S.Y.B.SC.(IT)

SEMESTER - III (CBCS)

DATA BASE MANAGEMENT SYSTEMS

SUBJECT CODE : USIT304

Prof. Suhas Pednekar
Vice Chancellor
University of Mumbai, Mumbai.

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai.

Prof. Prakash Mahanwar
Director
IDOL, University of Mumbai.

Programme Co-ordinator : **Prof. Mandar Bhanushe**
Head, Faculty of Science & Technology,
IDOL, University of Mumbai - 400 098.

Course Co-ordinator **: Gouri S. Sawant**
Assistant Professor B.Sc.IT, IDOL,
University of Mumbai- 400098.

Course Writers **: Ms. SeemaViswakarma**
Assistant Professor,
Vidyalankar School of Information Technology,
Wadala.

: Ms. Madhavi Amondhkar
Assistant Professor,
Vidyalankar School of Information Technology,
Wadala

: Mr. Amit Kukreja
Assistant Professor,
KJ Somaiya Institute of Engineering and Information Technology,
Sion.

: Ms. Ashwini Koyande
Assistant Professor,Vidyalankar School of Information Technology,
Wadala

July 2021, Print I

Published by **: Director**
Institute of Distance and Open Learning ,
University of Mumbai,
Vidyanagari, Mumbai - 400 098.

DTP COMPOSED AND PRINTED BY
Mumbai University Press
Vidyanagari, Santacruz (E), Mumbai - 400098.

CONTENT

*****

Syllabus

F.Y.B.Sc. (Information Technology)		**Semester – III**	
Course Name: Database Management Systems		**Course Code: USIT304**	
Periods per week (1 Period is 50 minutes)		**5**	
Credits		**2**	
		Hours	**Marks**
Evaluation System	**Theory**	**2½**	**75**
	Examination	**-**	**25**

Unit	Details	Lectures
I	**Introduction to Databases and Transactions** What is database system, purpose of database system, view of data, relational databases, database architecture, transaction management **Data Models** The importance of data models, Basic building blocks, Business rules, The evolution of data models, Degrees of data abstraction. **Database Design, ER Diagram and Unified Modeling Language** Database design and ER Model: overview, ER Model, Constraints, ER Diagrams, ERD Issues, weak entity sets, Codd's rules, Relational Schemas, Introduction to UML	**12**
II	**Relational database model:** Logical view of data, keys, integrity rules, Relational Database design: features of good relational database design, atomic domain and Normalization (1NF, 2NF, 3NF, BCNF). **Relational Algebra and Calculus** Relational algebra: introduction, Selection and projection, set operations, renaming, Joins, Division, syntax, semantics. Operators, grouping and ungrouping, relational comparison. **Calculus:** Tuple relational calculus, Domain relational Calculus, calculus vs algebra, computational capabilities	**12**
III	**Constraints, Views and SQL** Constraints, types of constrains, Integrity constraints, Views: Introduction to views, data independence, security, updates on views, comparison between tables and views SQL: data definition, aggregate function, Null Values, nested sub queries, Joined relations. Triggers.	**12**

IV	**Transaction management and Concurrency** Control Transaction management: ACID properties, serializability and concurrency control, Lock based concurrency control (2PL, Deadlocks), Time stamping methods, optimistic methods, database recovery management.	**12**
V	**PL-SQL**: Beginning with PL / SQL, Identifiers and Keywords, Operators, Expressions, Sequences, Control Structures, Cursors and Transaction, Collections and composite data types, Procedures and Functions, Exceptions Handling, Packages, With Clause and Hierarchical Retrieval, Triggers.	**12**

Books and References:

Sr. No.	**Title**	**Author/s**	**Publisher**	**Edition**	**Year**
1.	Database System and Concepts	A Silberschatz Sudarshan	McGrawHill	Fifth Edition	
2.	Database Systems	Rob Coronel	Cengage Learning	Twelth Edition	
3.	Programming with PL/SQL for Beginners	H. Dand, R. Patil and T. Sambare	X –Team	First	2011
4.	Introduction to Database System	C.J.Date	Pearson	First	2003

*****

UNIT I

1

INTRODUCTION TO DATABASE AND TRANSACTION

Unit structure

1.0 OBJECTIVES

The objective of the chapter is as follow

- To get familiar with core component of database management systems
- To understand the different architecture of database
- To understand the concept of relational database

1.1 INTRODUCTION

- A database management system (DBMS) is a collection of interrelated data and a set of programs to access those data.
- The collection of data, usually referred to as the database, contains information relevant to an enterprise.
- The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient
- Main purpose if to manage large bodies of information in a structured format

1.2 DATABASE SYSTEM APPLICATIONS

Databases are widely used many application. Some of them are mentioned below

- **Banking:** For customer information, accounts, and loans, and banking transactions.
- **Airlines:** For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner − terminals situated around the world accessed the central database system through phone lines and other data networks.
- **Universities:** For student information, course registrations, and grades.
- **Credit card transactions:** For purchases on credit cards and generation of monthly statements.
- **Telecommunication:** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
- **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
- **Sales:** For customer, product, and purchase information.

1.3 PURPOSE OF DATABASE SYSTEM

1.3.1 Data redundancy and inconsistency:

- Files and application programs created by different programmers have different structures and written using different programming languages.
- The same information may be duplicated in several places (files).
- This redundancy leads to higher storage and access cost.
- In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree. For example, a changed student address may be reflected in the one department records but not elsewhere in the system.

1.3.2 Difficulty in accessing data: The conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

1.3.3 Data isolation: Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult

1.3.4 Integrity problems: The data values stored in the database must satisfy certain types of consistency constraints

1.3.5 Atomicity problems: A computer system, like any other device, is subject to failure. Atomic—it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system

1.3.6 Security problems: Not every user of the database system should be able to access all the data

1.4 VIEW OF DATA

A database system is a collection of interrelated files and a set of programs that allow users to access and modify these files. A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

1.4.1 Data Abstraction:

- For the system to be usable, it must retrieve data efficiently.
- The need for efficiency has led designers to use complex data structures to represent data in the database.
- Since many users of databasesystems are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users interactions with the system:

Physical level: The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low−level data structures in detail.

Logical level: the next−higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures.

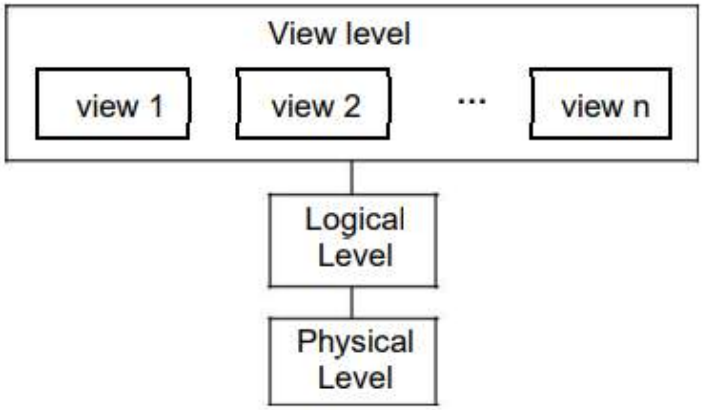


Fig. 1: The three levels of data abstraction.

1.4.2 Instances and Schema:

Databases change over time as information is inserted and deleted.

Instances:

- The collection of information stored in the database at a particular moment is called an **instance** of the database**.**
- The values of the variables in a program at a point in time correspond to an instance of a database schema

Schema:

- The overall design of the database is called the database **schema.**
- Schemas are changed infrequently.
- A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant.

1.5 RELATIONAL DATABASES

A relational database is based on the relational model and uses a collection of tables to represent both data and the relationship among those data. It also includes a DML and DDL.

1.5.1 Tables:

- The relational model is an example of a record-based model.
- Record-based models are so named because the database is structured in fixed-format records of several types.
- Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes.
- It is possible to create schemas in the relational model that have problems such as unnecessarily duplicated information. The relational model hides such low-level implementation details from database developers and users
- The columns of the table correspond to the attributes of the record type.
- The relational model hides such low-level implementation details from database developers and users.

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32143	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
3o456	Gold	Physic s	87000
76543	Singh	Finance	80000

(a) The instructor table.

dept_name	building	budget
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	35000
Music	Packard	80000
Finance	Painter	120000
History	Painter	130000
Phyics	Watson	70000

(b) The department table

Fig. 2 : A sample relational database.

1.6 DATA-MANIPULATION LANGUAGE

The SQL query language is nonprocedural. A query takes as input several tables (possibly only one) and always returns a single table. Here is an example of an SQL query that finds the names of all instructors in the History department:

select instructor.name
from instructor
where instructor.dept_name = 'History'

The query specifies that those rows from the table instructor where the dept nameis

History must be retrieved, and the name attribute of these rows must be displayed.

1.7 DATA-DEFINITION LANGUAGE

SQL provides a rich DDL that allows one to define tables, integrity constraints, assertions, etc.

For instance, the following SQL DDL statement defines the department table:

```
create table department
(depLnamechar(20),
building          char (15),
budget            numeric (12,2))
```

Execution of the above DDL statement creates the department table with three columns :dept_name, building, and budget, each of which has a specific data type associated with it. In addition, the DDL statement updates the data dictionary, which contains metadata. The schema of a table is an example of metadata

1.8 DATABASE ACCESS FROM APPLICATION PROGRAMS

SQL is not as powerful as a universal Turing machine; that is, there are some computations that are possible using a general-purpose programming language but are not possible using SQL. SQL also does not support actions such as input users, output to displays, or communication over the network. Such computations are supposed to be written in host language such as C, C++ or java with embedded SQL queries that access the data in the database. Application programs are programs that are used to interact with the database in this fashion

- To access the database?
- DML statements need to be executed from the host language. There are two ways to do this:
- By providing an application program interface (set of procedures) that can be used to send DML and DDL statements to the database and retrieve the results.
- The Open Database Connectivity (ODBC) standard for use with the C language is a commonly used application program interface standard. The Java Database Connectivity ODBC) standard provides corresponding features to the Java language.
- By extending the host language syntax to embed DML calls within the host language program. Usually a special character prefaces DML calls, and a preprocessor, called the DML precompiler, converts the DML statements to normal procedure calls in the host language.

1.9 DATABASE DESIGN

- Database systems are designed to manage large bodies of information. These large bodies of information do not exist in isolation. They are part of the operation of some enterprise whose end product may be information from the database or may be some device or service for which the database plays only a supporting role.
- Database design mainly involves the design of the database schema. The design of a complete database application environment that meets the needs of the enterprise being modelled requires attention to a broader set of issues.

1.9.1 Design Process:

- A high-level data model provides the database designer with a conceptual framework in which to specify the data requirements of the database users, and how the database will be structured to fulfill these requirements.
- The initial phase of database design, then, is to characterize fully the data needs of the prospective database users. The database designer needs to interact extensively with domain experts and users to carry out this task. The outcome of this phase is a specification of user requirements.
- The designer chooses a data model, and by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database. The schema developed at this conceptual-design phase provides a detailed overview of the enterprise.
- In a specification of functional requirements, users describe the kinds of operations (or transactions) that will be performed on the data. Example operations include modifying or updating data, searching for and retrieving specific data, and deleting data. At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements.
- In the logical-design phase, the designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used. The designer uses the resulting system-specific database schema in the subsequent physical-design phase, in which the physical features of the database are specified.

1.10 DATABASEARCHITECTURE

Database System Structure:

- The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs.

- Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines.
- Most users of a database system today are not present at the site of the database system, but connect to it through a network.
- This can be differentiated between client machines, on which remote database users work, and server machines, on which the database system runs
- Database applications are usually partitioned into two or three parts,
- In a two-tier architecture, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.
- In contrast, in a three-tier architecture, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, usually through a forms interface.
- The application server in turn communicates with a database system to access data. The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients.
- Three-tier applications are more appropriate for large applications, and for applications that run on the World Wide Web.

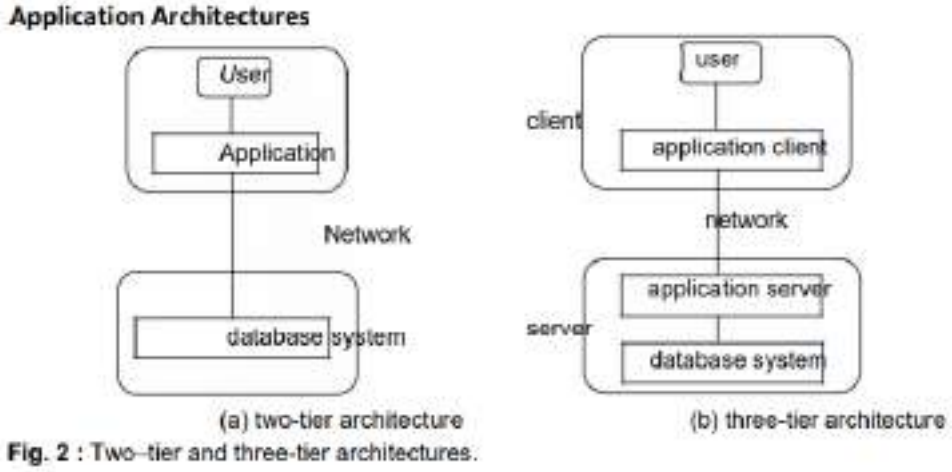


Fig. 2 : Two–tier and three-tier architectures.

1.11 TRANSACTION MANAGEMENT

1.11.1 Atomicity:

- Several operations on the database form a single logical unit of work.

- Consider an example of funds transfer, in which one department account(say A) is debited and another department account (say B) is credited.
- It is essential that either both the credit and debit occur, or that neither occur.
- This all-or-none requirement is called atomicity.

1.11.2 Consistency:

- In addition, it is essential that the execution of the funds transfer preserve the consistency of the database. That is, the value of the sum of the balances of A and B must be preserved. This correctness requirement is called consistency.

1.11.3 Durability:

- Finally, after the successful execution of a funds transfer, the new values of the balances of accounts A and B must persist, despite the possibility of system failure. This persistence requirement is called durability.
- A transaction is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any data base consistency constraints. That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates.

1.11.4 Recovery Manager:

- Ensuring the atomicity and durability properties is the responsibility of the database system itself specifically, of the recovery manager.

1.11.5 Failure recovery:

- The database must be restored to the state in which it was before the transaction in question started executing. The database system must therefore perform failure recovery, that is, detect system failures and restore the database to the state that existed prior to the occurrence of the failure.

1.11.6 Concurrency-control manager:

- When several transactions update the database concurrently, the consistency of data may no longer be preserved, even though each individual transaction is correct. It is the responsibility of the concurrency-control manager to control the interaction among the concurrent transactions, to ensure the consistency of the database.
- The transaction manager consists of the concurrency-control manager and the recovery manager

1.12 LET US SUM UP

- A database-management system (DBMS) consists of a collection of interrelated data and a collection of programs to access that data. The data describe one particular enterprise.
- The primary goal of a DBMS is to provide an environment that is both convenient and efficient for people to use in retrieving and storing information
- A data-manipulation language (DML) is a language that enables users to access or manipulate data. Nonprocedural DMLs, which require a user to specify only what data are needed, without specifying exactly how to get those data, are widely used today.
- A data-definition language (DDL) is a language for specifying the database schema and as well as other properties of the data
- Transaction management ensures that the database remains in a consistent (correct) state despite system failures. The transaction manager ensures that concurrent transaction executions proceed without conflicting.

1.13 LIST OF REFERENCE

A Silberschatz, H Korth, S Sudarshan, “Database System and Concepts”, fifth Edition McGraw- Hill

1.14 BIBLIOGRAPHY

- Database Systems ,RobCoronel,Cengage Learning.
- Programming with PL/SQL for Beginners, H. Dand, R. Patil and T. Sambare,X-Team.
- Introduction to Database System,C.J.Date,Pearson

1.15 UNIT END EXERCISE

1. Define database system and explain the purpose of database system in detail
2. Mention the views of database.
3. Explain DML and DDL in detail
4. Write a note on transaction management systems
5. Explain database architecture in detail.

*****

2

DATA MODELS

Unit Structure

2.0 OBJECTIVES

The objective of the chapter is as follow

- To get familiar with core data models in database management systems
- To understand the different types of database models
- To understand the concept of basic building blocks and business rules.

2.1 INTRODUCTION

- Data model gives an idea of how the final system or software will look after when the development is completed

- This concept is exactly like real world modelling in which before constructing any project (Buildings, Bridges, Towers) engineers create a model for it and gives the idea of how a project will look like after construction
- A data model is an overview of a software system which describes how data can be represented and accessed from software system after its complete implementation.

2.2 DATA MODELS

- Data models define data elements and relationships among various data elements for a specified system
- A data model is a way of finding tool for both business and IT professionals which uses a set of symbols and text to precisely explain a subset of real information to improve communication within the organisation and thereby lead to more flexible and stable application environment
- Data model is a simple abstraction of complex real world data gathering environment

2.2.1 Importance of Data Models:

- A data model is a set of concepts that can be used to describe the structure of data in a database.
- Data models are used to support the development of information systems by providing the definition and format of data to be involved in future system.
- Data model is acting like a guideline for development also gives an idea about possible alternatives to achieve targeted solution.
- A data model can sometimes be referred to as data structure especially in the context of programming languages.

2.2.2 Advantages of Data Models:

- Data model prevents the system from future risk and failure by defining structure of data in advance.
- As we got an idea of final system at the beginning of development itself so we can reduce the cost of project by proper planning and cost estimation as actual system is not yet developed.
- Data repetition and data type compatibility can be checked and removed with help of data model.
- We can improve Graphical User Interface (GUI) of system by making its model and get it approved by its future user so it will be simple for them to operate system and make entire system effective.

2.3 FILE MANAGEMENT SYSTEMS

- All data were permanently stored on a computer system, such as payroll and accounting records, was stored in individual files.
- A file management system, usually provided by the computer manufacturer as part of the computer's operating system, kept track of the names and locations of the files.
- The file management system basically had no data model; it knew nothing about the internal contents of files.
- To the file management system, a file containing a word processing document and a file containing payroll data appeared the same.
- Knowledge about the contents of a file which data is contained and how the data was organized was embedded in the application programs that used the file
- The problems of maintaining large file-based systems led in the late 1960s to the development of database management systems.
- The idea behind these systems was simple: take the definition of a file's content and structure out of the individual programs, and store it, together with the data, in a database.
- Using the information in the database, the DBMS that controlled it could take a much more active role in managing both the data and changes to the database structure.

2.3.1 Hierarchical Databases:

- This was developed by joined efforts of IBM and North American Rockwell known as Information management system.
- It was the first DBMS model
- The data is sorted hierarchically, either in top down or bottom up approach of designing.
- This model uses pointers to navigate between stored data.
- This model represents data as a hierarchical tree.
- Let us consider simple organizational structure as given below.
- CEO is root node having DU Heads below that managers who manages multiple project lead who organizes developer as shown below
- CEO -> Program manager (PM) -> Project Manager (PrM) -> Project Leader (PL) -> Team Leader (TL) -> Developer

Advantage:

Conceptual Simplicity:

Relationships between various levels is logically very simple. Hence database structure becomes easier to view.

Database Security:

Security is given by DBMS system itself it does not depends on whether programmer has given security or not.

Simple Creation, Updation and Access:

This model is simple to construct with help of pointers or similar concepts and very simple to understand also adding and deleting record is easy tin tree structure using pointers. This file system is faster and easy data retrieval through higher level records in tree structure.

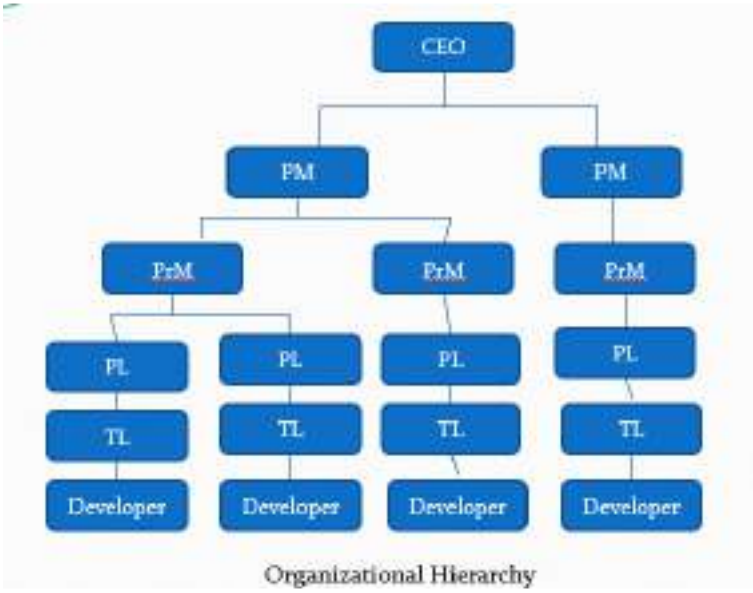
Disadvantages:

Complex implementation:

Only data independence is not enough for designer and programmers to build database system they need to have knowledge of physical data storage which may be complex.

Complex application programming:
Programmers must know how data is stored and path of data storage

Limitations in implementation:
1: N relationship can be implemented but implementing M:N relationship is difficult



Organizational Hierarchy

2.3.2 Network Databases:

- The simple structure of a hierarchical database became a disadvantage when the data had a more complex structure.
- To deal with applications such as order processing, a new network data model was developed. The network data model extended the hierarchical model by allowing a record to participate in multiple parent/child relationships
- Like the hierarchical model, this model also uses pointers toward data but there is no need of parent to child association so it does not necessarily use a downward tree structure. This model uses network databases
- This database model is similar to hierarchical model up to some aspect.
- A relationship between any two record types is called as a set.
- EXAMPLE – IDS (Integrated Data Store) one of the products based on network models developed by IBM and North American Rockwell

Advantages:

Simple design:
The network model is simple and easy to design and understand.

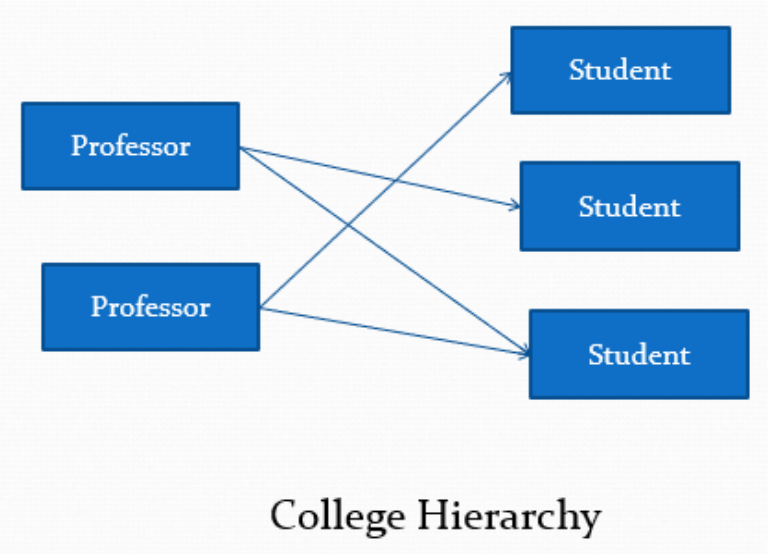
Ability to handle many types of relationship:
The network model can handle the one-t o-many or many-to-many or other relationships.
Hence network model manages multiuser user environment

Ease of data access:
In a network model an application can access a root (parent) record and all the member records within a set(child)

Disadvantages:

System complexity:
Data are accessed one record at a time hence the complexity for the system increases for accessing multiple records at a time.

Lack of structural independence:
Any changes made to the database structure (or data) requires the application programs to be modified before it can access data.



College Hierarchy

2.4 BASIC BUILDING BLOCK

The basic building block for any of model is Entities, Attributes , relationships and constraints.

2.4.1 Entity:

A fundamental component of a model. An entity is having its own independent existence in real world.

E.g.: A Student, Faculty, Subject having independent existence.
An entity may be an object with a physical existence, or it may have logical existence.

E.g.: Entities like Department, Section, adult(age>18) may have physical existence or it may have only logical existence.

2.4.2 Attributes:

Each entity has its own properties which describes that entity, such properties are called as attributes

A particular entity will have some value for each of its attributes

– Employee entity may be described by attributes name, age, phone etc.

2.4.3 Relationships:

It is an association among several entities for e. g. Employee works for Department.

The degree of the relationship is the number of participating entity types in a particular relation

Data model uses three types of relationships

2.4.4 Degree:

The degree of relationship type is number of participating entity types in a particular relation

2.5 TYPES OF RELATIONSHIPS

One is to one:

One entity is associated with at most one other entity.
E.g., One department can have only one manager

One is to Many:

One entity is associated with any number of entities in other entity.
E.g., One teacher may teach to many students

Many is to Many:
One entity is associated with any number of entities in other entity.
E.g., Books in library issued by students

2.6 BUSINESS RULES

- **Definition:** Business rules are statements of a discrete operational business policy or practice within specific organisations that constrains the business. It is intended to control or influence the behaviour of the business.
- Database designer needs to take help from concepts such as entity, attributes and relationships to build a data model, but the above things are not sufficient to describe a system completely.
- Business rules may define actors and prescribe how they should behave by setting constraints and help to manage business change in the system.

2.6.1 Characteristics of Business Rules:

Atomicity:
Rule should define any one aspect of the system environment.
E.g.: - College should have students in it.

Business format:
Rule should be expressed in business terms understandable tobusiness people.

E.g.: ER diagram, object diagram etc

Business ownership:
Each rule is governed by a businessperson who is responsible for verifying it, enforcing it, and monitoring need for change.
E.g.: End user or customer is responsible for requirements submitted by him.

Classification:
Each rule can be classified by its data and constraints.

Business Formalism:
Each rule can be implemented in the related information system. Business rules should be consistent and non-redundant.

EXAMPLES:
A student may take admission to college
One subject is taught by only one professor
A class consists of minimum 60 and maximum 80 students

2.6.2 Types of Business Rules:

DEFINITIONS:
Define some business terms. Definitions are incorporated in systems data dictionary.
E.g., A professor is someone who teaches to students

FACTS:
Connect business terms in ways that make business sense. Facts are implemented as relationships between various data entities
E.g., A professor may have student

CONSTRAINTS:
Shows how business rules and how business terms are connected with each other. Constraints usually state how many of one data entity can be related to another data entity
E.g., Each professor may teach up to four subjects.

DERIVATIONS:
Enable new knowledge or actions. Derivations are often implemented as formulas and triggers.
E.g. A student pending fees is his fees paid minus total fees.

2.7 DEGREES OF DATA ABSTRACTION

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. hide the complexity from users through

several levels of abstraction, to simplify users interactions with the system:

- **Physical level:** The lowest level of abstraction describes how the data are el data structures in detail.
- **Logical level:** the next higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple the user of the logical level does not need to be aware of this complexity. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.
- **View level:** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views
- for the same database.

Type customer = record
customer-id: string;
customer-name: string;
customer-street: string;
customer-city: string;
end;

This code defines a new record type called customer with four fields. Each field has a name and a type associated with it.

A banking enterprise may have several such record types, including account, with fields account number and balance employee, with fields employee name and salary

At the physical level, a customer, account, or employee record can be described as a block of consecutive storage locations (for example, words or bytes). The language compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest level storage details from database programmers.

Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of

these record types is defined as well. Programmers using a programming language work at this level of abstraction.

Similarly, database administrators usually work at this level of abstraction

Finally, at the view level, computer users see a set of application programs that hide details of the data types. Similarly, at the view level, several views of the database are defined, and database users see these views. In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database.

2.8 LET US SUM UP

- A database-management system (DBMS) consists of a collection of interrelated data and a collection of programs to access that data. The data describe one particular enterprise.
- The primary goal of a DBMS is to provide an environment that is both convenient and efficient for people to use in retrieving and storing information
- A data-manipulation language (DML) is a language that enables users to access or manipulate data. Nonprocedural DMLs, which require a user to specify only what data are needed, without specifying exactly how to get those data, are widely used today.
- A data-definition language (DDL) is a language for specifying the database schema and as well as other properties of the data
- Transaction management ensures that the database remains in a consistent (correct) state despite system failures. The transaction manager ensures that concurrent transaction executions proceed without conflicting.

2.9 LIST OF REFERENCE

- A Silberschatz, H Korth, S Sudarshan, “Database System and Concepts”, fifth Edition McGraw- Hill
- Introduction to Database System,C.J.Date,Pearson
- Database Systems ,Rob Coronel,Cengage Learning.

2.10 BIBLIOGRAPHY

- Programming with PL/SQL for Beginners, H. Dand, R. Patil and T. Sambare,X-Team.

*****

3

DATABASE DESIGN

Unit Structure

3.0 OBJECTIVES

The objective of the chapter is as follow

- To get familiar with the design of the database schema
- To understand the influence of design choice on database schema
- To understand the concept of ER Model

3.1 INTRODUCTION

- The task of creating a database application is a complex one, involving design of the database schema, design of the programs that access and update the data, and design of a security scheme to control access to data.
- The needs of the users play a central role in the design process
- The design of a complete database application environment that meets the needs of the enterprise being modelled requires attention to a broad set of issues.
- These additional aspects of the expected use of the database influence a variety of design choices at the physical, logical, and view levels.

3.2 ERRELATIONSHIP MODEL

- The entity-relationship (E-R) data model was developed to facilitate database
- Design by allowing specification of an enterprise schema that represents the overall
- Logical structure of a database.
- The E-R model is very useful in mapping the meanings and interactions of
- Real-world enterprises onto a conceptual schema
- The E-R data model
- Employs three basic concepts: entity sets, relationship sets, and attributes
- The E-R model also has an associated diagrammatic representation, the E-R diagram,

3.2.1 Entity:

An entity is a “thing” or “object” in the real world that is distinguishable from all other objects. For example, each person in an enterprise in an entity.

An entity has a set of properties, and the values for some set of properties may uniquely identity an entity. For instance, a person may have a person-id property whose value uniquely identifies that person

An entity may be concrete, such as a person or a book, or it may be abstract, such as a loan, or a holiday, or a concept.

3.2.2 Entity Set:

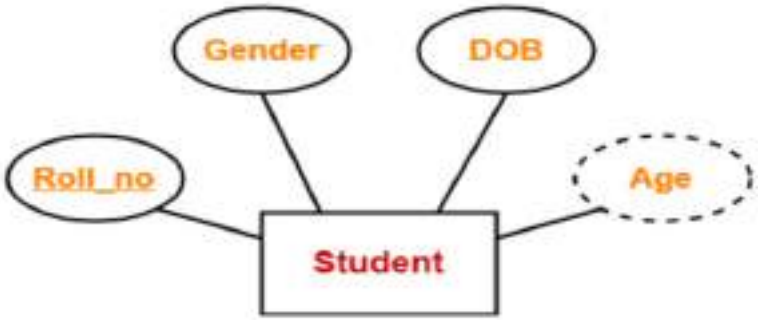
An entity set is a set of entities of the same type that share the same properties, or attributes. The set of all people who are instructors at a given university

3.2.3 Attributes:

An entity is represented by a set of attributes.

Attributes are descriptive properties possessed by each member of an entity set.

The designation of an attribute for an entity set expresses that the database stores similar information concerning each entity in the entity set; however, each entity may have its own value for each attribute. Each entity has a value for each of its attributes



3.2.4 Relationship:

A relationship is an association among several entities. For example, we can define a relationship advisor that associates instructor Katz with student Shankar. This relationship specifies that Katz is an advisor to student Shankar. A relationship set is a set of relationships of the same type.

(select customer-name
from borrower)

- The except operation automatically eliminates duplicates. Thus, in the preceding query, a tuple with customer name Jones will appear (exactly once) in the result only if Jones has an account at the bank but has no loan at the bank.
- If we want to retain all duplicates, we must write except all in place of except:

(select customer-name
from depositor)
except all
(select customer-name
from borrower)

- The number of duplicate copies of a tuple in the result is equal to the number of duplicate copies of the tuple in d minus the number of duplicate copies of the tuple in b, provided that the difference is positive.
- Thus, if Jones has three accounts and one loan at the bank, then there will be two tuples with the name Jones in the result. If, instead, this customer has two accounts and three loans at the bank, there will be no tuple with the name Jones in the result.

5.3 RENAMING & JOINS

- Unlike relations in the database, the results of relational-algebra expressions do not have a name that we can use to refer to them.
- It is useful to be able to give them names; the rename operator, denoted by the lowercase Greek letter rho (p), lets us do this, Given a relational algebra expression E the expression,

returns the result of expression E under the name x.

- A relation r by itself is considered a (trivial) relational algebra expression. Thus, we can also apply the rename operation to a relation r to get the same relation under a new name.
- A second form of the rename operation is as follows. Assume that a relational-algebra expression Ehas arity n.
- Then, the expression, $p_x(A_1, A_2, ….., A_n)$ (E) returns the result of expression E under the namex' and with the attributes renamed to A_1, A_2,….A_n.

- To illustrate renaming a relation, we consider the query "Find the largest account balance in the bank."
- Strategy is to,
  (1) compute first a temporary relation consisting of those balances that are not the largest and
  (2) take the set difference between the relation $\Pi_{balance}$ **(account)** and the temporary relation just computed, to obtain the result.

Step 1: To compute the temporary relation, we need to compare the values of all account balances. We do this comparison by computing the Cartesian product account ×account and forming a selection to compare the value of any two balances appearing in one tuple.

- First, we need to devise a mechanism to distinguish between the two balance attributes.
- We shall use the rename operation to rename one reference to the account relation; thus we can reference the relation twice without ambiguity.

Balance
500
400
700
750
350

- Result of the sub-expression would be,

$$\Pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho d\ (account)))$$

Balance
900

- Largest account balance in the bank.
- We can now write the temporary relation that consists of the balances that are not the largest:

$$\Pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho d(account)))$$

- This expression gives those balances in the account relation for which a larger balance appears some where in the account relation (renamed as d).
- The result contains all balances except the larges one.

- Step 2: The query to find the largest account balance in the bank can be written as:

$$\Pi_{balance}(account) - \Pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho d(account)))$$

- As one more example of the rename operation, consider the query "Find the names of all customers who live on the same street and in the same city as Smith."
- We can obtain Smith's street and city by writing,

$$\Pi_{customer-street,\ customer-city}(\sigma_{customer-name = \text{"Smith"}}(customer))$$

- However, to find other customers with this street and city, we must reference the customer relation a second time.
- In the following query, we use the rename operation on the preceding expression to give its result the name smith-addr, and to rename its attributes to street and city, instead of customer, street and customer-city:

$$\Pi_{customer.customer-name}(\sigma_{customer.customer-street = smith-addr.street \wedge customer.customer-city = smith-addr.city}(customer \times \rho_{smith-addr(street,city)}(\Pi_{customer-street,customer-city}(\sigma_{customer-name = \text{"Smith"}}(customer)))))$$

- The rename operation is not strictly required, since it is possible to use a positional notation for attributes.
- We can name attributes of a relation implicitly by using a positional notation, where \$1,\$2, … refer to the first attribute, the second attribute, and so on.
- The positional notation also applies to results of relational algebra operations.
- The result of above query looks like this,

Customer-name
Curry
Smith

5.4 DIVISION SYNTAX & SEMANTICS

- The division operation, denoted by ÷, is suited to queries that include the phrase "for all".
- Suppose that we wish to find all customers who have an account at all the branches located in Brooklyn. We can obtain all branches in Brooklyn by the expression,

$$r1 = \prod_{\text{branch-name}} (\sigma_{\text{branch-city} = \text{"Brooklyn"}} (\text{branch}))$$

- The result relation for this expression is shown below,

branch–name
Brighton
Downtown

- We can find all (customer-name, branch-name) pairs for which the customer has an account at abranch by writing,

$$r2 = \prod_{\text{customer-name, branch-name}} (\text{depositor} \bowtie \text{account})$$

- Now, we need to find customers who appear in r2 with every branch name in r1. The operation that
- provides exactly those customers is the divide operation. We formulate the query by writing,

$$\prod_{\text{customer-name, branch-name}} (\text{depositor} \bowtie \text{account}) \div \prod_{\text{branch-name}} (\sigma_{\text{branch-city} = \text{"Brooklyn"}} (\text{branch}))$$

- The result of this expression is a relation that has the schema (customer-name) and that contains the tuple (Johnson).
- Formally, let r(R) and s(S) be relations, and let $S \subseteq R$; that is, every attribute of schema S is also in schema R.
- The relation r÷s is a relation on schema R –S (that is, on the schema containing all attributes of schema R that are not is schema S). A tuple t is in r ÷ s if and only if both of two conditions hold:

  1. t is in ΠR–S(r)
  2. For every tuple ts in s, there is a tuple tr in r satisfying both of the following:

  (a) ts [S] = ts[S]

  (b) tr [R–S] = t

UNIT III

7

CONSTRAINT

Unit Structure

7.0 OBJECTIVES

- To understand Integrity, Domain constraint and Referential Integrity.
- To understand and apply primary key, unique key, Default, Foreign, Null and Check Integrity constraint.
- To understand and apply Assertions Triggers in SQL.
- To Understand and apply Security and Authorization in SQL.

7.1 INTRODUCTION

Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.

Example of integrity constrains are:

- An account balance cannot be null.
- No two students can have same roll no.

In general, an integrity constraint can be any arbitrary predicate pertaining to the database. However, arbitrary predicate can be costly to test. Hence, the most database systems allow one to specify integrity constraint that can be tested with minimal overhead.

7.2 DOMAIN CONSTRAINTS

- Domain constraints are the most elementary form of integrity constraint.
- They test values inserted in the database, and test queries to ensure that the comparisons make sense.
- New domain can be created from existing data types.

Example:

Create domain Dollars **numeric (12, 2)**

Create domain pounds **numeric (12, 2)**

- We cannot assign or compare a value of type Dollars to a value of type Pounds.

  However, we can convert type as below:

  (Cast r. A **as** Pounds)
- (Should also multiply by the dollar-to-pound conversion-rate)

Different types of constraints are:

7.3 REFERENTIAL INTEGRITY

A value that appears in one relation for given set of attributes also appears for a certain set of attributes in another relation. This called referential integrity.

Referential integrity in the E-R model:

```
Create table Deposit
(Branch_name char (15),
Acc_no char(10),
Cust_name char (20) not null,
Balance integer,
Primary key (Acc_no, Cust_name),
foreign key (Branch_name) references Branch,
foreign key (Cust_name) references Customer) ;
```

- **Cascading Actions in SQL**

```
create table Account
foreign key (Branch-name) references Branch
on delete cascade
                on update cascade
. . .)
```

- Due to the **on delete cascade** clause, if a delete of a tuple in Branch results in referential-integrity constraint violation, the delete_casades_ to the Account relation, deleting the tuple that refers to the branch that was deleted.
- Cascading updates are similar.
- If there is a chain of foreign-key dependencies across multiple relations, with **on delete cascade** specified for each dependency, a deletion or update at one end of the chain can propagate across the entire chain.
- If a cascading update to delete causes a constraint violation that caanot be handled by a further cascading operation, the system aborts the transaction. As a result, all the changes caused by the transaction and its cascading actions are undone.
- Referential integrity is only checked at the end of a transaction. Intermediate steps are allowed to violate referential integrity provided later steps remove the violation. Otherwise it would be impossible to create some database states.

Example:

Insert two tuples whose foreign keys point to each other:

```
# Example: spouse attribute of relation
Marriedperson (Name, Address, Spouse)
```

```
Alternative to cascading:
" on delete set null
                " on delete set default
```

```
        create assertion balance-constraint check
      (not exists (
select *  from loan
where not exists(
select*
from borrower, depositor, account
where loan.loan-number = borrower.loan-number
and borrower.customer-name = depositor. customer-name
and depositor.account-number = account.account-number
and account.balance > = 1000)))
```

7.6 TRIGGER

A trigger is a statement that the system executes automatically as a side effect of a modification to the database. To design a trigger mechanism, we must meet two requirements:

1. Specify when a trigger is to be executed. This is broken up into event that causes the trigger to be checked and as condition that must be satisfied for trigger execution to proceed.
2. Specify the actions to be taken when the trigger executes.

The above model of triggers is referred to as the **event-condition-action** model for trigger.

The database stores triggers just as if they were regular data, so that they are persistent and are accessible to all database operations. Once we enter a trigger into the database, the database system takes on the responsibility of executing it whenever the specified event occurs and the corresponding condition is satisfied.

7.6.1 Need for Triggers:

Triggers are useful mechanism for alerting humans for starting certain tasks automatically when certain conditions are met. For example, suppose that, instead of allowing negative account balances, the bank deals with overdrafts by setting the account balance to zero and creating a loan in the amount of the overdraft. The bank gives this loan a loan number identical to the account number of the overdraft. The bank gives this loan a loan number identical to the account number of the overdrawn account. For this example, the condition for executing the trigger is an update to the account relation that results in a negative balance value. Suppose that Jones withdrawal of some money from an account made the account balance negative. Let t denote the account tuple with a negative balance value. The actions to be taken are:

- Insert a new tuple s in the loan relation with

```
s[loan_no] = t [account_not]
```

```
Grant select on Student to Shyam, Om
```

2. Revoke command:

The revoke command is used to cancel the authorization. The form of revoke statement is :

```
Revoke < privilege list > on <relation name or view name > from < user / role list >
```

To revoke the above privileges write following statement,

```
Revoke select on Student from Shyam , Om
```

Limitation of SQL Authorization:

- SQL does not support authorization at a tuple level.
  - Example: We cannot restrict students to see only (the tuples storing) their own grades.
- With the growth in Web access to databases, database accesses come primarily from application servers.
  - End users don't have database user ids, they are all mapped to the same database user id.
- All end-users of an application (such as a web application) may be mapped to a single database user.
- The task of authorization in above cases falls on the application program, with no support from SQL :
  - Benefit is: Fine-granted authorizations, such as to individual tuples, can be implemented by the application.
  - Drawback is: Authorization is required to be done in application code. Checking for absence of authorization loopholes becomes very difficult since it requires reading large amounts of application code.

7.8 IMPORTANT QUESTIONS AND ANSWERS

Q1) Explain the integrity constraints: Not Null, Unique, Primary Key with an example each. Is the combination 'Not Null, primary Key' a valid combination. Justify.

Answer:

- **Not Null:** Should contain valid values and cannot ne NULL.

Q2) Explain referential integrity constraints with example.

Q3) What is need of trigger ? Explain when not to use trigger.

Q4) What is assertion ? Explain with example.

Q5) Write short note on :

a. Assertion b. Trigger

Q6) Write short note on security mechanism in database.

Q7) What do you mean by authorization and authentication in DBMS? Explain how it is implemented in SQL with suitable example.

Q8) Discuss different security and authorization mechanism in database.

Q9) What is meant by authorization in DBMS? What is granting of privileges? Explain security mechanisms in database.

REFERENCES

1. Peter Rob and Carlos Coronel, ― Database Systems Design, Implementation and

   Management‖, Thomson Learning, 9thEdition.

2. G. K. Gupta : "Database Management Systems", McGraw – Hill.

Text Books:

1. Korth, Slberchatz, Sudarshan, Database System Concepts, 6th Edition, McGraw Hill
2. Elmasri and Navathe, Fundamentals of Database Systems, 6thEdition, Pearson education
3. Raghu Ramkrishnan and Johannes Gehrke, Database Management Systems, TMH

*****

8

STRUCTURED QUERY LANGUAGE PART-I

Unit Structure

8.0 OBJECTIVE

- To understand characteristics of SQL, Literals, Operators.
- To understand and apply different types of SQL commands to real world problems.
- To Understand Data Definition Commands and solve real world problems.

8.1 INTRODUCTION

Structured Query Language (SQL) is the standard command set used to communicate with the relational database management systems. All tasks related to relational data management creating tables, querying the database for information, modifying the data in the database, deleting them, granting access to users and so on can be done using SQL.

INTERSECT	Returns only rows that match both queries
MINUS	Returns only row that do not match both queries

Fig. 8.5 Operator precedence

8.5 DATA DEFINITION COMMANDS

The standard query language for relational database is SQL (Structured Query Language). It is standardized and accepted by ANSI (American National Standards Institute). SQL, is a fourth-generation high-level nonprocedural language, using a nonprocedural language query, a user requests data from the DBMS. The SQL language uses English – like commands such as CREATE, INSERT, DELETE, UPDATE, and DROP. The SQL language is standardized and its syntax is same across most DBMS packages.

➢ **Different types of SQL commands are:**

- Data retrieval retrives data from the database, for example SELECT.
- Data manipulation Language (DML) inserts new rows, changes existing rows, and removes unwanted rows, for example INSERT, UPDATE, and DELETE.
- Data Definition Language (DDL) creates, changes, and removes a table structure, for example, CREATE, ALTER, DROP, RENAME, and TRUNCTATE.
- Transaction Control manages and changes logical transactions. Transactions are changes made to the data by DML statements grouped together, for example, COMMIT, SAVE POINT, and ROLLBACK.
- Data control Language (DCL) gives and removes rights to database objects, for example GRANT, and REVOKE.

➢ **SQL DDL**

SQL DDL is used to define relational database of a system. The general syntax of SQL sentence is:

VERB (parameter 1, parameter 2,…., parameter n);

In above syntax, parameters are separated by commas and the end of the verb is indicated by a semicolon. The relations are created using CREATE verb.

The different DDL commands are as follows:

- CREATE TABLE
- CREATE TABLE …AS SELECT
- ALTER TABLE …….ADD
- ALTER TABLE ……MODEIFY
- DROP TABLE

1) Create Table:

This command is used to create a new relation and the corresponding syntax is:

```
CREATE TABLE relation _name
(field 1 data type (size), field 2 data type (size)….., fieldn data type (size);
```

➢ **Example:**

The modern Book House mostly supplies books to institutions which frequently buy books from them. Various relations used are Customer, Sales Book Author, and Publisher. Design database scheme for the same.

1) The customer table definition is as follows:

```
SQL > create table Customer
            (Cust_no varchar (4) primary key,
             Cust_ name varchar (25), Cust_add varchar(30),
             Cust_ ph varchar(15);
```

Table created.

2) The sales table definition is as follows:

```
SQL > create table Sales
          (Cust_n0 varchar (4) , ISBN varchar (15),
           Qty number (3),
           Primary key (Cust_no, ISBN);
```

Table created.

3) The book table definition is as follows:

```
SQL > create table book
          (ISBN varchar (15) primary key,
          Title varchar (25), Pub_year varchar (4),
           Unit_price number (4),
           Author_name varchar (25);
```

Table created.

4) The author table definition is as follows:

```
SQL > create table Author
        (Author_name varchar (25) primary key,
        Country varchar (15);
```

2) CREATE TABLE …. AS SELECT

This type of create command is used to create the structure of new table from the structure of existing table.

The generalized syntax of this form is shown in below:

```
CREATE TABLE relation_name 1
( field1, field2,….,fieldn)
AS SELECT field1, field2,…..fieldn
FROM relation_name2;
```

Example:
Create the structure for special customer from the structure of Customer table.

The required command is shown below.

```
SQL > create table Special_customer
      (Cust_no, Cust_name, Cust_add )
      As select Cust_no, Cust_name, Cust_add
      From Customer;
```

Table created.

3) ALTER TABLE …..ADD……..

This is used to add some extra columns into existing table. The generalized format is given below.

```
ALTER TABLE relation_name
ADD (new field1 datatype (size),
New field 2 datatype (size) ,…..,
New field n datatype (size)) ;
```

Example:
ADD customer phone number and fax number in the customer relation.

```
SQL > ALTER TABLE Customer
      ADD (Cus_ph_no varchar (15),
      Cust_fax_no varchar (15);
```

Table created.

4) ALTER TABLE …… MODIFY:

This form is used to change the width as well as data type of existing relations. The generalized syntax of this shown below.

```
ALTER TABLE relation_name
MODIFY (field1 new data type (size),
        field2 new data type (size),
         ----------
        fieldn new data type (size);
```

Example:
Modify the data type of the publication year as numeric data type.

```
SQL > ALTER TABLE Book
      MODIFY (Pub-year number (4);
```

Table created.

Restrictions of the Alter Table:

Using the alter table clause you cannot perform the following tasks:

- Change the name of the table
- Change the name of the column
- Drop a column
- Decrease the size of a column if table data exists.

5) DROP TABLE:

This command is used to delete a table. The generalized syntax of this form is given below:

DROP TABLE relation-name

Example: Write the command for deleting special-customer relation.

SQL > DROP TABLE Special_customer

Table dropped**.**

6) Renaming a Table:

You can rename a table provided you are the owner of the table. The general syntax is:

RENAME old table name TO new table name;

Example:

SQL > RENAME Test To Test_info;

Table renamed.

7) Truncating a Table:

Truncating a table is removing all records from the table. The structure of the table stays intact. The SQL language has a DELETE statement which can be used to remove one or more (or all) rows from a table. Truncation releases storage space occupied by the table, but deletion does not. The syntax is :

TRUNCATE TABLE table name;

Example:

SQL > TRUNCATE TABLE Student;

8) Indexes:

Indexes are optional structures associated with tables. We can create indexes explicitly to speed up SQL statement execution on a table.

Example: Find all customers who have an account but no loan at the bank.

```
SQL > select Customer_name
      from Depositor
      minus
     select Customer_name
     from Borrower;
```

Output:

Customer_name
Ram
Sita

8.7 IMPORTANT QUESTIONS AND ANSWERS

Q.1 Consider the following relational schemas:

EMPLOYEE_NAME STREET, CITY)
WORKS (EMPLOYEE_NAME , COMPANYNAME, SALARY)
COMPANY (COMPANY_NAME, CITY)
Specify the table definitions in SQL.

Ans. ;

1) CREATE TABLE EMPLOYEE
(EMPLOYEE_NAME VARCHAR2(20) PRIMARY KEY,
STREET VARCHAR2(20),
CITY VARCHAR2(15);

2) CREATE TABLE COMPANY
(COMPANY_NAMEVARCHAR2(50) PRIMARY KEY,
CITY VARCHAR2(15);

3) CREATE TABLE WORK
(EMPLOYEE_NAME VARCHAR2(20)
REFERENCES EMPLOYEE EMPLOYEE_NAME,
(COMPANY_NAMEVARCHAR2(50)
REFERENCES COMPANY COMPANY_NAME,
SALARY NUMBER(6),
CONSTRAINT WORK_PK PRIMARY KEY (EMPLOYEE_NAME,
COMPANY_NAME));

Q.2 Consider the relations defined below:

PHYSICIAN (regno, name, telno, city)

PATIENT (pname, street, city)

VISIT (pname, regno, date_of_visit, fee)

Where the regno and pname identify the physician and the patient uniquely
Respectively. Express queries (i) to (iii) in SQL.

i) Get the name and regno of physicians who are in Mumbai.

ii) Find the name and city of patient(s) who visited a physician on 01 August 2012.

iii) Get the name of the physician and the total number of patients who have visited her.

iv) What does the following SQL query answer

```
SELECT DISTINCT name
FROM PHYSICAN P
WHERE NOT EXISTS
(SELECT *
FROM VISIT
WHERE regno = p.regno)
```

Ans.:

i) Select name, regno from PHYSICIAN where city = 'Mumbai',

ii) Select pname, city from PATIENT, VISIT where PATIENT.pname = VISIT .pname and date_of_visit = '01-Aug-12';

iii) select name, count(*) from PHSICAN, VISIT

where PHYSICIAN. Regno = VISIT.regno

group by Physician. Regno;

iv) This will give the name of physicians who have not visited any patient.

Q.3 Consider the following relations:

BRANCH (bno, street, area, city, pcode, Tel_no, Fax_no)

STAFF(Sno, Fname, Lname, address, position, salary, bno)

Express the following queries in SQL :

i) List the staff who work in the branch at 'Main Bazar'

ii) Find staff whose salary is larger than the salary of every member of staff at branch B1.

Ans.:

i) Select Fname, Lname
from STAFF, BRANCH

ii) Select Fname, Lname
from STAFF
where salary > (select max (salary) from Staff where bno ='B1');

Q.4 Consider the relations given below

Borrower (id_no, name)

Book (accno, title, author borrower_idno)

a) Define the above relations as tables in SQL making real world assumptions about the type of the fields. Define the primary keys and the foreign keys.

b) For the above relations answer the following queries in SQL

i. What are the titles of the books borrowed by the borrower whose id-no in 123.

ii. Find the numbers and names of borrowers who have borrowed books on DBMS in ascending order in id_no.

iii. List the names of borrowers who have borrowed at least two books.

Ans.:

a)

1) Create table Book
   (Accno int Primary Key,
   Title char(30), author char(30),
   Borrow-idno int references Borrower_id.no);
2) Create table borrower
   (id-no int Primary Key,
   Name char(30));

b)

i) Select title from Book
   where borrower.id=123

ii) Select id_no, name
   from borrower, Book
   where Borrower.id_no = Book.borrower_idno
   and title =’DBMS’
   order by id_no asc;

iii) Select name
   from Borrower, Book
   where Borrower.id_no = book.Borrower_id_no
   having count (*) > 2;

Q.5 Describe substring comparison in SQL. For the relation Person (name, address), write a SQL query which retrieves the names of people whose name begins with ‘A’ and address contains ‘Pune’.

Ans.:

i) **SUBSTR** is used to extract a set of charters from a string by specifying the character starting position and end position and length of characters to be fetched.

Example:

Substr(hello’, 2,3);
will return ‘ell.

ii) Select name
   from Person
   where name like ‘A%’and address ‘Pune’;

Q.6 Consider the following relations:

S(S#, SNAME, STATUS, CITY)
SP(S#, P#, QTY)
P(P#, PNAME, COLOR, WEIGHT, CITY)

Give an expression in SQL for each of queries below:

i) Get supplier names for supplier who supply at least one red part

ii) Get supplier names for supplier who do not supply part P2.

Ans.:

i)

```
SELECT SNAME FROM S
WHERE S# IN (SELECT S# FROM SP
WHERE P# IN (SELECT P# FROM P
WHERE COLOR = RED'));
```

ii)

```
SELECT SNAME FROM S
WHERE S# NOT IN (SELECT S# FROM SP WHERE P# = 'P2')
```

8.8 SUMMARY

All tasks related to relational data management creating tables, querying the database for information, modifying the data in the database, deleting them, granting access to users and so on can be done using SQL.SQL specifies what is required and not how it should be done.

There are four kinds of literal values supported in SQL. They are Character string, Bit string, Exact numeric and Approximate numeric. Data definition language is used to create, alter and delete database objects.

Arithmetic operators are used in SQL expressions to add, subtract, multiply, divide and negate data values. The result of this expression is a number value.

A logical operator is used to produce a single result from combining the two separate conditions. Set operators combine the results of two separate queries into a single result. Precedence defines the order that the DBMS uses when evaluating the different operators in the same expression.

Data retrieval retrives data from the database, for example SELECT.

Data Definition Language (DDL) creates, changes, and removes a table structure, for example, CREATE, ALTER, DROP, RENAME, and TRUNCTATE.

ALTER TABLE …..ADD……..is used to add some extra columns into existing table. **ALTER TABLE …… MODIFY** is used to change the width as well as data type of existing relations **.DROP TABLE** command is used to delete a table. Truncating a table is removing

```
        from Book;
View created.
SQL > select * from V_Book;
```

Output:

TITLE	AUTHOR_NAME
Oracle	Arora
DBMS	Basu
DOS	SInha
ADBMS	Basu
Unix	Kapoor

10.2.2 Selecting Data from a View:

Example: Display all the titles of books written by author 'Basu'.

```
SQL > select Title
      from V_Book
      Where Author_name = 'Basu';
```

Output:

TITLE
DBMS
ADBMS

10.2.3 Updatable Views:

Views can also be used for data manipulation i.e the user can perform Insert, Update, and the Delete operations on the view. The views on which data manipulation can be done are called Updatable Views

Views that do not allow data manipulation are called Ready only Views. When you give a view name in the Update, Insert, or Delete statement, the modifications to the data will be passed to the underlying table.

For the view to be updatable, it should meet following criteria:

- The view must be created on a single table.
- The primary key column of the table should be included in the view.
- Aggregate functions cannot be used in the select statement.
- The select statement used for creating a view should not include Distinct, Group by, or Having clause.
- The select statement used creating a view should not include subqueries.
- It must not use constants, strings or value expressions like total / 5.

10.2.4 Destroying a View:

A view can be dropped by using the DROP VIEW command.

Syntax:

```
DROP VIEW viewname;
```

Example:

```
DROP VIEW V_Book;
```

10.3 NESTED AND COMPLEX QUERIES

10.3.1 Nested Queries:

SQL provides a mechanism for nesting subqueries. A subquery is a select from where expression that is nested within another query. Mostly subqueries are used to perform tests for set membership to make set comparisons and determine set cardinality.

10.3.1.1 Set Memberships:

SQL uses in and not in constructs for set membership tests.

i) IN:

The in connective tests for set membership , where the set is a collection of values produced by a **select** clause.

Examples:

1) Display the title, author, and publisher name of all books published in 2000, 2002 and 2004.

```
SQL > select Title, Author_name, Publisher_name, Pub_year
      from Book
      where Pub_year in ('2000', '2004') ;
```

Output:

TITLE	AUTHOR _NAME	PUBLISHER _NAME	PUB_YEAR
Oracle	Arora	PHI	2004
DBMS	Basu	Technical	2004
ADBMS	Basu	Technical	2004
Unix	Kapoor	SciTech	2000

2) Get the details of author who have published book in year 2004.

```
SQL> select * from Author
     where Author_name in (select Author_name
     from Book
     where Pub_year = '2004');
```

Output:

AUTHOR_NAME	Country
Arora	U.S
Basu	India

ii) Not IN:

The **not in** connective tests for the absence of set membership.

Examples

1) Display title, author, and publisher name of all books except those which are published in year 2002, 2004 and 2005.

```
SQL > select Title, Author_name, Publisher_name, Pub_year
        from Book
        where Pub_year not in ('2002', '2004', '2005')
```

Output:

TITLE	AUTHOR_NAME	PUBLISHER_NAME	PUB_YEAR
DOS	SInha	Nirali	2003
Unix	Kapoor	SciTech	2000

2) Get the titles of all books written by authors not living in India.

```
SQL > select Title
        from Book
        where Author_name not in (select Author_name from
from author
                                  where country = 'India');
```

Output:

TITLE
Oracle
Unix

10.3.1.2 Set Comparison:

Nested subqueries are used to compare sets. SQL uses various comparison operators such as <, < = , > , > = , =, <>, any, all, and some, etc to compare sets.

Examples:

1) Display the titles of books that have price greater than at least one book published in year 2004.

For given example, we write the SQL query as:

```
SQL > select distinct B1. Title
```

```
from Book B1, Book B2
where B1. Unit_price > B2. Unit_price
and B2. Pub_year = '2004';
```

Output:

TITLE
ADBMS
DBMS

SQL offers alternative style for writing preceding query.

The phrase 'greater than at least one' is represented in SQL by > some.

Using > some, we can rewrite the query as:

```
SQL > select distinct Title
                    from Book
      where Unit_price > some (select Unit_price
                               from Book
                               where Pub_year = '2004');
```

Output:

TITLE
ADBMS
DBMS

The subquery generates the set of all unit price values of books published in year 2004. The **> some** comparison in the **where** clause of the outer select is true if the unit price value of the tuple is greater than at least one member of the set of all unit price values of books published in year 2004.

- SQL allows **< some, >some, <=some, >=some, =some, and <> some** comparisons.
- = **some** is identical to **in** and **<> some** is identical to **not in.**
- The keyword **any** is similar to **some.**

All

- The ">**all"** corresponds to the phrase 'greater than all'.
- SQL allows <all, <=all, >all, <>all, =all comparisons.
- all is identical to **not in** construct.

1) Display the titles of books that have price greater than all the books publisher in year 2004.

```
SQL > select distinct title
from Book
where Unit_price > all (select Unit_price
```

iii) select E. empname
from Empolyee E, Employee T, Managers M
where E.empname = M. empname and
e.city = T. city and T. empname = M. manager – name;

Q.3 Consider the employee database where the primary keys are underlined. Give an Expression in SQL for the following gueries:

Employee (employee-name, street, city)

Works (employee-name, company-name, salary)

Company (company-name, city)

manages (employee-name, manager-name)

i) Find all employees in the database who earn more than each employee of small Bank Corporation.

ii) Find all employees in the database who do not work for first Bank Corporation.

iii) Find all employees who earn more that the average salary of all employees of their company.

iv) Find the names of all employees who work for First Bank Corporation.

Ans.:

i) select employee name
from works
where salary > all
(select salary
from works
where company name = 'small Bank Corporation');

ii) select employee name
from works
where company name <> 'First bank Corporation';

iii) create view Avg-salary (salary) as
select avg (Salary)
from works;
select employee-name
from works
where salary > Avg-salary. Salary ;

iv) select employee name
from works
where company name = 'First Bank Corporation';

Q.4 For the following given database, write SQL queries:-
Person (driver_id #, name, address)
Car (license, model, year)
accident (report_no, date, location)

owns (driver_id #, license)
participated (drive cid, car, report_number, damage_amount)

i) Find the total number of people who of people who owned cars that involved in an accident in 2007.

ii) Find the number of accidents in which the cars belonging to “Ajay”. Were involved.

iii) Find the number of accidents that were reported in Mumbai region in the year 2004.

Ans.:

i) select count (distinct name)
from accident, participated, person
where accident. report number = participated.report number
and participated. Driver id = person.driver id
and date between ’01-01-2007’ and ’31-12-2007’;

ii) select count (distinct report_number)
from accident natural join participated natural join person
where name = ‘Ajay’;

iii) select count (distinct report_number)
from accident
where location = ‘Mumbai’ and date between ’01-01-2004’ and ’31-12-2004’;

Q.5 For the following given database, write SQL queries:-
person (driver_id#, name, address)
car (license, model, year)
accident (report_no, date, location)
owns (driver_id #, license)
participated (driver_id, car, report_number, damage_amount)

i) Find the total number of people who owned cars that were involved in accident in 1995.

ii) Find the number of accidents in which the cars belonging to Sunil K”, were involved.

iii) Update the damage amount for car with licence number “Mum2022” in the accident with report number “AR2197” to rs5000.

Ans.:

i) select count (distinct name)
from accident, participated, person
where accident.report number = participated. Report number
and participated. Driver id = person. Driver id
and date between ’01-01-1995’ and ’31-12-1995’;

```
p := 4 * p;

DBMS_OUTPUT.PUT_LINE( ‘pi is approximately : ‘ || p ); — print
result
END;
```
Sequential Control: GOTO and NULL Statements

The GOTO statement is seldom needed. Occasionally, it can simplify logic enough to warrant its use. The NULL statement can improve readability by making the meaning and action of conditional statements clear.

Overuse of GOTO statements can result in code that is hard to understand and maintain. Use GOTO statements sparingly. For example, to branch from a deeply nested structure to an error-handling routine, raise an exception rather than use a GOTO statement.

- **Using the GOTO Statement:**

The GOTO statement branches to a label unconditionally. The label must be unique within its scope and must precede an executable statement or a PL/SQL block. When executed, the GOTO statement transfers control to the labeled statement or block. The labeled statement or block can be down or up in the sequence of statements.

Example : Using a Simple GOTO Statement

```
DECLARE
p VARCHAR2(30);
n PLS_INTEGER := 37; — test any integer > 2 for prime
BEGIN
FOR j in 2..ROUND(SQRT(n)) LOOP
IF n MOD j = 0 THEN -- test for prime
p := ‘ is not a prime number’; — not a prime number
GOTO print_now;
END IF;
END LOOP;
p := ‘ is a prime number’;
<<print_now>>
DBMS_OUTPUT.PUT_LINE(TO_CHAR(n) || p);
END;
/
```

- **Using the NULL Statement:**

The NULL statement does nothing, and passes control to the next statement. Some languages refer to such an instruction as a no-op (no operation).**Example: Using the NULL Statement to Show No Action**

```
DECLARE
v_job_id VARCHAR2(10);
v_emp_id NUMBER(6) := 110;
BEGIN
SELECT job_id INTO v_job_id FROM employees WHERE employee_id
= v_emp_id;
IF v_job_id = 'SA_REP' THEN
UPDATE employees SET commission_pct = commission_pct * 1.2;
ELSE
NULL; — do nothing if not a sales representative
END IF;
END;
```

12.7 SUMMARY

This chapter surveys the main features of PL/SQL and points out the advantages they offer. It also acquaints you with the basic concepts behind PL/SQL and the general appearance of PL/SQL programs. You see how PL/SQL bridges the gap between database technology and procedural programming languages.

12.8 REFERENCES

1. Database System and Concepts A Silberschatz, H Korth, S Sudarshan McGraw-Hill Fifth Edition
2. "Fundamentals of Database Systems" by Elmsari, Navathe, 5th Edition, Pearson Education (2008).
3. "Database Management Systems" by Raghu Ramakrishnan, Johannes Gehrke, McGraw Hill Publication.
4. "Database Systems, Concepts, Design and Applications" by S.K.Singh, Pearson Education.

12.9 UNIT END QUESTIONS

MCQs:

1. Which of the following is not a section in PLSQL Block?
   a. Endif
   b. Begin

c. Declare
d. Exception

2. _____blocks are PL/SQL blocks which do not have any names assigned to them.
   a. Consistent
   b. Named
   c. Anonymous
   d. Begin

3. _____blocks are PLSQL blocks have a specific or unique name assigned to them.
   a. Consistent
   b. Named
   c. Anonymous
   d. Begin

4. Named PLSQL block always start with _____ Keyword
   a. Declare
   b. Begin
   c. Create
   d. End

5. Words used in a PL/SQL block are called _____
   a. Numerals
   b. Symbols
   c. Compound Units
   d. Lexical Units

6. Assignment Operator in PLSQL block
   a. (:!=)
   b. (:=)
   c. (:==)
   d. (:&&)

7. Which of the following is not an executable statements supported by PLSQL?
   a. Insert
   b. Grant
   c. Select
   d. Update

8. _____uses a selector which is an expression whose value is used to return one of the several alternatives.
   a. IF Else
   b. Searched Case
   c. Loop
   d. CASE

9. Which of the following loop is not supported by PLSQL?
   a. While
   b. Do While
   c. For
   d. Loop

10. Which of the following is syntactically correct for declaring and assigning value e to a variable in PLSQL Block?
    a. a int=5;
    b. b =: int 7;
    c. c int :=10;
    d. d int=12:;

Answer the following:

1. Explain advantages of PL/SQL
2. Explain PL/SQL block structure.
3. Explain scalar data types
4. Explain the following:

   i)%Type

   ii) % Rowtype

   iii) Sequences in PL/SQL

   iv) Bind Variables
5. Explain various data types conversion functions with examples.
6. Write a PL/SQL block to demonstrate cube of an input number
7. Write a PL/SQL program to demonstrate the use of basic arithmetic operators on the numbers input by user.
8. Write a PL/SQL program to find out square of inputed number by the user.
9. Explain various looping/iterative constructs in PL/SQL. 9. 10. Explain various conditional statements in PL/SQL.

*****

Collection Type	Number of Elements	Subscript Type	Dense or Sparse	Where Created	Can Be Object Type Attribute
Associative array (or index-by table)	Unbounded	String or integer	Either	Only in PL/SQL block	No
Nested table	Unbounded	Integer	Starts dense, can become sparse	Either in PL/SQL block or at schema level	Yes
Variablesize array (Varray)	Bounded	Integer	Always dense	Either in PL/SQL block or at schema level	Yes

We have already discussed varray in the chapter **'PL/SQL arrays'**. In this chapter, we will discuss the PL/SQL tables.

Both types of PL/SQL tables, i.e., the index-by tables and the nested tables have the same structure and their rows are accessed using the subscript notation. However, these two types of tables differ in one aspect; the nested tables can be stored in a database column and the index-by tables cannot.

Index-ByTable:

An **index-by** table (also called an **associative array**) is a set of **key-value**pairs. Each key is unique and is used to locate the corresponding value. The key can be either an integer or a string.

An index-by table is created using the following syntax. Here, we are creating an **index-by** table named **table_name**, the keys of which will be of the subscript_type and associated values will be of the *element_type*

TYPE type_name IS TABLE OF element_type [NOT NULL] INDEX BY subscript_type;

table_name type_name;

Example:
Following example shows how to create a table to store integer values along with names and later it prints the same list of names.

```
DECLARE
      TYPE salary IS TABLE OF NUMBER INDEX BY
      VARCHAR2(20);
      salary_list salary;
      nam e VARCHAR2(20);
BEGIN
```

```
— adding elements to the table
salary_list('Rajnish') := 62000;
salary_list('Minakshi') := 75000;
salary_list('Martin') := 100000;
salary_list('James') := 78000;

— printing the table
name := salary_list.FIRST;
WHILE name IS NOT null LOOP
      dbms_output.put_line
      ('Salary of ' || name || ' is ' || TO_CHAR(salary_list(name)));
      name := salary_list.NEXT(name);
END LOOP;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result:

```
Salary of James is 78000
Salary of Martin is 100000
Salary of Minakshi is 75000
Salary of Rajnish is 62000

PL/SQL procedure successfully completed
```

Example:

Elements of an index-by table could also be a **%ROWTYPE** of any database table or **%TYPE** of any database table field. The following example illustrates the concept. We will use the **CUSTOMERS** table stored in our database as:

```
Select * from customers
+----+---------+---+------------+-----------+

| 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   | 4500.00 |
+----+------------+---------------+-------------+-------------+
```

```
DECLARE
CURSOR c_customers is
select name from customers;

TYPE c_list IS TABLE of customers.Name%type INDEX BY
binary_integer;
    name_list c_list;
    counter integer :=0;
BEGIN
    FOR n IN c_customers LOOP
        counter := counter +1;
        name_list(counter) := n.name;
        dbms_output.put_line('Customer('||counter||'):'||name_lis
        t(counter));
    END LOOP;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result:

```
Customer(1): Ramesh
Customer(2): Khilan
Customer(3): kaushik
Customer(4): Chaitali
Customer(5): Hardik
Customer(6): Komal

PL/SQL procedure successfully completed
```

NestedTables:

A **nested table** is like a one-dimensional array with an arbitrary number of elements. However, a nested table differs from an array in the following aspects:

- An array has a declared number of elements, but a nested table does not. The size of a nested table can increase dynamically.
- An array is always dense, i.e., it always has consecutive subscripts. A nested array is dense initially, but it can become sparse when elements are deleted from it.

A nested table is created using the following syntax “

TYPE type_name IS TABLE OF element_type [NOT NULL];

table_name type_name;

This declaration is similar to the declaration of an **index-by** table, but there is no **INDEX BY** clause.

A nested table can be stored in a database column. It can further be used for simplifying SQL operations where you join a single-column table with a larger table. An associative array cannot be stored in the database.

Example:

The following examples illustrate the use of nested table:

```
DECLARE
   TYPE names_table IS TABLE OF VARCHAR2(10);
   TYPE grades IS TABLE OF INTEGER;
   names names_table;
   marks grades;
   total integer;
BEGIN
   names := names_table(‘Kavita’, ‘Pritam’, ‘Ayan’, ‘Rishav’, ‘Aziz’);
   marks:= grades(98, 97, 78, 87, 92);
   total := names.count;
   dbms_output.put_line(‘Total ‘|| total || ‘ Students’);
FOR i IN 1 .. total LOOP
   dbms_output.put_line(‘Student:’||names(i)||’, Marks:’ || marks(i));
end loop;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result:

```
Total 5 Students
Student:Kavita, Marks:98
Student:Pritam, Marks:97
Student:Ayan, Marks:78
Student:Rishav, Marks:87
Student:Aziz, Marks:92

PL/SQL procedure successfully completed.
```

COLLECTION_IS_NULL	06531	-6531	It is raised when a program attempts to apply collection methods other than EXISTS to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
DUP_VAL_ON_INDEX .	00001	-1	It is raised when duplicate values are attempted to be stored in a column with unique index
INVALID_CURSOR	01001	-1001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	01722	-1722	It is raised when the conversion of a character string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	01017	-1017	It is raised when a program attempts to log on to the database with an invalid username or password.
NO_DATA_FOUND	01403	+100	It is raised when a SELECT INTO statement returns no rows.
NOT_LOGGED_ON .	01012	-1012	It is raised when a database call is issued without being connected to the database.
PROGRAM_ERROR	06501	-6501	It is raised when PL/SQL has an

			internal problem
ROWTYPE_MISMATCH	06504	-6504	It is raised when a cursor fetches value in a variable having incompatible data type.
SELF_IS_NULL	30625	-30625	It is raised when a member method is invoked, but the instance of the object type was not initialized.
STORAGE_ERROR .	06500	-6500	It is raised when PL/SQL ran out of memory or memory was corrupted
TOO_MANY_ROWS	01422	-1422	It is raised when a SELECT INTO statement returns more than one row.
VALUE_ERROR	06502	-6502	It is raised when an arithmetic, conversion, truncation, or sizeconstraint error occurs.
ZERO_DIVIDE	01476	1476	It is raised when an attempt is made to divide a number by zero.

13.5 PACKAGES

Packages are schema objects that groups logically related PL/SQL types, variables, and subprograms.

A package will have two mandatory parts

- Package specification
- Package body or definition

Package Specification:

The specification is the interface to the package. It just **DECLARES** the types, variables, constants, exceptions, cursors, and subprograms that can be referenced from outside the package. In other words, it contains all information about the content of the package, but excludes the code for the subprograms.

All objects placed in the specification are called **public** objects. Any subprogram not in the package specification but coded in the package body is called a**private** object.

The following code snippet shows a package specification having a single procedure. You can have many global variables defined and multiple procedures or functions inside a package.

```
CREATE PACKAGE cust_sal AS
PROCEDURE find_sal(c_id customers.id%type);
END cust_sal;
```

When the above code is executed at the SQL prompt, it produces the following result:

```
Package created
```

Package Body:

The package body has the codes for various methods declared in the package specification and other private declarations, which are hidden from the code outside the package.

The **CREATE PACKAGE BODY** Statement is used for creating the package body. The following code snippet shows the package body declaration for the***cust_sal*** package created above. I assumed that we already have CUSTOMERS table created in our database as mentioned in the PL/SQL - Variableschapter.

```
CREATE OR REPLACE PACKAGE BODY cust_sal AS

PROCEDURE find_sal(c_id customers.id%TYPE) IS
c_sal customers.salary%TYPE;
BEGIN
   SELECT salary INTO c_sal
   FROM customers
   WHERE id = c_id;
   dbms_output.put_line('Salary: '|| c_sal);
END find_sal;
END cust_sal;
/
```

When the above code is executed at the SQL prompt, it produces the following result:

```
Package body created.
```

Using the Package Elements:

The package elements (variables, procedures or functions) are accessed with the following syntax:

```
package_name.element_name;
```

Consider, we already have created the above package in our database schema, the following program uses the ***find_sal*** method of the ***cust_sal*** package:

```
DECLARE
   code customers.id%type := &cc_id;
BEGIN
   cust_sal.find_sal(code);
END;
/
```

When the above code is executed at the SQL prompt, it prompts to enter the customer ID and when you enter an ID, it displays the corresponding salary as follows:

```
Enter value for cc_id: 1
Salary: 3000

PL/SQL procedure successfully completed.
```

Example:

The following program provides a more complete package. We will use the CUSTOMERS table stored in our database with the following records:

```
Select * from customer
+----+--------------+------+-----------+---------------+
| ID | NAME    | AGE | ADDRESS | SALARY |
+----+--------------+------+-----------+---------------+
| 1 | Ramesh | 32 | Ahmedabad | 3000.00 |
| 2 | Khilan | 25 | Delhi       | 3000.00 |
| 3 | kaushik | 23 | Kota       | 3000.00 |
| 4 | Chaitali | 25 | Mumbai | 7500.00 |
| 5 | Hardik  | 27 | Bhopal    | 9500.00 |
| 6 | Komal  | 22 | MP          | 5500.00 |
```

The Package Specification:

```
CREATE OR REPLACE PACKAGE c_package AS
```

```
— Adds a customer
    PROCEDURE addCustomer(c_id customers.id%type,
    c_name customerS.No.ame%type,
    c_age customers.age%type,
    c_addr customers.address%type,
    c_sal customers.salary%type);

— Removes a customer
    PROCEDURE delCustomer(c_id customers.id%TYPE);

--Lists all customers
    PROCEDURE listCustomer;

END c_package;
/
```

When the above code is executed at the SQL prompt, it creates the above package and displays the following result “

Package created.

Creating the Package Body

```
CREATE OR REPLACE PACKAGE BODY c_package AS
   PROCEDURE addCustomer(c_id customers.id%type,
       c_name customerS.No.ame%type,
       c_age customers.age%type,
       c_addr customers.address%type,
       c_sal customers.salary%type)
IS
BEGIN
   INSERT INTO customers (id,name,age,address,salary)
       VALUES(c_id, c_name, c_age, c_addr, c_sal);
END addCustomer;

PROCEDURE delCustomer(c_id customers.id%type) IS
BEGIN
   DELETE FROM customers
   WHERE id = c_id;
END delCustomer;

PROCEDURE listCustomer IS
CURSOR c_customers is
   SELECT name FROM customers;

TYPE c_list is TABLE OF customerS.No.ame%type;
name_list c_list := c_list();
```

```
counter integer :=0;
BEGIN
   FOR n IN c_customers LOOP
   counter := counter +1;
   name_list.extend;
   name_list(counter) := n.name;
   dbms_output.put_line('Customer(' ||counter|| ')'||name_list(counter));
   END LOOP;

END listCustomer;

END c_package;
/
```

The above example makes use of the **nested table**. We will discuss the concept of nested table in the next chapter.

When the above code is executed at the SQL prompt, it produces the following result:

```
Package body created.
```

Using The Package:

The following program uses the methods declared and defined in the package*c_package*.

```
DECLARE
    code customers.id%type:= 8;

BEGIN
    c_package.addcustomer(7, 'Rajnish', 25, 'Chennai', 3500);
    c_package.addcustomer(8, 'Subham', 32, 'Delhi', 7500);
    c_package.listcustomer;
    c_package.delcustomer(code);
    c_package.listcustomer;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result:

```
Customer(1): Ramesh
Customer(2): Khilan
Customer(3): kaushik
Customer(4): Chaitali
Customer(5): Hardik
Customer(6): Komal
Customer(7): Rajnish
```

Customer(8): Subham
Customer(1): Ramesh
Customer(2): Khilan
Customer(3): kaushik
Customer(4): Chaitali
Customer(5): Hardik
Customer(6): Komal
Customer(7): Rajnish

PL/SQL procedure successfully completed

13.6 HIERARCHICAL QUERIES

If a table contains hierarchical data, then you can select rows in a hierarchical order using the hierarchical query clause:

hierarchical_query_clause::=

Description of hierarchical_query_clause.gif follows
Description of the illustration hierarchical_query_clause.gif

START WITH specifies the root row(s) of the hierarchy.

CONNECT BY specifies the relationship between parent rows and child rows of the hierarchy.

The NOCYCLE parameter instructs Oracle Database to return rows from a query even if a CONNECT BY LOOP exists in the data. Use this parameter along with the CONNECT_BY_ISCYCLE pseudocolumn to see which rows contain the loop. Please refer to CONNECT_BY_ISCYCLE Pseudocolumn for more information.

In a hierarchical query, one expression in condition must be qualified with the PRIOR operator to refer to the parent row. For example,

... PRIOR expr = expr
or
... expr = PRIOR expr

If the CONNECT BY condition is compound, then only one condition requires the PRIOR operator, although you can have multiple PRIOR conditions. For example:

CONNECT BY last_name != ‘King’ AND PRIOR employee_id = manager_id ...
CONNECT BY PRIOR employee_id = manager_id and
PRIOR account_mgr_id = customer_id ...

PRIOR is a unary operator and has the same precedence as the unary + and - arithmetic operators. It evaluates the immediately following expression for the parent row of the current row in a hierarchical query.

PRIOR is most commonly used when comparing column values with the equality operator. (The PRIOR keyword can be on either side of the operator.) PRIOR causes Oracle to use the value of the parent row in the column. Operators other than the equal sign (=) are theoretically possible in CONNECT BY clauses. However, the conditions created by these other operators can result in an infinite loop through the possible combinations. In this case Oracle detects the loop at run time and returns an error.

Both the CONNECT BY condition and the PRIOR expression can take the form of an uncorrelated subquery. However, the PRIOR expression cannot refer to a sequence. That is, CURRVAL and NEXTVAL are not valid PRIOR expressions.

You can further refine a hierarchical query by using the CONNECT_BY_ROOT operator to qualify a column in the select list. This operator extends the functionality of the CONNECT BY [PRIOR] condition of hierarchical queries by returning not only the immediate parent row but all ancestor rows in the hierarchy.

13.7 TRIGGERS

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events:

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers:

- Triggers can be written for the following purposes “
- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables

- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers:

The syntax for creating a trigger is:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
    Declaration-statements
BEGIN
    Executable-statements
EXCEPTION
    Exception-handling-statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name "Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} "This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} "This specifies the DML operation.
- [OF col_name] "This specifies the column name that will be updated.
- [ON table_name] "This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] "This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] "This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) "This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Example

To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters:

Select * from customers;

```
+----+--------------+------+-----------+-------------+
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|z
| 6 | Komal     | 22 |     MP            | 4500.00 |
```

The following program creates a **row-level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values;

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result:

Trigger created.

The following points need to be considered here:

- OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.

- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it

again only after the initial changes are applied and the table is back in a consistent state.

- The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.

Triggering a Trigger:

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table "

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00);

When a record is created in the CUSTOMERS table, the above create trigger,**display_salary_changes** will be fired and it will display the following result:

```
Old salary:
        New salary: 7500
Salary difference:
```

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The UPDATE statement will update an existing record in the table:

```
UPDATE customersSET
salary = salary + 500
```

WHERE id = 2;

trigger,**display_salary_changes** will be fired and it will display the following result :

```
Old salary: 1500
New salary: 2000
Salary difference: 500
```

13.8 SUMMARY

This chapter defines a Cursors and its types. It also describes collections and composite data types, Procedure and Functions. Also gives an idea about Exception Handling and Packages.

13.9 REFERENCES

1. Database System and Concepts A Silberschatz, H Korth, S Sudarshan McGraw-Hill Fifth Edition
2. "Fundamentals of Database Systems" by Elmsari, Navathe, 5th Edition, Pearson Education (2008).
3. "Database Management Systems" by Raghu Ramakrishnan, Johannes Gehrke, McGraw Hill Publication.
4. "Database Systems, Concepts, Design and Applications" by S.K.Singh, Pearson Education.

13.10 UNIT END QUESTIONS

MCQs:

1. Which statements are used to control a cursor variable?
   a. OPEN-FOR
   b. FETCH
   c. CLOSE
   d. All mentioned above
2. Which of the following is used to declare a record?
   a. %ROWTYPE
   b. %TYPE
   c. Both A & B
   d. None of the above
3. Which of the following has a return type in its specification and must return a value specified in that type?
   a. Function
   b. Procedure
   c. Package
   d. None of the above
4. Which collection exception is raised when a subscript designates an element that was deleted, or a nonexistent element of an associative array?
   a. NO_DATA_FOUND
   b. COLLECTION_IS_NULL
   c. SUBSCRIPT_BEYOND_COUNT
   d. SUBSCRIPT_OUTSIDE_LIMIT
5. Observe the following code and fill in the blanks "

```
DECLARE
total_rows number(2);
BEGIN
UPDATE employees
SET salary = salary + 500;
IF _____________ THEN
dbms_output.put_line('no employees selected');
```

```
ELSIF _____ THEN
total_rows := _____;
dbms_output.put_line( total_rows || ' employees selected ');
END IF;
END;
```

a . %notfound, %found, %rowcount.
b . sql%notfound, sql%found, sql%rowcount.
c . sql%found, sql%notfound, sql%rowcount.
d . %found, %notfound, %rowcount.

6. Which of the following is true about PL/SQL index-by tables?
   A. It is a set of key-value pairs.
   B. Each key is unique and is used to locate the corresponding value.
   C. The key can be either an integer or a string.
   D. All of the above.

7. Which keyword is used instead of the assignment operator to initialize variables?
   a. NOT
   b. DEFAULT
   c. %TYPE
   d. %ROWTYPE

8. Which of the following returns all distinct rows selected by either query?
   a. INTERSECT
   b. MINUS
   c. UNION
   d. UNION ALL

9. For which Exception, if a SELECT statement attempts to retrieve data based on its conditions, this exception is raised when no rows satisfy the SELECT criteria?
   a. TOO_MANY_ROWS
   b. NO_DATA_FOUND
   c. VALUE_ERROR
   d. DUP_VAL_ON_INDEX

10. Which keyword and parameter used for declaring an explicit cursor?
    a. constraint
    b. cursor_variable_declaration
    c. collection_declaration
    d. cursor_declaration

Answer the following:

1. Explain PL/SQL Records with example.
2. Explain Explicit Cursors with example.
3. Explain Attributes of Explicit Cursors with example.

4. Explain the concept of Cursor for Loop with example.
5. Explain for Update clause and where current clause with example.
6. Explain Exception Handling in PL/SQL with example.
7. Differentiate Anonymous blocks and subprograms
8. Differentiate Procedures and Functions
9. Create a PL/SQL function to find out greatest two numbers. Call the function to display output.
10. Explain parts of Triggers with the help of example.

*****