL>\n" +
            1</BODY></HTML>")j
    }

    public void doPost(HttpServletRequest request,
            HttpServletResponse response)
        throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Figure 7.19 Extracting the User Name and Password From a Form

5 Sem CS DBMS

KARANAM SUNIL KUMAR
Asst. Prof. Dept of CSE
RNSIT, Blore
ph: 9972995720
Email: SunilKaranam72@gmail.com

MODULE - 4

Normalization: Database Design Theory

1. Introduction to Normalization using functional and Multi-valued dependencies.
2. Informal design guidelines for relation schema.
3. Functional Dependencies.
4. Normal Forms based on Primary keys.
5. Second and Third Normal forms
6. Boyce-Codd Normal Form
7. Multi-valued Dependency and Fourth Normal Form
8. Join Dependencies and Fifth Normal Form

Normalization Algorithms:

1. Inference Rules
2. Equivalence and Minimal Cover
3. Properties of relational Decompositions
4. Algorithms for Relational Database Schema Design,
5. Nulls and Dangling tuples and alternate Relational design
6. Further discussion of Multi-valued dependencies and 4NF
7. Other Dependencies and Normal Forms.

Introduction:

Normalization is a process of organizing a data to avoid data Redundancy and Anomalies

Design

Informal Guidelines for Relational Databases :

The one four informal measures of quality for relation schema design

1. Semantics of the relation attributes (Making Sure that the meaning of the attributes is clear in the relational schema.
2. Redundant Information in Tuples and update Anomalies (Reducing the redundant information that exists in a relation in the form of tuples.
3. Reducing the NULL values in tuples (Reducing the NULL values in tuples.
4. Spurious tuples (Disallowing the possibility of generating spurious tuples.

Semantics of the Relation Attribute :

Guidline1 : Design a relation schema so that it is easy to explain its meaning. Attribute from multiple entity types and relationship types must not be combined into a single relation.

→ Hence design a relation schema such that it has one entity type ~~and~~ or one relationship type.

Example Emp_Department Relation.

EMP-DEPT	ENO	NAME	SAL	DNO	DNAME
	128	Adarsh	15000	1	CS
	124	Chetan	20000	2	IS
	125	Bharath	25000	1	CS

Reducing the redundant Information in tuples:

Major objective of Schema design is to minimize the Storage Space that the base relation occupies and there by efficiency can improved.

→ Combining multiple entities into a single relation will create lot of redundancy and will have has a significant-effect on Storage Space.

→ For example letus combine below given relations.

FK PK

EMP	EID	ENAME	EDNO
	111	Govind	1
	112	Adarsh	1
	113	Bharath	2

DEPT	DNO	DNAME
	1	CS
	2	IS

EMP-DEPT	EID	ENAME	DNO	DNAME
	111	Govind	1	CS
	112	Adarsh	1	IS
	113	Bharath	2	IS

→ Space used by the relations EMP and DEPT is less Compared to Space used by "EMP-DEPT.)

→ Another Serious problem with using relation EMP-DEPT as base relation is the problem of update anomalies.

→ These can be classified into

1. Insertion Anomalies 2. Deletion Anomalies

3 Modification Anomalies.

Insertion Anomalies :

The problems that are associated with Insertion and referred as Insertion anomalies. There are two types of insertion anomalies.

- To insert a new tuple employee without department details into EMP-DEPT, we must include the attribute values of the department as NULL.

→ Example : To insert a new tuple for an employee <114, Chetan> who does not works in any department.

Design 1

PK

EMP	EID	ENAME	EDNO
	111	Govind	1
	112	Adarsh	1
	113	Bharath	2
	114	Chetan	NULL

FK

DEPT	DNO	DNAME
	1	CS
	2	IS

Design 2

EMP_DEPT	EID	ENAME	DNO	DNAME
	111	Govind	1	CS
	112	Adarsh	1	CS
	113	Bharath	2	IS
	114	Chetan	NULL	NULL

→ In design 2 there are more number of NULL as compared to design 1. More number of NULL means, there is more amount of memory wasted. Design 1 is more efficient as the number of NULL values are less and less wastage of memory.

ii. It is difficult to insert a new department that has no employee details in relation EMP_DEPT.

ex: let us insert <NULL, NULL, 3, IT> into a EMP-DEPT.

This representation causes a problem because ENO is Primary Key, inserting NULL values violates Entity Integrity constraint. Insertion will be rejected.

FK → PK

Design 1

EMP	EID	ENAME	EDNO
	111	Govind	1
	112	Adarsh	1
	113	Bharath	2

DEPT	DNO	DNAME
	1	CS
	2	IS
	3	IT

Insertion is not possible.

Design 2

EMP-DEPT	EID	ENAME	DNO	DNAME
	111	Govind	1	CS
	112	Adarsh	1	CS
	113	Bharath	2	IS
	NULL	NULL	3	IT

Insertion is not possible because Primary Key value cannot be NULL.

Deletion Anomalies

The problems that are associated with deletion are referred as deletion anomalies.

→ Deletion Anomalies is explained below with an example.

Assume: user interested to delete row 3

Design 1

EMP	EID	ENAME	EDNO
	111	Govind	1
	112	Adarsh	1
	113	Bharath	2

key to delete row

Dept	DNO	DNAME
	1	CS
	2	IS

Deletion is not possible

Design 2

EMP-DEPT	EID	ENAME	DNO	DNAME
	111	Govind	1	CS
	112	Adarsh	1	CS
	113	Bharath	2	IS

Deletion is possible But department 2 information will lose.

Modification Anomalies :

The problems associated with modification are referred as Modification Anomalies.

→ The Modification anomaly is explained below with update operation.

→ Consider the relation given below.

EMP	EID	ENAME	EDNO
	111	Govind	1
	112	Adarsh	1
	113	Bharath	2

DEPT	DNO	DNAME
	1	CS & E
	2	IS

Updation is on one tuple value CS to CS & E

EMP DEPT	EID	ENAME	DNO	DNAME
	111	Govind	1	CS & E
Design 2	112	Adarsh	1	CS & E
	113	Bharath	2	IS

Updation is done on multiple tuple values.

NULL values in Tuples.

In some database designs due to the requirement we may group many attributes togeather into a single relation and this relation is referred as "FAT" relation.

→ When a tuple values are inserted, if many of the attribute values do not apply to that tuple, then the value is substituted with NULL values.

Example for NULL values in tuples is given below.

EMPDEPT	EID	ENAME	EDNO	ELICENCENO	EPASSPORTNO	EPANNO
	111	Govind	1	NULL	ERY3545TY	34545HGF
	112	Adarsh	1	GFH445	NULL	NULL
	113	Bharath	2	NULL	NULL	NULL

Disadvantages of NULL values in Tuple :

1. This can waste space at the storage level since for NULL value, memory is allocated.
2. Meaning of NULL is confusing since NULL can be used to indicate.
   a) Value not present b) Not applicable c) Currently unknown and soon.
3. Specifying JOIN operation on the tuples having attribute value as NULL, will remove these tuples from the resulting relation since joins are applied on equality of foreign key equal to primary key.

EMP	EID	ENAME	EDNO
	111	Govind	1
	112	Adarsh	NULL
	113	Bharath	NULL

DEPT	DNO	DNAME
	1	CS
	2	IS

$EMP \bowtie_{EDNO = DNO} DEPT$

EID	ENAME	EDNO	DNO	DNAME
111	Govind	1	1	CS

→ Another problem with NULL values is associated with aggregate functions. Examples are given below.

$\mathcal{J}$ COUNT ELICENCE NO (EMPDEPT)

ELICENCE NO
NULL
GFH 445
NULL

Generation of Spurious tuples:

"Generation of Spurious tuples" happens due to improper division of tables.

→ Consider the relation given below.

Proj_Emp PNO	PNAME	PLOC	ENO	ENAME
P21	ISRO	BANGLORE	111	Anand
P22	IIMB	Mysore	112	KUMAR
P23	IISE	BANGLORE	113	MADAN

Since proj-Emp multiple relations are combined to form a single relation, in order to improve the efficiency division is applied. Let us assume the division is made horizontally and resultant of division is as follows.

Project PNO	PNAME	PLOC
P21	ISRO	BANLORE
P22	IIMB	Mysore
P23	IISE	BANGLORE

EMP PLOC	ENO	ENAME
BANGLORE	111	Anand
Mysore	112	Kumar
BANGLORE	113	MADAN

Division must be made such that when Natural JOIN is applied on ~~theb rel~~ divided relations, Orginal relation must be retsieved back.

Natural join on EMP * PROJECT.

PNO	PNAME	PLOC	PLOC	ENO	ENAME
P21	ISRO	BNG	BNG	111	~~ANAND~~
P21	ISRO	BNG	BNG	113	MADAN
P22	IIMB	MYS	MYS	112	KUMAR
P24	IISE	BNG	BNG	113	MADAN
P21	IISE	BNG	BNG	111	~~ANAND~~

↓

Spurious tuples.

→ To avoid Supurious tuple division operation must be based on Primary key and Foreign key Concept.

F ↓ K

Project	PNO	PNAME	PLOC	ENO
	P21	ISRO	BNG	111
	P22	IIMB	MYS	112
	P23	IISE	BNG	113

P ↓ K

EMP	ENO	ENAME
	111	~~ANAND~~
	112	KUMAR
	113	MADAN

If you apply a natural join on above relations (divided using Primary and Foreign key Concept), orginal relations ~~gd~~ retrieved.

Guideline: Relations can be joined with equality condition on attributes that are either primary key or foreign key. This join guarantees that "no Spurious tuples are generated".

Functional Dependencies:

A functional dependency is a constraint between two sets of attributes with in a Relation.

→ A functional depeandency denoted by $X \rightarrow Y$, between two sets of attributes X & Y that are subset 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 & t_2 in r that have $t_1[X] = t_2[X]$ we must also have $t_1[Y] = t_2[Y]$

relation instances.

→ The value of X component of a tuple uniquely determine the value of the Y component

$$X \rightarrow Y$$

→ FD's are derived from the real-world constraints on the attributes.

Examples of FD constraints.

1 Social Security number determines employee name

SSN → ENAME

2. Project-number determines project name and location

PNUMBER → { PNAME, PLOCATION }.

3. Employee SSN and Project numbes determines the hours per week that the employee works on the project

{ SSN, PNUMBER } → HOURS.

(a) EMP_DEPT

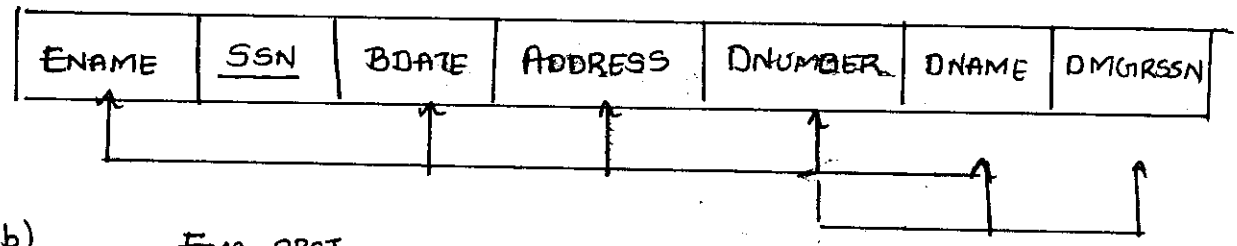


fig (a) EMP-DEPT relation Schema

(b) EMP-PROJ

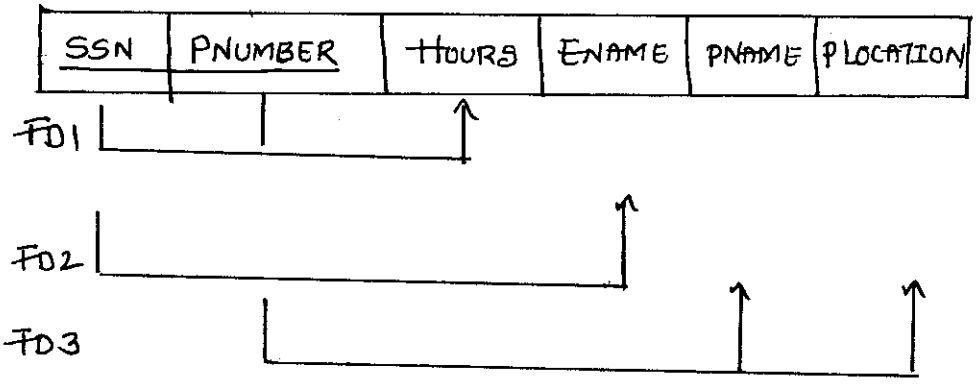


fig (b). EMP-proj relation Schema

Inference Rules for FDs :

The following six rules are well known as Inference rule.

IR1 (reflexive rule): If $x \supseteq y$, then $x \to y$

IR2 (Augmention rule): If $x \to y$ then $xz \to yz$

IR3 (Transitive) If $x \to y$ and $y \to z$ then $x \to z$

} Armstrong's inference rule.

IR4 (decomposition, or projection rule):

If $x \to yz$ then $x \to y$ and $x \to z$

IR5 (union) If $x \to y$ and $x \to z$ then $x \to yz$

IR6 (Pseudotransitivity): If $x \to y$ and $wy \to z$ and $wx \to z$

Note: The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2 and IR3 (Completeness Property)

Closure Set :

The set of all Functional dependencies is called Closure Set, denoted by F^+ — (F maybe attribute of a relation.)

→ Consider the below given relation schema.

— ex: R(A B C)

Dependencies

$A \rightarrow B$

$B \rightarrow C$ ↑ Not directly visible

Set of all functional dependencies → $F = F1 + F2$

↓

Directly visible.

$A \rightarrow B$, $B \rightarrow C$ } Directly visible

$A \rightarrow C$ } Directly Not visible

Closure Set of Attributes

$A^+ = \{ A\ B\ C \}$

$B^+ = \{ B\ C \}$

$C^+ = \{ C \}$

Find the closure set of attributes for the given set of attributes in a relation schema.

i. R(A B C D E F G)

Functional Dependencies

1 $A \rightarrow B$

2 $BC \rightarrow DE$

3 $AEG \rightarrow G$

$(AC)^+ = AC$

$= ABC$

$= ABCDE$

Problem 2 : R(A B C D E)

Function dependencies

$A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

find the Closure set of B^+

Solution: $(B)^+ = B$

$= BD$

Closure set can be determined by using inference rules IR1 through IR3 are known as ARMSTRONG Inference rules.

Algorithm: For determining x^+ closure of x under X

Algorithm:

$x^+ = x$

repeat

old $x^+ := x^+$;

for each functional dependency $(y \rightarrow z)$ in F do

if $x^+ \supseteq y$ then $x^+ := x^+ \cup z$;

until $(x^+ = \text{old } x^+)$;

Algorithm stated above starts by setting x^+ to all the attributes in X. By IR1, we know that all these attributes are functionally dependent on X. using Inference rules IR3 and IR4, we add attributes to x^+, using each functional dependency in F.

Example to find closure set-

$F = \{ SSN \rightarrow ENAME,\ PNO \rightarrow \{PNO, PNAME, PLOCATION\}$

$\{SSN, PNO\} \rightarrow HOURS \}$

$\{SSN\}^+ = \{ SSN, ENAME \}$

$\{PNO\}^+ = \{ PNO, PNAME, PLOCATION \}$

$\{SSN, PNO\}^+ = \{ SSN, PNUMBER, ENAME, PLOCATION, PNAME, HOURS \}$

Equivalence of Sets of Functional Dependencies :

Definition 1: A set of functional dependencies F is said to cover another set of E (G used in class Notes) if every FD in E is also in F^+; ie if every dependency in E can be inferred from F; alternatively, we can say that E is covered by F.

Definition 2 : " Two sets of functional dependencies E (or G) and F are equivalent if $E^+ = F^+$. Therefore, equivalence means that every FD in F can inferred (derived) from E; ie, E is equivalent to F if both the conditions — E covers F and F covers E.

Ex: Method 1. to check Equivalence of functional dependencies

Consider a relation with attributes like

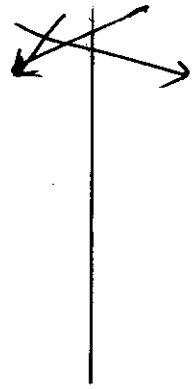
R(A C D E H)

Functional dependencies F

F :- $A \rightarrow C$
$AC \rightarrow D$
$E \rightarrow AD$
$E \rightarrow H$

Function dependences G (or E' according to definition)

G (or E) : $A \rightarrow CD$
$E \rightarrow AH$



$(A)^+ \rightarrow ACD$

$(Ac)^+ \rightarrow ACD$

$(E^+) \rightarrow ACDEH$

$A^+ = ACD$

$(E)^+ \rightarrow ACDEH$

Both functional dependencies are equal

Minimal Set of Functional Dependencies :

Informally, a minimal cover of a set of functional dependencies E is a set of functional dependencies F that satisfies the property that every dependency is in the closure F^+ of F.

Property 1 → In addition, this property is lost if any dependency from the set F is removed.

Property 2 → F must have no redundancies in it, and the dependencies in F are in a standard form.

To satisfy these properties, we can formally define a set of functional dependencies F to be minimal if it satisfies the following conditions

1. Every dependency in F has a single attribute for its right hand side.
2. We cannot replace any dependency $X \rightarrow A$ in F with a dependency $Y \rightarrow A$, where Y is proper subset of X, and still have a set of dependencies that is equivalent to F.
3. We cannot remove any dependency from F and still have a set of dependencies ie equivalent to F.

Definition of Minimal Cover : A minimal cover of a set of functional dependencies E is a minimal set of dependencies ie equivalent to E. We can always find at least one minimal cover F for any set of dependencies E using below given algorithm.

Algorithm To Find Minimal Cover :

Algorithm : Finding a Minimal Cover F for a Set of Functional Dependencies E

Input : A Set of functional dependencies E

1. Set F := E
2. Replace each functional dependency $X \rightarrow \{A_1, A_2 \cdots A_n\}$ in F by the n functional dependencies

$$X \rightarrow A_1,\ X \rightarrow A_2 \cdots\cdots X \rightarrow A_n$$

3. For each functional dependency $X \rightarrow A$ in F
   for each attribute B ie an element of X
   if $\{\{F - \{X \rightarrow A\}\} \cup \{\{X - \{B\}\} \rightarrow A\}\}$ ie 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

Example :

R = { Sno, Sname, Cno, Cname, instructor, address, office }

F = { Sno → Sname
Cno → Cname
Sno → address
Cno → Instructor
Instructor → office }

$\{Sno, cno\}^+ = Sno, cno$
$= Sno, cno, Sname$ (Sno → Sname)
$= Sno, cno, Sname, cname$ (Cno → Cname)
$= Sno, cno, Sname, cname, address$ (Sno → address)

= Sno, Cno, Sname, Cname, address, instructor (Cno → instructor)

{Sno, Cno}$^+$ = { Sno, Cno, Sname, Cname, address, instructor, office }

↳ Minimal Cover, and it is a Candidate Key.

Example 2 : R = { A, B, C, D, E }

F = { A → BC, CD → E, B → D, E → A }

Compute A$^+$ and B$^+$

A$^+$: A
AB C
A B C D
A B C D E

B$^+$: B
BD

Example 3 : Consider the universal relation R = { A, B, C, D, E, F, G, H, I, J } and set of functional dependencies.

F = { AB → C, A → DE, B → F, F → GH, D → IJ } Find the following.

1) Key of R :

AB$^+$ → AB
ABC
ABCDE
ABCDEF
ABCDEFGH

AB$^+$ = { ABCDEFGHIJ }

B$^+$ B
BF

B$^+$ = { BFGH }

F$^+$ = { FGH }

D$^+$ = { DIJ }

∴ AB is the Key (or minimal cover)

∴ It identify all the attributes

Introduction to Normalization :

Normalization is a systematic approach of decomposing relations to eliminate redundant data and undesirable characteristics like Insertion, Update and Deletion Anamolies.

→ There are various Normal Forms proposed by Dr. codd to represent relations.

→ Normal Form : Normal Form of a relation refers to the highest Normal form condition that it meets, and hence indicates the degree to which it has been normalized.

→ Basically there are 3 Normal Forms

1. 1NF
2. 2NF
3. 3NF

and

4. BCNF are based on Keys and FDS of a relation Schema.

4. 4NF is based on Keys and multivalued dependencies.

5. 5NF is based on Keys and Join Dependencies

Things to Remember :

1. A Super Key of a relation Schema $R = \{A1, A2, \ldots An\}$ is a Set of attributes S Subset of R with the property that no two tuples $t1$ and $t2$ in any legal relation State r of R will have $t1[S] = t2[S]$
2. A Key K is a Super Key with the additional property that removal of any attribute from K will cause K not to be a Superkey any more.
3. 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 others are called Secondary Keys

Introduction to Normalization :

Normalization is a systematic approch of decomposing relations to eliminate redundant data and undesirable characteristics like Insertion, Update and Deletion Anamolies.

→ There are various Normal Forms proposed by Dr. codd to represent relations.

→ Normal Form : Normal Form of a relation refers to the highest Normal form condition that it meets, and hence indicates the degree to which it has been normalized.

→ Basically there are 3 Normal Forms

1. 1NF
2. 2NF
3. 3NF

and

4. BCNF are based on Keys and FDS of a relation Schema.

4. 4NF is based on Keys and multivalued dependencies.

5. 5NF is based on Keys and Join Dependencies

Things to Remember :

1. A Super Key of a relation Schema $R = \{A1, A2, \ldots An\}$ is a Set of attributes S Subset of R with the property that no two tuples t1 and t2 in any legal relation state r of R will have $t1[S] = t2[S]$
2. A Key K is a Super Key with the additional property that removal of any attribute from K will cause K not to be a Superkey any more.
3. 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 others are called Secondary Keys.

→ A Primary Attribute must be a member of some Candidate key.

→ A Nonprime attribute → Attribute is not a member of any Candidate key.

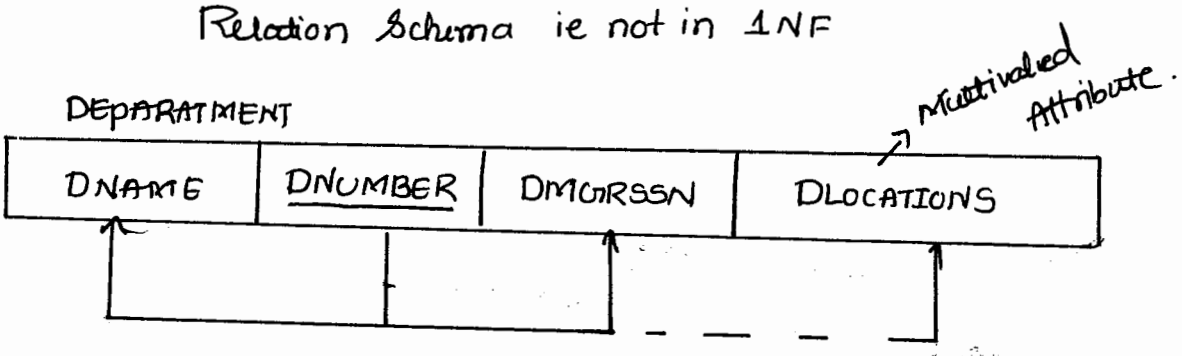
First Normal Form (1NF)

Definition: A relation schema R is said to be in First Normal Form (1NF) if every attribute is atomic.

→ Every cell in the table must contain atomic value.

Example 1:

Relation Schema ie not in 1NF



Example for Relation Instance

DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

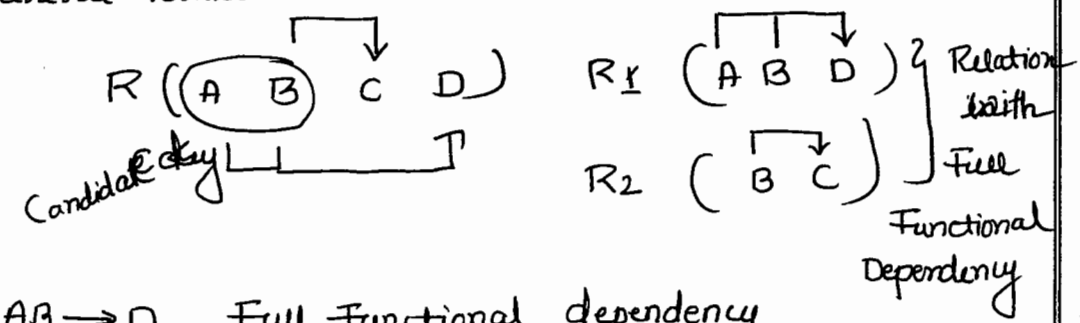
After Converting above one into 1NF: DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATION
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Second Normal Form: 2NF is based on the concept of full functional dependency.

→ A relation schema R is in second normal form (2NF) if every non-prime Attribute A in R is fully functionally dependent on the primary Key (candidate key)

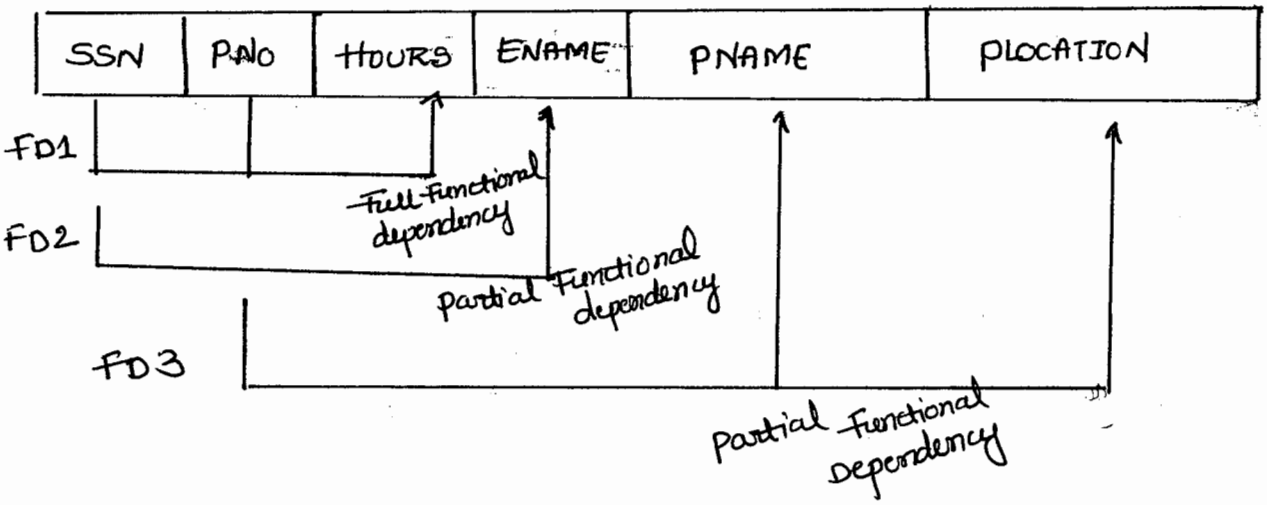
→ General format:



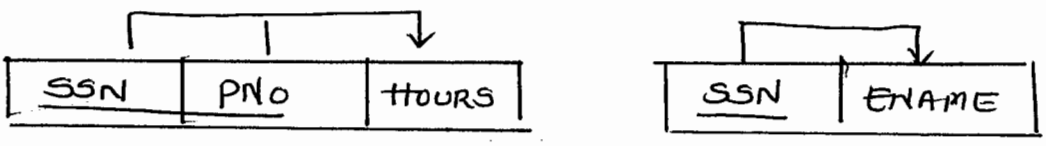
AB → D Full Functional dependency

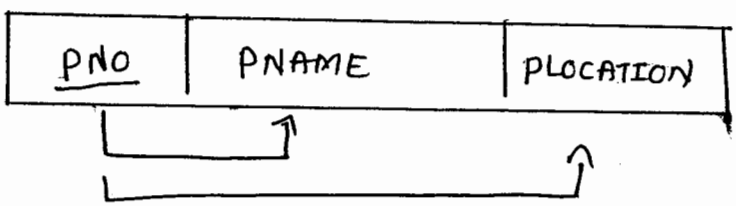
B → C Partial Functional dependency

→ Example : Emp_Proj



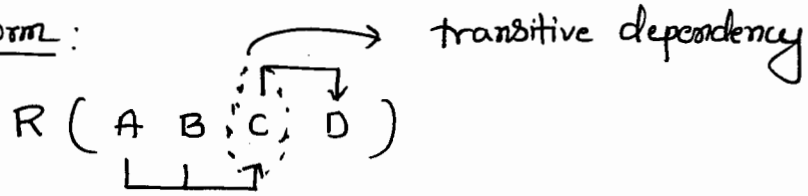
→ All partial dependency of Emp_Proj are indicated as Separate tables.



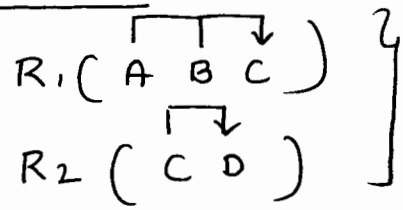


Third Normal Form (3NF) : A Relation schema R is in 3NF if it is in 2NF and no non-prime attribute A in R is transitively dependent on the primary key.

General Form :



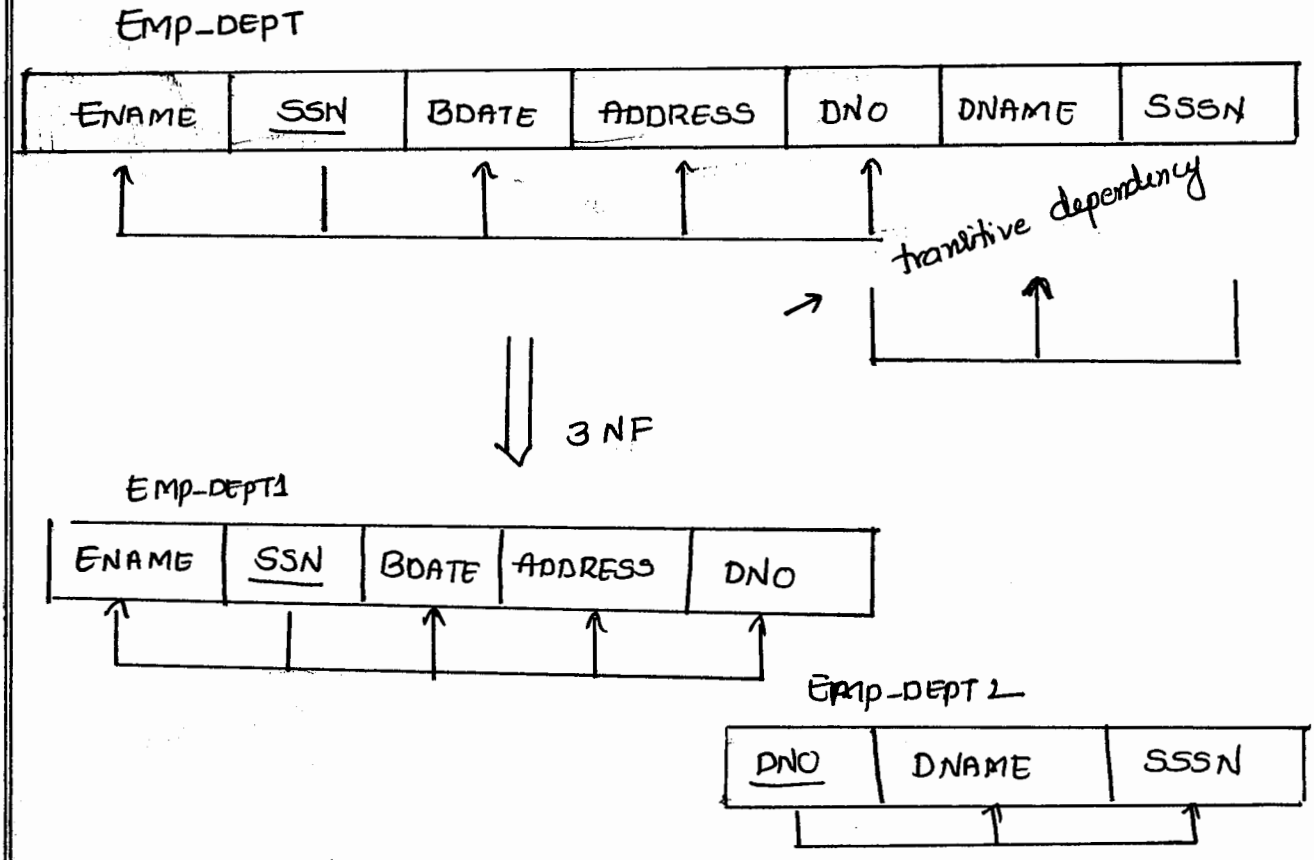
Convert it to 2NF:



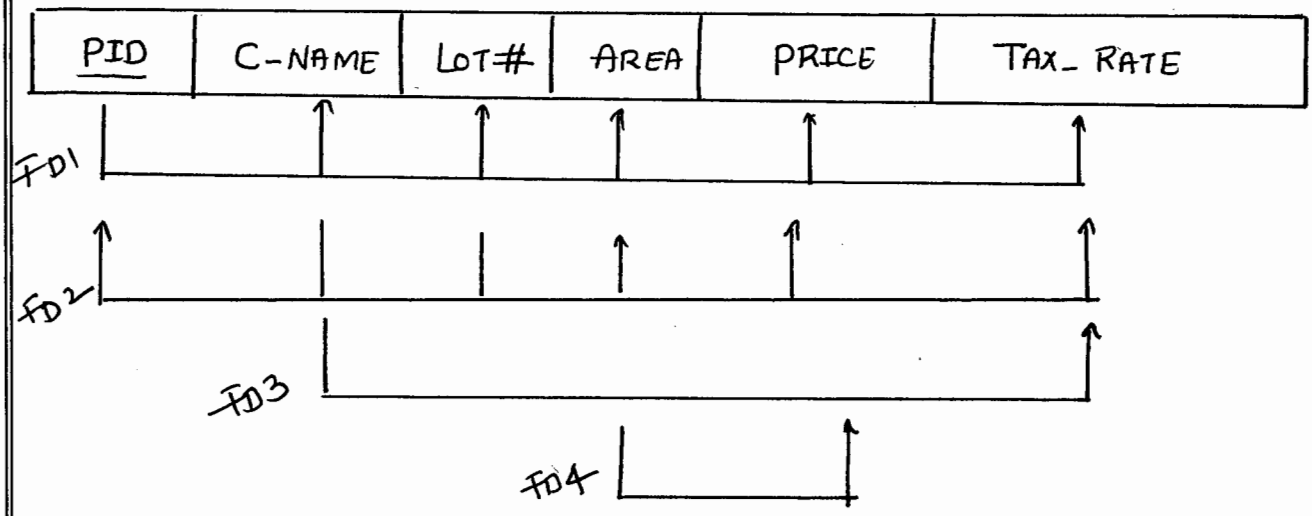
Need to make 'C' as prime attribute (C should have unique and Not NULL values)

If 'C' satisfies unique and NOT NULL property we call above decomposition in 3NF.

Example 1: Consider the relation EMP_DEPT

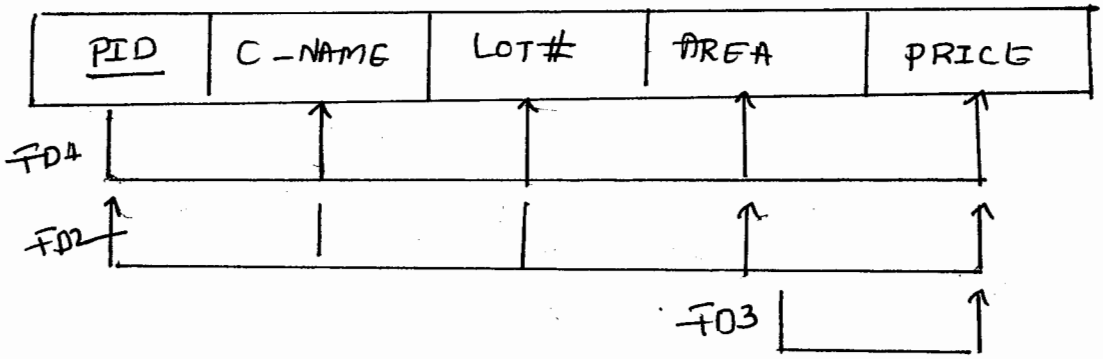


Example 2 : LOTS

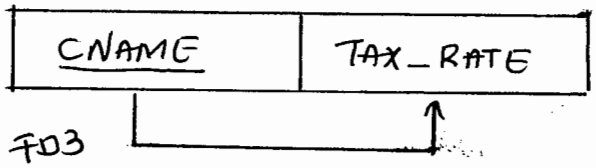


Keybase { PID } { CNAME, LOT# }

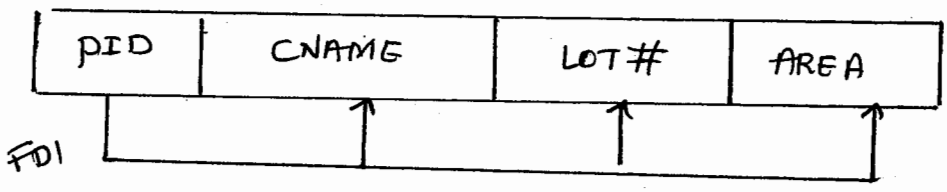
2NF : LOTS1



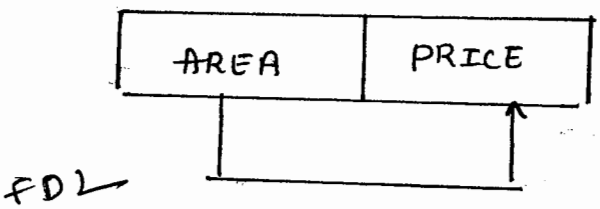
LOTS2



3NF LOTS1A



LOTS 1B



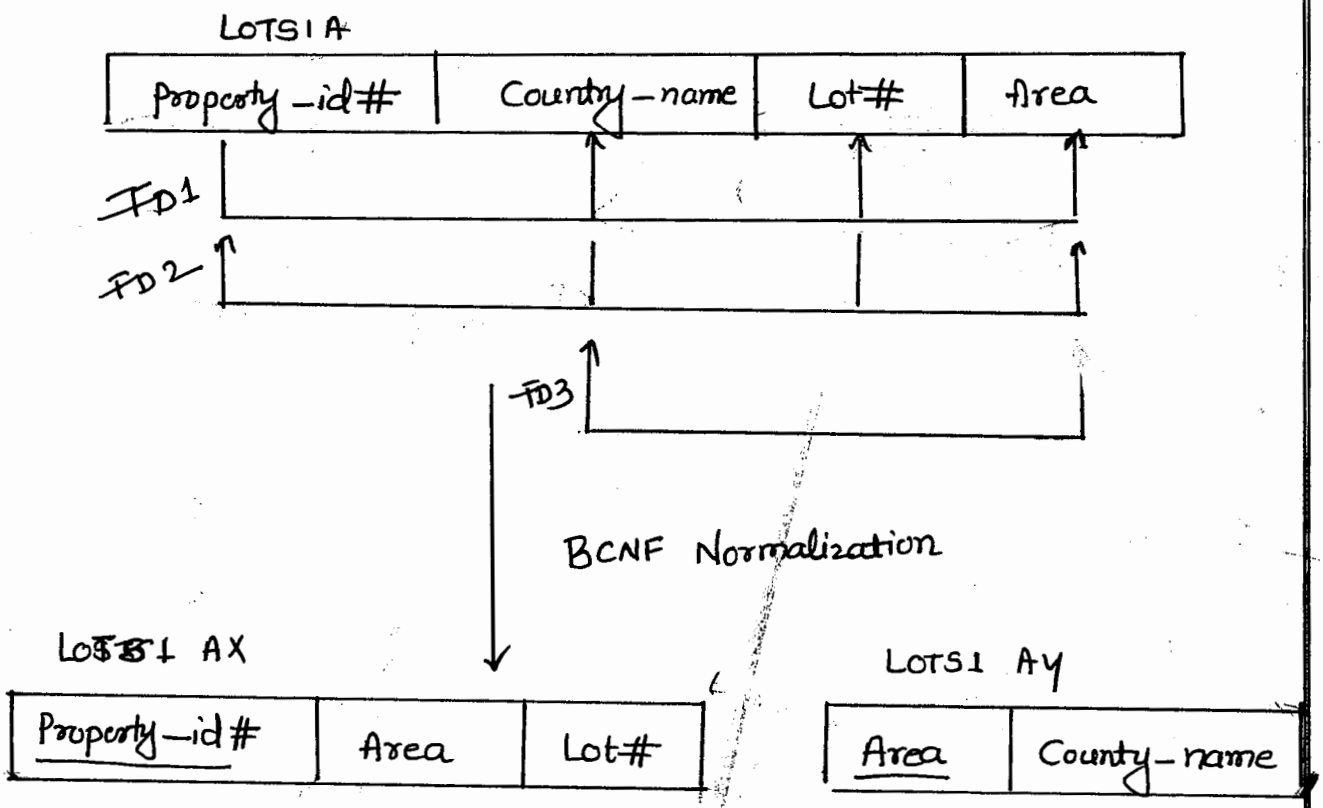
Boyce – Codd Normal Form : (BCNF)

"A Relation Schema R is in BCNF if whenever an Functional dependency $X \rightarrow A$ holds in R, then X is a Superkey of R

→ Each normal form is strictly ~~Strong~~ Stronger than the Previous one.

→ BCNF strictly stronger than 3NF, 2NF and 1NF.

Example :



Relation Decomposition and Inefficiency of Normal Forms :

Relation Decomposition : Before we move to Relation Decomposition it is necessary to understand "Universal Relation Schema Concept.

→ Universal relational schema $-R = \{A1, A2, A3 \ldots An\}$ that includes all the attributes of the database.

→ The process of decomposing the universal relation schema R into a set of relation schemas ~~into a set~~

$D = \{R_1, R_2, \ldots R_m\}$ that will become the relational database schema by using the functional dependencies.

→ This above process is also called as Decomposition of R

→ Each attribute of R must appear in atleast one relation $\bigcup_{i=1}^{m} R_i = R$ where i — denotes attribute

→ This is called the attribute preservation condition of a decomposition.

Decomposition and Dependency Preservation :

Each Functional dependency $X \rightarrow Y$ specified in F either appeared directly in one of the relational schema R_i in the decomposition D or could be inferred from the dependencies that appear in some R_i. Informally this is the dependency Preservation condition.

→ It is necessary to preserve the dependencies because each dependency in F represents a constraint on the database.

Definition : Given a set of dependencies F on R, the Projection of F on R_i, denoted by $\pi_{R_i}(F)$ where R_i is a subset of R, is the set of dependencies $X \rightarrow Y$ in F^+ such that the attributes in $X \cup Y$ are all contained in R_i

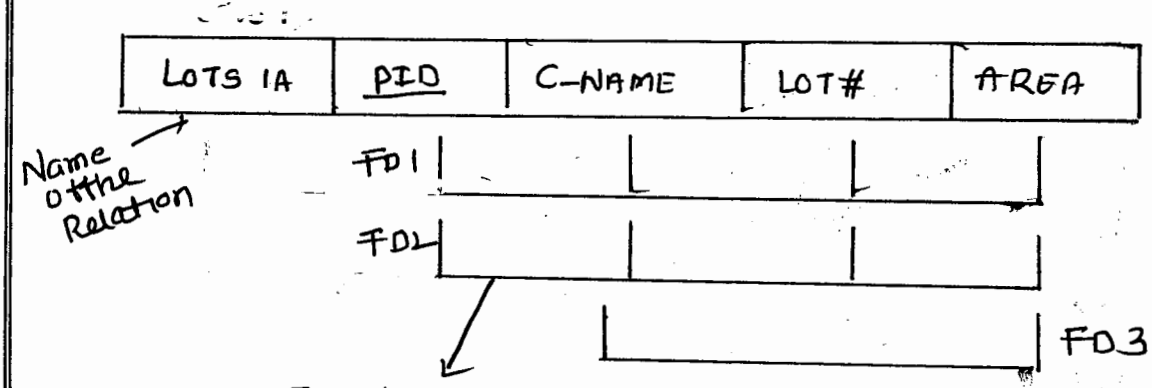
→ Hence the projection of F on each relation schema R_i in the decomposition D is the set of functional dependencies in F^+, the closure of F, such that all their left and right hand-side attributes in R_i.

→ Decomposition $D = \{R_1, R_2 \ldots R_m\}$ of R is dependency Preserving with respect to F if the union of the projections

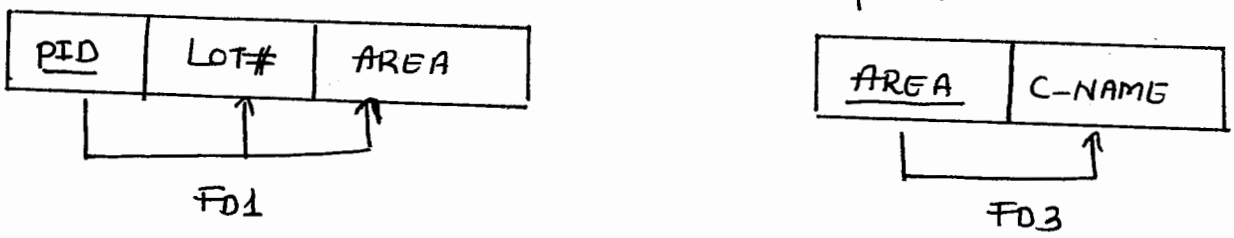
of F on each R_i in D is equivalent to F i.e.

$$((\pi_{R_1}(F)) \cup \cdots \cup (\pi_{R_M}(F)))^+ = F^+$$

Example : Decomposition that doesnot preserve dependencies

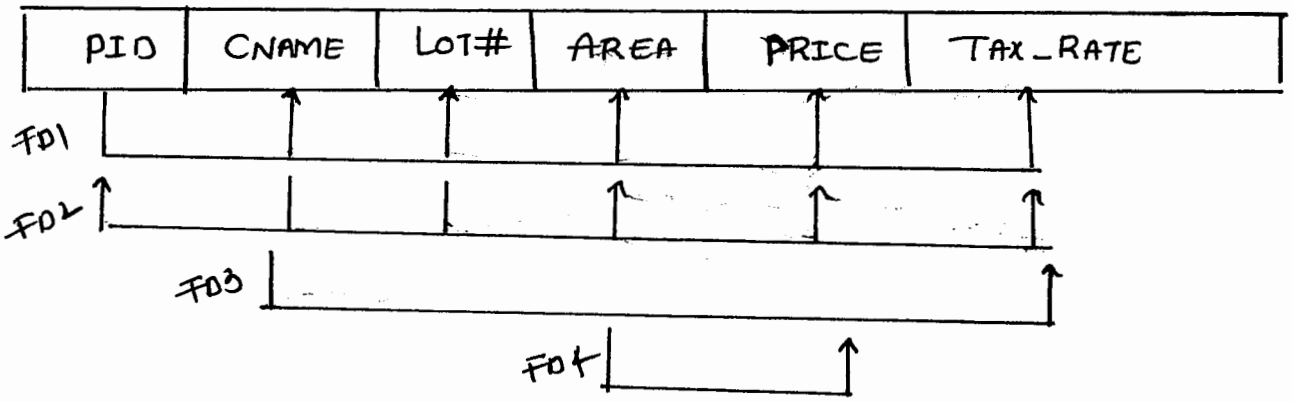


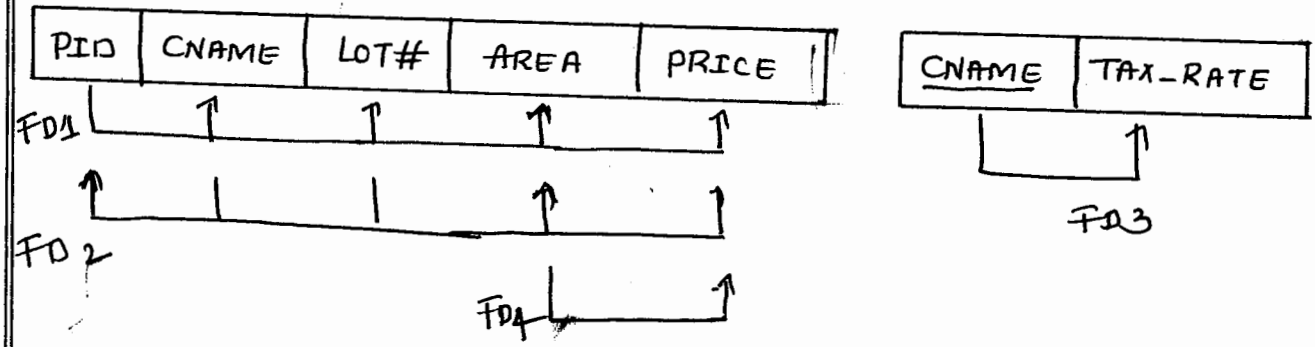
FD2 is lost in the below Decomposition



Example : Decomposition that preserves Decomposition.

LOTS





Decomposition and Lossless (Non-additive) Joins

Non-additive or lossless join ~~join~~ decomposition $D = \{R1, R2 \ldots, Rm\}$ of R ~~has~~ with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds, where * is the Natural join of all the relation in D

$$*(\pi_{R1}(r), \ldots \pi_{Rm}(r)) = R$$

Note : The word loss in lossless refers to loss of information not to loss of tuples. In fact for "loss of information" a better term is "Addition of Spurious tuples or information"

Algorithm : Testing for Non-Additive join property

Input : A universal relation R, a decomposition $D = \{R1, R2 \ldots Rm\}$ of R, and a set F of functional dependencies.

Note : Explanatory Comments are given at the end of some of the steps. The comment format is * Comment *

1. Create an Initial matrix S with one row i for each relation Ri in D and one column j for each attribute Aj in R.
2. Set S(i,j) := bij for all matrix entries (* each bij is a distinct symbol associated with indices (i,j) *)
3. For each row i representing relation schema Ri
   { for each column j representing attribute Aj
   { if (relation Ri includes attribute Aj) then Set S(i,j) := aj ; } (* each aj is a distinct symbol associated with index (j) *)

4. Repeat the following loop until a complete loop execution results in no changes to S.

{ for each functional dependency $X \rightarrow Y$ in F

{ for all rows in S that have the same symbols in the columns corresponding to attributes in X.

{ make the symbol in each column that correspond to an attribute in Y be the same in all these rows as follows: If any of the rows has an a symbol for the column, set other rows to that same a symbol in the column. If no a symbol exists for the attribute in any of the rows, choose one of the b symbols that appears in one of the rows for the attribute and set the other rows to that same b symbol in the column. }}}

5. If a row is made up entirely of a symbols, then the decomposition has the nonadditive join property; otherwise, it does not.

Example: Consider $R = (A1, A2, A3, A4, A5)$ and a set of

$$FD = \{A1 \rightarrow A3A5,\ A5 \rightarrow A1A4,\ A3A4 \rightarrow A2\}$$

Determine the decomposition

$$D = \{R1, R2, R3\}$$

$R1\{A1, A2, A3, A5\}$ $R2\{A1, A3, A4\}$ $R3(A4, A5)$ is

loss less join decomposition.

Step1 : Creation of Matrix

In given problem a relation is divided into 3 relations and hence represent 3 Separate rows, and 5 attributes.

R	A_1	A_2	A_3	A_4	A_5
R_1					
R_2					
R_3					

Step2 : Mention bij i - represent the row number j - represents the Column number.

R	A_1	A_2	A_3	A_4	A_5
R_1	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
R_2	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}
R_3	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}

Step 3.1 Mention a_i value i - represents the Column number.

In ~~first row~~ R_1 (A_1, A_2, A_3 A_5) ~~rest~~ attributes, those column must be filled by. a column_no in row 1 as per algorithm.

R	A_1	A_2	A_3	A_4	A_5
R_1	a_1	a_2	a_3	b_{14}	a_5
R_2	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}
R_3	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}

Step 3.2 In relation $R_2(A_1, A_3, A_4)$ attributes add a_{col_no} in the 2nd row for respetive attributes.

R	A_1	A_2	A_3	A_4	A_5
R1	a_1	a_2	a_3	b_{14}	a_5
R2	a_1	b_{22}	a_3	a_4	b_{25}
R3	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}

Step 3.3 In relation $R_3(A_4, A_5)$ attributes add a_{col_no} in the 3rd row for respetive attributes.

R	A_1	A_2	A_3	A_4	A_5
R1	a_1	a_2	a_3	b_{14}	a_5
R2	a_1	b_{22}	a_3	a_4	b_{25}
R3	b_{31}	b_{32}	b_{33}	a_4	a_5

Step4 : Compare each FD

$A_1 \rightarrow A_3 A_5$

R	A1	A2	A3	A4	A5
R1	a_1	a_2	a_3	b_{14}	a_5
R2	a_1	b_{22}	a_3	a_4	b_{25}
R3	b_{31}	b_{32}	b_{33}	a_4	a_5

this can be changed to a_5

here we cannot change.

Steps: applying changes

R	A1	A2	A3	A4	A5	
R1	a_1	a_2	a_3	b_{14}	a_5	
R2	a_1	b_{22}	a_3	a_4	a_5	
R3	b_{31}	b_{32}	b_{33}	a_4	a_5	

FD $A_5 \rightarrow A_1 A_4$

Changed a_4

	A1	A2	A3	A4	A5
R1	a_1	a_2	a_3	b_{14}	a_5
R2	a_1	b_{22}	a_3	a_4	a_5
R3	b_{31}	b_{32}	b_{33}	a_4	a_5

Changed to a_1

Steps: Applying changes

R	A1	A2	A3	A4	A5
R1	a_1	a_2	a_3	a_4	a_5
R2	a_1	b_{22}	a_3	a_4	a_5
R3	a_1	b_{32}	b_{33}	a_4	a_5

If any one of the row contains complete 'a' values then Decomposition is lossless.

Problem 2 :

Consider R = (ABCDE) which is decomposed into R

R_1 = (A, B, C)

R_2 = (C, D, E) FD = {A → BC, CD → E, B → D, E → A}

Show that above decomposition of the schema R is not a lossless join decomposition.

Step1 : Creation of Matrix

	A	B	C	D	E
R1					
R2					

Step2 : Mention bij – i represents row no, j represent column no

	A	B	C	D	E
R1	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
R2	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}

Step3 : Mention ai value – i represents the column no

R_1 = (A, B, C) R_2 = (C, D, E)

R	A	B	C	D	E
R1	a1	a2	a3	b_{14}	b_{15}
R2	b_{21}	b_{22}	a3	a4	a5

Step4 : Compare each of the FD

A → BC —— no change

CD → E Compare next FD

CD → E

R	A	B	C	D	E
R1	a1	a2	a3	b_{14}	b_{15}
R2	b_{21}	b_{22}	a3	a4	a5

a3 → b_{14} b_{15}

a3 → a4 a5

No change

B → D

Its Corresponding values in the matrix

	A	B	C	D	E
R1	a_1	a_2	a_3	b_{14}	b_{15}
R2	b_{21}	b_{22}	a_3	a_4	a_5

$a_2 \rightarrow b_{14}$ }
$b_{22} \rightarrow a_{14}$ }

No change.

Compare next FD

E → A

	A	B	C	D	E
R1	a_1	a_2	a_3	b_{14}	b_{15}
R2	b_{21}	b_{22}	a_3	a_4	a_5

$a_1 \rightarrow b_{15}$ }
$b_{21} \rightarrow a_5$ }

No change

After applying all the FD the resultant matrix would be

R#	A	B	C	D	E
R1	a_1	a_2	a_3	b_{14}	b_{15}
R2	b_{21}	b_{22}	a_3	a_4	a_5

Steps : None of the rows contain complete 'a' values and hence it is not a lossless decomposition and it generates spurious tuple.

Dangling Tuple :

The tuple which affects the resultant of a relation on applying different forms of JOIN is referred to as Dangling Tuple (Fluctuating tuple).

Consider the relation.

EMP	EID	ENAME	DNO
	111	Adarsh	1
	112	Amith	1
	113	Arun	2
	114	Bharath	NULL

DEPT	DNO	DNAME
	1	CS
	2	IS

When inner join is applied the resultant of join would be

EMP * DEPT

EID	ENAME	DNO	DNAME
111	Adarsh	1	CS
112	Amith	1	CS
113	Arun	2	IS

Tuple <114, Chetan, NULL> is not present in the resultant since it does not satisfy the join condition.

EMP ⋈ DEPT

EMP.DNO = DEPT.DNO

EID	ENAME	DNO	DNAME
111	Adarsh	1	CS
112	Amith	1	CS
113	Arun	2	IS
114	Bharath	NULL	NULL

Dangling tuple

The same tuple is now present in the resultant relation. This is due to the use of LEFT OUTER JOIN operator.

→ The inclusion of tuple <114, Chetan, NULL> in the resultant relation depends on the kind on join operation and hence this tuple is returned as Dangling tuple.

Multivalued Dependencies AND FOURTH NORMAL FORM :

Multivalued dependencies (MVD) are a consequence of first normal form (1NF), which disallowed an attribute in a tuple to have set of values.

→ If we have two or more multivalued independent attribute in the same relation schema, we get into a problem of having to repeat every value of one of the attribute with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attributes involved. This constraint is specified by a multivalued dependency.

Example : Consider the relation EMP-phone-proj

EMP-phone-proj

EID	EpNo	EpNo
111	(96204, 42310)	(22,44)

EID	EpNo	EpNo
1111	96204	22
111	96204	44
111	42310	22
111	42310	44

Formal Definition of Multivalued Dependency:

A multivalued dependency (MVD) $X \twoheadrightarrow Y$ specified on relation schema R where X and Y are both subsets of R, specifies the following constraint on any relation state r of R: If two tuples t_1 and t_2 exists in r such that $t_1[X] = t_2[X]$ then two tuples t_3 and t_4 should also exist in r with the following properties. where we use Z to denote $(R - (X \cup Y))$

- $t_3[X] = t_4[X] = t_1[X] = t_2[X]$
- $t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$
- $t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$

Whenever $X \twoheadrightarrow Y$ holds, we say that X multidetermines Y. Because of the symmetry in the definition, whenever $X \twoheadrightarrow Y$ holds in R, also $X \twoheadrightarrow Z$ holds in R, Hence it is written as $X \twoheadrightarrow Y \mid Z$.

Fourth Normal Form: A relation schema R is in 4NF with respect to a set of dependencies F (that includes functional and multivalued dependencies) if, for every nontrivial multivalued dependency $X \twoheadrightarrow Y$ in F^+, X is a superkey for R.

Note: F^+ is the set of functional dependencies.

Example: Consider the relation EMP

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Apply 4th Normal Form on EMP.

EMP_PROJECTS

ENAME	PNAME
Smith	X
Smith	Y

ENAME	DNAME
Smith	John
Smith	Anna

Join Dependency and Fifth Normal Form

A join Dependency (JD) denoted by JD(R1, R2 ... Rn) specified on relation schema R, specifies a constraint on the states r of R.

→ The constraint states that every legal state of R should have a non-additive join decomposition into R1, R2 ... Rn; ie for every such r we have

$$* (\pi_{R1}(r), \pi_{R2}(r), \ldots \pi_{Rn}(r)) = r$$

→ A join dependency JD(R1, R2 ... Rn) specified on relation schema R, is a trivial JD if one of the relation schema Ri in JD(R1, R2 ... Rn) is equal to R.

Fifth Normal Form:

A relation schema R is in fifth normal form (5NF) (or project-join Normal FORM (PJNF) with respect to set of F of functional, multivalued, join dependencies if for every nontrivial join dependency JD(R1, R2 ... Rn) in F^+, for every Ri is a super key of R.

Example:

Consider the below given relation Supply is in 4NF.

SNAME	PARTNAME	PROJNAME
Smith	Bolt	projx
Smith	Nut	proj y
Adamsky	Bolt	proj y
Walton	Nut	proj z
Adamsky	Nail	proj x
Adamsky	Bolt	proj x
Smith	Bolt	proj y

Decomposition of above relation using 5NF is given below.

R1

SNAME	PARTNAME
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R2

SNAME	PROJNAME
Smith	proj x
Smith	proj y
Adamsky	proj y
Walton	proj z
Adamsky	Proj x

R3

PARTNAME	projNAME
Bolt	Proj x
Nut	proj y
Bolt	Proj y
Nut	proj z
Nail	Proj x

Inclusion dependency:

Inclusion dependencies were defined in order to explain certain inter relational constraints within the relation or across the relation.

→ Foreign key constraint cannot be specified as a functional or multi-valued dependency.

→ Foreign key relates attributes from one relation to attribute within the same relation or attribute across a different relation, this can be specified as Inclusion dependency.

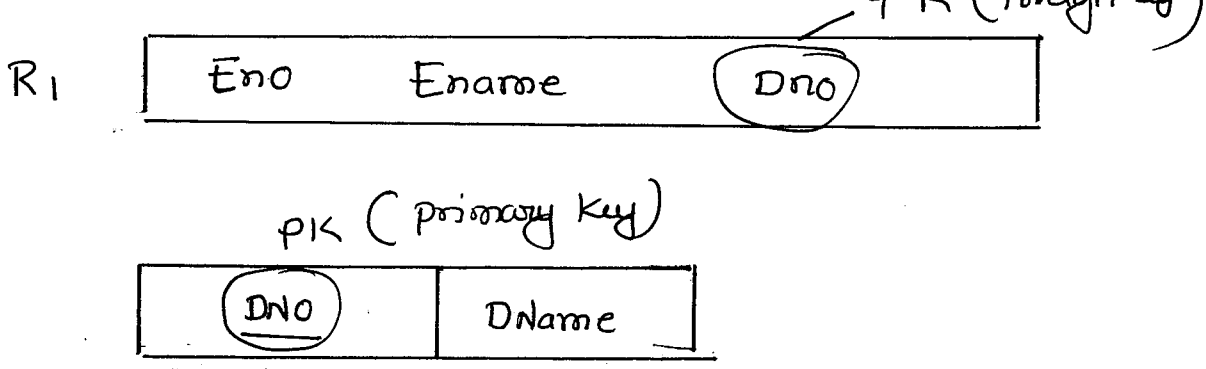
→ Inclusion dependency can also be used to represent the constraint between two relation that represent a class/subclass.

→ Formally an Inclusion dependency is represented as $R.X < S.Y$ between two sets of attributes X of relation schema R and Y of relation schema S.

$$\Pi_X (r(R)) \subseteq \Pi_Y (s(S))$$

→ Set of attributes on which inclusion dependency is specified X of R and Y of S must have same attributes and their domain must be same

Example: Consider below given Relation schema.



Template Dependency:

Constraint - that is based on definition and constraints based on semantics of attributes within a relation can be represented as Template dependency.

↳ The idea behind template dependencies is to specify a template or example that show how values are associated with each other and also define the resultant or condition that must be satisfied for the completion of the dependency.

→ Template contains two important regions

1. template hypothesis
2. template Conclusion.

→ A template hypothesis consists of a number of hypothesis tuples that show how the values should exist within a relation

→ The template conclusion can be indicated in two forms

1. Tuple-generating templates
2. Constraint generating templates.

In tuple-generating templates, the conclusion is represented as a set of tuples.

In Constraint-generating templates, the resultant is indicated as Condition.

Template for functional dependency $X \rightarrow Y$

$R = \{A, B, C\}$

$X = \{A\}$

$Y = \{B, C\}$

Hypothesis

$a_1 \quad b_1 \quad c_1$

$a_1 \quad b_2 \quad c_2$

.....

Conclusion $c_1 = c_2$ & $b_1 = b_2$

There are two tuples $<a_1, b_1, c_1>$ and $<a_1, b_2, c_2>$ according to the definition of FD if $t_1[X] = t_2[X]$ $(a_1 = a_1)$ then $t_1[Y] = t_2[Y]$ which is mentioned in the Colerion.

Template for Multi-valued dependency $X \rightarrow\rightarrow Y/Z$

$R = \{A, B, C\}$

$X = \{A\}$ $Y = \{B\}$

$Z = \{C\}$

Hypothesis a_1 b_1 c_1

a_1 b_2 c_2

Conclusion a_1 b_1 c_2

a_1 b_2 c_1

In MVD if $t_1[X] = t_2[X]$ $(a_1 = a_1)$ there must be additional tuple t_3 and t_4 in R which satisfies the following condition.

Templates for the Constraint that an employee's salary must be less than the supervisor's salary

EMPLOYEE = { NAME, SSN, SAL, SUPERVISORSSN }

Hypothesis a b c e

e d f g

Conclusion $c < f$

Let us assume the values as <112 (a), Bharath (b), 15000 (c), 111 (e) > & <111 (e), Arun (d), 20000 (f), NULL (g)> we observe Arun supervises bharath and hence 15000 (c) < 20000 (f).

Domain Key Normal FORM (DKNF) :

This normal form is considered as ultimate normal form. This normal form represents all the possible representation as constraints with a relation.

→ A relation schema is said to be in DKNF if the current state of the relation has values which satisfy constraints and dependencies which are valid for a relation. It also includes the key constraint and referential constraints also.

→ This can be enforced by using DBMS languages which has the capability of specifying domain constraint, key constraint and semantic constraints.

5 Sem CS DBMS

MODULE - 5

Transaction Processing

1. Introduction to Transaction Processing.
2. Transaction and System Concepts.
3. Desirable properties of Transactions.
4. Characterizing Schedules based on recoverability
5. Transaction support in SQL

Concurrency Control in Databases:

1. Two phase Locking techniques for concurrency control
2. Concurrency control based on Timestamp ordering
3. Multi version concurrency control techniques.
4. Validation concurrency control techniques
5. Granularity of Data items and Multiple Granularity Locking

Introduction to Database Recovery Protocols:

1. Recovery Concepts
2. NO-UNDO/REDO recovery based on Deferred update.
3. Recovery techniques based on immediate update.
4. Recovery techniques based on immediate update.
5. Shadow paging
6. Database backup and recovery from catastrophic failures.

Transaction : A transaction is a logical unit of database processing that includes one or more database access operation.

Definition 2: Execution of a single user request.

→ Basically there are 2 types of operations in a transaction

1. Read operation → do not update the database
2. Write operation → update the database

Example : Transaction 1 (T1)

T1

Balance Enquiry → R(A) → 5000 (Acc Balance)

Not updates the database

Transaction 2 (T2)

T1

Withdrawal → R(A) 5000 3000

W(A) 2000 (Acc Balance)

Single User vs Multiuser System :

One Criterion for classifying a database system is according to the number of users who can use the system concurrently.

→ A DBMS is single-user if 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.

ex: An airline reservation system, database used in banks, insurance agencies, stock exchanges, supermarket and many other applications are multiuser systems.

→ Multiple users can access databases — and use computer systems — simultaneously because of the concept of multi-programming which allows the operating system

of the computer to execute multiple programs or process at the same time.

→ Multiprogramming operating system execute some commands from one process then suspend that process and execute some commands from next process and so on.

→ However, Concurrent execution of processes is actually interleaved as illustrated in the fig below, which shows two processes A and B executing concurrently in an interleaved fashion.

→ In the computer system has multiple hardware processors (CPU). Parallel processing of multiple processes is possible as illustrated by process C and D in the below fig.

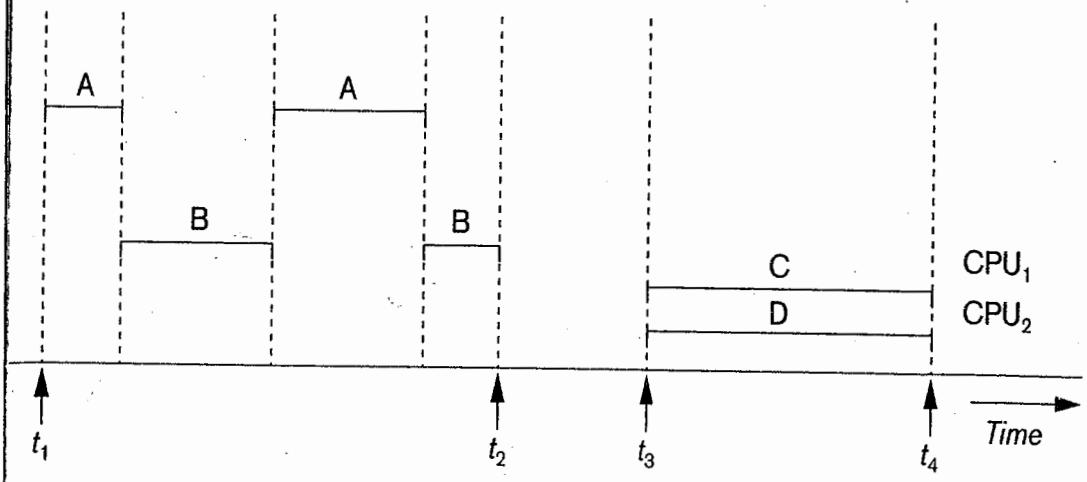


Figure 20.1
Interleaved processing versus parallel processing of concurrent transactions

Transactions, Database Items, Read and Write operations and DBMS Buffers.

A transaction is an executing program that forms a logical unit of database processing.

→ A transaction includes one or more database access operations – these can include insertion, deletion, modification or retrival operations.

→ The way of specifying the transaction boundries is by specifying explicit begin transaction and end transaction statements

Database operation in a transaction do not update the database but only retrieve data, the transaction is called read-only transaction. otherwise it is known as a "read-write operation

→ Database item or data item can be a database record but it can also be a larger unit. such as a whole disk block or even a smaller unit such as an individual field (attribute) value of some record in the database.

→ Database access operation that a transaction can include are as follows.

1 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.

2. write_item(x). Writes the value of program variable x into the database item named x.

→ Executing a red_item(x) command includes the following steps.

1. Find the address of the disk block that contains item x.
2. Copy the disk block into a buffer in main memory.
3. Copy item x from buffer to the program variable named x

Executing a write_item(x) command includes following steps

1. Find the address of the disk that contains item x.
2. Copy that disk block into a buffer in main memory.
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.

Why Concurrency Control is Needed :

Before we move to Concurry Control, let us understand what is Concurrent- execution and Need-for Concurrent-execution

Concurrent-Execution : Concurrent- execution is multiple transactions getting executed simultaneously.

→ Another name for concurrent- execution is "Interleaved execution"

Need for Concurrent-Execution :

1. Increases Throughput : Concurrent- execution Increases the throughput- by reducing waiting time. of Processor

2. Increases Average time taken to Complete a transaction.

Why Concurrency Control is Needed : Several problems will occor when Concurrent transactions execute in an uncontrolled manner.

→ Concurrency problems are explained by using "Airline reservations database in which a record is stored for each airline flight.

→ Each record includes the number of reserved seats on that flight as a named data item, among other information.

→ Figure below shows two transaction T_1 and T_2

(a)

T_1
read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

(b)

T_2
read_item(X);
$X := X + M$;
write_item(X);

Figure 20.2
Two sample transactions. (a) Transaction T_1. (b) Transaction T_2.

Fig 20.2(a) shows a transaction T_1 that transfers N reservations from one flight whose number of reserved seats is stored in the database item named X to another flight whose number of reserved seats is stored in the database item named Y.

Fig 20.2(b) shows a simpler transaction T_2 that just reserves M seats on the first flight (X) referenced in transaction T_1.

→ When a database access program is written, it has the flight-number, flight-date, and the number of seats to be booked as Parameters; hence the same program can be used to execute many different transactions each with a different flight number date and number of seats to be booked.

→ The various problems we encounter due to Concurrency are listed below.

1. Lost Update Problem
2. Temporary Update (or Dirty Read) Problem.
3. The Incorrect Summary Problem
4. Unrepeatable Read Problem.

Lost Update Problem :

This problem occurs when two transaction that access the same database items have their operations interleaved in a way that makes the value of some database items incorrect. The last update problem is explained below.

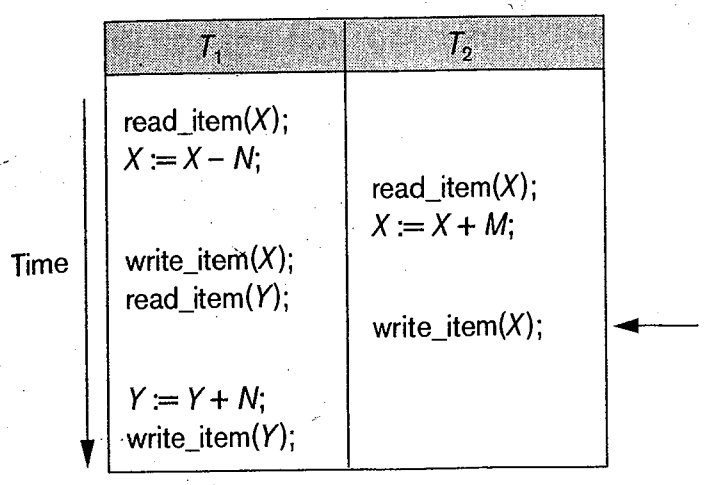


Figure 20.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

Suppose that transaction T_1 and T_2 are submitted at approximately the same time and suppose that their operations are interleaved as shown in above fig.

→ The final value of item X is incorrect because T_2 reads the value of X before T_1 changes it in the database and hence the updated value resulting from T_1 is lost.

→ For example

If X = 80 at the start (originally there were 80 reservations on the flight)

N = 5 (T_1 transfer 5 seat reservations from the flight corresponding to X to the flight corresponding to Y) and

M = 4 (T_2 reserves 4 seats on X) and

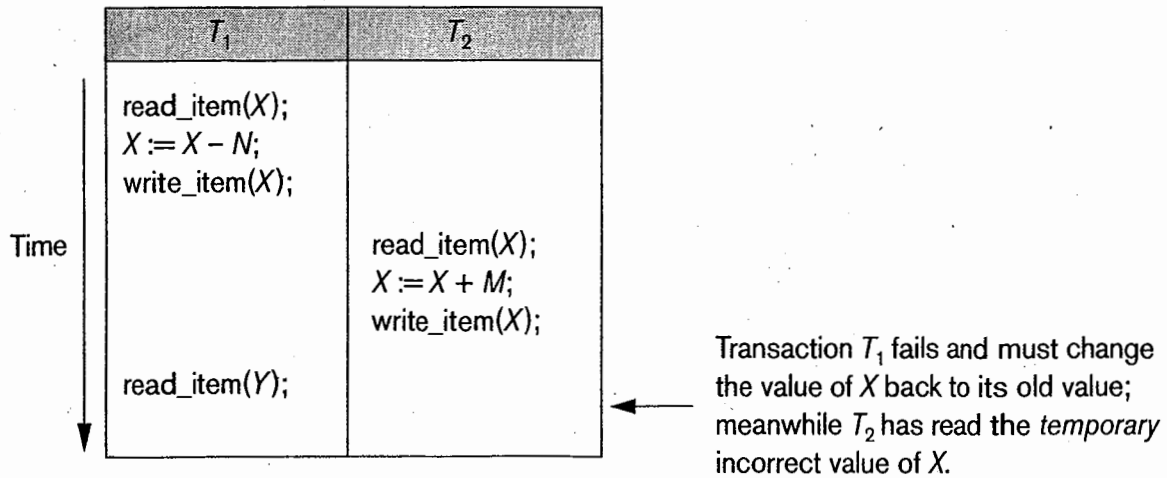
final result should be X = 79.

→ However in the interleaving of operations shown in the fig above, it is X = 84 because the update in T_1 that removed the five seats from X was lost.

The Temporary Update (or Dirty Read) Problem :

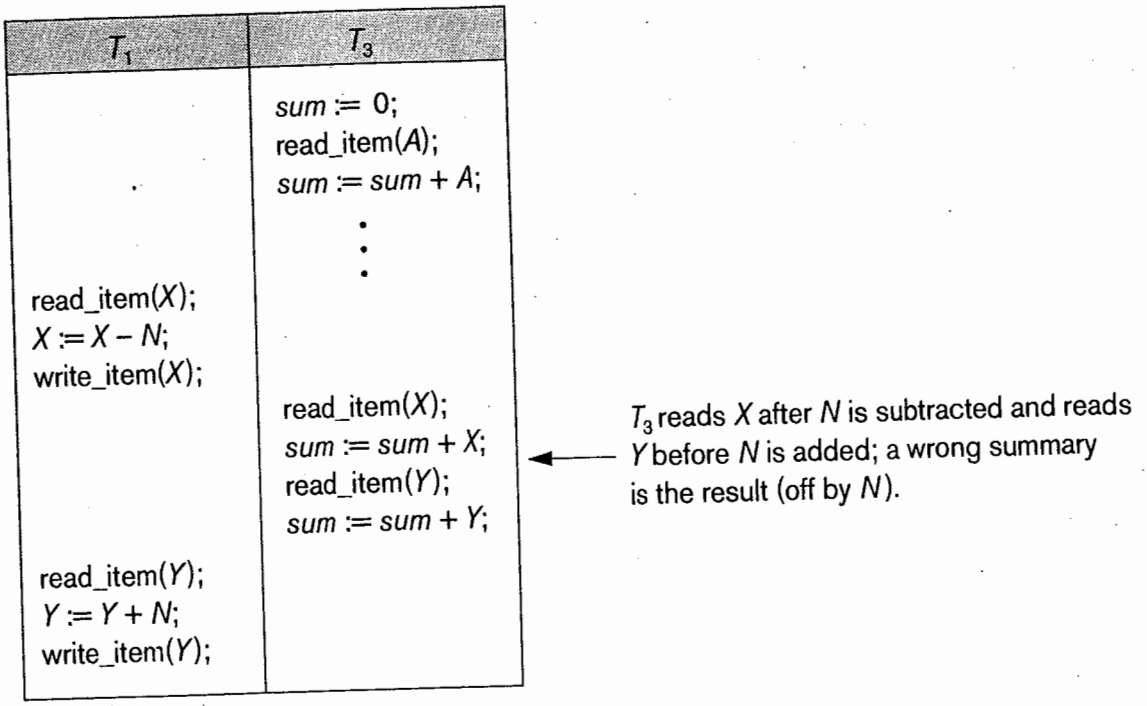
This problem occurs when one transaction updates a database item and then the transaction fails for some reason

→ Mean while, the updated item is accessed (read) by another transaction before it is changed back it's original value



The Incorrect Summary problem

If one transaction is calculating an aggregate summary function on a number of database items while other transaction are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.



For example, suppose meanwhile, transaction T1 is executing. If the interleaving of operations shown in fig above occurs, the result of T3 will be off by an amount N 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.

The Unrepeatable Read Problem:

Another problem that may occur is called unrepeatable read, where a transaction T reads the same item twice and the item is changed by another transaction T between the two reads.

Hence T receives different values for its two reads of the same item

This may occur, for example, during an airline reservation transaction, a customer enquires about seat availability on several flights.

→ When the customer decides on a particular flight, the transaction then reads the number of seats on that flight a second time before completing the reservation, and it may end up reading a different value for the item.

Why Recovery is Needed:

Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, or transaction doesn't have any effect on the database.

In the first underlined case, the transaction is said to be Committed, where in the second case, the transaction is aborted.

→ Transaction Abort takes place when " Transaction failure occurs, transaction fails after executing some its operations but before executing all of them, the operations already executed must be undone and have no lasting effect.

Types of Failure:

Failures are generally classified as transaction, system and media failures. There are several 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. Hardware crashes are usually media failures — 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 errors.

3. Logical 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, An exception condition such as insufficient account balance in a banking database may cause a transaction, such as a fund withdrawal to be canceled.

4. Concurrency Control enforcement: The concurrency control method may decide to abort a transaction because it violates serializability or it may abort one or more transaction to resolve a state of deadlock among several transactions.

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.

Transaction and System Concepts:

This section covers the concepts like

1. Transaction States and Additional Operations
2. The System Log
3. Commit point of a Transaction.

Transaction States and Additional Operations :

A transaction is an atomic unit of work that should either be completed in its entirety or not done at all.

→ The following fig shows state transition diagram illustrating the states for transaction execution.

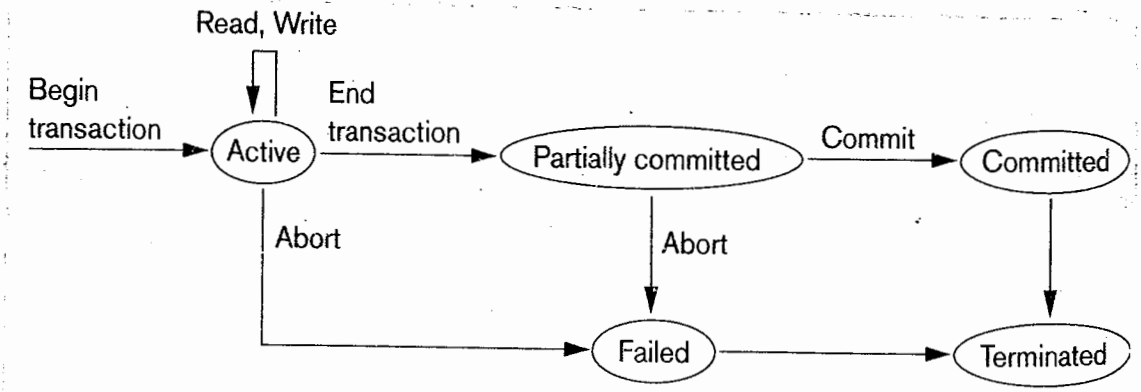


Figure 20.4
State transition diagram illustrating the states for transaction execution.

For recovery purposes, the system needs to keep track of when each transaction starts, terminates, and commits or aborts.

→ Therefore the recovery manager of the DBMS need to keep track of the following operations :

- Begin_Transaction : This marks the begining of the transaction.
- Read_write : These specify read or write operations on the database items that are executed as part of a transaction.
- END_Transaction : This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution.

→ However at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database or whether the transaction has to be aborted because it violates serializability.

- 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.

- 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.

The System Log:

To be able to recover from failures that affect transactions, the system maintains a log to keep track of all transaction operations that affect the values of database items, as well as other transaction information that may be needed to permit recovery from failures.

→ The log is a sequential, append-only file that is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure.

→ When the log buffer is filled, or when certain other conditions occur, the log buffer is appended to the end of the log file on disk.

→ The following are the types of entries called log records that are written to the log file and the corresponding action for each log record.

1. [start_transaction, T]. Indicates that transaction T has started execution.

2. [write_item, 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_item, 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 to the database.

5. [abort, T]. Indicates that transaction T has been aborted

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 operation on the database have been recorded in the log.

→ Beyond the commit point, the transaction is said to be committed and its effect must be permanently recorded in the database.

Desirable Properties of Transactions :

Transactions should possess several properties, often called the ACID properties; they should be enforced by the Concurrency Control and recovery methods of the DBMS. The following are the ACID properties.

1. Atomicity
2. Consistency Preservation.
3. Isolation
4. Durability or Permanency

Atomicity : A transaction is an atomic unit of processing it should either be performed in its entirety or not performed at all.

Consistency Preservation : A transaction should be consistency preserving meaning that it it is completely executed from begining to end without interference from other transactions,

it should take the database from one consistent state to another.

Isolation : A transaction should appear as though it is being executed in isolation from other transactions. Even though many transactions are executing concurrently. That is the execution of a transaction should not be interfered with any other transactions executing concurrently.

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.

Characterizing Schedules Based on Recoverability :

" Order of concurrent-execution of operations from all the various transactions is known as schedule.

→ A schedule S of n transactions $T_1, T_2 \ldots T_n$ is an ordering of the operations of the transactions. (operations from different transactions can be interleaved in the schedule S)

→ Each Transaction T_i that participates in the schedule S, the operations of T_i in S must appear in the same order in which they occur in T_i.

→ The order of operations in S is considered to be a total ordering, meaning that for any two operations in the schedule, one must occur before the other.

→ For the purpose of recovery and concurrency control, we are mainly interested in the read_item and write_item operations of the transactions as well as the commit and abort operations.

→ For example consider the schedules

(wrt diagram 2.0.3a) Sa: $r_1(X)$: $r_2(X)$: $w_1(X)$: $r_1(Y)$: $w_2(X)$: $w_1(Y)$:

(wrt diagram 2.0.3b) Sb: $r_1(X)$: $w_1(X)$; $R_2(X)$: $w_2(X)$: $r_1(Y)$: a_1 ;

A Schedule S of n transactions $T_1, T_2, \ldots T_n$ is said to be a Complete Schedule if the following conditions hold:

1. The operations in S are exactly those operations in $T_1, T_2 \ldots T_n$, including a commit or abort operation as the last operation for each transaction in the Schedule.

2. For any pair of operations from the same transaction T_i, their relative order of apperance in S is the same as their order of appearance in T_i

3. For any two conflicting operations, one of the two must occur before the other in the Schedule.

Characterizing Schedules Based on Recoverability :

For some Schedules it is easy to recover from transaction and and System failures, whereas for other Schedules the recovry Process can be quite involved. In some cases, it is even not possible to recover correctly after a failure.

Hence it is important to characterize the types of Schedules for which recovery is possible as well as those for which recovery is relatively simple.

→ Schedules can be catagorized into

1. Recoverable Schedule
2. Non Recoverable Schedule.

Recoverable Schedule : Durability property of transaction is not violated. The Schedule that theoretically meet this Criterion are called recoverable Schedule.

Non-Recoverable Schedule : The transaction doesn't meet ACID properties then Schedule is said to be Non-Recoverable.

→ In a recoverable schedule, no committed transaction ever needs to be rolled back, and so the definition of committed transaction as durable is not violated. Hence it is possible for a phenomenon as cascading rollback.

→ Because cascading rollback can be quite time consuming since numerous transactions can be rolled back, it is important to characterize the schedules where this phenomenon not to occur. A schedule is said to be cascadeless

→ Finally there is a third, more respective type of schedule called a strict schedule, in which transaction can neither read nor write an item X until the last transaction that wrote x has committed.

Characterizing Schedules Based on Serializability :

Serial Concurrent transactions are executing correctly as per the schedule (without any conflicts), such schedules are known as Serializable schedule.

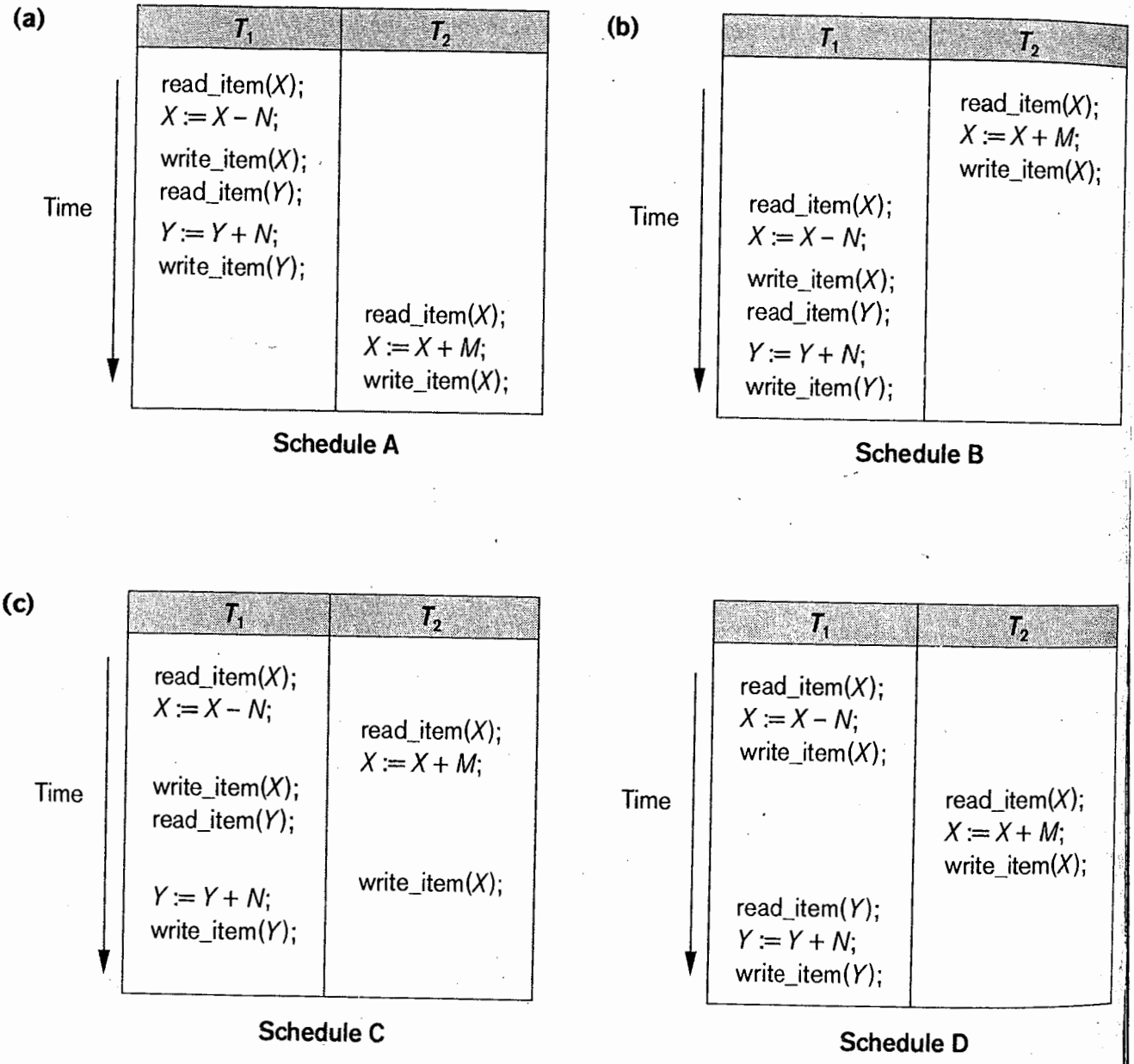
Consider the figure 20.2 for airline reservation agents - submit to the DBMS transactions T_1 and T_2

1. Execute all the operations of transaction T_1 (in sequence) followed by all the operations of transaction T_2 (in sequence).

2. Execute all the operations of transaction T_2 (in sequence) followed by all the operations of transaction T_1 (in sequence)

The below figure shows serial schedules and Non-serial schedules.

Figure 20.5
Examples of serial and nonserial schedules involving transactions T_1 and T_2. (a) Serial schedule A: T_1 followed by T_2. (b) Serial schedule B: T_2 followed by T_1. (c) Two nonserial schedules C and D with interleaving of operations.



These two Schedules – called Serial Schedules – are shown figure(a). and (b) respectively.

If interleaving of operations is allowed, there will be many possible orders in which the System can execute the individual operations of the transactions.

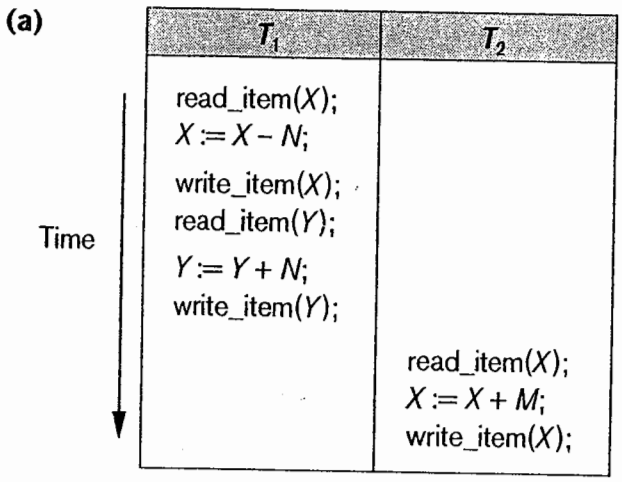
figure (c) and (d) are nonserial Schedule C and D with interleaving of operations.

Serial, Non Serial and Conflict – Serializable Schedules :

Schedule given below are called Serial because the operations of each transaction are executed consecutively, without any interleaved operations from the other transaction

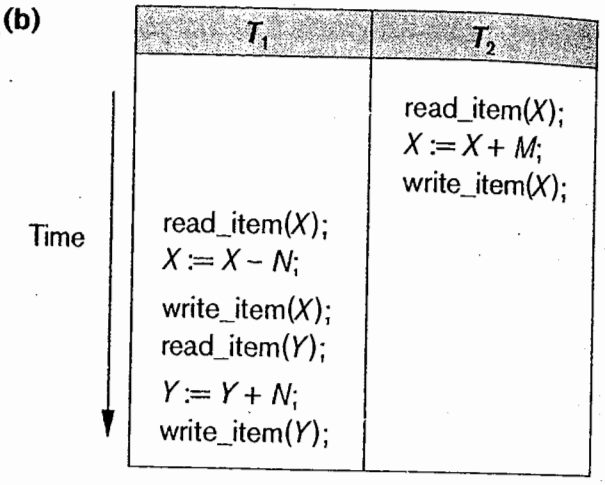
20.5(a) +(b)

(a)



Schedule A

(b)

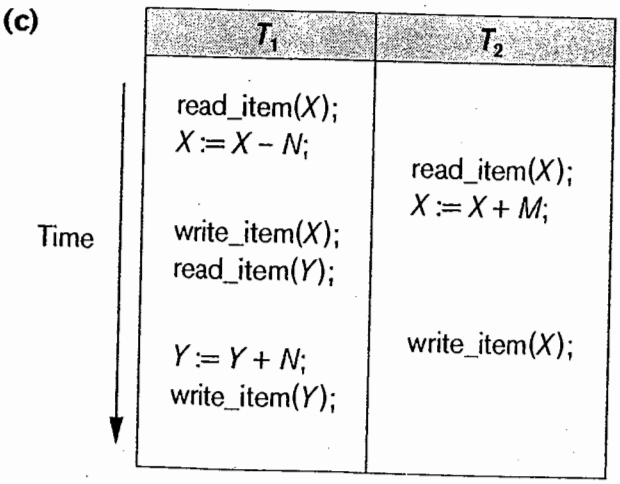


Schedule B

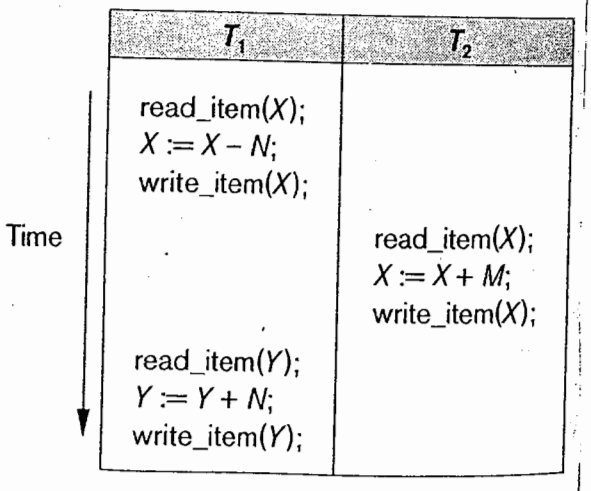
Formally a schedule S in serial, if for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule otherwise the schedule is called non serial.

20.5(c) (d)

(c)



Schedule C



Schedule D

Conflict — Serializable Schedules :

The definition of ~~elir~~ conflict equivalence of schedules is as follows : Two schedules are said to be conflict-equivalent- if the order of any two conflicting operations in the same in both schedules.

→ Two operations in a schedule are said to conflict- if they belong to different transactions, access the same database item, and either both are write_item operations or one is a write_item and the other a read_item.

Testing for Conflict Serializability of a Schedule :

There is a simple algorithm for determining whether a particular schedule is conflict- serializable or not.

→ Most- Concurrency control methods do not actually test for serializability.

Algorithm below given can be used to test a schedule for conflict- serializability.

Algorithm : Testing conflict serializability of a schedule S

1. For each transaction T_i participating in schedule S, create a node labeled T_i in the precedence graph.

2. For each case in S where T_j executes a read_item(x) after T_i executes a write_item(x), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.

3. For each case in S where T_j executes a write_item(x) after T_i executes a read_item(x), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.

4. For each case in S where T_j executes a write_item(x) after T_i executes a write_item(x), create an edge $(T_i \rightarrow T_j)$ in the precedence graph.

5. The Schedule S is Serializable if and only if the precedence graph has no cycles.

The precedence graph is constructed as described in the ~~below~~ above algorithm, if there is a cycle in the precedence graph Schedule S is not Serializable (conflict) if there is no cycles, then S is Serializable.

• The following fig shows construction of precedence graph for Schedules A to D. (fig 20.5 given earlier in the notes of module 5)

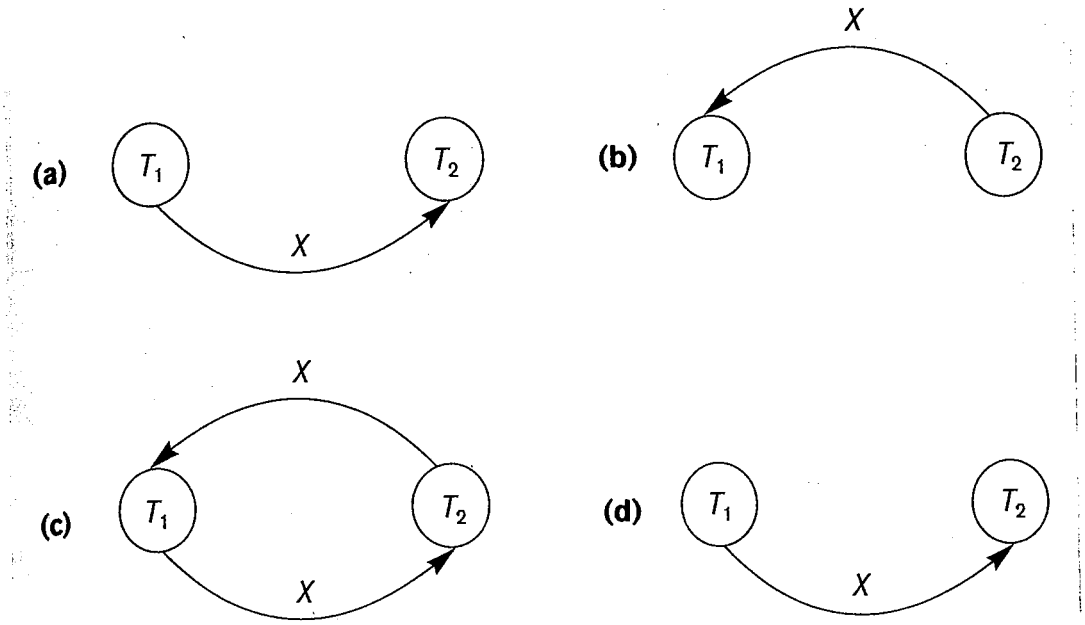


Figure 20.7
Constructing the precedence graphs for schedules A to D from Figure 20.5 to test for conflict serializability. (a) Precedence graph for serial schedule A. (b) Precedence graph for serial schedule B. (c) Precedence graph for schedule C (not serializable). (d) Precedence graph for schedule D (serializable, equivalent to schedule A).

How Serializability is used for Concurrency Control :

Schedule S is (conflict) Serializable – ie, S is equivalent to a Serial Schedule – is tantamount to saying that S is correct.

→ Being Serializable is distinct from being Serial, however.

→ A Serial Schedule represents inefficient processing because no interleaving of operations from different transaction is permitted.

→ This can lead to low cpu utilization while a transaction waits for disk I/O, or for another transaction to terminate, thus slowing down processing considerably.

→ A serializable schedule gives the benefits of concurrent-execution without giving up any correctness.

→ In practice it is quite difficult to test for the serializability of a schedule. The interleaving of operations from concurrent-transactions which are usually executed as processes by the operating system - is typically determined by the operating system scheduler, which allocates resources to all processes. Factors such as system load, time of transaction submission and priorities of processes contribute to the ordering of operations in a schedule

Hence it is difficult to determine how the operations of a schedule will be interleaved beforehand to ensure serializability.

→ DBMS Concurrency Control Subsystem - will ensure serializability of all schedules in which the transactions participate.

View Equivalence and View Serializability:

Less restrictive definition of equivalence of schedule is called view equivalence. This leads to another definition of serializability called view serializability.

Two schedules S and S' are said to be view equivalent if the following 3 conditions hold.

1. The same set of transactions participates in S and S' and S and S' include the same operations of those transactions

2. For any operation $r_i(X)$ of T_i in S, if the value of X read by the operation has been written by an operation $w_j(X)$ of T_j, the same condition must hold for the value of X read by operation $r_i(X)$ of T_i in S'.

3. If the operation $W_K(y)$ of T_K is the last operation to write item y in S, then $W_K(y)$ of T_K must also be the last operation to write item y in S'

Transaction Support in SQL :

A Single SQL Statement is always considered to be atomic — either it Completes execution without an error or it fails and leaves the database unchanged.

→ With SQL, there is no explicit Begin_Transaction initiation is done implicitly when particular SQL statements are encountered.

→ Every transaction must have an explicit end statement, which is either a COMMIT or a ROLLBACK.

The access mode can be specified as READ ONLY or READ WRITE. The default is READ, WRITE unless the isolation level of READ UNCOMMITTED is specified in which case READ ONLY is assumed.

The diagnostic area size option, DIAGNOSTIC SIZE, n specifies an integer value n, which indicates the number of conditions that can be held simultaneously in the diagnostic area.

The isolation level option is specified using the statement- ISOLATION LEVEL <isolation> where the value for <isolation> can be READ UNCOMMITTED. READ COMMITTED, REPEATABLE READ or SERIALIZABLE.

→ If a transaction executes at a lower isolation level than Serializable, then one or more of the following 3 violations may occur:

1. Dirty Read : A transaction T_1 may read the update of a transaction T_2, which has not yet committed. If T_2 fails and is aborted, then T_1 would have read a value

that does not exist and is incorrect.

2. Nonrepeatable Read : A transaction T_1 may read a given value from a table. If another transaction T_2 later updates that value and T_1 reads that value again, T_1 will see a different value.

3. phantoms : A transaction T_1 may read a set of rows from a table, perhaps based on some condition specified in the SQL WHERE – clause. Now suppose that a transaction T_2 inserts a new row that also satisfies the WHERE – clause condition used in T_1, into the table used by T_1. If T_1 is repeated then T_1 will see a phantom, a row that previously did not exist.

Table below shows, summarizes the possible violations for the different isolation levels.

Table 20.1 Possible Violations Based on Isolation Levels as Defined in SQL

	Type of Violation		
Isolation Level	**Dirty Read**	**Nonrepeatable Read**	**Phantom**
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

A sample SQL transaction might look like the following:

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
    READ WRITE
    DIAGNOSTIC SIZE 5
    ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno, Salary)
    VALUES ('Robert', 'Smith', '991004321', 2, 35000);
EXEC SQL UPDATE EMPLOYEE
    SET Salary = Salary * 1.1 WHERE Dno = 2;
EXEC SQL COMMIT;
GOTO THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END: ... ;
```

Concurrency Control in Databases : (Second part - in Module 5)

1. Two-phase Locking Techniques for Concurrency Control.
2. Concurrency Control based on Timestamp ordering.
3. Multiversion concurrency control techniques.
4. Validation Concurrency control techniques
5. Granularity of data items and Multiple Granularity Locking.

Two-phase Locking Techniques for Concurrency Control :

Main techniques used to control Concurrent execution of transactions are based on the concept of locking data items.

→ A lock is a variable associated with a dataitem that describes the status of the item with respect to possible operations that can be applied to it.

→ Generally there is one lock for each data item in the database.

→ Locks are used as a means of synchronizing the access by Concurrent transactions to the database item.

→ The below section describes nature and type of locks.

```
lock_item(X):
B:  if LOCK(X) = 0              (* item is unlocked *)
        then LOCK(X) ←1         (* lock the item *)
    else
        begin
        wait (until LOCK(X) = 0
            and the lock manager wakes up the transaction);
        go to B
        end;
unlock_item(X):
    LOCK(X) ← 0;                (* unlock the item *)
    if any transactions are waiting
        then wakeup one of the waiting transactions;
```

Figure 21.1
Lock and unlock operations for binary locks.

Types of Locks and System Lock Tables:

Several types of locks are used in Concurrency control. To introduce locking concepts gradually, first we discuss binary locks, which are simple, but are also too restrictive for database concurrency control purposes, and so are not used in Practice. Then we discuss shared/exclusive locks also known as read/write locks — which provide more general locking capabilities and are used in partical database locking schemes.

Binary Locks

A binary lock can have two states or values: locked and unlocked (or 1 and 0 for simplicity). A distinct lock is associated with each database item X. If the value of the lock on X is 1. item X cannot be accessed by a database by a database operation that requests the item. If the value of the lock on X is 0, the item can be accessed when requested and the lock value is changed to 1. We refer to the current value (or state) of the lock associated with item X as lock(X)

Two operations, lock_item and unlock_item are used with binary locking. A transaction requests access to an item X by first issuing a lock_item(X) operation. If Lock(X) = 1, the transaction is forced to wait. If Lock(X) = 0, it is set to 1 (the transaction locks the item) and the transaction is allowed to access item X.

→ When the transaction is through using the item, it issues an unlock_item(X) operation, which sets Lock(x) back to 0 (unlocks the item) so that X may be accessed by other transactions. Hence binary lock enforces mutual exclusion on the data item.

→ In its simplest form each lock can be recorded with 3 fields: <Data_item_name, Lock, locking transaction> plus a queue for transactions that are waiting to access the item.

The system needs to maintain *only these records for the items that are currently locked* in a **lock table,** which could be organized as a hash file on the item name. Items not in the lock table are considered to be unlocked. The DBMS has a **lock manager subsystem** to keep track of and control access to locks.

If the simple binary locking scheme described here is used, every transaction must obey the following rules:

1. A transaction T must issue the operation lock_item(X) before any read_item(X) or write_item(X) operations are performed in T.
2. A transaction T must issue the operation unlock_item(X) after all read_item(X) and write_item(X) operations are completed in T.
3. A transaction T will not issue a lock_item(X) operation if it already holds the lock on item X.[1]
4. A transaction T will not issue an unlock_item(X) operation unless it already holds the lock on item X.

The rules can be enforced by the lock manager module of the DBMS. Between the lock_item(X) and unlock_item(X) operations in transaction T, T is said to hold the lock on item X. At most one transaction can hold the lock on a particular item. Thus no two transaction can access the same item concurrently.

Shared/Exclusive (or Read/Write) Locks. The preceding binary locking scheme is too restrictive for database items because at most, one transaction can hold a lock on a given item. We should allow several transactions to access the same item X if they all access X for *reading purposes only*. This is because read operations on the same item by different transactions are not conflicting (see Section 20.4.1). However, if a transaction is to write an item X, it must have exclusive access to X. For this purpose, a different type of lock called a **multiple-mode lock** is used. In this scheme—called **shared/exclusive** or **read/write** locks—there are three locking operations: read_lock(X), write_lock(X), and unlock(X). A lock associated with an item X, LOCK(X), now has three possible states: *read-locked*, *write-locked*, or *unlocked*. A **read-locked item** is also called **share-locked** because other transactions are allowed to read the item, whereas a **write-locked item** is called **exclusive-locked** because a single transaction exclusively holds the lock on the item.

One method for implementing the preceding operations on read/write lock is to keep track of the number of transactions that hold a shared(read) lock on an item in the lock table

→ Each record in the lock table will have four fields

< Data_item_name , Lock , No_of_reads , Locking transaction(s) >

Again to save space, the system needs to maintain lock records only for locked items in the lock-table.

→ The value (state) of Lock is either read_locked or write_locked, suitably coded

→ If Lock(x) = write_locked, the value of locking_transaction(s) is a single transaction that holds the exclusive (write) lock on x. If Lock(x) = read_locked, the value of locking transaction(s) is a list of one or more transactions that hold the shared (read) lock on x. The three operations read-lock(x), write_lock(x) and unlock(x) as described below.

```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin LOCK(X) ← "read-locked";
                no_of_reads(X) ← 1
                end
    else if LOCK(X) = "read-locked"
        then no_of_reads(X) ← no_of_reads(X) + 1
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
write_lock(X):
B:  if LOCK(X) = "unlocked"
        then LOCK(X) ← "write-locked"
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
                wakeup one of the waiting transactions, if any
                end
    else it LOCK(X) = "read-locked"
        then begin
                no_of_reads(X) ← no_of_reads(X) –1;
                if no_of_reads(X) = 0
                    then begin LOCK(X) = "unlocked";
                            wakeup one of the waiting transactions, if any
                            end
                end;
```

Figure 21.2
Locking and unlocking operations for two-mode (read-write or shared-exclusive) locks.

When we use the shared/exclusive locking scheme, the system must enforce the following rules:

1. A transaction T must issue the operation read_lock(X) or write_lock(X) before any read_item(X) operation is performed in T.
2. A transaction T must issue the operation write_lock(X) before any write_item(X) operation is performed in T.

3. A transaction T must issue the operation unlock(X) after all read_item(X) and write_item(X) operations are completed in T.[3]
4. A transaction T will not issue a read_lock(X) operation if it already holds a read (shared) lock or a write (exclusive) lock on item X. This rule may be relaxed, as we discuss shortly.

5. A transaction T will not issue a write_lock(X) operation if it already holds a read (shared) lock or write (exclusive) lock on item X. This rule may also be relaxed,

6. A transaction T will not issue an unlock(X) operation unless it already holds a read (shared) lock or a write (exclusive) lock on item X.