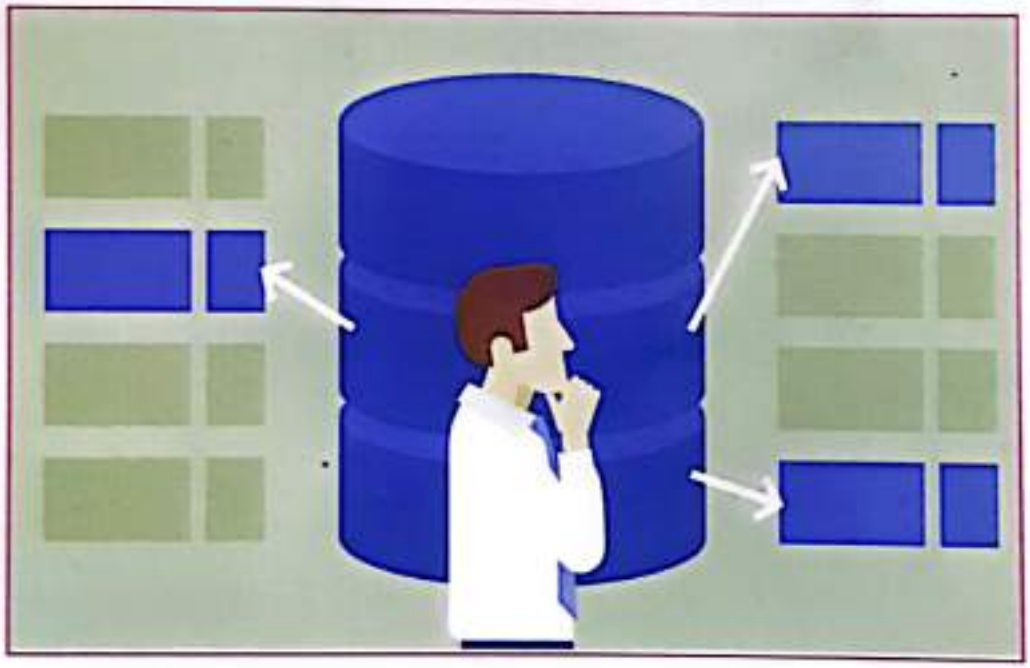
NEW SYLLABUS
CBCS PATTERN

B.B.A.
(Computer Application)
Semester-II

# RELATIONAL DATABASE

Dr. Ms. MANISHA BHARAMBE
ABHIJEET D. MANKAR



NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

SPPU New Syllabus

A Book Of

# RELATIONAL DATABASE

**For B.B.A.(Computer Application) : Semester - II**

**[Course Code 204 : Credit - 3]**

**CBCS Pattern**

**As Per New Syllabus, Effective from June 2019**


**Dr. Ms. MANISHA BHARAMBE**
M.Sc. (Computer Science), M.Phil. Ph.D.
Vice Principal & Head,
Department of Computer Science,
MES Abasaheb Garware College,
PUNE 4.

**ABHIJEET D. MANKAR**
M.C.S., SET
Assistant Professor,
Department of Computer Science,
Tuljaram Chaturchand College of Arts,
Science and Commerce (Autonomous),
BARAMATI.


**Price ₹ 260.00**


NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

N4944

# Syllabus ...

1. **Introduction to RDBMS**
   - Introduction to popular RDBMS product and their Features
   - Difference between DBMS and RDBMS
   - Relationship among Application Programs and RDBMS
2. **PL-SQL**
   - Overview of PLSQL Data Types, PLSQL BLOCK
   - Exception Handling
   - Functions, Procedures
   - Cursor
   - Trigger, Package
3. **Transaction Management**
   - Transaction Concept
   - Transaction Properties
   - Transaction States
   - Concurrent Execution
   - Serializability
4. **Concurrency Control and Recovery system**
   - Lock Based Protocol
   - Timestamp Based Protocol
   - Deadlock Handling
   - Failure Classification
   - Recovery and Atomicity
   - Recovery with Concurrent Transaction

# Contents ...



❖❖❖

# 1... Introduction to RDBMS

## Objectives...

- To understand concepts of RDBMS.
- To learn about the advantages of RDBMS and different RDBMS products.
- To know about the relationship between application programs and RDBMS.

## 1.1 INTRODUCTION

### 1.1.1 What is Database?

- A database is a collection of related data elements such as, Tables (entities), Columns (fields or attributes), and Rows (records).
- A database could be as simple as a text file with a list of names or it could be as complex as a large Relational Database Management System.
- A database is a collection of data organized in a particular way.
- Databases can be of many types such as Flat File Databases, Relational Databases, and Distributed Databases etc.

### 1.1.2 Database Management System (DBMS)

- A Database Management System (DBMS) is a computer program for managing a permanent, self-descriptive repository of data. This repository of data is called a database and is store in one or more files.
- A Database Management System (DBMS) is a software program that enables the creation and management of databases.
- Most of today's database systems are referred to as a Relational Database Management System (RDBMS), because of their ability to store related data across multiple tables.

- There are many reasons why you could use a DBMS which are given below:
  1. **Sharing between applications:** Multiple application programs can read and write data to the same database.
  2. **Crash recovery:** The database is protected from hardware crashes, disk media failures and some user errors.
  3. **Security:** Data can be protected against unauthorized read and write access.
  4. **Sharing between users:** Multiple users can access the database at the same time.
  5. **Data distribution:** The database may be partitioned across various sites, organizations and hardware platforms.
  6. **Extensibility:** Data may be added to the database without disruption of existing programs. Data can be reorganized for faster performance.
  7. **Integrity:** You can specify rules that data must satisfy. A DBMS can control the quality of its data over and above facilities that may be provided by application programs.

## 1.1.3 Relational Database Management System (RDBMS)

(S- 18, W-18)

- RDBMS stands for Relational Database Management System. Edgar F. Codd at IBM invented the relational database in 1970.
- RDBMS allows operations in a human logical environment. The main elements of RDBMS are based on Codd's 12 rules for a relational system.
- RDBMS data is structured in database tables, fields and record. Each RDBMS table consists of database table rows. Each database table row consists of one or more database table fields.
- RDBMS stores data into collection of tables, which might be related by common fields.
- RDBMS also provide relational operators to manipulate the data stored into database tables. Most RDBMS use SQL as database query language.
- A Relational Database Management System (RDBMS) provides a comprehensive and integrated approach to information management.
- A relational model provides the basis for a relational database. A relational model has three aspects: Structures, Operations and Integrity rules.
  1. **Structures:** It consists of a collection of objects or relations that store data. An example of relation is a table. You can store information in a table and use the table to retrieve and modify data.
  2. **Operations:** They are used to manipulate data and structures in a database. When using operations. You must adhere to a predefined set of integrity rules.
  3. **Integrity Rules:** The rules are laws that govern the operations allowed on data in a database. This ensures data accuracy and consistency.

## 1.1.3.1 Definition of RDBMS (W-17)

- Relational Database Management System ( RDBMS) refers to a relational database plus supporting software for managing users and processing SQL queries, performing backups/restores and associated tasks.
- RDBMS usually include an API so that developers can write programs that use them.

  Typical RDBMS includes:
  - Microsoft Access
  - Microsoft SQL Server
  - IBM DB2
  - Oracle
  - MySQL
  - Ingres
  - PostgreSQL
  - SQLite
  - Maria DB
  - Informix
  - Azure SQL

## 1.1.3.2 Characteristics of RDBMS (S-18)

- RDBMS consist of following characteristics:
  1. **Data abstraction:** Relational abstraction enhances program-data independence.
  2. **Self-describing data:** Metadata describing structure of data stored together with data.
  3. **Concurrency:** Supporting shared concurrent access (transactions).
  4. **Support for multiple views:** External users can be provided with different views of the data.
  5. **Security:** RDBMS offers different levels of security features such as Privacy/Confidentiality, Integrity, Availability and Accountability.

## 1.1.3.3 Components of RDBMS

- Relational database components includes: Table, Row, Column, Field, Primary key, Foreign key.
- Fig. 1.1 shows the components of RDBMS.
- **A Table** is a basic storage structure of an RDBMS and consists of columns and rows. A table represents an entity. For example, the E_DEPT table stores information about the departments of an organization.

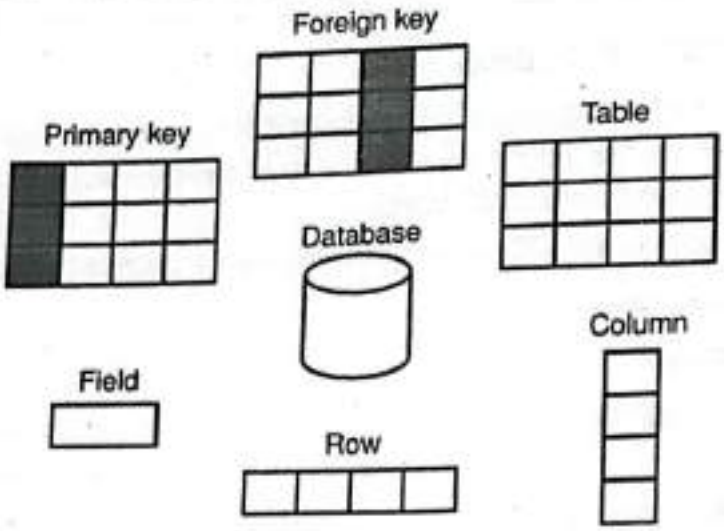



**Fig. 1.1: Components of Relational Database**

- **A Row** is a combination of column values in a table and is identified by a primary key. Rows are also known as **records**. For example, a row in the table E_DEPT contains information about one department.
- **A Column** is a collection of one type of data in a table. Columns represent the attributes of an object. Each column has a column name and contains values that are bound by the same type and size. For example, a **'Name'** column in the table E_DEPT specifies the names of the departments in the organization.
- **A Field** is an intersection of a row and a column. A field contains one data value. If there is no data in the field, the field is said to contain a NULL value.

E_DEPT

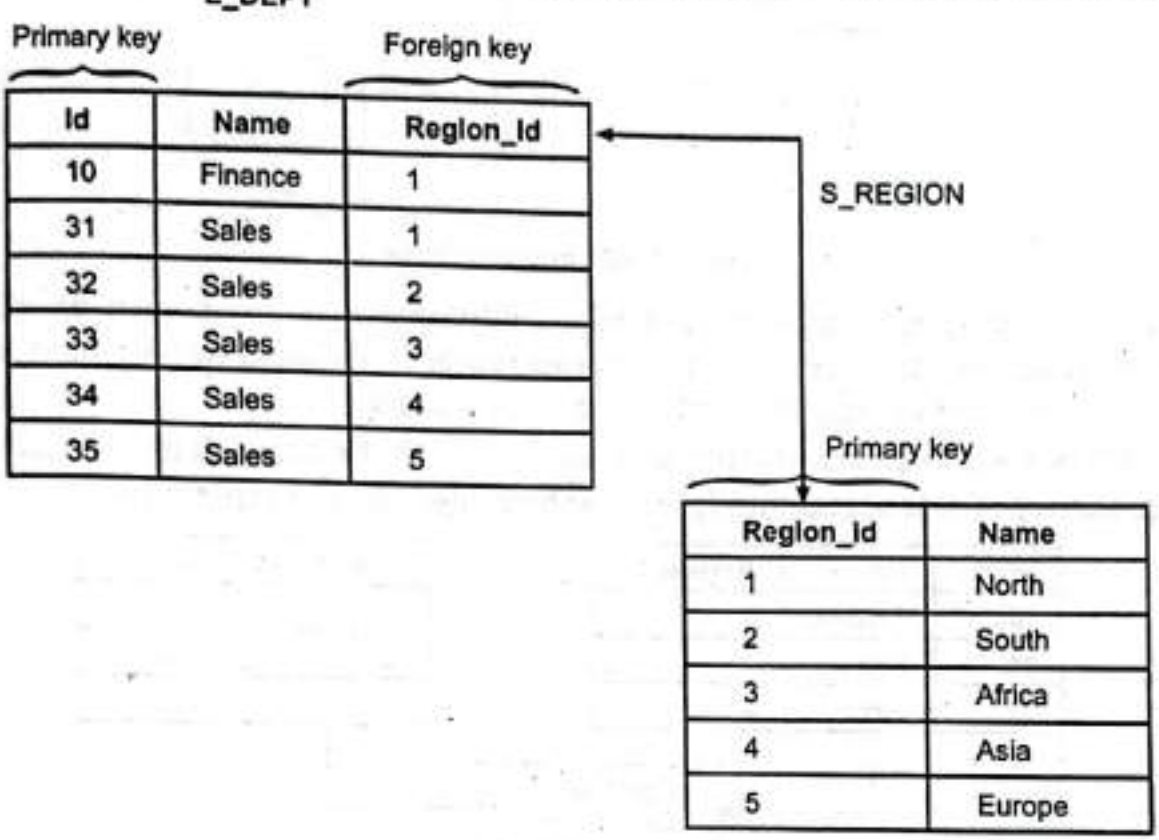
| Id | Name | Region_Id |
|---|---|---|
| 10 | Finance | 1 |
| 31 | Sales | 1 |
| 32 | Sales | 2 |
| 33 | Sales | 3 |
| 34 | Sales | 4 |
| 35 | Sales | 5 |

Table
Row
Field
Column

**Fig. 1.2: Table, Row, Column and Field of RDBMS**

- **A Primary key** is a column or a combination of columns that is used to uniquely identify each row in a table. For example, the column containing department numbers in the E_DEPT table is created as a **primary key** and therefore, every department number is different. A primary key must contain a value. It cannot contain a NULL value.

- A **Foreign key** is a column or set of columns that refers to a primary key in the same table or another table. You use foreign keys to establish principle connections between, or within, tables. A foreign key must either match a primary key or else be NULL. Rows are connected logically when required. The logical connections are based upon conditions that define a relationship between corresponding values, typically between a primary key and a matching foreign key. This relational method of linking provides great flexibility as it is independent of physical links between records.




| Id | Name | Region_Id |
|---|---|---|
| 10 | Finance | 1 |
| 31 | Sales | 1 |
| 32 | Sales | 2 |
| 33 | Sales | 3 |
| 34 | Sales | 4 |
| 35 | Sales | 5 |

| Region_Id | Name |
|---|---|
| 1 | North |
| 2 | South |
| 3 | Africa |
| 4 | Asia |
| 5 | Europe |

**Fig. 1.3: Primary and Foreign key of RDBMS**

## 1.1.3.4 Features of RDBMS

- In RDBMS data is stored in the form of rows and columns i.e. in tabular form. You can execute commands in the Structured Query Language (SQL) to manipulate data. SQL is the International Organization for Standardization (ISO) standard language for interacting with a RDBMS.
- An RDBMS provides full data independence. The organization of the data is independent of the applications that use it. You do not need to specify the access routes to tables or know how data is physically arranged in a database.
- A relational database is a collection of individual, named objects. The basic unit of data storage in a relational database is called **a table**. A table consists of rows and columns used to store values. For access purpose, the order of rows and columns is insignificant. You can control the access order as required.

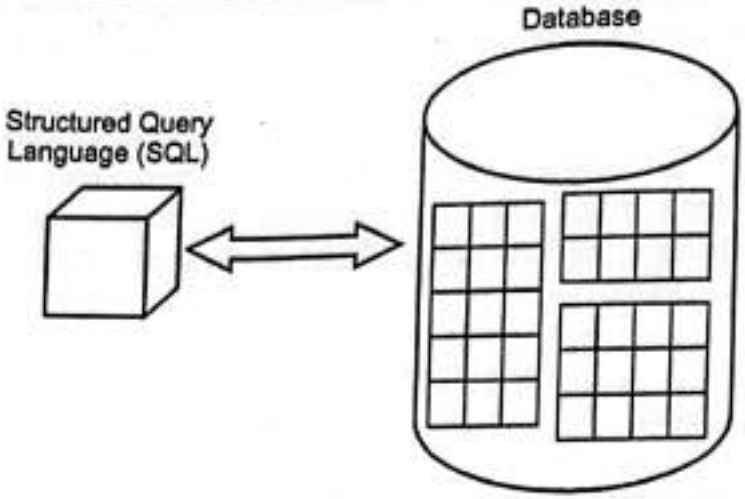



**Fig. 1.4: SQL and Database**

- When querying the database, you use conditional operations such as joins and restrictions (conditions). A join combines data from separate database rows. A restriction limits the specific rows returned by a query.
- An RDBMS enables data sharing between users. At the same time, you can ensure consistency of data across multiple tables by using integrity constraints.

| Id | Name | Region_Id |
|---|---|---|
| 10 | Finance | 1 |
| 31 | Sales | 1 |
| 32 | Sales | 2 |

| Region_Id | Name |
|---|---|
| 1 | North |
| 2 | South |
| 3 | Africa |

Conditional Operations

Results

| Id | Name | Region_name |
|---|---|---|
| 10 | Finance | North |
| 32 | Sales | South |

**Fig. 1.5: Conditional Operations**

- An RDBMS uses various types of data integrity constraints. These types include entity, column, referential and user-defined constraints.
- The entity constraint ensures uniqueness of rows, and the column constraint ensures consistency of the type of data within a column.
- The other type referential constraint ensures validity of foreign keys, and user-defined constraints are used to enforce specific business rules.
- An RDBMS minimizes the redundancy of data. Redundancy means unnecessary duplication of data. This means that similar data is not repeated in multiple tables.

- Allows for the virtual table creation. This table safely store and secure sensitive content.

## 1.1.4 Advantages of RDBMS (S-18)

- RDBMS consists of following advantages:
  - RDBMS is easy to implement and use.
  - Provides high level data security.
  - Have a definite procedure to work upon.
  - Data is retrieved using SQL queries only.
  - RDBMS is consistent, durable, supports isolation

## 1.1.5 Disadvantages of RDBMS (S-18)

- RDBMS consists of following disadvantages:
  - The relational database systems are easy to implement and use. Because of this feature, many persons or departments design their own databases and applications. These information islands prevent in integrating the information which is necessary for effective functioning of the organization.
  - Proper skill and training is required to work with RDBMS.
  - Requires supportive software and hardware.
  - Cost of execution is high.

## 1.1.6 Keys of RDBMS

- Every table must have some columns or combination of columns which uniquely identify each row in the table. For that we require a key. A key is simply a field used to identify a record. In other words, a key is subset of attributes with following properties:
  1. The value of key is unique for each tuple.
  2. No data redundancy.
- **Keys can be classified as:**
  1. Primary key
  2. Candidate key
  3. Foreign key

### 1.1.6.1 Primary Key

- In a relation, there is one attribute or a group of attributes with values that are unique within that relation and thus can be used to uniquely identify the tuples of that relation.
- This key which uniquely identifies a row is known as primary key of that relation.

- For example, consider a relation Customer, where Social Security Number of the entity set customer is sufficient to distinguish one customer entity from another. So Social Security Number is a primary key.

| Social_Security_Number | Customer name | Address |
|---|---|---|
| 100 - 205 | John | Paris |
| 103 - 202 | Jimmy | New York |
| 208 - 256 | Juliee | Rye |

## 1.1.6.2 Candidate Key

- The attribute which possess the unique identification property in a relation is called as Candidate Key. There can be more than one candidate keys in a relation.
- Candidate key should have following things:
  1. It must be unique.
  2. A candidate key's value must exist. It cannot be NULL.
  3. The value of the candidate key must be stable. It's value cannot change outside the control of the system. In Customer table, various employees were working and having unique identification number called as Social Security Number (SSN).

| Customer name | SSN | Basic |
|---|---|---|
| John | 001-256 | ₹ 14,000 |
| Martin | 005-123 | ₹ 2,000 |
| Paster | 008-200 | ₹ 1,000 |
| Mary | 101-401 | ₹ 18,000 |
| Johns | 102-030 | ₹ 48,000 |

Candidate key

## 1.1.6.3 Foreign Key

- If any key in a given relation has reference to the value of a primary key of some other relation then it is called as foreign key.
- Foreign key can have duplicate values. It is used to search a record from two relations.

Relation = Dept

| Dept_no | Name |
|---|---|
| 1 | Production |
| 2 | Purchasing |
| 3 | Marketing |

Relation = Emp.

Primary Key

Foreign Key

| Emp_id | Emp_name | Dept_no |
|---|---|---|
| 10 | Ramesh | 2 |
| 25 | Indranil | 1 |

## 1.1.6.4 Super Key

- Super key is a set of one or more attributes which taken collectively allows us to identify uniquely an entity in the entity set.
- Consider the entity set account with attributes {acc_no. cust_name, bank_name, balance, address}.
- Here, acc_no is super key of account entity set. Similarly, (acc_no, cust_name) is a super key and (acc_no., bank_name) is a super key. But cust_name and bank_name independently are not the super keys.
- Super key may contain extra attributes i.e. if k is a super key, then any super set of k is also a Super key.

**[Note:** Candidate key is minimal super key.]

## 1.1.6.5 Popular RDBMS Products

(W-17) (W-18)

**I. SQL SERVER**

- SQL server is Database management software, uses own file structure, own logins and own security.
- SQL server uses Access through two main ways:
  1. Query Analyzer
  2. Enterprise Manager
- SQL server consists of three Services:
  1. **MS SQL Server:** For Data and Query processing.
  2. **SQL Server Agent:** For Schedulable jobs and alerts.
  3. **Microsoft Distributed Transaction Co-ordinator:** For Handling data from multiple sources.
- SQL Server is a Relational Database Management System (RDBMS) developed and marketed by Microsoft. As a database server, the primary function of the SQL Server is to store and retrieve data used by other applications.
- Similar to other RDBMS software, SQL Server is built on top of SQL, a standard programming language for interacting with the relational databases. SQL server is tied to Transact-SQL, or T-SQL, the Microsoft's implementation of SQL that adds a set of proprietary programming constructs.

- It is platform dependent. It is both GUI and command based software. It supports SQL (SEQUEL) language which is an IBM product, non-procedural, common database and case insensitive language.

**Usage of SQL Server:**

- To create databases.
- To maintain databases.
- To analyze the data through SQL Server Analysis Services (SSAS).
- To generate reports through SQL Server Reporting Services (SSRS).
- To carry out ETL operations through SQL Server Integration Services (SSIS).

**Versions of SQL Server:**

- **1998:** This version included tools for query analysis like the Query Analyzer (which was replaced by the SSMS (SQL Server Management Studio) in SQL 2005). It introduced OLAP Services (SSAS now) and Data Transformation Services (SSIS now). This code base was now fully independent from Sybase.
- **2000:** SQL Server 2000 was an important version where the OLAP Services in SQL 7 was replaced by the Analysis Services. It was also introduced the XML support, User Defined Functions, Indexed Views, Replication enhancements, Log Shipping and more.
- **2005:** This version introduced the SSMS that we use now. The BIDS (Business Intelligence Development Studio) was created in this version. SQL Server included the Adventureworks database in the installer.
- **2008:** In SQL Server 2008, Adventureworks was not included in the Database installer like in SQL Server 2005. Another important difference was the backup compression. It was possible to create customized audits using Auditing.
- **2012:** In this version, the BIDS (Business Intelligence Development Studio) was replaced by SSDT (SQL Server Data Tools).
- **2014:** The main features in SQL Server 2014 were the integration to Azure and Memory Optimized Tables.
- **2016:** In SQL Server 2016, PolyBase support was introduced. With PolyBase, you can query NoSQL data like CSV files stored in Azure Blob Storage or in HDInsight. Another interesting feature is JSON support which includes new features to handle JSON data.
- **2017:** In this version, CLR assemblies are now be added. SQL Server 2017 includes Python support, Identity Cache, Graph Database, Resumable Indexes, New string functions, Adaptative Query Processing, Automatic tuning.
- SSIS is now supported in Linux. In SSDT, you have now a DAX editor and tabular databases are the default installation. In SSRS, you can run native DAX queries.
- **2019:** Microsoft recently announced the preview of SQL Server 2019 on September 24 at the Ignite 2018 Conference. In SQL Server 2019, we can combine big data with

the analytical database or traditional database system. This provides data scientists to access big data with simple T-SQL queries.

**II. ORACLE:**

- It is a very large and multi-user database management system. Oracle is a relational database management system developed by 'Oracle Corporation'.
- Oracle works to efficiently manage its resource, a database of information, among the multiple clients requesting and sending data in the network.
- It is an excellent database server choice for client/server computing. Oracle supports all major operating systems for both clients and servers, including MSDOS, NetWare, UnixWare, OS/2 and most UNIX flavors.
- Current version of Oracle database is 18c.

**Objects of Oracle:**

- A schema is a collection of logical structures of data, or schema objects. A schema is owned by a database user and has the same name as that user.
- Schema objects can be created and manipulated with SQL and include the following types of objects:
  - Clusters
  - Database links
  - Database triggers
  - Dimensions
  - External procedure libraries
  - Indexes and index types
  - Java classes, Java resources, and Java sources
  - Materialized views and materialized view logs
  - Object tables, object types, and object views
  - Operators
  - Sequences
  - Stored functions, procedures, and packages
  - Synonyms
  - Tables and index-organized tables
  - Views

**Details of some Oracle objects:**

- **Clusters :** Clusters are an optional method of storing table data. A cluster is a group of tables that share the same data blocks because they share common columns and are often used together. For example, the employees and departments table share the department_id column. When you cluster the employees and departments tables, Oracle physically stores all rows for each department from both the employees and departments tables in the same data blocks.

- **Indexes :** Indexes are optional structures associated with tables and clusters. You can create indexes on one or more columns of a table to speed SQL statement execution on that table. An Oracle index provides a faster access path to table data. Indexes are the primary means of reducing disk I/O when properly used. You can create many indexes for a table as long as the combination of columns differs for each index. You can create or drop an index at any time without affecting the base(actual) tables or other indexes. If you drop an index, all applications continue to work. Indexes require storage space.
- **Tables :** Tables are the basic unit of data storage in an Oracle database. Data is stored in rows and columns. You give each column a column name (such as employee_id, last_name, and job_id), a datatype (such as VARCHAR2, DATE, or NUMBER), and a width. The width can be predetermined by the datatype, as in DATE. If columns are of the NUMBER datatype, define precision and scale instead of width. A row is a collection of column information corresponding to a single record. Rules specified for each column are call integrity constraints. e.g. Not Null constraint.
- **Views :** A view is a representation of the data contained in one or more tables or other views. A view takes the output of a query and treats it as a table. Therefore, a view can be thought of as a stored query or a virtual table. You can use views in most places where a table can be used. For example, the employees table has several columns and numerous rows of information. If you want users to see only five of these columns or only specific rows, then you can create a view of that table for other users to access.

**Features of Oracle Database:** (W-17

**1. Scalability and Performance:**

- Oracle includes several software mechanisms to satisfy the following important requirements of an information management system:
- Data concurrency of a multiuser system must be increased.
- Data must be read and modified in a consistent manner. The data a user is accessing or modifying is not modified until the user is finished with the data.
- High performance is required for maximum productivity from the many users of the database system.

This contains the following:

(i) Concurrency
(ii) Read Consistency
(iii) Locking Mechanisms
(iv) Real Application Clusters
(v) Portability

**(i) Concurrency:** A primary responsibility of a multiuser database management system is how to control concurrency, which is the access of the same data by many users at

the same time. Without adequate concurrency controls, data could be updated or changed incorrectly, compromising data integrity. One way to manage data concurrency is to make each user wait for a turn. The aim of a database management system is to reduce that wait for each user so it is either nonexistent or negligible. All DML statements should precede with as little interference as possible and destructive interactions between concurrent transactions must be avoided. Neither performance nor data integrity can be sacrificed. Oracle handles such issues by using various types of locks and a multi-version consistency model. These features are based on the concept of a transaction. It is the application designer's responsibility to check that transactions fully use these concurrency and consistency features.

(ii) **Read Consistency:** Read consistency, as supported by Oracle, does the following:

- This guarantees that the set of data returned by a query statement comes from a single point in time and does not change during statement execution (statement-level read consistency).
- Ensures that readers of data do not wait for writers or other readers of the same data.
- Ensures that writers of data do not wait for readers of the same data.
- Ensures that writers only wait for other writers if they attempt to update same rows in concurrent transactions.

The simplest way to think of Oracle's implementation of read consistency is to imagine each user operating its own copy of the database, hence Oracle Database is the multisession consistency model.

(iii) **Locking Mechanisms:** Oracle uses locks to control concurrent access to data. When updating information, the data server holds that information with a lock until the update is committed. Until that happens, no one can make changes to the locked information. This guarantees the data integrity of the system.

Oracle provides unique non-escalating row-level locking. Oracle always locks only the row of information being updated. Because Oracle includes the locking information with the actual rows themselves, Oracle can lock large number of rows so users can work concurrently without unnecessary delays.

(iv) **Real Application Clusters:** Real Application Clusters (RAC) consists of several Oracle instances running on multiple clustered computers, which communicate with each other by means of a so-called interconnect. RAC uses cluster software to access a shared database that is stored on shared disk. RAC combines the processing power of these multiple interconnected computers to provide system redundancy, near linear scalability, and high availability. RAC also offers significant benefits for both OLTP and data warehouse systems and all systems and applications can efficiently take advantage of clustered environments.

You can scale applications in RAC environments to satisfy increasing data processing demands without changing the application code. As you add resources such as nodes or storage, RAC extends the processing powers of these resources beyond the limits of the individual components.

(v) **Portability:** Oracle provides unique portability across all major platforms and makes sure that your applications run without modification after changing platforms. This is because the Oracle code base is similar across platforms, so you have similar feature functionality across all platforms, for complete application transparency. Because of this portability, you can easily upgrade to a more powerful server as your requirements changes.

2. **Manageability:**

People who administer the operation of an Oracle database system, known as database administrators (DBAs), are responsible for creating Oracle databases, making sure their smooth operation, and monitoring their use. Oracle also offers the following features:

(i) **Self-Managing Database:** Oracle Database provides a high degree of self-management - automating routine DBA tasks and reducing complexity of space, memory, and resource administration. Oracle self-managing database features include the following:

1. Automatic undo management,
2. Dynamic memory management,
3. Oracle-managed files,
4. Mean time to recover,
5. Free space management,
6. Multiple block sizes, and
7. Recovery Manager (RMAN).

(ii) **Oracle Enterprise Manager:** Enterprise Manager is a system management tool that provides a combined solution for centrally controlling your heterogeneous environment. Enterprise Manager provides a comprehensive systems management platform for controlling Oracle products.

(iii) **Automatic Storage Manager:** Automatic Storage Manager automates and simplifies the layout of data files, control files and log files. Database files are automatically distributed across all available disks and database storage is rebalanced whenever the storage configuration is modified.

(iv) **Scheduler:** The Scheduler allows database administrators and application developers control when and where various tasks take place in the database environment. For example, database administrators can arrange and monitor database maintenance jobs such as backups or data warehousing loads and extracts.

(v) **Database Resource Manager:** The Database Resource Manager controls the distribution of resources among various sessions by controlling the execution schedule inside the database.

3. **Database Backup and Recovery:**

In every database system, the possibility of a system or hardware failure always exists. The goals after a failure are to make sure that the effects of all committed transactions are reflected in the recovered database and to return to normal operation as quickly as possible while protecting users from problems caused by the failure.

Oracle provides various mechanisms for the following:

- Database recovery required by different types of failures.
- Flexible recovery operations to match any situation.
- Availability of data during backup and recovery operations so users of the system can continue to work.

4. **High Availability:**

Oracle has a number of products and features that provide high availability in cases of unplanned or planned downtime. These include Fast-Start Fault Recovery, Real Application Clusters, Recovery Manager (RMAN), backup and recovery solutions, Oracle Flashback, partitioning, Oracle Data Guard, Log Miner, multiplexed redo log files, online reorganization. These can be used in various combinations to satisfy specific high availability needs.

5. **Business Intelligence:**

This feature describes several business intelligence features as the following.

(i) **Data Warehousing:** A data warehouse is a relational database designed for query and analysis rather than for transaction processing. It usually contains historical data derived from transaction data, but it can contain data from other sources. It clearly separates analysis workload from transaction workload and make possible for an organization to join data from several sources.

(ii) **Extraction, Transformation and Loading (ETL):** You must load your data warehouse regularly so that it can serve its purpose of making business analysis easier. To do this, data from one or more operational systems must be extracted and copied into the warehouse. The process of extracting data from source systems and bringing it into the data warehouse is commonly called ETL. ETL stands for Extraction, Transformation, and Loading.

(iii) **Data Mining:** With Oracle Data Mining, data never leaves the database — the data, data preparation, model building, and model scoring results all remain in the database. This enables Oracle to provide an infrastructure for application developers to combine data mining seamlessly with database applications. Some typical examples of the applications that data mining are used in are call centers, ATMs, ERM, and business planning applications. Data mining functions like model building, testing, and scoring are provided through a Java API.

6. **Content Management:**

Oracle includes data types to handle different types of rich Internet content such a relational data, object-relational data, XML, text, audio, video, image, and spatia These data types look as native types in the database. They can all be queried usin SQL. A single SQL statement can contain data belonging to any or all of these dat types.

(i) **XML in Oracle:** XML, (eXtensible Markup Language) is the standard way to recogniz and describe data on the Web. Oracle XML DB treats XML as a native data type in th database. Oracle XML DB gives a number of easy ways to create XML documents fro relational tables. The result of any SQL query can be automatically changed into a XML document. Oracle also includes a set of utilities, available in Java and C++, t simplify the task of creating XML documents.

(ii) **LOBs:** The LOB data types BLOB, CLOB, NCLOB, and BFILE make possible for you t store and manipulate large blocks of unstructured data in binary or characte format. They provide efficient, random, piece-wise access to the data.

(iii) **Oracle Text:** Oracle Text indexes any document or textual content to give fas accurate retrieval of information. Oracle Text allows text searches to be combine with regular database searches in a single SQL statement. The Oracle Text SQL AP makes it simple and intuitive for application developers and DBAs to create an preserve Text indexes and run Text searches.

(iv) **Oracle Spatial:** Oracle includes built-in spatial(dimensional) features that allow yo store, index, and manage location content and query location relationships using th power of the database. The Oracle Spatial Option gives advanced spatial feature such as linear reference support and co-ordinate systems.

7. **Security Features:**

Oracle contains security features that control how a database is accessed and use For example, security mechanisms:

- Prevent unauthorized database access.
- Prevent unauthorized access to schema objects.
- Audit user actions.

Associated with each database user is a schema by the same name. By default, eac database user creates and has access to all objects in the corresponding schema.

Database security can be classified into two categories: **System Security** and **Dat Security**.

- **System security** contains the mechanisms that control the access and use o the database at the system level.
- **Data security** contains the mechanisms that control the access and use of th database at the schema object level.

### III. MySQL

- MySQL is a Relational Database Management System (RDBMS). The program runs as a server providing multi-user access to a number of databases.
- MySQL was acquired by Oracle as a part of Sun Microsystems in 2009 .
- MySQL is free and open-source software under the terms of the GNU General Public License, and is also available under a variety of proprietary licenses.

**Features:**

- MySQL Server 8.0 was announced in April 2018, including NoSQL Document Store, atomic and crash safe DDL sentences and JSON Extended syntax, new functions, such as JSON table functions, improved sorting, and partial updates. Previous MySQL Server 8.0.0-dmr (Milestone Release) was announced 12 September 2016.
- MySQL is written in C and C++.

**Latest features of MySQL 8 are:**

1. **Transactional Data Dictionary:** Dictionary will now get stored in InnoDB tables. New advancements will enhance performance of the database.
2. **Enhanced JSON support:** Ranges are now supported by JSON path expressions.
3. **Persistent runtime configuration:** SET PERSIST lets runtime configuration to be reserved and survived a restart.
4. **Document Store:** This feature makes the MySQL more beneficial over other NoSQL databases in which developers don't have to give up transactional semantics.
5. **Auto_increment counter persistence over restart :** Auto-increment counters values will be stored in the redo log in MySQL 8.0 version and this allows retrieval upon restart.
6. **CTEs and Window functions:** MySQL has implemented CTEs (Common Table Expressions) or recursive queries along with Window functions.
7. **Optimizer Hints:** Optimizer hints are the better solution to the optimizer_switch variable. Now user can control optimizer behavior with hints on a per-query basis.
8. **SQL roles:** Roles are named collections of privileges (SELECT, INSERT, etc.) that offer convenient permissions management
9. **Cloud-friendly:** The new option availability of innodb_dedicated_server, the user can now auto-detect the system memory and have MySQL adjust properly without the need of editing the configuration files.
10. **Invisible indexes:** An invisible index is not applied by the optimizer and looks invisible. Invisible indexes are rather maintained in the background so that they can be switched back on any moment.
11. **Character sets and collations:** In MySQL 8.0, the new default character set will be set to utf8mb4

12. **UUID functions:** The User can now store UUIDs in VARBINARY (16) format whic was earlier stored in CHAR(36).

**IV. IBM DB2**

- DB2 is a database product from IBM. It is a Relational Database Management Syste (RDBMS). DB2 is designed to store, analyze and retrieve the data efficiently. DB product is extended with the support of Object-Oriented features and non-relationa structures with XML.

**Versions of DB2:**

- In early **2012,** IBM announced the next version of DB2, DB2 10.1 (code name Galile for Linux, UNIX, and Windows. DB2 10.1 contained a number of new dat management capabilities including row and column access control. IBM als introduced 'adaptive compression' capability in DB2 10.1, a new approach t compressing data tables.
- In June **2013,** IBM released DB2 10.5 (code name "Kepler").
- On 12 April **2016,** IBM announced DB2 LUW 11.1, and in June 2016, it was released.
- In mid-**2017,** IBM re-branded its DB2 and dashDB product offerings and amende their names to "DB2".
- On June 27, **2019,** IBM released DB2 11.5, the AI Database. It added AI functionality t improve query performance as well as capabilities to facilitate AI applicatio development.

**V. PostgreSQL**

- PostgreSQL is a free, general purpose, open source and object relational databas management system. PostgreSQL was initially designed to run on UNIX platform
- At present, PostgreSQL also runs on various platforms like Mac OS X, Solaris an Windows.
- The latest version of PostgreSQL is 12.1 which released on 14$^{th}$ November 2019.
- PostgreSQL is used by many companies like Instagram, Fujitsu etc.
- PostgreSQL is one of the top RDBMSs.
- PostgreSQL meets at least 160 out of the 179 compulsory features for SQL201 Core conformance.

The various features found in PostgreSQL are as follows:

- It supports wide range of data types, such as Numeric, String, Array, JSON XML, Polygon etc.
- It supports data integrity.
- It supports reliability and disaster recovery.(Write Ahead Logging, Replicatio etc.)
- It supports security.
- It supports concurrency and performance.
- It also supports extensibility. (PL/pgSQL, stored functions and procedures etc.)

**VI. Informix**

- Latest stable release 12.10.xC7. Coded in assembly, C, C++.

Few features of this tool are:

- Hardware uses less space, data is available at all times and does not need maintenance time. It is developed by IBM.
- It's licensed tool and the cost of each license is affordable.

**VII. MS-ACCESS**

- Microsoft Access is a Relational Database Management System (RDBMS), designed mainly for home or small business usage.
- Access is known as a Desktop Database System because its functions are intended to be run from a stand-alone computer. This is in contrast to a server database application (such as SQL Server), where it is intended to be installed on a server, then accessed remotely from multiple client machines.
- Microsoft (or MS) Access is a software package that you install just like any other software package, and is bundled as part of the Microsoft Office suite.
  1. **Tables:** The tables are the backbone and the storage container of the data entered into the database.
  2. **Relationships:** Relationships are the links you build between the tables. They join tables that have associated elements.
  3. **Queries:** Queries are the means of manipulating the data to display in a form or a report. Queries can sort, calculate, group, filter, join tables, update data, delete data, etc. Their power is great. The Microsoft Access database query language is SQL (Structured Query Language).
  4. **Forms:** Forms are the primary interface through which the users of the database enter data.
  5. **Reports:** Reports are the results of the manipulation of the data you have entered into the database.
  6. **Macros:** Macros are spontaneous way for MS Access to carry out a series of actions for the database. Access gives you a selection of actions that are carried out in the order you enter.
  7. **Modules:** Modules are the basis of the programming language that supports Microsoft Access, The module window is where you can write and store Visual Basic for Applications (VBA). Advanced users of Microsoft Access tend to use VBA instead of Macros.

**VIII. MariaDB**

- MariaDB is a community-developed, free, popular open source relational database. It's latest development version is 10.5.0 released on 3rd December 2019.

The various features of MariaDB are as follows:

- It uses popular querying language.
- It supports PHP, one of the most popular free web scripting language.
- There are a number of companies who are using MariaDB.
- It runs on a number of operating systems.

## 1.2 DIFFERENCE BETWEEN RDBMS AND DBMS

S-17, S-19; W-18

| RDBMS | DBMS |
|---|---|
| • RDBMS is a relational database management system | • DBMS is a database management system. |
| • RDBMS follows normalization concept. | • DBMS does not follow the normalization. |
| • RDBMS has the major difference of solving the queries easily as they are stored in table format and use many functional keys in solving the queries. | • DBMS is mainly a storage area and it does not employ any tables for storing the data or does not use any special function keys or foreign keys for the retrieval of the data. |
| • RDBMS supports client/server Architecture. | • DBMS does not support client/server Architecture. |
| • RDBMS allows simultaneous access of users to the database. | • Only one user can access the database at a time in DBMS. |
| • RDBMS stand for Relational Database Management System. This is the most common form of DBMS. Invented by E.F. Codd, the only way to view the data is as a set of tables. Because there can be relationships between the tables. | • DBMS stands for Database Management System which is a general term for a set of software dedicated to controlling the storage of data. |
| • It is used to establish the relationship concept between two database objects, i.e., tables. | • In DBMS no relationship concept. |
| • It treats data as tables internally. | • It treats data as files internally. |
| • It requires High software and hardware requirements. | • It requires low Software and hardware requirements. |
| • Examples: (i) SQL-Server, (ii) Oracle | • Examples: (i) FoxPro, (ii) IMS |

## 1.3 RELATION AMONG DBMS AND APPLICATION PROGRAMS

- A database is information that is stored in a disk. The Database Management System (DBMS) is a set of programs; middleware which allows read and write access to the database.
- A database application is a set of programs that utilize the DBMS in order to store, retrieve, manipulate and report the data in the database. Enhanced DBMS's, such as D3, offer much more than just read/write capabilities.
- A programmer is an application developer who creates the applications for the benefit of the end users. The end user is one who sits in front of a keyboard and exercises the application.
- The application programmer must have the complete knowledge about database management system and databases used in it. Then he can easily develop the application program. The application program provides an easy and user-friendly user-interface to access the database.
- A user runs software, which is a database application, to manipulate data contained in a database. One or more databases are stored in a DBMS and they are subject to the rules enforced and features provided by that DBMS.
- Outside of the technical definitions, users should not have any knowledge of the underlying mechanics of the database applications.

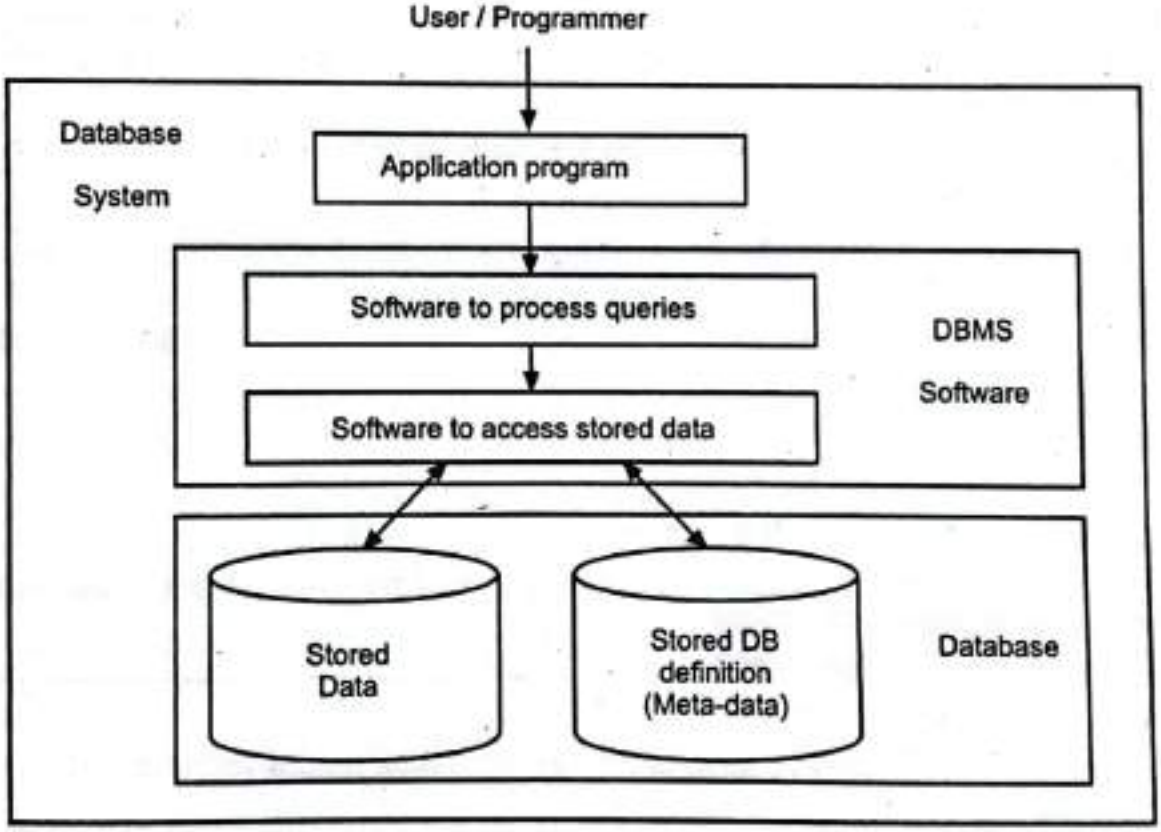



**Fig. 1.6: Relation between DBMS and Application Program**

- Application programs provides user interface. User uses this interface to perform different operations on database with the help of DBMS.
- User sends request to DBMS. After receiving this request DBMS processes it and sends results to the Application Program. User receives the results through application program.
- DBMS retrieves data from Database or performs other different operations on database according to the request send by the user.
- The Application Program is also known as front-end and Database is known as back-end.
- The term "front-end" refers to the user interface, while "back-end" means the server, application and database that work behind the scenes to deliver information to the user.
- Front-end developers work on what the user can see while back-end developers build the infrastructure that supports it.
- Front-end and back-end can also be used to describe situations where the customer has access to one view and employees have access to another. Front-end components are customer facing while rights to the back-end are exclusively for authenticated users.

## Summary

- RDBMS refers to a relational database plus software for managing users and processing SQL queries.
- The concept of RDBMS was invented by E. F. Codd in 1970 in his research paper.
- Primary key uniquely identifies a row in a table.
- Foreign key references to the value of a primary key of some other relation. It can have duplicate values.
- Some of the popular RDBMS products are: SQL Server, Oracle, MySQL, IBM DB2, MS-Access, PostgreSQL.
- Some RDBMS products are free while some requires license.
- RDBMS follows normalization concept.
- A database is information that is stored in a disk.
- The DBMS is a set of programs which will allow read and write access to the database.

## Check Your Understanding

**I. Multiple Choice Questions:**

1. Which of the following term is related with disk media failures and some user errors.

   (a) Security (b) Crash Recovery
   (c) Sharing (d) Integrity

2. You can specify rules that data must satisfy. Which of the following term is related with this statement?
   (a) Security (b) Sharing
   (c) Integrity (d) Extensibility
3. Who invented the concept of RDBMS.
   (a) E.F.Codd (b) Peter Chen
   (c) Gordon Everest (d) none of these
4. A ..... uniquely identifies each row in a table.
   (a) Foreign Key (b) both Primary Key and Foreign Key
   (c) Primary Key (d) none of these
5. The basic unit of data storage in a relational database is called a .....
   (a) table (b) column
   (c) row (d) none of these
6. There can be more than one ..... keys in a relation.
   (a) primary (b) both primary and candidate
   (c) candidate (d) none of these
7. ..... can have duplicate values.
   (a) Primary Key (b) Foreign Key
   (c) both Primary Key and Foreign Key
   (d) none of these
8. Which of the following is a free RDBMS product?
   (a) PostgreSQL (b) MS-Access
   (c) Oracle (d) none of these
9. Which of the following is a database product from IBM?
   (a) PostgreSQL (b) DB2
   (c) MySQL (d) none of these

**Ans.:** (1) b (2) c (3) a (4) c (5) a (6) c (7) b (8) a (9) b.

**II. State TRUE/FALSE.**

1. A database is a collection of related data elements.
2. Peter Chen invented relational model.
3. Integrity means you can specify rules that data must satisfy.
4. Crash Recovery means data can be protected against unauthorized read and write access.
5. Multiple users can access the database at the same time is called sharing between users.
6. RDBMS supports concurrency.
7. PostgreSQL is free and open source RDBMS.

8. MySQL is now under Oracle Corporation.
9. RDBMS supports client-server architecture.

**Answer:**

(1) TRUE (2) FALSE (3) TRUE (4) FALSE
(5) TRUE (6) TRUE (7) TRUE (8) TRUE
(9) TRUE

## Practice Questions

**Q.1 Answer the following questions in brief.**

1. What do you mean by database?
2. List various characteristics of RDBMS?
3. State advantages of RDBMS.
4. State disadvantages of RDBMS.
5. List the different RDBMS products.
6. Write short note on IBM DB2.
7. Write short note on SQL Server.
8. List the components of relational database.
9. State the difference between primary key and foreign key.

**Q.2 Answer the following questions.**

1. What are the characteristics of RDBMS?
2. What are the advantages and disadvantages of RDBMS?
3. What are the features of SQL Server?
4. What are the features of Oracle?
5. What are the features of MySQL?
6. Differentiate between RDBMS and DBMS.
7. Describe the relation between DBMS and application programs with suitab diagram.
8. Compare between MySQL and SQL Server.
9. What are components of RDBMS? Explain in detail.

**Q.3 Define the terms:**

(a) Database
(b) RDBMS
(c) Row
(d) Column
(e) Primary Key
(f) Foreign Key
(g) Candidate Key
(h) Super Key
(i) Table

## Previous Exams Questions

### Summer 2017

1. Give any two differences between DBMS and RDBMS. [2 M]

**Ans.** Please refer Section 1.2

2. Explain any four objects of oracle. [4 M]

**Ans.** Please refer Section 1.1.6.5.

### Winter 2017

1. What is RDBMS ? Enlist any two product of RDBMS. [2 M]

**Ans.** Please refer Section 1.1.6.5.

2. What are features of Oracle? [4 M]

**Ans.** Please refer Section 1.1.6.5.

### Summer 2018

1. What is RDBMS ? List any four characteristics of RDBMS. [2 M]

**Ans.** Please refer Section 1.1.3.

2. Explain advantages and disadvantages of RDBMS. [4 M]

**Ans.** Please refer Sections 1.1.4 and 1.1.5.

### Winter 2018

1. What is RDBMS? List the different products of RDBMS. [2 M]

**Ans.** Please refer Section 1.1.6.5.

2. Differentiate between DBMS and RDBMS. [4 M]

**Ans.** Please refer Section 1.2.

### Summer 2019

1. What is difference between DBMS and RDBMS? [2 M]

**Ans.** Please refer Section 1.2.

❖❖❖

# 2... PL/SQL

## Objectives...

- To understand basic building blocks of PL/SQL.
- To learn about Exception Handling.
- To learn about writing Functions, Procedures, Triggers and Packages.

## 2.1 OVERVIEW OF PL/SQL

- PL/SQL stands for Procedural Language/SQL.
- PL/SQL extends SQL by adding constructs found in procedural languages, resulting i a structural language that is more powerful than SQL.
- With PL/SQL, you can define and execute PL/SQL program units such as procedure functions, triggers, cursors and packages. PL/SQL program units generally ar categorized as anonymous blocks and stored procedures.
- PL/SQL supports variables, conditions, arrays and exceptions. Implementations fro version 8 of Oracle Database onwards have included features associated with objec orientation and some constructs such as loops.
- PL/SQL, however, as a complete procedural language that fills in these gaps, allow Oracle database developers to interface with the underlying relational database in a imperative manner.
- SQL statements can make explicit in-line calls to PL/SQL functions, or can caus PL/SQL triggers to fire upon pre-defined Data Manipulation Language (DML) events.

### 2.1.1 What is PL/SQL?

S-17,18; W-17,

- PL/SQL is Oracle's procedural language extension to SQL.
- PL/SQL allows you to mix SQL statements with procedural statements like I statement, Looping structures etc.
- PL/SQL is the superset of SQL. It uses SQL for data retrieval and manipulation an uses its own statements for data processing.
- PL/SQL program units are generally categorized as follows:
  1. Anonymous blocks and
  2. Stored procedures

1. **Anonymous block:** This is a PL/SQL block that appears within your application. In many applications PL/SQL blocks can appear where SQL statements can appear. Such blocks are called as Anonymous blocks.
2. **Stored Procedure:** This is a PL/SQL block that is stored in the database with a name. Application programs can execute these procedures using the name. Oracle also allows you to create functions, which are same as procedures but return a value and packages, which are a collection of procedures and functions.

- PL/SQL is a combination of SQL along with the procedural features of programming languages. It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

## 2.1.1 PL/SQL Engine (W-18)

- Oracle uses a PL/SQL engine to process the PL/SQL statements. A PL/SQL code can be stored in the client system (client-side) or in the database (server-side).
- Every PL/SQL block is first executed by PL/SQL engine.
- This is the engine that compiles and executes PL/SQL blocks.
- PL/SQL engine is available in Oracle Server and certain Oracle tools such as Oracle Forms and Oracle Reports etc.
- PL/SQL engine executes all procedural statements of a PL/SQL block, but sends SQL command to SQL Statement Executor in the Oracle RDBMS.
- That means PL/SQL separates SQL commands from PL/SQL commands and executes PL/SQL commands using Procedural Statement Executor, which is a part of PL/SQL engine, (Refer Fig. 2.1).

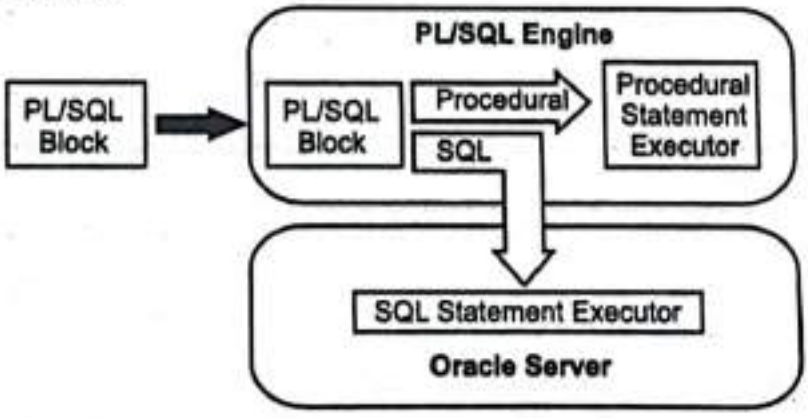



**Fig. 2.1: PL/SQL Engine**

## 2.1.3 Features of PL/SQL

- The important features of PL/SQL are listed below:

1. **Block structure:**

   PL/SQL is a block-structured language. Each program written in PL/SQL is written as a block. Blocks can also be nested. Each block is meant for a particular task.

2. **Variables and Constants:**
   PL/SQL allows you to declare variables and constants. Variables are used to st values temporarily. Variables and constants can be used in SQL and PL/S procedural statements just like an expression.

3. **Control structures:**
   PL/SQL allows control structures like IF statement, FOR loop, WHILE loop to used in the block. Control structures are most important extension to SQL PL/SQL. Control structures allow any data process possible in PL/SQL.

4. **Exception handling:**
   PL/SQL allows errors, called as exceptions, to be detected and handled. Whene there is a predefined error, PL/SQL raises an exception automatically. The exceptions can be handled to recover from errors.

5. **Modularity:**
   PL/SQL allows process to be divided into different modules. Subprograms call as procedures and functions can be defined and executed using the name. The subprograms can also take parameters.

6. **Cursors:**
   A cursor is a private SQL area used to execute SQL statements and sto processing information. PL/SQL implicitly uses cursors for all DML command and SELECT command that returns only one row. It also allows you to defi explicit cursor to deal with multiple row queries.

7. **Built-in functions:**
   Most of the SQL functions that we have seen so far in SQL are available in PL/SQ These functions can be used to manipulate variables of PL/SQL.

## 2.1.4 Advantages of PL/SQL

- PL/SQL is completely portable, high performance transaction processing languag that gives the following advantages.

1. **Block Structures:** PL/SQL consists of blocks of code which can be nested withi each other. Each block forms a unit of a task or a logical module. PL/SQL Block can be stored in the database and reused.
2. **Procedural Language Capability:** PL/SQL consists of procedural languag constructs such as conditional statements, (if else statements) and loops (FO WHILE loops etc.).
3. **Better Performance:** PL/SQL engine processes multiple SQL statement simultaneously as a single block, thereby reducing network traffic.
4. **Error Handling:** PL/SQL handles errors or exceptions effectively during th execution of a PL/SQL program. Once an exception is captured, specific actio can be taken depending upon the type of the exception or it can be displayed the user with a message.

5. PL/SQL supports both languages i.e. procedural language and object oriented language.
6. PL/SQL is high transaction processing language; it does not require and particular operating system to run.
7. PL/SQL having built-in libraries of packages.
8. PL/SQL is highly productive language, i.e. it works with the front-end to develop the application.
9. Performance of PL/SQL is high as in single line query entire block of statement can be processed.
10. PL/SQL also permits dealing with errors and facilitates displaying user-friendly messages when errors are found.

## 2.2 DATA TYPES (S-17; W- 17; 18)

- Every constant, variable, and parameter has a *datatype* (or *type*), which specifies a storage format, constraints and valid range of values.
- PL/SQL is Oracle's procedural language extension to SQL. PL/SQL allows you to mix SQL statements with procedural statements like IF statement, Looping structures etc.
- PL/SQL is the superset of SQL. It uses SQL for data retrieval and manipulation and uses its own statements for data processing.
- The data types is PL/SQL are :

**1. Character Datatypes :**

| Data Type | Description |
|---|---|
| char(size) | Where size is the number of characters to store. Fixed length strings. Space padded. |
| nchar(size) | Where size is the number of characters to store. Fixed length NLS string. Space padded. |
| nvarchar2(size) | Where size is the number of characters to store. Variable length NLS string. |
| varchar2(size) | Where size is the number of characters to store. Variable length string. |
| Long | Variable length strings. (backward compatible) |
| Raw | Variable length binary strings. |
| long raw | Variable length binary strings. (backward compatible) |

**2. Numeric Datatypes :**

| Data Type | Description |
|---|---|
| number(p,s) | Where p is the precision and s is the scale. |
| numeric(p,s) | To declare fixed-point numbers with a maximum prec of 38 decimal digits. |
| Double precision, Float | To declare floating-point numbers with a maxim precision of 126 binary digits, which is roughly equivale 38 decimal digits. |
| dec(p,s) | To declare fixed-point numbers with a maximum preci of 38 decimal digits. |
| decimal(p,s) | |
| integer,int,smallint | To declare integers with a maximum precision of 38 deci digits. |
| Real | To declare floating-point numbers with a maxim precision of 63 binary digits, which is roughly equivalen 18 decimal digits. |

**3. Date/Time Datatypes :**

| Data Type | Description |
|---|---|
| Date | To store fixed-length datetimes, which include the time day in seconds since midnight. |
| Timestamp(fractional seconds precision) | Includes year, month, day, hour, minute, and seconds. |
| timestamp(fractional seconds precision) with time zone | Includes year, month, day, hour, minute, and seconds wit time zone displacement value. |
| timestamp(fractional seconds precision) with local time zone | Includes year, month, day, hour, minute, and seconds wit time zone expressed as the session time zone. |
| interval year (year precision) to month | Time period stored in years and months. |
| interval day (day precision) to second (fractional seconds precision) | Time period stored in days, hours, minutes, and seconds. |

**4. Large Object (LOB) Datatypes :**

| Data Type | Description |
|---|---|
| BFILE | File locators that point to a binary file on the server file system(outside the database). |
| BOB | Stores unstructured binary large objects. |
| CLOB | Stores single-byte and multi-byte character data. |
| NCLOB | Stores Unicode data. |

**5. Rowid Datatypes :**

| Data Type | Description |
|---|---|
| Rowid | Fixed-length binary data. Every record in the database has a physical address or rowid. |
| urowid(size) | Universal rowid. Where size is optional. |

**1. Predefined Datatypes:**

- A *scalar* type has no internal components.
- A *composite* type has internal components that can be manipulated individually.
- A *reference* type holds values, called *pointers*, that designate other program items.
- A LOB type holds values, called LOB locators that specify the location of large objects (For example graphic images) stored out-of-line.

**2. User-Defined Subtypes:**

- Each PL/SQL base type specifies a set of values and a set of operations applicable to items of that type. Subtypes specify the same set of operations as their base type but only a subset of its values. Thus, a subtype does *not* introduce a new type; it merely places an optional constraint on its base type.
- Subtypes can increase reliability, provide compatibility with ANSI/ISO types, and improve readability by indicating the intended use of constants and variables.
- PL/SQL predefines several subtypes in package STANDARD. For example, PL/SQL predefines the subtypes CHARACTER and INTEGER as follows:

```
SUBTYPE CHARACTER IS CHAR;
SUBTYPE INTEGER IS NUMBER(38,0);  -- allows only whole numbers
```

- The subtype CHARACTER specifies the same set of values as its base type CHAR, so CHARACTER is an *unconstrained subtype*. But, the subtype INTEGER specifies only a subset of the values of its base type NUMBER, so INTEGER is a *constrained subtype*.

**(a) Defining Subtypes:**

- You can define your own subtypes in the declarative part of any PL/SQL block, subprogram, or package using the syntax:

```
SUBTYPE subtype_name IS base_type[(constraint)] [NOT NULL];
```

Where subtype_name is a type specifier used in subsequent declarations, base_type is any scalar or user-defined PL/SQL datatype and constraint applies only to types that can specify precision and scale or a maximum size.

Some examples follow:

```
DECLARE
   SUBTYPE BirthDate IS DATE NOT NULL;   -- based on DATE type
   SUBTYPE Counter IS NATURAL;           -- based on NATURAL subtype
   TYPE NameList IS TABLE OF VARCHAR2(10);
   SUBTYPE DutyRoster IS NameList;       -- based on TABLE type
   TYPE TimeRec IS RECORD (minutes INTEGER, hours INTEGER);
   SUBTYPE FinishTime IS TimeRec;        -- based on RECORD type
   SUBTYPE ID_Num IS emp.empno%TYPE;     -- based on column type
```

- You can use %TYPE or %ROWTYPE to specify the base type. When %TYPE provides the datatype of a database column, the subtype inherits the size constraint (if any) of the column. However, the subtype does *not* inherit other kinds of constraints such as NOT NULL.
- **Type Compatibility:** An unconstrained subtype is interchangeable with its base type. For example, given the following declarations, the value of amount can be assigned to total without conversion:

```
DECLARE
   SUBTYPE Accumulator IS NUMBER;
   amount NUMBER(7,2);
   total  Accumulator;
BEGIN
   ...
   total:= amount;
   ...
END;
```

**Datatype Conversion:**

- Sometimes, it is necessary to convert a value from one datatype to another. For example, if you want to examine a rowid, you must convert it to a character string.
- PL/SQL supports both explicit and implicit (automatic) datatype conversion.

1. **Explicit Conversion:** To convert values from one datatype to another, you use built-in functions. For example, to convert a CHAR value to a DATE or NUMBER value, you use the function TO_DATE or TO_NUMBER, respectively. Conversely, to convert a DATE or NUMBER value to a CHAR value, you use the function TO_CHAR.
2. Implicit Conversion: **When *it makes sense, PL/SQL can convert the datatype of a value implicitly. This lets you use literals, variables and parameters of a type where another type is expected. In the example below, the*** CHAR ***variab***

**start_time** and **finish_time** hold string values representing the number of seconds past midnight. The difference between those values must be assigned to the **NUMBER** variable **elapsed_time**. So, PL/SQL converts the **CHAR** values to **NUMBER** values automatically.

```
DECLARE
   start_time   CHAR(5);
   finish_time  CHAR(5);
   elapsed_time NUMBER(5);
BEGIN
   /* Get system time as seconds past midnight. */
   SELECT TO_CHAR(SYSDATE,'SSSSS') INTO start_time FROM sys.dual;
   -- do something
   /* Get system time again. */
   SELECT TO_CHAR(SYSDATE,'SSSSS') INTO finish_time FROM sys.dual;
   /* Compute elapsed time in seconds. */
   elapsed_time:= finish_time - start_time;
   INSERT INTO results VALUES (elapsed_time, ...);
END;
```

- Before assigning a selected column value to a variable, PL/SQL will, if necessary, convert the value from the datatype of the source column to the datatype of the variable. This happens, for example, when you select a DATE column value into a VARCHAR2 variable.
- Likewise, before assigning the value of a variable to a database column, PL/SQL will, if necessary, convert the value from the datatype of the variable to the datatype of the target column. If PL/SQL cannot determine which implicit conversion is needed, you get a compilation error. In such cases, you must use a datatype conversion function.

## 2.3 PL/SQL BLOCKS (W -17; S-18)

- The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other.
- Typically, each block performs a logical action in the program. A typical PL/SQL block shown in Fig. 2.2.
- Each program consists of SQL and statements which are from a PL/SQL block.
- A PL/SQL Block consists of three sections:
  1. The Declaration section (optional).
  2. The Execution section (mandatory).
  3. The Exception (or Error) Handling section (optional).

**1. Declaration Section:**

The Declaration section of a PL/SQL Block starts with the reserved keyword DECLARE.

This section is optional and is used to declare any placeholders like variable constants, records and cursors, which are used to manipulate data in the executi section. Placeholders may be any of Variables, Constants and Records, which store data temporarily. Cursors are also declared in this section.

2. **Execution Section:**
The Execution section of a PL/SQL Block starts with the reserved keyword BEGIN an ends with END. This is a mandatory section and is the section where the progra logic is written to perform any task. The programmatic constructs like loop conditional statements and SQL statements form the part of execution section.

3. **Exception Section:**
The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTIO This section is optional. Any errors in the program can be handled in this section, that the PL/SQL Blocks terminates gracefully. If the PL/SQL Block contains exceptio that cannot be handled, the Block terminates suddenly with errors.

Every statement in the above three sections must end with a semicolon;. PL/SQ blocks can be nested within other PL/SQL blocks. Comments can be used to docume code.

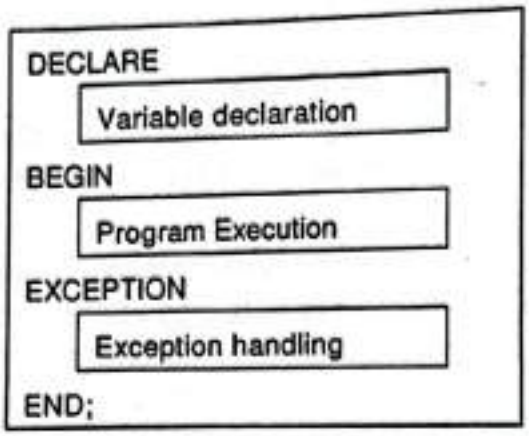



**Fig. 2.2: PL/SQL Block**

- For example:

```
declare
   v_rollno students.rollno%type;
begin
   -- get roll number of the students who joined most recently
   select max(rollno) into v_rollno;
   -- insert a new row into payments table
   insert into payments values (v_rollno, sysdate,1000);
   -- commit transaction
   commit;
end;
/
```

- Follow the procedure given below to create and run the above block.
  1. Type the above program in a text editor such as Notepad and save it in a text file. Assume the file is saved under the name INSPAY.SQL. And be sure to know the directory where the file is saved.
  2. Get into SQL*PLUS. Start it and logon if you have not already logged on.
  3. Use START command to execute the program that is in the file `inspay.sql`.

     ```
     SQL> start c:\orabook\inspay.sql
     ```

     PL/SQL procedure successfully completed.
- If the block is successfully executed then PL/SQL displays the above message. If there are any errors during the execution then you have to correct the program, save the program and rerun it until you succeed.

**Variables in PL/SQL:**

- Information transfer between PL/SQL program and database through variable.
- The general syntax to declare a variable is:

  ```
  variable_name Datatype[Size] [NOT NULL] := [ value ];
  ```
- Every variable stores certain type of values. Every variable has a specific type associated with it. These types are:
  1. One of the types used by SQL for database columns.
  2. A generic type used in PL/SQL such as NUMBER.
  3. Declared to be the same as the type of some database column.

**Values for declaring a variable:**

1. Variables must be declared first before the usage.
2. Only TRUE or FALSE can be assigned to BOOLEAN type of variable.
3. Attribute TYPE can be used to define a variable which is of type same as a database column's type definition.
4. For customization the variable type user can use TYPE ....IS statement.

**Scope of Variables:**

- PL/SQL allows the nesting of blocks within blocks i.e, the Execution section of an outer block can contain inner blocks. Therefore, a variable which is accessible to an outer block is also accessible to all nested inner blocks.
- The variables declared in the inner blocks are not accessible to outer blocks. Based on their declaration we can classify variables into two types.
  1. **Local variables:** These are declared in a inner block and cannot be referenced by outside blocks.
  2. **Global variables:** These are declared in a outer block and can be referenced by its itself and by its inner blocks.
- For Example: In the following example, we are creating two variables in the outer block and assigning their product to the third variable created in the inner block. The

variable 'var_ mult' is declared in the inner block, so cannot be accessed in the outer block i.e. it cannot be accessed after line 11. The variables 'var_num1' and 'var_num2' can be accessed anywhere in the block.

```
DECLARE
   var_num1 number;
   var_num2 number;
BEGIN
   var_num1:= 100;
   var_num2:= 200;
   DECLARE
    var_mult number;
    BEGIN
     var_mult:= var_num1  * var_num2;
    END;
   END;
   /
```

**Compiling and Executing a Simple block:**

- After typing code, type slash (/), SQL *plus sends your code to oracle for execution. After execution, your output displays line.
- PL/SQL procedure successfully completed.
- PL/SQL is always used with some other program or tool that handles input, output and other user interaction.
- Oracle now includes the DBMS_OUTPUT package with PL/SQL.

```
Declare
   a Number;
Begin
   a: = 5294;
   dbms_output.put_line('value of variable is =');
   dbms_output.put_line(a);
End;
/
```

- The dbms_out.put_line( ) procedure takes exactly one argument and output a line of text from the database server.
- To see this line of text as output use command.

```
SQL > SET SERVEROUTPUT ON
```

**Output:**

```
Value of variable is =
   5294
```

## Character set in PL/SQL:

- The basic character set includes the following:
  1. Uppercase alphabets (A – Z)
  2. Lowercase alphatets (a – z)
  3. Numerals (0 – 9)
  4. Symbols ( ) + – */ < > =! ;: . ' (a) %, "# $ ^ & – \ { }? [ ]
- Words used in a PL/SQL block are called **Lexical Units.** Blank spaces can be freely inserted between lexical units in a PL/SQL block.

**Literals:** A literal is a numeric value or a character string used to represent itself.

1. **Numeric Literal:** These can be either integers or floats.
   For example: `25, 6.33, +702, -6.`
2. **String Literal:** They are represented by one or more legal characters enclosed within single quotes.
   For example: `'Hello', 'Isn"t it'`
3. **Character Literal:** Character literals consisting of single characters.
   For example: `'*', 'B'`
4. **Logical Literal (Boolean Literal):** These are predetermined constants. The values that can be assigned to this data type are: `TRUE, FALSE, NULL.`

## PL/SQL Constants

- As the name implies a constant is a value used in a PL/SQL Block that remains unchanged throughout the program.
- A constant is a user-defined literal value. You can declare a constant and use it instead of actual value.
- For example: If you want to write a program which will increase the salary of the employees by 25%, you can declare a constant and use it throughout the program, Next time when you want to increase the salary again you can change the value of the constant which will be easier than changing the actual value throughout the program.
- The General Syntax to declare a constant is:

```
Constant_name CONSTANT datatype:= VALUE;
```

  - Constant_name is the name of the constant i.e. similar to a variable name.
  - The word CONSTANT is a reserved word and ensures that the value does not change.
  - VALUE - It is a value which must be assigned to a constant when it is declared. You cannot assign a value later.
- For example, to declare salary_increase, you can write code as follows:

```
DECLARE
salary_increase CONSTANT number(3):= 10;
```

- You must assign a value to a constant at the time you declare it. If you do not assign value to a constant while declaring it and try to assign a value in the executable section, you will get a error. If you execute the following PL/SQL block you will get error.

```
DECLARE
   Salary_increase CONSTANT number(3);
BEGIN
   salary_increase:= 100;
   dbms_output.put_line (salary-increase);
END;
```

## 2.3.1 %type, %rowtype

- Assign the same type to variable as that of the relation column declared in database. If there is any type mismatch, variable assignments and comparisons may not work the way you expect, so instead of hard coding the type of a variable, you should use the %TYPE operator.

**For example:**

```
DECLARE
 my_name emp.ename%TYPE;
```

gives PL/SQL variable `my_name` whatever type was declared for the ename column in emp table.

- The `%TYPE` attribute provides the datatype of a variable or database column. This is particularly useful when declaring variables that will hold database values. For example, assume there is a column named `title` in a table named `books`. To declare a variable named `my_title` that has the same datatype as column `title`, use dot notation and the `%TYPE` attribute, as follows:

```
my_title books.title%TYPE;
```

- Declaring `my_title` with `%TYPE` has two advantages. First, you need not know the exact datatype of `title`. Second, if you change the database definition of `title` (make it a longer character string for example), the datatype of `my_title` changes accordingly at run time.

**%rowtype:**

- A variable can be declared with `%rowtype` that is equivalent to a row of a table i.e. record with several fields. The result is a record type in which the fields have the same names and types as the attributes of the relation.

**For example:**

```
DECLARE
 Emp_rec  emp1%rowtype;
```

- This makes variable `emp_rec` be a record with fields `name` and `salary`, assuming that the relation has the schema emp1(name, salary).

- The initial value of any variable, regardless of its type, is NULL.
- In PL/SQL, records are used to group data. A record consists of a number of related fields in which data values can be stored. The %ROWTYPE attribute provides a record type that represents a row in a table. The record can store an entire row of data selected from the table or fetched from a cursor or cursor variable.
- Columns in a row and corresponding fields in a record have the same names and datatypes. In the example below, you declare a record named dept_rec. Its fields have the same names and datatypes as the columns in the dept table.

```
DECLARE
  dept_rec dept%ROWTYPE;  -- declare record variable
```

You use dot notation to reference fields, as the following example shows:

```
my_deptno:= dept_rec.deptno;
```

- If you declare a cursor that retrieves the last name, salary, hire date, and job title of an employee, you can use %ROWTYPE to declare a record that stores the same information, as follows:

```
DECLARE
   CURSOR c1 IS
      SELECT ename, sal, hiredate, job FROM emp;
emp_rec c1%ROWTYPE; -- declare record variable that represents
                    -- a row fetched from the emp table
```

- When you execute the statement,

```
FETCH c1 INTO emp_rec;
```

the value in the ename column of the emp table is assigned to the ename field of emp_rec, the value in the sal column is assigned to the sal field, and so on.

emp_rec

| | |
|---|---|
| emp_rec.ename | JAMES |
| emp_rec.sal | 950.00 |
| emp_rec.hiredate | 03-DEC-95 |
| emp_rec.job | CLERK |

**Program 2.1: Accept the deptno and print the number of employees working in that department.**

```
Declare
   v_deptno emp.deptno%type;
   v_count number;
Begin
   v_deptno:=&v_deptno;
   select count(*) into v_count
   from emp
   where deptno=v_deptno;
```

```
    dbms_output.put_line('No. of emp working in '|| v_deptno
                                                 'are' || v_cou

End;
/
```

**Output:**

```
SQL> /
Enter value for v_deptno: 20
old 5: v_deptno:=&v_deptno;
new 5: v_deptno:=20;
No. of emp working in 20 are 2
```

**Program 2.2: Accept the deptno and print the department name and location.**

```
Declare
   v_deptno dept.deptno%type;
   v_dname dept.dname%type;
   v_loc dept.loc%type;
Begin
   v_deptno:=&v_deptno;
   select dname,loc into v_dname,v_loc
   from dept
   where deptno=v_deptno;
   dbms_output.put_line('Department name is '|| v_dname ||'
       location is '|| v_loc);
End;
/
```

**Output:**

```
SQL> /
Enter value for v_deptno: 10
old 6: v_deptno:=&v_deptno;
new 6: v_deptno:= 10;
Department name is ACCOUNTING and location is NEW YORK
```

**Program 2.3: Print name of employee having maximum salary. Also print salary.**

```
Declare
   v_name emp.ename%type;
   v_sal emp.sal%type;
Begin
   select ename,sal into v_name,v_sal
   from emp
   where sal = (select max(sal) from emp);
   dbms_output.put_line(v_name ||' is having maximum salary
       '|| v_sal);
End;
/
```

**Output:**

```
SQL>/
KING is having maximum salary = 5000
```

**Program 2.4: Accept employee name and print date of joined.**

```
Declare
   v_name emp.ename%type;
   v_date date;
Begin
   v_name:='&v_name';
   select hiredate into v_date from emp
   where ename=v_name;
   dbms_output.put_line('Date of joined of '|| v_name || 'is'
       || v_date);
End;
/
```

**Output:**

```
SQL> /
Enter value for v_name: KING
old 5: v_name:='&v_name';
new 5: v_name:='KING';
Date of joined of KING is 17-NOV-81
```

**Program 2.5: Accept salary and print number of employees having salary less than or equal to accepted salary**

```
Declare
   v_sal emp.sal%type;
   v_count number;
Begin
   v_sal:=&v_sal;
   select count(*) into v_count
   from emp
   where sal<=v_sal;
   dbms_output.put_line('  No.of  employees  having  salary  less
     than or equal to '|| v_sal || ' are ' || v_count);
End;
/
```

**Output:**

```
SQL> /
Enter value for v_sal: 3000
old 5: v_sal:=&v_sal;
new 5: v_sal:=3000;
No. of employees having salary less than or equal to 3000 are 8.
```

## 2.3.2 Operators and Functions in PL/SQL

**Operators :**

- PL/SQL operators are either unary or binary. Binary operators act on two values. A example of binary operators is the addition operator, which adds two numbe together. Unary operators only operate on one value.
- The **negation** operator is unary. PL/SQL operators can be divided into t following categories:
  1. Arithmetic operators
  2. Comparison operators
  3. Relational operators
  4. Logical operators
  5. String operators

**1. Arithmetic Operators:**

Arithmetic operators are used for mathematical computations.

| | | |
|---|---|---|
| ** | 10**5 | The exponentiation operator. 10**5 = 100,000. |
| * | 2*3 | The multiplication operator. 2 * 3 = 6. |
| / | 6/2 | The division operator. 6/2 = 3. |
| + | 2+2 | The addition operator. 2+2 = 4. |
| - | 4-2 | The subtraction operator. 4 -2 = 2. |
| - | -5 | The negation operator. |
| + | +5 | It complements the negation operator. |

**For example:**

```
SQL>
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
       dbms_output.put_line(4 * 2);   --multiplication
       dbms_output.put_line(24 / 3); --division
       dbms_output.put_line(4 + 4);   --addition
       dbms_output.put_line(16 - 8); --subtraction
end;
/
```

**Exponentiation:**

```
SQL>
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
       dbms_output.put_line(4 ** 2);
end;
/
```

**2. Comparison Operators:**

- Comparison operators are used to compare one value or expression to another.
- All comparison operators return a boolean result.

| Operator | Example | *Usage* |
|---|---|---|
| = | IF A = B THEN | The equality operator. |
| <> | IF A <> B THEN | The inequality operator. |
| != | IF A != B THEN | Another inequality operator, synonymous with <>. |
| ~= | IF A ~= B THEN | Another inequality operator, synonymous with <>. |
| < | IF A < B THEN | The less than operator. |
| > | IF A > B THEN | The greater than operator. |
| <= | IF A <= B THEN | The less than or equal to operator. |
| >= | IF A >= B THEN | The greater than or equal to operator. |
| LIKE | IF A LIKE B THEN | The pattern-matching operator. |
| BETWEEN | IF A BETWEEN B AND C THEN | Checks to see if a value lies within a specified range of values. |
| IN | IF A IN (B,C,D) THEN | Checks to see if a value lies within a specified list of values. |
| IS NULL | IF A IS NULL THEN | Checks to see if a value is null. |

**3. Relational Operators:**

=, <>, !=, ~=, <, >, <=, >=

String comparisons are case-sensitive.

String comparisons are dependent on the character set being used.

String comparisons are affected by the underlying datatype.

Comparing two values as CHAR strings might yield different results than the same values compared as VARCHAR2 strings.

It's important to remember that Oracle dates contain a time component.

**Example:**

| True Expressions | False Expressions |
|---|---|
| 5 = 5 | 5 = 3 |
| 'AAAA' = 'AAAA' | 'AAAA ' = 'AAAA' |
| 5 != 3 | 5 <> 5 |

*Contd...*

| | |
|---|---|
| 'AAAA ' ~= 'AAAA' | 'AAAA' ~= 'AAAA' |
| 10 < 200 | 10.1 < 10.05 |
| 'Jeff' < 'Jenny' | 'jeff' < 'Jeff' |
| TO_DATE('15-Nov-61' < '15-Nov-97') | TO_DATE('1-Jan-97' < '1-Jan-96') |
| 'A' <= 'B' | 'B' <= 'A' |
| TO_DATE('1-Jan-17') <= TO_DATE('1-Jan-17) | TO_DATE('15-Nov-91') <= TO_DATE('15-Nov-90) |

**4. Logical Operators:**

- PL/SQL has three logical operators: AND, OR and NOT.
- The NOT operator is typically used to negate the result of a comparison expressio The AND and OR operators are typically used to link together multiple comparisons.
- The Syntax for the NOT Operator:

  `NOT boolean_expression`

  boolean_expression can be any expression resulting in a boolean, or true/false valu
- The Syntax for the AND Operator:

  `boolean_expression AND boolean_expression`

  `boolean_expression` can be any expression resulting in a boolean or true/fals value. The AND operator returns a value of true if both expressions evaluate to tru otherwise, a value of false is returned.

| Expression | Result |
|---|---|
| (5 = 5) AND (4 < 10) AND (2 >= 2) | true |
| (5 = 7) AND (5 = 5) | false |
| 'Mon' IN ('Sun','Sat') AND (2 = 2) | false |

- The Syntax for the OR Operator:

  `boolean_expression OR boolean_expression`

  `boolean_expression` can be any expression resulting in a boolean, or true/fals value. The OR operator returns a value of true if any one of the expressions evaluate to true.
- A value of false is returned only if both the expressions evaluate to false.

| Expression | Result |
|---|---|
| (5 <> 5) OR (4 >= 100) OR (2 < 2) | false |
| (7 = 4) OR (5 = 5) | true |
| 'Mon' IN ('Sun','Sat') OR (2 = 2) | true |

**Logical Operators in PL/SQL:**

| x | y | x AND y | x OR y | NOT x |
|---|---|---|---|---|
| True | True | True | True | False |
| True | False | False | True | False |
| False | True | False | True | True |
| False | False | False | False | True |

**5. String Operators:**

- PL/SQL has two operators specifically designed to operate only on character string data.
- These are the LIKE operator and the concatenation (||) operator.
- The Syntax for the Concatenation Operator:

```
string_1 || string_2
```

string_1 and string_2 are both character strings and can be string constants, string variables, or string expressions.

**For example:**

```
SQL>
SQL> SET SERVEROUTPUT ON
SQL>
SQL> declare
     a     varchar2(30);
     b     varchar2(30);
     c     varchar2(30);
    begin
     c:= 'A' || ' AND ' || 'B';
     dbms_output.put_line(c);
     a:= ' C ';
     b:= ' D ';
     dbms_output.put_line(a || ' ' || b || ',');
     a:= ' E ';
     b:= ' F';
     c:= a || b;
     dbms_output.put_line(c);
   END;
   /
A AND B
C   D ,
E  F
```

### 6. Numeric Functions

All number functions return a numeric values unless otherwise specifie number functions are given below :

| Function | Purpose |
|---|---|
| ABS | Gives absolute value of n |
| CEIL | Gives smallest integer >=n |
| COS | Gives cosine of angle. |
| COSH | Gives hyperbolic cosine of n |
| EXP | Gives value of $e^n$ |
| FLOOR | Gives largest integer <=n |
| LN | Gives natural logarithm of n where n>0 |
| LOG | Gives base-m log of n where m>1 and n>0 |
| MOD | Gives remainder of m/n |
| POWER | Gives value of $m^n$ |
| ROUND | Gives m rounded to n places |
| SIGN | Gives -1 for n<0, 0 for n=0 and 1 for n>0 |
| SIN | Gives sine of angle |
| SINH | Gives hyperbolic sine of n |
| SQRT | Gives square root of n |
| TAN | Gives tangent of angle |
| TANH | Gives hyperbolic tangent of n |
| TRUNC | Gives m truncated to n places |

### 7. Character Functions

| Function | Purpose |
|---|---|
| ASCII | Gives ASCII value of a character |
| CHR | Gives character for ASCII value |
| CONCAT | Gives string2 appended to string1 |
| INITCAP | Gives string1 with the first letter of each word in capitals and in small |
| INSTR | Gives starting position of string2 in string1. Search starts at for the *nth* occurrence. If *pos* is negative, the search is perform in backward direction. Both *pos* and *n* default to 1. The funct gives 0 if string2 not found. |

| | |
|---|---|
| INSTRB | Same like INSTR except *pos* is a byte position. |
| LENGTH | Gives character count in *str* and for data type CHAR, length contains trailing blanks. |
| LENGTHB | Same like LENGTH, gives byte count of *str* containing trailing blanks for CHAR |
| LOWER | Gives *str* with all letters in lowercase. |
| LPAD | Pads left-side string to specified length with the character mentioned. |
| LTRIM | Gives string with leading spaces(or characters mentioned in set) removed. |
| NLS_INITCAP | Same like INITCAP except a sort sequence is specified. |
| NLS_LOWER | Same like LOWER except a sort sequence is specified. |
| NLS_UPPER | Same like UPPER except a sort sequence is specified. |
| NLSSORT | Gives string in sort sequence specified. |
| REPLACE | Gives string1 with all occurrences of string2 replaced by string3. If string3 is not given, then all occurrences of string2 are removed. |
| RPAD | Pads right-side string to specified length with the character mentioned. |
| RTRIM | Gives string with trailing spaces(or characters mentioned in set) removed. |
| SOUNDEX | Gives phonetic representation of string. |
| SUBSTR | Gives substring of string starting at specified position for the length mentioned or to the end of string is length is not given. |
| SUBSTRB | Same like SUBSTR except works on bytes. |
| TRANSLATE | It replaces all occurrences of set1 with set2 characters in string. |
| UPPER | Gives all letters of the string in uppercase. |

**8. Date Functions**

All date functions except MONTHS_BETWEEN return a DATE value.

| Function | Purpose |
|---|---|
| ADD_MONTHS | Adds or subtract the specified number of months from date. |
| LAST_DAY | Gives last day of the month for given date. |
| MONTHS_BETWEEN | Gives the number of months between date1 and date2. |
| NEW_TIME | This function converts a date from time zone 1 to a date in time zone2. |

*Contd..*

| | |
|---|---|
| NEXT_DAY | Gives first day of the week for day that is greater than sp date |
| ROUND | Gives date rounded to specified unit in format. If no for specified, date is rounded to the nearest day. |
| SYSDATE | Gives current system date and time. |
| TRUNC | Gives date with the time of day truncated as specified by f |

**9. Conversion and Other Miscellaneous Functions**

| Function | Purpose |
|---|---|
| TO_CHAR (Date) | Changes date to varchar2 based on format. |
| TO_CHAR (Number) | Changes number to varchar2 based on format. |
| TO_DATE | Changes string or number to date value based on format. |
| TO_NUMBER | Changes string to number value as per the given format. |

Here are few examples based on above functions.

```
1.  SELECT ABS(-15) "Absolute" FROM DUAL;
    Absolute
    ----------
    15
2.  SELECT MOD(15,4) "Modulo" FROM DUAL;
    Modulo
    ----------
     3
3.  SELECT SQRT(121) FROM DUAL;
    SQRT(121)
    ----------
       11
4.  SELECT ASCII('A') FROM DUAL;
    ASCII('A')
    ----------
       65
5.  SELECT INITCAP('how are you') "init" FROM DUAL;
       init
    ----------
    How Are You
6.  SELECT INSTR('WHERE ARE YOU GOING','RE') "FIRST" FROM DUAL;
    FIRST
    ---------
      4
7.  SELECT INSTR('WHERE ARE YOU GOING','RE',1,2) "SECOND" FROM DUAL;
       SECOND
    ---------
      8
```

```
8.  SELECT LTRIM('****WHAT IS GOING','*') "LTRIM" FROM DUAL;
     LTRIM
    -------------
    WHAT IS GOING
9.  SELECT RTRIM('WHAT IS GOING--*#','*#') "RTRIM" FROM DUAL;
     RTRIM
    ---------------
    WHAT IS GOING--
10. (If sysdate is 17-MAY-14)
    SELECT ADD_MONTHS(SYSDATE,1) "NEXT" FROM DUAL;
     NEXT
    ---------
    17-JUN-14
11. (If sysdate is 17-MAY-14)
    SELECT LAST_DAY(SYSDATE) "LAST" FROM DUAL;
     LAST
    ---------
    31-MAY-14
12. To_CHAR(1234.56, '9999.9')
    Result: '1234.5'
13. TO_CHAR(Sysdate, 'Month DD, YYYY')
    Result : 'July 11, 2019'
```

## 2.3.3 Control Statements (S-17)

- Control structures are the most important PL/SQL extension to SQL. Not only does PL/SQL let you manipulate Oracle data but also it lets you process the data using conditional, iterative and sequential flow-of-control statements such as IF-THEN-ELSE, FOR-LOOP, WHILE-LOOP, EXIT-WHEN and GOTO. Collectively, these statements can handle any situation.

### 2.3.3.1 Conditional Statement in PL/SQL

- Often, it is necessary to take alternative actions depending on circumstances. The IF-THEN-ELSE statement lets you execute a sequence of statements conditionally. The IF clause checks a condition; the THEN clause defines what to do if the condition is true; the ELSE clause defines what to do if the condition is false or null.

**1. IF-THEN Statement:**

- The simplest form of IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF (not ENDIF), as follows:

```
IF condition THEN
   sequence_of_statements
END IF;
```

- The sequence of statements is executed only if the condition is true. If the condition is false or null, the IF statement does nothing. In either case, control passes to the next statement.

- An example follows:

```
IF sales > quota THEN
   compute_bonus(empid);
   UPDATE payroll SET pay = pay + bonus WHERE empno = emp_id;
END IF;
```

- You might want to place brief IF statements on a single line, as in

```
IF x > y THEN
high:= x;
END IF;
```

**2. IF-THEN-ELSE Statement:**

- The second form of IF statement adds the keyword ELSE followed by an alter sequence of statements, as follows:

```
IF condition THEN
   sequence_of_statements1
ELSE
   sequence_of_statements2
END IF;
```

- The sequence of statements in the ELSE clause is executed only if the condit false or null. Thus, the ELSE clause ensures that a sequence of statements is exe In the following example, the first UPDATE statement is executed when the cond is true, but the second UPDATE statement is executed when the condition is fal null:

```
IF trans_type = 'CR' THEN
   UPDATE accounts SET balance = balance + credit WHERE ...
ELSE
   UPDATE accounts SET balance = balance - debit WHERE ...
END IF;
```

- The THEN and ELSE clauses can include IF statements. That is, IF statements c nested, as the following example shows:

```
IF trans_type = 'CR' THEN
   UPDATE accounts SET balance = balance + credit WHERE ...
ELSE
   IF new_balance >= minimum_balance THEN
      UPDATE accounts SET balance = balance - debit WHERE ...
   ELSE
      RAISE insufficient_funds;
   END IF;
END IF;
```

**3. IF-THEN-ELSIF Statement:**

- Sometimes you want to select an action from several mutually exclusive alternati
- The third form of IF statement uses the keyword ELSIF (not ELSEIF) to introd

additional conditions, as follows:

```
IF condition1 THEN
   sequence_of_statements1
ELSIF condition2 THEN
   sequence_of_statements2
ELSE
   sequence_of_statements3
END IF;
```

- If the first condition is false or null, the ELSIF clause tests another condition. An IF statement can have any number of ELSIF clauses; the final ELSE clause is optional.
- Conditions are evaluated one by one from top to bottom. If any condition is true, its associated sequence of statements is executed and control passes to the next statement.
- If all conditions are false or null, the sequence in the ELSE clause is executed. Consider the following example:

```
BEGIN
   ...
   IF sales > 50000 THEN
      bonus:= 1500;
   ELSIF sales > 35000 THEN
      bonus:= 500;
   ELSE
      bonus:= 100;
   END IF;
   INSERT INTO payroll VALUES (emp_id, bonus, ...);
END;
```

- If the value of sales is larger than 50000, the first and second conditions are true. Nevertheless, bonus is assigned the proper value of 1500 because the second condition is never tested. When the first condition is true, its associated statement is executed and control passes to the INSERT statement.

**Program 2.6: Program to find largest of two numbers.**

```
DECLARE
   A NUMBER;
   B NUMBER;
BEGIN
   A:=&a;
   B:=&b;
   If (A>B) THEN
   Dbms_output.put_line ('A IS LARGEST');
```

```
    ELSE
    Dbms_output.put_line ('B IS LARGEST');
    END IF;
END;
/
SQL>/
Enter value for A:6
Old 5: A:=&a;
New 5: A:=6
Enter value for B:7
Old 6: B:=&b
New 5: B:=7
B IS LARGEST
PL/SQL procedure successfully completed.
```

**Program 2.7: Display whether the salary of Johnson is 50000 or not.**

```
DECLARE
Jsal EMP.Sal%type;
BEGIN
   SELECT Sal INTO Jsal
   From EMP
   Where LastName='Johnson';
   IF(Jsal==50000) Then
   Dbms_output.put_line ('Salary of Johnson is 50000');
   ELSE
   Dbms_output.put_line ('Salary of Johnson is not 50000');
   END IF;
END;
/
SQL>/
Salary of Johnson is 50000
PL/SQL procedure successfully completed.
```

**Program 2.8: Program for calculation of net salary.**

```
DECLARE
   ename varchar2(15); /*size of ename is 15*/
   basic number;
   da number;
   hra number;
   pf number;
   netsalary number;
BEGIN
   ename: = &ename;
```

```
    basic: = &basic;
    da: = basic * (41/100);
    hra: = basic * (15/100);
    IF (basic < 3000) THEN
    pf: = basic * (5/100);
    ELSIF (basic >= 3000 and basic <= 5000) THEN
    pf: = basic * (7/100);
    ELSIF (basic >= 5000 and basic <= 8000) THEN
    pf: = basic * (8/100);
    ELSE
    pf: = basic * (10/100);
    END IF;
    netsalary: = basic + da + hra - pf;
    dbms_output.put_line('Employee name: ' || ename);
    dbms_output.put_line('Providend Fund: ' || pf);
    dbms_output.put_line('Net salary: ' || netsalary);
End;
```

**Program 2.9: Program to find maximum of three numbers.**

```
DECLARE
    a number;
    b number;
    c number;
BEGIN
    dbms_output.put_line('Enter a:');
    a:=&a;
    dbms_output.put_line('Enter b:');
    b:=&b;
    dbms_output.put_line('Enter c:');
    c:=&c;
    IF (a>b) and (a>c) THEN
    dbms_output.putline('A is Maximum');
    ELSIF (b>a) and (b>c) THEN
    dbms_output.putline('B is Maximum');
    ELSE
    dbms_output.putline('C is Maximum');
    END IF;
    END;
```

### 2.3.3.2 Iterative Statements in PL/SQL

- PL/SQL LOOP statement is an iterative control statement.
- An iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times.

- There are following types of loops in PL/SQL:
  1. Simple Loop
  2. While Loop
  3. For Loop

**1. Simple Loop:**

- A simple loop is used when a set of statements is to be executed at least once be the loop terminates.
- An EXIT condition must be specified in the loop, otherwise the loop will get into infinite number of iterations.
- When the EXIT condition is satisfied the process terminates from the loop.
- The General Syntax to write a Simple Loop is:

```
LOOP
    statements;
    EXIT;
    or EXIT WHEN condition;
END LOOP;
```

- These are the important steps to be followed while using Simple Loop.
  1. Initialise a variable before the loop body.
  2. Increment the variable in the loop.
  3. Use a EXIT WHEN statement to exit from the Loop. If you use a EXIT statem without WHEN condition the statements in the loop is executed only once.

**Program 2.10: Insert each of the pairs (1, 1) upto (100, 100) into table T1.**

```
DECLARE
    i NUMBER:= 1;
BEGIN
    LOOP
    INSERT INTO T1 VALUES(i,i);
    i:= i+1;
    EXIT WHEN i>100;
    END LOOP;
END;
```

**Program 2.11: Print the numbers 1 to 5.**

```
DECLARE
    i NUMBER:=0;
BEGIN
    LOOP
    i:=i+1;
    Dbms_output.put_line(i);
    IF(i>=5) THEN
    EXIT;
```

```
    END IF;
    END LOOP;
END;
/
SQL>/
1
2
3
4
5
```

**Program 2.12: Print numbers 1 to 5 using EXIT WHEN <condition>.**

```
DECLARE
    i NUMBER:=0;
BEGIN
    LOOP
    i:=i+1;
    Dbms_output.put_line(i);
    EXIT WHEN i>=5;
    END LOOP;
END;
/
SQL>/
1
2
3
4
5
```

**2. While Loop:**

- A WHILE LOOP is used when a set of statements has to re-executed as long as a condition is true.
- The condition is evaluated at the beginning of each iteration. The iteration continues until the condition becomes false.
- The General Syntax to write a WHILE LOOP is:

```
WHILE <condition>
LOOP statements;
END LOOP;
```

- Important steps to follow when executing a while loop:
  1. Initialise a variable before the loop body.
  2. Increment the variable in the loop.
  3. EXIT WHEN statement and EXIT statements can be used in while loops but it's not done oftenly.

**Program 2.13: Print numbers 1 to 5.**

```
DECLARE
   i number:=0;
BEGIN
   WHILE i<=5 LOOP
   i:=i+1;
   dbms_output.put_line(i);
   END LOOP;
END;
/
SQL>/
1
2
3
4
5
```

**Program 2.14: Program to find sum of odd numbers from 1 to 100. (Hint: value = 100)**

```
DECLARE
   n NUMBER;
   value NUMBER;
   sum NUMBER default 0;
BEGIN
   value:=&value;
   n:=1;
   WHILE (n < value)
   LOOP
   sum:=sum+n;
   n:=n+2;
   END LOOP;
   dbms_output.put_line('Sum of odd numbers between 1 and ' ||
                                          endvalue || ' is ' || sum);
END;
/
```

**3. For Loop**

- A FOR Loop is used to execute a set of statements for a predtermined number of times. Iteration occurs between the start and end integer values.
- The counter is always started by 1. The loop exits when the counter reaches the value of the end integer.
- The General Syntax to write a FOR LOOP is:

  FOR counter IN val1..val2

LOOP statements;
END LOOP;

- val1 - Start integer value.
- val2 - End integer value.

- Important steps to follow when executing a while loop:
  1. The counter variable is implicitly declared in the declaration section, so it's not necessary to declare explicity.
  2. The counter variable is incremented by 1 and does not need to be incremented explicitly.
  3. EXIT WHEN statement and EXIT statements can be used in FOR loops but it's not done oftenly.

**Program 2.15: Print numbers 1 to 5.**

```
DECLARE
   N NUMBER:=5;
   FOR I in 1..5 LOOP
   Dbms_output.put_line(i);
   END LOOP;
END;
/
SQL>/
1
2
3
4
5
```

**Program 2.16: Program to find sum of odd numbers.**

```
DECLARE
   n NUMBER;
   sum number default 0;
   value Number;
begin
   value:=&value;
   n:=1;
   for n in 1.. value LOOP
   IF mod(n,2)=1 THEN
   sum:=sum+n;
   END IF;
   END LOOP;
   dbms_output.put_line('sum = ' || sum);
END;
/
```

## PROGRAMS

**Program 1: Accept two numbers and print the largest number.**

```
Declare
   n1 number;
   n2 number;
Begin
   n1:=&n1;
   n2:=&n2;
   if(n1>n2) then
     dbms_output.put_line(n1 ||' is largest');
   else if (n1<n2) then
      dbms_output.put_line(n2 ||' is largest');
     else
      dbms_output.put_line('Both are equal');
     end if;
   end if;
End;
```

**Output:**

```
SQL> /
Enter value for n1:23
old 5: n1:=&n1;
new 5: n1:=23;
Enter value for n2: 12
old 6: n2:=&n2;
new 6: n2:= 12;
23 is largest
SQL>/
Enter value for n1: 11
old 5: n1:=&n 1;
new 5: n1:=11;
Enter value for n2: 11
old 6: n2:=&n2;
new 6: n2:=11;
Both are equal
```

**Program 2: Accept a number and check whether it is odd or even. If it is even no. print square of it otherwise cube of it.**

```
Declare
   n1 number;
Begin
n1:=&n1;
   if(mod(n1,2)=0) then
     dbms_output.put_line(n1 ||' is even no .');
     dbms_output.put_line('Square of '|| n1 ||' is '|| n1*n1);
   else
     dbms_output.put_line(n1 at ||' is odd no.');
     dbms_output.put_line('Cube of '|| n1 ||' is '|| n1 *n1*n1);
   end if;
End;
```

**Output:**

```
SQL> /
Enter value for n1: 2
old 4: n1:=&n1;
new 4: n1:=2;
2 is even no.
Square of 2 is 4
SQL> /
Enter value for n1: 3
old 4: n1:=&n1;
new 4: n1:=3;
3 is odd no.
Cube of 3 is 27
```

**Program 3: Accept a number and print factorial of it.**

```
Declare
   n number;
   i number;
   fact number:= 1;
Begin
   n:=&n;
   for i in 1..n
   loop
     fact:=fact*i;
   end loop;
   dbms_output.put_line(n ||'!='||fact);
End;
/
```

**Output:**

```
SQL>/
Enter value for n: 3
old 6: n:=&n;
new 6: n:=3;
3! = 6
```

**Program 4: Accept a number and print it in reverse order.**

```
Declare
   n number;
   r number;
Begin
   n:=&n;
   r:=0;
   while n>0
   loop
      r:=mod(n,10);
      dbms_output.put_line (r);
      n:=floor(n/10);
   end loop;
End;
/
```

**Output:**

```
SQL> /
Enter value for n: 234
old 5: n:=&n;
new 5: n:=234;
4
3
2
```

**Program 5: Accept a string and print it in reverse order.**

```
Declare
   s varchar2(30);
   c varchar2(1);
   l number;
Begin
   s:='&s';
   l:=length(s);
   while l>0
   loop
      c:=substr(s,1,l);
```

```
        dbms_output.put_line(c);
        l=l - 1;
    end loop;
End;
/
```

**Output:**

```
SQL> /
Enter value for s: smita
old 6: s:='&s';
new 6: s:='smita';
a
t
i
m
s
```

**GOTO Statement:**

This statement changes flow of control within block. Entry point is marked using tags << userdefined name >>.

The GOTO statement can make use of this name to jump into that block for execution.

**Syntax:**

```
GOTO <codeblock name>;
```

**For example:**

```
Declare
   ________
   ________
   ________
Begin
   If a > 0 Then
     GOTO POS;
   Else
     GOTO NEG;
   END IF;
   <<POS>>
      dbms_output.put_line ('Positive Number');
   <<NEG>
      dbms_output.put_line ('Negative Number');
END;
```

(W-15; S-

## 2.4 EXCEPTION HANDLING

### 2.4.1 Concept of Exception

- The Exception section in PL/SQL block is used to handle an error that occurs dur the execution of PL/SQL program.
- If an error occurs within a block, PL/SQL passes control to the EXCEPTION sectio the block. If no EXCEPTION section exists within the block or the EXCEPTION sect does not handle the error that's occurred then the error is passed out to the h environment.
- Exceptions occur when either an Oracle error occurs, (this automatically raises exception) or you explicitly raise an error or a routine that executes corrective act when detecting an error.
- Thus, Exceptions are identifiers in PL/SQL that are raised during the execution o block to terminate its action.
- There are two classes of exceptions, these are:

1. **Predefined exception:** Oracle predefined errors which are associated with speci error codes.
2. **User-defined exception:** Declared by the user and raised when specifically reques within a block. You can associate a user-defined exception with an error code if y wish.

(S-17, 19; W-

### 2.4.2 What is Exception Handling ?

- PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Blo known as exception Handling.
- Using Exception Handling, we can test the code and avoid it from exiting sudden When an exception occurs a message which explains its cause is received.
- PL/SQL Exception message consists of three parts.
  1. **Type of Exception**
  2. **An Error Code**
  3. **A message**
- By Handling the exceptions we can ensure a PL/SQL block does not exit suddenly.
- The general syntax for coding the exception section is given below:

```
DECLARE
   Declaration section
BEGIN
   Exception section
EXCEPTION
WHEN ex_name1 THEN
   -Error handling statements
```

```
WHEN ex_name2 THEN
   -Error handling statements
WHEN Others THEN
   -Error handling statements
END;
```

- General PL/SQL statements can be used in the Exception Block.
- When an exception is raised, Oracle searches for an appropriate exception handler in the exception section.
- For example in the above example, if the error raised is 'ex_name1 ', then the error is handled according to the statements under it.
- Since, it is not possible to determine all the possible runtime errors during testing of the code, the 'WHEN Others' exception is used to manage the exceptions that are not explicitly handled. Only one exception can be raised in a Block and the control does not return to the Execution Section after the error is handled.
- If there are nested PL/SQL blocks like this:

```
DECLARE
   Declaration section
BEGIN
   DECLARE
      Declaration section
   BEGIN
      Execution section
   EXCEPTION
      Exception section
   END;
EXCEPTION
   Exception section
END;
```

- In the above case, if the exception is raised in the inner block it should be handled in the exception block of the inner PL/SQL block else the control moves to the Exception block of the next upper PL/SQL Block. If none of the blocks handle the exception the program ends suddenly with an error.
- There are three types of Exceptions:
  1. Named System Exceptions.
  2. Unnamed System Exceptions.
  3. User-defined Exceptions.

**Named System Exceptions:**

System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they

are pre-defined and given a name in Oracle which are known as Named Sys Exceptions.

**For example:** NO_DATA_FOUND and ZERO_DIVIDE are called Named Sys exceptions.

- Named system exceptions are:

(i) Not Declared explicitly.

(ii) Raised implicitly when a predefined Oracle error occurs.

(iii) Caught by referencing the standard name within an exception-handling rout

| Exception Name | Reason | Error Numb |
|---|---|---|
| CURSOR_ALREADY_OPEN | When you open a cursor that is already open. | ORA-06511 |
| INVALID_CURSOR | When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened. | ORA-01001 |
| NO_DATA_FOUND | When a SELECT...INTO clause does not return any row from a table. | ORA-01403 |
| TOO_MANY_ROWS | When you SELECT or fetch more than one row into a record or variable. | ORA-01422 |
| ZERO_DIVIDE | When you attempt to divide a number by zero. | ORA-01476 |

**For Example:** Suppose a NO_DATA_FOUND exception is raised in a procedure, we write a code to handle the exception as given below:

```
BEGIN
 Execution section
EXCEPTION
WHEN NO_DATA_FOUND THEN
 dbms_output.put_line ('A SELECT...INTO did not return
row.');
 END;
```

**2. Unnamed System Exceptions:**

- Those system exception for which oracle does not provide a name is known unnamed system exception.
- These exception do not occur frequently. These Exceptions have a code and associated message.

There are two ways to handle unnamed system exceptions:

(i) By using the WHEN OTHERS exception handler, or

(ii) By associating the exception code to a name and using it as a nam exception.

- We can assign a name to unnamed system exceptions using a **Pragma** called **EXCEPTION_INIT.**
- **EXCEPTION_INIT** will associate a predefined Oracle error number to a programmer _defined exception name.
- The general syntax to declare unnamed system exception using EXCEPTION_INIT is:

```
DECLARE
   exception_name EXCEPTION;
   PRAGMA
   EXCEPTION_INIT (exception_name, Err_code);
BEGIN
Execution section
EXCEPTION
   WHEN exception_name THEN
      handle the exception
END;
```

**For Example:** Lets consider the product table and order_items table from SQL joins. Here product_id is a primary key in product table and a foreign key in order_items table. If we try to delete a product_id from the product table when it has child records in order_id table, an exception will be thrown with oracle code number -2292. We can provide a name to this exception and handle it in the exception section as given below.

```
DECLARE
   Child_rec_exception EXCEPTION;
   PRAGMA
   EXCEPTION_INIT (Child_rec_exception, 2292);
BEGIN
   Delete FROM product where product_id= 104;
EXCEPTION
   WHEN Child_rec_exception
   THEN Dbms_output.put_line('Child records are present for this
product_id.');
END;
/
```

3. **User Defined Exceptions:**

- We will see explanation of this in section 2.4.4.

## 2.4.3 Predefined Exception

- The two most common errors originating from a SELECT statement occur when it returns no rows **(WHEN NO_DATA_FOUND)** or more than one row (remember that this is not allowed in PL/SQL select command).
- If no rows are selected from SELECT statement then WHEN NO_DATA_FOUND

exception is used and for more than one row WHEN TOO_MANY_ROWS exception used.

- The example below deals with these two conditions.

```
DECLARE
   TEMP_sal NUMBER(10,2);
BEGIN
   SELECT sal INTO TEMP_sal
   From emp
   WHERE empno>=7698;
   IF TEMP_sal > 1000 THEN
      UPDATE emp SET sal = (TEMP_sal*1.175)
      WHERE empno>=7698;
   ELSE
      UPDATE emp SET sal = 5000
      WHERE empno>=7698;
   END IF;
   COMMIT;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      Dbms.Output.put_line('Empno does not exists');
   WHEN TOO_MANY_ROWS THEN
      Dbms.Output.put_line('No. of rows selected');
END;
/
SQL> /
```

- The above block will generate an error either there are more than one record with empno greater than 7698 or emp table does not have a record with empno>=7698. T exception raised from this will be passed to the EXCEPTION section where ea handled action will be checked. The statements within the TOO_MANY_ROWS NO_DATA_FOUND will be executed before the block is terminated.
- But if some other error occurred this EXCEPTION section would not handle it becau it isn't defined as a checkable action. To cover all possible errors other than this, u WHEN OTHERS exception.

**For example:**

```
DECLARE
   TEMP_sal NUMBER(10,2);
BEGIN
   SELECT sal INTO TEMP_sal
   From emp
   WHERE empno>=7698;
IF TEMP_sal > 1000 THEN
```

```
      UPDATE emp SET sal = (TEMP_sal*1.175)
      WHERE empno>=7698;
   ELSE
      UPDATE emp SET sal = 5000
      WHERE empno>=7698;
   END IF;
   COMMIT;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      Dbms.Output.put_line('Empno does not exists');
   WHEN TOO_MANY_ROWS THEN
      Dbms.Output.put_line('No. of rows selected');
   WHEN OTHERS THEN
      Dbms.Output.put_line('SOME ERROR OCCURRED');
   END;
```

- This block will trap all errors. If the exception isn't no rows returned or too many rows returned then the OTHERS action will perform the error handling.
- PL/SQL provides two special functions for use within an EXCEPTION section, they are SQLCODE and SQLERRM.
- SQLCODE is the Oracle error code of the exception, SQLERRM is the Oracle error message of the exception. You can use these functions to detect what error has occurred (very useful in an OTHERS action). This is generally used to store errors occurs in PL/SQL program in table so SQLCODE and SQLERRM should be assigned to variables before you attempt to use them.

**For example:**

```
DECLARE
   TEMP_sal NUMBER(10,2);
   ERR_MSG VARCHAR2(100);
   ERR_CDE NUMBER;
BEGIN
   SELECT sal INTO TEMP_sal
   From emp
   WHERE empno>=7698;
   IF TEMP_sal > 1000 THEN
   UPDATE emp SET sal = (TEMP_sal*1.175)
   WHERE empno>=7698;
ELSE
   UPDATE emp SET sal = 5000
   WHERE empno>=7698;
END IF;
COMMIT;
```

```
EXCEPTION
WHEN NO_DATA_FOUND THEN
   INSERT INTO ERRORS (CODE, MESSAGE)
   VALUES (99, 'NOT FOUND');
WHEN TOO_MANY_ROWS THEN
   INSERT INTO ERRORS (CODE, MESSAGE)
   VALUES (99, 'TOO MANY');
 WHEN OTHERS THEN
   ERR_CDE:= SQLCODE;
   ERR_MSG:= SUBSTR(SQLERRM,1,100);
   INSERT INTO ERRORS (CODE, MESSAGE) VALUES(ERR_CDE,ERR_MSG
END;
```

- In this case ERRORS table contain fields code and message. According to the occurred in PL/SQL block, the values of code and messge will get stored into ERRORS table.
- **Predefined Exceptions are:**

  **NO_DATA_FOUND:** when no rows are returned.

  **CURSOR_ALREADY_OPEN:** when a cursor is opened in advance.

  **STORAGE_ERROR:** if memory is damaged.

  **PROGRAM_ERROR:** internal problem in PL/SQL.

  **ZERO_DIVIDE:** divide by zero.

  **INVALID_CURSOR:** if a cursor is not open and you are trying to close it.

  **LOGIN_DENIED:** invalid user name or password.

  **INVALID_NUMBER:** if you are inserting a string datatype for a number datat which is already declared.

  **TOO_MANY_ROWS:** if more rows are returned by select statement.
- Predefined exceptions are raised implicitly by the runtime system. For example you try to divide a number by zero, PL/SQL raises the predefined except ZERO_DIVIDE automatically.

## 2.4.4 User Defined Exceptions

- We can explicitly define exceptions based on business rules. These are called as use defined exceptions.
- Steps to use user-defined exception in PL/SQL:
  (i) Declare user-defined exception explicitly in the declaration section.
  (ii) In the execution section, explicitly raise it using RAISE statement.
  (iii) Handle it by referencing the user-defined exception name in the except section.

**Syntax:**

```
DECLARE
   exception_name  EXCEPTION;
BEGIN
   statements;
   IF condition THEN
      RAISE exception_name;
   END IF;
EXCEPTION
   WHEN exception_name THEN
      User defined message or action statement;
END;
```

- There are two methods of defining exception by user.
  1. RAISE statement.
  2. RAISE_APPLICATION_ERROR statement.

**1. RAISE Statement**

- If you explicitly need to raise an error then RAISE statement is used and you have to declare an exception variable in declare section.

**For example:**

```
DECLARE
   TEMP_sal NUMBER(10,2);
   NEGATIVE_SAL EXCEPTION;
BEGIN
   SELECT sal INTO TEMP_sal
   From emp
   WHERE empno=7698;
   IF TEMP_sal < 0 THEN
      Raise NEGATIVE_SAL;
   ELSE
      UPDATE emp SET sal = 5000
      WHERE empno=7698;
   END IF;
   COMMIT;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      Dbms_output.put_line('Record NOT FOUND');
   WHEN NEGATIVE_SAL THEN
      Dbms_output.put_line('Salary is negative');
END;
```

- If the above example find row with a salary less than 0 then PL/SQL raise user_defined NEGATIVE_SAL exception.

**2. RAISE_APPLICTAION_ERROR Statement**

- The RAISE_APPLICATION_ERROR takes two input parameters: the error number an error message. The error number must be between -20001 to -20999. You can ca RAISE_APPLICATION_ERROR within procedures, functions, packages and triggers.

**For examples:**

```
DECLARE
   TEMP_Sal NUMBER(10,2);
BEGIN
   SELECT sal INTO TEMP_sal
From emp
   WHERE empno=7698;
   UPDATE emp SET sal = TEMP_sal *1.5
   WHERE empno=7698;
COMMIT;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
   RAISE_APPLICATION_ERROR (-20100,'Record NOT FOUND');
END;
```

**Note that in this case exception variable declaration is not required.**

```
DECLARE
   TEMP_sal NUMBER (10, 2);
BEGIN
   SELECT sal INTO TEMP_sal
   from emp
   WHERE empno = 7698;
If TEMP_sal < 0 then
   RAISE_APPLICATION_ERROR (- 20010, 'Salary is negative');
else
   UPDATE emp
   SET sal = 5000
   WHERE empno = 7698;
end if;
EXCEPTION
   WHEN NO_DATA.FOUND THEN
   DBMS_OUTPUT.PUT_LINE('Record not found');
end;
```

**Program 2.17: Accept empno and check whether it is present in emp table or not.**

```
Declare
  v_no emp.empno%type;
  v_empno emp.empno%type;
Begin
```

```
    v_empno:=&v_empno;

    select empno into v_no
    from emp
    where empno=v_empno;

    if v_no=v_empno then
     dbms_output.put_line('Empno exists');
    end if;
    Exception
     When NO_DATA_FOUND then
        dbms_output.put_line('Empno does not exists');
  End;
```

**Output:**

```
  SQL> /
  Enter value for v_empno: 7768
  old 5: v_empno:=&v_empno;
  new 5: v_empno:=7768;
  Empno does not exists
  SQL> /
  Enter value for v_empno: 7698
  old 5: v_empno:=&v_empno;
  new 5: v_empno:=7698;
  Empno exists
```

**Program 2.18: Print name of emp getting second maximum salary.**

```
  Declare
    v_name    emp.ename%type;
  Begin
    select e2.ename into v_name from emp e1, emp e2
    where e1.sal>e2.sal;
    dbms_output.put_line(v_name || 'is getting second max salary');
  Exception
    When TOO_MANY_ROWS then
      dbms_output.put_line('More than one Empno getting second
                                                    max salary');
  End;
```

**Output:**

```
  SQL> /
  More than one Empno getting second max salary.
```

**Program 2.19: Accept empno and check whether commission is null or not. If commission is null raise an exception otherwise display commission.**

```
Declare
   v_comm   emp.comm%type;
   v_empno  emp.empno%type;
   check_comm exception;
Begin
   v_empno:=&v_empno;
   select comm into v_comm
   from emp
   where empno=v_empno;

   if v_comm is NULL then
    raise check_comm;
   else
    dbms_output.put_line('commission = '||v_comm);
   end if;
Exception
   When NO_DATA_FOUND then
    dbms_output.put_line('Empno does not exists');
   When check_comm then
    dbms_output.put_line('Empno getting null commission');
End;
```

**Output:**

```
SQL> /
Enter value for v_empno: 7566
old 6: v_empno:=&v_empno;
new 6: v_empno:= 7566;
Empno getting null commission
SQL> /
Enter value for v_empno: 7521
old 6: v_empno:=&v_empno;
new 6: v_empno:=7521;
commission = 500
```

## 2.5 FUNCTIONS (W-18; S-19)

- A function contains a PL/SQL block to perform specific task. It can read a list of values but it will explicitly return a single value which is normally assigned to a variable.

### 2.5.1 What is a Function?

- A function is a named PL/SQL Block which is similar to a procedure.
- The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

## 2.5.2 How to Create a Function?

- The General syntax to create a function is:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
RETURN return_datatype;
{IS|AS}
Declaration_section
BEGIN
Execution_section
Return return_variable;
EXCEPTION
exception section
Return return_variable;
END;
```

1. **Return Type:** The header section defines the return type of the function. The return datatype can be any of the oracle datatypes like varchar, number etc.
2. The **execution and exception** section both should return a value which is of the datatype defined in the header section.

For example, let's create a function called "employer_details_func' similar to the one created in stored procedure.

```
CREATE OR REPLACE FUNCTION employer_details_func
   RETURN VARCHAR (20);
IS
   emp_name VARCHAR(20);
BEGIN
   SELECT first_name INTO emp_name
   FROM emp_tbl where empID = '100';
   RETURN emp_name;
END;
/
```

- In the above example we are retrieving the 'first_name' of employee with empID 100 to variable 'emp_name'.
- The return type of the function is VARCHAR which is declared in line no 2.
- The function returns the 'emp_name' which is of type VARCHAR as the return value in second line.

**Program 2.20: Pass empno as a parameter to function and function will return salary.**

```
CREATE OR REPLACE FUNCTION  myfuncl(f_no  IN number)
RETURN number
IS
v_sal emp.sal%TYPE;
Begin
```

```
    Select sal into v_sal
    From emp
    Where emp_no=f_no;
  If  SQL%FOUND  then
    RETURN (v_sal);
  Else
    RETURN 0;
  End if;
END myfunc1;
```

## 2.5.3 How to Execute a Function?

- A function can be executed in the following ways.

1. A function returns a value we can assign it to a variable.

```
employee_name:=  employer_details_func;
```

If 'employee_name' is of datatype varchar we can store the name of the employee assigning the return type of the function to it.

2. As a part of a SELECT statement,

```
SELECT employer_details_func FROM dual;
```

3. In a PL/SQL Statements like,

```
dbms_output.put_line(employer_details_func);
```

## 2.5.4 How to Delete a Function ?

- To delete a function DROP FUNCTION command is used.
- **Syntax:**

```
DROP FUNCTION function_name;
```

**For example:**

```
DROP FUNCTION myfunc1;
```

## PROGRAMS FOR FUNCTIONS

**Program 1: Pass a number to a function and check whether it is divisible by 5 or not**

```
create or replace function f1 (f_no in number)
return number
as
Begin
  if (mod(f_no,5)=0) then
   return 1;
  else
   return 0;
  end if;
End f1;
/
```

Function can be executed using select statement by calling Program as follows:

```
SQL> select f1(10) from dual;
f1(10)
--------
     1
```

**Calling program:**

```
Declare
  no number;
  r number;
Begin
  no:=&no;
  r:=f1(no);
  if(r=1) then
   dbms_output.put_line(no | | ' is divisible by 5');
  else
  dbms_output.put_line(no || ' is not divisible by 5');
   end if;
End;
```

**Output:**

```
SQL> /
Enter value for no: 30
old 5: no:=&no;
new 5: no:=30;
30 is divisible by 5

SQL> /
Enter value for no: 12
old 5: no:=&no;
new 5: no:=12;
12 is not divisible by 5
```

**Program 2: Pass two strings to a function and print the largest string.**

```
create or replace function f2 (str1 in varchar2, str2 in varchar2)
return varchar2
as
  L1 number;
  L2 number;
Begin
  L1:=length(str1);
  L2:=length(str2);
  if (L1>L2) then
```

```
        return 'First string is largest';
      else if(L2 >L1 ) then
        return 'Second string is largest';
       else
        return 'Both have equal length;
      end if;
    end if;
    End f2;
/
```

**Output:**

```
SQL> select f2('smita','chavan') from dual;
F2('SMITA','CHAVAN')
------------------------------
Second string is largest
SQL> select f2('hello','hi') from dual;
F2('HELLO','HI')
-------------------------
First string is largest
SQL> select f2('smita','swati') from dual;
F2 ('SMITA','SWATI')
---------------------------
Both have equal length.
```

**Program 3: Pass a month to a function and print no. of employee joined in that mont**

```
create or replace function f3 (f_mon in varchar2)
return number
as
  v_count number;
Begin
   select count(*) into v_count
   from emp
   where to_char(hiredate,'mon')=f_mon;
   return v_count;
End f3;
/
SQL> select f3('apr') from dual;
F3('APR')
-----------
      2
```

**Calling program:**

```
Declare
  v_mon varchar2( 10);
```

```
    r number;
  Begin
    v_mon:=&v_mon;
    r:=f3(v_mon);
    if(r>0) then
     dbms_output.put_line('No. of emp joined in '|| v_mon ||
        '=' || r);
    else
     dbms_output.put_line('No emp is joined in ' ||  v_mon);
    end if;
  End;
```

**Output:**

```
SQL> /
Enter value for v_mon: 'dec'
old 5: v_mon:=&v_mon;
new 5: v_mon:='dec';
No. of emp joined in dec = 1
SQL> /
Enter value for v_mon: 'jan'
old 5: v_mon:=&v_mon;
new 5: v_mon:='jan';
No emp is joined in jan
```

**Program 4: Pass a character to a function and return how many employee name(s) start with accepted character.**

```
create or replace function f4 (f_ch in varchar2)
return number
as
  v_count number;
Begin
  select count(*) into v_count
  from emp
  where instr(ename,f_ch)=1;
  return v_count;
End f4;
/
```

**Output:**

```
SQL> select f4('B') from dual;
f4('B')
----------
         1
```

**Calling program:**

```
Declare
  v_ch varchar2(10);
  r number;
Begin
  v_ch:=&v_ch;
  r:=f4(v_ch);
  if(r>0) then
   dbms_output.put_line('No. of employee having name starting wi
    '||v_ch ||'=' || r);
  else
   dbms_output.put_line('No employee name start with ' || v_ch);
  end if;
End;
/
```

**Output:**

```
SQL> /
Enter value for v_ch: 'C'
old 5: v_ch:=&v_ch;
new 5: v_ch:='C';
No. of employee having name starting with C = 1

SQL> /
Enter value for v_ch: 'E'
old 5: v_ch:=&v_ch;
new 5: v_ch:='E';
No employee name start with E
```

**Program 5: Accept a deptno and Pass it to function and check whether it is present dept table or not. If it present print number of empolyee working in the dept.**

```
create or replace function f5 (f_no in number)
return number
as
  v_no number;
  v_count number;
Begin
  select deptno into v_no
  from dept
  where deptno=f_no;
  if (v_no=f_no) then
    select count(*) into v_count
    from emp
```

```
        where deptno=f_no;
        return v_count;
      end if;
    Exception
      when NO_DATA_FOUND then
        return -1;
    End f5;
/
```

**Output:**

```
SQL> select f5(10) from dual;
F5(10)
--------
       5
SQL> select f5(40) from dual;
F5(40)
--------
       0
SQL> select f5(70) from dual;
F5(70)
--------
      -1
```

**Calling program:**

```
Declare
  v_no number;
  r number;
Begin
  v_no:=&v_no;
  r:=f5(v_no);
  if(r>0) then
    dbms_output.put_line('No.  of  emp  working  in  dept  '  ||
     v_no || ' = ' || r);
  else if(r=0) then
    dbms_output.put_line('Dept ' || v_no || ' exist in dept table
     but no emp working in this dept');
  else
    dbms_output.put_line('Dept  '  ||  v_no  ||'  does  not  exist
     in dept table ');
  end if;
  end if;
  End;
```

**Output:**

```
SQL> /
Enter value for v_no: 30
old 5: v_no:=&v_no;
new 5: v_no:=30;
No. of emp working in dept 30 = 2

SQL> /
Enter value for v_no: 40
old 5: v_no:=&v_no;
new 5: v_no:=40;
Dept 40 exist in dept table but no emp working in this dept
SQL> /
Enter value for v_no: 60
old 5: v_no:=&v_no;
new 5: v_no:=60;
Dept 60 does not exist in dept table
```

**Program 6: Pass a salary to a function and check whether it is greater than maxim or less than minimum salary or in between. (use out variable wh indicate max or min value)**

```
create or replace function f6 (f_sal in number, f_no out number)
return number
as
  v_max number;
  v_min number;
Begin
  select max(sal) into v_max
  from emp;

  select min(sal) into v_min
  from emp;
   dbms_output.put_line('max sal = ' || v_max);
   dbms_output.put_line('min Sal = ' ||  v_min);
 if (v_max < f_no) then
    f_no: = v_max;
    return 1;
 else if (v_min > f_no) then
    f_no: = v_min;
    return 2;
 else
    return 0;
end if;
```

```
   end if;
End f6;
/
```

**Calling program:**

```
   Declare
     v_sal number;
     v_no number;
     r number;
   Begin
     v_sal:=&v_sal;
     r:=f6(v_sal,v_no);

     if(r=1) then
      dbms_output.put_line(v_sal || ' is greater than max Sal
        = ' || v_no);
     else if(r=2) then
      dbms_output.put_line(v_sal || ' is less than min sal = '
        || v_no);
      else
       dbms_output.put_line(v_sal || ' is in between max and
        min sal');
      end if;
     end if;
   End;
   /
```

**Output:**

```
   SQL> /
   Enter value for v_sal: 30000
   old 6: v_sal:=&v_sal;
   new 6: v_sal:=30000;
   max sal = 25000
   min sal = 1100
   30000 is greater than max sal = 25000
   SQL> /
   Enter value for v_sal: 200
   old 6: v_sal:=&v_sal;
   new 6: v_sal:=200;
   max sal = 25000 min sal = 1100
   200 is less than min sal = 1100

   SQL> / Enter value for v_sal: 2000
   old 6: v_sal:=&v_sal;
```

```
new 6: v_sal:=2000;
max sal = 25000
min sal= 1100
2000 is in between max and min sal
```

**Program 7: Pass name to a function and print number of years of service he has put**

```
create or replace function f7 (f_name in varchar2)
return number
as
  v_count number;
Begin
  select round((sysdate-hiredate)/365) into v_count
  from emp
  where ename=f_name;
  return v_count;
Exception
   when NO_DATA_FOUND then
      return -1;
End f7;
/
```

**Output:**

```
SQL> select f7('BLAKE') from dual;
F7('BLAKE')
---------------
  21

SQL> select f7('SEEMA') FROM DUAL;
F7('SEEMA')
---------------
  - 1
```

**Calling program:**

```
Declare
  v_name varchar2(20);
  r number;
Begin
  v_name:='&v_name';
  r:=f7(v_name);
  if(r>0) then
   dbms_output.put_line(v_name || ' is working
     organization since last '|| r || ' years');
  else if(r=-1) then
    dbms_output.put_line(v_name || ' employee does not exi
     in emp table');
```

```
    end if;
  end if;
End;
/
```

**Output:**

```
SQL> /
Enter value for v_name: MARTIN
old 5: v_name:='&v_name';
new 5: v_name:='MARTIN';
MARTIN is working in organization since last 21 years
SQL> /
Enter value for v_name: SANDEEP
old 5: v_name:='&v_name';
new 5: v_name:='SANDEEP';
SANDEEP employee does not exist in emp table
```

## 2.6 PROCEDURE (W-15; S-16; W-17; S-18)

- Procedures are simply a named PL/SQL block, that executes certain task. A procedure is completely portable among platforms in which Oracle is executed.
- Procedure is similar to a procedure in other programming languages. A procedure has a header and a body.
- The header consists of the name of the procedure and the parameters or variables passed to the procedure.
- The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block.
- A procedure is similar to an anonymous PL/SQL Block but it is named for repeated usage.

### 2.6.1 Create a Procedure (W- 15)

- A procedure is created using CREATE PROCEDURE command.

**Syntax:**

```
CREATE [OR REPLACE] PROCEDURE   procedure _name
[(argument   [in/out/in   out]   datatype   [,argument   [in/out/in   out]
      datatype.....])]
(IS / AS)
[variable declaration]
(PL/SQL block);
```

**Note:** Square brackets [ ] indicate optional part.

- CREATE PROCEDURE procedure_name will create a new procedure with the given procedure_name. OR REPLACE is an optional clause. It is used to change the definition of an existing procedure.

- If the procedure accept arguments specify argument details as,
  Argument_name IN / OUT / IN OUT datatype
  Argument_name indicate Variable_name
- **IN** indicates the variable is passed by the calling program to procedure.
- **OUT** indicates that the variable pass value from procedure to calling program.
- **IN OUT** indicates that the variable can pass values both in and out of a procedure.
- Datatype specify any PL/SQL datatype.

## 2.6.2 Execute a Procedure

- To execute the stored procedure simply call it by name in EXECUTE command as,

```
SQL> execute myproc1(7768);
```

- This will execute myproc1 with the value 7768.
- The second method of calling the procedure is given below:
  Write the following code in an editor.

```
Declare
   c_empno    number;
Begin
   Myproc1(&c_empno);
End;
/
Execute it as
   SQL >/
```

- In this case the value of variable `c_empno` is accepted from user and then it is passed to `myproc1` procedure.
- To see the effect of this procedure use command,

```
SQL>select * from emp;
```

## 2.6.3 Delete a Procedure

- To delete a procedure DROP PROCEDURE command is used.

**Syntax:**

```
DROP PROCEDURE procedure_name;
```

**For example:**

```
DROP PROCEDURE myproc1;
```

## PROGRAMS FOR PROCEDURES

**Program 1: Pass deptno to procedure and print maximum salary of emp working in that department. If deptno does not exist print message.**

```
create or replace procedure p1 (p_deptno in number)
as
  max_sal emp.sal%type;
```

```
Begin
   select max(sal) into max_sal
   from emp
   where deptno=p_deptno;
   if(max_sal > 0) then
    dbms_output.put_line('Max salary = ' || max_sal);
   else
    dbms_output.put_line('Deptno does not exists');
   end if;
End p1;
/
```

- **Procedure can be executed using execute statement or by calling program as follows:**

```
SQL> execute p1(90);
Deptno does not exists
SQL> execute p1(30);
Max salary = 1500
```

**Calling program:**

```
Declare
  v_deptno emp.deptno%type;
Begin
  v_deptno:=&v_deptno;
  p1(v_deptno);
End;
SQL> /
Enter value for v_deptno: 50
old 4: v_deptno:=&v_deptno;
new 4: v_deptno:=50;
Max salary = 3000
```

---

**Program 2: Pass empno and sal to procedure and print name and sal of employee having empno = accepted empno and modify sal with accepted sal.**

```
create or replace procedure p2 (p_no in number, p_sal in number)
as
  v_name emp.ename%type;
  v_sal emp.sal%type;
Begin
   select ename,sal into v_name,v_sal
   from emp
   where empno=p_no;
   update emp
   set sal=sal+p_sal
   where empno=p_no;
   v_sal:=v_sal+p_sal;
```

```
        dbms_output.put_line('Modified salary of '|| v_name ||' '|| v_sal);

    Exception
      when NO_DATA_FOUND then
       dbms_output.put_line('Empno does not exist!....');
    End p2;
/
```

**Output:**

```
SQL> execute p2(7698, 100);
Modified salary of BLAKE is 3150
SQL> execute p2(111,1000);
Empno does not exist!...
```

**Program 3: Pass commission to procedure and set this value of commission of emp having commission as null.**

```
create or replace procedure p3(p_comm in number)
as
Begin
  update emp
  set comm=p_comm
  where comm is null;
End p3;
SQL> execute p3(500);
```

To see the changes in emp table after execution of this procedure use `select * from emp;`

**Program 4: Write a procedure which will accept empno and return (using OUT variable) name of his supervisor.**

```
create or replace procedure p4(p_no in number, p_name out varchar2)
as
  v_name emp.ename%type;
Begin
  select ename into v_name
  from emp
  where empno =(select mgr from emp where empno=p_no);
  p_name:=v_name;
Exception
  when NO_DATA_FOUND then
    p_name:='no';
End p4;
/
```

**Calling program:**

```
Declare
  c_no number;
```

```
    c_name varchar2(30);
  Begin
    c_no:=&c_no;
    p4(c_no,c_name);

    if(c_name='no') then
     dbms_output.put_line('Empno does not exist! ..... ');
    else
     dbms_output.put_line('Name of supervisor is ' || c_name);
    end if,
  End;
```

**Output:**

```
SQL> /
Enter value for c_no: 7566
old 5: c_no:=&c_no;
new 5: c_no:=7566;
Name of supervisor is KING

SQL> /
Enter value for c_no: 90
old 5: c_no:=&c_no;
new 5: c_no:=90;
Empno does not exist!.....
```

**Program 5: Pass deptno to a procedure. Procedure will pass no. of employees working in that dept in same variable. (Use IN OUT variable)**

```
create or replace procedure p5
  (p_no in out number)
as
  v_no number;
Begin
  select count(*) into v_no
  from emp
  where deptno =p_no;
  p_no:=v_no;
Exception
  when NO_DATA_FOUND then
    p_no:=0;
End p5;
/
```

**Calling program:**

```
declare
  c_no number;
```

```
begin
  c_no:=&c_no;
  p5(c_no);

  if(c_no=0) then
   dbms_output.put_line('Dept does not exist!.....');
  else
   dbms_output.put_line('No. of emp = ' || c_no);
  end if;
end,
```

**Output:**

```
SQL> /
Enter value for c_no: 20
old 4: c_no:=&c_no;
new 4: c_no:=20;
No. of emp = 2

SQL> /
Enter value for c_no: 70
old 4: c_no:=&c_no;
new 4: c_no:=70;
  Dept does not exist!.....
```

**Program 6: Pass empno as an argument to procedure and modify salary of that emp.**

```
CREATE OR REPLACE PROCEDURE myproc1(p_no  IN  number)  /* argument */
IS
   v_sal   number(10,2);
BEGIN
   Select  sal into v_sal
   From emp
   Where empno=p_no;
If v_sal > 1000 then
   Update emp
   Set sal = v_sal*1.75
   Where empno=p_no;
Else
   Update emp
   Set sal = 5000
   Where empno=p_no;
End if;
EXCEPTION
WHEN NO_DATA_FOUND THEN
```

```
    Dbms_output.put_line('Emp_no doesn't exists');
END myproc1;
```

**Program 7: Pass a empno as argument to procedure and procedure will pass job to the calling program.**

```
CREATE OR REPLACE PROCEDURE myproc2
(p_no  IN   number, P_job OUT emp.job%TYPE) /* arguments */
IS
   v_job       emp.job%TYPE;
BEGIN
   Select  JOB  into v_job
   From emp
   Where empno=p_no;
   p_job:=v_job;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
    p_job:='NO';
END myproc2;
```

**Calling procedure myproc2 using following code:**

```
Declare
   c_empno    number;
   c_job      emp.job%TYPE;
Begin
   myproc2(&c_empno,c_job);
   If c_job='NO' then
   dbms_output.put_line('Emp_no doesn't exists');
   Else
      dbms_output.put_line('Job of emp. Is ' || c_job);
   End if;
End;
/
SQL> /
```

**Program 8: Pass salary to procedure and procedure will pass no. of employee(s) having salary equal to given salary in the same variable. (Use IN OUT variable).**

```
CREATE OR REPLACE PROCEDURE myproc3
(p_sal  IN  OUT   emp.sal%TYPE)                    /* arguments */
IS
   v_count   number;
BEGIN
   Select count(*) into v_count
   From emp
   Where sal=p_sal;
```

```
    P_sal:=v_count;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        P_sal:=0;
END myproc3;
```

**Calling procedure** `myproc3` **using following code:**

```
Declare
    c_sal       emp.sal%TYPE;
Begin
    c_sal:=&c_sal;
    myproc3(c_sal);
    If c_sal=0 then
        dbms_output.put_line('No employee is having salary equal to
        accepted salary');
    Else
        dbms_output.put_line('No. of emp. having salary = accepted
        salary are ' || c_sal);
    End if;
End;
```

## 2.7 CURSORS (S-17, 18; W-17, 18)

- Whenever, a SQL statement is issued the Database server opens an area of memory which called Private SQL area in which the command is processed and executed. An identifier for this area is called a cursor.
- When PL/SQL block uses a select command that returns more than one row, Oracle displays an error message and invokes the TOO_MANY_ROWS exception. To resolve this problem, Oracle uses a mechanism called cursor.
- There are two types of cursors.
  1. Implicit cursors.
  2. Explicit cursors.
- PL/SQL provides some attributes which allows to evaluate what happened when the cursor was last used. You can use these attributes in PL/SQL statements like functions but you cannot use them within SQL statements.
- The SQL cursor attributes are:
  1. **%ROWCOUNT:** The number of rows processed by a SQL statement.
  2. **%FOUND:** TRUE if at least one row was processed.
  3. **%NOTFOUND:** TRUE if no rows were processed.
  4. **%ISOPEN:** TRUE if cursor is open or FALSE if cursor has not been opened or has been closed. Only used with explicit cursors.

**What are Cursors ?**

- A cursor contains information on a select statement and the rows of data accessed by it. This temporary work area is used to store the data retrieved from the database, and manipulate this data.
- A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the *active* set.

## 2.7.1 Definition

- A cursor is a temporary work area used to store the data retrieved from the database and manipulate this data.

OR

- A cursor is a temporary work area created by system memory when SQL or /PL/SQL statements are executed.

**Declaring a Cursor:**

- Cursors are defined within a DECLARE section of a PL/SQL block with DECLARE command.

**Syntax:**

```
Cursor cursor_name [(parameters)] [RETURN return_type] IS SELECT query.
```

- The cursor is defined by the CURSOR keyword followed by the cursor identifier (Cursor_name) and then the SELECT statement.
- `Parameters` and `return` are optional part. When parameters are passed to cursor it is called as **Parameterized Cursor**.

**For example:**

```
DECLARE
   CURSOR   c_deptno   IS   SELECT  ename,sal,deptno FROM EMP;
```

**Opening a Cursor:**

- Cursors are opened with the OPEN statement. This populates the cursor with data.

**Syntax:**

```
OPEN Cursor_name[parameters];
```

**For example:**

```
DECLARE
   CURSOR  c_deptno   IS SELECT  ename,sal,deptno FROM EMP;
Begin
    Open  c_deptno;
End;
```

`Parameters` is an optional part. It is used in parameterized cursor.

**Accessing the Cursor Rows:**

- To access the rows of data within the cursor the FETCH statement is used.

**For example:**

```
DECLARE
   CURSOR   c_deptno   IS   SELECT  ename,sal,deptno FROM EMP;
   V_name     emp.ename%type;
   V_sal      emp.sal%type;
   V_deptno   emp.deptno%type;
Begin
  Open  c_deptno;
  FETCH c_deptno INTO v_name,v_sal,v_deptno;
  Dbms_output.put_line(V_NAME ||' '||V_deptno||' '||V_SAL);
End;
SQL > /
SMITH 800    20
```

- The FETCH statement reads the column values for the current cursor row and puts them into the specified variables. This can be as an equivalent to the SELECT INTO command. The cursor pointer is updated to point at the next row. If the cursor has no more rows the variables will be set to null on the first FETCH attempt, subsequent FETCH attempts will raise an exception.
- To process all the rows within a cursor use a FETCH command in a loop and check the cursor NOTFOUND attribute to see if we successfully fetched a row or not as follows:

```
DECLARE
    CURSOR   c_deptno   IS   SELECT  ename,sal,deptno FROM EMP;
   V_name        emp.ename%type;
   v-sal         emp.sal%type;
   v_deptno      emp.deptno%type;
Begin
   Open  c_deptno;
    Loop
     FETCH c_deptno INTO v_name,v_sal,v_deptno;
     Exit when c_deptno%NOTFOUND;
      Dbms_output.put_line(V_NAME ||' '||V_deptno||' '||V_SAL);
    End loop;
  close c_deptno;
End;
```

**sing a Cursor:**

The CLOSE statement releases the cursor and any rows within it, you can open the cursor again to refresh the data in it.

**Syntax:**

```
CLOSE Cursor_name;
```

**For example:**

```
Close c_deptno;
```

**Using Cursor For.... Loop:**

- In the cursor FOR loop, the result of SELECT query are used to determine the number of times the loop is executed.
- In a Cursor FOR loop, the opening, fetching and closing of cursors is performed implicitly.
- When you use it, Oracle automatically declares a variable with the same name as that is used as a counter in the FOR command. Just precede the name of the selected field with the name of this variable to access its contents.

**For example:**

```
DECLARE
   CURSOR   c_deptno  IS  SELECT ename, sal, deptno  FROM EMP;
Begin
  For x in c_deptno
  Loop
    Dbms_output.put_line(x.ename ||' '||x.deptno||' '||x.SAL);
  End loop;
End;
```

- In above example a Cursor For loop is used, there is no `open` and `fetch` command.
- `For x in c_deptno` implicitly opens the `c_deptno` cursor and fetches a value into the `x` variable. Note that `x` is not explicitly declared in the block.
- When no more records are in the cursor, the loop is exited and cursor is closed. There is no need to check the cursor %NOTFOUND attribute-that is automated via the cursor FOR loop. And also there is no need of close command.

**Cursor Variables:**

- A cursor variable is a reference type. It can refer to different storage locations as the program runs.
- To use cursor variable, first the variable has to be declared. And the storage has to allocated. It is then opened, fetched and closed similar to a static cursor.

**Syntax:**

```
TYPE type_name IS REF CURSOR [RETURN return_type];
```

- Where type_name is the name of new reference type and return.type is a record type.

```
DECLARE
   ---.  Defination using Rowtype
   TYPE student_R is REF CURSOR RETURN students%ROWTYPE;
```

```
---. Define a new record type
TYPE Student_Rec IS RECORD (
  fname          students.f_name%TYPE,
  lname          students_l_name%TYPE);
  variable of new type
  Namerec        Student_REC;

BEGIN
_______
_______
END;
```

**Constrained and Unconstrained Cursor variables:**

1. **Constrained Variables:**

- When the cursor variables are declared for a specific return type only, then these variables called constrained variables.
- When these variables are later opened, it must be opened for a query whose 'select' list matches the return type of the cursor. Otherwise predefined exception ROWTYPE_MISMATCH is raised.

2. **Unconstrained Variables:**

- An unconstrained cursor variable does not have a RETURN clause. When this variable is later opened, it can be for any query.
- Example of unconstrained variable are given below:

```
TYPE Stock_cur-price IS REF CURSOR;
----- variable of that type
v_cursor var stock_cur_price;

BEGIN
_______
_______
END;
```

**Opening a cursor variable for a query:**

**Syntax:**

```
OPEN cursor_variable FOR select_statement;
```

Where cursor_variable is a previously declared cursor variable and select_statement is the desired query. If the cursor variable is constrained, the select statement must match the return type of the cursor. Otherwise it raised error.

```
OPEN Student_R FOR
 Select * from students;
```

**Closing the Cursor Variables:**

- The CLOSE statment closes or deactivates the previously opened cursor and makes the active set undefined. After the cursor is closed, user cannot perform any operation on it.

```
CLOSE Cursor_name;
```

where cursor_name is previously opened cursor name.

- It does not necessary to free the storage for the cursor variable itself. It is fixed when the variable goes out of scope.

**Restrictions on Using cursor variables:**

- Cursor variable are a powerful feature of allowing different kinds of data to be returned in same variable. But, there are a number of restrictions with their use.

1. PL/SQL collections (index-by-tables, Nested tables and V arrays) cannot store cursor variables. Similarly, database tables and views can not store REF CURSOR columns.
2. The query associated with a cursor-variable in the OPEN-FOR statement can not be FOR UPDATE.
3. Remote subprograms cannot return the value of a cursor variable. They can be passed between client and server side PL/SQL (from oracle forms clients), but not between two servers.
4. You cannot use cursor variables with dynamic SQL in pro*c.

## 2.7.2 Types of Cursor (W-18)

- There are two types of cursors : Implicit cursor and Explicit cursor.

### 2.7.2.1 Implicit Cursor

- These are created by default when DML statements like INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.
- When the executable part of a PL/SQL block issues a SQL command, PL/SQL creates an implicit cursor which has the identifier SQL. PL/SQL internally manages this cursor.

**Program 2.24: Print no. of rows deleted from emp.**

```
DECLARE
  ROW_DEL_NO NUMBER;
BEGIN
  DELETE     FROM EMP;
  ROW_DEL_NO:= SQL%ROWCOUNT;
   DBMS_OUTPUT.PUT_LINE('No. of rows deleted are:'||
      ROW_DEL_NO);
END;
/
SQL> /
```

```
No. of rows deleted are: 14
PL/SQL procedure successully completed
```

**Program 2.25: Accept empno and print it's details(using cursor).**

```
DECLARE
  V_NO          EMP.EMPNO%TYPE:=&V_NO;
  V_NAME        EMP.ENAME%TYPE;
  V_JOB         EMP.JOB%TYPE;
  V_SAL         EMP.SAL%TYPE;
BEGIN
  SELECT ename,job,sal  INTO  V_NAME,V_JOB,V_SAL
  FROM emp
  WHERE empno=V_NO;

IF SQL%FOUND THEN  /* SQL%FOUND  is true if empno=v_no */
  Dbms_output.put_line(V_NAME ||' '||V_JOB||' '||V_SAL);
Exception
  when NO_DATA_FOUND then
  dbms_output.put_line ('Empno does not exists');
End;
SQL > /
Enter value for v_no: 34
Old 2: v_no emp.empno%type:=&v_no;
New 2: v_no emp.empno%type:=34
Empno does not exists
PL/SQL Procedure successfully completed

SQL > /
Enter value for v_no: 7369
SMITH CLERK 800
PL/SQL procedure successfully completed.
```

## 2.7.2.2 Explicit Cursor

- If SELECT statements in PL/SQL block return multiple rows then you have to explicitly create a cursor which is called as Explicit Cursor.
- The set of rows returned by a explicit cursor is called a result set. The row that is being processed is called the current row.
- Oracle uses four commands to handle Cursors. They are:
  1. **DECLARE:** Defines the name and structure of the cursor together with the SELECT statement.
  2. **OPEN:** Executes the query and the number of rows to be returned is determined.

3. **FETCH:** Loads the row addressed by the cursor pointer into variables and moves the cursor pointer on to the next row ready for the next fetch.
4. **CLOSE:** Releases the data within the cursor and closes it.

**Syntax of explicit cursor:**

Cursor cursorname is SQL select statement;

Cursor can be controlled using following 3 control statements. They are :

**OPEN:** Open statement identifies the active set...i.e. query returned by select statement.

**Syntax of open cursor:**

```
open cursorname;
```

**FETCH:** Fetch statement fetches rows into the variables. Cursors can be made into use using cursor for loop and fetch statement.

**Syntax of fetch:**

```
fetch cursorname into variable1, variable2...;
```

**CLOSE:** Close statement closes the cursor.

**Syntax of close:**

```
close cursorname;
```

## 2.7.3 Parameterised Cursor (W-17)

- Syntax for declaring a parameterized cursor:

```
CURSOR Cursorname (variable name Data type)
 is <select statement, ….>
```

- Opening a parameterized cursor and passing values to the Cursor.

```
OPEN CursorName (value/variable/Expression)
```

- **Patameterised cursor** passing parameter to cursor.

**Program 2.26: Accept a salary and print name, salary and job of employee having salary less than equal to accepted salary.**

```
Declare
  cursor c1(c_sal emp.sal%type) is select * from emp where
  sal<=c_sal;
  emp_rec c1%rowtype;
  v_sal emp.sal%type;
Begin
  v_sal:=&v_sal;
  dbms_output.put_line('Name salary job');
  dbms_output.put_line('--------------------');
  for emp_rec in c1(v_sal)
  loop
   dbms_output.put_line(emp_rec.ename || ' ' || emp_rec.sal
   ||'  '|| emp_rec.job);
```

```
    end loop;
  End;
```

**Output:**

```
SQL> /
Enter value for v_sal: 2000
old 7: open c1(&v_sal);
new 7: open c1 2000);
```

| Name | Salary | Job |
|---|---|---|
| WARD | 1250 | SALESMAN |
| MARTIN | 1250 | SALESMAN |
| TURNER | 1500 | SALESMAN |
| ADAMS | 1100 | CLERK |

**Program 2.27: Accept job and print details of employees having job as accepted j Also print the number of employee and name(s) of employee wi maximum salary.**

```
Declare
   cursor c1 (c_job emp.job%type) is select * from emp wh
job=cjob;
   emp_rec c1%rowtype;
   v_count number;
   v_job emp.job%type;
   max_sal emp.sal%type;
   name emp.ename%type;
Begin
   v_job:=&v_job;
   select ename,sal into name,max_sal
   from emp
   where job=v_job and sal =(select max(sal) from emp
      where job=v_job);
   dbms_output.put_line('Name salary job deptno');
   dbms_output.put_line('----------------------------');
   for emp_rec in c1(v_job)
   loop
   dbms_output.put_line(emp_rec.ename || ' ' || emp_rec.sal || ' '
    emp_rec.job || ' ' || emp_rec.deptno);
   v_count:=c1%rowcount;
  end loop;
  dbms_output.put_line('No. of employees = ' || v_count);
  dbms_output.put_line(name ||' having max sal =' || max_sal);
End;
```

**Output:**

```
SQL>/
Enter value for v_job: 'SALESMAN'
old 9: v_job:=&v_job;
new 9: v_job:='SALESMAN';
```

| Name | salary | Job | deptno |
|---|---|---|---|
| WARD | 1250 | SALESMAN | 50 |
| MARTIN | 1250 | SALESMAN | 30 |
| TURNER | 1500 | SALESMAN | 30 |

```
No. of employees = 3
TURNER having max sal = 1500
```

**Program 2.28: Accept deptno and employee details who are working in that department. Also print no. of employees.**

```
Declare
  cursor c1(c_deptno emp.deptno%type) is select * from emp where
deptno=c_deptno;
  emp_rec c1%rowtype;
  v_deptno emp.deptno%type,
  v_count number;
Begin
  dbms_output.put_line('Name salary job deptno');
  dbms_output.put_line ('--------------------'),
  for emp_rec in c1 (&v_deptno)
  loop
   dbms_output.put_line(emp_rec.ename ||' '|| emp_rec.sal ||' '||
    emp_rec.job ||' '|| emp_rec.deptno),
   v_count:=c1%rowcount;
  end loop;
  dbms_output.put_line('No. of employees = ' || v_count);
End;
```

**Output:**

```
SQL> /
Enter value for v_deptno: 20
old 7: open c1(&v_deptno);
new 7: open c1(20);
```

| Name | salary | job | deptno |
|---|---|---|---|
| SCOTT | 3000 | ANALYST | 20 |
| ADAMS | 1100 | CLERK | 20 |

```
No. of employees = 2.
```

**Program 2.29: Write a procedure which displays name, sal, dname of first 4 highe paid employee.**

```
create or replace procedure p6
as
  cursor c1 is select ename,sal,dname
    from emp,dept
    where emp.deptno=dept.deptno
    order by sal desc;
  e_rec c1%rowtype;
Begin
  dbms_output.put line('Name sal dname');
  dbms_output.put_line (' - - - - - - - - - -');

  for e_rec in c1
  loop
   if c1%rowcount<=4 then
    dbms_output.put_line(e_rec.ename || ' ' || e_rec.sal ||
      ' ' || e_rec.dname);
   end if;
  end loop;
End p6;
```

**Output:**

```
SQL> execute p6;
```

| Name | sal | dname |
|---|---|---|
| KING | 5000 | ACCOUNTING |
| BLAKE | 3150 | ACCOUNTING |
| SCOTT | 3000 | RESEARCH |
| JONES | 2975 | ACCOUNTING |

**Program 2.30: Write a procedure which will list names of employee who are reportin to 'BLAKE'.**

```
create or replace procedure p7
as
  cursor c1 is select ename
      from emp
      where mgr = (select empno from emp
       where ename='BLAKE');
  e_rec c1%rowtype;
Begin
  dbms_output.put_line('Name');
  dbms_output.put_line('-------');

  for e_rec in c1
```

```
    loop
      dbms_output.put_line (e_rec.ename);
    end loop;
  End p7;
```

**Output:**

```
SQL> execute p7;
Name
--------
WARD
MARTIN
TURNER
```

**Program 2.31: Pass job to a procedure and Print names and sal of emp whose salary is less than average salary of accepted job.**

```
create or replace procedure p8 (p_job in varchar2)
as
  cursor c1 is select ename,sal
      from emp
      where sal <= (select avg(sal)
       from emp
       where job=p_job);
  e_rec c1%rowtype;
Begin
  dbms_output.put_line('Name sal');
  dbms_output.put_line('------------');

  for e_rec in c1
  loop
   dbms_output.put_line(e_rec.ename || ' ' || e_rec.sal);
  end loop;
End p8;
/
```

**Output:**

| Name | Sal |
|---|---|
| WARD | 1250 |
| MARTIN | 1250 |
| CLARK | 2450 |
| TURNER | 1500 |
| ADAMS | 1100 |

**Program 2.32: Pass deptno to a function and print details of salesmen working in the dept.**

```
create or replace function f8
(f_no in number)
return number
as
  v_count number;
  cursor c_dept is select ename,sal,job,deptno
      from emp
      where deptno=f_no and job='SALESMAN';
Begin
  dbms_output.put_line('Name sal job deptno');
  dbms_output.put_line('----------------------');
  for e_rec in c_dept
  loop
   dbms_output.put_line(e_rec.ename || ' ' || e_rec.sal || ' ' ||
    e_rec.job || ' ' || e_rec.deptno);
   v_count:=c_dept%rowcount;
  end loop;
  if(v_count>1) then
    return v_count;
  else
    return -1;
  end if;
End f8;
/
```

**Calling program:**

```
Declare
  v_no number;
  r number;
Begin
  v_no:=&v_no;
  r:=f8(v_no);
  if(r>0) then
   dbms_output.put_line('No. of emp working in dept ' || v_
     || ' = ' || r);
  else if(r=-1) then
    dbms_output.put_line('Dept'|| v_no ||' does not exist
     emp table');
   end if;
  end if;
End
```

**Output:**

```
SQL>/
Enter value for v_no: 40
old 5: v_no:=&v_no;
new 5: v_no:=40;
Name sal job deptno
----------------------
Dept 40 does not exist in emp table

SQL> /
Enter value for v_no: 30
old 5: v_no:=&v_no;
new 5: v_no:=30;
```

| Name | sal | job | deptno. |
|---|---|---|---|
| MARGIN | 1250 | SALESMAN | 30 |
| TURNER | 1500 | SALESMAN | 30 |

No. of emp working in dept 30 = 2.

## PROGRAMS FOR CURSOR

**Program 1: Print the name, salary and deptno of all employees who belongs to deptno 10.**

```
Declare
  cursor c1 is select ename,sal,deptno
      from emp
      where deptno=10;
  v_name emp.ename%type;
  v_sal emp.sal%type;
  v_deptno emp.deptno%type;
Begin
  open c1;
  dbms_output.put_line('Name salary deptno');
  dbms_output.put_line('--------------------');
  loop
   fetch c1 into v_name,v_sal,v_deptno;
   exit when c1%notfound;
   dbms_output.put_line(v_name ||' '||v_sal||' '||v_deptno);
  end loop;
  close c1;
End;
```

**Output:**

```
SQL> /
```

| Name | Salary | deptno |
|---|---|---|
| JONES | 2975 | 10 |
| BLAKE | 3050 | 10 |
| CLARK | 2450 | 10 |
| KING | 5000 | 10 |

**Program 2: Print the name, job of employees having job as MANAGER or ANALYST.**

```
Declare
  cursor c1 is select ename,job
      from emp
      where job='MANAGER' or job='ANALYST';
  v_name emp.ename%type;
  v_job emp.job%type;
Begin
  open c1;
  dbms_output.put_line('Name job');
  dbms_output.put_line ('---------------');
  loop
   fetch c1 into v_name,v_job;
   exit when c1 %notfound;
   dbms_output.put_line(v_name || ' ' ||v_job);
  end loop;
  close c1;
End;
```

**Output:**

```
SQL> /
```

| Name | Job |
|---|---|
| SONU | ANALYST |
| JONES | MANAGER |
| BLAKE | MANAGER |
| CLARK | MANAGER |
| SCOTT | ANALYST |

**Program 3: Write a PL/SQL block to print 3rd, 6th and 7th record from emp.**

```
Declare
  cursor c1 is select * from emp;
```

```
    emp_rec c1 %rowtype;
  Begin
    open c1;
    dbms_output.put_line('Name salary job');
    dbms_output.put_line('--------------------');
    loop
     fetch c1 into emp_rec;
     exit when c1%notfound;
     if(c1%rowcount=3 or c1%rowcount=6 or c1%rowcount=7) then
     dbms_output.put_ line(emp_rec.ename ||' '|| emp_rec.sal ||' '||
      emp_ rec. job);
     end if;
     end loop;
    close c1;
  End;
```

**Output:**

```
SQL> /
```

| Name | Salary | Job |
|---|---|---|
| WARD | 1250 | SALESMAN |
| BLAKE | 3050 | MANAGER |
| CLARK | 2450 | MANAGER |

**Program 4: Write a PL/SQL block which assign comm=500 for those employee who are getting null comm.**

```
  Declare
    cursor c1 is select * from emp
        where comm is null;
    emp_rec c1%rowtype;
  Begin
    dbms_output.put_line('Name salary comm');
    dbms_output.put_line('-------------');
    for emp_rec in c1
    loop
     emp_rec.comm:=500;
      dbms_output.put_line(emp_rec.ename ||' '|| emp_rec.sal ||'
'|| emp_rec. comm);
    end loop;
  End;
```

**Output:**

```
SQL> /
```

| Name | Salary | Comm. |
|---|---|---|
| JONES | 2975 | 500 |
| BLAKE | 3050 | 500 |
| CLARK | 2450 | 500 |
| SCOTT | 3000 | 500 |
| KING | 5000 | 500 |
| ADAMS | 1100 | 500 |

**Note:** When cursor for loop is used then open, fetch and close operations are automatically performed.

## 2.8 TRIGGER (W-17, 18)

- A trigger is PL/SQL code block which is executed by an event which occurs to a database table.
- Triggers are implicitly called (executed) when INSERT, UPDATE or DELETE command is executed.
- A trigger is associated to a table or a view. When a view is used, the base table triggers are normally enabled.
- Triggers are stored as text and compiled at execute time, because of this it is not to include much code in them.
- You may not use COMMIT, ROLLBACK and SAVEPOINT statements within trigger blocks.
- The advantages of using trigger are:
  1. It creates consistency and access restrictions to the database.
  2. It implements the security.

### 2.8.1 What is Trigger? (S-17; W-17)

- A trigger is a PL/SQL block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table.
- A trigger is triggered automatically when an associated DML statement is executed.

### 2.8.2 How to Create a Trigger? (S-19)

- A trigger is created with CREATE TRIGGER command.
- **Syntax:**

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE / AFTER / INSTEAD OF}
{DELETE / INSERT / UPDATE [OF column [,column….]}
[OR {DELETE / INSERT / UPDATE [OF column
[,column….]}]]
ON {TABLE/VIEW}
```

```
FOR EACH  (ROW / STATEMENT)
(WHEN (condition))
PL/SQL block.
```

- Triggers may be called BEFORE or AFTER the following events. INSERT, UPDATE and DELETE.
  1. The BEFORE trigger is used when some processing is needed before execution of the command.
  2. The AFTER trigger is triggered only after the execution of the associated trigger commands.
  3. INSTEAD OF trigger is applied to view only. Triggers may be **ROW** or **STATEMENT** types.
- **ROW type trigger** which is also called as **ROW** level trigger is executed on all the rows that are affected by the command.
- **STATEMENT type trigger** (STATEMENT level trigger) is triggered only once. For example, if an DELETE command deletes 15 rows, the commands contained in the trigger are executed only once, and not with every processed row.
- The trigger can be activated by a SQL command or by system event or a user event which are called triggering events.

**Types of Triggers:**

- PL/SQL trigger consist of following types:
  1. **TABLE triggers:** Applied to DML commands (INSERT / DELETE / UPDATE)
  2. **SYSTEM EVENT triggers:** Such as startup, shutdown of the database and server error message event.
  3. **USER EVENT triggers:** Such as User logon and logoff, DDL commands (CREATE, ALTER, DROP), DML commands (INSERT, DELETE, UPDATE).
- WHEN clause is used to specify trigger restriction i.e. it specifies what condition must be true for the trigger to be activated.
- PL/SQL block is a trigger action. Thus, every trigger is divided into three components as:
  1. Triggering event.
  2. Triggering restriction.
  3. Triggering action.

**Access the Value of Column Inside a Trigger:**

- A value of a column of a ROW-LEVEL trigger can be accessed using NEW and OLD variable.
- **Syntax:**

```
Column_name: NEW
Column_name: OLD
```

- Depending on the commands INSERT, UPDATE and DELETE, the values NEW and OLD will be used as follows:
  1. **INSERT command:** The value of the fields that will be inserted must be preceded by: NEW
  2. **UPDATE command:** The original value is accessed with: OLD and the new value will be preceded by: NEW.
  3. **DELETE command:** The values in this case must be preceded by: OLD.

**For example:**

```
SQL> create trigger tr_sal
     before insert on emp
     for each row
     begin
     if :new.sal=0 then
     Raise_application_error('-20010','Salary should be greater
     than 0');
     end if;
     end;
SQL> /
Trigger created
```

- When you insert data into an emp table with salary 0 at that time this trigger will get executed.

## 2.8.3 Modify a Trigger

- A trigger can be modified using OR REPLACE clause of CREATE TRIGGER command.

**For example:**

```
SQL> create or replace trigger tr_sal
     before insert on emp
     for each row
     begin
     if :new.sal<=0 then
      Raise_application_error('-20010','Salary should be greater than
      0');
     end if;
     end;
SQL> /
Trigger created
```

- When you insert data into an emp table with salary 0 or less than 0 at that time this trigger will get executed.

## 2.8.4 Enable/Disable a Trigger

- To enable or disable a specific trigger, ALTER TRIGGER command is used.
- **Syntax:**

```
ALTER TRIGGER trigger_name ENABLE / DISABLE;
```

- When a trigger is created, it is automatically enabled and it gets executed according to the triggering command. To disable the trigger, use DISABLE option as,

```
ALTER TRIGGER tr_sal DISABLE;
```

- To enable or disable all the triggers of a table, ALTER TABLE command is used.
- **Syntax:**

```
ALTER TABLE table_name ENABLE / DISABLE ALL TRIGGERS;
```

**For example:**

```
ALTER TABLE emp DISABLE ALL TRIGGERS;
```

## 2.8.5 Delete a Trigger

- To delete a trigger, use DROP TRIGGER command.
- **Syntax:**

```
DROP TRIGGER trigger_name;
```

**For example:**

```
DROP TRIGGER  tr_sal;
```

## 2.9 PACKAGE (W- 17; S-18, S-19)

- A package is a collection of related functions, procedures and/or routines. Package may contain subprograms that can be called from trigger, procedure or function.
- Packages are used to group together PL/SQL code blocks which make up a common application or are attached to a single business function.
- A package improves machine performance, because it simultaneously transfers several objects to the memory.
- A *package* is a collection of PL/SQL objects that are grouped together.
- There are a number of benefits to using packages, including information hiding, object-oriented design, top-down design, object persistence across transactions, and improved performance.
- Elements that can be placed in a package include procedures, functions, constants, variables, cursors, exception names, and TYPE statements (for index-by tables, records, REF CURSORs, etc.).

## 2.9.1 Package Structure

- A package has two sections:
  1. Specification section
  2. Body section.

- Physically, the package specification section and the body section are separate.

1. **Specification Section:**

- The package specification lists the public interfaces to the blocks within the packa body.
- In this section, the names of procedures and functions are declared together with th variable and constant names which are included in the initialization.
- To create a package specification use the CREATE PACKAGE command as follows:
- **Syntax:**

```
CREATE [OR REPLACE] PACKAGE package_name
{IS/AS} PL/SQL_package _spec
```

- OR REPLACE is optional, it is use to re_create the package specification if it is alread exists. If a package specification changes, Oracle recomplies it.

**For example:**

```
CREATE OR REPLACE PACKAGE mypack AS
   PROCEDURE myproc (p_no in number);
   FUNCTION myfun (f_no in number)
   RETURN number;
END mypack;
```

- In this example of package(mypack) a procedure(myproc) and a functio (myfun) are declared. The END of the package is indicated as END package_name.

2. **Package Body:**

- The package body contains the PL/SQL blocks which defines procedures an functions referenced in the specification section.
- The package body is created with the CREATE PACKAGE BODY command as follows:
- **Syntax:**

```
CREATE [OR REPLACE] PACKAGE BODY package_name
{IS/AS} PL/SQL_package_body
```

- To create a package body we now specify each PL/SQL block that makes up th package, note that we are not creating these blocks separately (no CREATE O REPLACE is required for the procedure and function definitions).

**For example:**

```
CREATE OR REPLACE PACKAGE BODY mypack AS
-- Procedure mypack

PROCEDURE myproc(p_no in number)
IS
  Temp_sal number(10,2);
BEGIN
```

```
        Select  SAL into Temp_sal
          From emp
          Where empno=p_no;

      If Temp_sal > 1000 then
           Update emp
           Set  SAL=(Temp_sal*1.75)
           Where empno=p_no;
      Else
           Update emp
           Set  SAL=5000
           Where empno=p_no;
      End if;
Exception
      When NO_DATA_FOUND then
       Dbms_output.put_line('Empno does not exists');
End myproc;
-************
--Function myfun
FUNCTION myfun
(f_no in number)
RETURN number
IS
 v_sal number(10,2);
BEGIN
   Select SAL into v_sal
   From emp
   Where empno=f_no;
If  SQL%FOUND then
         RETURN(v_sal);
Else
         RETURN 0;
End if;
END myfun;

END mypack;
```

- The procedure in package is executed using EXECUTE command as:
  Execute mypack.myproc(7789);
- And the function is execute using SELECT command like:
  Select mypack.myfun(7698) from dual;

**For example:**

```
CREATE OR REPLACE PACKAGE time_pkg IS
   FUNCTION  GetTimestamp  RETURN DATE;
   PRAGMA RESTRICT_REFERENCES (GetTimestamp, WNDS);

   PROCEDURE ResetTimestamp;
END time_pkg;
CREATE OR REPLACE PACKAGE BODY time_pkg IS
   StartTimeStamp   DATE:= SYSDATE;
   -- StartTimeStamp is package data

   FUNCTION GetTimestamp RETURN DATE IS
   BEGIN
      RETURN StartTimeStamp;
   END GetTimestamp;

   PROCEDURE ResetTimestamp IS
   BEGIN
      StartTimeStamp:= SYSDATE;
   END ResetTimestamp;

END time_pkg;
```

**Package Data:**

- Data structures declared within a package specification or body, but outside any procedure or function in the package are *package data*.
- The scope of package data is your entire session; it spans transaction boundaries acting as global for your programs.
- Keep the following guidelines in mind as you work with package data:
  - The state of your package variables is not affected by COMMITs and ROLLBACKs.
  - A cursor declared in a package has global scope. It remains OPEN until you close it explicitly or your session ends.
  - A good practice is to *hide* your data structures in the package body and provide "get and set" programs to read and write that data. This technique protects your data.

## 2.9.2 Package Initialization

- The first time a user references a package element, the entire package is loaded into the SGA of the database instance to which the user is connected. That code is then shared by all sessions that have EXECUTE authority on the package.
- Any package data are then instantiated into the session's UGA (User Global Area), a private area in either the SGA or PGA (Program Global Area). If the package body contains an initialization section, that code will be executed.

- The initialization section is optional and appears at the end of the package body, beginning with a BEGIN statement and ending with the EXCEPTION section (if present) or the END of the package.
- The following package initialization section runs a query to transfer the user's minimum balance into a global package variable.
- Programs can reference the packaged variable (via the function) to retrieve the balance, rather than executing the query repeatedly.

```
CREATE OR REPLACE PACKAGE usrinfo
IS
   FUNCTION minbal RETURN VARCHAR2;
END usrinfo;
/
CREATE OR REPLACE PACKAGE BODY usrinfo
IS
   g_minbal NUMBER; -- Package data
   FUNCTION minbal RETURN VARCHAR2
      IS BEGIN RETURN g_minbal; END;
BEGIN  -- Initialization section
   SELECT minimum_balance INTO g_minbal
      FROM user_configuration
      WHERE username = USER;
EXCEPTION
   WHEN NO_DATA_FOUND
   THEN g_minbal:= NULL;
END usrinfo;
```

## Programs

(S-17,18,19; W-17,18)

**Program 1:** **Consider the following relational database**
**Doctor (dno, dname, dcity)**
**Hospital (hno, hname, hcity)**
**Doct — Hosp (dno, hno)**
**Write a function to return count of number of hospitals located in Pune city.**

**Answer :**

```
create or replace function f_hcount return number as
  hcnt number;
begin
  select count(*) into hcnt from hospital where hcity='PUNE';
  return hcnt;
end f_hcount;
/
```

**Output:**

```
SQL> select f_hcount from dual;
 F_HCOUNT
---------
        2
```

**Program 2:** **Consider the following relational database**
**Book (bno, bname, pubname, price)**
**Author (ano, aname)**
**Book — Author (bno, ano)**
**Define a trigger that restricts insertion or updation of books having price less than 0.**

**Answer:**

```
create or replace trigger book_price_trig before insert or update o
book for each row
declare
  bn number;
begin
if inserting then
  if :new.price<0 then
    raise_application_error('-20010','Cannot insert ! Price should
    be above 0');
  end if;
end if;
if updating then
  if :new.price<0 then
    raise_application_error('-20010','Cannot update ! Price should
    be above 0');
  end if;
end if;
end book_price_trig;
/
```

**Output:**

```
SQL> insert into book values(5,'ALGORITHMS','ABC',-500);
insert into book values(5,'ALGORITHMS','ABC',-500)
            *
ERROR at line 1:
ORA-20010: Cannot insert ! Price should be above 0
ORA-06512: at "SCOTT.BOOK_PRICE_TRIG", line 6
ORA-04088: error during execution of trigger 'SCOTT.BOOK_PRICE_TRIG'
```

```
SQL> insert into book values(5,'ALGORITHMS','ABC',500);
1 row created.

SQL> update book set price=price-1000 where bno=1;
update book set price=price-1000 where bno=1
       *
ERROR at line 1:
ORA-20010: Cannot update ! Price should be above 0
ORA-06512: at "SCOTT.BOOK_PRICE_TRIG", line 11
ORA-04088: error during execution of trigger 'SCOTT.BOOK_PRICE_TRIG'
```

**Program 3:** **Consider the following relational database**
**Customer (cno, cname, ccity)**
**Loan (lno, lamt, no_of years, cno)**
**Write a procedure to display total loan amount from Pune city.**

**Answer :**

```
create or replace procedure p_disp_amt
as
  t_loan number;
begin
  select sum(lamt) into t_loan from loan where cno in (select cno
from customer where ccity='PUNE');
  dbms_output.put_line('Total loan amount from PUNE city is :
'||t_loan);
end p_disp_amt;
/
```

**Output:**

```
SQL> execute p_disp_amt;
Total loan amount from PUNE city is : 500000
PL/SQL procedure successfully completed.
```

**Program 4:** **Consider the following relational database**
**Employee (eno, ename, city, deptname)**
**Project (pno, pname, status)**
**Employee_ Project (eno, pno, no_of days)**
**Write a cursor to display departmentwise details of employee working in the department.**

**Answer :**

```
declare
  cursor c_dept is select deptname,empno,empname,pname from
employee,
  project,employee_project where employee.empno=employee_project.eno
  and project.pno=employee_project.pno order by deptname;
```

```
  drec c_dept%rowtype;
  dname varchar(10);
  cnt int;
begin
open c_dept;
  dname:=' ';
loop
  fetch c_dept into drec;
  exit when c_dept%notfound;
  if(drec.deptname<>dname) then
    cnt:=1;
    dbms_output.put_line('--------------------');
    dbms_output.put_line(drec.deptname||'     '||cnt||')'||
    drec.empno||'  '||drec.empname||'  '||drec.pname);
    dname:=drec.deptname;
  else
    cnt:=cnt+1;
    dbms_output.put_line(RPAD('*',LENGTH(DREC.DEPTNAME),'  ')||
    '     '||cnt||')'||drec.empno||'  '||drec.empname||
    '  '||drec.pname);
  end if;
end loop;
close c_dept;
end;
/
```

**Output:**

```
--------------------
DEV     1)101  SAURAV  DMDW
*       2)101  SAURAV  DATABASE
*       3)102  SUNIL  WEB
*       4)103  CHINMAY  DATABASE
--------------------
QA      1)104  MANDAR  DATABASE
*       2)104  MANDAR  WEB
*       3)104  MANDAR  GAMING
*       4)105  AKSHAY  GAMING
*       5)106  AJAY  MULTIMEDIA
PL/SQL procedure successfully completed.
```

**Program 5:** **Write a package which consist of one procedure and one function. For this consider following relational database**
**Movie (mno, mname, releaseyear) Actor (ano, aname)**
**Movie — Actor (mno, ano)**
**(i) Pass movie number as a parameter to a procedure and display movie details.**
**(ii) Pass actor number as a parameter to a function and return total number of movies in which given actor is acting.**

**Answer:**

```
create or replace package mov_pack as
procedure p_disp_mov(mn IN number);
function f_mcount(an IN number) return number;
PRAGMA RESTRICT_REFERENCES(p_disp_mov, WNDS);
PRAGMA RESTRICT_REFERENCES(f_mcount, WNDS);
end mov_pack;
/
create or replace package body mov_pack as
procedure p_disp_mov(mn in number) as
  mrec movie%rowtype;
  cursor c1(mnm mov_act.mno%type) is select actor.ano,aname from
  actor,mov_act where actor.ano=mov_act.ano and mov_act.mno=mnm;
  an actor.ano%type;
  anm actor.aname%type;
begin
  select * into mrec from movie where mno=mn;
  dbms_output.put_line('  MNO:'||mrec.mno||'  MNAME:'||mrec.mname||
  '  RELEASE:'||mrec.relyear);
  open c1(mrec.mno);
  loop
    fetch c1 into an,anm;
    exit when c1%notfound;
    dbms_output.put_line('  ANO:'||an||'  ANAME:'||anm);
  end loop;
  close c1;
end p_disp_mov;

function f_mcount(an in number) return number as
  acnt number;
begin
  select count(*) into acnt from mov_act where ano=an;
  return acnt;
end f_mcount;
```

```
end mov_pack;
/
```

**Output:**

```
SQL> execute mov_pack.p_disp_mov(1);
MNO:1  MNAME:SHOLAY  RELEASE:1975
ANO:101  ANAME:AMITABH
ANO:103  ANAME:DHARMENDRA
PL/SQL procedure successfully completed.

SQL> select mov_pack.f_mcount(101) from dual;
MOV_PACK.F_MCOUNT(101)
----------------------
                     2
```

**Program 6: Consider the following Relational Database :**
**Customer (cno, cname, city)**
**Account (ano, acc – type, balance, cno)**
**Define a trigger that restricts insertion or updation of account having balance less than 100.**

**Answer :**

```
create or replace trigger customer_trig before insert or update on
acc for each row
begin
if updating then
  if :new.bal<100 then
    raise_application_error('-20010','Cannot update ! Balance should
    be above 100');
  end if;
end if;
if inserting then
  if :new.bal<100 then
    raise_application_error('-20010','Cannot insert! Balance should
    be above 100');
  end if;
end if;
end customer_trig;
/
```

**Output :**

```
SQL> insert into acc values(105,'SAVINGS',50,1);
insert into acc values(105,'SAVINGS',50,1)
            *
ERROR at line 1:
```

```
ORA-20010: Cannot insert! Balance should be above 100
ORA-06512: at "SCOTT.CUSTOMER_TRIG", line 9
ORA-04088: error during execution of trigger 'SCOTT.CUSTOMER_TRIG'

SQL> insert into acc values(105,'SAVINGS',100,1);
1 row created.

SQL> update acc set bal=50 where ano=105;
update acc set bal=50 where ano=105
       *
ERROR at line 1:
ORA-20010: Cannot update ! Balance should be above 100
ORA-06512: at "SCOTT.CUSTOMER_TRIG", line 4
ORA-04088: error during execution of trigger 'SCOTT.CUSTOMER_TRIG'

SQL> update acc set bal=150 where ano=105;
1 row updated.
```

**Program 7:** **Consider the following Relational Database :**
**Book (bno, bname, pubname, price)**
**Author (ano, aname)**
**Book-Author (bno, ano)**
**Write a cursor to display authorwise their book details.**

**Answer :**

```
declare
  cursor c_auth is select * from author order by aname;
  arec author%rowtype;
  bn book_author.bno%type;
  bnm book.bname%type;
  pub book.pubname%type;
  pr book.price%type;
  cursor c_book(an author.ano%type) is select book_author.bno,bname,
  pubname,price from book_author,book where book_author.ano=an and
  book.bno=book_author.bno order by book_author.bno;
begin
  open c_auth;
  loop
    fetch c_auth into arec;
    exit when c_auth%notfound;
    dbms_output.put_line('---------------------');
    dbms_output.put_line('Author Name : '||arec.aname);
    open c_book(arec.ano);
    loop
```

```
        fetch c_book into bn,bnm,pub,pr;
        exit when c_book%notfound;
        dbms_output.put_line('Book No:'||bn||'  Name: '||bnm||
        '  Publisher: '||pub||'  Price: '||pr);
      end loop;
      close c_book;
    end loop;
    close c_auth;
  end;
  /
```

**Output :**

```
--------------------
Author Name : BHARAT
Book No:1  Name: C PROGRAMMING  Publisher: ACTIVE  Price: 500
Book No:4  Name: C++ PROGRAMMING  Publisher: LUCKY  Price: 500
--------------------
Author Name : PRADIP
Book No:3  Name: C# PROGRAMMING  Publisher: LUCKY  Price: 900
--------------------
Author Name : SACHIN
Book No:2  Name: JAVA PROGRAMMING  Publisher: PROSPECTS  Price: 700
--------------------
Author Name : WILLIAM
Book No:3  Name: C# PROGRAMMING  Publisher: LUCKY  Price: 900
PL/SQL procedure successfully completed.
```

**Program 8: Consider the following Relational Database :**
**Party (party code, party name)**
**Politician (pno, pname, description, partycode)**
**Write a function which will take partycode as a parameter and return total number of politicians of a given party.**

**Answer :**

```
create or replace function tot_pol_count(pn in number) return number
as
  pcnt number;
begin
  select count(*) into pcnt from party,politician where
  party.partycode=politician.partycode and party.partycode=pn;
  return pcnt;
end tot_pol_count;
/
```

**Output :**

```
SQL> select tot_pol_count(1) from dual;
TOT_POL_COUNT(1)
----------------
               2
SQL> select tot_pol_count(2) from dual;
TOT_POL_COUNT(2)
----------------
     1
```

**Program 9: Consider the following Relational Database :**
**Student (rollno, name, class, totalmarks)**
**Teacher (tno, tname)**
**Student-Teacher (rollno, tno, subject)**
**Write a procedure to display details of all students of class 'FY'.**

**Answer :**

```
create or replace procedure disp_stud as
cursor c_class is select * from student where class='FY';
 srec student%rowtype;
cursor c_stud(rno student.rollno%type) is select
 student_teacher.tno,tname,subject from student_teacher,teacher,
 student where student.rollno=student_teacher.rollno and
 teacher.tno=student_teacher.tno and student_teacher.rollno=rno;
 tn student_teacher.tno%type;
 tnm teacher.tname%type;
 sub student_teacher.subject%type;
begin
 dbms_output.put_line('---------------------');
open c_class;
loop
 fetch c_class into srec;
 exit when c_class%notfound;
 dbms_output.put_line('Roll No.: '||srec.rollno||' Name: '||
 srec.name||' Marks: '||srec.tot_marks);
 open c_stud(srec.rollno);
 loop
  fetch c_stud into tn,tnm,sub;
  exit when c_stud%notfound;
  dbms_output.put_line('Teacher No: '||tn||
  '  Teacher Name: '||tnm||'  Subject: '||sub);
end loop;
close c_stud;
```

```
    dbms_output.put_line('--------------------');
end loop;
close c_class;
end disp_stud;
/
```

**Output :**

```
SQL> execute disp_stud;
--------------------
Roll No.: 11 Name: AMISH Marks: 650
Teacher No: 1  Teacher Name: RAJESH  Subject: DBMS
Teacher No: 2  Teacher Name: RAM  Subject: PROGRAMMING
Teacher No: 3  Teacher Name: AJIT  Subject: ALGORITHMS
--------------------
Roll No.: 12 Name: SWARAJ Marks: 750
Teacher No: 1  Teacher Name: RAJESH  Subject: DBMS
Teacher No: 2  Teacher Name: RAM  Subject: PROGRAMMING
Teacher No: 3  Teacher Name: AJIT  Subject: ALGORITHMS
--------------------
PL/SQL procedure successfully completed.
```

**Program 10: Write a package, which consists of one procedure and one function.**

**(i) Pass a number as a parameter to a procedure and print whether a number is odd or even.**

**(ii) Pass employee number as a parameter to a function and print salary of that employee. For this consider the following relation: Employee (eno, ename, eaddr, salary).**

**Answer :**

```
create or replace package emp_pack
as
procedure p_odd_even(num in number);
function f_sal(en in number) return number;
PRAGMA RESTRICT_REFERENCES(f_sal, WNDS);
end emp_pack;
/

create or replace package body emp_pack as
procedure p_odd_even(num in number) as
begin
  if mod(num,2)=1 then
    dbms_output.put_line(num||' is odd');
  else
    dbms_output.put_line(num||' is even');
```

```
  end if;
end p_odd_even;
function f_sal(en in number) return number as
  erec empl%rowtype;
begin
  select * into erec from empl where empno=en;
  return erec.sal;
end f_sal;
end emp_pack;
/
```

**Output :**

```
SQL> execute emp_pack.p_odd_even(1);
1 is odd
PL/SQL procedure successfully completed.

SQL> execute emp_pack.p_odd_even(2);
2 is even
PL/SQL procedure successfully completed.

SQL> select emp_pack.f_sal(101) from dual;
EMP_PACK.F_SAL(101)
-------------------
              25000
SQL> select emp_pack.f_sal(102) from dual;
EMP_PACK.F_SAL(102)
-------------------
              20000
```

**Program 11: Consider the following Relational Database :**
**Supplier (sid, sname, saddr)**
**Parts (pid, pnmae, pdesc)**
**Supp_Part (sid, pid, cost)**
**Define a trigger that restricts insertion or updation of Supp_Part having cost <= 0.**

**Answer :**

```
create or replace trigger trig_sup_part before insert or update on
supp_part for each row
begin
if inserting then
  if :new.cost<=0 then
    raise_application_error('-20010','Cannot insert ! Cost should be
    greater than zero');
```

```
    end if;
  end if;
  if updating then
    if :new.cost<=0 then
      raise_application_error('-20010','Cannot update ! Cost should be
      greater than zero');
    end if;
  end if;
  end;
  /
```

**Output:**

```
SQL> INSERT INTO SUPP_PART VALUES(1,103,0);
INSERT INTO SUPP_PART VALUES(1,103,0)
            *
ERROR at line 1:
ORA-20010: Cannot insert ! Cost should be greater than zero
ORA-06512: at "SCOTT.TRIG_SUP_PART", line 4
ORA-04088: error during execution of trigger 'SCOTT.TRIG_SUP_PART'

SQL> UPDATE SUPP_PART SET COST=0 WHERE SID=1 AND PID=101;
UPDATE SUPP_PART SET COST=0 WHERE SID=1 AND PID=101
       *
ERROR at line 1:
ORA-20010: Cannot update ! Cost should be greater than zero
ORA-06512: at "SCOTT.TRIG_SUP_PART", line 9
ORA-04088: error during execution of trigger 'SCOTT.TRIG_SUP_PART'

SQL> INSERT INTO SUPP_PART VALUES(1,103,100);
1 row created.

SQL> UPDATE SUPP_PART SET COST=150 WHERE SID=1 AND PID=103;
1 row updated.
```

**Program 12: Consider the following Relational Database:**
**Employee (eid, ename, eaddr)**
**Project (pid, prname, budget)**
**Emp_Proj (eid, pid, duration)**
**Write a cursor to display projectwise list of employees working on project.**

**Answer:**

```
declare
  cursor c_emp is select pname,budget,emp_1.eid,ename from emp_1,
  proj_1,emp_proj_1 where emp_1.eid=emp_proj_1.eid and
```

```
    proj_1.pid=emp_proj_1.pid order by pname;
    cnt number;
  begin
    dbms_output.put_line('PROJECT NAME'||RPAD(' ',18,' ')||'EID'||
    '  '||'EMP NAME');
    dbms_output.put_line(RPAD('*',50,'*'));
  for cur in c_emp
  loop
    cnt:=30-length(cur.pname);
    dbms_output.put_line(cur.pname||RPAD(' ',cnt,' ')||cur.eid||
    '  '||cur.ename);
  end loop;
  end;
  /
```

**Output :**

```
PROJECT NAME                    EID  EMP NAME
**************************************************
DATABASE                        101  SAURAV
DATABASE                        103  CHINMAY
DATABASE                        104  MANDAR
DMDW                            101  SAURAV
GAMING                          104  MANDAR
GAMING                          105  AKSHAY
MULTIMEDIA                      106  AJAY
WEB                             102  SUNIL
WEB                             104  MANDAR
PL/SQL procedure successfully completed.
```

**Program 13: Consider the following Relational Database :**
**Medical (mno, mname, city)**
**Drug (dno, dname, company, price)**
**Medial_Drug (mno, dno, quantity)**
**Write a procedure to display details of drugs available in medical number `123'.**

**Answer :**

```
create or replace procedure disp_drugs as
  cursor c_drug is select medical_drug.dno,dname,company,qty,price
  from drug,medical_drug where drug.dno=medical_drug.dno and
  medical_drug.mno=123;
  drec c_drug%rowtype;
  cnt1 number;
  cnt2 number;
begin
```

```
    dbms_output.put_line('DRUG NO.'||'  '||'DRUG NAME'||rpad(' ',11,
    '  ')||'COMPANY'||rpad(' ',11,' ')||'QTY'||'   '||'PRICE');
    dbms_output.put_line(rpad('*',60,'*'));
  open c_drug;
  loop
    fetch c_drug into drec;
    exit when c_drug%notfound;
    cnt1:=20-length(drec.dname);
    cnt2:=20-length(drec.company);
    dbms_output.put_line(drec.dno||'     '||drec.dname||
    rpad(' ',cnt1,' ')||drec.company||rpad(' ',cnt2,' ')||drec.qty||
    '  '||drec.price);
  end loop;
  close c_drug;
  exception
  when no_data_found then
    dbms_output.put_line('Medical number does not exist');
  end disp_drugs;
  /
```

**Output :**

```
SQL> execute disp_drugs;
DRUG NO.  DRUG NAME          COMPANY              QTY   PRICE
************************************************************
1001      PENICILLIN         HA                   100   200
1002      PARACETAMOL        CIPLA                700   150
1003      AMOXYCILLIN        RANBAXY              500   300
PL/SQL procedure successfully completed.
```

**Program 14: Consider the following Relational Database :**
**Bill (bill no, bill_date, bill amt**
**Item (item_no, item_name, price)**
**Bill_Item(bill_no, item_no, quantity)**
**Write a function which will take bill_no as a parameter and return tot number of items included in a given bill.**

**Answer :**

```
create or replace function disp_bill(bno in number) return number as
  tcnt number;
begin
  select count(*) into tcnt from bill_item where bill_no=bno;
  return tcnt;
end disp_bill;
/
```

**Output :**

```
SQL> select disp_bill(11) from dual;
DISP_BILL(11)
-------------
           3
```

**Program 15: Write a package which consists of one procedure and one function.**

**(i) Pass a number as a parameter to a procedure and print whether a number is positive or negative.**

**(ii) Pass customer number as a parameter to a function and return mobile number of that customer, for this consider the relation : Customer(cno, ename, caddy, cmob_no)**

**Answer :**

```
create or replace package cust_pack
as
procedure p_pos_neg(num in number);
function f_mob(cn in number) return number;
PRAGMA RESTRICT_REFERENCES(f_mob, WNDS);
end cust_pack;
/

create or replace package body cust_pack as
procedure p_pos_neg(num in number) as
begin
  if num<0 then
    dbms_output.put_line(num||' is negative');
  else
    dbms_output.put_line(num||' is positive');
  end if;
end p_pos_neg;
function f_mob(cn in number) return number as
  mrec customer%rowtype;
begin
  select * into mrec from customer where cno=cn;
  return mrec.c_mob;
end f_mob;
end cust_pack;
/
```

**Output :**

```
SQL> execute cust_pack.p_pos_neg(1);
1 is positive
PL/SQL procedure successfully completed.
```

```
SQL> execute cust_pack.p_pos_neg(-1);
-1 is negative
PL/SQL procedure successfully completed.

SQL> select cust_pack.f_mob(1) from dual;
CUST_PACK.F_MOB(1)
------------------
        1.235E+09
```

**Program 16: Consider the following relational database.**
**Employee (empno, empname, city, deptname)**
**Project (projno, proj name, status)**
**Emp-proj (empno, proj no, number-of-days)**
**Write a function which will return total number of employees working on any project for more than 60 days.**

**Answer:**

```
create or replace function f_cnt_emp return number as
   cnt number;
begin
   select count(*) into cnt from employee_project where nod>60;
   return cnt;
end f_cnt_emp;
/
```

**Output :**

```
SQL> select f_cnt_emp() from dual;
F_CNT_EMP()
-----------
          5
```

**Program 17: Consider the following relational database.**
**Department (deptno, deptname, location)**
**Employee (empno, empname, salary, commission, designation, deptno)Write a trigger for an employee table that restricts insertion or updation or deletion of ata on 'sunday'.**

**Answer:**

```
create or replace trigger empl_time_trig before insert or update
or delete on empl for each row
begin
   if(to_char(SYSDATE,'fmday')='sunday') then
   raise_application_error('-20010','Operation not allowed');
end if;
end empl_time_trig;
/
```

**Output:**

```
SQL> update empl set sal=10000 where empno=109;
update empl set sal=10000 where empno=109
*
ERROR at line 1:
ORA-20010: Operation not allowed
ORA-06512: at "SCOTT.EMPL_TIME_TRIG", line 3
ORA-04088: error during execution of trigger
'SCOTT.EMPL_TIME_TRIG'
If the command is executed on a day other than Sunday:
SQL> update empl set sal=10000 where empno=109;
1 row updated.
```

**Program 18: Consider the following relational database.**
**Book (bno, bname, pubname, price, dno)**
**Department (dno, dname)**
**Write a procedure which will display total expenditure on books by a given department.**

**Answer:**

```
create or replace procedure p_tot_exp(dpname in varchar) as tot
number;
  begin
    select sum(price) into tot from dept_info,book_info where
    dept_info.dno=book_info.dno and dname=dpname;
    dbms_output.put_line('Total Expenditure of '||dpname|| '
department is : '||tot);
  end p_tot_exp;
/
```

**utput:**

```
SQL> execute p_tot_exp('COMPUTER');
Total Expenditure of COMPUTER department is : 450
PL/SQL procedure successfully completed.
```

**ogram 19: Write a package which consist of one procedure and one function, consider relation student.**
**Student (Roll-no, stud-name, class, stud-addr, percentage) procedure of a package will display details of given student. Function of a package will count total number of students having percentage greater than 80 and class 'TYBCA'.**

**swer:**

```
create or replace package stud_pack_1 as
Procedure p_stud(num in number);
function f_stud_count return number;
PRAGMA RESTRICT_REFERENCES(f_stud_count, WNDS);
```

```
    end stud_pack_1;
    /
    create or replace package body stud_pack_1 as
    procedure p_stud(num in number) as
    srec stud_info%rowtype;
    begin
       select * into srec from stud_info where rollno=num;
       dbms_output.put_line('Name    :    '||srec.name||'    Address
'||srec.addr||' Class :'||srec.class||' Percentage :'||srec.perc);
    end p_stud;
    function f_stud_count return number as
       cnt number;
    begin
       select count(*) into cnt from stud_info where class='TYBCA'
       and perc>80;
       return cnt;
    end f_stud_count;
    end stud_pack_1;
    /
```

**Output:**

```
SQL> set serveroutput on;
SQL> execute stud_pack_1.p_stud(1);
Name : ANIL Address : PUNE Class :TYBCA Percentage :77.71
PL/SQL procedure successfully completed.
SQL> select stud_pack_1.f_stud_count from dual;
F_STUD_COUNT
------------
           2
```

**Program 20:** **Consider the following relational database.**
**Student (sno, sname, city, class)**
**Subject (subno, subname)**
**Stud-sub (sno, subno)'**
**Write a function which will take class as a parameter and will return total number of students.**

**Answer:**

```
create or replace function f_classtot(cname in varchar) return
number as
  cnt number;
begin
  select count(*) into cnt from stud where class=cname;
  return cnt;
end f_classtot;
/
```

**Output:**

```
SQL> select f_classtot('SYBCA') from dual;
F_CLASSTOT('SYBCA')
-------------------
                  3
```

**Program 21: Consider the following relational database**
**Publisher (pno, pname, pcity)**
**Book (bno, bname, price, pno)**
**Write a trigger which will restrict insertion or updation on price, price should not be less than zero.**

**Answer:**

```
create or replace trigger book_price_trig before insert or update on
book for each row
declare
  bn number;
begin
if inserting then
  if :new.price<0 then
    raise_application_error('-20010','Cannot insert ! Price should
    be above 0');
  end if;
end if;
if updating then
  if :new.price<0 then
    raise_application_error('-20010','Cannot update ! Price should
    be above 0');
  end if;
end if;
end book_price_trig;
/
```

**Output:**

```
SQL> insert into book values(5,'ALGORITHMS','ABC',-500);
insert into book values(5,'ALGORITHMS','ABC',-500)
            *
ERROR at line 1:
ORA-20010: Cannot insert ! Price should be above 0
ORA-06512: at "SCOTT.BOOK_PRICE_TRIG", line 6
ORA-04088: error during execution of trigger 'SCOTT.BOOK_PRICE_TRIG'

SQL> insert into book values(5,'ALGORITHMS','ABC',500);
1 row created.
```

```
SQL> update book set price=price-1000 where bno=1;
update book set price=price-1000 where bno=1
       *
ERROR at line 1:
ORA-20010: Cannot update ! Price should be above 0
ORA-06512: at "SCOTT.BOOK_PRICE_TRIG", line 11
ORA-04088: error during execution of trigger 'SCOTT.BOOK_PRICE_TRIG'
```

**Program 22: Consider the following relational database**
**Wholesaler (wno, wname, city)**
**Product (pno, pname, price)**
**Wp(wno, pno)**
**Write a cursor to display wholesalerwise product details.**

**Answer:**

```
declare
  cursor wcur is select * from wholesaler;
  cursor plist(wn varchar) is select wp.pno,pname,price from
  wholesaler,product,wp where wholesaler.wno=wp.wno and
  product.pno=wp.pno and wname=wn;
  wrec wholesaler%rowtype;
  pn wp.pno%type;
  pnm product.pname%type;
  prc product.price%type;
begin
open wcur;
loop
  fetch wcur into wrec;
  exit when wcur%notfound;
  dbms_output.put_line('Wholesaler Name : '||wrec.wname);
  open plist(wrec.wname);
  loop
    fetch plist into pn,pnm,prc;
    exit when plist%notfound;
    dbms_output.put_line('Product No. : '||pn||'  '||
    'Product Name : '||pnm||'  '||'Product Price : '||prc);
  end loop;
    dbms_output.put_line('-----------------------------------
                          -----');
    close plist;
end loop;
close wcur;
end;
/
```

**Output:**

```
Wholesaler Name : AMIT TRADERS
Product No. : 11  Product Name : STEEL BAR  Product Price : 340
Product No. : 12  Product Name : STEEL ANGLE  Product Price : 200
Product No. : 13  Product Name : STEEL ROOF  Product Price : 250
-----------------------------------------
Wholesaler Name : DIVYA AGENCY
Product No. : 12  Product Name : STEEL ANGLE  Product Price : 200
Product No. : 13  Product Name : STEEL ROOF  Product Price : 250
-----------------------------------------
Wholesaler Name : SWAPNIL TRADING CO
Product No. : 13  Product Name : STEEL ROOF  Product Price : 250
-----------------------------------------
PL/SQL procedure successfully completed.
```

**Program 23: Consider the following relational database**
**party (pcode, pname)**
**politician (pno, pname, pcity, pcode)**
**Write a procedure to display details of all politician of the given party.**

**Answer:**

```
create or replace procedure p_polit(pbname in varchar) as
  cursor bcur is select politician_info.* from
  party_info,politician_info where
  party_info.pcode=politician_info.pcode                    and
party_info.pname=pbname; prec politician_info%rowtype;
begin
  dbms_output.put_line('---------------------');
  dbms_output.put_line('PARTY : '||pbname);
  dbms_output.put_line('---------------------');
open bcur;
loop
  fetch bcur into prec;
  exit when bcur%notfound;
  dbms_output.put_line('PNO : '||prec.pno||'  '||
                'NAME    :    '||prec.pname||'        '||'CITY    :
'||prec.pcity);
end loop;
close bcur;
end p_polit;
/
```

```
SQL> execute p_polit('AJP');
--------------------
PARTY : AJP
--------------------
PNO : 101  NAME : ANIL  CITY : AMRITSAR
PNO : 102  NAME : JEEVAN  CITY : ALLAHABAD
PNO : 104  NAME : SHIRISH  CITY : PUNE
PL/SQL procedure successfully completed.
```

**Program 24: Write a package which consist of one procedure and one function. Pass a number as a parameter to a procedure and print whether no. is +ve or -ve. Pass students rollno as a parameter to a function and print percentage of student. For this consider the following relation : student (rollno, name, addr, total, per).**

**Answer:**

```
create or replace package stud_pack
as
procedure p_pos_neg(num in number);
function f_perc(rn in number) return number;
PRAGMA RESTRICT_REFERENCES(f_perc, WNDS);
end stud_pack;
/
create or replace package body stud_pack as
procedure p_pos_neg(num in number) as
begin
  if num<0 then
    dbms_output.put_line(num||' is negative');
  else
    dbms_output.put_line(num||' is positive');
  end if;
end p_pos_neg;
function f_perc(rn in number) return number as
  per number;
begin
  select perc into per from stud_info where rollno=rn;
  return per;
end f_perc;
end stud_pack;
/
```

**Output:**

```
SQL> execute stud_pack.p_pos_neg(-1);
-1 is negative
PL/SQL procedure successfully completed.
SQL> select stud_pack.f_perc(1) from dual;
STUD_PACK.F_PERC(1)
-------------------
              77.71
```

**Program 25: Consider the following relational database.**
**Employee (eno, ename, city, deptname)**
**Project (pno, pname, status)**
**Emp-proj (eno, pno, no-of-days)**
**Write a function which will return total number of employees working on given project.**

**Answer:**

```
create or replace function f_emp_proj(pn in varchar) return number
as
  cnt number;
begin
  select count(*) into cnt from employee,project,employee_project
  where employee.empno=employee_project.eno and
  project.pno=employee_project.pno and pname=pn;
  return cnt;
end f_emp_proj;
/
```

**Output:**

```
SQL> select f_emp_proj('DATABASE') from dual;
F_EMP_PROJ('DATABASE')
----------------------
                     3
```

**Program 26: Consider the following relational database.**
**student (roll_no, name, class, percentage)**
**teacher (tno, tname)**
**stud_teach(roll_no, tno, subject)**
**Write a trigger which will restrict insertion or updation of student having percentage greater than 100.**

**Answer:**

```
create or replace trigger student_trig before insert or update on
student for each row
begin
if updating then
```

```
    if :new.percentage>100 then
      raise_application_error('-20010','Cannot update ! Percentage
      should not be greater than 100');
    end if;
  end if;
  if inserting then
    if :new.percentage>100 then
      raise_application_error('-20010','Cannot    insert!    Percentage
  should
      not be greater than 100');
    end if;
  end if;
  end student_trig;
  /
```

**Output:**

```
SQL> insert into student values(15,'MANDAR','FY',101);
insert into student values(15,'MANDAR','FY',101)
            *
ERROR at line 1:
ORA-20010: Cannot insert! Percentage should not be greater than 100
ORA-06512: at "SCOTT.STUDENT_TRIG", line 9
ORA-04088: error during execution of trigger 'SCOTT.STUDENT_TRIG'
SQL> update student set percentage=110 where rollno=13;
update student set percentage=110 where rollno=13
       *
ERROR at line 1:
ORA-20010: Cannot update ! Percentage should not be greater than 100
ORA-06512: at "SCOTT.STUDENT_TRIG", line 4
ORA-04088: error during execution of trigger 'SCOTT.STUDENT_TRIG'
```

**Program 27: Consider the following relational database.**
**Customer (cno, cname, city)**
**Account (ano, acc_type, balance, cno)**
**Write a procedure which will display account and customer details of given account number.**

**Answer:**

```
create or replace procedure acc_cust_proc(acno in number) as
cn cust.cno%type;
cnm cust.cname%type;
ct cust.city%type;
act acc.actype%type;
balance acc.bal%type;
begin
```

```
    select cust.cno,cust.cname,cust.city,acc.actype,acc.bal into
    cn,cnm,ct,act,balance from cust,acc where cust.cno=acc.cno and
    acc.ano=acno;
    dbms_output.put_line('Cust No.:'||cn||' Name :'||cnm||
    ' City :'||ct||' Acc. Type :'||act||' Balance :'||balance);
  exception
  when no_data_found then
     dbms_output.put_line('Account Number does not exist');
  end acc_cust_proc;
  /
```

**Output:**

```
SQL> set serveroutput on;
SQL> execute acc_cust_proc(101);
Cust No.:1 Name :AMIT City :PUNE Acc. Type :SAVINGS Balance :300
PL/SQL procedure successfully completed.
```

**Program 28: Consider the following relational database**
**Book (bno, bname, pubname, price)**
**Dept_book (dno, bno)**
**Write a cursor to display details of all books purchased for a 'computer' department.**

**Answer:**

```
declare
  cursor bcur is select book_data.bno,bname,pubname,price from
  department,book_data,dept_book where department.dno=dept_book.dno
  and book_data.bno=dept_book.bno and dname='COMPUTER';
  brec book_data%rowtype;
begin
open bcur;
loop
  fetch bcur into brec;
  exit when bcur%notfound;
  dbms_output.put_line('-----------------------------------------
----------------------');
  dbms_output.put_line('Book No. : '||brec.bno||
  ' Book Name :'||brec.bname||' Publisher: '||brec.pubname||
  ' Price : '||brec.price);
  dbms_output.put_line('-----------------------------------------
----------------------');
end loop;
close bcur;
end;
/
```

**Output :**

```
---------------------------------------------------------------
Book No. : 11 Book Name : C PROGRAMMING Publisher: ABC Price : 300
---------------------------------------------------------------
---------------------------------------------------------------
Book No. : 12 Book Name : PHP PROGRAMMING Publisher: PQR Price : 250
---------------------------------------------------------------
PL/SQL procedure successfully completed.
```

**Program 29: Consider the following relational database :**
**Movie (mvno, mvname, releaseyear)**
**Write a package which will consist of one procedure and one function.**

**(i) Pass movie name as a parameter to producer and display details of movie.**

**(ii) Pass release year as a parameter to function and return total number of movies release in a given year.**

**Answer:**

```
CREATE OR REPLACE package movie_pack AS
PROCEDURE p_disp_mov(mn IN varchar);
FUNCTION f_mcount(ryear IN number) RETURN number;
PRAGMA RESTRICT_REFERENCES(p_disp_mov, WNDS);
PRAGMA RESTRICT_REFERENCES(f_mcount, WNDS);
END movie_pack;
/
CREATE OR REPLACE package body movie_pack AS
PROCEDURE p_disp_mov(mn IN varchar) AS
  mrec movie%rowtype;
  an actor.ano%type;
  anm actor.aname%type;
BEGIN
  select * into mrec from movie where mname=mn;
  dbms_output.put_line('  MNO:'||mrec.mno||'  MNAME:'||mrec.mname||
  '  RELEASE:'||mrec.relyear);
END p_disp_mov;

FUNCTION f_mcount(ryear IN number) return number AS
  mcnt number;
BEGIN
  select count(*) into mcnt from movie where relyear=ryear;
  return mcnt;
END f_mcount;
END movie_pack;
/
```

**Output:**

```
SQL> EXECUTE movie_pack.p_disp_mov('KRISH');
MNO:3  MNAME:KRISH  RELEASE:2010
PL/SQL procedure successfully completed.

SQL> select movie_pack.f_mcount(2010) from dual;
MOVIE_PACK.F_MCOUNT(2010)
-------------------------
                        1
```

**Program 30: Consider the following relational database**
**Item(itemno, itemname, qty)**
**Supplier(sno, sname, address, city, phno)**
**Item-supp(itemno, sno, rate, discount)**
**Write a trigger which will restrict insertion or updation of rate value. Item rate should not be less than zero.**

**Ansser:**

```
create or replace trigger rate_trig before insert or update on
item-supp for each row
begin
if inserting then
   if :new.rate < 0 then
   raise_application_error('-20010','Cannot Insert ! Rate should not
be less than zero');
   end if;
end if;
if updating then
   if :new.rate < 0 then
   raise_application_error('-20010','Cannot Update ! Rate should not
be less than zero');
   end if;
end if;
end rate_trig;
/
```

**Program 31: Consider the following relational database :**
**Company(c_no, c_name, c_addr, c,_city, c_ share-value)**
**Person(p_no, p_name, p_addr, p—city,' p_phone-no)**
**Comp-per(c_no, p_no, no. of shares),**
**Write a function which will take company name as a parameter and return number of persons who are shareholders of that company.**

**Answer:**

```
create or replace function f_comp_num(cmp in varchar) return number
as
pn number;
```

```
begin
  select count(*) into pn from comp,pers,comp_pers where
  comp.cno=comp_pers.cno and pers.pno=comp_pers.pno and cname=cmp;
return pn;
end f_comp_num;
/
```

**Program 32: Consider the following relational database**
**Customer(cno, cname, city)**
**Loan(lno, loan—amt, no_of years, cno)'**
**Write a cursor which will display details of all customers who have taken loan for more than 10 years.**

**Answer:**

```
declare
  crec cust%rowtype;
  cursor c1 is select cust.* from cust,loan where cust.cno=loan.cno
and
  loan.n_o_y >10;
begin
open c1;
loop
  fetch c1 into crec;
  exit when c1%notfound;
  dbms_output.put_line('CNO: '||crec.cno||'  CNAME: '||crec.cname||
  '  CITY: '||crec.city);
end loop;
close c1;
end;
/
```

**Program 33: Consider the following relational database**
**Book(bno, bname, pubname, price)**
**Author(ano, aname)**
**Book_Author(bno, ano)**
**Write a procedure which will display-book details along with author details of a given publisher.**

**Answer:**

```
create or replace procedure p_displ_book as
  bn book.bno%type;
  bnm book.bname%type;
  publ book.pubname%type;
  price book.price%type;
  anm author.aname%type;
```

```
  cursor  c1  is  select  book.bno,bname,pubname,price,aname  from
book,author,
  book_author where book.bno=book_author.bno and
  author.ano=book_author.ano and book.pubname='ACTIVE';
begin
open c1;
loop
  fetch c1 into bn,bnm,publ,price,anm;
  exit when c1%notfound;
  dbms_output.put_line('BNO: '||bn||'   BNAME: '||bnm||'   PUBLISHER:
'||publ||
'  PRICE: '||price||'   ANAME: '||anm);
end loop;
close c1;
end p_displ_book;
/
```

**Program 34: Consider the following relational database**
**Employee(emp_id, emp_name, join—date, salary, city, designation)**
**Write a package which will consist of, one procedure and one function. Pass emp_id as a parameter to a procedure and display employee details. Pass city as a parameter to a function and count total number of employees from given city.**

**Answer:**

```
create or replace package empl_pack as
procedure p_emp_dtl(num in number);
function f_city(ct in varchar) return number;
PRAGMA RESTRICT_REFERENCES(f_city, WNDS);
end empl_pack;
/
create or replace package body empl_pack as
Procedure p_emp_dtl(num in number) as
  erec  employee%rowtype;
begin
  select employee.* into erec from employee where empno=num;
  dbms_output.put_line('ENO'||erec.empno||' ENAME'||erec.empname||
'           JDATE'||erec.jdate||'              SALARY'||erec.salary||'
CITY'||erec,city||'DESG'||erec.desg);
end p_emp_dtl;

function f_city(ct in varchar) return number as
  cnt number;
begin
  select count(*) into cnt from employee where city=ct;
```

```
    return cnt;
  end f_city;
  end empl_pack;
  /
```

**Program 35:** **Consider the following relational database.**
**Employee (eno, ename, city, deptname)**
**Project (pno, pname, status)**
**Emp-proj (eno, pno, no-of-days)**
**Write a procedure which will take employees no as a parameter and display total no. of projects on which given employee works.**

**Answer:**

```
create or replace procedure tot_proj_count(en in number) as
  pcnt number;
begin
  select count(*) into pcnt from employee,project,
  employee_project where employee.empno=employee_project.eno and
  project.pno=employee_project.pno and employee.empno=en;
  dbms_output.put_line('Total no. of projects : '||pcnt);
end tot_proj_count;
  /
```

**Program 36:** **Consider the following relational database.**
**Book(bno, bname, pubname, price, dno)**
**Dept(dno, tname, location)**
**Write a function which will return total expenditure on books of a given department.**

**Answer:**

```
create or replace function f_tot_exp(dn IN varchar) return number as
   tot number;
   begin
   select sum(price) into tot from dept_info, book_info where
dept_info.dno=book_info.dno and dept_info.dname=dn;
   return tot;
   end f_tot_exp;
/
```

**Program 37:** **Consider the following relational database.**
**Dept(dno, dname, location)**
**Employee(empno, empname, salary, comm, designation, dno)**
**Define a trigger that will take care of the constraint that employee salary should not be less than zero.**

**Answer:**

```
create or replace trigger sal_trig before insert or update on emp_11
for each row
```

```
begin
if inserting then
  if :new.sal<0 then
    raise_application_error('-20010','Cannot Insert! Salary below
    zero');
  end if;
end if;
if updating then
  if :new.sal<0 then
    raise_application_error('-20010','Cannot Update! Salary below
    zero');
  end if;
end if;
end sal_trig;
/
```

**Program 38: Consider the following relational database:**
**Party (P-code, P-name)**
**Politician (pno, pname, designation, p-code)**
**Write a cursor to display details of all politician of 'BJP' party.**

**Answer:**

```
  declare
cursor c_pol is select politician.* from party, politician where
party.partycode=politician.partycode and party.partyname='BJP';
polrec politician%rowtype;
begin
open c_pol;
loop
  fetch c_pol into polrec;
  exit when c_pol%notfound;
  dbms_output.put_line('POLITICIAN NO. : '||polrec.pno||
  '  NAME : '||polrec.pname||'  DESIGNATION: '||polrec.desig);
end loop;
close c_pol;
end;
/
```

**Program 39: Write a package which consist of one procedure and one function.**
**Consider relation student:**
**Student(Roll-no, stud-name, class, stud-addr, percentage)**
**Procedure of a package will display details of given student function of a package will count total number of students having percentage >70 and class 'SYBCA'.**

**Answer:**

```
create or replace package stud_pack1 as
procedure p_stdt1(sn in number);
function f_perc return number;
PRAGMA RESTRICT_REFERENCES(p_stdt1, WNDS);
PRAGMA RESTRICT_REFERENCES(f_perc, WNDS);
end stud_pack1;
/
create or replace package body stud_pack1 as
procedure p_stdt1(sn in number) as
 rn stud_info.rollno%type;
 nm stud_info.name%type;
 addr stud_info.addr%type;
 perc stud_info.perc%type;
 class stud_info.class%type;
begin
 select rollno,name,addr,perc,class into rn,nm,addr,perc,class from
stud_info where rollno=sn;
 dbms_output.put_line('ROLL NO : '||rn||' NAME : '||nm||
' ADDRESS : '||addr||' PERCENTAGE : '||perc||' CLASS : '||class);
end p_stdt1;

function f_perc return number as
  cnt number;
begin
  select count(*)into cnt from stud_info where perc>70 and
  class='SYBCA';
  return cnt;
end f_perc;
end stud_pack1;
/
```

**Program 40: Consider the following Relational Database.**
**Customer (Cust_no, Cust_name, Cust_city)**
**Account (Acc_no, Acc-type, balance, Cust_no)**
**Write a procedure which will take balance as a parameter and will display customer name having account balance greater than or equal to given balance.**

**Answer:**

```
create or replace procedure displ_cust_info(pbal in number) as
cursor custcur is select cust.cname from cust, acc where
cust.cno=acc.cno and acc.bal>=pbal;
cnm cust.cname%type;
```

```
begin
dbms_output.put_line('Customers having balance greater than or equal
to '||pbal);
dbms_output.put_line('----------------------------------------
');
open custcur;
loop
   fetch custcur into cnm;
   exit when custcur%notfound;
   dbms_output.put_line(cnm);
   dbms_output.put_line('----------------------------------------');
end loop;
close custcur;
end displ_cust_info;
/
```

**Output:**

```
SQL> execute displ_cust_info(5000);
Customers having balance greater than or equal to 5000
----------------------------------------
Amit
----------------------------------------
Mandar
----------------------------------------
Mahesh
----------------------------------------
PL/SQL procedure successfully completed.
```

**Program 41: Consider the following relationship.**
**Publisher (P_no, P_name, P_add)**
**Book (book_no, book_name, price, P_no)**
**Write a function which will return total no. of books having price greater than 300.**

**Answer:**

```
create or replace function f_bcount return number as
bcnt number;
begin
select count(*) into bcnt from book where bprice>300;
return bcnt;
end f_bcount;
/
```

**Output:**

```
SQL> select f_bcount from dual;
 F_BCOUNT
--------
       2
```

**Program 42: Consider the following Relational Database.**
**Party(partycode, partyname)**
**Politician(pno, pname, description, partycode)**
**Write a cursor which will display partywise details of politician.**

**Answer:**

```
declare
cursor pcur is select * from party;
cursor plist(pn number) is select politician.* from party,politician
where
 party.partycode=politician.partycode and party.partycode=pn;
prec party%rowtype;
polrec politician%rowtype;
begin
open pcur;
loop
fetch pcur into prec;
exit when pcur%notfound;
dbms_output.put_line('Party Name : '||prec.partyname);
dbms_output.put_line('----------------------------');
dbms_output.put_line('Politician Name     '||'Description');
dbms_output.put_line('----------------------------');
open plist(prec.partycode);
loop
fetch plist into polrec;
exit when plist%notfound;
dbms_output.put_line(rpad(polrec.pname,20,'')||rpad(polrec.descr,20,
' '));
end loop;
dbms_output.put_line('----------------------------');
close plist;
end loop;
close pcur;
end;
/
```

**Output:**

```
Party Name : AJP
----------------------------
Politician Name     Description
----------------------------
Ramesh              MP
Anil                MLA
----------------------------
```

```
Party Name : INP
---------------------------
Politician Name      Description
---------------------------
Ramakant             MP
Mahesh               MLC
---------------------------
PL/SQL procedure successfully completed.
```

**Program 43: Consider the following Relational Database:**
**University (u_no, u_name, city)**
**College (c_no, c_name, city, establish_yr, u_no)**
**Write a trigger that restricts insertion of college record having year of establishment greater that current year.**

**Answer:**

```
create or replace trigger col_trig before insert on college for each
row
declare
   cyr number;
   begin
   cyr:=to_char(sysdate,'yyyy');
   if :new.estbl_yr > cyr then
      raise_application_error('-20010','Establishment Year should
   not be greater than current year');
   end if;
end;
/
```

**Output:**

```
SQL> insert into college values(1,'New Horizon
College','Pune',2019,1);
insert into college values(1,'New Horizon College','Pune',2019,1)
            *
ERROR at line 1:
ORA-20010: Establishment Year should not be greater than current
year
ORA-06512: at "SCOTT.COL_TRIG", line 6
ORA-04088: error during execution of trigger 'SCOTT.COL_TRIG'

SQL> insert into college values(1,'New Horizon
College','Pune',2018,1);
1 row created.
```

**Program 44: Write a package which will consist of one function and one procedure.**
**Consider relation :**
**Researcher(rno, rname, rcity)**
**(i) Write a function which will return total no. of researcher from 'PUNE' city.**
**(ii) Write a procedure which will display details of given researcher.**

**Answer:**

```
create or replace package research_pack
as
function f_rcount return number;
procedure p_disp_research(rn in varchar);
PRAGMA RESTRICT_REFERENCES(f_rcount, WNDS);
end research_pack;
/
create or replace package body research_pack as
function f_rcount return number as
rcnt number;
begin
  select count(*) into rcnt from researcher where rcity='PUNE';
  return rcnt;
end f_rcount;
procedure p_disp_research(rn in varchar) as
rec researcher%rowtype;
begin
  select * into rec from researcher where rname=rn;
  dbms_output.put_line('    RNO:'||rec.rno||'    NAME:'||rec.rname||'
CITY:'||rec.rcity);
end p_disp_research;
end research_pack;
/
```

**Output:**

```
SQL> select research_pack.f_rcount from dual;
 F_RCOUNT
---------
        2
SQL> execute research_pack.p_disp_research('Anil');
RNO:1  NAME:Anil  CITY:PUNE
PL/SQL procedure successfully completed.
```

**Program 45: Consider the following relational database:**
**Movie (mno, mname, releaseyear)**
**Actor (Ano, Aname)**
**Relation between Movie and Actor is Many to Many.**

**(i) Create or replace procedure, details of all movies of actor 'AKSHAY KUMAR' should be displayed.**

**Answer:**

```
create or replace procedure mov_details as
cursor mcur is select movie.* from movie,actor,movie_actor where
actor.ano=movie_actor.ano and movie.mno=movie_actor.mno and
actor.aname='AKSHAY KUMAR' ;
mrec movie%rowtype;
begin
  open mcur;
  loop
  fetch mcur into mrec;
  exit when mcur%notfound;
  dbms_output.put_line('Movie Number : '||mrec.mno||
  ' Movie Name   : '||mrec.mname||' Release Year : '||mrec.relyear);
  end loop;
  close mcur;
end mov_details;
/
```

**(ii) Write a cursor to display the moviename with their actorname.**

**Answer:**

```
declare
  cursor mcur is select mname,aname from movie,actor,movie_actor
where
  movie.mno=movie_actor.mno and actor.ano=movie_actor.ano order by
  movie.mno;
  mov_name varchar(20);
  actor_name varchar(30);
begin
open mcur;
loop
  fetch mcur into mov_name,actor_name;
  exit when mcur%notfound;
  dbms_output.put_line('Movie Name : '||mov_name||
                       ' Actor Name : '||actor_name);
end loop;
close mcur;
end;
/
```

**(iii) Write a trigger to restrict that movie release year must be entered after 2000.**

**Answer:**

```
create or replace trigger mov_trig before insert or update on movie
for each row
begin
if inserting then
  if :new.relyear<=2000 then
    raise_application_error('-20010','Cannot Insert! Year should be
> 2000');
  end if;
end if;
if updating then
  if :new.relyear<=2000 then
    raise_application_error('-20010','Cannot Update! Year should be
> 2000');
  end if;
end if;
end mov_trig;
/
```

**(iv) Write a function to find total number of movies acted by 'AISHWARYA RAI'.**

**Answer:**

```
create or replace function f_actor_tot return number as
  cnt number;
begin
  select count(*) into cnt from movie,actor,movie_actor where
  movie.mno=movie_actor.mno and actor.ano=movie_actor.ano and
  actor.aname='AISHWARYA RAI';
  return cnt;
end f_actor_tot;
/
```

**Program 46: Write a package which includes one procedure and one function:**

**(i) Write a function to calculate number of movies released in year 2005.**

**(ii) Write a procedure to display details of actor having actor no. is 1001.**

**Answer:**

```
create or replace package mov_pack
as
procedure p_actor;
function f_mov return number;
PRAGMA RESTRICT_REFERENCES(p_actor, WNDS);
PRAGMA RESTRICT_REFERENCES(f_mov, WNDS);
end mov_pack;
/
create or replace package body mov_pack as
```

```
procedure p_actor as
arec actor%rowtype;
begin
select actor.* into arec from actor where ano=1001;
dbms_output.put_line('Actor Number : '||arec.ano||
                     ' Actor Name : '||arec.aname);
end p_actor;
function f_mov return number as
  cnt number;
begin
  select count(*) into cnt from movie where relyear=2005;
  return cnt;
end f_mov;
end mov_pack;
/

SQL> execute mov_pack.p_actor;
Actor Number : 1001 Actor Name : AKSHAY KUMAR
PL/SQL procedure successfully completed.
SQL> select mov_pack.f_mov from dual;
   F_MOV
---------
     2
```

## Summary

- PL/SQL extends SQL by adding constructs found in procedural languages, that is more powerful than SQL.
- Every PL/SQL block is first executed by PL/SQL engine.
- PL/SQL is a block structured language. Blocks can also be nested.
- Each block is meant for a particular task.
- PL/SQL allows you to declare variables and constants.
- PL/SQL allows control structures.
- PL/SQL handles errors or exceptions during the execution of a PL/SQL program.
- The data types in PL/SQL are : character, numeric, date/time, large object, rowid etc.
- A PL/SQL block has three sections : Declaration(optional), Execution (compulsory), Exception(optional).
- Variables in a PL/SQL block are classified into two types, i.e. local variables(declared in a inner block), global variables(declared in a outer block).
- The %type attribute provides the datatype of a variable or database column.
- A variable can be declared with %rowtype that is equivalent to a row of a table.
- PL/SQL operators are classified into these types: arithmetic, comparison, relational, logical, string.

- There are two classes of exceptions, i.e. predefined and user-defined.
- PL/SQL exception message consists of three parts: type of exception, an error code, a message.
- There are two methods of defining exception by user: RAISE statement, RAISE_APPLICATION_ERROR statement.
- The main difference between a procedure and a function is that, a function must always return a value, but a procedure may or may not return a value.
- A cursor is an identifier for an area of memory which temporarily stores the data fetched by a SQL statement.
- There are two types of cursor: implicit, explicit.
- A trigger is a PL/SQL code block which is executed automatically when some event is occurred.
- Triggers are implicitly called when INSERT, UPDATE or DELETE command is executed.
- A package is a collection of related functions, procedures and/or routines. It is used to improve performance.

## Check Your Understanding

**I. Multiple Choice Questions:**

1. PL/SQL exception message has .....
   (a) type of exception (b) a message
   (c) an error code (d) all of these
2. Which of following allow you to select multiple rows and then process them individually?
   (a) Trigger (b) Cursor
   (c) Select (d) none of these
3. Which of the following exception is raised when select statement does not fetch any row from a database table?
   (a) NO_DATA_FOUND (b) NOT_FOUND
   (c) DATA_NOT_FOUND (d) none of these
4. Which of the following exception is raised when select statement returns more than one row?
   (a) MANY_ROWS (b) MANY_ROWS_FOUND
   (c) TOO_MANY_ROWS (d) ROWS_FOUND
5. If you don't specify a mode for a parameter, what is the default mode?
   (a) OUT (b) IN
   (c) DEFAULT (d) none of these
6. Which kind of parameters cannot have a DEFAULT value?
   (a) IN (b) OUT
   (c) CONSTANT (d) none of these

7. Which of the following is not a parameter mode for procedures?
   (a) IN (b) OUT
   (c) CONSTANT (d) none of these
8. Where do you declare an explicit cursor in PL/SQL?
   (a) Declaration Section (b) Body Section
   (c) Exception Section (d) none of these
9. How many different types of triggers can be defined on a table in PL/SQL?
   (a) 15 (b) 12
   (c) 10 (d) 9

**Ans.:** (1) d (2) b (3) a (4) c (5) b (6) b (7) c (8) a (9) b.

**II. State TRUE/FALSE.**

1. PL/SQL built in exception has an error code.
2. Stored procedure is called automatically.
3. PL/SQL is the superset of SQL.
4. PL/SQL is not a block structured language.
5. PL/SQL don't have exception section.
6. The Execution section of a PL/SQL block starts with keyword BEGIN.
7. Local variables are declared in an inner block and cannot be referenced by outside blocks.
8. %type attribute provides the datatype of a variable or database column.
9. TOO_MANY_ROWS is not a predefined exception.

**Answer:**

(1) TRUE (2) FALSE (3) TRUE (4) FALSE
(5) FALSE (6) TRUE (7) TRUE (8) TRUE
(9) FALSE

## Practice Questions

**Q1. Answer the following questions in brief.**

1. What is PL/SQL?
2. State advantages of PL/SQL.
3. What is an exception?
4. Which are the parts of PL/SQL block?
5. Explain in brief about trigger.
6. Write the syntax for creating a trigger.
7. Explain the difference between predefined and user defined exception.
8. What are the cursor attributes?
9. What is the difference between stored procedure and trigger?

**Q2. Answer the following questions.**

1. What is a package in PL/SQL? Explain in detail with proper example.

2. Write a procedure to change the salary to 15000 of an employee having emp_id 101.
3. Employee is a table having emp_id, emp_name, desg, salary.
4. What are the different data types in PL/SQL? Explain in detail.
5. What are the different operators in PL/SQL? Explain in detail.
6. Explain in detail about the character functions used in PL/SQL.
7. Explain in detail about the numeric functions used in PL/SQL.
8. What are the different control structures used in PL/SQL? Explain with proper syntax.
9. Write a PL/SQL block to find the maximum of three numbers.
10. What is a function? Explain with suitable example.

**Q.3 Define the terms:**

(a) Cursor
(b) Exception
(c) Declaration Section in PL/SQL
(d) Execution Section in PL/SQL
(e) Scope of variables
(f) LPAD function
(g) Constrained variables
(h) Implicit Cursor
(i) Explicit Cursor

## Previous Exams Questions

### Summer 2017

1. What is trigger? List types of trigger. [2 M]

**Ans.** Please refer Section 2.8.1.

2. Write syntax of nested if statement in PL/SQL with example. [2 M]

**Ans.** Please refer Section 2.3.3.

3. Explain the following predefined exception:
no-data-found, zero-divide, two-many-rows, duplicate-val-on-index. [4 M]

**Ans.** Please refer Section 2.4.2.

4 .What is PL/SQL? Explain different data types in PL/SQL. [4 M]

**Ans.** Please refer Section 2.1.1 and 2.2.

### Winter 2017

1. What is PL/SQL ? List the data types of PL/SQL. [2 M]

**Ans.** Please refer Section 2.1.1 and 2.2.

2. What is block ? List its types. [2 M]

**Ans.** Please refer Section 2.3.

3. What is trigger? List the types of trigger. [2 M]

**Ans.** Please refer Section 2.8.

4. What is cursor ? Explain various attributes of cursor with example. [4 M]

**Ans.** Please refer Section 2.7.

5. What is exception handling? Explain user defined exception with example. [4 M]

Ans. Please refer Section 2.4.2.

## Summer 2018

1. What is PL/SQL ? List the section of a PL/SQL block. [2 M]

Ans. Please refer Sections 2.1.1 and 2.3.

2. What is procedure in PL/SQL ? Give syntax of procedure. [2 M]

Ans. Please refer Section 2.6.

3. What is cursor ? Explain various attributes of cursor. [4 M]

Ans. Please refer Section 2.7.

4. Write a note on package in PL/SQL. [4 M]

Ans. Please refer Section 2.9.

## Winter 2018

1. What is PL/SQL? Draw a block diagram. [2 M]

Ans. Please refer Section 2.1.1 and 2.1.2.

2. What is cursor? Explain different types of cursor. [4 M]

Ans. Please refer Sections 2.7 and 2.7.2.

3. What is function? Explain with an example. [4 M]

Ans. Please refer Section 2.5.

4. Explain data types in PL/SQL. [4 M]

Ans. Please refer Section 2.2.

5. What is trigger? Describe types of trigger. [4 M]

Ans. Please refer Section 2.8.

## Summer 2019

1. Write the syntax for trigger. [2 M]

Ans. Please refer to 2.8.2.

2. Explain %type and %row type with an example. [4 M]

Ans. Please refer Section 2.3.1.

3. What is function? Explain with an example. [4 M]

Ans. Please refer Section 2.5.

4. What is exception handling? Explain system defined exceptions. [4 M]

Ans. Please refer Sections 2.4.1. and 2.4.2.

5. List RDBMS packages. Explain any one in detail. [4 M]

Ans. Please refer Section 2.9.

❖❖❖

# 3... Transaction Management

## Objectives...

- To understand the properties and states of transaction.
- To understand the concept of schedule and serializability.
- To know about the advantages of concurrent execution

## 3.1 INTRODUCTION

- Collection of operations that forms a single logical unit of work is called **transaction**.
- A transaction accesses and possibly updates various data items. After every transaction, the database should be in a consistent state.
- Usually a transaction is the result of execution of a user program, written in a high-level data manipulation language or programming language.
- Every transaction is delimited by statements or function calls of the form begin transaction and end transaction.
- A computer system is an electronic device and it is subject to failures of various types.
- During the execution of a transaction if some failure occurs, the transaction may result in some inconsistent state of database.
- Hence, the reliability of DBMS is linked to the reliability of computer system, and some solution must be there to deal with such computer system failures.
- Recovery system, the main component of Transaction management/Processing unit, deals with such failures. It makes the database fault tolerant.
- Number of transactions can be executed at the same time and they may be accessing the same database. Such concurrent access to the database, may result in some inconsistent state of database.
- Concurrency control/ management unit of transaction management preserves the consistency of database in case of concurrent accesses.

This chapter deals with the following topics:

1. Basic concepts of transaction,
2. Concurrency control, and
3. Recovery system.

## 3.2 TRANSACTION CONCEPTS (S-17,18,19; W-17,18)

- A transaction is a program unit whose execution accesses and possibly updates the contents of a database.
- If the database was in consistent state before a transaction, then after execution of transaction, the database will be remaining in a consistent state.
- In day-to-day life, a transaction can be defined as "an act of giving something to a person and receiving something from that person in return."
- For Example, If a customer wants to buy a commodity from a shopkeeper, the customer first pays an amount for that commodity and then receives the commodity from the shopkeeper in return. Here a customer cannot receive a commodity without paying amount. Similarly, after receiving an amount, the shopkeeper has to handover the commodity to customer. It means that, omission of any step will result into invalid transaction.
- Clearly, a transaction is an atomic (not splittable) unit. Either all instructions from transaction should be completed or none of the instruction should be completed.

## 3.3 TRANSACTION PROPERTIES (S-17, 19; W-17, 18)

- To ensure the integrity of data, database system maintains following properties of transaction.

1. **Atomicity:** Atomicity property ensures that at the end of the transaction, either no changes have occurred to the database or the database has been changed in a consistent manner. At the end of a transaction, the updates made by the transaction will be accessible to other transactions and processes outside the transaction.
2. **Consistency:** Consistency property of transaction implies that if the database was in consistent state before the start of a transaction, then on termination of a transaction, the database will also be in a consistent state.
3. **Isolation:** Isolation property of transaction indicates that action performed by a transaction will be hidden from outside the transaction until the transaction terminates. Thus each transaction is unaware of other transactions executing concurrently in the system.
4. **Durability:** Durability property of a transaction ensures that once a transaction completes successfully (commits), the changes it has made to the database persist, even if there are system failures.

These four properties are often called ACID (Atomicity, Consistency, Isolation, and Durability) properties of transaction.

## 3.3.1 Significance of ACID Properties

- Consider a banking system consisting of several accounts and a set of transactions that accesses and updates those accounts. Here consider that the database resides on disk, but some portion of database is temporarily stored in main memory.
- Following are the functions to access the database:
  - **read (X):** Which transfers the data item X from the database to a local buffer, belonging to the transaction that executed the read operation.
  - **write (X):** Which transfers the data item X from the local buffer of the transaction that executed the write back to the database.
- Let $T_i$ be a transaction that transfers ₹50 from account A to account B. This transaction can be defined as:

$T_i$ : Read (A);
A := A – 50;
write (A);
Read (B);
B:= B + 50;
write (B).

Let us now consider the significance of ACID properties.

- **Atomicity:** Suppose that just before the execution of transaction $T_i$, the values of account A and B are ₹ 100 and ₹ 200 respectively. If a failure has occurred during the execution of transaction $T_i$ and that prevented $T_i$ from completing its execution successfully. Suppose that failure happened after the write(A) operation was executed but before the write(B) operation was executed. In this case, the values of accounts A and B reflected in database are ₹ 50 and ₹ 200. But now A + B is no longer preserved and the database is in inconsistent state.

  But if the atomicity property is provided, all actions of the transaction are reflected in the database or none are reflected i.e. the database contents are ₹ 100 and ₹ 200 or ₹ 50 and ₹ 250.
- **Consistency:** Here, consistency requirement is that the sum of accounts A and B must be unchanged. It can be easily verified that if database is consistent before an execution of transaction and the database remains consistent after the execution of transaction. This task may be fascilitated by atomic testing of integrity constraints.
- **Isolation:** Even if the consistency and atomicity properties are ensured for each transaction, if several transactions are executed concurrently, their operations may interleave in some undesirable way. The isolation property ensures that the concurrent execution of transactions results in system state that is equivalent to a state that could have been obtained by execution of one transaction at a time. Each of these properties is ensured by components of transaction management unit.

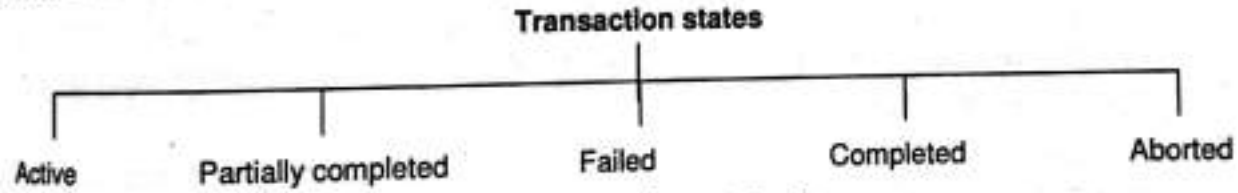
- **Durability:** The durability assures that once transaction completes successfully, all updates that it carried out on the database persist even if there is a system failure. We can guarantee durability by ensuring that either:
  1. The updates carried out by a transaction have been written to disk before the transaction completes.
  2. Information about updates carried out by the transaction and written to disk is sufficient to enable the database to reconstruct the updates when the database system is restarted after failure.

## 3.4 TRANSACTION STATES (S-17, 18, 19; W-17)

Following are the possible states of a transaction during its execution:

1. **Active:** Transaction is active when it is executing. This is the initial state of transaction.
2. **Partially committed:** When a transaction completes its final statement, it enters in partially committed state.
3. **Failed:** If the system decides that the normal execution of the transaction can no longer proceed, then transaction is termed as failed.
4. **Committed:** When the transaction completes its execution successfully it enters committed state from partially committed state.
5. **Aborted:** To ensure the atomicity property, changes made by failed transaction are undone i.e. the transaction is rolled back. After rollback, that transaction enters in Aborted state.

- A state is said to be terminated if it is Committed or Aborted.




- The state diagram corresponding to transaction states is:

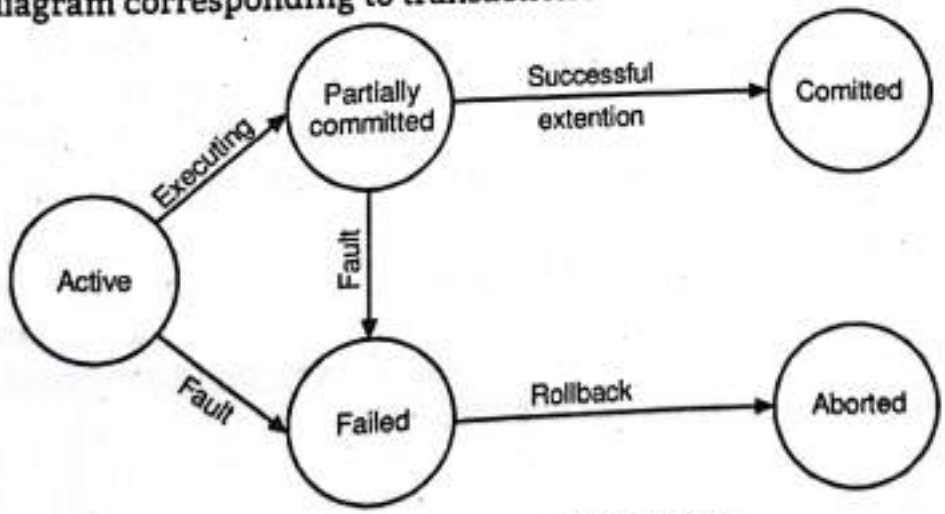



**Fig. 3.1: State diagram of Transaction**

- A transaction always starts with Active state. It remains in Active state till all commands of that transaction are executed.
- When it completes executing the last command of that transaction, it enters Partially committed state and when the transaction execution is successfully completed, and enters in Committed state.
- But, if some failure occurs in Active state or partially committed state transaction enters Failed state. When the transaction is in failed state, it rollbacks that transaction and enters in aborted state.
- When the transaction is in aborted state, the system has two options:

1. If the transaction was aborted as a result of some hardware or software error (software error which is not created because of some internal logic of transaction), then such transaction can be restarted and that transaction is considered to be a new transaction.
2. If the transaction was aborted because of some internal logical error which can be corrected only by rewriting of the application program or because of the input was bad or the desired data were not found in the database, then system can kill such transactions.

### 3.4.1 Schedule

(S-18, 19; W-18)

- Schedule represents the chronological order in which instructions are executed in the system.
- Two schedule types are Serial Schedule and Concurrent Schedule.
  1. **Serial Schedule:** It consists of a sequence of instructions from various transactions where the instructions belonging to one single transaction appear together in that schedule. Thus, for a set of n transactions, there exist n! different valid serial schedules.
  2. **Concurrent Schedule:** When several transactions are executed concurrently, the corresponding schedule is called Concurrent Schedule. Several execution sequences are possible since the various instructions from both transactions may now be interleaved. In general it is not possible to predict exactly how many instructions of a transaction will be executed before CPU switches to other transaction. Thus, the number of possible concurrent schedules for a set of n transactions is much larger than n !
- **Examples of Schedules:** Consider the banking system of several accounts and a set of transactions that accesses and updates those accounts. Let $T_1$ and $T_2$ be two transactions. Assume initial balances of A and B are ₹1000 and ₹2000 respectively.

  $T_1$: Transfers ₹ 50 from account A to account B.

  $T_1$: read (A);

  A = A − 50;

write (A);

read (B);

B = B + 50;

write (B).

$T_2$: Transfers 10% of balance from account A to B.

$T_2$: read (A);

temp: = A . 0.1;

A:= A – temp;

write (A);

read (B);

B:= B + temp;

write (B).

For $T_1$ and $T_2$, two serial schedules are possible:

1. **<$T_1$, $T_2$> Serial Schedule:** It will execute transaction $T_1$ first and then $T_2$.

After executing both transactions, account balance of A is ₹ 855 and account balance of B is ₹ 2145. i.e. <$T_1$ $T_2$> preserves A + B i.e. the database are consistent.

**Table 3.1: Schedule 1**

| $T_1$ | $T_2$ |
|---|---|
| read (A) | |
| A:= A – 50 | |
| write (A) | |
| read (B) | |
| B:= B + 50 | |
| Write (B) | |
| | read (A) |
| | temp:= A * 0.1 |
| | A:= A – temp |
| | write (A) |
| | read (B) |
| | B:= B + temp |
| | write (B) |

2. **<$T_2$, $T_1$> Serial Schedule:** It executes $T_2$ first and then $T_1$.

**Table 3.2: Schedule 2**

| $T_1$ | $T_2$ |
|---|---|
| | read (A)<br>temp:= A * 0.1<br>A:= A – temp<br>write (A)<br>read (B)<br>B:= B + temp<br>write (B) |
| read (A)<br>A:= A – 50<br>write (A)<br>read (B)<br>B:= B + 50<br>write (B) | |

- Here, after executing both the transactions, values of A and B are ₹ 850 and ₹ 2150 respectively. In this case also A + B is constant and it preserves the consistency.
- For $T_1$ and $T_2$, number of concurrent schedules is possible. But, not all the transactions are consistency preserving.
- A concurrent schedule for $T_1$ and $T_2$ is given in Table 3.3.

3. **Consistent concurrent schedule for $T_1$ and $T_2$:** This concurrent schedule preserves the consistency of database and A + B is constant.

**Table 3.3: Schedule 3**

| $T_1$ | $T_2$ |
|---|---|
| read (A)<br>A:= A – 50<br>write (A) | |
| | read (A)<br>temp:= A * 0.1<br>A:= A – temp<br>write (A) |
| read (B)<br>B:= B + 50<br>write (B) | |
| | read (B)<br>B:= B + temp<br>write (B) |

4. Inconsistent concurrent schedule for $T_1$ and $T_2$:

Table 3.4: Schedule 4 (concurrent schedule)

| $T_1$ | $T_2$ |
|---|---|
| read (A) | |
| A:= A – 50 | |
| | read (A) |
| | temp:= A * 0.1 |
| | A:= A – temp |
| | write (A) |
| | write (B) |
| write (A) | |
| read (B) | |
| B:= B + 50 | |
| write (B) | |
| | B:= B + temp |
| | write (B) |

## 3.5 CONCURRENT EXECUTION (S-19, W-17)

- The DBMS interleaves the actions of different transactions to improve performance, in terms of increased throughput or improved response times for short transactions, but not all interleaving should be allowed.
- Ensuring transaction isolation while permitting such concurrent execution is difficult but is necessary for performance reasons.
  - (i) While one transaction is waiting for a page to be read in from disk, the CPU can process another transaction. This is because I/O activity can be done in parallel with CPU activity in a computer. Overlapping I/O and CPU activity reduces the amount of time disks and processors are idle, and increases system throughput (the average number of transactions completed in a given time).
  - (ii) Interleaved execution of a short transaction with a long transaction usually allows the short transaction to complete quickly. In serial execution, a short transaction could get stuck behind a long transaction leading to unpredictable delays in response time, or average time taken to complete a transaction.

| $T_1$ | $T_2$ |
|---|---|
| Read (A)<br>Write (A) | |
| | Read (A)<br>Write (B) |
| Read (C)<br>Write (C) | |

**Problems of Concurrent Execution:**

Some Concurrency Problems in transaction are :

- Lost Update problem occurs when multiple transactions select the same row and update the row based on the value selected.
- The uncommitted dependency problem / The Temporary Update (or Dirty Read) Problem occurs when the second transaction selects a row which is updated by another transaction.
- The Non-Repeatable Read problem occurs when a second transaction is trying to access the same row several times and reads different data each time.
- The Incorrect Summary problem occurs when one transaction takes summary over the value of all the instances of a repeated data-item and second transaction updates few instances of that specific data-item. In that situation, the resulting summary does not reflect a correct result.

## 3.6 SERIALIZABILITY (S-17, 18; W-17, 18)

- For a transaction always a serial schedule results in a consistent database and not every concurrent schedule can result in consistent database.
- But a concurrent schedule results in a consistent state if its result is equivalent to a serial schedule of that transaction. Such concurrent schedule is known as serializable.
- **A serializable schedule** is defined as:

  Given (an interleaved execution) a concurrent schedule for n transactions; the following conditions hold for each transaction in the set.

  1. All transactions are correct i.e. if any one of the transactions is executed on a consistent database, the resulting database is also consistent.
  2. Any serial execution of the transactions is also correct and preserves the consistency of the database.
- There are two forms of serializability:

  (i) Conflict serializability.

  (ii) View serializability,

### 3.6.1 Conflict Serializability

- Consider that $T_1$ and $T_2$ are two transactions and S is a schedule for $T_1$ and $T_2$. $I_i$ and $I_j$ are two instructions. If $I_i$ and $I_j$ refer to different data items, then $I_i$ and $I_j$ can be executed in any sequence.
- But, if $I_i$ and $I_j$ refer to same data items then the order of two instructions may matter. Here, $I_i$ and $I_j$ can be a read or write operation only. Hence, following 3 conditions are possible.

  (i) $I_i$ = read (A)

  $I_j$ = read (A)

The order of $I_i$ and $I_j$ does not matter because both are reading the data.

(ii) $I_i$ = read (A) $I_j$ = write (A)

$I_i$ = write (A) $I_j$ = read (A)

Here, if read (A) is executed before write (A) then it will read the original value of A otherwise it will read that value of A which is written by write (A). Hence, the order of $I_i$ and $I_j$ matters.

(iii) $I_i$ = write (A) $I_j$ = write (A)

Here, order of $I_i$ and $I_j$ does not affect either $T_i$ or $T_j$. But the database is changed, and it makes difference for next read.

- We say that $I_i$ and $I_j$ conflict if they are operated by different transactions on the same data item and at least one of them is write operation. i.e. only in case 1, $I_i$ and $I_j$ do not conflict.
- Consider an example of concurrent schedule 5.

**Table 3.5: Concurrent schedule 5**

| $T_1$ | $T_2$ |
|---|---|
| read (A) | |
| write (A) | |
| | read (A) |
| | write (A) |
| read (B) | |
| write (B) | |
| | read (B) |
| | write (B) |

- Here, write (A) of $T_1$ conflicts with read (A) of $T_2$, similarly write (B) of $T_1$ conflicts with read (B) of $T_2$. But write (A) of $T_2$ does not conflict with read (B) of $T_1$ because both are accessing different data items.
- If $I_i$ and $I_j$ are two consecutive instructions of schedule S and if they do not conflict, then we can swap the order of $I_i$ and $I_j$, to produce new schedule S'. We say that S and S' are equivalent since all instructions appear in the same order except for $I_i$ and $I_j$ whose order does not matter.
- Equivalent schedule for a schedule given in Table 3.5 can be obtained by following swap.
- Swap write (A) of $T_2$ with read (B) of $T_1$.

**Table 3.6: Schedule 6 (Schedule after swapping instructions)**

| $T_1$ | $T_2$ |
|---|---|
| Read (A) | |
| write (A) | |
| | read (A) |
| read (B) | |
| | write (A) |
| write (B) | |
| | read (B) |
| | write (B) |

- Similarly swap the following in schedule given in Table 3.6.
  1. read (B) instruction and read (A) instruction of $T_1$ and $T_2$ respectively.
  2. write (B) instruction and write (A) instruction of $T_1$ and $T_2$ respectively.
  3. write (B) instruction and read (A) instruction of $T_1$ and $T_2$ respectively.
- The final schedule S' after these swapping is given below:

**Table 3.7: Schedule 7**

| $T_1$ | $T_2$ |
|---|---|
| read (A) | |
| write (A) | |
| read (B) | |
| write (B) | |
| | read (A) |
| | write (A) |
| | read (B) |
| | write (B) |

- Which is a serial schedule of $T_1$ and $T_2$. Thus concurrent schedule S is transferred to serial schedule S' by a series of swaps of non-conflicting instructions and schedules S and S' are conflict equivalent.
- We say that a schedule S is conflict serializable, if it is conflict equivalent to a serial schedule. Consider the schedule shown in Table 3.8.

**Table 3.8: Schedule 8**

| $T_1$ | $T_2$ |
|---|---|
| read (Q) | |
| | write (Q) |
| read (Q) | |

- This schedule is not conflict serializable, since it is not conflict equivalent to any serial schedule $<T_1, T_2>$ or $<T_2, T_1>$.
- There may be any two schedules which are not conflict equivalent but produce same outcome. Schedule given in Table 3.9 is not conflict serializable.

**Table 3.9: Schedule 9 (Concurrent schedule)**

| $T_1$ | $T_5$ |
|---|---|
| read (A) | |
| A:= A – 50 | |
| write (A) | |
| | read (B) |
| | B:= B – 10 |
| | write (B) |
| read (B) | |
| B:= B + 50 | |
| write (B) | |
| | read (A) |
| | A:= A + 10 |
| | write (A) |

- Result of above schedule is same as serial schedule $<T_1, T_5>$, but this is not conflict serializable, since, in above schedule write (B) of $T_5$ conflicts with read (B) of $T_1$. Thus, we cannot move all instructions of $T_1$ before those of $T_5$ by swapping consecutive non-conflicting instructions.

## 3.6.2 View Serializability (S-17)

- Consider two schedules S and S', where same set of transactions participate in both schedules.
- The schedules S and S' are said to be view equivalent if the following three conditions are satisfied:

1. For each data item Q if transaction $T_i$ reads the initial value of Q in schedule S, then transaction $T_i$ in schedule S' must also read the initial value of Q.
2. For each data item Q if transaction $T_i$ executes read (Q) in schedule S, and that value was produced by transaction $T_j$ (if any), then transaction $T_i$ in schedule S', must also read the value of Q that was produced by $T_j$ transaction.
3. For each data item Q, the transaction that performs the final write (Q) operation in schedule S must perform the final write (Q) operation in schedule S'.

- A schedule S is **view serializable** if it is view equivalent to a serial schedule.
- **Example of view equivalence:** Schedule 1 is not view equivalent to schedule 2 since, in schedule 1 the value of account A read by transaction $T_2$ was produced by $T_1$, whereas this is not the case in schedule 2. Schedule 1 is view equivalent to schedule 3, because values of account A and B read by transaction $T_2$ were produced by $T_1$ in both schedules.

- **Example of view serializable schedule:** A schedule given in Table 3.10 is view serializable schedule.

**Table 3.10: Schedule 10**

| $T_3$ | $T_4$ | $T_6$ |
|---|---|---|
| read (A) | | |
| | write (A) | |
| write (A) | | |
| | | write (A) |

This schedule is view equivalent to serial schedule $<T_3, T_4, T_6>$.

**Note:** Transactions $T_4$ and $T_6$ perform write (A) operations without having performed a read (A) operation. Writes of this form are called blind writes.

- Every conflict serializable schedule is view serializable, but there are view serializable schedules that are not conflict serializable.
- A view serializable schedule in which blind write appear is not a conflict serializable.
- Schedule 10 is view serializable but it is not conflict serializable.

**Testing for Serializability:**

- A serializability schedule gives same result as some serial schedule.
- A serial schedule always gives correct result. i.e. a schedule that is serializable schedule is always correct.
- Hence, we must show that schedules generated by concurrency control scheme are serializable. This section deals with methods for determining conflict and view serializability.

**1. Conflict Serializability:**

- There is an algorithm to establish the serializability of a given schedule for a set of transactions.
- This algorithm uses a directed graph called precedence graph, constructed from given schedule.
- **Precedence graph:** It consists of a pair G = (V, E) where,

  V – set of vertices.
- The set of vertices consists of all transactions participating in the schedule.

  E – the set of edges.
- The set of edges consists of all edges $T_i \rightarrow T_j$ for which one of the following three conditions hold:

1. $T_i$ executes write (Q) before: $T_j$ executes read (Q)
2. $T_i$ executes read (Q) before: $T_j$ executes write (Q)
3. $T_i$ executes write (Q) before: $T_j$ executes write (Q)

- A precedence graph is said to be acyclic if there are no cycles in the graph otherwise it is a cyclic graph.

**Algorithm: Conflict serializability**

**Step 1** : Construct a precedence graph G for given schedule S.

**Step 2** : If the graph G has a cycle, schedule S is not conflict serializable.

- If the graph is acyclic, then find, using the topological sort given below, a linear ordering of transactions, so that if there is arc from $T_i$ to $T_j$ in G, $T_i$ precedes $T_j$.
- Find a serial schedule as follows:

(i) Initialize the serial schedule as empty.

(ii) Find a transaction $T_i$, such that there are no arcs entering $T_i$, $T_i$ is the next transaction in the serial schedule.

(iii) Remove $T_i$ and all edges emitting from $T_i$. If the remaining set is non-empty, return to step (ii), otherwise the serial schedule is complete.

**Examples:**

1. **Given schedule is:**

**Table 3.11**

| $T_{11}$ | $T_{12}$ | $T_{13}$ |
|---|---|---|
| read (A) | | |
| | read (B) | |
| $A := f_1(A)$ | | |
| | | read (C) |
| | $B := f_2(B)$ | |
| | write (B) | |
| | | $C := f_3(C)$ |
| | | write (C) |
| write (A) | | |
| | read (A) | |
| | $A := f_4(A)$ | |
| read (C) | | |
| | write (A) | |
| $C := f_5(C)$ | | |
| write (A) | | |
| | | $B := f_6(B)$ |
| | | write (B) |

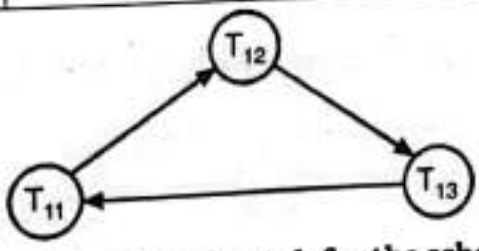



**Fig. 3.2: Precedence graph for the schedule**

- This graph contains cycle. Hence, the schedule is not conflict serializable.

2. **The given schedule is:**

**Table 3.12**

| $T_{14}$ | $T_{15}$ | $T_{16}$ |
|---|---|---|
| Read (A) | | |
| A:= $f_1$ (A) | | |
| Read (C) | | |
| write (A) | | |
| A:= $f_2$ (C) | | |
| | Read (B) | |
| write (C) | | |
| | Read (A) | |
| | | Read (C) |
| | B:= $f_3$ (B) | |
| | write (B) | |
| | | C:= $f_4$ (C) |
| | | Read (B) |
| | | write (C) |
| | A:= $f_5$ (A) | |
| | write (A) | |
| | | B:= $f_6$ (B) |
| | | write (B) |

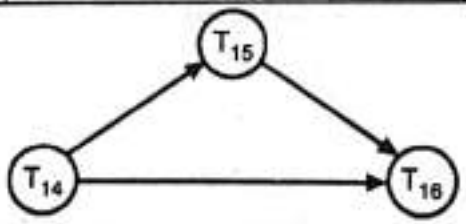



**Fig. 3.3: The precedence graph for the given schedule**

- The graph is acyclic. The conflict equivalent serial schedule for given schedule can be obtained using step 2 of algorithm.
- $T_{14}$ is the transaction with no arcs entering in $T_{14}$. Hence $T_{14}$ is the first transaction in serial schedule. Remove $T_{14}$ and all edges emitting from $T_{14}$.
- $T_{15}$ is the next schedule, since it has no incoming edges. Remove $T_{15}$ and edges emitting from $T_{15}$.
- $T_{16}$ is the last schedule. Hence the serial schedule which is conflict equivalent to given schedule is:

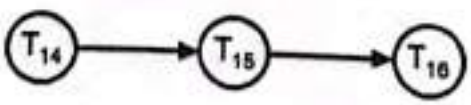



**Fig. 3.4: Precedence graph**

- Hence, the schedule is conflict serializable.

3. Consider schedule 3, the precedence graph for it is given as:


**Fig. 3.5: Precedence graph for schedules**

- Since, $T_1$ executes write (A) and write (B) before $T_2$ executes read (A) and read (B).
- This graph is acyclic and the conflict equivalent serial schedule is $T_1 \rightarrow T_2$. Hence, schedule 3 is conflict serializable.

4. Consider schedule 4, the precedence graph for it is:


**Fig. 3.6: Precedence graph for schedules**

which is a cyclic graph and hence it is not conflict serializable.

5. Consider the precedence graph given in Fig. 3.7.

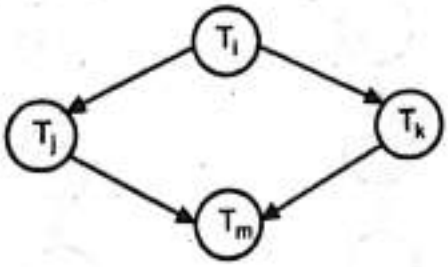
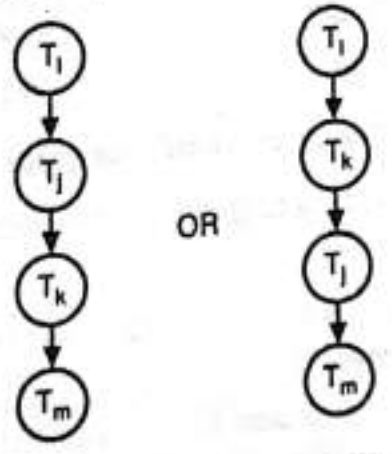



**Fig. 3.7: Precedence graph**

- The graph is acyclic. The conflict equivalent serial schedule is equivalent to this are:




(a) (b)

**Fig. 3.8: Precedence graph**

- Hence, the schedule corresponding to precedence graph in Fig. 3.8 is conflict serializable.

6. Consider the precedence graph in Fig. 3.9.

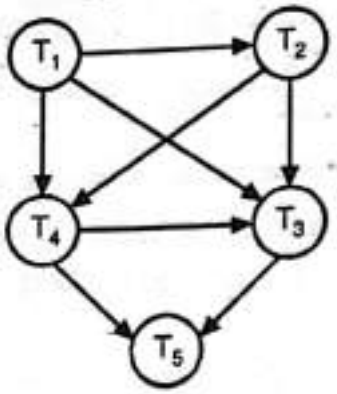



**Fig. 3.9: Precedence graph**

- The graph is acyclic. The serial schedules that are conflict equivalent to given schedule are:

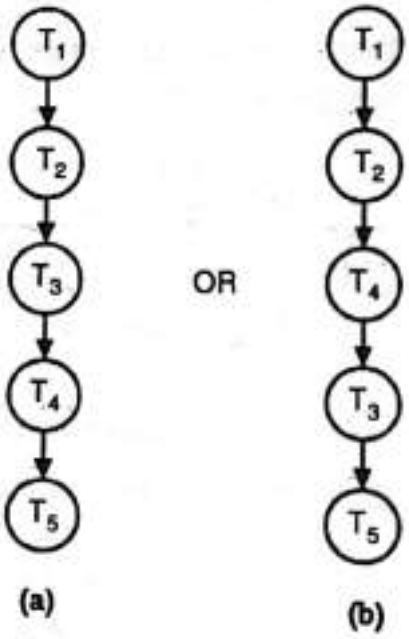



**Fig. 3.10: Precedence graph**

- Hence the schedule corresponding to given precedence graph is conflict serializable.

**2. View Serializability**

- Testing for view serializability is complicated. It has been shown that testing for view serializability is itself NP-complete.
- Thus there exists no algorithm to test for view serializability. However concurrency control schemes can still use sufficient conditions for view serializability.
- That is if sufficient conditions are satisfied, the schedule is view serializable schedule. But there may be view serializable schedules that do not satisfy the sufficient conditions.

# 3.7 RECOVERABILITY

## 3.7.1 Recoverable Schedule

- A recoverable schedule is one where for each pair of transactions $T_i$ and $T_j$ such that $T_j$ reads a data item previously written by $T_i$, the commit operation of $T_i$ appears before the commit operation of $T_j$. Otherwise the schedule is non-recoverable.
- Consider the schedule given in Table 3.13. Transaction $T_9$ reads the data written by $T_8$. commit of transaction $T_8$ occurs after commit of transaction $T_9$. Hence, it is a non-recoverable schedule.

**Table 3.13**

| $T_8$ | $T_9$ |
|---|---|
| Read(A) | |
| Write(A) | |
| | Read(A) |
| Read(B) | |

## 3.7.2 Cascadeless Schedules

(S-17, W-17)

- Even if a schedule is recoverable, to recover correctly from the failure of a transaction $T_i$, we may have to rollback the transaction.
- Consider the partial schedule shown below:

| $T_{10}$ | $T_{11}$ | $T_{12}$ |
|---|---|---|
| Read(A) | | |
| Read(B) | | |
| Write(A) | | |
| | Read(A)<br>Write(A) | Read(A) |

- Transaction $T_{10}$ writes a value of A that is read by transaction $T_{11}$. Transaction $T_{11}$ writes a value of A that is read by transaction $T_{12}$. Suppose that at a point transaction $T_{10}$ fails. $T_{10}$ must be rolled back. Since, $T_{11}$ is dependent on $T_{10}$, $T_{11}$ must be rolled back. Since, $T_{12}$ is dependent on $T_{11}$, $T_{12}$ must be rolled back.
- This concept, in which a single transaction failure results in a series of transaction rollbacks, is called **cascading rollback.**
- A **cascadeless schedule** is one where, for each pair of transactions $T_i$ and $T_j$ such that $T_j$ reads a data item previously written by $T_i$, the commit operation of $T_i$ appears before commit operation of $T_j$.

## SOLVED EXAMPLES

(S-17, 18, 19;W-17, 18)

**Example 1:** Consider the following Transaction. Give two non-serial schedules that the serializable :

| $T_1$ | $T_2$ |
|---|---|
| Read (X) | Read (Z) |
| X = X + 100 | Read (X) |
| Write (X) | X = X – Z |
| Read (Y) | Write (X) |
| Read (Z) | Read (Y) |
| Y = Y + Z | Y = Y – 100 |
| Write (Y) | Write (Y) |

**Ans:**

(i)

| $T_1$ | $T_2$ |
|---|---|
| Read(X) | |
| X=X+100 | |
| Write(X) | |
| | Read(Z) |
| | Read(X) |
| | X=X-Z |
| | Write(X) |
| Read(Y) | |
| Read(Z) | |
| Y=Y+Z | |
| Write(Y) | |
| | Read(Y) |
| | Y=Y-100 |
| | Write(Y) |

(ii)

| $T_1$ | $T_2$ |
|---|---|
| Read(X) | |
| X=X+100 | |
| Write(X) | |
| | Read(Z) |
| Read(Y) | |
| Read(Z) | |
| | Read(X) |
| | X=X-Z |
| | Write(X) |
| Y=Y+Z | |
| Write(Y) | |
| | Read(Y) |
| | Y=Y-100 |
| | Write(Y) |

**Example 2:** Consider the following transactions. Give two non-serial schedules that are serializable.

| $T_1$ | $T_2$ |
|---|---|
| Read (Y) | Read (X) |
| Read (a) | Read (a) |
| Y = Y + a | X = X + a |
| Write (Y) | Write (X) |
| | Read (Y) |
| | Y = Y + a |
| | Write (Y) |

Ans:

(i)

| $T_1$ | $T_2$ |
|---|---|
| Read(Y) | |
| Read(a) | |
| | Read(X) |
| | Read(a) |
| Y=Y+a | |
| Write(Y) | |
| | X=X+a |
| | Write(X) |
| | Read(Y) |
| | Y=Y+a |
| | Write(Y) |

(ii)

| $T_1$ | $T_2$ |
|---|---|
| | Read(X) |
| | Read(a) |
| | X=X+a |
| | Write(X) |
| Read(Y) | |
| Read(a) | |
| Y=Y+a | |
| Write(Y) | |
| | Read(Y) |
| | Y=Y+a |
| | Write(Y) |

**Example 3:** **Consider the following Transaction. Give two non-serial schedules that the serializable :**

| $T_1$ | $T_2$ |
|---|---|
| Read (A) | Read (B) |
| A = A – 1000 | B = B + 100 |
| Write (A) | Write (B) |
| Read (B) | Read (C) |
| B = B – 100 | C = C + 100 |
| Write (B) | Write (C) |

Ans:

(i)

| $T_1$ | $T_2$ |
|---|---|
| Read(A) | |
| A=A-1000 | |
| Write(A) | |
| | Read(B) |
| | B=B+100 |
| | Write(B) |
| Read(B) | |
| B=B-100 | |
| Write(B) | |
| | Read(C) |
| | C=C+100 |
| | Write(C) |

(ii)

| $T_1$ | $T_2$ |
|---|---|
| Read(A) | |
| A=A-1000 | |
| | Read(B) |
| Write(A) | |
| | B=B+100 |
| | Write(B) |
| Read(B) | |
| B=B-100 | |
| | Read(C) |
| Write(B) | |
| | C=C+100 |
| | Write(C) |

**Example 4:** **Consider the following transactions. Give two non-serial schedules that are serializable.**

| $T_1$ | $T_2$ |
|---|---|
| Read(x) | Read(x) |
| x=x+10 | x=x-10 |
| Write(x) | Write(x) |
| Read(y) | Read(y) |
| y=y+20 | y=y-20 |
| Write(y) | Write(y) |
| Read(z) | |
| z=z+30 | |
| Write(z) | |

Give two non serial schedules that are serializable.

**Ans:**

**(i)**

| $T_1$ | $T_2$ |
|---|---|
| Read(x) | |
| x=x+10 | |
| Write(x) | |
| | Read(x) |
| | x=x-10 |
| | Write(x) |
| Read(y) | |
| y=y+20 | |
| Write(y) | |
| | Read(y) |
| | y=y-20 |
| | Write(y) |
| Read(z) | |
| z=z+30 | |
| Write(z) | |

**(ii)**

| $T_1$ | $T_2$ |
|---|---|
| Read(x) | |
| x=x+10 | |
| Write(x) | |
| | Read(x) |
| Read(y) | |
| | x=x-10 |
| | Write(x) |
| y=y+20 | |
| Write(y) | |
| | Read(y) |
| Read(z) | |
| | y=y-20 |
| | Write(y) |
| z=z+30 | |
| Write(z) | |

**Example 5:** **Consider the following Transaction. Give two non-serial schedules that are serializable to serial schedule $<T_1, T_2, T_3>$.**

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| Read (a) | Read (c) | Read (b) |
| a = a + 100 | Read (b) | b = b + 100 |
| Write (a) | b = b + c | Write (b) |
| Read (b) | Write (b) | Read (c) |
| b = b – 100 | Read (a) | c = c – 100 |
| Write (b) | a = a – c | Write (c) |
| | Write (a) | |

Ans:

(i)

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| Read(a) | | |
| a=a+100 | | |
| Write(a) | | |
| Read(b) | | |
| | Read(c) | |
| b=b-100 | | |
| Write(b) | | |
| | Read(b) | |
| | b=b+c | |
| | Write(b) | |
| | | Read(b) |
| | | b=b+100 |
| | | Write(b) |
| | Read(a) | |
| | a=a-c | |
| | Write(a) | |
| | | Read(c) |
| | | c=c-100 |
| | | Write(c) |

(ii)

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| Read(a) | | |
| a=a+100 | | |
| Write(a) | | |
| | Read(c) | |
| Read(b) | | |
| b=b-100 | | |
| Write(b) | | |
| | Read(b) | |
| | b=b+c | |
| | Write(b) | |
| | | Read(b) |
| | | b=b+100 |
| | | Write(b) |
| | Read(a) | |
| | | Read(c) |
| | a=a-c | |
| | Write(a) | |
| | | c=c-100 |
| | | Write(c) |

**Example 6:** **Consider the following transactions. Give two non-serial schedules that are serializable to serial schedule <$T_1$, $T_2$>.**

| $T_1$ | $T_2$ |
|---|---|
| Read (c) | Read (c) |
| Read (a) | Read (a) |
| a = a – c | a = a + c |
| Write (a) | Write (a) |
| Read (b) | |
| b = b – c | |
| Write (b) | |

**Ans:**

**(i)**

| $T_1$ | $T_2$ |
|---|---|
| Read(c) | |
| Read(a) | |
| a=a-c | |
| Write(a) | |
| | Read(c) |
| | Read(a) |
| Read(b) | |
| | a=a+c |
| | Write(a) |
| b=b-c | |
| Write(b) | |

**(ii)**

| $T_1$ | $T_2$ |
|---|---|
| Read(c) | |
| Read(a) | |
| | Read(c) |
| a=a-c | |
| Write(a) | |
| | Read(a) |
| Read(b) | |
| b=b-c | |
| Write(b) | |
| | a=a+c |
| | Write(a) |

**Example 7: Consider the following transactions. Give two non-serial schedules that are serializable.**

| $T_1$ | $T_2$ |
|---|---|
| Read (A) | Read (A) |
| A=A+ 1000 | A = A – 1000 |
| Write (A) | Write (A) |
| Read (C) | Read (B) |
| C=C-1000 | B=B-1000 |
| Write (C) | Write (B) |
| Read (B) | |
| B =B + 1000 | |
| Write (B) | |

**Ans.:** Following are two serializable schedules among the various concurrent (non-serial) schedules that are serializable.

**(i)**

| $T_1$ | $T_2$ |
|---|---|
| Read(A) | |
| A=A+1000 | |
| Write(A) | |
| Read(C) | |
| C=C-1000 | |
| Write(C) | |
| | Read(A) |
| | A=A-1000 |
| | Write(A) |
| Read(B) | |
| B=B+1000 | |
| Write(B) | |
| | Read(B) |
| | B=B-1000 |
| | Write(B) |

**(ii)**

| $T_1$ | $T_2$ |
|---|---|
| | Read(A) |
| | A=A-1000 |
| | Write(A) |
| Read(A) | |
| A=A+1000 | |
| Write(A) | |
| Read(C) | |
| C=C-1000 | |
| Write(C) | |
| | Read(B) |
| | B=B-1000 |
| | Write(B) |
| Read(B) | |
| B=B+1000 | |
| Write(B) | |

**Example 8:** **Consider the following transactions. Give two non-serial schedules that are serializable.**

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| Read(A) | Read (C) | Read (B) |
| A=A+100 | Read (B) | B = B + 200 |
| Write(A) | B = B + C | Write (B) |
| Read(B) | Write (B) | Read (C) |
| B=B+100 | Read (A) | C = C + 200 |
| Write (B) | A = A – C | Write (C) |
| | Write (A) | |

**Ans.:** Following are two serializable schedules among the various concurrent (non-serial) schedules that are serializable.

i)

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| Read(A) | | |
| A=A+100 | | |
| Write(A) | | |
| Read(B) | | |
| | Read(C) | |
| B=B+100 | | |
| Write(B) | | |
| | Read(B) | |
| | B=B+C | |
| | Write(B) | |
| | | Read(B) |
| | | B=B+200 |
| | | Write(B) |
| | Read(A) | |
| | A=A-C | |
| | Write(A) | |
| | | Read(C) |
| | | C=C+200 |
| | | Write(C) |

ii)

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| Read(A) | | |
| A=A+100 | | |
| Write(A) | | |
| | Read(C) | |
| Read(B) | | |
| B=B+100 | | |
| Write(B) | | |
| | Read(B) | |
| | B=B+C | |
| | Write(B) | |
| | | Read(B) |
| | | B=B+200 |
| | | Write(B) |
| | Read(A) | |
| | | Read(C) |
| | A=A-C | |
| | Write(A) | |
| | | C=C+200 |
| | | Write(C) |

**Example 9: Consider the following transactions. Give two non-serial schedules that are serializable.**

| $T_1$ | $T_2$ |
|---|---|
| Read (X) | Read (z) |
| X = X + 1000 | Read (Y) |
| Write (X) | Y = Y – Z |
| Read (Y) | Write (Y) |
| Y = Y – 10 | Read (X) |
| Write (Y) | X = X + Z |
| | Write (X) |

**Ans. :**

**i)**

| $T_1$ | $T_2$ |
|---|---|
| Read(X) | |
| X = X + 10 | |
| Write(X) | |
| | Read(Z) |
| Read(Y) | |
| Y = Y – 10 | |
| Write(Y) | |
| | Read(Y) |
| | Y = Y – Z |
| | Write(Y) |
| | Read(X) |
| | X = X + Z |
| | Write(X) |

**ii)**

| $T_1$ | $T_2$ |
|---|---|
| Read(X) | |
| X = X + 10 | |
| | Read(Z) |
| Write(X) | |
| Read(Y) | |
| Y = Y – 10 | |
| Write(Y) | |
| | Read(Y) |
| | Y = Y – Z |
| | Write(Y) |
| | Read(X) |
| | X = X + Z |
| | Write(X) |

**Example 10: Consider the following non-serial schedule. Is this schedule serializable ?**

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| Read (A) | | |
| | Read (A) | |
| Write (A) | | |
| | | Read (A) |
| | | Write(A) |

**Ans.:** The schedule given in the problem is serializable. The schedule is serializable and produces the result identical to $T_2 \rightarrow T_1 \rightarrow T_3$.

**Example 11: Consider the following transactions. Find out two non-serial schedules that are serializable.**

| $T_1$ | $T_2$ |
|---|---|
| Read (P) | Read (P) |
| P = P + 100 | P = P – 100 |
| Write (P) | Write (P) |
| Read (Q) | Read (Q) |
| Read (R) | Q = Q – 200 |
| Q = Q + 200 | Write (Q) |
| Write (Q) | |
| R = R + 300 | |
| Write (R) | |

Ans. :

(i)

| $T_1$ | $T_2$ |
|---|---|
| Read(P) | |
| P=P+100 | |
| Write(P) | |
| | Read(P) |
| | P=P-100 |
| | Write(P) |
| Read(Q) | |
| Read(R) | |
| Q=Q+200 | |
| Write(Q) | |
| R=R+300 | |
| Write(R) | |
| | Read(Q) |
| | Q=Q-200 |
| | Write(Q) |

(ii)

| $T_1$ | $T_2$ |
|---|---|
| | Read(P) |
| | P=P-100 |
| | Write(P) |
| Read(P) | |
| P=P+100 | |
| Write(P) | |
| | Read(Q) |
| | Q=Q-200 |
| | Write(Q) |
| Read(Q) | |
| Read(R) | |
| Q=Q+200 | |
| Write(Q) | |
| R=R+300 | |
| Write(R) | |

**Example 12: Consider the following non-serial schedule. Is this schedule serializable ?**

| T1 | T2 |
|---|---|
| Read (x) | |
| Read (m) | |
| x = x + m | |
| | Read (n) |
| | Read (x) |
| | x = x + n |
| Write (x) | |
| | Write (x) |
| Read (y) | |
| y = y + m | |
| | Write (y) |

**Ans. : The given schedule is not serializable to serial schedule <$T_1$,$T_2$>.**

By considering initial values for x=100, n=10, m=20, y=200, the serial schedule <T1,T2> after execution gives x=130 and y=220.

The given concurrent schedule after execution gives X=110 and Y=220.

Therefore, it is not serializable to serial schedule <T1,T2>.

The given schedule is also not serializable to serial schedule <T2,T1>.

By considering initial values for x=100, n=10, m=20, y=200, the serial schedule <T2,T1> after execution gives x=130 and y=220.

The given concurrent schedule after execution gives X=110 and Y=220.

**Example 13: Consider the following non-serial schedule. Is this schedule serializable ?**

| $T_1$ | $T_2$ |
|---|---|
| Read (A) | |
| A= A+ 1000 | |
| Write (A) | |
| | Read (B) |
| | Read (C) |
| | B = B + 500 |
| | Write (B) |
| Read (B) | |
| B = B – 1000 | |
| Write (B) | |
| | C = C – 500 |
| | Write (C) |

Ans. :

i)

| $T_1$ | $T_2$ |
|---|---|
| Read(A) | |
| A = A + 1000 | |
| | Read(B) |
| Write(A) | |
| | Read(C) |
| | B = B + 500 |
| | Write(B) |
| Read(B) | |
| B = B − 1000 | |
| | C = C − 500 |
| | Write(C) |
| Write(B) | |

→

ii)

| $T_1$ | $T_2$ |
|---|---|
| | Read(B) |
| | Read(C) |
| | B = B + 500 |
| | Write(B) |
| Read(A) | |
| A = A + 1000 | |
| Write(A) | |
| | C = C − 500 |
| | Write(C) |
| Read(B) | |
| B = B − 1000 | |
| Write(B) | |

→

iii)

| $T_1$ | $T_2$ |
|---|---|
| | Read(B) |
| | Read(C) |
| | B = B + 500 |
| | Write(B) |
| Read(A) | |
| A = A + 1000 | |
| Write(A) | |
| | C = C − 500 |
| | Write(C) |
| Read(B) | |
| B = B − 1000 | |
| Write(B) | |

**Example 14: Consider the following transactions. Find out two non-serial schedules that are serializable**

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| Read (x) | Read (y) | Read (z) |
| Read (y) | Read (z) | Read (x) |
| y = y * x | z = z * y | x = x * z |
| Write (y) | Write (z) | Write (x) |

Ans. :

(i)

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| | Read(y) | |
| Read(x) | | |
| | Read(z) | |
| | z = z * y | |
| | Write(z) | |
| | | Read(z) |
| Read(y) | | |
| y = y * x | | |
| Write(y) | | |
| | | Read(x) |
| | | x = x * z |
| | | Write(x) |

(ii)

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| | | Read(z) |
| | Read(y) | |
| | | Read(x) |
| | | x = x * z |
| | | Write(x) |
| Read(x) | | |
| | Read(z) | |
| | z = z * y | |
| | Write(z) | |
| Read(y) | | |
| y = y * x | | |
| Write(y) | | |

**Example 15: Consider the following transactions.**

| $T_1$ | $T_2$ |
|---|---|
| Read(z) | Read(x) |
| z = z + 100 | Read(y) |
| Write(z) | y = y − x |
| Read(y) | Write(y) |
| y = y − 100 | |
| Write(y) | |

**Give two non-serial schedules that are serializable.**

**Ans. :**

**i)**

| $T_1$ | $T_2$ |
|---|---|
| | Read(x) |
| Read(z) | |
| z=z+100 | |
| Write(z) | |
| | Read(y) |
| | y=y-x |
| | Write(y) |
| Read(y) | |
| y=y-100 | |
| Write(y) | |

**ii)**

| $T_1$ | $T_2$ |
|---|---|
| Read(z) | |
| z=z+100 | |
| Write(z) | |
| | Read(x) |
| Read(y) | |
| y=y-100 | |
| Write(y) | |
| | Read(y) |
| | y=y-x |
| | Write(y) |

**Example 16 : Consider the following transaction:**

| $T_1$ | $T_2$ |
|---|---|
| Read(A) | Read(A) |
| A = A + 1000 | A = A − 1000 |
| Write(A) | Write(A) |
| | Read(B) |
| | B = B + 1000 |
| Write(B) | |

**Give two non-serial schedules that are serializable.**

**Ans. :**

**i)**

| $T_1$ | $T_2$ |
|---|---|
| | Read(A) |
| | A=A-1000 |
| | Write(A) |
| Read(A) | |
| A=A+1000 | |
| Write(A) | |
| | Read(B) |
| | B=B+1000 |
| | Write(B) |

**ii)**

| $T_1$ | $T_2$ |
|---|---|
| | Read(A) |
| | A=A-1000 |
| | Write(A) |
| Read(A) | |
| A=A+1000 | |
| | Read(B) |
| | B=B+1000 |
| | Write(B) |
| Write(A) | |

**Example 17: Consider the following transactions.**

| $T_1$ | $T_2$ |
|---|---|
| Read (Z) | Read (X) |
| Z = Z * 10 | Read (Z) |
| Write (Z) | X = X + Z |
| Read (Y) | Write (X) |
| Read (Z) | |
| Y = Y + Z | |
| Write (Y) | |

**Give two Non-Serial Schedules that are serializable.**

**Ans. :**

**(i)**

| $T_1$ | $T_2$ |
|---|---|
| Read(Z) | |
| Z=Z*10 | |
| Write(Z) | |
| | Read(X) |
| | Read(Z) |
| | X=X+Z |
| | Write(X) |
| Read(Y) | |
| Read(Z) | |
| Y=Y+Z | |
| Write(Y) | |

**(ii)**

| $T_1$ | $T_2$ |
|---|---|
| Read(Z) | |
| Z=Z*10 | |
| Write(Z) | |
| | Read(X) |
| Read(Y) | |
| Read(Z) | |
| | Read(Z) |
| | X=X+Z |
| | Write(X) |
| Y=Y+Z | |
| Write(Y) | |

**Example 18: Consider the following transaction:**

| $T_1$ | $T_2$ |
|---|---|
| Read (A) | Read (B) |
| A = A * 5 | B = B * 5 |
| Write (A) | Write (B) |
| Read (B) | Read (A) |
| Read (C) | A = A * 10 |
| B = B * 10 | Write (A) |
| Write (B) | |
| C = C * 5 | |
| Write (C) | |

**Give two Non-Serial Schedules that are serializable.**

**Ans. :**

(i)

| $T_1$ | $T_2$ |
|---|---|
| Read(A) | |
| A=A*5 | |
| Write(A) | |
| Read(B) | |
| Read(C) | |
| B=B*10 | |
| Write(B) | |
| | Read(B) |
| | B=B*5 |
| | Write(B) |
| C=C*5 | |
| Write(C) | |
| | Read(A) |
| | A=A*10 |
| | Write(A) |

(ii)

| $T_1$ | $T_2$ |
|---|---|
| Read(A) | |
| A=A*5 | |
| Write(A) | |
| Read(B) | |
| Read(C) | |
| B=B*10 | |
| Write(B) | |
| | Read(B) |
| | B=B*5 |
| C=C*5 | |
| Write(C) | |
| | Write(B) |
| | Read(A) |
| | A=A*10 |
| | Write(A) |

**Example 19:** **Consider the following transactions. Find out two non-serial schedules that are serilaizable:**

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| Read (A) | Read (C) | Read (B) |
| A = A + 500 | Read (B) | Read (C) |
| Write (A) | B = B + C | B = B + 450 |
| Read (B) | Write (B) | Write (B) |
| B = B – 500 | Read (A) | C = C + B |
| Write (B) | A = N – C | Write (B) |
| | Write (A) | |

**Ans. :** In $T_3$, the second Write(B)is ambiguous. As there is C=C+B, we can assume that it is Write(C).

(i)

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| | Read(C) | |
| | Read(B) | |
| | B=B+C | |
| | Write(B) | |
| | | Read(B) |
| | | Read(C) |
| | | B=B+450 |
| | | Write(B) |
| | Read(A) | |
| | A=N-C | |
| | Write(A) | |
| Read(A) | | |
| A=A+500 | | |
| Write(A) | | |
| | | C=C+B |
| | | Write(C) |
| Read(B) | | |
| B=B-500 | | |
| Write(B) | | |

(ii)

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| | Read(C) | |
| | Read(B) | |
| | B=B+C | |
| | Write(B) | |
| | | Read(B) |
| | Read(A) | |
| | A=N-C | |
| | Write(A) | |
| | | Read(C) |
| | | B=B+450 |
| | | Write(B) |
| Read(A) | | |
| A=A+500 | | |
| Write(A) | | |
| | | C=C+B |
| | | Write(C) |
| Read(B) | | |
| B=B-500 | | |
| Write(B) | | |

**Example 20: Consider the following transaction. Find out two non-serial schedules that are serializable :**

| $T_1$ | $T_2$ |
|---|---|
| Read (x) | Read (x) |
| x=x+1000 | x=x–1000 |
| Write (x) | Write (x) |
| Read (y) | Read (y) |
| Read (z) | y=y–2000 |
| y=y+2000 | Write (y) |
| Write (y) | |
| z=z+3000 | |
| Write (z) | |

**Ans. :**

(i)

| $T_1$ | $T_2$ |
|---|---|
| Read(x) | |
| x=x+1000 | |
| Write(x) | |
| | Read(x) |
| | x=x-1000 |
| | Write(x) |
| Read(y) | |
| Read(z) | |
| y=y+2000 | |
| Write(y) | |
| | Read(y) |
| | y=y-2000 |
| | Write(y) |
| z=z+3000 | |
| Write(z) | |

(ii)

| $T_1$ | $T_2$ |
|---|---|
| Read(x) | |
| x=x+1000 | |
| Write(x) | |
| | Read(x) |
| Read(y) | |
| Read(z) | |
| | x=x-1000 |
| | Write(x) |
| y=y+2000 | |
| Write(y) | |
| | Read(y) |
| | y=y-2000 |
| | Write(y) |
| z=z+3000 | |
| Write(z) | |

**Example 21: Give two non-serial schedule that are serializable. Consider the following transactions:**

| $T_1$ | $T_2$ |
|---|---|
| Read (Q) | Read (R) |
| Q = Q + 100 | R = R + 200 |
| Write (Q) | Write (R) |
| Read (R) | Read (P) |
| R = R + 250 | |
| Write (R) | |

**Ans.:**

(i)

| $T_1$ | $T_2$ |
|---|---|
| Read(Q) | |
| Q=Q+100 | |
| Write(Q) | |
| | Read(R) |
| | R=R+200 |
| | Write(R) |
| Read(R) | |
| R=R+250 | |
| Write(R) | |
| | Write(P) |

(ii)

| $T_1$ | $T_2$ |
|---|---|
| | Read(R) |
| | R=R+200 |
| | Write(R) |
| Read(Q) | |
| Q=Q+100 | |
| Write(Q) | |
| | Write(P) |
| Read(R) | |
| R=R+250 | |
| Write(R) | |

## Summary

- A transaction is a single logical unit of work.
- To ensure the integrity of data, database system maintains ACID properties (A: Atomicity, C: Consistency, I: Isolation, D: Durability).
- The possible states of a transaction are : Active, Partially Committed, Failed, Committed, Aborted.
- Schedule is the chronological order in which instructions are executed in the system.
- Serial Schedule is a schedule in which transactions are executed completely one after another.
- Concurrent Schedule is a schedule in which the operations of multiple transactions are interleaved.
- A serializable schedule is a concurrent schedule which executes on any consistent database and produces the result identical to that of some serial schedule.
- There are two forms of serializability. i.e. Conflict Serializability, View Serializability.
- Precedence graph is used to check whether a schedule is conflict serializable or not.
- If a schedule is view equivalent to its serial schedule, then the given schedule is said to be View Serializable.

- In the following example S1 is view serializable. It is because of the following three reasons:
- In S1 transaction T1 first reads the value of A. In S2 also T1 first reads the value of A.
  In S1 transaction T1 first reads the value of B. In S2 also T1 first reads the value of B.

i. In S1, final write operation on A is done by T2. In S2 also T2 performs final write on A.

ii. In S1, the final write on B is done by T1. In S2 also the final write on B is done by T1.

iii. In S1, T2 reads the value of A modified by T1. In S2 also T2 reads the value of A modified by T1.

**S1**

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| | W(A) |
| R(B) | |
| W(B) | |

**S2**

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| R(B) | |
| W(B) | |
| | R(A) |
| | W(A) |

## Check Your Understanding

**I. Multiple Choice Questions:**

1. Which of the following is not a transaction property?
   (a) Integrity (b) Isolation
   (c) Atomicity (d) Consistency
2. Once a transaction completes successfully, the changes it has made to the database persist, even if there are system failures. This is known as .....
   (a) atomicity (b) consistency
   (c) isolation (d) durability
3. When a transaction executes its final operation, it is said to be in a ..... state.
   (a) committed (b) partially committed
   (c) active (d) aborted
4. A transaction is in a ..... state, if it executes all its operations successfully.
   (a) committed (b) partially committed
   (c) active (d) aborted
5. Either all instructions are executed or none of the instruction is executed. This is called as .....
   (a) isolation (b) atomicity
   (c) consistency (d) durability

6. A transaction that has not completed successfully is called as ..... transaction.
   (a) compensating (b) aborted
   (c) active (d) none of these
7. The sequence of execution in concurrency control is known as .....
   (a) sequence (b) order
   (c) schedule (d) none of these
8. Which one of the following is NOT a part of the ACID properties of transaction?
   (a) Consistency (b) Atomicity
   (c) Isolation (d) none of these
9. A serializable schedule gives same result as some ..... schedule.
   (a) concurrent (b) serial
   (c) another (d) none of these

**Ans.:** (1) a (2) d (3) b (4) a (5) b (6) b (7) c (8) d (9) b.

**I. State TRUE/FALSE.**

1. Every conflict serializable schedule is view serializable, but there are view serializable schedules that are not conflict serializable.
2. If the database was in consistent state before a transaction, then after execution of transaction, the database will not be in a consistent state.
3. Each transaction is unaware of other transactions executing concurrently in the system.
4. Transaction is active when it is executing.
5. When several transactions are executed concurrently, the corresponding schedule is called concurrent schedule.
6. A concurrent schedule producing results that are equivalent to a serial schedule is known as serializable schedule.
7. When the transaction completes its execution successfully, it enters committed state from partially committed state.
8. For a set of n transactions, there exist n! different valid serial schedules.
9. If the precedence graph for a schedule is cyclic, then it is not conflict serializable.

**Answer:**

(1) TRUE (2) FALSE (3) TRUE (4) TRUE
(5) TRUE (6) TRUE (7) TRUE (8) TRUE
(9) TRUE

## Practice Questions

**Q.1. Answer the following questions in brief.**

1. What is a transaction?
2. Differentiate between serial and serializable schedule.

3. State the different states of a transaction.
4. Why there is a need for concurrent execution of transactions?
5. What is a failed state of a transaction?
6. Differentiate between committed and aborted state of a transaction.
7. Differentiate between serial and concurrent schedule.
8. State the use of precedence graph.
9. What is an isolation property of a transaction?

**Q.2 Answer the following questions.**

1. Describe the ACID properties of transaction.
2. Describe the different transaction states with diagram.
3. Explain the significance of ACID properties with suitable example.
4. What is schedule? Explain serial and concurrent schedules.
5. What is view serializability? Explain in detail.
6. What is conflict serializability? Explain in detail.
7. Explain how to use precedence graph for testing conflict serializability with suitable example.
8. What is concurrent execution? Explain in detail.
9. Explain the concept of serializability in detail.

**Q.3 Define the terms:**

(a) Atomicity property
(b) Durability property
(c) Serial schedule
(d) Consistency property
(e) Conflict Serializability
(f) Serializable schedule
(g) View Serializability
(h) Precedence graph
(i) Aborted state of a transaction

## Previous Exams Questions

**Summer 2017**

1. What is transaction? List properties of transaction. [2 M]

**Ans.** Please refer Section 3.2 and 3.3.

2. Define cascadeless schedule. [2 M]

**Ans.** Please refer Section 3.7.2.

3. Explain states of transaction with the help of suitable diagram. [4 M]

**Ans.** Please refer Section 3.4.

4. What is searializability? Explain view serializability with example. [4 M]

**Ans.** Please refer Section 3.6 and 3.6.2.

## Winter 2017

1. List the state of Transaction. [2 M]
Ans. Please refer Section 3.4.
2. What is transaction ? Explain properties of transaction. [4 M]
Ans. Please refer Section 3.2 and 3.3.
3. What are the problems in concurrent execution of transaction ? [4 M]
Ans. Please refer Section 3.5.
4. What is searializability? Explain conflict serializability with example. [4 M]
Ans. Please refer Section 3.6.

## Summer 2018

1. What is Schedule ? List the types of Schedule. [2 M]
Ans. Please refer Section 3.4.1 and 3.7.
2. What is transaction ? Explain the states of transaction with the help of diagram. [4 M]
Ans. Please refer Sections 3.2 and 3.4.

## Winter 2018

1. Define transaction. [2 M]
Ans. Please refer Section 3.2.
2. What is schedule? List the types of schedule. [2 M]
Ans. Please refer Section 3.4.1.
3. Explain the properties of transaction. [4 M]
Ans. Please refer Section 3.3.
4. What is serializability? Explain types with example. [4 M]
Ans. Please refer Section 3.6.

## Summer 2019

1. Define transaction. [2 M]
Ans. Please refer Section 3.2.
2. List states of transaction. [2 M]
Ans. Please refer Section 3.4.
3. Explain advantages of concurrent execution. [4 M]
Ans. Please refer Section 3.5.
4. List and explain properties of transaction. [4 M]
Ans. Please refer Section 3.3.
5. Explain problems of concurrent execution of transaction. [4 M]
Ans. Please refer Section 3.5.
6. What is schedule? Explain its types. [4 M]
Ans. Please refer Section 3.4.1.

❖❖❖

# 4... Concurrency Control and Recovery System

## Objectives...

- To understand the concept of Locks and its use in Lock Based Protocol.
- To learn about Deadlock and how it can occur.
- To know different types of Failures.
- To understand the Recovery Techniques.

## 4.1 INTRODUCTION

- This section deals with some concurrency control schemes. These schemes ensure that the schedules produced by concurrent transaction are serializable.
- Following are the concurrency control schemes:
  1. Lock-based protocol.
  2. Time-stamp based protocol.
  3. Validation based protocol.
  4. Multiple granularity.
  5. Multiversion schemes.
- A computer system is subject to failure of various types. If some failure occurs during the execution of a transaction, it results in inconsistent database.
- Recovery system is an integral part of database system. It is responsible for restoration of the database to a consistent state that existed before the occurrence of the failure.

## 4.2 LOCK-BASED PROTOCOL

- Serializability can easily be ensured if access to database is done in mutually exclusive manner i.e. if one transaction is accessing a data item, no other transaction can modify that data item.
- The most common method to implement mutual exclusion is to use locks.

## 4.2.1 Lock

(W-18,S-19)

- Consider that database is made up of data-items. A lock is a variable associated with each data item. Manipulating the value of lock is called **locking**.
- The value of lock is used in locking schemes to control the concurrent access and to manipulate the associated data items.

Following are the two types or modes of locks.

1. **Shared:** If a transaction $T_i$ has obtained a shared mode lock (denoted by S) on item A, then $T_i$ can read but it cannot write A.
2. **Exclusive:** If a transaction $T_i$, has obtained an exclusive mode lock (denoted by X) on item A, then $T_i$ can both read and write A.

- Depending on the type of the operation, the transaction requests a lock in an appropriate mode on data item.
- The request is made to the concurrency-control manager, and the transaction can proceed with operation only after the concurrency-control manager grants the lock to that transaction.

### 4.2.1.1 Compatibility Function

(W-17)

Given a set of lock modes, the compatibility function can be defined as:

- Let A and B represent arbitrary lock modes. Suppose that a transaction $T_i$ requests a lock of mode A on item Q on which transaction $T_j$ currently holds a lock of mode B.
- If transaction $T_i$ can be granted a lock on Q immediately, in spite of presence of the lock of mode B, then we say that mode A is compatible with B. Such a function can be represented conveniently by a matrix. Compatibility of two modes of lock is given in compatibility matrix.

| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

### 4.2.1.2 Compatibility Function

- If comp (A, B) is true it means that A is compatible with B. Notice that shared mode (S) is compatible with shared mode (S).
- At any time, several shared-mode locks can be held simultaneously on a particular data item.
- A transaction requests a shared lock on data item Q by executing the lock_S(Q) instruction and an exclusive lock is requested through lock_X(Q) instruction. A data item Q can be unlocked via the unlock (Q) instruction.
- Transaction $T_i$ can unlock the data item Q by executing unlock (Q) instruction. But transaction must hold a lock on a data item, as long as it accesses the data item.

- Locking protocols indicate when a transaction may lock and unlock each of the data items. Each transaction must follow the set of rules specified by locking protocols.
- Locking protocols restrict the number of possible serializable schedules. This section deals with only those locking protocols which allow conflict serializable schedules.

### 4.2.1.3 Starvation of Locks

- Suppose that transaction:
  $T_2$ - has a shared mode lock on data item, and
  $T_1$ - requests an exclusive mode lock on same data item.
- Clearly $T_1$ has to wait for $T_2$ to release the shared mode lock.
- Meanwhile, suppose that,
  $T_3$ - request a shared mode lock on same data item.
- This lock request is compatible with the lock granted to $T_2$, so $T_3$ may be granted the shared mode lock.
- At this point, $T_2$ may release the lock but still $T_1$ has to wait for $T_3$. But again there may be a new transaction $T_4$ requesting a shared mode lock on the same data-item.
- It is possible that there is a sequence of transactions that each requests a shared mode lock on same data item and $T_1$ never gets the exclusive mode lock on the data item. The transaction $T_1$ may never make progress and is said to be starved.
- Starvation of transactions can be avoided by granting locks as follows. When a transaction $T_i$ requests a lock on a data item Q in a particular mode M, the lock is granted provided that:
  1. There is no other transaction holding a lock on Q in a mode that conflicts with M.
  2. There is no other transaction that is waiting for a lock on Q and that made its lock request before $T_i$.

### 4.2.2 Granting of Lock

- When a request is queued, no other request which would conflict with the request in queue is granted. For example, assume program A has locked a data entry in data set X, and program B wants to lock data set X and data set Y. Program B's request is queued. When program C requests to lock a data entry in data set Y, that request is queued because program B is waiting to lock data set Y.
- When using DBLOCK wait modes, the programmer should be careful to avoid possible deadlock conditions. If a program makes multiple resource requests using the commands DBLOCK, LOCK# or REQUEST, a potential deadlock can occur if another program is also making requests for the same resources. For example, suppose program A holds resource 1 and is queued waiting for resource 2 which is held by program B. If program B makes a wait request for resource 1, a deadlock situation occurs. Programs requesting the same resources can avoid deadlock by making their resource requests in the same order.

- The following table summarizes the conditions for granting a lock:

| Lock Mode Requested | Locking Mode Held | | | | |
|---|---|---|---|---|---|
| | None | Access | Read | Write | Exclusive |
| Access or Checksum | Lock Granted | Lock Granted | Lock Granted | Lock Granted | Request Queued* |
| Read | Lock Granted | Lock Granted | Lock Granted | Request Queued* | Request Queued* |
| Write | Lock Granted | Lock Granted | Request Queued* | Request Queued* | Request Queued* |
| Exclusive | Lock Granted | Request Queued* | Request Queued* | Request Queued* | Request Queued* |

## 4.3 Two-phase Locking Protocol (S-17;18)

- This protocol requires that each transaction issue a lock and unlock requests in two phases:
  - (1) **Growing phase:** A transaction may obtain locks, but may not release any lock.
  - (2) **Shrinking phase:** A transaction may release locks, but may not obtain any new locks. Initially the transaction is in growing phase. In this phase it acquires locks as needed. Once, the transaction releases a lock, it enters the shrinking phase and it can issue no more lock requests.
- The point in the schedule where the transaction has obtained its final lock (the end of its growing phase) is called the lock point of the transaction. The transactions can be ordered according to lock points.
- This ordering gives the serializability ordering for transaction. This serial schedule is conflict equivalent i.e. the two-phase locking protocol ensures conflict serializability.
- **Example:** Following two transactions are transactions.

$T_3$: lock_X (B);

read (B);
B: = B – 50;
write (B);
lock_X(A);
read (A);
A: = A + 50;
write (A);
Unlock (B);
Unlock (A).

$T_4$ : lock_S (A);
read (A);
lock_S (B);
read (B);
display (A + B);
Unlock (A);
Unlock (B).

- The unlock instructions do not need to appear at the end of the transaction.
- Two-phase locking does not ensure freedom from deadlock. Transactions $T_3$ and $T_4$ are two phase, but they are deadlocked in the schedule.

**Table 4.1**

| $T_3$ | $T_4$ |
|---|---|
| Lock_X (B) | |
| read (B) | |
| B:= B – 50 | |
| write (B) | |
| | lock_ S (A) |
| | read (A) |
| | lock_S (B) |
| Lock_X (A) | |
| read (A) | |
| A:= A + 50 | |
| write (A) | |
| unlock (B) | |
| unlock (A) | |
| | read (B); |
| | display (A + B); |
| | unlock (A); |
| | unlock (B) |

Cascading rollback may occur under two-phase locking.

## 4.2.3.1 Variations on Two-Phase Locking

(S-18, W-18)

**1. Strict Two-Phase Locking Protocol: (Strict 2PL)**

Cascading rollbacks can be avoided by a modification of two-phase locking called strict two-phase locking protocol.

It requires that in addition to locking being two-phase, all exclusive-mode locks taken by a transaction must be held until that transaction commits.

This requirement ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits. This prevents any other transaction from reading the data.

2. **Rigorous Two-Phase Locking Protocol: (Rigorous 2PL)**

It requires that all locks to be held until the transaction commits.

With rigorous two-phase locking, transactions can be serialized in the order in which they commit. Most database system implements strict or rigorous two-phase locking.

3. **Two-Phase Locking with Lock Conversion:**

Basic two-phaselocking is modified and lock conversions are allowed. We shall provide a mechanism for upgrading a shared lock to an exclusive lock and downgrading an exclusive lock to shared lock.

**Upgrade (Q):** Convert shared lock lock_S (Q) to exclusive lock lock_X (Q).

**Downgrade (Q):** Convert exclusive lock lock_X (Q) to shared lock lock_S (Q).

- Lock conversion can not be allowed to occur arbitrarily. Upgrading can take place in growing phase and downgrading can take place in only the shrinking phase.
- Two-phase locking with lock conversion generates conflict serializable schedule.

**Example:** Consider the following two transactions with only read and write operations:

$T_8$: read ($a_1$);
read ($a_2$);
read ($a_3$);
read ($a_4$);
read ($a_5$);
write ($a_1$)

$T_9$: read ($a_1$);
read ($a_2$);
display ($a_1 + a_2$);

If we employ locking with lock conversion, the schedule can be given as:

- Automatic generation of appropriate lock and unlock instructions for a transaction is also possible.
- When a transaction issues a read (a) operation, the system issues lock_S(a) instruction followed by read (a) operation.
- When a transaction $T_i$ issues a write (Q) operation, the system checks to see whether $T_i$ already holds a shared lock on Q. If it does, then the system issues an upgrade (Q) instruction, followed by the write (Q) instruction.
- Otherwise the system issues lock_X(Q) instruction followed by write (Q) operation.

Table 4.2

| $T_8$ | $T_9$ |
|---|---|
| lock_S ($a_1$) | |
| read ($a_1$) | |
| | lock_S ($a_1$) |
| | read ($a_1$) |
| lock_S ($a_2$) | |
| read ($a_2$) | |
| | lock_S ($a_2$) |
| | read ($a_2$) |
| lock_S ($a_3$) | |
| lock_S ($a_4$) | |
| | display ($a_1 + a_2$) |
| | unlock ($a_1$) |
| | unlock ($a_2$) |
| lock_S ($a_5$) | |
| read ($a_5$) | |
| upgrade ($a_1$) | |
| write ($a_1$) | |

## 4.2.3.2 Graph - Based Protocols

- Graph-based protocol is not a two-phase locking protocol and it requires prior knowledge of how each transaction will access the database.
- To acquire such prior knowledge it uses data graph.
  Let D = $\{d_1, d_2, ..., d_n\}$ is the set of all data items. If $d_i \rightarrow d_j$, then any transaction accessing both $d_i$ and $d_j$ must access $d_i$ before accessing $d_j$. This partial ordering may be a result of either the logical or physical organization of data or it may be imposed for the purpose of concurrency control.
- The partial ordering implies that the set D may be viewed as a directed acyclic graph, called **database graph**. For simplicity we shall consider only those graphs which are rooted trees.
- A simple protocol called **tree-protocol**, which is restricted to exclusive locks is stated here.
- Each transaction $T_i$ can lock (lock_X(Q)) a data item at most once and observe the following rules:
  1. The first lock by $T_i$ may be on any data item.
  2. Subsequently, a data item Q can be locked by $T_i$ only if the parent of Q is currently locked by $T_i$.
  3. Data item may be unlocked at any time.

4. A data item that has been locked and unlocked by $T_i$ cannot subsequently be relocked by $T_i$.

All schedules that are legal under the tree protocol are conflict serializable.

**Example:** Consider the database graph in Fig. 4.1. The following four transactions follow the tree protocol on this graph.

$T_{10}$: lock_X (B);
lock_X (E);
lock_X (D);
unlock (B);
unlock (E);
lock_X (G);
unlock (D);
unlock (G).

$T_{11}$: lock_X (D);
lock_X (H);
unlock (D);
unlock (H).

$T_{12}$: lock_X (B);
lock_X (E);
unlock (E);
unlock (B).

$T_{13}$: lock_X (D);
lock_X (H);
unlock (D);
unlock (H).

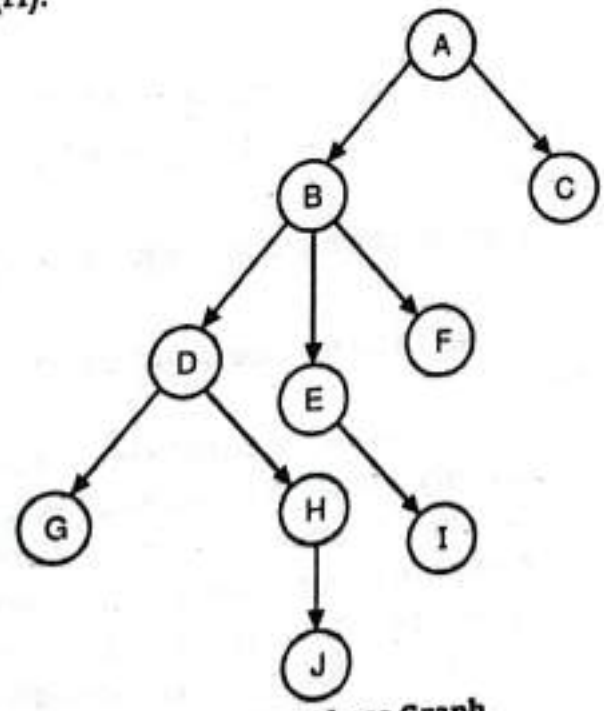



**Fig. 4.1: Database Graph**

- The schedule following tree-protocol is given below:

Table 4.3

| $T_{10}$ | $T_{11}$ | $T_{12}$ | $T_{13}$ |
|---|---|---|---|
| lock_X (B) | | | |
| | lock_X (D) | | |
| | lock_X (H) | | |
| | unlock (D) | | |
| lock_X (E) | | | |
| lock_X (D) | | | |
| unlock (B) | | | |
| unlock (E) | | | |
| | | lock_X (B) | |
| | | lock_X (E) | |
| | unlock (H) | | |
| lock_X (G) | | | |
| unlock (D) | | | |
| | | | lock_X (D) |
| | | | lock_X (H) |
| | | | unlock (D) |
| | | | unlock (H) |
| | | unlock (E) | |
| | | unlock (B) | |
| unlock (G) | | | |

- This schedule is conflict serializable. Tree-protocol ensures conflict serializability and it is free from deadlocks.

**Advantages:**

- Tree locking protocol has an advantage over Two-phase locking protocol:
  1. Unlocking may occur earlier. Earlier unlocking may lead to shorter waiting times and an increase in concurrency.
  2. It is deadlock free and hence no rollbacks are required.

**Disadvantages:**

- But it has one disadvantage that in some cases, a transaction may have to lock data items that it does not access.
- **Example:** A transaction that needs to access data items A and J in the database graph given in Fig. 4.1 must lock not only A and J but also data items B, D and H.
- These additional locking results in increased locking overhead, the possibility of additional waiting time and potential decrease in concurrency.
- Further, without prior knowledge of what data item will need to be locked, transactions will have to lock the root of tree and that can reduce the concurrency greatly.

## 4.3 TIMESTAMP BASED PROTOCOL (S- 19)

- This protocol decides the ordering of transactions in advance to determine the serializability order. For this it uses time-stamp ordering scheme.

### 4.3.1 Timestamp

- With each transaction $T_i$ in the system, a fixed value called timestamp is associated, denoted by TS ($T_i$). This timestamp is assigned by database system before $T_i$ starts execution.
- If transaction $T_i$ is assigned a timestamp TS ($T_i$) and a new transaction $T_j$ enters the system, then TS ($T_i$) < $T_s$ ($T_j$). There are two simple methods for implementing timestamp scheme.
  1. Use the value of system clock as the timestamp i.e. a transaction's timestamp is equal to the value of system clock, when the transaction enters the system.
  2. Use a logical counter that is incremented after a new timestamp has been assigned. i.e. a transaction's timestamp is equal to the value of the counter when the transaction enters the system.
- Timestamp of transaction determines the serializability order.
- If TS ($T_i$) < TS ($T_j$) then the system must ensure that the resultant schedule is equivalent to serial schedule in which $T_i$ appears before $T_j$.
- To implement this scheme, two timestamp values are associated with each data item Q.
- **W_Timestamp (Q):** It denotes the largest timestamp of any transaction that executed write (Q) successfully.
- **R_Timestamp (Q):** It denotes the largest timestamp of any transaction that executed read (Q) successfully.
- Whenever, a new read(Q) or write(Q) instruction is executed, these timestamps are updated.

### 4.3.2 Timestamp Ordering Protocol

- Timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.
- This protocol works as follows:
  1. **Suppose that transaction $T_i$ issues read (Q).**
     (a) If TS ($T_i$) < W_Timestamp (Q), then $T_i$ needs to read a value of Q that was already overwritten. Hence, the read operation is cancelled, and $T_i$ is rolled back.
     (b) If TS ($T_i$) ≥ W_Timestamp (Q), then the read operation is executed, and R_Timestamp (Q) is set to maximum of R_Timestamp (Q) and TS ($T_i$).

2. **Suppose that $T_i$ issues write (Q).**
   (a) If TS ($T_i$) < R_Timestamp (Q) then the value of Q that $T_i$ is producing was needed previously and the system assumed that value would never be produced. Hence, the write operation is cancelled, and $T_i$ is rolled back.
   (b) If TS ($T_i$) < W_Timestamp (Q), then $T_i$ is attempting to write an outdated value of Q. Hence, this write operation is cancelled, $T_i$ is rolled back.
   (c) Otherwise the write operation is executed and W_Timestamp is set to TS ($T_i$).

- A transaction $T_i$, that is rolled back by the concurrency control scheme as a result of either a read or write operation being issued, is assigned a new timestamp and is restarted.
- **Example:** Consider the given transactions $T_{14}$ and $T_{15}$.

$T_{14}$ : read (B);
read (A);
display (A + B)

$T_{15}$ : read (B);
B:= B – 50;
write (B);
read (A);
A:= A + 50;
write (A);
display (A + B).

- Concurrent schedule for the two transactions is given below.
- Here, we consider that TS ($T_{14}$) < TS ($T_{15}$). The schedule given in Table 4.4 is possible under the timestamp protocol.

Table **4.4**

| $T_{14}$ | $T_{15}$ |
|---|---|
| read (B) | |
| | read (B) |
| | B: = B – 50 |
| | write (B) |
| read (A) | |
| | read (A) |
| display (A + B) | |
| | A: = A + 50 |
| | write (A) |
| | display (A + B) |

1. read (B) of $T_{14}$ is executed because TS ($T_{14}$) ≥ W_TIMESTAMP (B) and it sets R_Timestamp (B) = TS ($T_{14}$).
2. read (B) of $T_{15}$ is executed because TS ($T_{15}$) ≥ W_TIMESTAMP (B) and it sets R_Timestamp (B) = TS ($T_{15}$).
3. write (B) of $T_{15}$ is executed because TS ($T_{15}$) = R_Timestamp (B).

Similarly, read (A) of $T_{14}$ and write (A) of $T_{15}$ are executed.

## 4.3.3 Thomas's Write Rule

(S-17)

Consider the schedule given in Table 4.5.

**Table 4.5**

| $T_{16}$ | $T_{17}$ |
|---|---|
| read (Q) | |
| | write (Q) |
| write (Q) | |

- Apply Timestamp-ordering protocol to given schedule.
- Since, $T_{16}$ starts before $T_{17}$, TS ($T_{16}$) < TS ($T_{17}$), read (Q) of $T_{16}$ and write (Q) operations succeed. When $T_{16}$ attempts its write (Q) operation, we observe that,
- TS ($T_{16}$) < W_Timestamp (Q), since W_timestamp (Q) = TS ($_{17}$) According to Timestamp protocol, write (Q) must be rejected, $T_{16}$ will be rolled back.
- Timestamp ordering protocol rolls back the transaction $T_{16}$, but the value of write (Q) Operation of $T_{16}$ is already written by write (Q) of $T_{17}$, and the value that write (Q) of $T_{16}$ is attempting to write will never be read. i.e. we can ignore the write (Q) of $T_{16}$.
- This leads to a modification of Timestamp-ordering protocol. This modified protocol operates as follows:

1. **Suppose that transaction $T_i$ issues read (Q):**
   (a) If TS ($T_i$) < W_Timestamp (Q), then $T_i$ needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and $T_i$ is rolled back.
   (b) If TS ($T_i$) ≥ W_Timestamp (Q), then read operation is executed, and R_Timestamp (Q) is set to the maximum of R_Timestamp (Q) and TS ($T_i$).
2. **Suppose that transaction $T_i$ issues write (Q):**
   (a) If TS ($T_i$) < R_Timestamp (Q) then the value of Q that $T_i$ is producing was previously needed, and it was assumed that the value would never be produced. Hence, the write operation is cancelled, and $T_i$ is rolled back.
   (b) If TS ($T_i$) < W_Timestamp (Q), then $T_i$ is attempting write an outdated value of Q. Hence, the write operation can be ignored.

(c) Otherwise, the write operation is executed, and W_Timestamp (Q) is set to TS ($T_i$).

i.e. here if TS ($T_i$) < W_Timestamp (Q), then we ignore the absolute write operation.

This modification to timestamp ordering protocol is called **Thomas Write Rule.**

- Thomas write rule for schedule 8 given in table results in a serial schedule $< T_{16}, T_{17} >$ which is view equivalent to given schedule.

## 4.4 VALIDATION BASED PROTOCOL (W-17)

- We assume that each transaction $T_i$ executes in two or three phases depending on read only operation or an update operation.
- Following are the three phases of execution.
  1. **Read phase:** During this phase, the execution of transaction $T_i$ takes place. The values of various data item are read and stored in variable local to $T_i$. All write operations are performed on local variables, without updates of actual database.
  2. **Validation phase:** It performs a validation test to determine whether it can copy the results of write operations stored in temporary local variables to database without causing violation of serializability.
  3. **Write phase:** If transaction $T_i$ succeeds in validation phase, then write phase, actually updates the database, otherwise $T_i$ is rolled back. Each transaction must go through the three phases in the order shown. The three phases of concurrently executing transaction can be interleaved.
- Three different timestamps are associated with transaction $T_i$ to determine when the various phases of transaction $T_i$ take place.

**Start ($T_i$):** The time when $T_i$ started its execution.

**Validation ($T_i$):** The time when $T_i$ finished its read phase and started its validation phase.

**Finish ($T_i$):** The time when $T_i$ finished its write phase.

Timestamp of transaction $T_i$ is TS ($T_i$) = validation ($T_i$)

- If TS ($T_j$) < TS ($T_k$) then any resultant schedule must be equivalent to a serial schedule in which transaction $T_j$ appears before transaction $T_k$.

The validation test of validation phase for transaction $T_j$ requires that, for all transactions $T_i$ with TS ($T_i$) < TS ($T_j$), one of the following two conditions must hold.

1. Finish ($T_i$) < Start ($T_j$), since if $T_i$ completes its execution before $T_j$ started, then the serializability order is maintained.
2. The set of data items written by $T_i$ does not intersect with the set of data items read by $T_j$, and $T_i$ completes its write phase before $T_j$ starts its validation phase i.e. Start ($T_j$) < Finish ($T_i$) < Validation ($T_j$)

This condition ensures that writes of $T_i$ and $T_j$ do not overlap.

Since, the write of $T_i$ does not affect read of $T_j$ and since, $T_j$ cannot affect the read of $T_i$, the serializability order is maintained.

**Example:** Consider again the transactions $T_{14}$ and $T_{15}$. A schedule produced using validation is given in Table 4.6.

**Table 4.6**

| $T_{14}$ | $T_{15}$ |
|---|---|
| read (B) | |
| | read (B) |
| | B:= B – 50 |
| | read (A) |
| | A:= A + 50 |
| read (A) | |
| <Validate> | |
| display (A + B) | |
| | <Validate> |
| | write (B) |
| | write (A) |

## 4.5 DEADLOCK HANDLING (S-17,18;W-17,18)

A system is in deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set.

If $\{T_0, T_1, ..., T_n\}$ is the set of transactions such that $T_0$ is waiting for a data item that is held by $T_1$ and $T_1$ is waiting for a data item that is held by $T_2$ ... and $T_{n-1}$ is waiting for a data item that is held by $T_n$.

$T_n$ is waiting for a data item that is held by $T_0$. Hence, none of the transactions can make progress in such situation. That is the system is in deadlock state.

There are three methods for dealing with deadlock problems:

1. Deadlock prevention.
2. Time-out based schemes.
3. Deadlock detection and deadlock recovery.

### 4.5.1 Deadlock Prevention (W-16;S-17)

There are two approaches to deadlock prevention.

1. It ensures that cyclic waits can be avoided by ordering the requests for locks or requiring all locks to be acquired together.
2. It performs transaction rollbacks instead of waiting for a lock, whenever the wait could potentially result in a deadlock.

- Following are the schemes under first approach. Lock all the data items before a transaction begins its execution. But there are two main disadvantages of using this protocol.
  1. It is often hard to predict, before the transaction what data items need to be locked.
  2. Data item utilization may be very low, since many of the data items may be locked but unused for a long time.
- Another scheme is to impose a partial ordering of all data items, and transaction can lock the data items only in that order. Using tree-protocol, this scheme can be implemented.
- Following are the schemes for second approach. The second approach uses preemption and transaction rollbacks.
- When a transaction $T_2$ requests a lock that is held by a transaction $T_1$, the lock granted to $T_1$, may be preempted by rolling back of $T_1$ and granting of lock to $T_2$.
- To control the preemption, we assign a unique timestamp to each transaction. These timestamps will be used to decide whether the transaction should wait or rollback. Locking is used for concurrency control. If a transaction is rolled back, it retains the old timestamp when restarted.
- Two different deadlock prevention schemes using timestamps under the second approach are:

1. **Wait-Die:** Scheme is based on non-preemptive technique. When a transaction $T_i$ requests a lock on a data item currently held by $T_j$, $T_i$ is allowed to wait only if it has a timestamp smaller than that of $T_j$. Otherwise $T_i$ is rolled back (die).

   **Example:** $TS(T_i) = 5$
   $TS(T_j) = 10$
   $TS(T_k) = 15$

   If $T_i$ requests a lock on a data item held by $T_j$ then $T_i$ will wait since $TS(T_i) < TS(T_j)$. If $T_k$ requests a lock on a data item held by $T_j$, then $T_k$ will be rolled back since $TS(T_k) > TS(T_j)$.

2. **Wound-wait:** This scheme is based on preemptive technique. When a transaction $T_i$ requests a lock on a data item, currently held by $T_j$, $T_i$ is allowed to wait only if it has a timestamp larger than that of $T_j$. Otherwise $T_j$ is rolled back ($T_j$ is wounded by $T_i$).

   **Example:** Consider the timestamps given in previous example for $T_i$, $T_j$ and $T_k$ transactions.

   If $T_i$ requests a lock on data item held by $T_j$, then $T_j$ will be rolled back.

   $$TS(T_i) < TS(T_j)$$

   If $T_k$ requests a lock on a data item held by $T_j$, then $T_k$ will wait since,

   $$TS(T_k) > TS(T_j)$$

Comparison of wait-die and wound-wait schemes:

1. Both the wait-die and wound-wait schemes avoid starvation.
2. In wait-die scheme, an older transaction must wait for younger one to release the data item. Whereas in wound-wait scheme the older transaction never waits for a younger transaction.
3. Number of rollbacks in wound-wait scheme are fewer as compared to wait-die scheme.
4. Major problem in both the schemes is unnecessary rollbacks.

**Time-out Based Schemes:**

- In this approach deadlock handling is based on lock time-outs. A transaction that has requested a lock waits for at most a specified amount of time. If the lock has not been granted within that time, the transaction is said to time out and it rolls back itself and restarts.
- If there was a deadlock, one or more transactions involved in deadlock will time-out and rollback allowing others to complete their execution.

**Advantages and Disadvantages:**

1. It is easy to implement.
2. It works well if transactions are short and if long waits are likely to be due to deadlocks.
3. It is difficult to decide how long a transaction must wait before time-out.
   - Very long wait results in unnecessary delays once a deadlock has occurred.
   - Very short wait results in transaction rollbacks even when there is no deadlock, leading to wasted resources.
4. Starvation is the possibility with this scheme.

## 4.5.2 Deadlock Detection (W-18)

- This is one method for dealing with deadlock. It allows the system to enter a deadlock state, and then try to recover using deadlock detection and dead-lock recovery scheme.
- If the probability that system enters deadlock state is relatively low, this method is efficient.
- An algorithm that examines the state of the system is executed periodically to determine whether a deadlock has occurred.
- If it has occurred then the system must attempt to recover from the deadlock.
- Following are the requirements for deadlock detection and recovery:

1. Keep information about the current allocation of data items to transactions, as well as any outstanding data item requests.
2. Provide an algorithm that uses this information to find out whether the system has entered a deadlock state.
3. Recover from the deadlock when detection algorithm finds out that a deadlock exists.

- Deadlocks can be detected using a directed graph called **wait for graph.**
- The wait for graph consists of a pair G = (V, E) where V is a set of vertices and E is a set of edges. The set of vertices consists of all transactions in the system. Each element in the set E of edges is an ordered pair $T_i \rightarrow T_j$.
- A directed edge $T_i \rightarrow T_j$ in graph imply that $T_i$ is waiting for transaction $T_j$ to release the data item.
- The edge $T_i \rightarrow T_j$ is removed when $T_j$ is no longer holding a data item needed by transaction $T_i$.
- A deadlock exist in the system if the wait for graph for that system contain a cycle (or loop).
- Consider the wait graph given in Fig. 4.3.

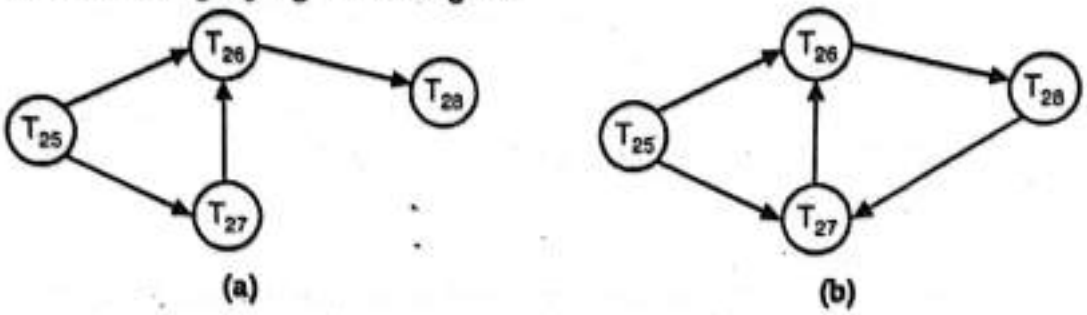



**Fig. 4.3: Wait Graph**

Transaction $T_{25}$ is waiting for $T_{26}$ and $T_{27}$.

Transaction $T_{26}$ is waiting for $T_{28}$.

Transaction $T_{27}$ is waiting for $T_{26}$.

- The graph contains no cycle. Hence, the system is not in deadlock state.

But now consider the Fig. 4.3 (b). It contains a loop,

$$T_{26} \rightarrow T_{28} \rightarrow T_{27} \rightarrow T_{26}$$

- Hence, the system is in deadlock state. In this way using this algorithm deadlocks can be detected. Depending on the frequency of deadlock occurrence, the deadlock detection algorithm should be executed.

## 4.5.3 Deadlock Recovery

(W-17)

- The most common solution to recover from deadlock is to rollback one or more transactions to break the deadlock.

A transaction can be recovered using following three actions:

1. **Selection of victim:** Determine which transaction to rollback. Those transactions that will incur the minimum cost will be rolled back. The cost of rollback can be decided by following factors.
   (i) How long the transaction has computed and how much longer the transaction will compute before it completes the assigned task ?
   (ii) How many data items it has used ?

(iii) How many more data items it needs to complete ?

(iv) How many transactions will be involved in the rollback ?

2. **Rollback:** One method to rollback a transaction is to abort transaction and restart it. The other more effective method is to rollback the transaction only as far as necessary to break the deadlock. But this require additional information about all the running transactions.

3. **Starvation:** It may happen that the same transaction is always selected as victim. This results in starvation. The most common solution is to include the number of rollbacks in the cost factor.

## 4.6 FAILURES AND ERRORS (S-19,W-18)

**Failures and Errors:**

- A system is reliable if it works as per its specifications and produces correct set of output values for a given set of input values.
- The **failure** of a system occurs when the system does not work according to its specifications and fails to deliver the service for which it was intended.
- An **error** in the system occurs when a component of a system assumes a state that is not desirable. A fault is detected either when an error is propagated from one component to another or the failure of the component is observed.
- There are two types of failures:
  1. Failure that results in loss of information.
  2. Failure that does not results in loss of information.
- A computer system is electrical device, and causes different failures.
- There are various causes of failure, including power failure, software error, disk crash, a fire in the machine room etc. In each of these failure cases, information may be lost.
- Therefore, the database system must take actions in advance to ensure the durability and atomicity properties of transaction.
- An integral part of a database is a recovery scheme i.e. responsible for the restoration of the database to a consistent state that existed before the occurrence of the failure.

### 4.6.1 Failure Classification (S-17,19)

- There are different types of failure that may occur in a system, each of failure needs to be dealt with in a different manner.
- The simplest type of failure to deal with is one that does not result in the loss of information or data in the system.
- The failures that are more difficult or critical to deal with are those that result in loss of information or data (content).

Following are different types of failures:

### 4.6.1.1 Transaction Failure (S-18)

- There are two types of errors that may lead to transaction failure. They are:
  - (i) **System Error:** The system has entered an undesirable state as a result of which a transaction cannot continue with its normal execution. The transaction, however, can be re-executed at a later time.
  - (ii) **Logical Error:** The transaction can no longer continue with its normal execution, owing the some internal condition, such as data not found, bad input or resource limit overflow exceeded.

### 4.6.1.2 System Crash

- There is a hardware bug or error in the database software or the operating system that causes the loss of the content of volatile storage and leads transaction processing to a halt.
- The content of non-volatile storage remains undamaged and is not corrupted.

### 4.6.1.3 Disk Failure

- In disk failures disk block losses its content as a result of either a head failure during a data transfer operation. Copies of the data on other disks on tertiary media, such as tapes, disks etc., are used to recover from the failure.
- To find out how the system should recover from failure or crash, we need to recognize the failure modes of those devices used for storing data.
- Next, we must consider how these failure modes or crash modes affect the contents or information of the database.

**Note:**

1. Transaction failure does not result in loss of information.
2. System crash loses the contents of volatile storage but it does not corrupt the non-volatile storage contents. (This is known as fail-stop assumption).
3. Disk failure loses the data stored on disk. Copies of data on other disks or archival backups on tertiary media such as tapes are used to recover from the failure.

## 4.7 RECOVERY AND ATOMICITY (W-18)

- Consider banking system and transaction Ti that transfers ₹ 1000 from account X to account Y, with initial values of X and Y being ₹ 5000 and ₹ 10000 respectively.
- Suppose that a system crash has occurred during the execution of transaction $T_i$, after output (YA) has taken place, but before output (YB) was executed, where YA and YB denote the buffer blocks on which X and Y are stored. Since, the memory contents were lost, user do not know the fate of the transaction; thus, we could use one of two possible recovery procedures.

- **Re-execute $T_i$:** This procedure will result in the value of X becoming ₹ 3000, rather than ₹ 4000. Thus, the system enters an inconsistent state.
- **Do not re-execute $T_i$:** The current system state is value of ₹ 4000 and ₹ 10000 for X and Y respectively. Thus, the system enters an inconsistent state.
- To achieve our goal of atomicity, we must first output information describing the updations of stable storage, without modifying the database itself. As we shall see, this procedure will allow us to output all the updations made by a committed transaction, despite failures.

**Recovery Algorithms:**

- The algorithms which ensure database consistency and transaction atomicity despite failures are known as recovery algorithms and they have two parts:
  1. Actions taken during normal transaction processing to make sure that enough information exists to allow recovery from failures.
  2. Actions taken following a failure to recover the database contents to a state that ensures database consistency, transaction atomicity and durability.

**Recoverable Schedule:**

- A recoverable schedule is defined as a schedule where for each pair of transactions $T_i$ and $T_j$ such that $T_j$ reads a data item previously written by $T_i$, the commit operation of $T_i$ appears before the commit operation of $T_j$.
- If some failure occurs during the execution of a transaction and database is in some inconsistent state following are two simple recovery procedures.
  1. **Re-execute:** Execute the same transaction. But because of previous incomplete execution of that transaction, the database is already in an inconsistent state. Hence it results in an inconsistent database.
  2. **Do not re-execute:** If we do not re-execute the transaction, the database will remain in same inconsistent state. The atomicity property (perform either all or no database modifications) is not achieved. To achieve the goal of atomicity, information about modifications must be stored to stable storage before modifying the database. In this procedure, if some failures occur and modification information is complete then system can reexecute the transaction. Otherwise system will not reexecute the transaction.
- Following are the two schemes to achieve the recovery from transaction failures:
  1. Log-based recovery.
  2. Shadow paging.

## 4.7.1 Log-based Recovery (S-18;W-18)

- It assumes that transactions are executed serially, i.e. only one transaction is active at a time. It uses a structure called log to store the database modifications.

- There are two techniques for using log to achieve the recovery and ensure atomicity in case of failures.
  1. Deferred database modification
  2. Immediate database modification.
- Deferred modification technique ensures transaction atomicity by recording all database modifications in the log, but deferring (delaying) the execution of all write operations of transaction until the transaction partially commits.
- The immediate update technique allows database modifications to be output to the database while the transaction is still in the active state. Data modifications written by active transactions are called uncommitted modifications. If a failure occurs during execution, the system must use the old value field of log records.

**Log:**

- Log is a structure used to store the database modifications.
- It is a sequence of log records and maintains a record of all the update activities in the database.
- There are several types of log records to record significant events during transaction processing.
  1. Start of transaction:
     denoted as; $<T_i$ start$>$
  2. Update log:
     It describes a single database write and it is denoted as:
     $<T_i, X_j, V_1, V_2>$
     where, $T_i$ - Transaction identifier
     $X_j$ - Data item identifier
     $V_1$ - Old value of $X_j$
     $V_2$ - New value of $X_j$ after the write operation.
  3. Transaction commits:
     denoted as: $<T_i$ commit$>$
  4. Transaction abort:
     denoted as: $<T_i$ abort$>$

  Log is stored in stable storage.

## 4.7.2 Deferred Database Modification (W-17)

- Deferred database modification technique stores the database modifications in the log.
- Execution of write operation is done when transaction is in partially committed state.
- According to transaction state diagram, a transaction is in partially committed state when transaction completes executing the last instruction.

Execution of transaction $T_i$ proceed as follows:

- Before $T_i$ starts its execution record, a record $<T_i$ start$>$ is written to log. Write (X) operation of $T_i$ results in writing a new record $<T_i, X, V_2>$ to the log.
  [**Note:** It doesn't write $V_1$ - old value of X.]
- When $T_i$ partially commits a record $<T_i$ commit$>$ is written to log. When $T_i$ partially commits, the records associated with it in the log are used in executing the deferred writes. If system crashes before the transaction completes its execution or if the transaction aborts, then the information on the log is ignored.
- If some failure occurs while updating the database using log records, the log record is written in some stable storage, hence, the transaction can resume its database modification. Using log the system can handle any failure that results in loss of information on volatile storage. The recovery scheme uses the following recovery procedure.
- **redo ($T_i$):** It sets the value of all data items updated by transaction $T_i$ to the new values. The set of data items and their respective new values can be found in the log. The redo operation must be idempotent i.e. executing it several times must be equivalent to executing it once.
- A transaction can execute redo ($T_i$) if the log contains both the record $<T_i$ start$>$ and the record $<T_i$ commit$>$.
- **Example:** Consider the transaction $T_i$, it transfers ₹ 50 from account A to account B. Original values of account A and B are ₹ 1000 and ₹ 2000 respectively. This transaction is defined as:

```
T0 : read (A);
     A:= A – 50;
     write (A);
     read (B);
     B:= B + 50;
     write (B).
```

- Consider the second transaction $T_1$ that withdraws 100 rupees from account C. This transaction is defined as: (Assume that original values of account C is 700)

```
T1 : read (C);
     C:= C – 100;
     write (C).
```

- These two transactions are executed serially $<T_0, T_1>$.
- The log containing relevant information on these two transactions is given below:

$<T_0$, start$>$
$<T_0, A, 950>$
$<T_0, B, 2050>$
$<T_0$, commit$>$

<$T_1$, start>
<$T_1$, C, 600>
<$T_1$ , commit>

- It shows the log that result from the complete execution of T0 and T1.
- The actual output can take place to database system in various orders. One such order is given below:

| log | Database |
|---|---|
| <$T_0$, start> | |
| <$T_0$, A, 950> | |
| <$T_0$, B, 2050> | |
| <$T_0$ commit> | |
| | A = 900 |
| | B = 2050 |
| <$T_1$ start> | |
| <$T_1$, C, 600> | |
| <$T_1$ commit> | C = 600 |

- If system crashes before the completion of transactions:

**Case 1:** Crash occurs just after the log record for write (B) operation.

- Log contents after the crash are:

<$T_0$, start>
<$T_0$, A, 950>
<$T_0$, B, 2050>

- <$T_0$ commit> is not written; hence no redo operation is possible.
- The values of account remain unchanged i.e. account balance of A is ₹ 1000 and B is ₹ 2000.

**Case 2:** Crash occurs just after log record for write (C) operation.

- The log contents after the crash are:

<$T_0$ ,start>
<$T_0$, A, 950>
<$T_0$, B, 2050>
<$T_0$ ,commit>
<$T_1$, start>
<$T_1$, C, 600>

- When the system comes back it finds <$T_0$ start> and <$T_0$ commit>. But there is no <$T_1$ commit> for <$T_0$ start>. Hence, system can execute redo ($T_0$) but not redo <$T_1$>. Hence, the value of account C remains unchanged.

Case 3: Crash occurs just after the log record $<T_1$ commit$>$.

- The log contents after the crash are:

  $<T_0$ start$>$

  $<T_0$, A, 950$>$

  $<T_0$, B, 2050$>$

  $<T_0$, commit$>$

  $<T_1$ start$>$

  $<T_1$, C, 600$>$

  $<T_1$, commit$>$

- When system comes back it can execute both redo ($T_0$) and redo ($T_1$) operations.
- For each commit record, the redo ($T_i$) operation is performed. redo ($T_i$) writes the values to the database independent of the values currently in the database. Hence, the redo ($T_i$) is idempotent.

## 4.7.3 Immediate Database Modification (W-17; S-18)

- It allows the database modifications to be output to the database while transaction is still in active state.
- Database modifications written by active transactions are called uncommitted modifications.
- Execution of transaction proceeds as follows:
  1. Before $T_i$ starts its execution, the record $<T_i$ start$>$ is written to the log.
  2. Before executing any write(X) i.e. before modifying the database for write operation, it writes an update record $<T_i, X, V_1, V_2>$ to the log.
- When $T_i$ partially commits, the record $<T_i$ commit$>$ is written to log.
- As an illustration consider the same example of bank accounts and transactions $T_0$ and $T_1$. Transactions $T_0$ and $T_1$ are executed serially.
- The log corresponding to this execution is given below:

  $<T_0$ start$>$

  $<T_0$, A, 1000, 950$>$

  $<T_0$, B, 2000, 2050$>$

  $<T_0$ ,commit$>$

  $<T_1$ start$>$

  $<T_1$, C, 700, 600$>$

  $<T_1$ commit$>$

- The order in which output took place to both database system and log as a result of execution of $T_0$ and $T_1$ is:

| Data | Database |
|---|---|
| $<T_0$, start> | |
| $<T_0$, A, 1000, 950> | |
| $<T_0$, B, 2000, 2050> | |
| $<T_0$, commit> ⟶ | A = 950<br>B = 2050 |
| $<T_1$, start> | |
| $<T_1$, C, 700, 600> ⟶ | C = 600 |
| $<T_1$, commit > | |

- Using the log, the system can handle any failure. Two recovery procedures are there to recover:
  1. **Undo ($T_i$):** It restores the value of all data items updated by transaction $T_i$ to the old values.
  2. **Redo ($T_i$):** It sets the value of all data items updated by transaction $T_i$ to the new values.
- The undo ($T_i$) and redo ($T_i$) operations must be idempotent.
- Depending on the log, the recovery scheme determines which transaction need to be redone and which need to be undone.
- If the log record contains the record $<T_i$ , start> but does not contain the record $<T_i$ ,commit> then $T_i$ needs to be undone.
- If the log record contains both the records $<T_i$ start> and $<T_i$ ,commit> then $T_i$ needs to be redone i.e. If a failure occurs before $<T_i$ commit> the database modifications are rolled back by undo ($T_i$) operation. If a failure occurs after $<T_i$ commit> then the transaction is re-executed by redo ($T_i$) operation.
- Consider the following conditions of failure for transactions $T_0$ and $T_1$.

  **Case 1:** Failure occurs just after the log record for write (B) operation.

  Log contents and database are:

| Log | Database |
|---|---|
| $<T_0$ , start> | |
| $<T_0$ , A, 1000, 950> | |
| $<T_0$, B, 2000, 2050> | |
| | A = 950 |
| | B = 2050 |

- Log contains $<T_0$, start> but does not contain $<T_0$, commit>. Hence $T_0$ must be undone hence undo ($T_0$) is executed; the values of account A and B are restored to 1000 and 2000 respectively.

**Case 2:** If some failure occurs just after log record for write (C) $T_i$ has written to log.

- The log and database contents are:

| Log | Database |
|---|---|
| $<T_0$ start> | |
| $<T_0$ A, 1000, 950> | |
| $<T_0$ B, 2000, 2050> | |
| database ⟶ | A = 950 |
| B =2050 | |
| $<T_0$ commit> | |
| $<T_1$ start> | |
| $<T_1$ C, 700, 600> C= 600 | |
| database ⟶ | A = 950 |

- Log contains $<T_0$, start> and $<T_0$, commit>. Hence, the redo ($T_0$) is executed and values of accounts A and B are restored to the same (or new values) 950 and 2050 respectively. But the log doesn't contain $<T_1$, commit> for $<T_1$, start>. Hence, the value of account C to restored to old value by undo ($T_1$) operation. Hence value of C is `700.

**Case 3:** If system crashes just after the log record $<T_1$ commit> has been written to log.

- The log and database contents are:

| Log | Database |
|---|---|
| $<T_0$, start> | |
| $<T_0$, A, 1000, 950> | |
| $<T_0$, B, 2000, 2050> | |
| database ⟶ | A = 950 |
| | B = 2050 |
| $<T_0$, commit > | |
| $<T_1$, start > | |
| $<T_1$, C, 700, 600 > | |
| database ⟶ | C = 600 |
| $<T_1$, commit> | |

- Log contains $<T_0$ ,start>, $<T_0$, commit> and $<T_1$ ,start> $<T_1$, commit> operations. Hence recovery system executes redo ($T_0$) and redo ($T_1$) operations. The values of accounts A, B and C are ₹ 950, ₹ 2050 and ₹ 600 respectively.

## 4.7.4 Checkpoints

(W- 16, 17)

- When a system failure occurs, some transactions need to be redone and some need to be undone. Log record can find out this. But for that we need to search the entire log.

There are two major difficulties with this approach:

1. The search process is time consuming.
2. Most of the transactions that will be redone have already written their updates into the database. Hence, it is better to avoid such redo operations.

- To reduce these types of overhead, checkpoints are introduced. During the execution the system keeps up log using immediate database modification technique or deferred database modification technique.
- In addition, the system periodically performs checkpoints, which require following sequence of operations:
  1. Output onto stable storage all log records currently stored in main memory.
  2. Output to the disk all modified buffer blocks.
  3. Output onto stable storage a log record <checkpoint>.
- Transactions are not allowed to perform any update actions, such as writing to a buffer block or writing a log record, while a checkpoint is in progress.
- The presence of <checkpoint> record in log allows the system to streamline its recovery procedure. After the failure has occurred, the recovery system examines the log to determine the most recent transaction $T_i$ that started execution before the most recent checkpoint took place.
- It can find such a transaction by searching the log backward from the end of the log until it finds the first <checkpoint> record, then it continues the search backward until it finds the next <$T_i$, start> record. This record identifies a transaction $T_i$.
- Once, the transaction is identified, redo or undo operation can be applied to transaction $T_i$ and all the transactions T executing after $T_i$. The earlier part of transaction can be ignored.
- The recovery can be done by using immediate database modification or deferred database modification technique.

**Shadow Paging:**

- Shadow paging is an alternative to log-based crash recovery technique. This is one possible form of indirect page allocation.
- **Paging:** Paging scheme is used in operating system for virtual memory management. The memory that is addressed by a process is called virtual memory.
- It is divided into pages, that are assumed to be of a certain size (1 KB or 4 KB). The virtual or logical pages are mapped onto physical memory blocks of same size. The mapping of pages is provided by means of table called as **page table**.

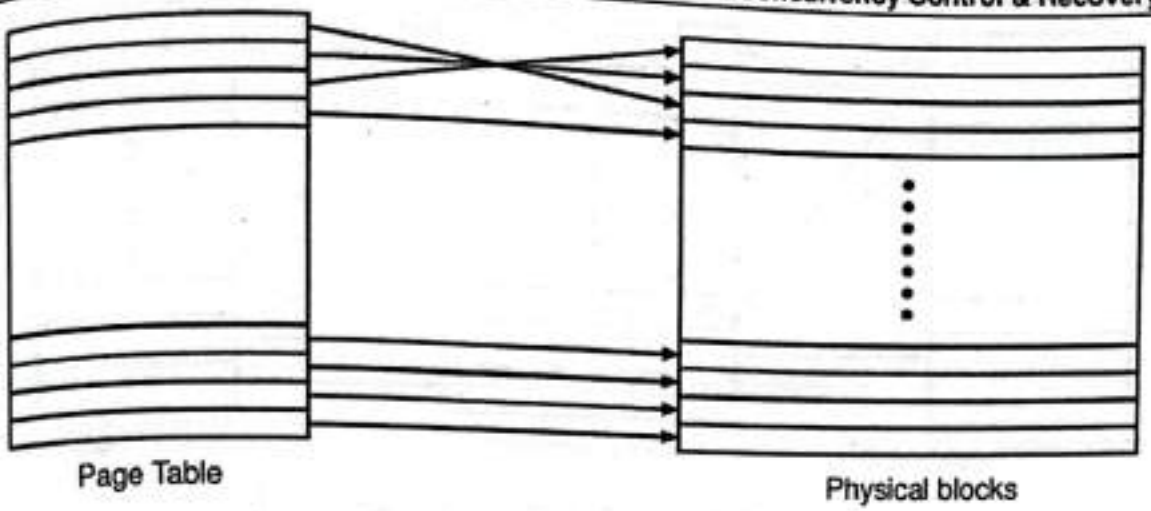
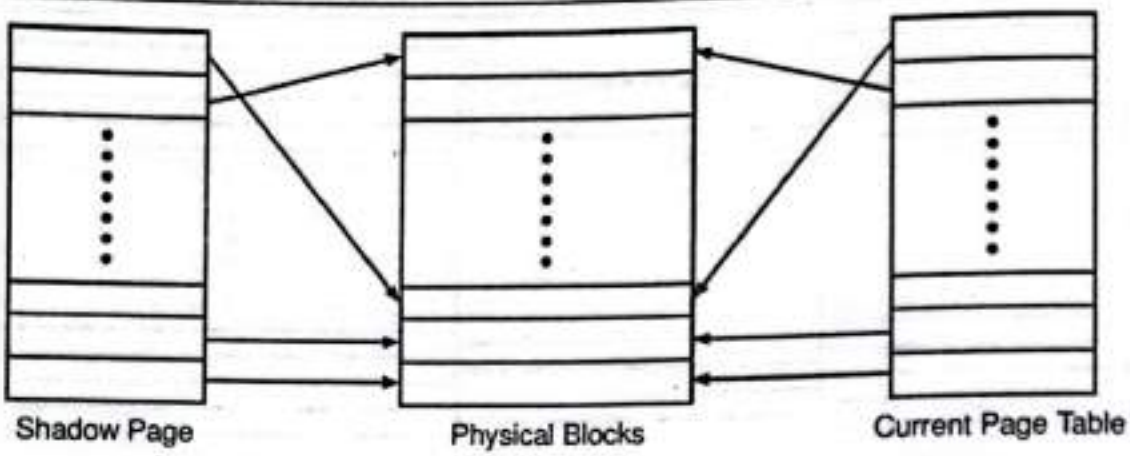



**Fig. 5.2: Page Table**

- Page table contains one entry for each logical page of the processes virtual address space. Page table is shown in the Fig. 5.2.
- In the shadow page scheme, the database is considered to be made up of logical units of storage called pages. The pages are mapped into physical blocks of storage by means of a page table with one entry for each logical page. This entry contains the block number of physical storage where this page is stored. The shadow page scheme uses two page tables:
  1. Current page table.
  2. Shadow page table.
- Transaction addresses the database using current page table. It may change the current page table entries. The changes are made whenever the transaction executes write operation.
- To modify a page, it copies that page to new blocks of physical storage. The page table entry corresponding to that page is made to point to new block of storage.
- The shadow page table is the original page table. It contains the entries that existed prior to the start of transaction. It remains unaltered by the transaction and it is used for undoing the transaction.

Now, let us see how the transaction accesses data.

- The transaction uses the current page table to access the database blocks. The shadow paging scheme handles a write operation of transaction as follows:
  1. A free block of non-volatile storage is located from the pool of free blocks accessible by the database system.
  2. The block to be modified is copied onto this block.
  3. The original entry in the current page table is changed to point to this new block.
  4. The updates are propagated to the block pointed to by the current page table which in this case would be newly created block.




**Before Starting the Transaction**

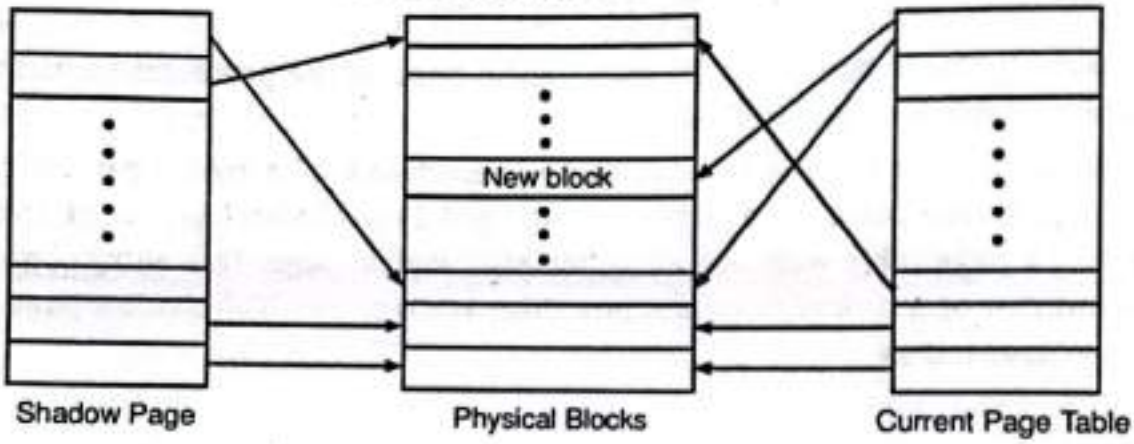



**Fig. 5.3: After Executing Write Operation**

- Any changes made to the database are propagated to the blocks pointed to by the current page table. Once, a transaction commits, all modifications made by the transaction and still in buffers are propagated to physical database. It causes the current page table to be written to non-volatile storage.
- In case of system crash before the transaction commits, the shadow page table and the corresponding blocks containing the old database will continue to be accessible.
- The old database is made accessible just by modifying a single pointer from shadow page table to current page table.
- Once, the transaction completes its execution successfully, the shadow block can be returned to the pool of available non-volatile storage blocks to be used for further transactions.
- **Advantage** of shadow paging scheme is:

  Recovery from system crash is relatively inexpensive and this is achieved without the overhead of logging.
- **Disadvantages** of shadow paging scheme are:

  1. Over a period of time the database will be scattered over the physical memory and related records may require a very long access time.

2. When the transaction completed its execution, shadow blocks have to be returned to the pool of free blocks. If this is not done successfully, when a transaction commits, such blocks become inaccessible. This is called as garbage collection operation.
3. The commit of a single transaction using shadow paging requires multiple blocks to be output (the actual data blocks, the current page table, and disk address of the current page table log based schemes need to output only the log records.

## 4.7.5 RECOVERY with Concurrent Transactions

Now we will discuss how user can modify and extend the log-based recovery scheme to deal with multiple concurrent transactions.

- The number of concurrent transactions, computer system has a single disk buffer and a single log. The buffer blocks are shared by all transactions.
- User allow immediate updates and the permit a buffer block to have data items updated by one or more transactions.

### 4.7.5.1 Transaction Rollback

- User rollback a failed transaction, $T_i$, using the log.
- The log is scanned backward; for every log record of the form $<T_i, X_j, V_1, V_2>$ found in the log, the data item $X_j$ is restored to its old value $V_1$.
- The scanning of the log terminates when the log record $<T_i$, start> is found.
- Scanning the log backward is more important, since a transaction may have updated a data item more than once. Consider the pair of log records as given below:

  $<T_i, A, 10, 20>$

  $<T_i, A, 20, 30>$

- They represent a modification of data item A by $T_i$, followed by another modification of A by $T_i$. Scanning the log backward set A correctly to 10.
- If the log were scanned in the forward direction, A would be set to 20, which value is incorrect.
- If strict two-phase locking is used for concurrency control, locks held by a transaction T may be released only after the transaction has been rolled back as described.
- Once, transaction T has updated a data item, no other transaction could have updated the same data item, due to the concurrency-control requirements.

**Interaction with Concurrency Control:**

- The recovery scheme depends on the concurrency-control technique that is used. To roll back a failed transaction, user must undo the updates or modification performed by the transaction.
- Suppose that a transaction $T_0$ has to be rolled back and a data item Q that was updated by $T_0$ has to be restored to its old value. Using the log-based schemes for recovery, we restore the value using the undo information in a log record.

- Suppose a second transaction $T_1$ has performed yet another update on Q before $T_0$ is rolled back. Then, the updation performed by $T_1$ will be lost if $T_0$ is rolled back.
- If a transaction T has updated a data item Q, no other transaction may update the same data item until T has committed. We can ensure this requirement easily by using strict two-phase locking.

### 4.7.5.2 Restart Recovery

(S-18;W-18)

When the system recovers from a crash or failure, it constructs two lists:

1. **Undo List::** Which consists of transactions to be undone.
2. **Redo-list:** Which consists of transactions to be redone?

- These two lists are constructed on recovery, initially, they are both empty. User scan the log backward, examining each record, until the first <checkpoint> record is found:
  - For each record found of the form <$T_i$ commit>, he/she adds $T_i$ to redo-list.
  - For each record found of the form <$T_i$ start>. If $T_i$ is not in redo-list, then we add $T_i$ to undo-list.
- When all the appropriate log records have been examined, user checks the list L in the checkpoint record. For each transaction $T_i$ in L, if $T_i$ is not in *redo-list* then we add $T_i$ to the *undo-list*.
- All transactions on the *undo-list* have been undone; those transactions on the *redo-list* are redone. It is important, to process the log forward. When the recovery process has completed, transaction-processing resumes.
- It is very important to undo the transaction in the *undo-list* before redoing transactions in the *redo-list*, using the preceding algorithm.
- Otherwise, the following problem may occur, suppose that data item X initially has the value 20. Suppose that a transaction $T_i$ updated data item X to 30 and aborted; transaction rollback would bring back A to the value 20.
- Suppose that another transaction $T_j$ then updated data item X to 40 and committed, after this the system crashed. The state of the log at the time of the crash or failure is,

  <$T_i$, X, 20, 30>

  <$T_j$, X, 20, 40>

  <$T_j$ commit>
- If the redo pass is performed first, X will be set to 40; then, in the undo pass, X will be set to 20 which are wrong. The final value of 'X' should be 40, which we can ensure by performing undo before performing redo.

**Buffer Management:**

- Now, we consider several obvious details that are essential to the implementation of a crash-recovery scheme that ensures data consistency and causes a minimal amount of overhead on interactions with the database.

**Log-Record Buffering:**

- We discussed earlier every log record is output to stable storage at the time it is created. This assumption causes a high overhead on system execution for the following reasons.
- Typically, output to stable storage is in units of blocks. In many cases, a log record is much smaller than a block. Thus, the output of each log record translates to a much larger output at the physical level.
- Due to the use of log buffering, a log record may be stored in only main memory for considerable time before it is output to stable storage.
- Since, such log records are lost if the system crashes or fails we must impose additional requirements on the recovery techniques to ensure transaction atomicity.
  1. $T_i$ Transaction enters that commit state after the <$T_i$ commit> log record has been output to stable storage.
  2. Before the <$T_i$ commit> log record can be output to stable storage, all log records belonging to transaction $T_i$ must have been output to stable storage.
  3. Before a block of data in main memory can be output to the database, all log records belonging to data in that block must have been output to stable storage.

**Database Buffering:**

- The database is stored in non-volatile storage such as disks, tape etc. and blocks of data are brought into main memory as needed.
- Since, main memory is typically much smaller than the entire database, it may be necessary to overwrite $B_1$ block in main memory when another $B_2$ block needs to be brought into memory.
- If $B_1$ block has been changed, $B_1$ must be output before the input of $B_2$. This storage hierarchy is the standard operating system concept of virtual memory.
- The rules for the output of log records limit the freedom of the system to output blocks of data. If the input of $B_2$ block cause $B_1$ block to be chosen for output, all log records be belonging to data in $B_1$ block must be output to stable storage before $B_1$ block is output. Thus, the sequence of actions by the system would be as given below:
  - Output log records to stable storage until all log records belonging to $B_1$ block have been output.
  - Output $B_1$ block to disk.
  - Input $B_2$ block from disk to main memory.
- It is very important that no writes to the block $B_1$ be in progress while the preceding sequence of actions is carried out. User can ensure that there are no writes in progress by using a special means of locking.

## SOLVED EXAMPLES:

(S-17, 18, 19; W-17, 18, 19)

**Example 1:** **Following is the list of events in an interleaved execution of set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL (Two-phase Lock). Is there a Deadlock? If yes, which transactions are involved in Deadlock?**

| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, S) |
| $t_3$ | $T_3$ | Lock (A, S) |
| $t_4$ | $T_1$ | Lock (C, X) |
| $t_5$ | $T_2$ | Lock (D, X) |
| $t_6$ | $T_1$ | Lock (D, S) |
| $t_7$ | $T_2$ | Lock (C, S) |

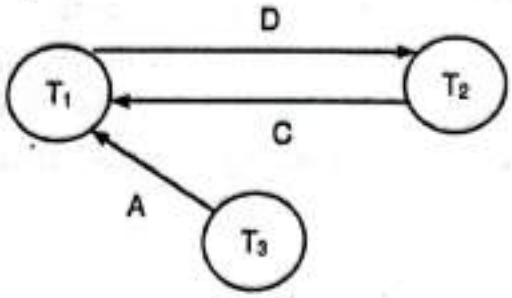
**Ans:**

As per the given list of events (from time $t_1$ to $t_7$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ |
|---|---|---|
| X(A) | | |
| | S(B) | |
| | | S(A) |
| X(C) | | |
| | X(D) | |
| S(D) | | |
| | S(C) | |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is cycle between $T_1$, $T_2$. Hence there is deadlock.

**Example 2:** **Following is the list of events in an interleaved execution of set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL. Is there a Deadlock? If yes, which transactions are involved in Deadlock?**

| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, X) |
| $t_3$ | $T_3$ | Lock (A, S) |
| $t_4$ | $T_4$ | Lock (B, S) |
| $t_5$ | $T_1$ | Lock (B, S) |
| $t_6$ | $T_2$ | Lock (D, S) |
| $t_7$ | $T_3$ | Lock (C, S) |
| $t_8$ | $T_4$ | Lock (C, X) |

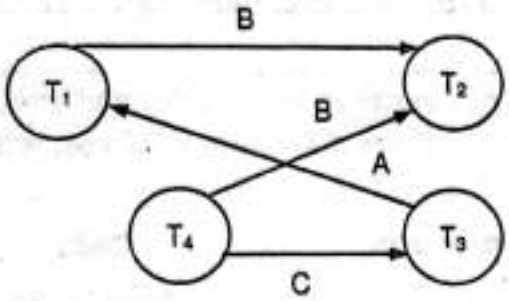
**Ans:**

As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | X(B) | | |
| | | S(A) | |
| | | | S(B) |
| S(B) | | | |
| | S(D) | | |
| | | S(C) | |
| | | | X(C) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is no cycle. Hence there is no deadlock.

**Example 3:** **Following is the list of events in an interleaved execution of set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL (Two-phaseLock). Is there a Deadlock? If yes, which transactions are involved in Deadlock?**

| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, X) |
| $t_3$ | $T_3$ | Lock (A, S) |
| $t_4$ | $T_4$ | Lock (B, S) |
| $t_5$ | $T_1$ | Lock (B, S) |
| $t_6$ | $T_3$ | Lock (D, X) |
| $t_7$ | $T_2$ | Lock (D, S) |
| $t_8$ | $T_4$ | Lock (C, X) |

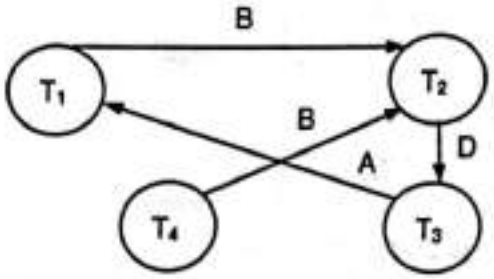
**Ans.**

As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | X(B) | | |
| | | S(A) | |
| | | | S(B) |
| S(B) | | | |
| | | X(D) | |
| | S(D) | | |
| | | | X(C) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is cycle among T1, T2 and T3. Hence there is deadlock.

---

**Example 4:** **Following is the list of events in an interleaved execution of set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL. Is there a Deadlock? If yes, which transactions are involved in Deadlock?**

| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, S) |
| $t_3$ | $T_3$ | Lock (A, S) |
| $t_4$ | $T_4$ | Lock (B, S) |
| $t_5$ | $T_1$ | Lock (B, X) |
| $t_6$ | $T_2$ | Lock (C, S) |
| $t_7$ | $T_3$ | Lock (D, S) |
| $t_8$ | $T_4$ | Lock (D, X) |

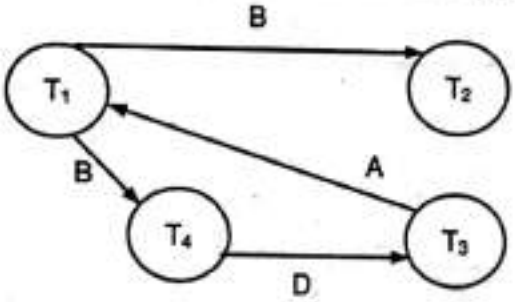
**Ans.**

As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | S(B) | | |
| | | S(A) | |
| | | | S(B) |
| X(B) | | | |
| | S(C) | | |
| | | S(D) | |
| | | | X(D) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is cycle among $T_1$, $T_4$ and $T_3$. Hence there is deadlock.

**Example 5:** **Following is the list of events in an interleaved execution of set of transaction $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL. Is there a deadlock? If yes, which transactions are involved in deadlock?**

| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock(A,X) |
| $t_2$ | $T_2$ | Lock(A,S) |
| $t_3$ | $T_3$ | Lock(A,S) |
| $t_4$ | $T_4$ | Lock(B,S) |
| $t_5$ | $T_1$ | Lock(B,X) |
| $t_6$ | $T_2$ | Lock(C,X) |
| $t_7$ | $T_3$ | Lock(D,S) |
| $t_8$ | $T_4$ | Lock(D,X) |

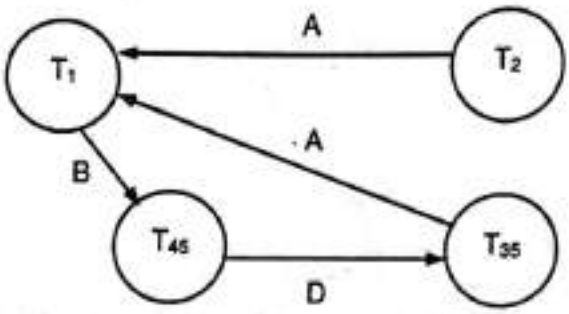
**Ans:**

As per the given list of events (from time $t_1$ to $t_7$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | S(A) | | |
| | | S(A) | |
| | | | S(B) |
| X(B) | | | |
| | X(C) | | |
| | | S(D) | |
| | | | X(D) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is cycle among $T_1$, $T_4$ and $T_3$. Hence there is deadlock.

**Example 6:** **Following is II list of events in an interleaved execution of set of transactions $T_1$, $T_2$, $T_3$ and $T_4$ with two-phasel ocking protocol.**

| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (B, S) |
| $t_2$ | $T_2$ | Lock (A, X) |
| $t_3$ | $T_3$ | Lock (C, S) |
| $t_4$ | $T_4$ | Lock (B, S) |
| $t_5$ | $T_1$ | Lock (A, S) |
| $t_6$ | $T_2$ | Lock (C, X) |
| $t_7$ | $T_3$ | Lock (A, X) |
| $t_8$ | $T_4$ | Lock (C, S) |

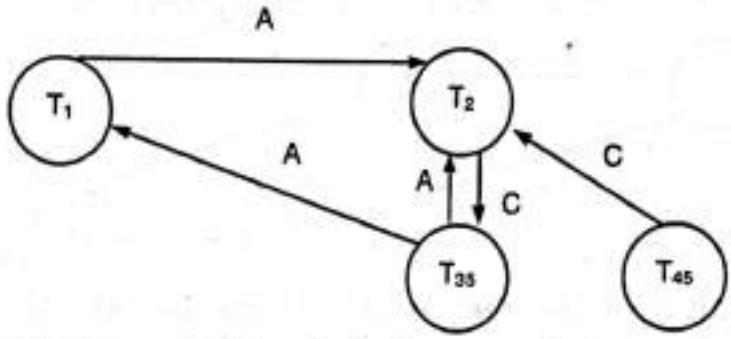
Construct a wait for graph according to above request. Is there deadlock at any instance? Justify.

**Ans.:** As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| S(B) | | | |
| | X(A) | | |
| | | S(C) | |
| S(A) | | | S(B) |
| | X(C) | | |
| | | X(A) | |
| | | | S(C) |

Where X(A) indicates exclusive lock on object A.

The wait-for graph for the above schedule is as given below:




From the above wait-for graph, there is cycle. Hence there is a deadlock.

**Example 7: Following is a list of events in an interleaved execution of set of transactions $T_1$, $T_2$, $T_3$ and $T_4$ with two-phase locking protocol.**

| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, S) |
| $t_3$ | $T_3$ | Lock (A, S) |
| $t_4$ | $T_4$ | Lock (C, S) |
| $t_5$ | $T_1$ | Lock (B, X) |
| $t_6$ | $T_2$ | Lock (C, X) |
| $t_7$ | $T_3$ | Lock (D, S) |
| $t_8$ | $T_4$ | Lock (D, X) |

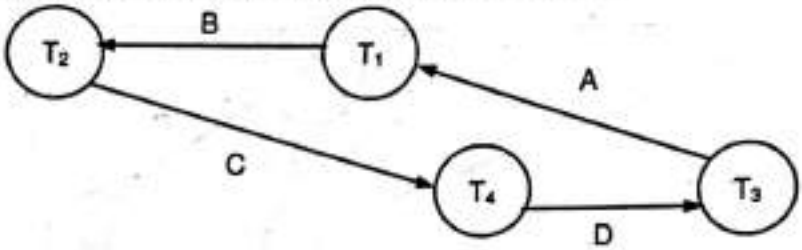
Construct a wait for graph according to above requests. Is there deadlock at any instance? Justify.

**Ans.:** As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | S(B) | | |
| | | S(A) | |
| | | | S(C) |
| X(B) | | | |
| | X(C) | | |
| | | S(D) | |
| | | | X(D) |

Where X(A) indicates exclusive lock on object A.

The wait-for graph for the above schedule is as given below:




From the above wait-for graph, there is cycle. Hence there is a deadlock.

**Example 8: Following are the log entries at the time of system crash.**

[start - transaction, T1]
[write - item, T1, A, 100]
[commit, T1]
[start - transaction, T3]
[write - item, T3, B, 200]
[checkpoint]
[commit, T3]
[start - transaction, T2]
[write - item, T2, B, 300]
[start - transaction, T4]
[write - item, T4, D, 200]
[write - item, T2, C, 300] → System crash

*If deferred update technique with checkpoint is used, what will be the recovery procedure?*

***Ans.:*** *Since transaction T1 is committed before the checkpoint, it need not be considered. Transaction T3 is committed after the checkpoint. Hence system can execute redo operation for T3. Transactions T2 and T4 do not have commit instruction. Hence no redo for them and the values of B, C, D remain unchanged.*

**Example 9:** **The following is the list of events in an interleaved execution if set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2 PL. Is there a deadlock? If yes, which transactions are involved in deadlock ?**

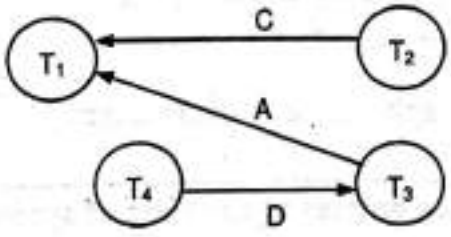
| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, S) |
| $t_3$ | $T_3$ | Lock (A, S) |
| $t_4$ | $T_4$ | Lock (B, S) |
| $t_5$ | $T_1$ | Lock (C, S) |
| $t_6$ | $T_2$ | Lock (C, X) |
| $t_7$ | $T_3$ | Lock (D, S) |
| $t_8$ | $T_4$ | Lock (D, X) |

**Ans.:** As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | S(B) | | |
| | | S(A) | |
| | | | S(B) |
| S(C) | | | |
| | X(C) | | |
| | | S(D) | |
| | | | X(D) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is no cycle. Hence there is no deadlock.

**Example 10:** **The following is the list of events in an interleaved execution if set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2 PL. Is there a deadlock? If yes, which transactions are involved in deadlock ?**

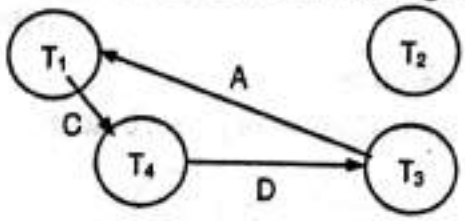
| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, S) |
| $t_3$ | $T_3$ | Lock (A, S) |
| $t_4$ | $T_4$ | Lock (C, S) |
| $t_5$ | $T_1$ | Lock (B, X) |
| $t_6$ | $T_2$ | Lock (C, X) |
| $t_7$ | $T_3$ | Lock (D, X) |
| $t_8$ | $T_4$ | Lock (D, S) |

**Ans.:** As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | S(B) | | |
| | | S(A) | |
| | | | S(C) |
| X(C) | | | |
| | X(B) | | |
| | | X(D) | |
| | | | S(D) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is cycle among $T_1$, $T_4$ and $T_3$. Hence there is deadlock.

**Example 11: Following are the log entries at the time of system crash.**

[start - transaction, $T_1$]
[write - item, $T_1$, A, 100]
[write - item, $T_1$, B, 100]
[commits, $T_1$]
[checkpoint]

[start - transaction, $T_3$]
[write - item, $T_3$, D, 500]
[commit, $T_3$]
[start - transaction, $T_4$]
[write - item, $T_4$, E, 400]
[start - transaction, $T_2$]
[write - item, $T_2$, C, 300] ← System crash

If deferred update technique with checkpoint is used, what will be the recovery procedure?

**Ans.:** $T_1$ is committed before the last system checkpoint, hence it need not be redone. $T_3$ is redone because its commit point is after the last system checkpoint.

$T_2$ and $T_4$ are not redone because there is no commit entry for these transactions. Hence, the values of C and E remain unchanged.

---

**Example 12: The following is the list of events in an interleaved execution of set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL. Is there a deadlock? If yes, which transactions are involved in deadlock?**

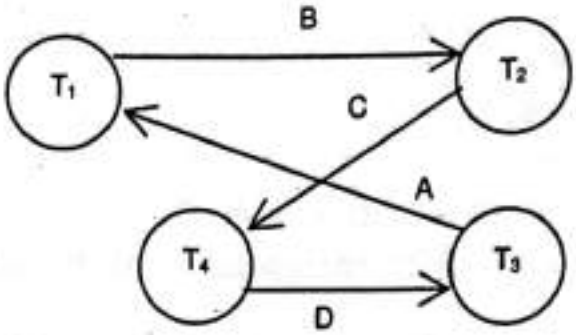
| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, S) |
| $t_3$ | $T_3$ | Lock (A, S) |
| $t_4$ | $T_4$ | Lock (C, S) |
| $t_5$ | $T_1$ | Lock (B, X) |
| $t_6$ | $T_2$ | Lock (C, X) |
| $t_7$ | $T_3$ | Lock (D, S) |
| $t_8$ | $T_4$ | Lock (D, X) |

**Ans.:** As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | S(B) | | |
| | | S(A) | |
| | | | S(C) |
| X(B) | | | |
| | X(C) | | |
| | | S(D) | |
| | | | X(D) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is cycle $T_1$-$T_2$-$T_4$-$T_3$-$T_1$. Hence there is deadlock.

**Example 13: The following is a list of events in an interleaved execution if set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL. Is there a deadlock ? If yes, which transactions are involved in deadlock?**

| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, X) |
| $t_3$ | $T_3$ | Lock (C, X) |
| $t_4$ | $T_4$ | Lock (A, S) |
| $t_5$ | $T_1$ | Lock (C, S) |
| $t_6$ | $T_2$ | Lock (D, S) |
| $t_7$ | $T_3$ | Lock (D, S) |
| $t_8$ | $T_4$ | Lock (B, S) |

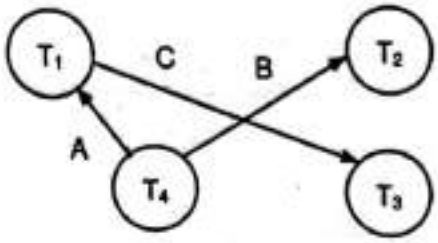
**Ans.:**

As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | X(B) | | |
| | | X(C) | |
| | | | S(A) |
| S(C) | | | |
| | S(D) | | |
| | | S(D) | |
| | | | S(B) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is no cycle. Hence there is no deadlock.

**Example 14: The following are the log entries at the time of system crash:**

[write - item, $T_1$, P, 10, 20]
[commit $T_1$]
[start - transaction, $T_2$]
[write_item, $T_2$, Q, 30, 20]
[write_item, $T_2$, R, 20, 30]
[commit, $T_2$]
[checkpoint]
[start-transaction, $T_4$]
[write_item, $T_4$, S, 20, 10]
[start-transaction, $T_3$]
[write_item, $T_3$, $T_1$, 10, 30] ← System crash

If immediate update with checkpoint is used, what will be the recovery?

**Ans.:** Log contains start instructions for $T_3$ and $T_4$, but does not contain commit instructions for $T_3$ and $T_4$. Hence, $T_3$ and $T_4$ need to be undone. Hence the value of the object modified by $T_4$ is restored to 20 and the object modified by $T_3$ is restored to 10. $T_1$ is committed before the last system checkpoint. Hence, it need not be redone.

The following is the list representing the sequence of events in an interleaved execution of set transaction $T_1$, $T_2$, $T_3$ and $T_4$ assuming two-phase locking protocol. Is there a deadlock? If yes, which transactions are involved in deadlock ?

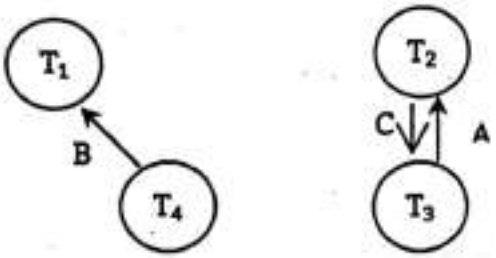
| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (B, S) |
| $t_2$ | $T_2$ | Lock (A, X) |
| $t_3$ | $T_3$ | Lock (C, S) |
| $t_4$ | $T_4$ | Lock (B, X) |
| $t_5$ | $T_1$ | Lock (D, S) |
| $t_6$ | $T_2$ | Lock (C, X) |
| $t_7$ | $T_3$ | Lock (A, S) |
| $t_8$ | $T_4$ | Lock (D, S) |

**Ans.:** As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| S(B) | | | |
| | X(A) | | |
| | | S(C) | |
| | | | X(B) |
| S(D) | | | |
| | X(C) | | |
| | | S(A) | |
| | | | S(D) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, cycle is present between $T_2$ and $T_3$. Hence there is a deadlock.

**Example 15:** **The following is the list representing the sequence of events in an interleaved execution of set transactions $T_1$, $T_2$, $T_3$ and $T_4$ assuming two-phase locking protocol. Is there a deadlock? If yes, which transactions are involved in deadlock?**

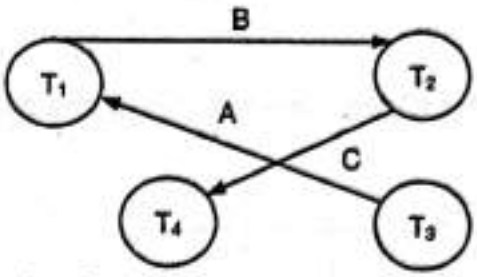
| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, S) |
| $t_3$ | $T_3$ | Lock (A, X) |
| $t_4$ | $T_4$ | Lock (C, S) |
| $t_5$ | $T_1$ | Lock (B, X) |
| $t_6$ | $T_2$ | Lock (D, S) |
| $t_7$ | $T_3$ | Lock (C, X) |
| $t_8$ | $T_4$ | Lock (D, S) |

***Ans.:*** *As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:*

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | S(B) | | |
| | | X(A) | |
| | | | S(C) |
| X(B) | | | |
| | X(C) | | |
| | | S(D) | |
| | | | S(D) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, cycle is not present. Hence there is no deadlock.

**Example 16: The following are the log entries at the time of system crash.**

[start_transaction, $T_1$]
[write item, $T_1$, A, 200, 400]
[commit, $T_1$]
[checkpoint]
[start_transaction, $T_3$]
[write_item, $T_3$, B, 100, 200]
[start_transaction, $T_4$]
[write_item, $T_4$, C, 300, 400]
[commit, $T_3$]
[write item, $T_4$, D, 200, 300] ← system crash

If immediate update with checkpoint is used, what will be the recovery?

**Ans.:** Log contains start instruction for $T_4$, but does not contain commit instructions for $T_4$ before the system crash. Hence, $T_4$ need to be undone. The values of C and D will now be changed to 300 and 200 respectively.

Log contains start and commits instruction for $T_3$ before the system crash. Hence, $T_3$ need to be redone. The value of B will be overwritten to 200.

$T_1$ is committed before the last system checkpoint. Hence, it need not be redone.

**Example 17:** **The following is the list of events in an interleaved execution of set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL protocol. Is there a deadlock? If yes, which transactions are involved in deadlock?**

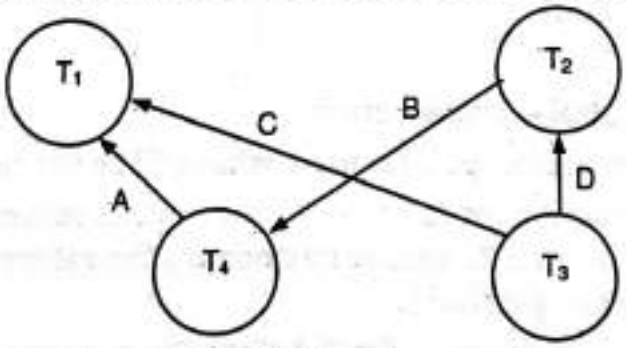
| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, X) |
| $t_3$ | $T_3$ | Lock (C, X) |
| $t_4$ | $T_4$ | Lock (A, S) |
| $t_5$ | $T_1$ | Lock (C, S) |
| $t_6$ | $T_2$ | Lock (D, S) |
| $t_7$ | $T_3$ | Lock (D, X) |
| $t_8$ | $T_4$ | Lock (B, X) |

**Ans.:** As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | X(B) | | |
| | | X(C) | |
| | | | S(A) |
| S(C) | | | |
| | S(D) | | |
| | | X(D) | |
| | | | X(B) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is no cycle. Hence there is deadlock.

**Example 18:** **The following is a list of representing the sequence of events in an interleaved execution of set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL protocol. Construct a wait for graph according to request. Is there a deadlock at any instance? Justify.**

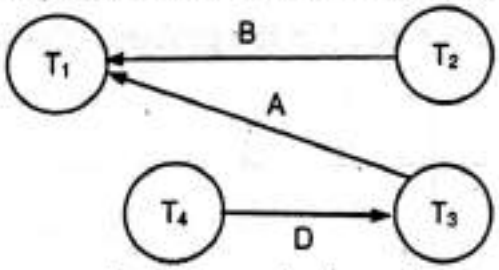
| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (C, S) |
| $t_3$ | $T_3$ | Lock (A, S) |
| $t_4$ | $T_4$ | Lock (C, S) |
| $t_5$ | $T_1$ | Lock (B, X) |
| $t_6$ | $T_2$ | Lock (B, S) |
| $t_7$ | $T_3$ | Lock (D, S) |
| $t_8$ | $T_4$ | Lock (D, X) |

**Ans.:** As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | S(C) | | |
| | | S(A) | |
| | | | S(C) |
| X(B) | | | |
| | S(B) | | |
| | | S(D) | |
| | | | X(D) |

Where X(A) indicates exclusive lock on object A.

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is no cycle. Hence there is no deadlock.

**Example 19:** **The following are the log entries at the time of system crash:**

[start - transaction, $T_2$]

[write - item, $T_1$, B, 200]

[commit , $T_1$]

[checkpoint]

[start - transaction, $T_2$]
[write_item, $T_2$, C, 250]
[commit, $T_2$]
[start-transaction, $T_3$]
[write_item, $T_3$, C, 300]
[start-transaction, $T_4$]
[write_item, $T_3$, A, 400]
[write_item, $T_3$, D, 250] ← System crash

If deferred update technique with checkpoint is used, what will be recovery procedure?

**Ans.:** Since transaction $T_1$ is committed before the checkpoint, it need not be considered. Transaction $T_2$ is committed after the checkpoint. Hence system can execute redo operation for $T_2$. Transactions $T_3$ and $T_4$ do not have commit instruction. Hence no redo for them.

**Example 20: The following is the list of events in an inter-leaved execution of set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL protocol. Is there a deadlock? If yes, which transactions are involved in deadlock.**

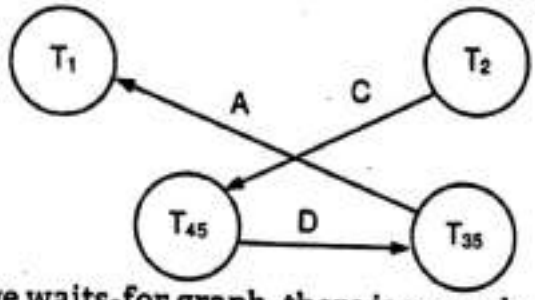
| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, S) |
| $t_3$ | $T_3$ | Lock (A, S) |
| $t_4$ | $T_4$ | Lock (C, S) |
| $t_5$ | $T_1$ | Lock (B, S) |
| $t_6$ | $T_2$ | Lock (C, X) |
| $t_7$ | $T_3$ | Lock (D, X) |
| $t_8$ | $T_4$ | Lock (D, S) |

**Ans.:**

As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | S(B) | | |
| | | S(A) | |
| | | | S(C) |
| S(B) | | | |
| | X(C) | | |
| | | X(D) | |
| | | | S(D) |

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is no cycle. Hence there is no deadlock.

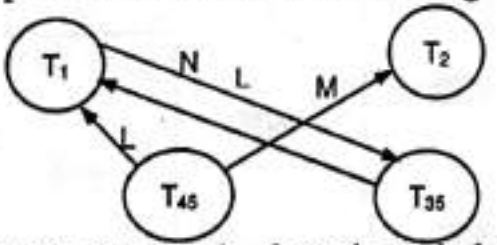
**Example 21: The following is a list of Events in an inter-leaved execution of set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL protocol. Is there a deadlock ? If yes, which transactions are involved in deadlock?**

| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (L, X) |
| $t_2$ | $T_2$ | Lock (M, X) |
| $t_3$ | $T_3$ | Lock (N, S) |
| $t_4$ | $T_4$ | Lock (L, S) |
| $t_5$ | $T_1$ | Lock (N, X) |
| $t_6$ | $T_3$ | Lock (L, S) |
| $t_7$ | $T_2$ | Lock (O, X) |
| $t_8$ | $T_4$ | Lock (M, S) |

**Ans.:** As per the given list of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(L) | | | |
| | X(M) | | |
| | | S(N) | |
| | | | S(L) |
| X(N) | | | |
| | | S(L) | |
| | X(O) | | |
| | | | S(M) |

The waits-for graph for the above schedule is as given below:




From the above waits-for graph, there is cycle between $T_1$ and $T_3$. Hence there is a deadlock.

**Example 22: The following are the Log entries at the time of system crash:**

[Write - item, $T_1$, C, 200]
[Commit $T_1$]
[Checkpoint]
[Start_transaction, $T_2$]
[Write_item, $T_2$, D, 100]
[Commit, $T_2$]
[Start_Transaction, $T_3$]
[Write_item, $T_3$, D, 150]
[Write_item, $T_3$, C, 250] ← System Crash

If deferred update technique with checkpoint is used, what will be recovery procedure?

**Ans.:** The recovery procedure by using deferred update technique with checkpoint: $T_1$ is committed before the last system checkpoint; hence it need not be redone. $T_2$ is redone because its commit point is after the last system checkpoint. Transaction $T_3$ does not have commit instruction before the system crash. Hence no redo for $T_3$ and the values of C, D remain unchanged.

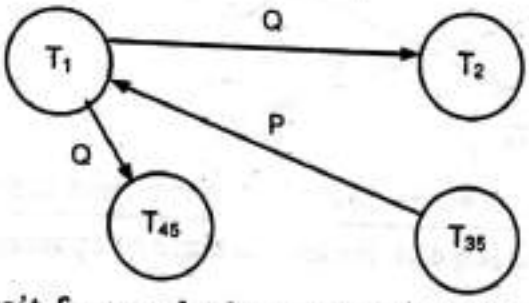
**Example 23: The following is the list of events in an interleaved execution of set $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2 PL. Is there a deadlock? If yes, which transactions are involved?**

| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (P, X) |
| $t_2$ | $T_2$ | Lock (Q, S) |
| $t_3$ | $T_3$ | Lock (P, S) |
| $t_4$ | $T_4$ | Lock (Q, S) |
| $t_5$ | $T_1$ | Lock (Q, X) |
| $t_6$ | $T_2$ | Lock (R, X) |
| $t_7$ | $T_3$ | Lock (U, S) |
| $t_8$ | $T_4$ | Lock (D, X) |

**Ans.:** As per the given sequence of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(P) | | | |
| | S(Q) | | |
| | | S(P) | |
| | | | S(Q) |
| X(Q) | | | |
| | X(R) | | |
| | | S(U) | |
| | | | X(D) |

Where X(A) indicates exclusive lock on object A.




From the above wait-for graph, there is no cycle. Hence there is no deadlock.

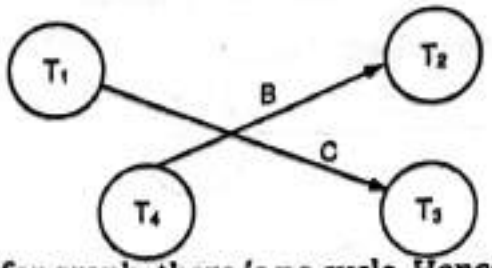
**Example 24: The following is a list of events in an interleaved execution of set of transaction $T_1$, $T_2$, $T_3$ and $T_4$ with two-phaselocking protocol. Construct the wait-for-graph. Is there a deadlock? If yes, specify which transactions are involved?**

| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, S) |
| $t_2$ | $T_2$ | Lock (B, X) |
| $t_3$ | $T_3$ | Lock (C, X) |
| $t_4$ | $T_4$ | Lock (A, S) |
| $t_5$ | $T_1$ | Lock (C, X) |
| $t_6$ | $T_2$ | Lock (A, S) |
| $t_7$ | $T_3$ | Lock (D, X) |
| $t_8$ | $T_4$ | Lock (B, S) |

**Ans.:** As per the given sequence of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| S(A) | | | |
| | X(B) | | |
| | | X(C) | |
| | | | S(A) |
| X(C) | | | |
| | S(A) | | |
| | | X(D) | |
| | | | S(B) |

Where X(A) indicates exclusive lock on object A.



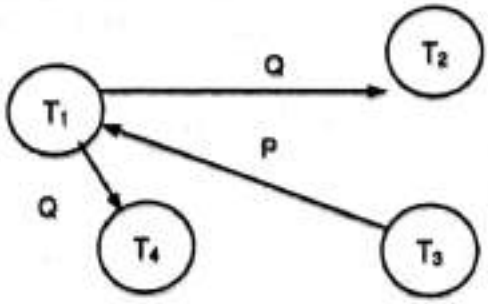

From the above wait-for graph, there is no cycle. Hence there is no deadlock.

**Example 25: The following are the log entries at the time of system crash:**

[Start-Transaction, $T_1$]
[Read item, $T_1$, U]
[Write-item, $T_1$, U, 40]
[Commit, T1]
[Check point]
[Start-Transaction, $T_2$]
[Read_item, $T_2$, Q]
[Write_item, $T_2$, Q, 32]
[Start-transaction, $T_3$]
[Write_item, $T_3$, P, 40]
[Read_item, $T_2$, U]
[Write_item, $T_3$, U, 45] ← System crash

If deferred update technique with checkpoint is used, what will be recovery procedure?

**Ans.:** As transaction $T_1$ is committed before the last system checkpoint, it need not be redone. Transactions $T_2$ and $T_3$ have not committed before the system crash. Hence no redo for them.




**Example 26: The following is the list of events in an interleaved execution of sets $T_1$, $T_2$, $T_3$ and $T_4$ assuming 2PL. Is there a deadlock? If yes, which transactions are involved in deadlock?**

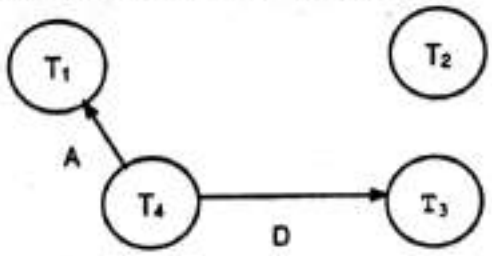
| Time | Transaction | Code |
|---|---|---|
| $t_1$ | $T_1$ | Lock (A, X) |
| $t_2$ | $T_2$ | Lock (B, S) |

| | | |
|---|---|---|
| $t_3$ | $T_3$ | Lock (C, S) |
| $t_4$ | $T_4$ | Lock (A, S) |
| $t_5$ | $T_1$ | Lock (C, X) |
| $t_6$ | $T_2$ | Lock (B, X) |
| $t_7$ | $T_3$ | Lock (D, X) |
| $t_8$ | $T_4$ | Lock (D, S) |

**Ans.:** As per the given sequence of events (from time $t_1$ to $t_8$) in the problem, the schedule can be written as follows:

| $T_1$ | $T_2$ | $T_3$ | $T_4$ |
|---|---|---|---|
| X(A) | | | |
| | S(B) | | |
| | | S(C) | |
| | | | S(A) |
| S(C) | | | |
| | X(B) | | |
| | | X(D) | |
| | | | S(D) |

Where X(A) indicates exclusive lock on object A.




From the above wait-for graph, there is no cycle. Hence there is no deadlock.

**Example 27: The following are the log entries at the time of system crash:**

[Start Transaction, $T_1$]
[Read_item, $T_1$, A]
[Write_item, $T_1$, 200]
[Commit $T_1$]
[Start Transaction, $T_1$]
[Read_item, $T_1$, A]
[Write_item, $T_1$, 200]
[Commit $T_1$]
[Start Transaction, $T_2$]

[Read_item, $T_2$, B]

[Write item, $T_2$, B]

[Checkpoint]

[Start Transaction, $T_3$]

[Read_item, $T_3$, A]

[Write item, $T_3$, 600]

[Commit]

[Write item $T_3$, B, 600] ← System Crash.

If immediate update with checkpoint technique is used, what will be the recovery procedure.

**Ans.:** The $12^{th}$ log entry given in the problem is ambiguous, as there is only Commit instruction without transaction identifier.

If we observe the log entries carefully, we find Commit for $T_1$ and $T_3$ is continued after $12^{th}$ log entry. Hence we can assume that $12^{th}$ log entry belongs to $T_2$.

Transaction $T_1$ is committed before the last system checkpoint, hence it need not be redone. Transaction $T_2$ is committed after the checkpoint, hence it needs to be redone. Log entries for transaction $T_3$ does not contain commit instruction before the system crash. Hence $T_3$ need to be undone.

**Example 28: The following are the log entries as the time of system crash:**

[Start Transaction, $T_1$]

[Write item, $T_1$, A, 500]

[Commit, $T_1$]

[Start Transaction, $T_2$]

[Write item, $T_2$, B, 200]

[Commit, $T_1$]

[Checkpoint]

[Start Transaction, $T_3$]

[Write item, $T_3$, C, 100]

[Write item, $T_3$, C, 200] ← System Crash.

If deferred update technique with checkpoint is used, what will be the recovery procedure?

**Ans.:** Transaction $T_1$ and $T_2$ are committed before the last system checkpoint; hence there is no need for redo.

$T_3$ is not redone because there is no commit entry for this transaction in the log after the last system checkpoint.

## Summary

- A lock is a variable associated with each data item. Manipulating the value of lock is called locking.
- There are two types of locks: Shared, Exclusive.
- Two-Phase Locking protocol contains growing phase and shrinking phase.
- In growing phase, a transaction may obtain locks, but may not release any lock.
- In shrinking phase, a transaction may release locks, but may not obtain any new locks.
- In strict 2PL, in addition to locking being two-phase, all exclusive-mode locks taken by a transaction must be held until that transaction commits.
- In rigorous 2PL, it requires that all locks to be held until the transaction commits.
- Upgrading of lock can take place in growing phase and downgrading can take place in only the shrinking phase.
- Two-phase locking with lock conversion generates conflict serializable schedule.
- There are two methods for implementing timestamp scheme: (i)use the value of system clock as the timestamp (ii) use a logical counter that is incremented after a new timestamp has been assigned.
- Timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.
- A system is in deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set.
- There are two deadlock prevention schemes, namely wait-die and wound-wait.
- Wait-die scheme is based on non-preemptive technique and Wound-wait scheme is based on preemptive technique.
- Wait-die and Wound-wait avoid starvation.
- If the probability that system enters deadlock state is relatively low, then deadlock detection method is efficient.
- Deadlocks can be detected using a directed graph called wait for graph.
- A transaction can be recovered from deadlock using these three actions:(i) Selection of victim, (ii) Rollback, (iii) Starvation
- Failures are classified into transaction failure, system crash, disk failure.
- There are two techniques for using log to achieve the recovery and ensure atomicity in case of failures: (i) Deferred database modification, (ii) Immediate database modification.
- The <checkpoint> record in log allows the system to streamline its recovery procedure.

## Check Your Understanding

**I. Multiple Choice Questions:**

1. A ..... is a variable associated with each data item.
   (a) lock (b) transaction
   (c) deadlock (d) none of these
2. A ..... lock can allow a transaction to read a data item, but cannot allow to modify(write) it.
   (a) shared (b) exclusive
   (c) both shared and exclusive (d) none of these
3. A ..... lock can allow a transaction to read and modify(write) a data item.
   (a) shared (b) exclusive
   (c) both shared and exclusive (d) aborted
4. Locking protocols restrict the number of possible ..... schedules.
   (a) serializable (b) serial
   (c) non serializable (d) none of these
5. In ..... phase, a transaction may obtain locks, but may not release any lock.
   (a) shrinking (b) growing
   (c) both growing and shrinking (d) none of these
6. In ..... 2PL all locks are held until the transaction commits.
   (a) strict (b) rigorous
   (c) simple (d) none of these
7. ..... scheme is based on non-preemptive technique.
   (a) Wound-wait (b) Wound-die
   (c) Wait-die (d) none of these
8. Which of the following is not a transaction failure?
   (a) System Error (b) Logical Error
   (c) System Crash (d) none of these
9. There is a hardware bug or error in the database software or the operating system that causes the loss of the content of volatile storage and leads transaction processing to a halt. Which type of failure is mentioned in this statement?
   (a) System Crash (b) Transaction Failure
   (c) Disk Failure (d) none of these
10. ..... is a structure used to store the database modifications.
    (a) Log (b) Register
    (c) Lock (d) none of these

11. Database modifications written by active transactions are called ..... modifications.
    (a) committed (b) uncommitted
    (c) direct (d) none of these

**Ans.:** (1) a (2) a (3) b (4) a (5) b (6) b (7) c (8) c (9) a (10) a (11) b.

**II. State TRUE/FALSE.**

1. Serializability can easily be ensured if access to database is done in mutually exclusive manner.
2. Locking protocols do not indicate when a transaction may lock and unlock each of the data items.
3. In shrinking phase, a transaction may release locks, but may not obtain any new locks.
4. In strict 2PL, all exclusive-mode locks taken by a transaction must be held until that transaction commits.
5. Wound-wait scheme is based on preemptive technique.
6. Those transactions that will incur the minimum cost will be rolled back in deadlock recovery.
7. A system is reliable if it works as per its specifications and produces correct set of output values for a given set of input values.
8. The failure of a system occurs when the system works according to its specifications.
9. System Error and Logical Error are types of System Crash.
10. Deferred modification technique ensures atomicity by recording all database modifications in the log, but delaying the execution of all write operations of transaction until the transaction partially commits.
11. To reduce the overheads in transaction recovery, checkpoints are introduced.

**Answer:**

| | | | |
|---|---|---|---|
| (1) TRUE | (2) FALSE | (3) TRUE | (4) FALSE |
| (5) TRUE | (6) TRUE | (7) TRUE | (8) FALSE |
| (9) FALSE | (10) TRUE | (11) TRUE | |

## Practice Questions

**Q.1 Answer the following questions in brief.**

1. What is concurrency control?
2. State the two phases of 2PL.
3. State the difference between shared and exclusive lock.
4. How to detect a deadlock?
5. What do you mean by deadlock handling?

6. Differentiate between wound-wait and wait-die protocol.
7. List the factors for selection of victim in deadlock recovery.
8. What are the types of transaction failures?
9. What is deferred update method?

**Q.2 Answer the following questions.**

1. Explain two-phase lock protocol with example.
2. Explain timestamp based and lock based protocols.
3. When do deadlocks happen, how to prevent them, and how to recover if deadlock takes place?
4. Describe the different variations of 2PL.
5. Compare between deadlock detection and deadlock prevention.
6. Explain in detail about deadlock detection.
7. Differentiate between deferred update and immediate update method.
8. Explain the different types of failures in detail.
9. Explain immediate update method with the help of an example.

**Q.3 Define the terms:**

(a) Lock (b) 2PL (c) Strict 2PL (d) Rigorous 2PL
(e) Deadlock (f) wait for graph (g) Starvation (h) Checkpoint
(i) Log

## Previous Exams Questions

### Summer 2017

**1.** Define: (i) Growing phase, (ii) Shrinking phase [2 M]
**Ans.** Please refer Section 4.2.3.

**2.** Give Lock compatibility matrix. [2 M]
**Ans.** Please refer Section 4.2.1.1.

**3.** What is deadlock? Explain deadlock prevention methods. [4 M]
**Ans.** Please refer Section 4.5 and 4.5.1.

**4.** Explain Thomas write rule. [4 M]
**Ans.** Please refer Section 4.3.3.

**5.** List different types of failures. [2 M]
**Ans.** Please refer Section 4.5.

### Winter 2017

**1.** What is deadlock ? [2 M]
**Ans.** Please refer Section 4.5.

2. Explain, how deadlock is recovered. [4 M]

**Ans.** Please refer Section 4.5.3.

3. Explain Validation Based Protocol. [4 M]

**Ans.** Please refer Section 4.4.

4. List and define basic operations used to recover from failure. [2 M]

**Ans.** Please refer Section 4.5

5. Write a short note on checkpoint. [4 M]

**Ans.** Please refer Section 4.7.4.

## Summer 2018

1. What is Deadlock ? Explain how deadlock is handled. [4 M]

**Ans.** Please refer Section 4.5.

2. Explain two-phase locking protocol with example. [4 M]

**Ans.** Please refer Section 4.2.3.

3. Define : (i) Redo, (ii) Undo [2 M]

**Ans.** Please refer Section 4.7.5.2.

4. Define : (i) Logical error, (ii) System error. [2 M]

**Ans.** Please refer Section 4.6.1

5. Explain immediate database modification technique in detail with example. [4 M]

**Ans.** Please refer Section 4.7.3.

6. What is Log ? Explain log-based recovery. [4 M]

**Ans.** Please refer Section 4.7.1.

7. Define: [2 M]
   (i) Upgrading
   (ii) Downgrading

**Ans.** Please refer Section 4.2.3.1.

## Winter 2018

1. What is lock? [2 M]

**Ans.** Please refer Section 4.2.1.

2. Define: [2 M]
   (i) Upgrading
   (ii) Downgrading

**Ans.** Please refer Section 4.2.3.1.

3. Define: (i) Redo, (ii) Undo [2 M]

**Ans.** Please refer Section 4.6.6

4. List the failure type. [2 M]

**Ans.** Please refer to section 4.5

5. Explain log-based recovery in detail. [4 M]

**Ans.** Please refer Section 4.6.1

6. What is deadlock? Explain deadlock detection. [4 M]

**Ans.** Please refer Sections 4.5 and 4.5.2.

**Summer 2019**

1. Define lock. State types of lock. [2 M]

**Ans.** Please refer Section 4.2.1.

2. What is timestamp? [2 M]

**Ans.** Please refer Section 4.3.1.

3. List the failure types. [2 M]

**Ans.** Please refer Section 4.5.

4. What is deadlock? Explain any two methods of prevent deadlock. [4 M]

**Ans.** Please refer Sections 4.5 and 4.5.1.

5. Explain two-phase locking protocol in detail. [4 M]

**Ans.** Please refer Section 4.2.3.

❖❖❖

## SALIENT FEATURES OF THE BOOK

- Most Recommended and Useful Text Books for **B.B.A. (Computer Application) Semester-II with CBCS Pattern**
- As Per Latest Syllabus of SPPU
- Carefully Written, for Best of the Inputs for Graduate Students
- Well Suited for the General referencing for any other courses
- All topics covered with variety of programs

## BEST COVERAGE ON

This book on **"RELATIONAL DATABASE "** covers the following topics:

- Introduction to RDBMS
- PL/SQL
- Transaction Management
- Cocurrency Control and Recovery System

## OUR MOST RECOMMENDED TEXT BOOKS FOR B.B.A.(C.A.) : SEMESTER-II

| | |
|---|---|
| • Organizational Behavior And Human Resource Management | : Gauri Girish Jadhav |
| • Financial Accounting | : Dr. Suhas Mahajan, Dr. Mahesh Kulkarni |
| • Business Mathematics | : A. V. Rayarikar, Dr. P. G. Dixit |
| • Relational Database | : Dr. Ms. Manisha Bharambe, Abhijeet Mankar |
| • Web Technology (HTML-JSS-CSS) | : Bhupesh Taunk, Aniket Nagane |

## BOOKS AVAILABLE AT


**PRAGATI BOOK CENTER - Email: pbcpune@pragationline.com**

- 157 Budhwar Peth, Opp. Ratan Talkies, Next To Balaji Mandir, **Pune** 411002 • **Mobile :** 9657703148
- 676/B Budhwar Peth, Opp. Jogeshwari Mandir, **Pune** 411002 Tel : (020) 2448 7459 • **Mobile :** 9657703147 / 9657703149
- 152 Budhwar Peth, Near Jogeshwari Mandir, **Pune** 411002 **Mobile :** 8087881795
- 28/A Budhwar Peth, Amber Chambers, Appa Balwant Chowk, **Pune** 411002 • **Tel :** (020) 6628 1669 • **Mobile** : 9657703142

**PRAGATI BOOK CORNER - Email: niralimumbai@pragationline.com**

- Indira Niwas, 111 - A, Bhavani Shankar Road, Dadar (W), **Mumbai** 400028. Tel: (022) 2422 3526/6662 5254

**niralipune@pragationline.com | www.pragationline.com**

**Also find us on** **www.facebook.com/niralibooks**

**@nirali.prakashan**



N4944

ISBN 978-93-89768-678-7