SURESH GYAN VIHAR UNIVERSITY
Accredited by NAAC with 'A+' Grade

Bachelor of Computer Application

(B.C.A.)

Relational Database Management System

Semester-III

Author- Mr. Gautam A. Kudale

SURESH GYAN VIHAR UNIVERSITY

Centre for Distance and Online Education
Mahal, Jagatpura, Jaipur-302025

Published by:

S. B. Prakashan Pvt. Ltd.

WZ-6, Lajwanti Garden, New Delhi: 110046

Tel.: (011) 28520627 | Ph.: 9205476295

Email: info@sbprakashan.com | Web.: www.sbprakashan.com

Designed & Graphic by : S. B. Prakashan Pvt. Ltd.

Printed at :

Syllabus

Relational Database Management System

Learning Objectives

- Understand the basic concepts and the applications of database systems.
- Master the basics of SQL and construct queries using SQL.
- Understand the relational database design principles.
- Familiar with the basic issues of transaction processing and concurrency control.
- Familiar with database storage structures and access techniques.

Unit I

Database System Architecture – Data Abstraction, Data Independence, Data Definitions and Data Manipulation Languages. Data models – Entity Relationship (ER), Mapping ER Model to Relational Mode, Network. Relational and Object Oriented Data Models, Integrity Constraints and Data Manipulation Operations.

Unit II

Relation Query Languages, Relational Algebra, Tuple and Domain Relational Calculus, SQL and QBE. Relational Database Design: Domain and Data dependency, Armstrong's Axioms, Normal Forms, Dependency Preservation, Lossless design, Comparison of Oracle & DB2.

Unit III

Query Processing and Optimization: Evaluation of Relational Algebra Expressions, Query Equivalence, Join strategies, Query Optimization Algorithms.

Unit IV

Storage Strategies: Indices, B-Trees, Hashing, Transaction processing: Recovery and Concurrency Control, Locking and Timestamp based Schedulers, Mult version and Optimistic Concurrency Control Schemes. Advanced Topics: Object-Oriented and Object Relational databases. Logical Databases, Web Databases, Distributed Databases, Data Warehouse and Data Mining.

References

- Database System Concepts by Sudarshan, Korth (McGraw-Hill Education)
- Fundamentals of Database System By Elmasari &Navathe- Pearson Education
- An introduction to Database System – Bipin Desai, Galgotia Publications
- Database System: concept, Design & Application by S.K.Singh (Pearson Education)
- Database management system by leon &leon (Vikas publishing House).
- Database Modeling and Design: Logical Design by Toby J. Teorey, Sam S. Lightstone, and Tom Nadeau, "", 4thEdition, 2005, Elsevier India Publications, New Delhi
- Fundamentals of Database Management System – Gillenson, Wiley India

Contents

Chapter 1

INTRODUCTION TO RDBMS

1. Introduction

The relational model was first introduced by *Ted Codd* of IBM research in 1970. The relational model represents the database as a collection of relations. Informally, each relation resembles a table of values or to some extent a 'flat' file of records. In the formal relational model terminology, a row is called 'tuple', a column header is called an 'attribute', and the table is called a 'relation'. The data type describing the types of values that can appear in each column is called a 'domain'.

2. Introduction to Popular RDBMS Product And their Features

A Relational Database Management System (RDBMS) is a Database Management System (DBMS) that is based on relational model as introduced by *Dr. Edger F.Codd.* Most popular commercial and open source databases currently in use are based on the relational model.

RDBMS stores data in the form of related tables. RDBMS are powerful because they require few assumptions about how data is related or how it will be extracted from the database. The same database can be viewed in many different ways.

Characteristics/Features of RDBMS

4

Oct.15,Apr.15,12 – 2M
What is RDBMS? List any two features of RDBMS.

Oct. 2010 – 2M
Explain any two distinguishing characteristics of RDBMS.

i. **Data independence:** Application programs do not depend on data. The structure of data is stored separately in the system catalog from the asses of application programs. Any updates on data application programs are not identified.

ii. **Data integrity:** Components like roll back operations, referential integrity and transaction oriented operations are designed to ensure integrity constraints.

iii. **Fast response rate:** Data is centrally located so request of data can be completed immediately.

iv. **Controlled redundancy:** According to relational databases data replication, wastage of storage space, data normalization concepts at higher level redundancy can be removed.

v. **Restricting unauthorized access:** RDBMS provides security and data authorization by creating users at different level.

vi. **Multiple user interface:** Many database software's are provided by using several programming interfaces, query languages, forms, menu driven interfaces.

vii. **Concurrency control:** This mechanism is used to manage multiple users accessing the same resources.

viii. **Backup and recovery:** This facility is used for data recovery from both hardware and software failures.

ix. RDBMS supports client server architecture.

x. It provides security, protection, maintenance, reliability and performance on operation of data.

The popular commercial RDBMS for large database includes Oracle, Microsoft Access, Microsoft SQL Server, Sybase SQL Server and IBM'S DB2.

2

Oct.2015 – 2M
Enlist the RDBMS products.

Oct.2014 – 2M
What is RDBMS? List any two products of RDBMS.

Products of RDBMS

i. **ThinkSQL:** It is a **cross platform** RDBMS.

ii. **Microsoft access:** It is an **entry level DBMS** from Microsoft.

iii. **MySql**

a. It is an open source RDBMS.

b. Available on many different platforms including windows, linux, UNIX and Mac OS.

2. **Queries:** Allow the user to view, change and analyze data in different ways. Queries can also be stored and used as the source of records for forms, reports and data access pages.
3. **Forms:** Can be used for variety of purposes such as create a data entry form to enter data into a table.
4. **Reports:** Allow data in the database to be presented in an effective way in a customized printed format.
5. **Pages:** A (data access) page is a special type of web page designed for viewing and working with data from the Internet or an Intranet.
6. **Macros:** A set of one or more actions that each performs a particular operation, such as opening a form or printing a report.
7. **Modules:** A collection of VBA declarations and procedures those are stored together as a unit.

Microsoft Access can be used as a standalone system on a single PC or as a multi-user system on a PC network. With the release of Access 2000, there is a choice of two data engines in the product: the original jet engine and the new Microsoft Data Engine (MSDE), which is compatible with Microsoft's back office SQL Server.

Microsoft Access provides four main ways of working with a database that is shared among users on a network.

1. **File-server solutions:** An access database is placed on a network so that multiple users can share it.
2. **Client-server solutions:** An access project (.adp) file can also be created, which can store forms, reports, macros, and VBA modules locally and can connect to a remote SQL server database using OLE DB (Object Linking and Embedding for Databases) to display and work with tables, views, relationships and stored procedures.
3. **Database replication solutions:** These allow data or database design changes to be shared between copies of an access database in different locations without having to redistribute copies of the entire database.
4. **Web-based database solutions:** A browser displays one or more data access pages that dynamically link to a shared access or SQL server database.

Oracle

The Oracle Corporation is the world's leading supplier of software for information management and the world's second largest independent software company. The user interacts with Oracle and develops a database using a number of objects.

1

Apr.2012 – *4M*
Explain any four objects of Oracle.

The main objects in oracle are:

1. **Tables:** A table is organized into columns and rows.
2. **Objects:** A way to extend Oracle's relational data type system.

3. **Clusters**: A set of tables physically stored together as one table that shares a common column.
4. **Indexes:** A structure used to help retrieve data more quickly and efficiently.
5. **Views:** Virtual tables.
6. **Synonyms:** An alternative name for an object in the database.
7. **Sequences:** Generates a unique sequence of numbers in cache.
8. **Functions/Procedures:** A set of SQL or PL/SQL statements used together to execute a particular function.
9. **Packages:** A collection of procedures, functions, variables and SQL statements that are grouped together and stored as a single program unit.
10. **Triggers:** Code stored in the database and invoked–triggered-by-events that occur in the application.

2
Oct.2010 – *4M*
What are the features of oracle?
Apr.2012 – *4M*
What is RDBMS? List any two features of RDBMS.

Features of Oracle

1. It is a Relational database management system.
2. It is used in many database applications on several operating system platform including Unix and Windows.
3. It was the first commercial RDBMS that becomes available on Linux.
4. It offers technology having comprehensive pre-integrated business applications.
5. It provides security, protection, maintenance, reliability and performance on operation on data.
6. It provides efficient and fast database recovery.

3. Difference between DBMS and RDBMS

A Database Management System (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is hence a general–purpose software system that facilitates the processes defining, constructing and manipulating database for various applications.

Defining a database involves specifying the data types, structures and constraints for the data to be stored in the database.

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

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

For example, Let us consider the database of university for marinating information concerning students, courses and grades in a university environment.

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

A relational database schema S is a set of relation schemas S = {R1, R2,...,Rm} and a set of integrity constraints IC.

In relational model all data is logically structured within relations (tables). Each relation has a name and is made up of named attributes (columns) of data. Each tuple (row) contains one value per attribute.

6

Oct.15,14,11,Apr.11–4M
Differentiate between DBMS and RDBMS with example.

Oct.12, Apr.10 – 2M
Give any two differences between DBMS and RDBMS.

The main difference between DBMS and RDBMS is as follows:

	DBMS	RDBMS
i.	It is Database Management System.	It is Relational Database Management System.
ii.	It includes theoretical part how data is stored in table.	It is the procedural way that includes SQL syntax for relating tables with one another and handling the stored data in table.
iii.	It is feasible for small organizations.	It maintains relationships among large amount of data.
iv.	It does not follow normalization concept.	It follows normalization concept.
v.	In DBMS relationship between two tables or files are maintained programmatically.	In RDBMS relationship between two tables or files can be specified at the time of table creation.
vi.	DBMS does not support Client/Server Architecture.	Most of the RDBMS supports Client/Server Architecture.
vii.	DBMS does not support Distributed databases.	Most of the RDBMS supports Distributed databases.
viii.	In DBMS there is no security of data.	In RDBMS there are multiple level of security like i. Logging in at O/S level. ii. Command level (i.e. at RDBMS level). iii. Object level.
ix.	Each table is given an extension in DBMS.	Many tables are grouped in one database in RDBMS.
x	It is single user system.	It is multi-user system.
xi.	**Naming Conventions**	
	Field	Column, Attributes, Data Field
	Record	Row, Tuple, Entity
	File	Table, Relation, Entity class
xii.	e.g., FoxPro, IMS.	e.g., SQL server, Oracle.

4. Relationship among Application Programs and RDBMS

Here we reviewed two representative and very popular Relational Database Management System (RDBMS) products: Oracle and Microsoft Access. We introduced the typical architecture and functionality of a high-end product like oracle and a PC-based smaller RDBMS like access. While we may call oracle a full-fledged RDBMS, we may call Access a data management tool that is geared for the less sophisticated user. We have also described the main functions of the oracle system, and reviewed some of the tools available in Oracle for database design and application development. We have also provided an overview of Microsoft Access, its architecture and reviewed some additional features and functionality of Access.

Application of a RDBMS

1

Oct.2009 – *2M*
What is RDBMS? State popular commercial RDBMS Applications.

i. **Banking: all kinds of transactions:** The Banking Application can look into customer processing, Account processing, Loan processing, and address all the functionalities of the Bank.

ii. **Airlines: reservations, schedules:** Reservation System could be of any type Airlines, Railway or Bus. The processing remains almost same with few changes in the processing which includes: customer request processing, ticket reservation, schedule processing and finally the billing of the customers.

iii. **Universities: registration, grades:** This system looks into admission processing, short listing, entrance examination, scheduling interviews, etc. and post admission. It addresses all the stages of a student processing, from his admission, to examination, attendance, (library processing) and ultimately the grades.

iv. **Sales: customers, products, purchases:** This system looks into customer processing as regards, enquiries, sales orders, delivery with bill and finally billing of the customer. Sales could either be an extended application of purchases or retail industry. If the purchases are considered then it involves supplier processing, quotation processing, purchase orders, billing of suppliers and most important the inventory processing.

v. **Manufacturing:** Production, inventory, orders, supply chain.

PU Questions

2 Marks

[Oct.2015 – 2M] 1. Enlist the RDBMS products.

[Oct.14,15,Apr.15,12 – 2M] 2. What is RDBMS? List any two features of RDBMS.

[Oct.2014 – 2M] 3. What is RDBMS? List any two products of RDBMS.

[Oct.12,Apr.10 – 2M] 4. Give any two differences between DBMS and RDBMS.

[Oct.2011 – 2M] 5. What are the main objects in MS-Access?

[Apr.2011 – 2M] 6. List four products of RDBMS.

[Oct.2010 – 2M] 7. Explain any two distinguishing characteristics of RDBMS.

[Oct.2009 – 2M] 8. What is RDBMS? State popular commercial RDBMS Applications.

4 Marks

[Oct.15,14,11, Apr.11–4M] 1. Differentiate between DBMS and RDBMS with example.

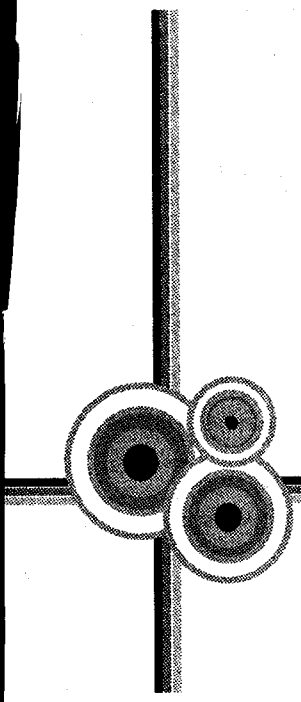
[Apr.2015 – 4M] 2. Explain any two popular products of RDBMs.

[Oct.2012 – 4M] 3. Explain any two popular products of RDBMS.

[Apr.2012 – 4M] 4. Explain any four objects of Oracle.

[Oct.2010 – 4M] 5. What are the features of oracle?

[Oct.2009 – 4M] 6. Write a note on any two products of RDBMS.



Chapter 2

PL/SQL

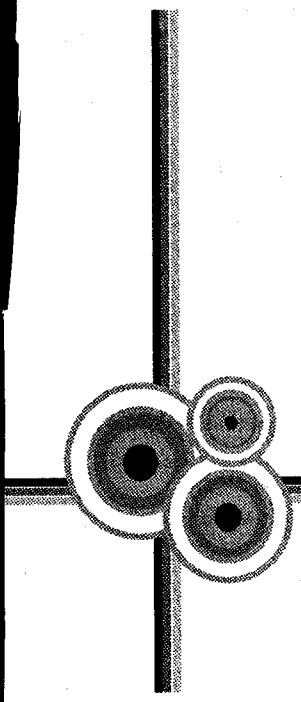
1. Overview of PL/SQL

PL/SQL stands for Procedural Language/ Structured Query Language. PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. 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.

Though SQL is the natural language of the DBA, it suffers from various inherent disadvantages, when used as a conventional programming language.

1. SQL does not have any procedural capabilities i.e. does not provide the programming techniques of condition checking, looping and branching that is vital for data testing before its permanent storage.

2. SQL statements are passed to the Oracle engine one at a time. Each time an SQL statement is executed, a call is made to the engine's resources. This adds to the traffic on the network, thereby decreasing the speed of data processing, especially in a multi-user environment.



Chapter 2

PL/SQL

1. Overview of PL/SQL

PL/SQL stands for Procedural Language/ Structured Query Language. PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. 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.

Though SQL is the natural language of the DBA, it suffers from various inherent disadvantages, when used as a conventional programming language.

1. SQL does not have any procedural capabilities i.e. does not provide the programming techniques of condition checking, looping and branching that is vital for data testing before its permanent storage.

2. SQL statements are passed to the Oracle engine one at a time. Each time an SQL statement is executed, a call is made to the engine's resources. This adds to the traffic on the network, thereby decreasing the speed of data processing, especially in a multi-user environment.

3. While processing a SQL sentence if an error occurs, the Oracle engine displays its own error messages. It has no facility for programmed handling of errors that arise during the manipulation of data.

Although SQL is a very powerful tool, its set of disadvantages prevents it from being a fully structured programming language. For a fully structured programming language, Oracle provides PL/SQL.

As the name suggests, PL/SQL is a superset of SQL. PL/SQL bridges the gap between database technology and procedural programming language.

Advantages of PL/SQL

1

Oct.2015 – *2M*
What is PL/SQL? Give advantages of PL/SQL.

3

Oct.11, 09, Apr.11, – *4M*
Explain advantages and disadvantages of PL/SQL.

1. PL/SQL is a development tool that not only supports SQL data manipulation but also provides facilities of conditional checking, branching and looping.
2. PL/SQL sends an entire block of SQL statements to the Oracle engine all in one go. Communication between the program block and the Oracle engine reduces considerably, reducing network traffic.

   Since the Oracle engine got the SQL statements as a single block, it processes this code much faster than if it got the code one sentence at a time. There is a definite improvement in the performance time of the Oracle engine. As an entire block of SQL code is passed to the Oracle engine at one time for execution, all changes made to the data in the table are done or undone, in one go.
3. PL/SQL also permits dealing with errors as required, and facilitates displaying user-friendly messages, when errors are encountered.
4. PL/SQL allows declaration and use of variables in blocks of code. These variables can be used to store intermediate results of a query for later processing, or calculate values and insert them into an Oracle table later. PL/SQL variables can be used anywhere, either in SQL statements or in PL/SQL blocks.
5. Via PL/SQL, all stores of calculations can be done quickly and efficiently without the use of Oracle engine. This considerably improves transaction performance.
6. Applications written in PL/SQL are portable to any computer hardware and operating system, where Oracle is operational. Hence, PL/SQL code written for a DOS version of Oracle will run on its Linux/UNIX version, without any modifications at all.

Use of PL/SQL

PL/SQL is used to access relational database from varoius environments which is fully block structured.

1 Apr.2012 – 2M
Define PL/SQL. What is use of PL/SQL?

1. **Better performance**: PL/SQL processes multiple SQL statements simultaneously which reduces network traffic.
2. **Error Handling**: PL/SQL handles errors and exception written in PL/SQL program.
3. It supports procedural and object oriented language.
4. Programmes written in PL/SQL are portable.
5. It has built in libraries and packages.
6. It has transaction processing language.

2. Data Types

3 Oct. 14,Apr.12,10 – 4M
Explain different Data Types in PL/SQL.

Information is transmitted between a PL/SQL program and the database through variables. Every variable has a specific type associated with it.

2 Oct.10,Apr.15 – 2M
What is the difference between % type and %rowtype?

The variable type can be

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.

Different Data types in PL/SQL are:

Data type	Explanation	Syntax	Example
Number	The most commonly used generic type is NUMBER. The default data type that can be declared in PL/SQL is number. Variables of type NUMBER can hold either an integer or a real number.	var_name number(p,s); p = precision s = scale	Employee_id number; Employee_sal number(8,3);
varchar2	To store variable length character strings with a maximum length of 4000 bytes.	var_name varchar2(10);	stud_name varchar2(10);

Char	To hold fixed length character strings.	var_name char(size);	stud_name char(10);
Date	To store date and time.	var_name date;	birth_date date;
Boolean	For storing TRUE, FALSE or NULL. Note that PL/SQL allows BOOLEAN variables, even though Oracle does not support BOOLEAN as a type for database columns.	var_name boolean;	stud_attendance boolean;
Rowed	Acts as a unique identifier for every row in the database and are stored internally as fixed length binary quantity.	var_name rowid;	emp_rowid rowed;
%type and % row type	Used to define variables in PL/SQL as per data type of columns, rows in table.	%type; %rowtype;	bname book.bookname%type; brec book%rowtype;

Number, char, varchar and date data types can have null values.

For example, we might declare:

```
DECLARE
   price NUMBER;
   Bookname VARCHAR(20);
```

NOT NULL: Causes creation of a variable or a constant that cannot be assigned a null value. If an attempt is made to assign the value NULL to a variable or a constant that has been assigned a NOT NULL constraint, Oracle senses the exception condition automatically and an internal error is returned.

The initial value of any variable, regardless of its type, is NULL. We can assign values to variables, using the ":=" operator. The assignment can occur either immediately after the type of the variable is declared, or anywhere in the executable portion of the program.

For example

```
DECLARE
   a NUMBER := 3;
BEGIN
   a:= a + 1;
END;
```

3. PL/SQL Block

4

Apr.2015 – *4M*
What is PL/SQL? Explain block of PL/SQL.

Apr.2012 – *2M*
What is structure of PL/SQL block?

Oct.2011 – *2M*
What is PL/SQL? Give PL/SQL block structure.

Oct.2010 – *4M*
What is PL/SQL? Give PL/SQL block structure and explain its details.

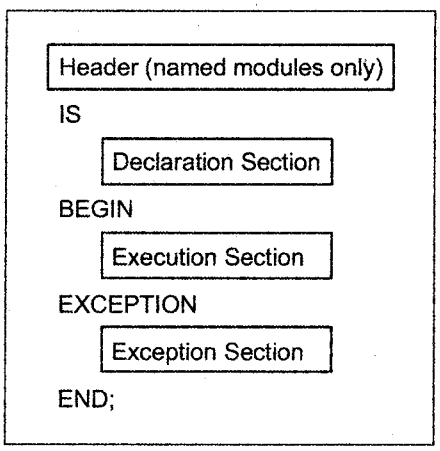
PL/SQL stands for Procedural Standard Query Language. The programming language used to access relational database from various environments is PL/SQL.

PL/SQL is a block-structured language. Each of the basic programming units that is written to build the application is (or should be) a logical unit of work. The PL/SQL block allows to reflect that logical structure in the physical design of the programs.

The block determines both the scope of identifiers (the area of code in which a reference to the identifier can be resolved) and the way in which exceptions are handled and propagated. A block may also contain nested sub-blocks of code, each with its own scope.

There is a common block structure to all the different types of modules. The block is broken up into four different sections, as follows:

1. **Header:** Relevant for named blocks only, the header determines the way that the named block or program must be called. The header includes the name, parameter list, and RETURN clause (for a function only).
2. **Declaration section:** The part of the block that declares variables, cursors, and sub-blocks that are referenced in the execution and exception sections. The declaration section is optional, but if there is one, it must come before the execution and exception sections.
3. **Execution section:** The part of the PL/SQL blocks containing the executable statements, the code that is executed by the PL/SQL run-time engine. The execution section contains the IF-THEN-ELSE, LOOPs, assignments, and calls to other PL/SQL blocks. Every block must have at least one executable statement in the execution section.
4. **Exception section:** The section that handles exceptions to normal processing (warnings and error conditions). This final section is optional. If it is included, control is transferred to this section when an error is encountered. This section then either handles the error or passes control to the block that is called the current block. Following diagram shows PL/SQL block structure for procedures and functions.



A PL/SQL block has the following structure:

```
DECLARE
  /* Declarative section: memory variables, types,
     and constants  */
BEGIN
   /* Executable section: procedural and SQL
       statements go here. */
   /* This is the only section of the block that
       is required. */
EXCEPTION
   /* Exception handling section: error-handling
       statements go here. */
END;
```

Only the executable section is required. The other sections are optional. The only SQL statements allowed in a PL/SQL program are SELECT, INSERT, UPDATE, DELETE and several other data manipulation statements plus some transaction control. However, the SELECT statement has a special form in which a single tuple is placed in variables. Data definition statements like CREATING, DROPPING, or ALTER is not allowed. The executable section also contains constructs such as assignments, branches, loops, procedure calls, and triggers. PL/SQL is not case sensitive. C style comments (/* ... */) may be used.

Comments

A comment can have two forms:

1. The comment line begins with a double hyphen (--). The entire line will be treated as a comment.
2. The comment line begins with a slash followed by an asterisk (/*) till the occurrence of an asterisk followed by a slash (*/). All lines within are treated as comment. This form of specifying comments can be used to span across multiple line.

3.1 Operators, Functions, Comparison, Numeric, Character, Date

The Character Set

The basic character set includes the following:

i. Uppercase alphabets {A-Z}.

ii. Lowercase alphabets {a-z}.

iii. Numerals {0-9}.

iv. Symbols ( ) + - * / < > = ! ; : . ' @ % , " # $ _ \ { } ? [ ]

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. The blank spaces have no effect on the PL/SQL block.

The ordinary symbols used in PL/SQL block are

( ) + - * / < > = ; % ' " [ ]

Compound symbols used in PL/SQL block are

<> != ~= ^= <= >= := ** .. || << >>

Literals

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

i. **Numeric Literals:** These can be either integers or floats. If a float is being represented, then the integer part must be separated from the float part by a period.

Example: 52, 5.35, 5g8, 45e-04, .2, 2.e7, +48, -9

ii. **String Literals:** These are represented by one or more legal characters and must be enclosed within single quotes. Writing it twice in a string literal can represent the single quote character. This is definitely not the same as a double quote.

Example:

```
'Hello world`,`Don`t go without saving your work`
```

iii. **Character Literals:** These are string literals consisting of single characters.

Example: '*', 'A','U'.

iv. **Logical (Boolean) Literals:** These are predetermined constants. The values that can be assigned to this data type are: TRUE, FALSE, and NULL.

Constant

In PL/SQL the keyword CONSTANT must be added to the variable name and a value assigned immediately.

```
<constantname> CONSTANT <datatype> <size>:=value;
```

Example:

```
PI CONSTANT number (5, 2): =3.14;
```

Operators

i. Arithmetic operators

+	Addition	*	Multiplication
-	Subtraction	* *	Exponentiation
/	Division	()	Enclosed operation

ii. Logical comparisons

PL/SQL supports the comparison between variables and constants in SQL and PL/SQL statements. These comparisons, often called Boolean expressions, generally consist of simple expression separated by

Relational operators <, >, =, >=, <=, <> that can be connected by

Logical operators AND, OR, NOT.

A Boolean expression will always evaluate to TRUE, FALSE or NULL.

3.2 Control Statement

2
Oct.11,09 – 4M
Explain different control structures used in PL/SQL with proper example.

The flow of control statement can be classified into the following categories:

i. Conditional Control
ii. Iterative Control
iii. Sequential Control

Conditional Control

PL/SQL allows the use of an IF statement to control the execution of a block of a code. In PL/SQL, the IF-THEN-ELSEIF-ELSE-END IF construct in code blocks allows specifying certain conditions under which a specific block of code should be executed.

Syntax

```
IF <condition> THEN
   <Statement_list>
ELSEIF <condition> THEN
   <Statement_list>
ELSE
   <Statement_list>
END IF;
```

Example: Write a 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;
/
```

At least one of the statements in <loop_body> should be an EXIT statement of the form

```
EXIT WHEN <condition>;
```

The loop breaks if <condition> is true.

Example: Write a program to print first 10 numbers. (1..10) (Using EXIT WHEN <condition>)

```
Declare
   J number:=0;
Begin
   Loop
   J:=J+1;
   dbms_output.put_line(J);
   EXIT when J>=10;
   End loop;
End;
/
```

Iterative Control

Simple loop

Syntax:

```
Loop
   <Statement_list>
   END LOOP;
```

Example:

a. Write a program to print first 10 numbers. (1..10)

```
Declare
      J number:=0;
Begin
     Loop
        J:=J+1;
        dbms_output.put_line(J);
     If (J>=10) then
        Exit;
        End if;
        End loop;
        End;
        /
```

b. Create a simple loop such that a message is displayed when a loop exceeds a particular value.

```
DECLARE
   J= number:=0;
   BEGIN
      LOOP
         J:=j+2;
      EXIT WHEN J>12;
   END LOOP;
   dbms_output.put_line('Loop exited as the value of J has reached '
          || to_char(J));
END;
```

Output: Loop exited as the value of j has reached 14. PL/SQL procedure successfully completed.

The WHILE loop

A WHILE loop can be formed with

1

Oct.2014 – 2M
Write syntax and example of while loop in PL/SQL.

Syntax:

```
WHILE <condition>
LOOP
    <Statement_ list>
END LOOP;
```

Examples

a. Write a program to print first 10 numbers. (1..10)(Using while)

```
Declare
  J number:=0;
Begin
  While J<=10 loop
    J:=J+1;
  dbms_output.put_line(J);
End loop;
End;
/
```

b. Write a PL/SQL block to calculate the area of a circle for a value of radius varying from 2 to 8. Store the radius and the corresponding values of calculated area in an empty table named area, consisting of two columns radius and area.

Table Name: area

RADIUS	AREA

Create table area as follows:

Create table area (RADIUS number (5), AREA number (14,2));

```
DECLARE
   pi constant number (4,2):=3.14;
   radius number (5);
   area number (14,2);
BEGIN
  radius:=2;
  While radius<=8
LOOP
  area:=pi*power(radius,2);
   insert into area values(radius, area);
   radius:=radius+1;
END LOOP;
END;
```

Output: Table Name: area

RADIUS	AREA
2	7.14
3	28.26
4	50.24
5	78.5
6	113.04
7	153.86
8	200.96

The FOR loop

A simple FOR loop can be formed with:

1

Apr.2015 – *2M*
Write syntax of for loop in PL/SQL with example.

Syntax:

```
FOR <variable> IN [REVERSE] <start>..<end>
LOOP
   <statement_list>
END LOOP;
```

The variable in the For Loop need not be declared. Also the increment value cannot be specified. The For Loop variable is always incremented by 1.

Examples:

a. Write a program to print first 10 numbers. (1..10)(Using for)

```
Declare
J number:=10;
Begin
   For J in 1..10 loop
      dbms_output.put_line(J);
End loop;
End;
/
```

b. Write a PL/SQL block of code reversing a number 5687 to 7865.

```
DECLARE
   input_no varchar(5):="5687";
    str_length number(2);
    reversed_no varchar(5);
BEGIN
   Str_length:=length (input_no);
     For cntr in reverse 1..str_length
     LOOP
        reversed_no:=reversed_no || substr(input_no,cntr,1);
              END LOOP;
dbms_output.put_line('The given number is' ||input_no);
dbms_output.put_line('The reversed number is' ||reversed_no);
END;
```

Output:

The given number is 5687
The reversed number is 7865

4. Exceptional Handling

1

Oct.2015 – *4M*
Write a note on exception handling in PL/SQL.

i. predefined
ii. no_data_found,
iii. cursor_already_open,
iv. dup_val_on_index,
v. storage_error,
vi. program_error,
vii. zero_divide,
viii. invalid_cursor,
ix. login_denied,
x. invalid_number,
xi. too_many_rows,
xii. DBMS_output,
xiii. user defined exceptions

1

Oct.2014 – *4M*
What is exception handling? Explain user defined exception with example.

1

Apr.2012 – *4M*
What is Exception Handling? Explain Predefined and User. Defined Exception with example.

Error Handling in PL/SQL

Every PL/SQL block of code encountered by the Oracle engine is accepted as a client. Hence the Oracle engine will make an attempt to execute every SQL sentence within the PL/SQL block. However while executing the SQL sentences anything can go wrong and the SQL sentence can fail.

When an SQL sentence fails the Oracle is the first to recognize this as an exception condition. The Oracle engine immediately tries to handle the exception condition and resolve it. This is done by raising a built-in exception handler.

An exception handler is nothing but a code block in memory that will attempt to resolve the current exception condition.

Oracle's named Exception Handlers

The Oracle engine has a set of pre-defined Oracle error handlers called named exceptions. Theses error handlers are referenced by their name. The following are some pre-defined named exception handlers.

There are two classes of exception:

i. **Predefined exception:** Oracle predefined errors, which are associated with specific error codes.

ii. **User defined exception:** Declared by the user and raised when specifically requested within a block. You can associate a user-defined exception with an error code if you wish.

There are two methods of defining exception by user.

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

Example:

```
Declare
   T_sal number(8,2);
   NEAGTIVE_SALARY EXCEPTION;
Begin
   Select sal into t_sal
   From emp
   Where empname="Mr.Mahesh";
If t_sal < 0 then
    Raise NEAGTIVE_SALARY;
Else
   Update emp set salary=10000
   Where empname="Mr.Mahesh";
   End if;
Commit;
Exception
   When no_data_found then
   dbms_output.put_line ('record not found`);
   When NEAGTIVE_SALARY then
   dbms_output.put_line ('salary is negative`);
End;
```

Here PL/SQL raises user_defined NEAGTIVE_SALARY exception.

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

```
Declare
   T_sal number(8,2);
Begin
   Select sal into t_sal
   From emp
```

```
    Where empname="Mr.Mahesh";
    Update emp set salary=10000
    Where empname="Mr.Mahesh";
Commit;
Exception
    When no_data_found then
    RAISE_APPLICATION_ERROR (-20005,`Record is not found`);
End;
```

Pre-determined internal PL/SQL exceptions

DUP_VAL_ON_INDEX	Raised when an insert or update attempts to create two rows with duplicate values in columns constrained by a unique index.
LOGIN_DENIED	Raised when an invalid username/password was used to log onto Oracle.
NO_DATA_FOUND	Raised when a select statement returns zero row.
PROGRAM_ERROR	Raised when PL/SQL has an internal problem.
TOO_MANY_ROWS	Raised when a select statements returns more than one row to be mapped into a set of variable.
NOT_LOGGED_ON	Raised when PL/SQL issues an Oracle call without being logged onto Oracle.
TIMEOUT_ON_RESOURCE	Raised when Oracle has been waiting to access a resource beyond the user-defined timeout limit.
VALUE_ERROR	Raised when the data type or data size is invalid.
Invalid_number	Raised when the data-type or data size or number is invalid.
Cursor_already_open	Raised when SQL cursor is open.
OTHERS	Stands for all other exceptions not explicitly named.

Displaying user Messages on the VDU screen

Programming tools require a method through which messages can be displayed on the VDU screen.

DBMS_OUTPUT is a package that includes a number of procedures and functions that accumulate information in a buffer so that it can be retrieved later. These functions can also be used to display messages.

PUT_LINE puts a piece of information in the package buffer followed by an end-of-line marker. It can also be used to display message. PUT_LINE expects a single parameter of character data type. If used to display message, it is the message string.

To display message, the SERVEROUTPUT should be set to ON. SERVEROUTPUT is a SQL* PLUS environment parameter that displays the information passed as a parameter to the PUT_LINE function.

Syntax

```
SET SERVEROUTPUT[ON/OFF]
```

5. Functions, Procedures

A procedure or function is a logically grouped set of SQL and PL/SQL statements that perform a specific task. A stored procedure or function is a named PL/SQL code block that has been compiled and stored one of the Oracle engines system tables.

To make a procedure or function dynamic either of them can be passed parameters before execution. A procedure or function can then change the way it works depending upon the parameters passed prior to its execution.

Procedures and functions are stored in the Oracle database. They can be invoked or called by any PL/SQL block that appears within an application. Before a procedure or function is stored, the Oracle engine parses and compiles the procedure or function.

The compilation process of procedures and functions does not display the errors. These errors can be viewed using the select statements;

```
SELECT * FROM USER_ERRORS;
```

Procedure and functions are made up of

i. A declarative part
ii. An executable part
iii. An optional exception-handling part

Creating Stored Procedures

2

Oct.2015 – 2M
Give syntax of stored procedure in PL/SQL.

Apr.2011 – 2M
Give proper syntax of procedure in PL/SQL.

Syntax:

```
CREATE OR REPLACE PROCEDURE
[Schema.]<ProcedureName>
   (<Argument> {IN, OUT, IN OUT} <Data type>, ...)
{IS, AS}
   <Variable declarations>;
   <Constant declarations>;
BEGIN
   <PL/SQL Subprogram body>;
EXCEPTION
   <Exception PL/SQL block>;
END;
```

Keywords and parameters

The keywords and parameters used for creating database procedures are explained below:

OR REPLACE	Recreates the procedure if it already exists. This option is used to change the definition of an existing procedure without dropping, recreating and re-granting object privileges previously granted on it. If a procedure is redefined the Oracle engine recompiles it.
Schema	Is the schema, which contains the procedure. The Oracle engine takes the default schema to be the current schema, if it is omitted.
Procedure	Is the name of the procedure to be created.
Argument	Is the name of an argument to the procedure. Parentheses can be omitted if no arguments are present.
IN	Indicates that the parameters will accept a value from the user.
OUT	Indicates that the parameters will return a value to the user.
IN OUT	Indicates that the parameters will either accept a value from the user or return a value to the user.
Data type	Is the data type of an argument? It supports any data type supported by PL/SQL.
PL/SQL Subprogram body	Is the definition of procedure consisting of PL/SQL statements.

Creating a Function

Syntax:

```
CREATE OR REPLACE FUNCTION [Schema.] <FunctionName>
   (<Argument> IN  <Data type>, …)
   RETURN <Data type> {IS, AS}
   <Variable declarations>;
   <Constant declarations>;
BEGIN
   <PL/SQL Subprogram body>;
EXCEPTION
   <Exception PL/SQL block>;
END;
```

1

Oct.2010 – 2M

Give proper syntax for function in PL/SQL.

Keywords and Parameters

The keywords and parameters used for creating database functions are explained below.

OR REPLACE	Recreates the function if it already exists. This option is used to change the definition of an existing function without dropping, recreating and re-granting object privileges previously granted on it. If a function is redefined the Oracle engine recompiles it.
Schema	Is the schema, which contains the function. The Oracle engine takes the default schema to be the current schema, if it is omitted.
Function	Is the name of the Function to be created.
Argument	Is the name of an argument to the function. Parentheses can be omitted if no arguments are present.
IN	Indicates that the parameters will accept a value from the user.
RETURN Data type	Is the data type of the function's return value. Because every function must return a value, this clause is required. It supports any data type supported by PL/SQL.
PL/SQL Subprogram body	Is the definition of function consisting of PL/SQL statements.

Deleting a stored procedure or function

A procedure or function can be deleted by using the following syntax:

Syntax:

```
DROP PROCEDURE <ProcedureName>;
```

Example:

```
DROP PROCEDURE proc_empsalcheck;
```

Output:

Procedure dropped

Syntax:

```
DROP FUNCTION <FunctionName>;
```

Example:

```
DROP FUNCTION f_empacctcheck;
```

Output:

Function dropped

Example of Function, Procedure

i. Write a script for the following

Accept the name of an actor from the user and for the specified actor list the details of all the movies the actor has acted in.

The function name is get_movie(), accepts name of the actor from the user and returns details of all the movies the actor has acted in it. The following are the relations,

```
create table movie
(
  m_no  integer,
  m_name text,
  year integer
);
create table actor
(
  act_no integer,
  act_name text
);
create table mov_act
(
  m_no integer references movie (mv_no) on delete cascade,
  act_no integer references actor (act_no)on delete cascade,
  rate integer
);
PL-SQL Block

create or replace procedure get_movie(act_name in varchar2)
as
  cursor c1 is select m_name,year
       from movie,actor,mov_act
       where movie.m_no=mov_act.m_no
       and actor.act_no = mov_act.act_no
       and actor.act_name = act_name;
  c c1%rowtype;
begin
  open c1;
  loop
        fetch c1 into c;
        exit when c1%notfound;
        dbms_output.put_line('*****   ACTOR INFORMATION  ***** ');
        dbms_output.put_line('ACTOR NAME:'||act_name);
        dbms_output.put_line('MOVIE NAME:'||c.m_name);
        dbms_output.put_line('RELEASE YEAR:'||c.year);
end loop;
close c1;
end;
```

6. Cursor

6.1 What is a Cursor?

The Oracle engine uses a work area for its internal processing in order to execute an SQL statement. This work is private to SQL's operations and is called a cursor.

The data that is stored in the cursor is called as **Active Data Set**. Conceptually, the size of the cursor in memory is the size required to hold the number of rows in the **Active Data Set**. The Oracle engine determines the actual size, built in memory management capabilities and the amount of RAM available; Oracle has a predefined area in main memory set aside, within which cursors are opened. Hence the cursor's size will be limited by the size of this pre-defined area.

The values retrieved from a table are held in a cursor opened in memory by the Oracle's engine. This data is then transferred to the client machine via network. In order to hold this data, a cursor is opened at the client end. If the number of rows returned by the Oracle engine is more than the area available in the cursor opened on the client, the cursor data and the retrieved data are swapped between the operating system's swap area and RAM.

6.2 Types of Cursors

There are two types of cursors:

1

Oct.2010 – *4M*
What is cursor? Explain two types of cursors.

1. **Implicit Cursor:** Cursors are classified depending on the circumstances under which they are opened. If the Oracle engine opened a cursor for its internal processing it is known as an Implicit Cursor.
2. **Explicit Cursor:** A cursor can also be opened for processing data through a PL/SQL block, on demand. Such a user-defined cursor is known as an Explicit Cursor.

General cursor attributes

When the Oracle engine creates an implicit or explicit cursor, cursor control variables are also created to control the execution of the cursor. There are a set of four system variables, which keeps track of the current status of a cursor. These cursor variables can be accessed and used in a PL/SQL code block.

Both implicit and explicit cursors have four attributes.

They are described below.

Attribute Name	Description
%ISOPEN	Returns TRUE if cursor is open, FALSE otherwise.
%FOUND	Returns TRUE if record was fetched successfully, FALSE otherwise.
%NOTFOUND	Returns TRUE if record was not fetched successfully, FALSE otherwise.
%ROWCOUNT	Returns number of records processed from the cursor.

8

Oct.2015, – *2M*
Define cursor. Enlist attributes of cursor.

Apr.15,12 – *4M*
What is Cursor? Explain different Attributes used in it.

Oct.2012 – *2M*
Which are different attributes of cursor?

Oct.2011 – *2M*
What is Cursor? List the Attributes of Cursor.

Apr.2011 – *4M*
What is cursor? List the attributes of cursor with suitable example.

Apr.10,Oct.09 – *2M*
What is Cursor? Which are the various attributes of Cursor?

1. **Implicit Cursor:** The Oracle engine implicitly opens a cursor on the server to process each SQL statement. Since the implicit cursor is opened and managed by the Oracle engine internally, the function of reversing an area in memory, populating this area with appropriate data, processing that data in the memory area, releasing the memory area when the processing is complete is taken care by the Oracle engine.

   The resultant data is then passed to the client machine via the network. A cursor is then opened in memory on the client machine to hold the rows returned by the Oracle engine. The number of rows held in the cursor on the client is managed by the client's operating system and it's swap area.

   Implicit cursor attributes can be used to access information about the status of the last insert, update, delete or single-row select statements. This can be done by preceding the implicit cursor attribute with the cursor name (i.e. SQL). The values of the cursor attributes always refer to the most recently executed SQL statements, wherever the statements appears. If an attribute value is to be saved for later use, it must be assigned to a (Boolean) memory variable.

Implicit cursor attributes are as follows:

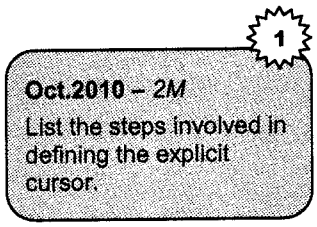
Attribute Name	Description
%ISOPEN	The Oracle engine automatically opens and closes the SQL statements that have been processed in case of implicit cursors. Thus the SQL%ISOPEN attribute of an implicit cursor cannot be referenced outside of its SQL statement. As a result, SQL%ISOPEN always evaluates to FALSE.
%FOUND	Evaluates to TRUE if an insert, update or delete affected one or more rows, or a single-row SELECT returned one or more rows. Otherwise it evaluates to FALSE. The syntax for accessing this attribute is SQL%FOUND.
%NOTFOUND	Is the logical opposite of %FOUND. It evaluates to TRUE, if an insert, update or delete affected no rows, or a single-row SELECT returns no rows. Otherwise, it evaluates to FALSE. The syntax for accessing this attribute is SQL%NOTFOUND.
%ROWCOUNT	Returns number of rows affected by an insert, update or delete or select into statement. The syntax for accessing this attribute is SQL%ROWCOUNT.

2. **Explicit Cursor:** When individual records in a table have to be processed inside a PL/SQL block a cursor is used. This cursor will be declared and mapped to an SQL query in the Declare Section of the PL/SQL block and used within its Executable Section. A cursor thus created and used is known as an **Explicit Cursor**.

Explicit cursor attributes are as follows:

Attribute Name	Description
%ISOPEN	Evaluates to TRUE, if an explicit cursor is open, or to FALSE, if it is closed. The syntax for accessing this attribute is CursorName%ISOPEN.
%FOUND	Evaluates to TRUE if the last fetch succeeded because a row was available; or to FALSE, if the last fetch failed because no more rows were available. The syntax for accessing this attribute is CursorName%FOUND.
%NOTFOUND	Is the logical opposite of %FOUND. It evaluates to TRUE, if the last fetch has failed because no more rows were available; or to FALSE, if the last fetch returns a row. The syntax for accessing this attribute is CursorName %NOTFOUND.
%ROWCOUNT	Returns number of rows fetched from the active set. It is set to zero when the cursor is opened. The syntax for accessing this attribute is CursorName%ROWCOUNT.

Explicit cursor Management



The steps involved in using an explicit cursor and manipulating data in its active set are,

i. Declare a cursor mapped to a SQL select statement that retrieves data for processing.

ii. Open the cursor.

iii. Fetch data from the cursor one row at a time into memory variables.

iv. Process the data held in the memory variables as required using a loop.

v. Exit from the loop after processing is complete.

vi. Close the cursor.

6.3 Cursor Declaration

A cursor is defined in the declarative part of a PL/SQL block. This is done by naming the cursor and mapping it to a query. When a cursor is declared, the Oracle engine is informed that a cursor of the said name needs to be opened. The declaration is only intimation. There is no memory allocation at this point in time. The three commands used to control the cursor subsequently are **open, fetch and close.**

The functionality of open, fetch and close commands.

Initialization of a cursor takes place via the open statement, this:

i. Defines a private SQL area named after the cursor name.

ii. Executes a query associated with the cursor.

iii. Retrieves table data and populates the named private SQL area in memory i.e. creates the Active Data Set.

iv. Sets the cursor row pointer in the Active Data Set to the **first** record.

A **fetch** statement then moves the data held in the Active Data Set into memory variables. Data held in the memory variables can be processed as desired.

A fetch statement is placed inside a loop...end loop construct, which causes the data to be fetched into the memory variables and processed until all the rows in the Active Data Set are processed. The fetch loop then exits. The exiting of the fetch loop is user controlled.

After the fetch loop exits, the cursor must be closed with the close statement. This will release the memory occupied by the cursor and its Active Data Set. A PL/SQL block is necessary to declare a cursor and create an Active Data Set. The cursor name is used to reference the Active Data Set held within the cursor.

Syntax:

```
CURSOR CursorName IS <SELECT statement>;
```

Opening a Cursor

Opening a cursor executes the query and creates the active set that contains all rows, which meet the query search criteria. An open statement retrieves record from a database table and places the records in the cursor (i.e., named private SQL area in memory). A cursor is opened in the server memory.

Syntax:

```
OPEN CursorName;
```

Fetching a record from the cursor

The fetch statement retrieves the rows from the active set opened in the server into memory variables declared in the PL/SQL code block on the client one row at a time. The memory variables are opened on the client machine. Each time a fetch is executed, the cursor pointer is advanced to the next row in the Active Data Set.

A standard loop structure (loop-End Loop) is used to fetch records from the cursor into memory variables one row at a time.

Syntax:

```
FETCH CursorName INTO Variable1, Varable2, ...
```

There must be a memory variable for each column value of the Active Data Set. Data types must match. These variables will be declared in the **DECLARE** section of the PL/SQL block.

Closing a cursor

The close statement disables the cursor and the active set becomes undefined. This will release the memory occupied by the cursor and its data set both on the client and on the server.

Syntax:

```
CLOSE CursorName;
```

Once a cursor is closed, the reopen statement causes the cursor to be opened.

6.3 Cursor For Loops

Another technique commonly used to control the Loop...End Loop within a PL/SQL block is the **FOR** variable **IN** value construct. This is an example of a machine defined loop exit i.e., when all the values in the FOR construct are exhausted looping stops.

Syntax:

```
FOR memory variable IN CursorName
```

Here, the verb FOR automatically creates the memory variable of the %rowtype. Each record in the opened cursor becomes a value for the memory variable of the %rowtype.

The FOR verb ensures that a row from the cursor is loaded in the declared memory variable and the loop executes once. This goes on until all the rows of the cursor have been loaded into the memory variable. After this loop stops.

A cursor for loop automatically does the following:

1. Implicitly declares its loop index as a %rowtype record.
2. Opens a cursor.
3. Fetches a row from the cursor for each loop iteration.
4. Closes the cursor when all rows have been processed.

A cursor can be closed even when an exit or a goto statement is used to leave the loop prematurely, or if an exception is raised inside the loop.

Example:

```
Declare cursor c_actname is select aname, rate, mvno from actor;
Begin
For y in c_actname
Loop
Dbms_output.put_line(y.mvno ||`  '||y.aname||`  '||y.rate);
End loop;
End;
```

6.4 Parameterized Cursors

2

Oct.12, Apr.11 – *4M*
What is Parameterized Cursor? Explain it with example.

Till now, all the cursors that have been declared and used fetched a pre-determined set of records. Records, which satisfy conditions, set in the WHERE clause of the SELECT statement mapped to the cursor. In other words, the criterion on which the Active Data Set is determined is hard coded and never changes.

Commercial applications required that the query, which, defines the cursor, be generic and the data that is retrieved from the table be allowed to change according to need.

Oracle recognizes this and permits the creation of parameterized cursors for use. The contents of a parameterized cursor will constantly change depending upon the value passed to its parameter.

Since the cursor accepts user-defined values into its parameters, thus changing the result set extracted, it is called as **parameterized cursor.**

Declaring a Parameterized Cursor

Syntax:

```
CURSOR CursorName(VariableName Datatype)IS <SELECT statement…>
```

Opening a Parameterized Cursor and Passing Values to the Cursor

Syntax:

```
OPEN CursorName(Value/Variable/Expression)
```

The scope of cursor parameters is local to that cursor, which means that they can be referenced only within the query declared in the cursor declaration. Each parameter in the declaration must have a corresponding value in the **open** statement.

Example of Parameterized Cursor

Here is the *example* of Parameterized Cursor. To print department wise list of employees, we passed department number as a parameter to a cursor. Following are the two relations,

```
create table dept
(
   d_no integer,
   d_name text,
);
create table emp
(
```

```
  emp_no integer,
  e_name text,
  basic_salary  float,
  d_no integer references dept(d_no)on delete cascade
);
```

PL-SQL Block

```
declare
  cursor c1 is select d_no from dept;
  cursor c2(dno number) is select d_name,e_name from emp,dept
        where dept.d_no=emp.d_no and emp.d_no = dno;
        group by d_name,e_name;
  c c1%rowtype;
  d c2%rowtype;
begin
  open c1;
  loop
       fetch c1 into c;
       exit when c1%notfound;
       open c2(c.d_no);
       loop
              fetch c2 into d;
              exit when not found;
              dbms_output.put_line(d.d_name||'     '||d.e_name);
       end loop;
       close c2;
       end loop;
  close c1;
end;
```

7. Database Triggers

Database triggers are database objects created via the SQL* Plus tool on the client and stored on the server in the Oracle engines system table.

These database objects consists of the following distinct sections

1. A named database event
2. A PL/SQL block that will execute when the event occurs

The Oracle engine allows the definition of procedures that are implicitly executed (i.e. executed by the Oracle engine itself), when an insert, update or delete is issued against a table from SQL* plus or through an application. These procedures are called database triggers. The major issues that make these triggers standalone are that, they are fired implicitly (i.e. internally) by the Oracle engine itself and not explicitly i.e. called by the user.

4

Oct.2015 – *4M*
What is Trigger? Explain types of trigger in detail.

Apr.2015 – *4M*
What is trigger? Explain trigger with proper syntax and example.

Apr.2012 – *2M*
What is Trigger? What are the types of Trigger?

Apr.2010 – *4M*
What is Trigger? Explain any two types of triggers.

7.1 Types of Triggers

Following are the types of the database triggers:

i. Row triggers
ii. Statement triggers
iii. Before triggers
iv. After triggers
v. Combinations triggers
vi. Before statement trigger
vii. Before row trigger
viii. After statement trigger
ix. After row trigger

4

Oct.12,Apr.10,09 – *2M*
Give proper syntax of trigger.

Apr.2011 – *2M*
List modes of trigger and its syntax.

Syntax for Creating a Trigger

```
CREATE OR REPLACE TRIGGER[Schema.]<TriggerName>
   {BEFORE, AFTER}
   {DELETE, INSERT, UPDATE[OF Column,…]}
ON [Schema.] <TableName>
   [REFERENCING {OLD AS old,NEW AS new}]
   [FOR EACH ROW[WHEN Condition]]
DECLARE
   <Variable declarations>;
   <Constant declarations>;
BEGIN
   <PL/SQL Subprogram body>;
EXCEPTION
   <Exception PL/SQL block>;
END;
```

Keywords and Parameters

The keywords and the parameters used for creating database triggers are explained below.

OR REPLACE	Recreates the trigger if it already exists. This option can be used to change the definition of an existing trigger without requiring the user's to drop the trigger first.
Schema	Is the schema, which contains the trigger. If the schema is omitted, the Oracle engine creates the trigger in the user's own schema.
Triggername	Is the name of the trigger to be created.
BEFORE	Indicates that the Oracle engine fires the trigger before executing the triggering statement.
AFTER	Indicates that the Oracle engine fires the trigger after executing the triggering statement.
DELETE	Indicates that the Oracle engine fires the trigger whenever a DELETE statement removes a row from the table.
INSERT	Indicates that the Oracle engine fires the trigger whenever an INSERT statement adds a row to table.
UPDATE	Indicates that the Oracle engine fires the trigger whenever an UPDATE statement changes a value in one of the columns specified in the OF clause. If the OF clause is omitted, the Oracle engine fires the trigger whenever an UPDATE statement changes a value in any column of the table.
ON	Specifies the schema and name of the table, upon which the trigger is to be created. If schema is omitted, the Oracle engine assumes the table is in the users own schema. A trigger cannot be created on a table in the schema SYS.
REFERENCING	Specifies correlation names. Correlation names can be used in the PL/SQL block and WHEN clause of a row trigger to refer specifically to old and new values of the current row. The default correlation names are OLD and NEW. If the row trigger is associated with a table named OLD or NEW, this clause can be used to specify different correlation names to avoid confusion between table name and the correlation name.
FOR EACH ROW	Designates the trigger to be a row trigger. The Oracle engine fires a row trigger once for each row that is affected by the triggering statement and meets the optional trigger constraint defined in the WHEN clause. If this clause is omitted the trigger is a statement trigger.
WHEN	Specifics the trigger restriction. The trigger restriction contains a SQL condition that must be satisfied for the Oracle engine to fire the trigger. This condition must contain correlation names and cannot contain a query. Trigger restriction can be specified only for the row trigger. The Oracle engine evaluates this condition for each row affected by the triggering statement.
PL/SQL BLOCK	Is the PL/SQL block that the Oracle engine executes when the trigger is fired.

The PL/SQL block cannot contain transaction control SQL statements (COMMIT, ROLLBACK, and SAVEPOINT)

Deleting a Trigger

Syntax:

```
DROP TRIGGER <TriggerName>;
```

where, TriggerName is the name of the trigger to be dropped.

Example of Trigger

A trigger that will take care of the constraint that movie released after 2005 be entered in the movie table. Following are the relations,

```
create table movie
(
  mv_no   integer,
  mv_name text,
  rel_year integer
);
create table actor
(
  act_no integer,
  act_name text
);
create table ma
(
  mv_no integer references movie(mv_no) on delete cascade,
  act_no integer references actor(act_no) on delete cascade
);
```

PL-SQL Block

```
create or replace trigger t_mov_2005
before insert or update on movie
for each row

begin
  if(:new.relyear < 2005) then
    raise_application_error(-2001,' YEAR SHOULD BE > 2005');
  end if;
end;
```

8. Oracle Packages

1

Oct.2010 – 4M
What is the package in PL/SQL? Explain with example.

A package is an Oracle object, which holds objects within it. Objects commonly held within a package are procedures, functions, variables, constants, cursors and exceptions. The tool used to create a package is SQL* plus. It is a way of creating generic, encapsulated, re-usable code.

Packages can contain PL/SQL blocks of code, which have been written to perform some process entirely on their own. These PL/SQL blocks of code do not require any kind of input from other PL/SQL blocks codes. These are the packages standalone subprograms.

8.1 Components of an Oracle package

A package has usually two components, a **specification** and a **body**. A package's specification declares the types (variables of the record type), memory variables, constants, exceptions, cursors and subprograms that are available for use.

Package Specification: The package specification contains:

1. Name of the package.
2. Names of the data types of any arguments.
3. This declaration is local to the database and global to the package.

This means that procedures, functions, variables, constants, cursors and exceptions and other objects declared in a package are accessible from anywhere in the package. Therefore, all the information a package needs to execute a stored subprogram is contained in the package specifications itself.

Syntax:

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

Example:

```
CREATE OR REPLACE PACKAGE mypack AS
   PROCEDURE myproc (p_number in number);
   FUNCTION myfunction (f_number in number);
END mypack;
```

Advantages of Packages

1. Packages enable the organization of commercial applications into efficient modules. Each package is easily understood and the interfaces between packages are simple, clear and well defined.
2. Packages allow granting privileges efficiently.
3. A packages public variables and cursors persist for the duration of the session. Therefore all cursors and procedures that execute in this environment can share them.
4. Packages enable the overloading of procedures and functions when required.
5. Packages improve performance by loading multiple objects into memory at once. Therefore, subsequent calls to related subprograms in the package require no I/O.
6. Packages promote code reuse through the use of libraries that contain stored procedures and functions, thereby reducing redundant coding.

Solved Examples

Following are the solved examples of PL/SQL Block using all types of Control statements, Function, Cursor, Parameterized cursor, Trigger using all types of trigger etc.

1.

```
create table doctor
(
   doc_no integer,
   doc_name text,
   addresstext,
   city text,
   area text,
);
create table hosp
(
   hosp_no integer,
   hosp_name text,
   hosp_city text,
);
create table doc_hosp
(
   doc_no integer references doctor(doc_no) on delete cascade,
   hosp_no integer references hosp(hosp_no) on delete cascade,
);
```

i. Script to list the names of doctors who visit every hospital located in the city where they do not live.

PL/SQL BLOCK

```
declare
  cursor c1 is select doc_no,doc_name,city
            from doctor;
  cursor c2(dcity varchar2) is  select hosp_no
            from hosp;
            where hosp_city <> dcity;
  cursor c3(dno number,hno number) is select doc_no,hosp_no
            from doc_hosp;
            where doc_no = dno;
            and hosp_no = hno;
       c c1%rowtype;
       d c2%rowtype;
       e c3%rowtype;
begin
  open c1;
  loop
  fetch c1 into c;
  exit when c1%notfound;
       open c2(c.city);
       loop
        fetch c2 into d;
  exit when c2%notfound;
     open c3(c.doc_no,d.hosp_no) ;
      loop
        fetch c3 into e;
  exit when c3%notfound;
        dbms_output.put_line('DOCTOR = '||e.doc_name);
     end loop;
      close c3;
   end loop;
    close c2;
  end loop;
  close c1;
end;
```

ii. A script of cursor to print the list showing the doctor wise list of hospitals.

PL/SQL BLOCK

```
declare
   cursor c1 is select doc_name,hosp_name
        from doctor,hospital,doc_hos where
        doc_hos.doc_no=doctor.doc_no
        and doc_hos.hosp_no=hospital.hosp_no
        group by doc_name,hosp_name;
   c c1%rowtype;
begin
   dbms_output.put_line('DocName||'    '||HospName);
   open c1;
   loop
   fetch c1 into c;
   exit when c1%notfound;
          dbms_output.put_line(c.doc_name||'    '||c.hosp_name);
   end loop;
   close c1;
end;
```

2.

```
create table item
(
   item_no integer,
   item_name text,
   qty integer
);
create table supp
(
   supp_nointeger,
   supp_name text,
   addresstext,
   city text,
   phno integer
);
create table item_supp
(
   item_nointeger references item(item_no) on delete cascade,
```

```
  supp_no integer references supp(supp_no)on delete cascade,
  rate integer,
  discount integer
);
```

i. Define a trigger before updation on discount field, if the difference in the old discount and new discount be entered is > 5% raise an exception and display corresponding message.

PL/SQL Block

```
create or replace trigger t_itemsup
  before update of discount of it_sup
  for each row
declare
  olddisc it_sup.discount%type;
  newdisc it_sup.discount%type;
  diff number;
begin
  olddisc:=:old.discount;
  newdisc:=:new.discount;
  diff:=newdisc-olddisc;
  if(diff > olddisc/20) then
    dbms_output.put_line('Difference bet new and old discount should be
                          less than 5%');
  end if;
end;
```

ii. Write a script to list the suppliers who live in same city but supply different set of items.

PL/SQL Block

```
declare
  cursor c1 is select supp.supp_no,item.item_no,supp_name,city
       from supp,item,item_supp
       where supp.supp_no=item_supp.supp_no
       and item.item_no=item_supp.item_no;
  cursor c2(sno number) is select supp_name,city,item.item_no
       from supp,item,item_supp
       where supp.supp_no<>sno
```

```
        and supp.supp_no=item_supp.supp_no
        and item.item_no=item_supp.item_no;
   c c1%rowtype;
   d c2%rowtype;
begin
   open c1;
   loop
     fetch c1 into c;
     exit when c1%notfound;
     open c2(c.supp_no);
     loop
       fetch c2 into d;
       exit when c2%notfound;
     if(c.city = d.city) then
         if(c.item_no <> d.item_no) then
           dbms_output.put_line(c.supp_name||d.supp_name||c.city);
        end if;
    end if;
    end loop;
    close c2;
   end loop;
   close c1;
end;
```

3.

```
create table dept
(
  dept_nointeger,
  dept_name text,
   location text
);
create table emp
(
  emp_no integer,
  emp_name text,
  sex text,
  joining_date date,
  designation text,
  salary float,
  dept_nointeger references dept(dept_no) on delete cascade
);
```

i. Write a script to list the names of all employees who are men and earning maximum salary in their department.

PL/SQL Block

```
declare
  cursor c1 is select dept_no,dept_name from dept;
  cursor c2(dno number) is select emp_name from emp,dept
       where emp.salary in (select max(salary),emp.emp_name
       from emp,dept where emp.dept_no=dno
       and dept.dept_no=emp.dept_no and sex='m'
       group by dept.dept_no);
  c c1%rowtype;
  d c2%rowtype;
begin
  open c1;
  loop
  fetch c1 into c;
  exit when c1%notfound;
  open c2(c.dept_no);
  loop
  fetch c2 into d;
  exit when c2%notfound;
    dbms_output.put_line(c.dept_name||d.emp_name);
  end loop;
  close c2;
  end loop;
  close c1;
end;
```

ii. Write a script to give raise in salary by 5% for all the employees earning less than 10000 and 9% for all employees earning more than or equal to 10000. Also print total numbers of employees in each case.

PL/SQL Block

```
declare
  cursor c2 is select emp_no,salary from emp,dept
       where dept.dept_no=emp.dept_no;
  c c2%rowtype;
  cnt1 number;
  cnt2 number;
begin
```

```
   cnt1:=0;
   cnt2:=0;
   open c2;
   loop
   fetch c2 into c;
   exit when c2%notfound;
   if(c.salary < 10000) then
        update emp set salary = salary + salary*0.05
         where emp.emp_no = c.emp_no;
   cnt1:= cnt1+1;
   end if;
   if(c.salary >= 10000) then
          update emp set salary = salary + salary * 0.09
            where emp.emp_no = c.emp_no;
   cnt2:=cnt2+1;
        end if;
   end loop;
   close c2;
      dbms_output.put_line('No of employees getting 5% increase= '||cnt1);
     dbms_output.put_line('No of employees getting 9% increase= '||cnt2);
end;
```

4.

```
create table emp1
(
   emp_no integer,
   emp_name  text,
   salary float,
   comm integer,
   mgr_no integer,
  dept_no integer references dept(dept_no) on delete cascade
);
create table dept1
(
   dept_nointeger,
   dept_name text,
   location text
);
```

i. Write a script to transfer all employees of dept. "C" of location "CB" earning the commission of 50% of their salary to dept "B". Also print the total number of employees of dept. "C" transferred to dept "B".

PL/SQL Block

```
declare
   cursor c1 is select dept_no from dept1 where dept_name='B';
   cursor c2 is select emp_no,salary,dept_name,location,comm from
      emp1,dept1 where dept1.dept_no=emp1.dept_no;
   c c1%rowtype;
   r c2%rowtype;
   cnt number;
begin
   cnt:= 0;
   open c2;
   loop
   fetch c2 into r;
   exit when c2%notfound;
   open c1;
   loop
   fetch c1 into c;
   exit when c1%notfound;
      if(r.location ='CB' and r.dept_name='C') then
       if(r.comm >= (r.salary/2)) then
          update emp1 set dept_no = c.dept_no
           where emp1.emp_no = r.emp_no;
           cnt:=cnt+1;
       end if;
     end if;
   end loop;
   close c1;
   end loop;
   close c2;
  dbms_output.put_line('Total no of emp of dept C transferred to dept CB
     are='||cnt);
end;
```

ii. Write a script for the following: give the names of all those locations, which has total of at least 5 depts. in it. Out of which at least 3 depts are spending approximate Rs. 50000/- as salary of the employee.

PL/SQL Block

```
declare
   cursor c1 is select location from emp1,dept1
        where dept1.dept_no=emp1.dept_no group by location
        having count(dept1.dept_no)>=5;
   cursor c2(loc varchar2) is select location from emp1,dept1
          where dept1.location=loc and
          dept1.dept_no=emp1.dept_no group by location
```

```
          having count(dept1.dept_no)>= 3 and
                sum(salary) >= 50000;
   c c1%rowtype;
   d c2%rowtype;
begin
   open c1;
   loop
   fetch c1 into c;
   exit when c1%notfound;
         open c2(c.location);
         loop
            fetch c2 into d;
            exit when c2%notfound;
            dbms_output.put_line('Location = '||d.location);
         end loop;
         close c2;
   end loop;
   close c1;
end;
```

5.

```
create table company
(
   c_no integer,
   c_name text,
   c_addr text,
   c_city text,
   c_share integer
);
create table person
(
   p_no integer,
   p_name text,
   p_addr text,
   p_city text,
   p_phone_no  integer
);
create table comp_per
(
   c_no integer references company(c_no) on delete cascade,
   p_no integer references person(p_no) on delete cascade,
   no_of_shares integer
);
```

i. Write a trigger, which gets activated when company tuple is updated. It should delete all the related tuples when share value of company becomes < Rs.10/-.

PL/SQL Block

```
create or replace trigger t_comp
after insert or update on company
for each row
declare
   c company.c_no%type;
   v company.c_share%type;
begin
    select c_no,c_share into c,v from company
   where c_share < 10;
   delete from company where c_no=c;
    delete from comp_per where c_no=c;
end;'
```

ii. Write a procedure/function, which will take company name as parameter and will find names of persons who are shareholders of the company.

PL/SQL Block

```
create or replace procedure get_rec(cname in varchar2)
as
   cursor c1 is select company.c_no,c_name,person.p_no,p_name,c_share
        from company,person,comp_per
        where comp_per.p_no=person.p_no
        and comp_per.c_no=company.c_no;
   c c1%rowtype;
begin
   open c1;
   loop
         fetch c1 into c;
         exit when c1%notfound;
         if(c.c_name = cname) then
             dbms_output.put_line('COMPANY = '||c.c_name);
             dbms_output.put_line('SHARE HOLDER = '||c.p_name);
         end if;
```

```
   end loop;
   close c1;
end;
```

6.

```
create table dept
(
   d_no integer,
   d_name text
);
create table employee
(
   e_no integer,
   e_name text,
   basic_salary  float,
   d_no integer references dept(d_no) on delete cascade
);
```

i. Write a script to calculate the salary of each employee as follows
HRA- Rs 2000 if basic_sal <= Rs 8000, Rs 2500 otherwise,
DA - 35% of basic_sal
CA -Rs 500 if basic_sal < Rs 6000
Rs 800 if basic_sal > Rs 6000 and <=Rs 9000
Rs 1200 otherwise
PF -11% of basic_sal
PT- Rs 200 if basic_sal <= Rs. 8000
Rs 250 otherwise
Net salary is calculated as basic_sal+ HRA+ DA+ CA- Pf- PT

PL/SQL Block

```
declare
   cursor c1 is select * from emp;
   hra number;
   da number(20,4);
   pf number(20,4);
   ca number;
   pt number;
   total_sal number;
   c c1%rowtype;
begin
   open c1;
   loop
         fetch c1 into c;
```

```
        exit when c1%notfound;
        total_sal=0;
        if(c.salary <= 8000) then
          hra:=2000;
          pt :=200;
        else
          hra:=2500;
          pt :=250;
        end if;
         if(c.salary < 6000) then
              ca:=500;
         end if;
         if(c.salary > 6000)then
               if(c.salary <= 9000) then
                     ca:= 800;
                else
                     ca:= 1200;
                end if;
         end if;
        pf := 0.11 * c.salary;
        da := 0.35 * c.salary;
   total_sal:=c.salary + hra + da + ca - pf - pt;
   dbms_output.put_line('EMPLOYEE : '||c.e_name);
   dbms_output.put_line(' HRA= '||hra);
   dbms_output.put_line('DA = '||da);
   dbms_output.put_line('CA = '||ca);
   dbms_output.put_line('PF = '||pf);
   dbms_output.put_line('PT = '||pt);
   dbms_output.put_line('TOTAL SALARY IS '||total_sal);
   end loop;
   close c1;
end;
```

ii. Write a script using parameterized cursor to print department wise list of employees, pass department number as a parameter to a cursor.

PL/SQL Block

```
declare
   cursor c1 is select d_no from dept;
   cursor c2(dno number) is select d_name,e_name from emp,dept
          where dept.d_no=emp.d_no and emp.d_no = dno;
          group by d_name,e_name;
   c c1%rowtype;
   d c2%rowtype;
```

```
begin
   open c1;
   loop
         fetch c1 into c;
         exit when c1%notfound;
         open c2(c.d_no);
         loop
            fetch c2 into d;
            exit when not found;
            dbms_output.put_line(d.d_name||''||d.e_name);
         end loop;
         close c2;
         end loop;
   close c1;
end;
```

7. Accept the deptname and print the no of employees working in that department.

```
Declare
   v_deptname emp.deptname%type;
   v_count number;
Begin
   v_deptname:=&v_deptname;
   select count(*) into v_count
   from emp
   where deptname=v_deptname;
   dbms_output.put_line('No. of employees working in` | | v_deptname | |
   'are`||v_count);
End;
```

8. Accept the deptname and print the department no and location of that department.

```
Declare
   v_deptname dept.deptname%type;
   v_deptno dept.deptno%type;
   v_deptloc dept.deptloc%type;
Begin
   v_deptname:=&v_deptname;
   select dno,deptloc into v_deptno,v_deptloc
   from dept
```

```
  where deptname =v_deptname;
  dbms_output.put_line('Department no is`| |v_deptno | | 'and location
   is`| |v_deptloc);
End;
```

9. Accept empno and Print the name of employee with his salary.

```
Declare
  v_empno emp.empno%type;
  v_empname emp.empname%type;
  v_empsal emp.empsal%type;
Begin
  v_empno:=&v_empno;
  select empname,empsal into v_empname,v_empsal
  from emp
  where empno=v_empno;
  dbms_output.put_line('Name of employee`| |v_empname 'and salary is`|
  |v_empsal);
End;
```

10. Print the name of employee and salary, having minimum salary.

```
Declare
  v_name emp.empname%type;
  v_sal emp.empsal%type;
Begin
  Select empname,empsal into v_name,v_sal
  from emp
    where sal=(select min(sal) from emp);
    dbms_output.put_line(v_name | | 'is having min. salary =` | |v_sal);
End;
```

11. Accept employee number and print date of joined.

```
Declare
  v_eno emp.eno%type;
  v_date_of_joined date;
Begin
  v_eno:=&v_eno;
  select date_of_joined into v_date_of_joined
  from emp
    where eno=v_eno;
    dbms_output.put_line('Date of joined is | |v_date_of_joined);
End;
```

12. Accept salary and print number of employees having salary greater than or equal to accepted salary.

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

13. List all the employees having salary less than 3000 Rs.

```
Declare
   v_ename emp.ename%type;
   v_esal emp.esal%type;
Begin
   Select ename  into v_ename
   from emp
      where esal<3000;
      dbms_output.put_line('The employees having salary less than Rs.3000
       are` | |v_ename);
End;
```

14. Program to find smallest number of two numbers.

```
Declare
   N1 number;
   N2 number;
Begin
   N1:=&n1;
   N2 :=&n2;
   If(N1<N2) then
      Dbms_output.put_line('N1 is smallest`);
   Else
      Dbms_output.put_line ('N2 is smallest`);
   End if;
End;
```

15. Check whether the salary of Gaurav is 35000 or not.

```
Declare
   Gsal emp.sal%type;
Begin
   Select sal into Gsal
```

```
   From emp
     Where empname= 'Gaurav`;
   If(Gsal >35000) then
     Dbms_output.put_line('Salary of Gaurav is 35000 `);
   Else
     Dbms_output.put_line('Salary of Gaurav is not 35000 `);
   End if;
End;
```

---

16. Program for minimum of three numbers.

```
Declare
   a number;
   b number;
   c number;
Begin
   Dbms_output.put_line('Enter value of a:`);
   a:= &a;
   Dbms_output.put_line('Enter value of b:`);
   b:=&b;
   Dbms_output.put_line('Enter value of c:`);
   c:=&c;
   If(a<b) and (a<c) then
     Dbms_output.put_line('a is minimum`);
   Elseif(b<a) and (b<c) then
     Dbms_output.put_line('b is minimum`);
   Else
     Dbms_output.put_line('c is minimum`);
   End if;
End;
```

---

17. Print the numbers between 1 to 20.

```
Declare
   n number:=0;
Begin
   Loop
     n=n+1;
     dbms_put.put_line(n);
     if(n<20) then
          exit;
     end if;
     end loop;
End;
```

18. Print all the even numbers 1 to 15

```
Declare
   i number:=0;
Begin
   While i<=15 loop
      i=i+1;
      if (i%2==0)
         dbms_ouput.put_line('The even nos are:`| | i);
      End if;
   End loop;
End;
```

19. Program to find sum of 1st 10 numbers.

```
Declare
   num number:=1;
   sum number:=0;
Begin
   For num in 1…10 loop
      sum=sum+num;
      dbms_output.put_line('The sum of 1st 10 nos. is`: | | sum);
   End loop;
End;
```

20. Print the sum of odd numbers between 1 to 25.

```
Declare
   num number:=1;
   sum number:=0;
Begin
   For num in 1 …. 25 loop
      If(num%2!=0)
         Sum=sum+num;
      End if;
     Dbms_output.put_line ('The sum of odd numbers between 1 to 25 is:` | |
   sum);
   End loop;
End;
```

21. Accept a no and check it is even or odd.

```
Declare
   n number;
Begin
```

```
  Dbms_ouput.put_line('Enter the number:`);
  n:=&n;
  if( n%2==0) then
    dbms_output.put_line('Number is even`);
  else
    dbms_output.put_line('Number is odd`);
  end if;
end;
```

---

22. Accept number and print its square and cube.

```
Declare
  num number;
Begin
  Dbms_ouput.put_line('Enter a number:`);
  num:=&num;
  Dbms_ouput.put_line('Square of a number:`| |num*num);
  Dbms_ouput.put_line('Cube of  a number:`| | num*num*num);
End;
```

---

23. Accept a no and find its factorial.

```
Declare
  num number;
  f number:=1;
Begin
  Dbms_ouput.put_line('Enter a number:`);
  num:=&num;
  while n>0 loop
  f:=f*num;
  num--;
  end loop;
  dbms_output.put_line('Factorial of number is:`| |f);
end;
```

---

24. Accept a string and reverse that string.

```
Declare
  str varchar(30);
  ch varchar(1);
  len number;
Begin
  str:='&str`;
  len:=length(str);
```

```
   While len>0 loop
   ch:=substr(str, 1 , 1);
   Dbms_ouput.put.line('The reverse string is:`| |ch);
   len=len - 1;
   End loop;
End;
```

25. Accept employee no and check whether it is present in emp table or not.

```
Declare
   v_no emp.eno%type;
   v_eno emp.eno%type;
Begin
   v_eno:=&v_eno;
   Select eno into v_eno
     From emp
       Where eno=v_eno;
       If v_no=v_ eno then
         Dbms_ouput.put_line('Employee is  present`);
       End if;
     When data_not_found then
     Dbms_output.put.line('Employee is not present`);
End;
```

26. Accept employee name and check whether commission is null or not. If commission is num raise an exception otherwise display commission.

```
Declare
   v_comm emp.emp%type;
   v_ename emp.ename%type;
   chk_comm exception;
Begin
   V_ename:=&v_ename;
   Select comm into v_comm
   From emp
   Where ename=v_ename;
     If v_comm is Null then
       Raise chk_comm;
     Else
       Dbms_ouput.put_line( v_comm);
     End if;
  Exception;
  When data_not_found then
  Dbms_ouput.put_line('Ename do not present.`);
  When chk_comm then
  Dbms_ouput.put_line('Ename is null `);
End;
```

27. Accept employee number and prints its details using cursor.

```
Declare
  V_no emp.eno%type;
  V_name emp.ename%type;
  V_desig emp.edesg%type;
  V_sal emp.esal%type;
Begin
  Select ename,edesg ,esal into v_name,v_desig,v_sal
  From emp;
    Where eno=v_no;
      If sql%found then
    Dbms_ouput.put_line('Name:`||v_name || 'Designation:`|| v_desig |
      | 'Salary:`| | v_sal);
  Exception;
  When data_not_found then
  Dbms_output.put_line('Eno do not present.`);
End;
```

28. Accept a salary and print name, salary and designation of employees having salary less than accepted salary.

```
Declare
  Cursor csr(c_sal emp.sal%type) is select  * from emp
  Where sal<c_sal;
  Emp_rec csr%type;
  V_sal emp.sal%type;
Begin
  V_sal =&v_sal;
  Dbms_output.put_line('Name    Salary  Designation`);
  For emp_rec in  csr(v_sal)
    Loop
      Dbms_ouput.put_line(emp_rec.name | | '`| |emp_rec.sal | | '` |
        |emp_rec.desig);
  End loop;
End;
```

29. Print the details of employees who belongs to department no 5.

```
Declare
  Cursor csr is select empname,empsal,empdesg,dno
  From emp
  Where dno=5;
  V_n emp.empname%type;
  V_s emp.empsal%type;
  V_des emp.empdesg%type
  V_dno emp.dno%type;
```

```
Begin
   Open csr;
   Dbms_ouput.put_line('Name designation salary departmentNo`);
   Loop;
   Fetch csr into v_n,v_s,v_dno;
   Exit when csr%not found;
   Dbms_ouput.put_line(v_n| | '`  | |v_des | | '`| |  v_s | | '` | |v_dno );
   End loop;
End;
```

---

30. Print name and salary of employee having designation as clerk and assistant.

```
Declare
   Cursor c is select empname,empsal
   From emp
   Where empdesg= 'clerk` and empdesg= 'assistant`;
   V_n emp.empname%type;
   V_s emp.empsal%type;
Begin
   Open c;
   Dbms_output.put_line('Name salary`);
   Loop
   Fetch c into v_n,v_s;
   Exit when c% not found;
   Dbms_ouput.put_line(v_n | | '` | |v_s);
   End loop;
End;
```

---

31. To Create/modify a trigger

```
Create or replace trigger trigger_sal
   Before insert on emp
   For each row
Begin .
   If acc.sal<=0 then
   Dbms_output.put_line('Salary must be non-negative`);
   End if;
End;
```

---

32. Passing eno as a parameter to procedure and modifying salary of that employee.

```
Create or replace procedure emp_proc
   (no in number)Is
   V_empsal number;
Begin
   Select sal into v_empsal
```

```
  From emp
    Where eno=no;
    If v_empsal > 2000 then
      Update emp
      Set sal=v_empsal*1.75
      Where eno=no;
    Else
      Update emp
      Set sal=7000
      Where eno=no;
    End if;
  Exception
  When No_Data_Found then
  Dbms_output.put_line('Eno do not present`);
End proc;
```

---

33. Designation to the calling program.

```
Create or replace procedure proc1
(p_no IN number,p_desg OUT emp.desg%type)
Is
  V_desg emp.desg%type
Begin
  Select desg into v_desg
  From emp
    Where eno=p_no;
    P_desg:=v_desg;
  Exception
  When No_data_found then
  P_desg= 'No`;
End proc1;
Declare
  F_eno number;
  F_desg emp.desg%type;
Begin
  Proc1(&f_eno,f_desg);
  If f_desg= 'No`;
  Dbms_output.put_line('Eno do not exists);
Else
  Dbms_ouput.put_line('Designation of employee is:` | |f_desg);
End if;
End;
```

34. Passing employee name as an argument to function and function will return its designation.

```
Create or replace function func1 (f_name In Character)
Return character
Is
   V_desg emp.desg%type;
Begin
   Select desg into v_desg
   From emp
   Where ename=f_name;
   If sql%found then
   Return(v_desg);
   Else
   Return null;
   End if;
End func1;
```

35. Pass department number to procedure and print maximum salary of employee working in that department. If department number does not exist print message.

```
Create or replace procedure proc1 (p_dno in number)
As
   Max_sal emp.empsal%type;
Begin
   Select max(empsal) into max_sal
   From emp
   Where dno=p_dno;
   If(max_sal >0) then
   Dbms_output.put_line('Maximum salary :` | |max_sal);
   Else
   Dbms_output.put_line('Dno does not exists`);
   End if;
End proc1;
```

Calling program

```
Declare
   V_dno emp.dno%type;
Begin
   V_dno:=&v_dno%type;
   Proc1(v_dno);
End;
```

36. Pass department number to a procedure. Procedure will number of employees working in that dept using In Out variable.

```
Create or replace procedure proc2
(pr_num in out number)
As
   V_num number;
Begin
   Select count(*)into v_num
   From emp
   Where dno:=pr_num;
      Pr_num:=v_num;
   Exception
      When No_data_found then
      Pr_num:=0;
End proc2;
```

Calling program

```
Declare
   P_no number;
Begin
   P_no:=&p_no;
   Proc2(p_no);
   If(p_no=0) then
      Dbms_output.put_line('Passed dno do not exist.`);
   Then
      Dbms_output.put_line('No.of employees=` | |p_no);
   End if;
End;
```

---

37. Create a procedure, which display employee details and department name of 1st 5 lowest paid employees.

```
Create or replace procedure proc3 as
   Cursor c is select empname,empsal,empdesg,dname
   From emp,dept
   Where emp.dno=dept.dno
      Order by empsal  asc;
      Emp_rec c%rowtype;
Begin
   Dbms_output.put_line('Name Designation Salary Department`);
   For emp_rec in c
   Loop
      If c%rowcount<=5 then
      Dbms_output.put_line(emp_rec.empname | |`` | |emp_rec_empdesg | |
          '`| | emp_rec.empsal | | '` | |emp_rec.dname);
   End if;
   End loop;
End proc3;
```

38. Pass designation to a procedure and print names and salary of employee whose salary is more than average salary of accepted designation.

```
Create or replace procedure proc4(pr_desg in varchar)
As
   Cursor c is select empname,empsal from emp
   Where empsal<=(select avg(empsal) from emp
   where desg=pr_desg);
   emp_rec c%rowtype;
begin
   dbms_output.put_line('Name Salary`);
   for emp_rec in c;
      loop
         dbms_output.put_line(emp_rec.empname| | '`| |emp_rec.empsal);
   end loop;
end proc4;
```

39. Pass a number to a function and check whether it is divisible by 3 or not.

```
Create or replace function div(f_no number in number)
Return number
As
Begin
   If(mod(f_no,3)=0) then
      Return 1;
   Else
   Return 0;
   End if;
End div;
```

Calling program

```
Declare
   Num number;
   Rev number;
Begin
   Num:=&num;
   Rev:=div(num);
   If(rev=1) then
      Dbms_output.put_line('Given no is divisible by 3`);
   Else
      Dbms_output.put_line('Given no is not  divisible by 3`);
   End if;
End;
```

40. Pass two strings to a function and print, which string is smallest.

```
Create or replace function str
(s1 in varchar, s2 in varchar)
Return varchar;
As
   Len1=length(s1);
   Len2=length(s2);
   If(len1>len2)
   Then
     Return '2nd  string is smallest`;
     Else if(len1<len2)
       Return '1st string is smallest.`;
   Else
     Return 'Both are equal`;
     End if;
 End if;
End str;
```

41. Pass a character to a function and print numbers of employees having name starting with the passed character.

```
Create or replace function no_of_emp( f_ch in varchar2)
Return number;
As
   V_cnt number;
   Select count(*) into v_cnt
   From emp
     Where instr(empname,f_ch)=1;
   Return v_cnt;
End no_of_emp;
```

Calling program

```
Declare
   V_ch varchar(20);
   n number;
Begin
   V_ch:=&v_ch;
   n:=no_of_emp(v_ch);
   if(n>0) then
     Dbms_output.put_line('No of employees having name starting with ` |
     |v_ch | | 'are:` | |v_cnt);
   Else
     Dbms_output.put_line('No one emp having name starting with`| |v_ch);
   End if;
End;
```

42. Write a package, which works as a arithmetic calculator.

```
Create or replace package calci
As
Procedure prc(p_num1 number,p_num2 in number, p_oper in varchar);
End calci;
Create or replace package body calci
As
   Procedure prc(p_num1.p_num2,p_oper in varchar) as
   answer number;
begin
   if(p_oper= '+`) then
      answer:=p_num1 + p_num2;
      elseif(p_oper= '-`) then
         answer:=p_num1 - p_num2;
      elseif(p_oper= '*`) then
         answer:=p_num1 * p_num2;
      elseif(p_oper= '/`) then
         answer:=p_num1 / p_num2;
   end if;
   dbms_output.put_line('Answer=` | |answer);
   exception
   when Zero_Divide then
   dbms_output.put_line('Divide by zero error bcoz- 2nd  number is 0`);
end prc;
end calci;
```

43. Write a package, which consist of 1 procedure and 1 function; Pass a number to procedure and print factorial of it. Pass name to function and print details of that employee.

```
Create or replace package pack
As
Procedure prc1(p_num in number);
Function func(f_name in varchar)
Return varchar;
End pack;
Create or replace package body pack
As
Procedure prc1(p-num number) as
   f number:=1;
     n number;
begin
```

```
  for n in 1…p_num
  loop
    f:=f*n;
  end loop;
  dbms_output.put_line('Factorial=` | | f);
end prc1;
function func (f_name in varchar)
return varchar
as
  v_name emp.ename%type;
  v_desg emp.edesg%type;
  v_sal emp.esal%type;
begin
  select ename ,edesg,esal  into v_name,v_desg,v_sal from emp
  where ename=f_name;
  return v_name,v_desg,v_sal;
  end func;
end pack;
```

Calling program

```
Declare
  num number;
  name varchar(20);
begin
  num:=&num;
  pack.prc1(num);
  name:=&name;
  dbms_output.put_line('Name Designation  Salary` | | func(name)  );
end;
```

---

44. A Function for the list of the actor name and movie name in which actor's rate is greater than 5 lakhs.

```
declare
  cursor c1 is select mname,aname
       from movie,actor,mov_act where
       mov_act.mno=movie.mno
       and mov_act.ano=actor.ano
       and rate>500000;
  c c1%rowtype;
begin
  dbms_output.put_line('MovieName||'    '||ActorName);
  open c1;
  loop
```

```
   fetch c1 into c;
   exit when c1%notfound;
          dbms_output.put_line(c.mname||'    '||c.aname);
   end loop;
   close c1;
end;
```

45. Define a trigger before insert or update of each row of movie, that movies released after 2004 be entered into movie table.

```
create or replace trigger t_mov_2004
before insert or update on movie
for each row
begin
   if(:new.relyear < 2004) then
     raise_application_error(-20001,' Release YEAR SHOULD BE > 2004');
   end if;
end;
```

46. Write a script to list the names of all employees who are female and earning maximum salary in their department.

```
declare
   cursor c1 is select dept_no,dept_name from dept;
   cursor c2(dno number) is select emp_name from emp,dept
        where emp.salary in (select max(salary),emp.emp_name
        from emp,dept where emp.dept_no=dno
        and dept.dept_no=emp.dept_no and sex='F'
        group by dept.dept_no);
   c c1%rowtype;
   d c2%rowtype;
begin
   open c1;
   loop
   fetch c1 into c;
   exit when c1%notfound;
   open c2(c.dept_no);
   loop
   fetch c2 into d;
   exit when c2%notfound;
     dbms_output.put_line(c.dept_name||d.emp_name);
   end loop;
   close c2;
   end loop;
   close c1;
end;
```

47. Write a script to give raise in salary by 5% for all the employees earning less than 3000 and 9% for all employees earning more than or equal to 3000. Also print total numbers of employees in each case.

```
declare
  cursor c2 is select emp_no,salary from emp,dept
       where dept.dept_no=emp.dept_no;
  c c2%rowtype;
  cnt1 number;
  cnt2 number;
begin
  cnt1:=0;
  cnt2:=0;
  open c2;
  loop
  fetch c2 into c;
  exit when c2%notfound;
  if(c.salary < 3000) then
        update emp set salary = salary + salary * 0.05
                    where emp.emp_no = c.emp_no;
  cnt1:=cnt1+1;
  end if;
  if(c.salary >= 3000) then
        update emp set salary = salary + salary * 0.09
            where emp.emp_no = c.emp_no;
  cnt2:=cnt2+1;
       end if;
  end loop;
  close c2;
     dbms_output.put_line('No of employees getting 5% increase= '||cnt1);
  dbms_output.put_line('No of employees getting 9% increase= '||cnt2);
end;
```

48. Write a script to transfer all employees of dept. "A" of location "AB" earning the commission of 50% of their salary to dept "B". Also print the total number of employees of dept. "A" transferred to dept "AB".

```
declare
  cursor c1 is select dept_no from dept1 where dept_name='A';
  cursor c2 is select emp_no,salary,dept_name,location,comm from
emp1,dept1
       where dept1.dept_no=emp1.dept_no;
  c c1%rowtype;
  r c2%rowtype;
  cnt number;
```

```
begin
   cnt := 0;
   open c2;
   loop
   fetch c2 into r;
   exit when c2%notfound;
   open c1;
   loop
   fetch c1 into c;
   exit when c1%notfound;
     if(r.location ='AB' and r.dept_name='A') then
           if(r.comm >= (r.salary/2)) then
                        update emp1 set dept_no = c.dept_no
                       where emp1.emp_no = r.emp_no;
                      cnt:=cnt+1;
            end if;
     end if;
   end loop;
   close c1;
   end loop;
   close c2;
        dbms_output.put_line('Total no of emp of dept A transferred to
dept AB are = '||cnt);
end;
```

49. Write a script for the following: give the names of all those locations, which has total of at least 6 depts in it. Out of which at least 6 depts are spending approximate Rs. 40000/- as salary of the employee.

```
declare
   cursor c1 is select location from emp1,dept1
        where dept1.dept_no=emp1.dept_no group by location
        having count(dept1.dept_no)>=6;
   cursor c2(loc varchar2) is select location from emp1,dept1
          where dept1.location=loc and
          dept1.dept_no=emp1.dept_no group by location
          having count(dept1.dept_no)>= and
                sum(salary) >= 40000;
   c c1%rowtype;
   d c2%rowtype;
begin
   open c1;
   loop
   fetch c1 into c;
   exit when c1%notfound;
        open c2(c.location);
```

```
        loop
             fetch c2 into d;
              exit when c2%notfound;
                 dbms_output.put_line('Location = '||d.location);
        end loop;
        close c2;
  end loop;
  close c1;
end;
```

50. Write a trigger, which gets activated when company tuple is updated. It should delete all the related tuples when share value of company becomes < Rs.15/-.

```
create or replace trigger t_comp
after insert or update on company
for each row
declare
  c company.c_no%type;
  v company.c_share%type;
begin
     select c_no,c_share into c,v from company
     where c_share < 15;
           delete from company where c_no=c;
             delete from comp_per where c_no=c;
end;'
```

51. Write a script, which gets update the book details of book no. entered by user. Raise exception if the given book number is not present or if the price of the book is greater than 4000.

```
create or replace procedure update_book(bno in number,bname in
varcahr2,bval in number)
as
  c1 cursor for select * from book;
  c c1%rowtype;
  flag number;
begin
  flag = 0;
  open c1;
  loop
       fetch c1 into c;
       exit when not found;
       if(c.b_no=bno and c.price<=4000) then
              update book set b_name=bname,price=bval
              where b_no=bno;
              flag = 1 ;
              dbms_output.put_line('VALUES UPDATED SUCCESSFULLY');
```

```
        end if;
   end loop;
   close c1;
   if(flag = 0) then
                dbms_output.put_line('BOOK NOT FOUND');
   end if;
end;
```

52. Write a script, which will take publisher name as parameter and will display details of books.

```
create or replace procedure get_book(pname in varchar2)
as
   flag number;
   cursor c1 is select p_name,b_name from book,publisher,book_pub
        where publisher.p_no=book_pub.p_no
        and book.b_no=book_pub.b_no;
   c c1%rowtype;
begin
   flag = 0;
   open c1;
   loop
        fetch c1 into c;
        exit when c1%notfound;
        if(c.p_name=pname) then
                dbms_output.put_line('Book = '||c.b_name);
                flag=1;
        end if;
   end loop;
   close c1;
   if(flag = 0) then
          dbms_output.put_line('ERROR :: BOOK_NOT_FOUND');
     end if;
end;
```

53. A trigger that will take care of the constraint that movie released after 1995 be entered in the movie table.

```
create or replace trigger t_mov_1995
before insert or update on movie
for each row
begin
   if(:new.relyear < 1995) then
     raise_application_error(-20001,' YEAR SHOULD BE > 1995');
   end if;
end;
```

54. Write a script for the following

List the names of publishers who have published at least 2 books whose prices are greater than 200 respectively for a department named computer science.

```
declare
  cursor c1 is select p_no from publisher;
  cursor c2(pno number) is select p_name,b_name,d_name,price
         from book,publisher,dept
         where book.p_no = publisher.p_no
         and book.d_no=dept.d_no
         and book.p_no = pno
         having count(book.p_no)>=2;
  c c1%rowtype;
  d c2%rowtype;
begin
  open c1;
  loop
       fetch c1 into c;
       exit when c1%notfound;
       open c2(c.p_no);
       loop
              fetch c2 into d;
              exit when c2%notfound;
              if(d.price > 200 and d.d_name='comp') then
                     dbms_output.put_line('PUBLISHER : '||d.p_name);
                     dbms_output.put_line('BOOK  : '||d.b_name);
                     dbms_output.put_line('PRICE : '||d.price);
              end if;
       end loop;
       close c2;
  end loop;
  close c1;
end;
```

---

55. Write a script for the following

List the names of doctors who visit every hospital located in the city where they do not live.

```
declare
  cursor c1 is select doc_no,doc_name,city
       from doctor;
  cursor c2(dcity varchar2) is  select hosp_no
            from hosp
            where hosp_city != dcity;
  cursor c3(dno number,hno number) is select doc_no,hosp_no
            from doc_hosp
```

```
            where doc_no = dno
            and hosp_no = hno;
   c c1%rowtype;
   d c2%rowtype;
   e c3%rowtype;
begin
   open c1;
   loop
   fetch c1 into c;
   exit when c1%notfound;
         open c2(c.city);
         loop
         fetch c2 into d;
   exit when c2%notfound;
                  open c3(c.doc_no,d.hosp_no) ;
     loop
          fetch c3 into e;
     exit when c3%notfound;
              dbms_output.put_line('DOCTOR = '||e.doc_name);
          end loop;
           close c3;
        end loop;
        close c2;
   end loop;
   close c1;
end;
```

---

56. Write a script for the following:

Increase the fare of AC rooms by 80% and NON AC rooms by 30%.

```
declare
  cursor c1 is select * from room;
  c c1%rowtype;
begin
  open c1;
  loop
       fetch c1 into c;
       exit when c1%notfound;
       if(c.r_type='AC') then
               update room set fare = c.fare * 1.8
               where r_no = c.r_no;
       else
               update room set fare = c.fare * 1.30
               where r_no = c.r_no;
       end if;
```

```
  end loop;
  close c1;
    dbms_output.put_line(' FARES UPDATED SUCCESSFULLY ');
end;
```

57. Program to find largest number between two numbers.

```
Declare
  N1 number;
  N2 number;
Begin
  N1:=&n1;
  N2 :=&n2;
  If(N1>N2) then
    Dbms_output.put_line('N1 is Largest`);
  Else
    Dbms_output.put_line ('N2 is Largest`);
  End if;
End;
```

58. Consider the following Relational Database:

1

Oct.2012 – 4M

Doctor(d_no, d_name, d_city)

Hospital(h_no, h_name, h_city)

Doc_Hosp(d_no, h_no)

Write a function which will count number of doctors visiting to 'Poona' hospital.

Solution

```
create or replace function cdoctor()return number is
dno Doc_Hosp.d_no%type;
begin
select count(*)into dno
from Doctor, Hospital, Doc_Hosp
where Doctor.d_no=Doc_Hosp.d_no
and
Hospital.h_no=Doc_Hosp.h_no
and Hospital.h_name=`Poona Hospital`;
return dno;
exception
when_no_data_found then
raise_application_error(-20000,` Doctors does not exist`);
end;
```

59. Consider the following Relational Database:

1 Oct.2012 – 4M

Book (b_no, b_name, pub_name, price)
Author (a_no, a_name)
Book-Auth (b_no,a_no)
Write a procedure to display details of all books written by 'Mr. Mohite'.

Solution

```
create or replace procedure dbooks() is cursor c1 is
select Book.b_no,b_name,pub_name,price from
Book, Author,Book_Auth
where
    Book.b_no=Book_Auth.b_no and
    Author.a_no=Book_Auth.ano and a_name='Mr. Mohite`;
    bno Book.b_no%type;
    bname Book.b_name%type;
    pname Book.pub_name%type;
    bprice Book.price%type;
  begin
    open c1
loop
  fetch c1 into bno,bname,pname,bprice;
  exit when c1%notfound;
  dbms_ouput.put_line(bno|| bname || pname || bprice);
end loop;
close c1;
end;
```

1 Oct.2012 – 4M

60. Consider the following Relational Database:
Customer(c_no, c_name, c_city)
Loan(l_no,l_amt, no_of_years, c_no)
Define a trigger that restricts updation of Loan Amount.

Solution

```
create or replace trigger tloan
before update on Loan
for each row
begin
if(:new.l_amt <>:old.l_amt) then
raise_application_error(-202,`cannot update`);
end if;
end;
```

61. Consider the following Relational Database: 1

Employee(e_no, e_name, city,dept_name)
Project(p_no, p_name, status)
Emp_Proj(e_no, p_no, no_of_days)

Oct.2012 – 4M

Write a cursor to display details of all projects having status 'Completed'.

Solution

```
declare
cursor c1 is
select p_no,p_name,status
from Project,Employee,Emp_Proj
where Employee.e_no=Emp_proj.e_no
and Project.p_no=Emp_Proj.p_no
And status=`completed`;
prec  Project%row type;
begin
open c1
loop
fetch c1 into prec;
    exit when c1%not found;
    dbms_output.put_line(prec.p_no|| prec.p_name||
prec.status);
end loop;
end;
```

---

62. Write a package which consists of one procedure and one function. For this consider the following Relational Database:

1

Oct.2012 – 4M

Customer (cust_no, cust_name, cust_city)
Account (acc_no, acc_type, balance, cust_no)

i. **Pass account number as a parameter to a procedure and display account details.**
ii. **Pass customer number as a parameter to a function and return total number of accounts of given customer.**

Solution

```
Create or replace package pack1
as
procedure p1(ano in number);
function f1(cno in number);
return number;
end pack1;
/
create or replace package body pack1
as
```

```
procedure p1(a in number)
is
arec Account%row type;
begin
select * into arec
from account
where acc_no=a;
dbms_output.put_line(arec. acc_no || ' '||arec.acc_type||` `||
arec.balance
||` `|| arec.cust_no);
end p1;
function f1(cno in number) return number
is
n number;
begin
select count(*) into n
from Account, Customer
where Customer.cust_no=Account.cust_no
and Account.cust_no=cno;
return n;
end f1;
end pack1;
/
```

1 Apr.2012 – 4M

63. Consider the following relational database:
Department (D_no, D_name, Location)
Employee (Eno, Ename, Edesg, Esalary, D_no)
Write a cursor to display the department details of employee "Mr. Joshi"

Solution

```
DECLARE
        CURSOR c1 is
        select D_no,D_name,Location
        from Department, Employee
        where Department.D_no=Employee.D_no
        and Ename= 'Mr.Joshi`;

Dno Department.D_no%type;
Dname Department.D_name%type;
Loc Department.Location%type;

BEGIN
dbms_output.put_line(Department NO||'  '||Department Name||`
```

```
||Location);
OPEN c1;
if c1%ISOPEN  THEN
 LOOP
FETCH c1 into Dno,Dname,loc;
EXIT WHEN c1%NOTFOUND;
if c1%FOUND THEN
dbms_output.put_line(Dno|| '    '||Dname||`   '||loc);
END IF;
END LOOP;
ELSE
dbms_output.put_line('unable to open cursor');
END IF;
CLOSE c1;
END;
/
```

---

64. Consider the following relational database:
Item(Itemno, Itemname, Qty)
Supplier (Supplierno, Suppliername, Address, City, Phno)
I_S(Itemno, Supplierno, Rate, Discount)
Define a trigger before updation on discount field, if the difference in the old discount and new discount entered is > 15% raise an exception and display corresponding message.

1 Apr.2012 – *4M*

Solution

```
create or replace trigger ISDiscount
before insert or update
on I_S
for each row
begin
if: old.Discount - new.Discount > 0.15 then
raise_application_error(-20002,'Discount is greater than 15%`);
endif;
end;
```

---

65. Consider the following relational database:
Game (Game_no, Game_name, Team_size, Name_of_coach)
Player (Player_no, Player_name, Player_city)
Game _Player (Game_no, Player_no)
Write a function which will take game name as a parameter and return total number of players playing that game.

1 Apr.2012 – *4M*

Solution

```
Create or replace function rtotalbooks(gname varchar2)return
number is
Total player.player_no%type
begin
select count(Game_Player.Player_no)into Total
from Game,Player,Game_Player
where
Game.Game_no=Game_Player.Game_no
and Player.Player_no=Game_Player.Player_no
and Game_name=gname;
return(Total);
exception
when no_data_found then
raise_application_error(-20000,`Game does not exist`);
end;
```

1

Apr.2012 – *4M*

66. Consider the following relational database:

Publisher (P_no, P_name, P_addr);

Book (B_no, B_name, Price, P_no);

Write a script, which will update the book details of book number entered by user. Raise exception if the given book number is not present or if the price of the book is greater than 500.

Solution

```
declare
mbook_no Book.B_no%type;
mbook_name Book.B_name %type;
mprice Book.Price%type;
moreprice exception;
begin
select B_no into mbook_no
from Book
where
B_no=`&B_no`;
if mbook_no is not null
then
mbook_name=`&B_name`;
mprice:=`& Price`;
if mprice>500 then
raise moreprice;
else
update Book
set B_name=mbook_name, Price=mprice
where B_no=mbook_no;
end if;
```

```
end if;
exception
when no_data_found then
raise_application_error(-20000,`Book not found`);
when moreprice then
raise_application_error(-20001,`Price > 500/-');
end;
```

1

Apr.2012 – *4M*

67. Write a package, which consists of one procedure and one function; pass a number to procedure and print addition of two numbers. Pass city name as a parameter to function and display number of hospitals located in that city for consider following relation:

Hospital (Hno, Hname, Hcity)

Solution

```
Create or replace package pack1
as
procedure p1(a in number, b in number);
function f1(fname varchar2)
return number;
end pack1;
/
create or replace package body pack1
as
procedure p1(a in number, b,in number)
as
c number;
begin
c=a+b;
dbms_output.put_line('addition`||c);
end p1;
function f1
(cname varchar2)
return number
as
n number;
begin
select (count*) into n from hospital
where Hcity=cname;
return n;
end f1;
end pack1;
/
```

1

Oct.2011 – *4M*

68. Consider the following relational database:
Doctor(doct_no,doct_name,d_city)

Hospital(hosp_no,hosp_name,h_city)

Doc-Hos(doct_no,hosp_no)

Write a script of cursor to print the list showing the hospital-wise list of doctors.

Solution

```
declare
  cursor c1 is select * from hospital;
  cursor c2(hno varchar ) is
select d.doct_name
  from doctor d,Doc-Hos
  where hno=Doc-Hos.hosp_no and d.doct_no=Doc-Hos.doct_no;
begin
  for x in c1 loop
    dbms_output.put_line('Hospital:`||` '||x.hosp_name);
    dbms_output.put_line('doctor:`);
  for y in c2(x.hosp_no) loop
    dbms_output.put_line('  '||y.doc_name);
end loop;
end loop;
end;
```

1 Oct.2011 – 4M

69. Consider the following relational database:

Customer(cust_no,cust_name,cust_city)

Account(acc_no,acc_type,balance)

Cust_Acc(cust_no,acc_no)

Define a trigger before insert or update of each row of account table for existing customer, if the customer is having balance less than Rs.500 in his account then raise an exception and display corresponding message.

Solution

```
create or replace trigger CAccount
Before insert or update
on Account
for each row
begin
  if: new.balance <500 then
    raise_application_error(-20000,`Balance should be greater
    than 500`);
  end if;
end;
```

70. Consider the following relational database

Publisher(p_no,p_name,p_addr)

1 Oct.2011 – 4M

Book(b_no,b_name,price)

Pub-Book(p_no,b_no)

Write a function that will accept publisher name as parameter and return number of books published by that publisher.

Solution

```
create or replace listofbooks(pubname varchar)return numberis
    Total Book.b_no%type;
begin
  select count(b_no)into total from Publisher,Book,Pub-Book
where
  Publisher.p_no=Pub-Book.p_no and
  Book.b_no=Pub-Book.b_no
  and p_name=pubname;
  return(total);
  exception
  when no_data_found then
    raise_application_error(-20000,`publisher does not
  exist`);

end;
```

---

71. Consider the following relational database

1

Oct.2011 – *4M*

Department(d_no,d_name,location)

Employee(e_no,e_name,e_addr,e_salary,d_no)

Write a procedure which will take department name as parameter and will display details of employees working in that department.

Solution

```
create or replace procedure detailsofemp(dname varchar)
cursor c1 is
select e_no,e_name,e_addr,e_salary
from Employee,Department
where
  Department.d_no=Employee.d_no and d_name=dname;
  Empno Employee.e_no%type;
  Empname Employee.e_name%type;
  Empaddr Employee.e_addr%type;
  Empsal Employee.e_salary%type;
begin
  open c1;
loop
```

```
    fetch c1 into Empno,Empname,Empaddr,Empsal;
    exit when c1%notfound;
    dbms_output.put_line(Empno||Empname||Empaddr||Empsal);
end loop;
  close c1;
end;
```

1 Oct.2011 – 4M

72. Write a package, which consist of one procedure and one function. Pass two numbers to procedure and print largest number. Pass department number to function and print location of that department for this consider following relation: Department(d_no,d_name,location)

Solution

```
create or replace package pack1
as
  procedure p1(n,m in number);
    function f1(dno number) return varchar
end pack1;
create or replace package body pack1
as
  procedure p1
  (n,m in number)
begin
  if(n>m)
    dbms_output.put_line(' n is largest number`||n);
  else
    dbms_output.put_line('m is largest number`||m);
  end if;
  end;
  function f1
    (dno number)
  return varchar
  as
    loc varchar;
  begin
   select location into loc from department where d_no=dno;
  return loc;
  end f1;
end pack1;
```

73. Consider the following Relational Database:
Employee(eno., ename, city, deptname)

1 Oct.2014 – 4M

Project (pno., pname, status)

Emp-proj(eno,pno, no-of-days)

Write a cursor which will display project wise list of employee.

Solution

```
declare
cursor C1.is
select pno,pname,status
from Employee,Project,Emp-proj
where Employee.eno = Emp-proj.eno
and Project.pno = Emp-proj.pno
rec C1%rowtype;
begin
open C1;
loop
fetch
fetch C1 into rec;
exit when c1%not found;
dbms_output.put_line (rec.pno || rec.pname || rec.status)
end loop;
close C1;
end;
```

---

74. Consider the following relational database: 1

Dept(deptno., deptname, location)

Emp (empno, empname, sal, comm, designation, deptno)

Oct.2014 – *4M*

Write a procedure to increase salary of given employee by 5% and display updated salary.

Solution

```
create or replace procedure
p1(name in varchar2)as
salary number; en number;
                nm varchar2;
begin
update Emp set sal=sal+0.05*sal
where empname=name;
select empno,empname,sal into
en,nm, salary from Emp
where empname = name
dbms_output.put_line(en||nm||salary);
end;
```

1 Oct.2014 – 4M

75. Consider the following relational database:

Movie (mno, mname, relyear)
Actor(ano, aname)
Mov-act(mno,ano)

Define a trigger before insert or update of each row of movies that movie released after 2010 be entered into movie table.

Solution

```
create or replace trigger t_mov
before insert or update on Movie
for each row
begin
if(:new.relyear<2010)then
raise_application_error(-20001,'Release year should be > 2010`);
end if;
end;
```

1 Oct.2014 – 4M

76. Consider the following Relational Database:
Politician (pno, pname, desig, partycode)
Party(partycode, partyname)
Write a function to return total number of politicians of a given party.

Solution

```
create or replace function
totalpoliticians(pname in varchar2)
return number is
total number;
begin
select count(Politician.partycode)
into total from Politician, Party
where partyname = pname and Politician.partycode = Party.partycode;
return(total);
exception
when no_data_found then
raise_application_error(-20000; 'party does not exist`);
end;
```

1 Oct.2014 – 4M

77. Write a package which consists of one procedure and one function.
Consider relation person.
Person(pno, pname, paddr, pcity,phno)

Procedure of a package will display details of given person. Function of a package will count number of person from Pune city.

Solution

```
create or replace package pack1
as
procedure p1(no in number);
function f1();
return number;
end pack1;
/
create or replace package body pack1
procedure p1(no in number)
is
rec Person%rowtype;
begin
select * into rec from Person
where pno=no;
dbms_output.put_line(rec. pno || rec.pname||rec.paddr||
rec.pcity||rec.phno);
end p1;
function f1() return number
is
cnt number;
begin
select count(*) into cnt
from Person where pcity = "Pune";
return cnt;
end f1;
end pack1;
/
```

PU Questions

2 Marks

1. What is PL/SQL? Give advantages of PL/SQL. **[Oct.2015 – 2M]**
2. Give syntax of stored procedure in PL/SQL. **[Oct.2015 – 2M]**
3. Define cursor. Enlist attributes of cursor. **[Oct.15,14 – 2M]**
4. What is difference between % type and % row type? **[Apr.15,Oct.10 – 2M]**
5. Write syntax of for loop in PL/SQL with example. **[Apr.2015 – 2M]**
6. Write syntax and example of while loop in PL/SQL. **[Oct.2014 – 2M]**
7. Give proper syntax of trigger. **[Oct.12,Apr.10,09 – 2M]**
8. Which are different attributes of cursor? **[Oct.2012 – 2M]**
9. Define PL/SQL. What is use of PL/SQL? **[Apr.2012 – 2M]**
10. What is structure of PL/SQL block? **[Apr.2012 – 2M]**
11. What is Trigger? What are the types of Trigger? **[Apr.2012 – 2M]**

[Oct.2011 – 2M] 12. What is Cursor? List the Attributes of Cursor.

[Oct.2011 – 2M] 13. What is PL/SQL? Give Pl/SQL block structure.

[Apr.2011 – 2M] 14. List modes of trigger and its syntax.

[Apr.2011 – 2M] 15. Give proper syntax of procedure in PL/SQL.

[Oct.2010 – 2M] 16. List the steps involved in defining the explicit cursor.

[Oct.2010 – 2M] 17. Give proper syntax for function in PL/SQL.

[Apr.10,Oct.09 – 2M] 18. What is Cursor? Which are the various attributes of Cursor?

4 Marks

[Oct.2015 – 4M] 1. 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).

[Oct.2015 – 4M] 2. 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.

[Oct.2015 – 4M] 3. Consider the following relational database.
Wholesaler (wno, wname, city)
Product (pno, pname, price)
Wp (wno, pno)
Write a cursor to display wholesalerwise product details.

[Oct.2015 – 4M] 4. Consider the following relational database:party (pcode, pname)
politician (pno, pname, pcity, pcode)

[Oct.2015 – 4M] 5. 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

[Oct.2015 – 4M] 6. 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.

[Oct.2015 – 4M] 7. Write a note on exception handling in PL/SQL.

[Oct.2015 – 4M] 8. What is Trigger? Explain types of trigger in detail.

9. 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'. **[Apr.2015 – 4M]**

10. Consider the following relational database. **[Apr.2015 – 4M]**
    Book (bno, bname, pubname, price, dno)
    Department (dno, dname)
    Write a procedure which will display total expenditure on books by a given department.

11. Consider the following relational database. **[Apr.2015 – 4M]**
    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 data on 'Sunday'.

12. Consider the following relational database. **[Apr.2015 – 4M]**
    Politician (pno, pname, description, partycode)
    Party (partycode, partyname)
    Write a clusor to display partywise details of politicians.

13. Consider the following relational database. **[Apr.2015 – 4M]**
    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.

14. What is cursor? Explain different attributes used in it. **[Apr.15,12 – 4M]**

15. What is trigger? Explain trigger with proper syntax and example. **[Apr.2015 – 4M]**

16. What is PL/SQL? Explain block of PL/SQL. **[Apr.2015 – 4M]**

17. What is exception handling? Explain user defined exception with example. **[Oct.2014 – 4M]**

18. What is the difference between function and procedure, explain it with example. **[Oct.2014 – 4M]**

19. Explain different data types in PL/SQL. **[Oct.14 Apr.12,10 – 4M]**

[Oct.2014 – 4M] 20. Consider the following Relational Database:
Employee(eno., ename, city, deptname)
Project (pno., pname, status)
Emp-proj(eno,pno, no-of-days)
Write a cursor which will display project wise list of employee.

[Oct.2014 – 4M] 21. Consider the following relational database:Dept(deptno., deptname, location)
Emp (empno, empname, sal, comm, designation, deptno)
Write a procedure to increase salary of given employee by 5% and display updated salary

[Oct.2014 – 4M] 22. Consider the following relational database:
Movie (mno, mname, relyear)
Actor(ano, aname)
Mov-act(mno,ano)
Define a trigger before insert or update of each row of movies that movie released after 2010 be entered into movie table.

[Oct.2014 – 4M] 23. Consider the following Relational Database:
Politician (pno, pname, desig, partycode)
Party(partycode, partyname)
Write a function to return total number of politicians of a given party.

[Oct.2014 – 4M] 24. Write a package which consists of one procedure and one function.
Consider relation person.
Person(pno, pname, paddr, pcity,phno)
Procedure of a package will display details of given person.
Function of a package will count number of person from Pune city.

[Oct.2012 – 4M] 25. Explain following pre-defined exceptions.
no_data_found,zero_divide,too_manyrows, dup_val_on_index.

[Oct.12,Apr.11 – 4M] 26. What is Parameterized Cursor? Explain it with example.

[Oct.2012 – 4M] 27. Consider the following Relational Database:
Doctor(d_no, d_name, d_city)
Hospital(h_no, h_name, h_city)
Doc_Hosp(d_no, h_no)
Write a function which will count number of doctors visiting to 'Poona' hospital.

[Oct.2012 – 4M] 28. Consider the following Relational Database:
Book (b_no, b_name, pub_name, price)
Author (a_no, a_name)
Book-Auth (b_no,a_no)
Write a procedure to display details of all books written by 'Mr. Mohite'.

29. Write a package which consists of one procedure and one function. For this consider the following Relational Database: [Oct.012 – 4M]
Customer (cust_no, cust_name, cust_city)
Account (acc_no, acc_type, balance, cust_no)
   i. Pass account number as a parameter to a procedure and display account details.
   ii. Pass customer number as a parameter to a function and return total number of accounts of given customer.

30. What is Cursor? Explain different Attributes used in it. [Oct.2012 – 4M]

31. What is Exception Handling? Explain Predefined and User. Defined Exception with example. [Oct.2012 – 4M]

32. Consider the following relational database: [Apr.2012 – 4M]
Department (D_no, D_name, Location)
Employee (Eno, Ename, Edesg, Esalary, D_no)
Write a cursor to display the department details of employee "Mr. Joshi"

33. Consider the following relational database: [Apr.2012 – 4M]
Item(Itemno, Itemname, Qty)
Supplier (Supplierno, Suppliername, Address, City, Phno)
I_S(Itemno, Supplierno, Rate, Discount)
Define a trigger before updation on discount field, if the difference in the old discount and new discount entered is > 15% raise an exception and display corresponding message.

34. Consider the following relational database: [Apr.2012 – 4M]
Game (Game_no, Game_name, Team_size, Name_of_coach)
Player (Player_no, Player_name, Player_city)
Game _Player (Game_no, Player_no)
Write a function which will take game name as a parameter and return total number of players playing that game. [Apr.2012 – 4M]

35. Consider the following relational database:
Publisher (P_no, P_name, P_addr);
Book (B_no, B_name, Price, P_no);
Write a script, which will update the book details of book number entered by user. Raise exception if the given book number is not present or if the price of the book is greater than 500.

[Apr.2012 – 4M] 36. Write a package, which consists of one procedure and one function; pass a number to procedure and print addition of two numbers. Pass city name as a parameter to function and display number of hospitals located in that city for consider following relation:
Hospital (Hno, Hname, Hcity)

[Oct.11,09,Apr.11 – 4M] 37. Explain advantages and disadvantages of PL/SQL.

[Oct.11,09 – 4M] 38. Explain different control structures used in PL/SQL with proper example.

[Oct.2011 – 4M] 39. Consider the following relational database:
Doctor(doct_no,doct_name,d_city)
Hospital(hosp_no,hosp_name,h_city)
Doc-Hos(doct_no,hosp_no)
Write a script of cursor to print the list showing the hospital-wise list of doctors.

[Oct.2011 – 4M] 40. Consider the following relational database:
Customer(cust_no,cust_name,cust_city)
Account(acc_no,acc_type,balance)
Cust_Acc(cust_no,acc_no)
Define a trigger before insert or update of each row of account table for existing customer, if the customer is having balance less than Rs.500 in his account then raise an exception and display corresponding message.

[Oct.2011 – 4M] 41. Consider the following relational database
Publisher(p_no,p_name,p_addr)
Book(b_no,b_name,price)
Pub-Book(p_no,b_no)
Write a function that will accept publisher name as parameter and return number of books published by that publisher.

[Oct.2011 – 4M] 42. Consider the following relational database
Department(d_no,d_name,location)
Employee(e_no,e_name,e_addr,e_salary,d_no)
Write a procedure which will take department name as parameter and will display details of employees working in that department.

[Oct.2011 – 4M] 43. Write a package, which consist of one procedure and one function. Pass two numbers to procedure and print largest number. Pass department number to function and print location of that department for this consider following relation:
Department(d_no,d_name,location)

Employee(e_no,e_name,e_addr,e_salary,d_no)
Write a procedure which will take department name as parameter and will display details of employees working in that department.

44. Write a package, which consist of one procedure and one function. Pass two numbers to procedure and print largest number. Pass department number to function and print location of that department for this consider following relation: Department(d_no,d_name,location) **[Oct.2011 – 4M] [Apr.2011 – 4M]**

45. What is cursor? List the attributes of cursor with suitable example.

46. Consider the following relational database **[Apr.2011 – 4M]**
Publisher(Pub_no,Pub_name,Pub_city)
Book(Book_no,book_name,book_price)
Pub_Book(Pub_no,Book_no)
Write a function which will take publisher name as parameter and will return total number of books published by given publisher.

47. Consider the following relational database **[Apr.2011 – 4M]**
Customer(cust_no,cust_name,cust_city)
Account(Account_no,Account_type,balance,cust_no)
Write a procedure which will take account type as a parameter and will display customer name having accounts of given type.

48. Consider the following relational database **[Apr.2011 – 4M]**
Doctor (Doct_no,Doc_name,doc_city)
Hospital(Hosp_no,Hosp_name,hosd_city)
Doc-Hosp(Doct_no,Hosp_no)
Write a script using cursor to print doctor wise list of hospitals visited

49. Consider the following relational database **[Apr.2011 – 4M]**
Department(Dept_no,Dept_name)
Employee(EMp_no,Emp_name,dersignation,salary,dept_no)
Define a trigger that will take care of the constraint that employee's salary should not be less than zero.

50. Write a package which consists of one procedure and one function. Pass a number as parameter to a procedure and print whether a number is positive or negative. Pass roll number **[Apr.2011 – 4M]**

of student as a parameter to function and return percentage of that student for this consider following relation.
Student(Roll_no,Stud_name,Stud_addr,Stud_percentage)

[Oct.2010 – 4M] 51. What is cursor? Explain two types of cursors.

[Oct.2010 – 4M] 52. What is PL/SQL? Give PL/SQL block structure and explain its details.

[Oct.2010 – 4M] 53. What is the package in PL/SQL? Explain with example.

[Oct.2010 – 4M] 54. Consider the following relational database:
doctor(doct_no,doct_name,doct_city)
hospital(hosp_no,hosp_name,hosp_city)
doct_hosp(doct_no,hosp_no);
Write a script using cursor to print hospitalwise list of doctors.

[Oct.2010 – 4M] 55. Consider the following relational database:
department(dept_no,dept_name)
employee(emp_no,emp_name,designation,salary,dept_no)
Define a trigger that will take care of the constraint that employee salary should not be less than zero.

[Oct.2010 – 4M] 56. Consider the following relational database
publisher(pub_no,pub_name,pub_city)
book(book_no,book_name,price)
pub_book(pub_no,book_no)
Write a procedure which will take publisher name as parameter and will display books published by that publisher.

[Oct.2010 – 4M] 57. Consider the following relational database
customer(cust_no,cust_name,cust_city)
account(acc_no,acc_type,balance,cust_no)
Write a function which will take acc_type as a parameter and will return total number of accounts of given acc-type.

[Oct.2010 – 4M] 58. Write a package which consist of one procedure and one function. Pass a number as a parameter to a procedure and print whether a number is even or odd. Pass per_no of a person as a parameter to a function and return ph_no of that person. For this consider the following relation:
Person (per_no, per_name, per_addr, per_city, ph_no);

[Apr.2010 – 4M] 59. What is Trigger? Explain any two types of triggers.

Chapter 3

TRANSACTION MANAGEMENT

1. Transaction Concept

A transaction is a unit of program execution that accesses and possibly updates various data items. A transaction results from the execution of a user program written in a high level data manipulation language or programming language (*for e.g.* SQL, COBOL, C, PASCAL) and is delimited by statements of the form *begin transaction and end transaction.* The transaction consists of all operations executed between begin and end of the transaction.

For example: A transaction includes read and write operations to access and update the database.

T0
Read (X)
X =X + N
Write (X)
Read (Y)
Y = Y + N
Write (Y)

2. Transaction Properties

4

Oct.12,10 – 2M
Apr.12,11 – 2M
What is Transaction? List Properties of Transaction.

Transactions should have several properties. These are called the ACID properties and they should be enforced by the concurrency control and recovery methods of the DBMS.

*The following are the **ACID** properties of transactions:*

4

Apr.15,Oct.11 – 4M
What is Transaction? Explain ACID properties of Transaction.

Oct.14, Apr.12 – 4M
Explain ACID Properties of Transaction in detail.

1. **Atomicity:** A transaction is treated as a unit of operation. Either all the transactions actions are completed or none of them are. It is also known as 'all-or-nothing property'. If the transaction fails to complete for some reason, the recovery manager must undo any effects of the transaction of the database.

2. **Consistency:** The consistency of a transaction is simply its correctness. It implies that if the database was in a consistent state before the start of a transaction then on termination of a transaction the database will also be in a consistent state.

Each transaction, run by itself with no concurrent execution of other transactions, must preserve the consistency of database. This property must hold for each transaction.

The user's who submits the transaction must ensure that when run to competition by itself against a consistent database instance, the transaction will leave the database in a consistent state.

For example, the fund transfers between bank accounts should not change the total amount of money in the accounts. To transfer money from one account to another, a transaction must debit one account, temporarily leaving the database inconsistent in a global sense, even though the new account balance may satisfy any integrity constraints with respect to the range of acceptable account balances. The user's notion of a consistent database is preserved when the second account is credited with the transferred amount.

3. **Isolation:** It indicates that actions performed by a transaction will be isolated or hidden from outside the transaction until the transaction terminates. It gives the transaction a measure of relative independence.

4. **Durability:** It ensures that once a transaction commits, its results are permanent and cannot be erased from the database. These changes must not be lost because of any failure.

3. Transaction States

A transaction is an atomic unit of work that is either completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts. A transaction that completes the execution successfully is called as a committed transaction. The committed transaction should always take the database to the new consistent state. The changes made by the committed transaction should be permanently stored in the database even if there is any system failure.

1

Oct.2015 – *2M*
What is transaction? State operations performed on transaction.

A database transaction is a **logical unit of database operations** which are executed as a whole to process user requests for retrieving data or updating the database.

There are five states of transaction:

7

Oct.2015 – *4M*
Explain various states of transaction in detail.

Oct.11, 09, Apr.10 – *2M*
List the states of transaction.

Apr.11, Oct.10, 09 – *4M*
What is transaction? Explain states of transaction with diagram.

1. **BEGIN:** The transaction on the database begins by the execution of the first_statement of the transaction that is it becomes active.
2. **ACTIVE:** In this state, the transaction is modifying the database state. In this state, the transaction is performing read or write operations on database state. At the end of this state the transaction will enter into three states, i.e., start, commit, abort or error.
3. **COMMIT:** In the start-commit state, the transaction instructs the DBMS to reflect the change into the database. Once these changes are done in database the transaction is said to be in a commit state.
4. **ROLLBACK:** It may be possible that all changes made by the transaction are not reflected top the database due to any kind of **failure**. In this situation, transaction go to abort or error state. An aborted transaction that made no changes to the database is terminated without the need for **rollback**.

4

Apr.15,12 – *2M*
Define the following terms:
i. Commit
ii. Rollback

Oct.11, 10, 09 – *2M*
Define:
Commit and Rollback

5. **END:** A transaction can end in three different states:
   a. *Successful termination:* A transaction ends after a commit operation.
   b. *Suicidal termination:* A transaction detects an error during its processing and thus aborts and performs a rollback operation.
   c. *Murderous termination:* The operating system or the DBMS can force the transaction to be aborted for any reason.

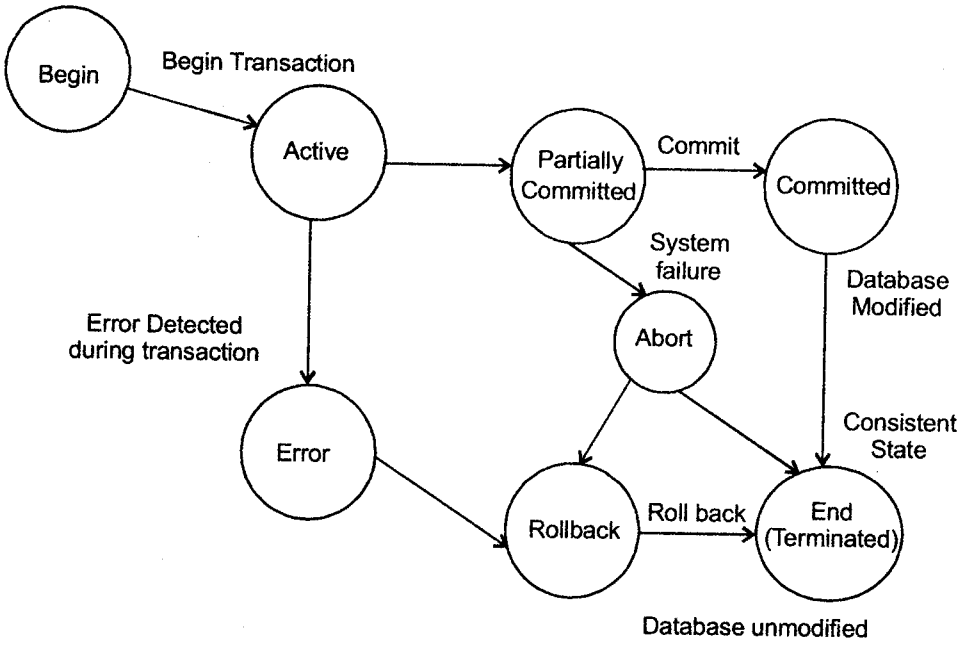


Figure 3.1: Different states of a transaction

4. Concurrent Execution

Transaction processing systems usually allow multiple transactions to run concurrently. Concurrent execution of multiple transactions causes several complications with the consistency of data and may result in some inconsistent database, whereas serial execution of transactions is much easier to implement and maintain the consistency of database.

1

Apr. 2011– *4M*
Explain concurrent execution of transaction with example and advantages of concurrent execution.

T0	T1
Read (X)	
Write (X)	
	Read (Y)
	Write (Y)
Read (Z)	
Write (Z)	

A schedule involving two transactions

The schedule shown above represents an interleaved execution of two transactions. Ensuring transactions isolation while permitting such concurrent execution is difficult, but is necessary for performance reasons.

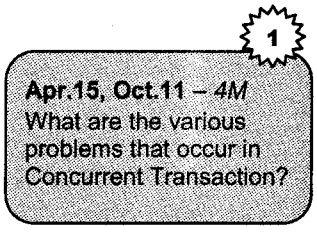
Following are the two advantages of concurrent execution:

1. **Improved resource utilization and throughput:** While one transaction is waiting for a page to be read 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.

   Throughput is the number of transactions executed in a given amount of time. Because of this resource utilization has also increased as the idle time is reduced.

2. **Reduced waiting time:** There is mix of transactions running on a system. Some transaction may be short and some long. If transactions are run serially short transaction may have to wait for a preceding long transaction to complete. But if we run them concurrently then the waiting time of short transaction is reduced. It also reduces the average response time. Average response time is the average time of the transaction to be completed after it has been submitted.

Problems in Concurrent execution



1. A schedule involving consistent, committed transactions could run against a consistent database and leave it in an inconsistent state.
2. Two actions on the same data object conflict if at least one of them is write.
3. The three situations can be described in terms of when the actions of two transactions T0 and T1 conflict with each other.

   i. Reading uncommitted data (WR conflicts): A transaction T1 could read a database object X that has been modified by another transaction T0 which has not yet committed. Such a read is called a **dirty** read.

   For example, consider two transactions T0 and T1 each of which run alone, preserve database consistency. Transactions T0 transfer 200 Rs. from account X to Y and transaction T1 add both X and Y by 8% interest to each account.

T0	T1
Read (X)	Read (X)
X= X -200	X = X + (X * 0.8)
Write (X)	Write (X)
Read (Y)	Read (Y)
Y = Y+ 200	Y = Y + (Y * 0.8)
Write (Y)	Write (Y)
Commit	Commit

Suppose that their actions are interleaved (Transaction T0 and T1 interleaved) so that.

T0	T1
Read (X)	
X= X -200	
Write (X)	
	Read (X)
	X = X + (X * 0.8)
	Write (X)
	Read (Y)
	Y = Y + (Y * 0.8)
	Write (Y)
	Commit
Read (Y)+	
Y = Y+ 200	
Write (Y)	
Commit	

The account transfer transaction T0 deducts 200 Rs. from account X. The interest deposit transaction T1 reads current value of accounts X and Y and adds 8 % inertest to each. The account transfer transaction T1 credits 200 Rs. to account Y.

The problem is that:

a. Transaction T0 may write some value for X that makes database inconsistent.

b. As long as T0 overwrites this value with a correct value of X before committing no harm is done if.

T0 and T1 run in same serial order because transaction T1 would not see the temporary inconsistency.

c. Its interleaved execution can expose this inconsistency and lead to an inconsistent final database state.

ii. *Unrepeatable Reads (RW conflicts):* A transaction T1 could change the value of an object X that has been read by transaction T0 while T1 is still in progress. Transactions T0 reads the value of X again and it is changed by another transaction T1 between the two reads. Transaction T0 and T1 read the same value.

iii. *Overwriting uncommitted Data (WW conflicts):* A transaction T1 could overwrite the value of an object X, which has already been modified by a transaction T0, while T0 is still in progress. Even if T1 does not read the value of X written by T0.

For example. Suppose that Deepak and Sourabh are two employees and their salaries must be equal. Transaction T0 set to salaries to Rs. 2000 and T1 set to salaries Rs. 3000. If we execute serially in the serial order T0 followed by T1 both receive Rs. 3000, the serial order T1 followed T0 both receive Rs. 2000. Notice that neither transaction read a salary value before writing it such write is called blind write.

Schedule

A schedule is a list of actions (reading, writing, aborting, committing) from a set of transactions and the order in which two actions of transactions T appear in a schedule must be the same as the order in which they appear in T.

Schedules represent sequential order in which instructions are executed in the system. S schedule of n transactions T0,T1,T2,.....,Tn is an ordering of the operations of the transactions subject to the constraint that for each transaction Ti that is in S, the operations of Ti in S must appear in the same order in which they occur in Ti. The operations of other transactions Tj can be interleaved with the operations of Ti in S.

For example: Consider the simple banking system which has number of accounts and a set of transactions that access and update those accounts. Consider two transactions T0 and T1 which transfer funds from one account to another. Transactions T0 transfer Rs.200 from account P to account Q. it is defined as,

Transaction T1 transfer 20 percent of the balance from account P to account Q. it is defined as,

T1
Read (P).
Temp =P * 0.2;
Write (P);
Read (Q);
Q= Q + Temp;
Write (Q);

There are two types of schedule:

Oct.2015 – *4M*
What is Schedule? Explain types of schedule with example.

1. **Serial Schedule:** The transactions that are executed from start to end one by one is called serial schedule. It consists of a sequence of instructions from various transactions where the instructions belonging to one single transactions appear together in that schedule.

Schedule 1: A serial schedule T0 followed by T1.

4

Oct.15,11,09, Apr.11 – *2M*
What is Schedule? Give types of Schedule.

Schedule 1

T0	T1
Read (A); A = A -200; Write (A); Read (B); B = B + 200; Write (B);	
	Read (A); Temp = A * 0.2; A = A – temp; Write (A); Read (B); B = B + Temp; Write (B);

Schedule 2: A serial schedule T1 followed by T0.

Schedule 2

T0	T1
	Read (A); Temp = A * 0.2; A = A – temp; Write (A); Read (B); B = B + Temp; Write (B);
Read (A); A = A -200; Write (A); Read (B); B = B + 200; Write (B);	

2. **Concurrent Schedule**: When several transactions are executed concurrently the corresponding schedule is called concurrent schedule.

Schedule 3: A concurrent schedule

Schedule 3

T0	T1
Read (A)	
A = A – 200	
Write (A)	Read (A)
	Temp= A * 0.2
	A = A – Temp
	Write (A)
Read (B)	
B = B + 200	
Write (B)	Read (B)
	B = B + Temp
	Write (B)

Several execution sequences are possible. The schedule 3 will produce the same result as schedule 1. But all concurrent executions may not result in a correct state.

5. Serializability

4

Apr.15,11 – 2M
What is Serializability? List types of Serializability.

Apr.12,10 – 2M
What is Serializability?

Serializability is the generally accepted criterion for correctness for the execution of a given set of transaction. Transaction is considered to be correct if it is serializable i.e. it produce the same result as some serial execution of the same transaction, running them one at a time.

A serializable schedule is called as given interleaved execution of a set of 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 by itself on a consistent database, the resulting database will be consistent.
2. Any serial execution of the transactions is also correct and preserves the consistency of the database.

There are two types Serializablity

1. Conflict Serializability
2. View Serializability

5.1 Conflict Serializability

2

Oct. 10, 09 – *4M*
What is serializability? Explain conflict serializability.

A schedule is conflict serializable if it is conflict equivalent to some serial schedule. Every conflict serializable schedule is serializable.

Consider that T0 and T1 are two transactions and S is schedule for T0 and T1.Ii and Ij are two instructions. If Ii and Ij refer to different data items then Ii and Ij can be executed in any sequence. But if Ii and Ij refer to same data items then the order of two instructions may matter.

Here Ii and Ij can be a read or write operation only. There are 4 conditions that need to be considered.

1. Ii=Read (X) and Ij=Read (X).

   The order of Ii and Ij does not matter because both are reading the data.

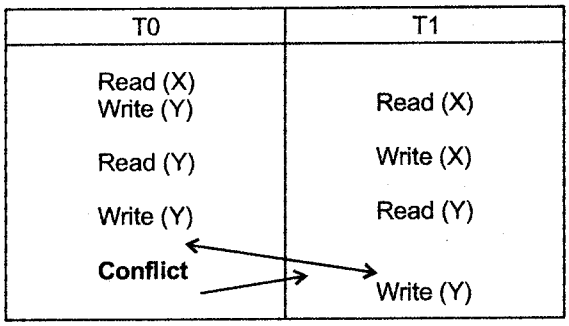
2. Ii=Read (X) and Ij=Write (X). If Ii come before Ij then Ti does not read the value of X that is written by T1 in Ij. If Ij comes before Ii the T0 reads the value of X i.e. written by T1.Thus order Ii and Ij matters.

3. Ii=Write (X) and Ij=Read (X) The order of Ii and Ij matters for the same reasons in step 2.

4. Ii=Write (X) and Ij=Write (X). If both are Write operations their order does not affect either in T0 or T1. But if next operation is Read (X) in S then the order is important. If there is no operation after Ii and Ij in S then the order of Ii and Ij directly affects the final value X in the database that results from schedule.

We say that Ii and Ij conflict if they are operations by different transactions on the same data item and at least one of these instructions is a Write operation.

Let us see the concept of conflict serializability with example of schedule 1. The Write (X) instruction of T0 conflicts with Read (X) instruction of T1. The Write (X) instruction of T1 does not conflict with Read (Y) instruction of T1 because they access different data items.



Schedule 2 is generated after swapping Write (X) instruction of T1 with Read (Y) instruction of T0.

In the same way we could swap:

1. Swap the Read (Y) instruction of T0 with Read (X) instruction of T1.
2. Swap the Write (Y) instruction of T0 with Write (X) instruction of T1.
3. Swap the Write (Y) instruction of T0 with Read (X) instruction of T1.

The Final Result of these Swaps is shown as Schedule S'.

Schedule S'

T0	T1
Read (X) Write (X) Read (Y) Write (Y)	
	Read (X) Write (X) Read (Y) Write (Y)

If a schedule S can be transformed into a schedule S' by a series of Swaps of non-conflicting instructions. We call S and S' are Conflict equivalent.

Testing for Conflict Serializability

To check conflict serializablity is to draw a precedence graph for given schedule and if the cycle is not present in the schedule then we can say that it is conflict serializable schedule.

Precedence graph contain vertices, that is, transactions present in schedule and edges are conflicting instructions in transactions.

Algorithm can be used to test a schedule for conflict serializability. The algorithm looks at only read and write operations in a schedule to a construct precedence graph (or serialization graph) which is

directed graph G=(N,E) that consists of a set of nodes N={ T0,T1,.....,Tn} and a set of directed edges E={e1,e2,....,en)}.

Algorithm: Testing conflict serializability of a schedule S

1. For each transaction Ti participating in schedule S, create a node labeled Ti in the precedence graph.
2. For each case in S where Tj executes a read (X) after Ti executes a write (X), create an edge (Ti → Tj) in the precedence graph.
3. For each case in S where Tj executes a write (X) after Ti executes a read (X), create an edge (Ti → Tj) in the precedence graph.
4. For each case in S where Tj executes a write (X) after Ti executes a write (X), create an edge (Ti → Tj) in the precedence graph.
5. The schedule S is serializable if and only if the precedence graph has no cycles.

For example: Consider following transactions

T0	T1
Read (X)	Read (X)
X =X – N	X = X + M
Write (X)	Write (X)
Read (Y)	
Y = Y + N	
Write (Y)	

Precedence graph

1

Oct.2014 – 2M
What is a precedence graph?

Oct. 2010 – 2M
What is precedence graph? Explain its use.

A precedence graph, also named conflict graph and serializability graph, is used in the study of database theory within the realm of computer science.

The precedence graph for a schedule S contains:

i. A node for each committed transaction in S.

ii. An arc from Ti to Tj if an action of Ti precedes and conflicts with one of Tj's actions.

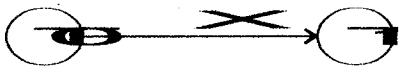
Use of precedence graph

It is used for checking deadlock. If cycle exists in a graph then there is deadlock in the system.

1. Serial schedule 1 transaction T0 followed by T1

Schedule 1

T0	T1
Read (X) X =X – N Write (X) Read (Y) Y = Y + N Write (Y)	
	Read (X) X = X + M Write (X)



Precedence graph for serial schedule 1

2. Serial schedule 2 transaction T1 followed by T0

Schedule 2

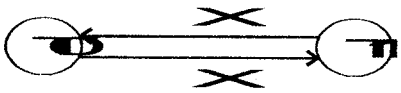
T1	T0
Read (X) X = X + M Write (X)	
	Read (X) X =X – N Write (X) Read (Y) Y = Y + N Write (Y)

Precedence graph for serial schedule 2

3. Non- serial schedule 3 transaction T0 followed by T1

Schedule 3

T0	T1
Read (X)	
X =X – N	
	Read (X)
	X = X + M
Write (X)	
Read (Y)	
	Write (X)
Y = Y + N	
Write (Y)	

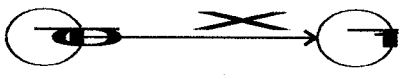


Precedence graph for non-serial schedule 3 (not Serializable)

4. Non- serial schedule 4 transaction T0 followed by T1

Schedule 4

T0	T1
Read (X)	
X =X – N	
Write (X)	
	Read (X)
	X = X + M
	Write (X)
Read (Y)	
Y = Y + N	
Write (Y)	



Precedence graph for non-serial schedule 4(Serializable)

5.2 View Serializability

A schedule S is view serializability if it is view equivalent to a serial schedule. Every conflict serializable schedule is also view serializable but there are view serializable schedules that are not conflict serializable.

Consider two schedules S and S'. The schedules S and S' are said to be view equivalent if the following three conditions hold

1. For each data item X if transaction Ti reads the initial value of X in schedule S, then transaction Ti must in schedule S' also read the initial value of Q.
2. For each data item X, if transaction Ti executes Read (A) in schedule S and that value was produced by transaction, Ti must in schedule S' also read the value of X that was produced by transaction Tj.
3. For each data item X, the transaction that performs the final Write (X) operation in schedule S must perform the final Write (X) operation in schedule S'.

A condition 1 and 2 ensures that same value is read in both the schedules. Condition 3 together with 1 and 2 ensures that final result is same.

For example: Following schedule is a view serializable schedule

T0	T1	T2
Read (X)		
	Write (X)	
Write (X)		
		Write (X)

It is View Equivalent to the serial schedule <T0, T1, T2> since the Read (X) instructions reads the initial value of X in both schedules and T2 performs the final write of X in both schedules.

In above example Transactions T1 and T2 perform Write (X) operations without having performed a Read (X) operation. Write of this sort are called **blind write**. Blind write appear in any view serializable schedule that is not conflict serializable.

6. Recoverability

We have studied that schedules are acceptable from the viewpoint of consistency of the database, assuming implicitly that there are no transaction failures. We now address the effect of transaction failures during concurrent execution.

If transaction Ti fails for whatever reason we need to undo the effect of this transaction to ensure the atomicity property of the transaction. In a system that allows concurrent execution it is necessary also to ensure that any transaction Tj that is dependent on it is also aborted. To achieve this surety we need to place restrictions on the type of schedules permitted in the system. In the following two subsections we address the issue of what schedules are acceptable from the viewpoint described.

6.1 Recoverable Schedule

A recoverable schedule is one where for each pair of transaction Ti and Tj such that Ti reads the data item previously written by Ti the commit operation of Ti appear before commit operation of Tj.

Consider schedule shown in which T0 is a transaction that performs only one instruction Read (A). Suppose that the system allows T1 to commit immediately after executing the Read (A) instruction. Thus T1 commits before T0 does. Now suppose that T0 fails before it commits. Since T1 has read the value of data item X written by T0 we must about T1 to ensure transaction atomicity. However T1 has already committed and cannot be aborted. Thus we have a situation where it is impossible to recover correctly from the failure of T0.

1

Apr.2015 – 2M
Define Recoverable schedule.

T0	T1
Read (X)	
Write (X)	
	Read (X)
Read (Y)	

In the above schedule with the commit happening immediately after the Read (X) instruction is an example of **non –recoverable schedule**, which should not be allowed. Most of the database system requires that all schedules be recoverable.

6.2 Cascadless Schedule

1
Oct.2012 – 2M
What is Cascadless schedule?

Cascadless schedule is recoverable schedule. A single transaction failure leads to a series of transaction rollbacks, this is called cascading rollback. A cascadless schedule is one where for each pair of transaction Ti and such that Tj reads a data item previously written by Ti, the commit operation of appears before the read operation of Tj.

2
Oct. 14,10– 4M
Explain recoverable schedule and cascadless schedule with example.

Consider a transaction T0 writes a value of X that is read by transaction T1. Transactions T1 writes a value of X that is read by transaction T2. Suppose that, at this point T0 fails, T0 must be rolled back. Since T2 is dependent on T1, T2 must be rolled back. This event in which a single transaction failure leads to a serial of transaction rollbacks is called **cascading rollback.**

T0	T1	T2
Read (X) Read (Y) Write (X)		
	Read (X) Write (X)	
		Read (X)

Cascading rollback is undesirable, since it leads to undoing of a significant amount of work. It is desirable to restrict the schedules to those where cascading rollbacks cannot occur.

Solved Examples

1
Oct. 2009– 4M

1. Consider the following transaction. Find out two schedules serializable to serial schedule <T1, T2, T3>

T1	T2	T3
Read (X)	Read (Z)	Read (Y)
X = X + 100	Read (Y)	Read (Z)
Write (X)	Y = Y + Z	Y =Y + 50
Read (Y)	Write (Y)	Write (Y)
Y = Y -100	Read (X)	Z = Z+Y
Write (Y)	X = N –Z	Write (Y)
	Write (X)	

Solution

First we write serial schedule <T1, T2, T3>

T1	T2	T3
Read (X) X = X + 100 Write (X) Read (Y) Y = Y -100 Write (Y)		
	Read (Z) Read (Y) Y = Y + Z Write (Y) Read (X) X = N –Z Write (X)	
		Read (Y) Read (Z) Y =Y + 50 Write (Y) Z = Z+Y Write (Y)

Serializable schedule 1

T1	T2	T3
Read (X) X = X + 100 Write (X)		
	Read (Z)	
Read (Y) Y = Y -100 Write (Y)		
	Read (Y) Y = Y + Z Write (Y)	
		Read (Y)
	Read (X) X = N –Z Write (X)	
		Read (Z) Y =Y + 50 Write (Y) Z = Z+Y Write (Y)

Serializable schedule 2

T1	T2	T3
	Read (Z)	
Read (X) X = X + 100 Write (X) Read (Y) Y = Y -100 Write (Y)		
	Read (Y) Y = Y + Z Write (Y)	
		Read (Y) Read (Z) Y =Y + 50 Write (Y)
	Read (X) X = N –Z Write (X)	
		Z = Z+Y Write (Y)

---

2. Consider the following transaction

a. Find out non-serial schedule, which is serializable to serial schedule <T1, T2, T3>

b. Find out non-serial schedule, which is serializable to serial schedule <T3, T1, T2>.

T1	T2	T3
Read (X) Read (Z) X = n + Z Write (X)	Read (Z) Z = Z + 10 Read (Y) Y = Y + Z Write (Z) Write (Y)	Read (X) Read (Y) Y =Y - X Write (Y)

Solution

First we write serial schedule <T1, T2, T3>

T1	T2	T3
Read (X) Read (Z) X = n + Z Write (X)		
	Read (Z) Z = Z + 10 Read (Y) Y = Y + Z Write (Z) Write (Y)	
		Read (X) Read (Y) Y =Y - X Write (Y)

a.

T1	T2	T3
Read (X)		
Read (Z)		
	Read (Z)	
	Z = Z + 10	
X = n + Z		
Write (X)		
	Read (Y)	
	Y = Y + Z	Read (X)
	Write (Z)	
	Write (Y)	
		Read (Y)
		Y =Y - X
		Write (Y)

Non-serial schedule for serial <T1, T2, T3>

T1	T2	T3
Read (X)		
Read (Z)		
X = n + Z		
	Read (Z)	
	Z = Z + 10	
	Read (Y)	
Write (X)		
	Y = Y + Z	
	Write (Z)	
		Read (X)
	Write (Y)	
		Read (Y)
		Y =Y - X
		Write (Y)

b. First we write serial schedule <T3, T1, T2>

T3	T1	T2
Read (X)		
Read (Y)		
Y =Y - X		
Write (Y)		
	Read (X)	
	Read (Z)	
	X = n + Z	
	Write (X)	
		Read (Z)
		Z = Z + 10
		Read (Y)
		Y = Y + Z
		Write (Z)
		Write (Y)

Non-serial schedule for serial <T3, T1, T2>

T3	T1	T2
Read (X)		
	Read (X)	
	Read (Z)	
		Read (Z)
		Z = Z + 10
		Read (Y)
Read (Y)		
Y =Y - X		
Write (Y)		
		Y = Y + Z
		Write (Z)
		Write (Y)
	X = n + Z	
	Write (X)	

3. Consider the following transaction. Give at least 2 serial schedules.

T0	T1
Read (A)	Read (A)
A =A -50	t = A * 0.1
Write (A)	A=A-t
Read (B)	Write (A)
B =B + 50	Read (B)
Write (B)	B = B +t
	Write (B)

Solution

Suppose we consider Transaction T0 followed by Transaction T1.

Serial Schedule 1

T0	T1
Read (A)	
A =A -50	
Write (A)	
Read (B)	
B =B + 50	
Write (B)	
	Read (A)
	t = A * 0.1
	A=A-t
	Write (A)
	Read (B)
	B = B +t
	Write (B)

Suppose we consider Transaction T1 followed by Transaction T0

Serial Schedule 2

T0	T1
	Read (A) t = A * 0.1 A=A-t Write (A) Read (B) B = B +t Write (B
Read (A) A =A -50 Write (A) Read (B) B =B + 50 Write (B)	

4. Consider the following transaction. Find out concurrent schedule, which is serializable to serial schedule <T0, T1>

T0	T1
Read (A) A =A -50 Write (A) Read (B) B =B + 50 Write (B)	Read (A) t = A * 0.1 A=A-t Write (A) Read (B) B = B +t Write (B)

Solution

Following schedule is called as concurrent, which is serializable

T0	T1
Read (A) A =A -50 Write (A)	
	Read (A) t = A * 0.1 A=A-t Write (A)
Read (B) B =B + 50 Write (B)	
	Read (B) B = B +t Write (B)

5. Consider following transactions. Give two Non-serial Schedules that are serializable:

1

Oct. 2012– *4M*

T_1	T_2
Read(X)	Read(Y)
X=X-1000	Y=Y+5000
Write(X)	Write(Y)
Read(Y)	Read(Z)
Y=Y+1000	Z=Z+5000
Write(Y)	Write(Z)

Solution

T_1	T_2
Read(X)	
X=X-1000	
Write(X)	
	Read(Y)
	Y=Y+5000
	Write(Y)
Read(Y)	
Y=Y+1000	
Write(Y)	
	Read(Z)
	Z=Z+5000
	Write(Z)

T_1	T_2
Read(X)	
X=X-1000	
Write(X)	
	Read(Y)
	Y=Y+5000
	Write(Y)
	Read(Z)
	Z=Z+5000
	Write(Z)
Read(Y)	
Y=Y+1000	
Write(Y)	

1

Oct. 2012, – 4M

6. Consider the following Transactions. Give two Non-serial schedules that are serializable:

T_1	T_2
Read(X)	Read(Y)
X=X+10	Y=Y-10
Write(X)	Write(Y)
Read(Y)	Read(Z)
Y=Y+20	Z=Z-20
Write(Y)	Write(Z)
Read(Z)	
Z=Z+30	
Write(Z)	

Solution

Following are the two schedules which are serializable.

T_1	T_2
Read(X)	
X=X+10	
Write(X)	
	Read(Y)
	Y=Y-10
	Write(Y)
Read(Y)	
Y=Y+20	
Write(Y)	
	Read(Z)
	Z=Z-20
	Write(Z)
Read(Z)	
Z=Z+30	
Write(Z)	

T_1	T_2
Read(X)	
X=X+10	
Write(X)	
	Read(Y)
	Y=Y-10
	Write(Y)
Read(Y)	
Y=Y+20	
Write(Y)	
Read(Z)	
Z=Z+30	
Write(Z)	
	Read(Z)
	Z=Z-20
	Write(Z)

7. Consider the following Non-serial Schedule

3

Apr. 12, Oct.09, 11– *4M*

T_1	T_2
Read(X)	
X: = X – N	
	Read(X)
	X: = X + N
Write(X)	
Read(Y)	
	Write(X)
Y: = Y + N	
Write(Y)	

Is the schedule serializable to a serial schedule <T_1, T_2 >?

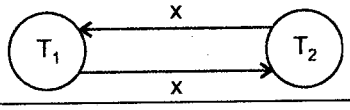
Solution

i. First we draw serial schedule T_1, T_2

T_1	T_2
Read(X)	
X=X-N	
Write(X)	
Read(Y)	
Y=Y+N	
Write(Y)	
	Read(X)
	X=X+N
	Write(X)

In the given non serial schedule Read(X) of T_2 reads the original value of X where in serial schedule <T_1,T_2>read(X) of T_2 reads the value updated by Write(X) of T_1. Thus the order of conflicting instructions is not same in both the schedules. Thus, the given non serial schedule is not serializable.

ii. We can draw precedence graph for given non serial schedule. There exists a cycle in the graph so given non serial schedule is not serializable.



8. Consider the following transaction.

1

Apr. 2012– *4M*

T_0	T_1
Read(A)	Read(A)
A: = A – 70	t: = A * 0.1
Write (A)	A = A – t
Read(B)	Write (A)
B: = B + 70	Read(B)
Write(B)	B = B + t
	Write(B)

Give at least two serial schedules.

Solution

Non serial schedule S_1:

T_0	T_1
Read(A)	
A=A-70	
Write(A)	
	Read(A)
	t=A*0.1
	A=A-t
	Write(A)
Read(B)	
B=B+70	
Write(B)	
	Read(B)
	B=B+t
	Write(B)

Non serial schedule S_2:

T_0	T_1
Read(A)	
A=A-70	
Write(A)	
	Read(A)
Read(B)	
	t=A*0.1
	A=A-t
	Write(A)
B=B+70	
Write(B)	
	Read(B)
	B=B+t
	Write(B)

1 Oct. 2011 – 4M

9. Consider the following transactions. Find out two concurrent schedule, which will be serializable to serial schedule $<T_1,T_2>$:

T_1	T_2
Read (A)	Read (B)
A:= A – 70	B:= B + 10
Write (A)	Write (B)
Read (B)	Read (C)
B:= B + 70	C:= C + 50
Write (B)	Write (C)

Solution

The schedules S_1 and S_2 shown below can be 2 non serial schedules that are serializable to given schedule.

Following is schedule S_1

T_1	T_2
Read (A)	
A:= A – 70	
Write (A)	
	Read (B)
	B:= B + 10
	Write (B)
Read (B)	
B:= B + 70	
Write (B)	
	Read (C)
	C:= C + 50
	Write (C)

Following is schedule S_2

T_1	T_2
	Read (B)
	B:=B + 10
	Write (B)
	Read (C)
	C:= C+ 50
	Write (C)
Read (A)	
A:=A – 70	
Write (A)	
Read (B)	
B:= B + 70	
Write (B)	

10. Consider the following transaction. Give two non serial schedules that are serializable.

1

Apr. 2011 – *4M*

T_1	T_2
Read(A)	Read(X)
A=A+5	x=x-10
Write(A)	Write(X)
Read(B)	Read(B)
Read(C)	B=B-20
B=B+10	Write(B)
Write(B)	
C=C+15	
Write(C)	

Solution

The schedules S1 and S2 shown below can be 2 non serial schedules that are serializable to given schedule.

Following is schedule S_1:

T_1	T_2
	Read(X)
	X=X-10
	Write(X)
	Read(B)
	B=B-20
	Write(B)
Read(A)	
A=A+5	
Write(A)	
Read(B)	
Read(C)	
B=B+10	
Write(B)	
C=C+15	
Write(C)	

Following is schedule S_2:

T_1	T_2
Read(A)	
A=A+5	
Write(A)	
	Read(X)
	X=X-10
	Write(X)
Read(B)	
Read(C)	
B=B+10	
Write(B)	
	Read(B)
	B=B-20
	Write(B)
C=C+15	
Write(C)	

1 Apr. 2011 – 4M

11. Consider the following transaction. Find out a non serial schedule which is serializable to serial schedule <T_1,T_2,T_3>

T_1	T_2	T_3
Read(X)	Read(X)	Read(Z)
Read(Y)	Read(Z)	Read(Y)
Y=Y-X	X=X+Z	Y=Y+Z
Write(Y)	Write(X)	Write(Y)

Solution

Non serial schedule S_1:

T_1	T_2	T_3
Read(X)		
	Read(X)	
	Read(Z)	
	X=X+Z	
	Write(X)	
Read(Y)		
Y=Y-X		
Write(Y)		
		Read(Z)
		Read(Y)
		Y=Y+Z
		Write(Y)

Non serial schedule S_2:

T_1	T_2	T_3
Read(X)		
	Read(X)	
		Read(Z)
Read(Y)		
Y=Y-X		
Write(Y)		
		Read(Y)
		Y=Y+Z
		Write(Y)
	Read(Z)	
	X=X+Z	
	Write(X)	

12. Consider the following transaction. Give two non-serial schedules that are serializable.

T_1	T_2
read (x)	read (x)
x = x – m	x = x + n
write (x)	write (x)
read (y)	
y = y + m	
write (y)	

Solution

First schedule S_1

T_1	T_2
read (x)	
x = x - m	
write (x)	
	read (x)
	x = x + n
	write (x)
read (y)	
y = y + m	
write (y)	

Second schedule S_2

T_1	T_2
read (x)	
x = x - m	
write (x)	
	read (x)
	x = x + n
read (y)	
y = y + m	
write (y)	
	write (x)

13. Consider the following transaction. Find out a non serial schedule which is serializable to serial schedule <T_1, T_2, T_3>:

T_1	T_2	T_3
read (x)	read (z)	read (y)
x = x + 100	read (y)	read (z)
write (x)	y = y + z	y = y + 50
read (y)	write (y)	write (y)
y = y – 100	read (X)	z = z + y
write (y)	x = n – z	write (y)
	write (x)	

Solution

Two non serial schedules S_1, S_2 that are serializable to above serial schedule <T_1, T_2, T_3> are as follows.

Non serial schedule S_1

T_1	T_2	T_3
read(x)		
x=x+100		
write(x)		
	read(z)	
	read(x)	
	x=x-z	
	write(x)	
read(y)		
y=y-100		
write(z)		
	read(y)	
	y=y+z	
	write(y)	
		read(y)
		read(z)
		y=y+50
		write(y)
		z=z+y
		write(z)

No serial schedule S_2

T_1	T_2	T_3
read(x)	read(z)	
x=x+100		
write(x)		
read(y)		
y=y-100		
write(z)		
	read(y)	
	y=y+z	
	write(y)	
		read(y)
		read(z)
		y=y+50
		write(y)
	read(x)	
	x=x-z	
		z=z+y
		write(z)
	write(x)	

14. Consider the following transactions.

T1	T2
Read(x)	Read(y)
x = x + 1000	y = y – 500
Write(x)	Write(y)
Read(y)	Read(z)
y = y + 1000	z = z – 500
Write(y)	Write(z)

1

Oct. 2014 – *4M*

Give two non serial schedules that are serializable.

Solution

T1	T2
Read(x)	
x = x + 1000	
Write(x)	
	Read(y)
	y = y – 500
	Write(y)
Read(y)	
y = y + 1000	
Write(y)	
	Read(z)
	z = z – 500
	Write(z)

T1	T2
Read(x)	
x = x + 1000	
Write(x)	
	Read(y)
	y = y – 500
	Write(y)
	Read(z)
	z = z – 500
	Write(z)
Read(y)	
y = y + 1000	
Write(z)	

15. Consider the following transactions.

T1	T2
Read(x)	**Read(z)**
Read (z)	**z = z+ 100**
x = x + z	**Write(z)**
Write(x)	**Read(y)**
	y = y + 200
	Write(y)

1

Oct. 2014 – *4M*

Give two non serial schedules that are serializable.

Solution

T1	T2
Read(x)	
Read(z)	
x = x + z	
	Read(z)
	z = z+ 100
	Write(z)
	Read(y)
	y = y + 200
	Write (y)
Write (x)	

T1	T2
	Read(z)
	z = z + 100
	Write(z)
Read(x)	
Read(z)	
	Read(y)
	y = y + 200
	Write (y)
x = x + z	
Write(x)	

PU Questions

2 Marks

Oct.2015 – 2M
1. What is transaction? State operations performed on transaction.

[Apr.2015 – 2M]
2. Define Recoverable schedule.

[Apr.15,12 – 2M]
3. Define: i. commit ii. Rollback

[Apr.15,11– 2M]
4. What is serializability? List the types of serializability.

[Oct.14,11,09,Apr.11 – 2M]
5. What is schedule? Give types of schedule.

[Oct.2014 – 2M]
6. What is a precedence graph?

[Oct.2012 – 2M]
7. What is Cascadeless schedule?

[Oct.12,10,Apr.12,11 – 2M]
8. What is Transaction? List Properties of Transaction.

[Apr.12,10 – 2M]
9. What is Serializability?

[Oct.11,Apr.10,09 – 2M]
10. List the states of transaction.

4 Marks

[Oct.2015 – 4M]
1. Consider the following transactions. Find out tow non-serial schedules that are serializable.

T_1	T_2
Read (x)	Read (z)
x = x + 10	Read (y)
Write (x)	y = y – z
Read (y)	Write (y)
y = y – 10	Read (x)
Write (y)	x = x + z
	Write (x)

[Oct.2015 – 4M]
2. Consider the following non-serial schedules. Is this schedule serializable?

T_1	T_2	T_3
Read (A)		
	Read (A)	
Write (A)		
		Read (A)
		Write (A)

[Oct.2015 – 4M]
3. Explain various states of transaction in detail.

[Oct.2015 – 4M]
4. What is Schedule? Explain types of schedule with example.

5. What are the various problems that occur in concurrent transaction?

[Apr.15,Oct.11 – 4M]
6. What is transaction? Explain ACID properties of transaction.

[Apr.15,Oct.11 – 4M]
7. 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)	

8. Consider the following transactions. Give two non-serial schedules that are serializable. **[Apr.2015 – 4M]**

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)	

9. Consider the following transactions. Give two non-serial schedules that are serializable. **[Apr.2015 – 4M]**

T_1	T_2
Read (A)	Read (A)
A = A + 1000	A = A – 1000
Write (A)	Write (A)
Read (BC)	Read (B)
C = C – 1000	B = B – 1000
Write (C)	Write (B)
Read (B)	
B = B + 1000	
Write (B)	

10. Explain ACID properties of transaction in detail. **[Oct.14,Apr.12 – 4M]**
11. Explain recoverable schedule and cascadeless schedule with example. **[Oct.14,10 – 4M]**
12. Consider the following transactions. **[Oct.2014 – 4M]**

T1	T2
Read(x)	Read(y)
x = x + 1000	y = y – 500
Write(x)	Write(y)
Read(y)	Read(z)
y = y + 1000	z = z – 500
Write(y)	Write(z)

Give two non serial schedules that are serializable.

13. Consider the following transactions. **[Oct.2014 – 4M]**

T1	T2
Read(x)	Read(z)
Read (z)	z = z+ 100
x = x + z	Write(z)
Write(x)	Read(y)
	y = y + 200
	Write(y)

Give two non serial schedules that are serializable.

14. Consider following the transactions. Give two Non-serial Schedules that are serializable: **[Oct.2012 – 4M]**

T_1	T_2
Read(X)	Read(Y)
X=X-1000	Y=Y+5000
Write(X)	Write(Y)
Read(Y)	Read(Z)
Y=Y+1000	Z=Z+5000
Write(Y)	Write(Z)

[Oct.2012 – 4M] 15. Consider the following Transactions. Give two Non-serial schedules that are serializable:

T_1	T_2
Read(X)	Read(Y)
X=X+10	Y=Y-10
Write(X)	Write(Y)
Read(Y)	Read(Z)
Y=Y+20	Z=Z-20
Write(Y)	Write(Z)
Read(Z)	
Z=Z+30	
Write(Z)	

[Apr.12,Oct.11,09–4M] 16. Consider the following Non-serial Schedule

T_1	T_2
Read(X)	
X: = X – N	
	Read(X)
	X: = X + N
Write(X)	
Read(Y)	
	Write(X)
Y: = Y + N	
Write(Y)	

Is the schedule serializable to a serial schedule $<T_1, T_2>$?

[Apr.2012 – 4M] 17. Consider the following transaction.

T_0	T_1
Read(A)	Read(A)
A: = A – 70	t: = A * 0.1
Write (A)	A = A – t
Read(B)	Write (A)
B: = B + 70	Read(B)
Write(B)	B = B + t
	Write(B)

Give atleast two serial schedules.

[Oct.2011 – 4M] 18. Explain different types of Failure.

[Oct.2011 – 4M] 19. Consider the following transactions. Find out two concurrent schedule, which will be serializable to serial schedule $<T_1,T_2>$:

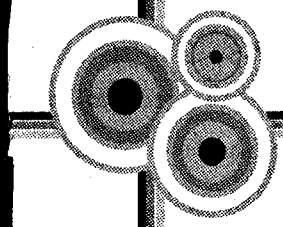
T_1	T_2
Read (A)	Read (B)
A:= A –70	B:= B + 10
Write (A)	Write (B)
Read (B)	Read (C)
B:= B +70	C:= C +50
Write (B)	Write (C)

[Apr.11, Oct.10,09 – 4M] 20. What is transaction? Explain states of transaction with diagram.

[Apr.2011 – 4M] 21. Explain concurrent execution of transaction with example and advantages of concurrent execution.

Chapter 4

CONCURRENCY CONTROL



1. Concurrency Control

Concurrent execution of multiple transactions causes several complications with the consistency of data and may result in some inconsistent database whereas serial execution of transactions is much easier to implement and maintain the consistency of database.

There are **two reasons** for using concurrency.

1. A transaction consists of multiple steps. Some involve I/O activity (Read/Write) others involve CPU activity. The CPU and disks in computers can operate in parallel. Therefore, I/O activity can be done in parallel with processing at CPU. This can be used to run multiple transactions in parallel. This concurrent execution of transactions increases the throughput of system i.e. it will increase the number of transactions that can be executed in a given amount of time.
2. The processor running on the system may differ in execution time i.e. some short and some long transactions. If transactions are running serially a short transaction may have to wait for a preceding long transaction to complete, which can lead to unpredictable delays in running a transaction. If the transactions are on different parts of database, it is better to run them concurrently, sharing the CPU cycles and disk accesses among them. It reduces the unpredictable delay in running the transaction.

2. Lock Based Protocols

One way to ensure serializability is to require that data items be accessed in a mutually exclusive manner, that is, while one transaction is accessing a data item, no other transaction can modify that data item. The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item.

2.1 Locks

3

Oct.12, 09, Apr.11 – 2M
Define Lock. List different types of Lock.

A lock is a variable associated with the data item that describes the status of the item with respect to possible operations that can be applied to the item. Generally there is one lock for each data item in the database, which is, used to synchronise the access by concurrent transactions to the database. There are various modes in which a data item may be locked.

Types of Lock

1. **Binary Locks:** A binary lock can have two stages or values.
   - i. Lock state (1)
   - ii. Unlock state (0)

   If the value of the lock on X is 1 then item X cannot be accessed by a database operation that requests the item. If the value of the lock on X is 0 the item can be accessed by a database when requested.

   Two operations lock item and unlocked item are used in binary locking.

   A transaction requests access to an item X by first requesting a lock (X) operation. If X is locked by another transaction then the transaction is forced to wait and if X is not already locked by any other transaction then the transaction is allowed to access item X. When the transaction finishes it's processing on item X it should unlock the item X.

   For a binary locking scheme every transaction must follow the rules below:
   - i. A transaction T must issue the operation lock (X) before any read (X) or write (X) operations are performed in T.

ii. Transactions T must issue the operation unlock (X) after all read (X) and write (X) operations are completed in T.

iii. A transaction T will not issue a lock (X) operation if it already holds the lock on X.

iv. A transaction T will not issue an unlock (X) operation unless it already holds the lock on X.

Lock and unlock operations must be implemented as indivisible units i.e. no interleaving should be allowed once a lock or unlock operation is started until the operation terminates or the transaction waits.

2. Shared or Exclusive Lock

Shared: If a transaction Ti has obtained a shared-mode lock (denoted by S) on item Q, then Ti can read, but cannot write Q. The shared lock is also called a *read lock.*

The intention of this mode of locking is to ensure that the data item does not undergo any modifications while it is locked in this mode. Any number of transactions can concurrently lock and access a data-item in the shared mode, but none of these transactions can modify the data-item.

Exclusive: If a transaction Ti has obtained an exclusive-mode lock (denoted by X) on item Q, then Ti can both read and write Q. The exclusive lock is also called an update or a *write lock.*

The intention of this mode of locking is to provide exclusive use of the data-item to one transaction. If a transaction T locks a data-item Q in an exclusive mode, no other transaction can access Q, not even to read Q, until the lock is released by transaction T.

We require that every transaction request a lock in an appropriate mode on data item Q, depending on the types of operations that it will perform on Q. The transaction makes the request to the concurrency control manager. The transaction can proceed with the operation only after concurrency control manager grants the lock to the transaction.

Share/Exclusive locking scheme must follow the rules:

i. A transaction T must issue the operation read lock (X) or write lock (X) before any read (X) operation is performed in T.

ii. A transaction T must issue the operation write lock (X) before any write (X) operation is performed in T.

iii. A transaction T must issue the operation unlock (X) after all read (X) and write (X) operations are completed in T.

iv. A transition T will not issue read lock (X) operation if it already holds a read lock (X) or a write lock (X).

v. A transaction T will not issue write lock (X) operation if it already holds a read lock (X) or write lock (X)

vi. A transaction T will not issue an unlock (X) operation unless it already holds a read lock (X) or write lock (X).

2.2 Granting of Locks

When a transaction requests a lock on a data-item in a particular mode, and no other transaction has a lock on the same data-item in a conflicting mode, the lock can be granted. However, care must be taken to avoid the following scenario. Suppose a transaction T2 has a shared mode lock on a data-item and another transaction T1 request an exclusive-mode lock on the data-item. Clearly T1 has to wait for T2 to release the shared-mode lock. Meanwhile, a transaction T3 may request a shared-mode lock on the same data-item. The lock request is compatible with the lock granted to T2, so T3 may be granted the shared mode-lock. At this point T2 may release the lock, but still T1 has to wait for T3 to finish. But again, there may be a new transaction T4 that requests a shared-mode lock on the same data-item, and is granted the lock before T3 releases it. In fact, it is possible that there is a sequence of transactions that each requests a shared-mode lock on the data-item, and each transaction releases the lock a short while after it is granted, but T1 never gets the exclusive-mode lock on the data-item. The transaction T1 may never make progress, and is said to be starved.

We can avoid starvation of transactions by granting locks in the following manner: When transaction T1 requests a lock on a data-item Q in a particular mode M, the concurrency-control manager grants the lock provided that

1. There is no other transaction holding a lock on Q in a mode that conflict with M.
2. There is no other transaction that is waiting for a lock on Q and that made its lock request before Ti.

Thus, a lock request that is made later will never block a lock request.

2.3 Two-Phase Locking Protocol

4

Oct.14,09 Apr. 12,– *4M*
Explain two phase locking protocol with example.

Oct. 2011 – *4M*
Explain strict two phase locking protocol with example.

One protocol that ensures serializability is the two-phase locking protocol. This protocol requires that each transaction issue 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.

In the beginning, a transaction is in the growing phase. The transaction acquires locks as needed. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock requests. Two-phase locking protocol ensures conflict serializability.

5

Oct. 14,12,11
Apr.13 – *2M*
Define Growing Phase and Shrinking Phase.

Oct. 2010 – *2M*
Define Growing Phase.

Consider any transaction. 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. At this moment transactions can be ordered according to their lock points. This ordering is the serializability ordering for the transactions.

1

Oct.2009 – *4M*
Define following terms:
i. Upgrading and Downgrading
ii. Growing Phase
iii. Shrinking Phase
iv. Lock Point

Two phase locking does not ensure freedom from deadlock.

Example: Following two transactions are two phase transactions

T1	T2
lock –X (B)	lock – S (A)
lock –X (B)	read (A)
B = B – 50	lock - S (B)
write (B)	read (B)
lock X (A)	display (A +B)
A = A + 50	unlock (A)
Write (A)	unlock (B)
unlock (B)	
unlock (A)	

Transaction T1 and T2 are two phase but they are **deadlocks.**

T1	T2
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)

(S – Shared, X- Exclusive locks)

Here T1 has a X lock and T2 wants a S on B and T2 has a S lock and T1 wants a X on A.

The two-phase locking protocol also has the cascading rollback. The cascading rollback can be avoided by the variation of two-phase locking protocol called **strict two-phase locking protocol.** It has two phases as it also needs that all exclusive mode locks taken by a transaction should be held until that transaction commits, because of which cascading rollbacks does not occur.

Another variation of two-phase locking protocol **is rigorous two-phase locking protocol**. It has two phases. In addition it requires that all locks be held until the transaction commits. Most database systems implement either strict or rigorous two phase locking.

Consider the following two transactions T1 and T2

T1
Read (X1)
Read (X2)
.....
....
Read (Xn)
Write (X1)

T2
Read(X1)
Read(X2)
Display (X1 + X2)

If we use the two-phase locking protocol, then T1 must lock X1 in exclusive mode for any concurrent execution of both transactions to a serial execution. But T1 needs an exclusive lock on X1 only at the end of its execution, when it writes X1. Therefore, if T1 could initially lock X1 in shared mode, and then could later change the lock to exclusive mode, we could get more concurrency, since T1 and T2 could access X1 and X2 simultaneously.

This observation leads to the basic two-phase locking protocol, in which lock conversion are allowed, **upgrading** a shared lock to an exclusive lock, and **downgrading** an exclusive lock to a shared lock. Lock conversion cannot be allowed at any time. Upgrading can take place in only the *growing phase*, whereas downgrading can take place in only the *shrinking phase.*

i. **Upgrading:** In growing phase of 2PL, transaction may issue a lock on database item. Transaction can acquire shared lock on database item. Transaction can also acquire exclusive

lock on database item. Converting shared lock on exclusive lock on same database item is called as upgrading.

ii. **Downgrading:** In shrinking phase of 2PL, transaction release a lock on database item. Transaction can release shared lock on database item. Transaction can also release exclusive lock on database item. Converting exclusive lock on shared lock on same database item is called as downgrading.

iii. **Lock point**: In 2phase locking protocol, required data base items are locked in advance and operations on the data base items are performed. So the "Lock Point" is when all locks are held for the whole transaction in growing phase. Following diagram shows the lock point.

Consider the *following example* for upgrade and downgrade.

T1	T2
lock –S (X1)	
	lock – S (X1)
lock –S (X2)	
	lock – S (X2)
lock –S (X3)	
lock –S (X4)	
	unlock (X1)
	unlock (X2)
lock –S (Xn)	
upgrade (X1)	

Example for upgrade and downgrade

1

Apr.2015 – *4M*
Define:
i. upgrading
ii. downgrading

The database systems automatically generate the lock and unlock instructions for a transaction on the basis of read and write requests from the transaction.

1. When a transaction Ti issues a read (Q) operation the system issues a lock-S (Q) instruction followed by the read (Q) instruction.
2. When Ti issues a write (Q) operation the system checks if Ti already holds a shared lock on Q. If it does the system issues an upgrade (Q) instruction followed by the write (Q) instruction. Otherwise the system issues a lock X (Q) instruction followed by the write (Q) instruction.
3. All locks obtained by a transaction are unlocked after that transaction commits or aborts.

For a set of transactions, there may be conflict-serializable schedules that cannot be obtained through the two-phase locking protocol. However, to obtain conflict-serializable schedules through non-two-phase locking protocols, we need either to have additional information about the transaction or to impose some structure or ordering on the set of data items in the database. In the absence of such

information, two-phase locking is necessary for conflict serializability. If Ti is a non-two-phase transaction, it is always possible to find another transaction Tj that is two phases so that there is schedule possible for Ti and Tj that is not conflict serializable.

Strict two-phase locking and rigorous two-phase locking (with lock conversions) are used extensively in commercial data base systems.

3. Timestamp-Based Protocols

The locking protocols that have been described thus far determine the order between every pair of conflicting transactions at execution time by the first lock that both members of the pair request that involves incompatible modes. Another method for determining the serializability order is to select an ordering among transactions in advance. The most common method for doing so is to use a timestamp-ordering scheme.

3.1 Timestamp

A timestamp is a unique identifier created by the DBMS to identify a transaction. Timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be thought of as the *transaction start time*. We will refer to the timestamp of transaction T as TS (T). Concurrency control techniques based on timestamp ordering do not use locks, thus *deadlocks cannot occur.*

With each transaction Ti in the system, we associate a unique fixed timestamp, denoted by TS (Ti). This timestamp is assigned by the database system before the transaction Ti starts execution. If a transaction Ti has been assigned timestamp TS (Ti), and a new transaction Tj enters the system, then TS (Ti) <TS (Tj). There are two simple methods for implementing this scheme:

1. Use the value of the system clock as the timestamp; that is, a transaction's time stamp is equal to the value of the clock when the transaction enters the system.
2. Use a logical counter that is incremented after a new timestamp has been assigned; that is, a transaction's timestamp is equal to the value of the counter when the transaction enters the system.

The timestamps of the transactions determine the serilizability order. Thus, if TS (Ti) <TS (Tj), then the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction Ti appears before transaction Tj.

To implement this scheme, we associate with each data item Q two timestamp values:

1. W-timestamp (Q) denotes the largest timestamp of any transaction that executed write (Q) successfully.
2. R-timestamp (Q) denotes the largest timestamp of any transaction that executed read (Q) successfully.

1
Oct.2014 – *2M*
Define:
i. W-timestamp
ii. R-timestamp

These timestamps are updated whenever a new read (Q) or write (Q) instruction is executed.

1
Apr.2015 – *4M*
Explain Timestamp ordering protocol.

3.2 Timestamp-Ordering Protocol

The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows:

1. Suppose that transaction Ti issues read (Q).
   a. If TS (Ti) < W-timestamp (Q), then Ti needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and Ti is rolled back.
   b. If TS (Ti) >= W-timestamp (Q), then the read operation is executed, and R-timestamp (Q) is set to the maximum of R-timestamp (Q) and TS (Ti)
2. Suppose that transaction Ti issues write (Q).
   a. If TS (Ti) < R-timestamp (Q), then the value of Q that Ti is producing was needed previously, and the system assumed that value would never be produced. Hence, the system rejects the write operation and rolls Ti back.
   b. If TS (Ti) < W-timestamp (Q), then Ti is attempting to write an obsolete value of Q. Hence, the system rejects this write operation and rolls Ti back.
   c. Otherwise, the system executes the write operation and sets W-timestamp (Q) to TS (Ti).

If a transaction Ti is rolled back by the concurrency-control scheme as result of issuance of either a read or write operation, the system assigns it a new timestamp and restarts it.

To illustrate this protocol we consider transactions T0 and T1. Transaction T0 displays the contents of Account X and Y.

T0
Read (Y)
Read (X)
Display (X + Y)

Transaction T1 transfers Rs.50 from account X and then displays the contents of both.

T1
Read (Y)
Y = Y – 50
Write (Y)
Read (X)
X= X + 50
Write (X)
Display (X+ Y)

In presenting schedules under the timestamp protocol we shall assume that a transaction is assigned a timestamp immediately before its first instruction. Thus in ***schedule 1*** TS (T0) < TS (T1) and the schedule is possible under the timestamp protocol.

We note that preceding execution can also be produced by the two-phase locking protocol. There are however schedules that are possible under the two-phase locking protocol, but are not possible under the timestamp protocol and vice versa.

Schedule 1

T0	T1
Read (Y)	
	Read (Y)
	Y = Y -50
	Write (Y)
Read (X)	
	Read (X)
Display (X +Y)	
	X = X +50
	Write (X)
	Display (X + Y)

The timestamp ordering protocol ensures conflict serializablity. This is because conflicting operations are processed in timestamp order. The protocol ensures freedom from deadlock since no transaction ever waits.

The protocol can generate schedules that are not recoverable; however it can be extended to make the schedule recoverable.

It can be done in following ways

1. Recoverability and cascadlessness can be ensured by performing all writes together at the end of the transactions. The writes must be atomic i.e. while the writes are in progress no transaction is permitted to access any of the data items that have been written.
2. Recoverability and cascadlessness can also be guaranteed by using a limited form of locking whereby reads of uncommitted items are postponed until the transaction that updated the item commits.
3. Recoverability alone can be ensured by tracking uncommitted writes and allowing a transaction Ti to commit only after the commit of any transaction that wrote a value of Ti read.

3.3 Thomas write rule

A modification of the basic timestamp ordering protocol known as Thomas write rule.

Consider the following schedule.

Schedule 1

T1	T2
read (Q)	
	write (Q)
write (Q)	

Let us consider schedule 1 and apply the timestamp ordering protocol.

We assume that TS (T1) < TS (T2). The read (Q) operation of T1 succeeds, as does the write (Q) operation of T2. When T1 attempts its write (Q) operation, we find that TS (T1) < W-timestamp (Q), since W-timestamp (Q) = TS (T2). Thus, the write (Q) by T1 is rejected and transaction T1 must be rolled back.

The rollback of T1 is required by the timestamp ordering protocol, it is unnecessary. Since T2 has already written Q, the value that T1 is attempting to write is one that will never need to be read. Any transaction Ti with TS (Ti) < TS (T2) that attempts a read (Q) will be rolled back, since TS (Ti) <W-timestamp (Q). Any transaction Tj with TS (Tj) > TS (T2) must read the value of Q written by T2, rather than the value written by T1.

This observation leads to a modified version of the timestamp-ordering protocol in which obsolete write operations can be ignored under certain circumstances. The protocol rules for write operations, however, are slightly different from the timestamp-ordering protocol. *Thomas* write rule as follows.

Suppose that transaction Ti issues write (Q):

1. If TS (Ti) < R-timestamp (Q), then the value of Q that Ti is producing was previously needed, and it had been assumed that the value would never be produced. Hence, the system rejects the write operation and rolls Ti back.
2. If TS (Ti) < W-timestamp (Q), then Ti is attempting to write an obsolete value of Q. Hence, this write operation can be ignored.
3. Otherwise, the system executes the write operation and sets W-timestamp (Q) to TS (Ti).

Thomas' write rule makes use of view serializability by, in effect, deleting obsolete write operations from the transactions that issue them. This modification of transactions makes it possible to generate serializable schedules that would not be possible under the other protocols. *For example*, schedule1 is not conflict serializable and, thus, is not possible under the two-phase locking protocol, the tree protocol, or the timestamp-ordering protocol. Under Thomas' write rule, the write (Q) operation of T1 would be ignored. The result is a schedule that is view equivalent to the serial schedule <T1, T2>.

4. Validation-Based Protocols

1

Oct. 2012 – *4M*
Explain Validation based Protocol.

1

Oct.2015 – *4M*
What is validation based protocol? Explain in detail the conditions for the validation test

In timestamp ordering the transaction timestamp is checked against the read and write timestamps of the item. Such checking represents overhead during transaction execution with the effect of slowing down the transactions.

In optimistic concurrency control techniques also known as validation or certification techniques no checking is done while the transaction is executing. Updates in the transaction are not applied directly to the database items until the transaction reaches its end. During transaction execution all updates are applied to local copies of the data items that are kept for the transaction.

At the end of transaction execution a validation phase checks whether any of the transaction updates violate serializablity. Certain information needed by the validation phase must be kept by the system. If serializablity is not violated the transaction is committed and the database is updated from the local copies otherwise the transaction is aborted and then restarted later.

There are three phases:

1. **Read phase**: During the phase, the system executes transaction Ti. It reads the values of various data items and stores them in variables local to Ti. It performs all write operations on temporary local variables, without updates of the actual database.

2. **Validation phase**: Transaction Ti performs a validation test to determine whether it can copy to the database the temporary local variables that hold the results of write operations without causing a violation of serializability.

3. **Write phase**: If transaction Ti succeeds in validation (step2), then the system applies the actual updates to the database. Otherwise, the system rolls back Ti.

To perform the validation test, we need to know when the various phases of transactions Ti took place. We shall, therefore, associate three different timestamps with transaction Ti:

1. **Start (Ti),** the time when Ti started its execution.

2. **Validation (Ti),** the time when Ti finished its read phase and started its validation phase.

3. **Finish (Ti),** the time when Ti finished its write phase.

We determine the serializability order by the timestamp-ordering technique, using the value of the timestamp Validation (Ti). Thus, the value TS (Ti) =Validation (Ti) and, if TS (Tj) < TS (Tk), then any produced schedule must be equivalent to a serial schedule in which transaction Tj appears before transaction Tk. The reason we have chosen Validation (Ti), rather than Start (Ti), as the timestamp of transaction Ti is that we can expect faster response time provided that conflict rates among transactions are indeed low.

The validation test for transaction Tj requires that, for all transactions Ti with TS (Ti) < TS (Tj), one of the following two conditions must hold:

1. Finish (Ti) < Start (Tj). Since Ti completes its execution before Tj started, the serializability order is indeed maintained.

2. The set of data items written by Ti does not intersect with the set of data items read by Tj, and Ti completes its write phase before Tj starts its validation phase (Start (Tj)<Finish (Ti)<Validation (Tj)). This condition ensures that the writes of Ti and Tj do not overlap. Since the writes of Ti do not affect the read of Tj, and since Tj cannot affect the read of Ti, the serializability order is indeed maintained.

T1	T2
Read (B)	
	Read (B) B = B - 50 Write (A) A= A + 50
Read (A) validate Display (A + B)	
	validate Write (B) Write (A)

Consider transactions T1 and T2. Suppose that TS (T1) < TS (T2), then the validation phase succeeds. Note that the writes to the actual variables are performed only after the validation phase of T2, thus, T1 reads the old values of B and A, and this schedule is serializable.

The validation scheme automatically guards against cascading rollbacks, since the actual writes take place only after the transaction issuing the write has committed.

5. Deadlock Handling

A deadlock can occur when two or more users are waiting for data locked by each other. Deadlocks prevent some transactions from continuing to work.

A system is in a 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 exists a set of waiting transactions {T0, T1, --, Tn} such that T0 is waiting for a data item that is held by T1, and T1 is waiting for a data item that is held by T2, and .... and Tn-1 is waiting for a data item that is held by Tn, and Tn is waiting for a data item that is held by T0. In such situation no transaction can proceed.

For example, there are two transactions T1 and T2 are in deadlock state. T1 is waiting for transaction T2 and T2 is waiting for transaction T1.

2

Oct.15,09 – 4M

Define Deadlock.

T1	T2
read lock (Y)	
read (Y)	
	read lock(X)
	read (X)
write (X)	
	write lock (Y)

1

Oct. 2009 – 2M

Explain Validation based Protocol.

To recover from this problem the system should take some action such as rolling back some transactions invoked in the deadlock. Rollback of a transaction can be partial, not the complete transaction. Transaction may be rolled back till the point where it obtained a lock whose release will resolve the deadlock.

There are two methods to deal with the deadlock problem. The first is the **deadlock-prevention protocol** to ensure that the system will never enter a deadlock state. **The second is deadlock-detection and deadlock-recovery scheme**. In this system will enter a deadlock state and then try to recover it using the above scheme. In both methods transaction rollback takes place.

Deadlock prevention is preferred where the probability of deadlock state is relatively high. Otherwise deadlock detection and recovery are more efficient. Deadlock detection and recovery require some overheads which is runtime lost of the protocol.

5.1 Deadlock Prevention

3

Apr.12,10 – *4M*
What is Deadlock? How to prevent Deadlock.

Oct.2009 – *4M*
Explain Deadlock Prevention Methods.

There are two approaches to deadlock prevention. One approach ensures that no cyclic waits can occur by ordering the requests for locks, or requiring all locks to be acquired together. The other approach is closer to deadlock recovery, and performs transaction rollback instead of waiting for a lock, whenever the wait could potentially result in a deadlock.

The simplest scheme under the first approach requires that each transaction locks all its data items before it begins execution. Either all are locked in one step or none are locked. There are two main disadvantages to this protocol.

1. It is not possible to predict, before the transaction begins, 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 for preventing deadlocks is to impose a partial ordering of all data items, and to require that a transaction lock a data item only in the order specified by the partial order.

A slight variation of this approach is to use a total order of data items. Once a transaction has locked a particular item, it cannot request locks on items that precede that item in the ordering.

The second approach for preventing deadlocks is to use preemption and transaction rollbacks. In preemption, when a transaction T2 requests a lock that is held by transaction T1, the lock granted toT1 may be preempted by rolling back of T1, and granting of the lock to T2. To control the preemption, we assign a unique timestamp to each transaction. The system uses these timestamps only to decide whether a transaction should wait or roll back. Locking is still used for concurrency

control. If a transaction is rolled back, it retains its old timestamp when restarted. Two different deadlock-prevention schemes using timestamp have been proposed:

1. **The wait-die scheme** is based on a non-preemptive technique. When transaction Ti requests a data item currently held by Tj. Ti is allowed to wait only if it has a timestamp smaller than that of Tj (that is, Ti is older than Tj). Otherwise, Ti is rolled back (dies).

   For example the transactions T0, T1, T2 have the timestamp 10, 15, 20 respectively. If T0 requests a data item held by T1 then T0 will wait. If T2 requests a data item held by T1 then T2 will rollback.

2. **The wound-wait scheme** is based on a preemptive technique and is a counterpart to the wait-die scheme. When transaction Ti request a data item currently held by Tj, Ti is allowed to wait only if it has timestamp larger than that of Tj (that is,Ti is younger than Tj). Otherwise, Tj is rolled back(Tj is wounded by Ti).

The same *example*, with transaction T0, T1, and T2 have the timestamp 10, 15, 20 respectively. If T0 requests a data item held by T1 then the data item will be preempted from T1 and T1 will be rolledback. If T2 requests a data item held by T1 then T2 will wait.

Whenever transactions are rolled back, it is important to ensure that there is no **starvation,** that is, no transaction gets rolled back repeatedly and is never allowed to make progress.

Both the wound-wait and the wait-die schemes avoid starvation: At any time, there is a transaction with the smallest timestamp. This transaction cannot be required to roll back in either scheme. Since timestamps always increase, and since transactions are not assigned new timestamps when they are rolled back, a transaction that is rolled back will eventually have the smallest timestamp. Thus, it will not be rolled back again.

There are differences in the way the two schemes operate:

1. In the wait-die scheme, an order transaction must wait for a younger one to release its data item. Thus, the order the transaction gets, the more it tends to wait. By contrast, in the would-wait scheme, an older transaction never waits for a younger transaction.

2. In the wait-die scheme, if the transaction Ti dies and is rolled back because it requested a data item held by transaction Tj, then Ti may reissue the same sequence of requests when it is restarted. If the data item is still held by Tj, then Tj will die again. Thus Ti may die several times before acquiring the needed data item. Contrast this series of events with what happens in the wound-wait scheme. Transaction Ti is wounded and rolled back because Tj requested a data item that it holds. When Ti is restarted and requests the data item now being held by Tj, Ti waits, thus, there may be fewer rollbacks in the wound-wait scheme.

The major problem with both of these schemes is that unnecessary rollbacks may occur.

Deadlock Detection and Recovery

If a system does not use some protocol that ensures deadlock freedom, then a detection and recovery scheme must be used. An algorithm that examines the state of system is invoked periodically to determine whether a deadlock has occurred. If one has, then the system must attempt to recover from the deadlock. To do so, the system must

1. Maintain 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 determine whether the system has entered a deadlock state.
3. Recover from the deadlock when the detection algorithm determines that a deadlock exists.

5.2 Deadlock Detection

2

Apr. 11, Oct.10 – *4M*
What is deadlock? Explain how deadlock is detected?

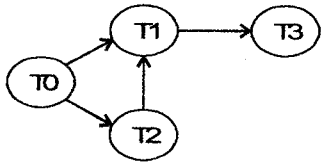
Deadlocks can be described specifically in terms of a directed graph called a **wait-for graph.** This graph consists of a pair G= (V, E) where V is set a vertices and E is a set of edges. The set of vertices consists of all the transactions in the system. Each element in the set E of edges is an ordered pair Ti→Tj. If Ti→Tj is in E, then there is a directed edge from transaction Ti to Tj, implying that transactions Ti is waiting for transactions Tj to release a data item that it needs.

When transaction Ti requests a data item currently being held by transaction Tj, then the edge Ti→Tj is inserted in the wait for graph. This edge is removed only when transaction Tj is no longer holding a data item needed by transaction Ti.

A deadlock exists in the system if and only if the wait for graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked. To detect deadlocks, the system needs to maintain the wait for graph and periodically to invoke an algorithm that searches for a cycle in the graph.

For example: Consider a wait for graph, which depicts the following situation

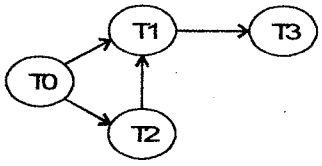
1. Transaction T0 is waiting for transaction T1 and T2.
2. Transaction T2 is waiting for transaction T1.
3. Transaction T1 is waiting for transaction T3.



Wait for graph with no cycle

Since the graph has no cycle the system is not in deadlock state.

Suppose now that transaction T3 is requesting an item held by T2. The edge T3→T2 is added to the wait for graph, resulting in the new system state in following diagram.



Wait for graph with a cycle

This time the graph contains the cycle.

T1 →T3→ T2 → T1.

Implying that transactions T1, T2 and T1 are all deadlocked.

If deadlock occurs frequently then the detection algorithm should be invoked more frequently than usual. Data items allocated to deadlocked transactions will be unavailable to other transactions until the deadlock can be broken. In addition the number of cycles in the graph may also grow. In the worst case we could invoke the detection algorithm every time a request for allocation could not be granted immediately.

1 **Oct.2015 – 2M** How is deadlock detected and how to recover deadlock?

3 **Apr.15,Oct. 10,12 –2/4 M** What is deadlock? Explain how deadlock is recovered.

5.3 Deadlock Recovery

When a detection algorithm determines that a deadlock exists, the system must recover from the deadlock. The most common solution is to roll back one or more transactions to break the deadlock. Three actions need to be taken:

1. **Selection of a victim:** Given a set of deadlocked transactions, we must determine which transaction (or transactions) to roll back to break the deadlock.

We should roll back those transactions that will incur the minimum cost. Unfortunately, the term minimum cost is not a precise one. Many factors may determine the cost of rollback, including:

a. How long the transaction has computed, and how much longer the transaction will compute before it completes its designated task?

b. How many data items the transaction has used?

c. How many more data items the transaction needs for it to complete?

d. How many transactions will be involved in the rollback?

2. **Rollback:** Once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be rolled back. The simplest solution is a **total rollback.** Abort the transaction and then restart it. However it is more effective to roll back the transaction only as far as necessary to break the deadlock. But this method requires the system to maintain additional information about the state of all the running transactions.

3. **Starvation:** In a system where the selection of victim is based primarily on cost factors, it may happen that the same transaction is always picked as a victim. As a result, this transaction never completes its designed task. This situation is called as **starvation.** We must ensure that a transaction can be picked as a victim only a small finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

3
Apr. 11, 10, Oct.10 – *4M*
Define following terms:
i. Upgrading
ii. Downgrading
iii. Lock point
iv. Starvation

Starvation refers to the use of all resources. As an example a poorly constructed database, poor concurrency controls will result in dbms starvation as usage increases. In effect the database subsystem is STARVING for resources because all available resources are being consumed. As a result you will see a severe degrade in performance.

Solved Examples

1. **The following is a list of events in an interleaved execution if set of transaction T0, T1, T2 with two phase locking protocol.**

Time	Transaction	Code
t1	T0	Lock (A,X)
t2	T1	Lock (B,S)
t3	T2	Lock (A,S)
t4	T0	Lock (C,X)
t5	T1	Lock (D,X)
t6	T0	Lock (D,S)
t7	T1	Lock (C,S)
t8	T2	Lock (B,S)

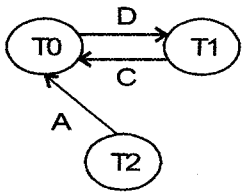
Construct a wait for graph according to above request. Is there deadlock at any instance? Justify.

Solution

First we convert set of instructions in the form of transactions.

T0	T1	T2
X(A)		
	S(B)	
		S(A)
X(C)		
	X(D)	
S(D)		
	S(C)	
		S(B)

To find deadlock we use wait for graph transaction are vertices and waiting for element is edges.



In the above wait for graph cycle occurs. Transaction T1 is waiting for T0 to unlock C and transaction T0 is waiting for T1 to unlock D. *Hence Deadlock occurs.*

---

2. Following is the list of events in an interleaved execution if set T1, T2, T3, and T4 has 2PL (two phase lock). Is there a deadlock? If yes which transactions are involved in deadlock.

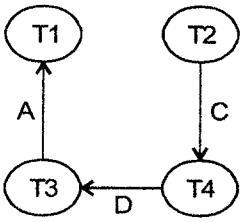
Time	Transaction	Code
t1	T1	Lock (A,X)
t2	T2	Lock (C,S)
t3	T3	Lock (A,S)
t4	T4	Lock (C,S)
t5	T1	Lock (B,X)
t6	T2	Lock (C,X)
t7	T3	Lock (D,X)
t8	T4	Lock (D,S)

Solution

First we convert set of instructions in the form of transactions.

T1	T2	T3	T4
X(A)			
	S(C)		
		S(A)	
			S(C)
X(B)			
	X(C)		
		X(D)	
			S(D)

To find deadlock situation we draw wait for graph. In vertices are transactions and edges are transactions waiting for another transaction to release lock.



The above graph does not contain cycle *therefore no deadlock occurs.*

3. Following is the list of events in an interleaved execution if set T1, T2, T3, and T4 have 2PL (two phase lock). Is there a deadlock? If yes which transaction are involved in deadlock.

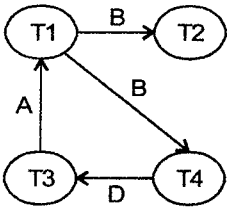
Time	Transaction	Code
t1	T1	Lock (A,X)
t2	T2	Lock (B,S)
t3	T3	Lock (A,S)
t4	T4	Lock (B,S)
t1	T1	Lock (B,X)
t2	T2	Lock (C,X)
t3	T3	Lock (D,S)
t4	T4	Lock (D,X)

Solution

First we convert set of instructions in the form of transactions.

T1	T2	T3	T4
X(A)			
	S(B)		
		S(A)	
			S(B)
X(B)			
	X(C)		
		S(D)	
			X(D)

To find deadlock situation we draw wait for graph. In vertices are transactions and edges are transactions waiting for another transaction to release lock.



A graph contains cycle like T1→T4 →T3→ T1. So *deadlock is present.*

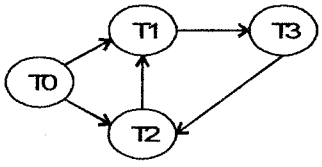
4. Following is the list of events in an interleaved execution if set T1, T2, T3, and T4 assuming 2PL (two phase lock). Is there a deadlock? If yes which transactions are involved in deadlock.

Time	Transaction	Code
t1	T1	Lock (A,X)
t2	T2	Lock (B,X)
t3	T3	Lock (C,S)
t4	T4	Lock (A,S)
t5	T1	Lock (C,X)
t6	T2	Lock (A,S)
t7	T3	Lock (D,X)
t8	T4	Lock (B,S)

Solution

First we convert set of instructions in the form of transactions.

T1	T2	T3	T4
X(A)			
	X(B)		
		S(C)	
			S(A)
X(C)			
	S(A)		
		X(D)	
			S(B)



Wait for graph with a cycle

There is no deadlock at any transaction as there is no dependency on two consecutive transactions on same variable. So there is no wait for variables

5. Following is the list of events in an interleaved execution if set T1, T2, T3, and T4 assuming 2PL. (two phase lock). Is there a deadlock? If yes which transactions are involved in deadlock.

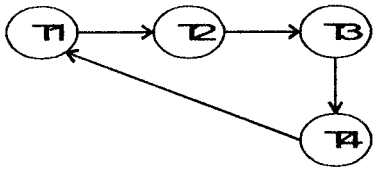
Time	Transaction	Code
t1	T1	Lock (A,X)
t2	T2	Lock (B,S)
t3	T3	Lock (A,S)
t4	T4	Lock (B,S)
t5	T1	Lock (B,X)
t6	T2	Lock (C,X)
t7	T3	Lock (D,S)
t8	T4	Lock (D,X)

Solution

First we convert set of instructions in the form of transactions.

T1	T2	T3	T4
X(A)			
	S(B)		
		S(A)	
			S(B)
X(B)			
	X(C)		
		S(D)	
			X(D)

To find deadlock situation we draw wait for graph in vertices are transactions and edges are transactions waiting for another transactions to release lock.



From this wait for graph a transactions t7 and t8 are involved in a deadlock as both require D in Shared (S) and Exclusive (X) mode.

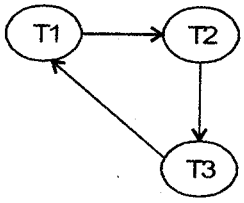
---

6. Following is the list of events in an interleaved execution if set T1, T2, and T3 with 2PL. (two phase lock). (Locks are released when transaction commits.) Is there a deadlock? If yes which transactions are involved in deadlock. Construct wait for graph.

Time	Transaction	Code
t1	T1	Lock (A,S)
t2	T2	Lock (B,X)
t3	T3	Lock (A,X)
t4	T1	Lock (C,S)
t5	T2	Lock (A,S)
t6	T3	Lock (D,X)
t7	T1	DISP(A-C)
t8	T2	Lock (D,S)
t9	T3	Lock(C,X)
t10	T1	COMMIT
t11	T2	Lock (C,S)

Solution

Wait for graph will be as follows.



After COMMIT instruction locks are released. There is no deadlock situation in this wait graph.

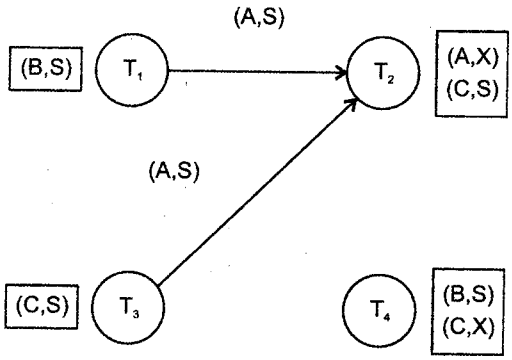
1 Oct.2012 – 4M

7. **Following is the list of events in an interleaved execution of setT_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 (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, S)
t_8	T_4	Lock (C, X)

Solution

T_1	T_2	T_3	T_4
S_LOCK(B)			
	X_LOCK(A)		
		S_LOCK(C)	
			S_LOCK(B)
S_LOCK(A)			
	X_LOCK(C)		
		S_LOCK(A)	
			X_LOCK(C)



8. Following is the list of events in an interleaved execution of sets T_1, T2, T3, T4 assuming 2PL. Is there a deadlock? If yes, which transactions are involved in deadlock?

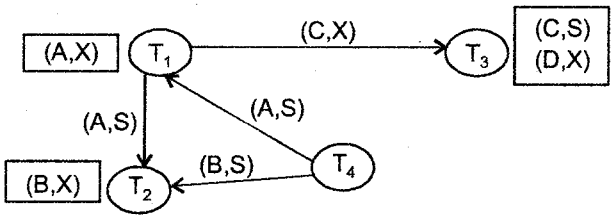
3
Apr.12,11, Oct.11 – *4M*

Time	Transaction	Code
t_1	T_1	LOCK(A,X)
t_2	T_2	LOCK(B,X)
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(A,S)
t_7	T_3	LOCK(D,X)
t_8	T_4	LOCK(B,S)

Solution

T_1	T_2	T_3	T_4
X_LOCK(A)			
	X_LOCK(B)		
		S_LOCK(C)	
			S_LOCK(A)
X_LOCK(C)			
	S_LOCK(A)		
		X_LOCK(D)	
			S_LOCK(B)

Wait for graph is drawn as follows for the above situation:



There is no cycle in wait for graph. So deadlock doesn't exist in the above situation.

9. 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?

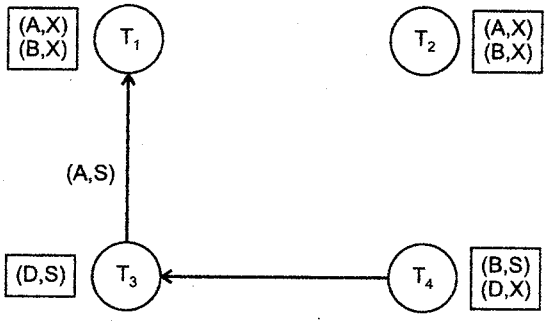
1
Apr.2012 – *4M*

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,X)
t_7	T_3	LOCK(D,X)
t_8	T_4	LOCK(D,X)

Solution

T_1	T_2	T_3	T_4
X_LOCK(A)			
	S_LOCK(B)		
		S_LOCK(A)	
			S_LOCK(B)
X_LOCK(B)			
	X_LOCK(C)		
		X_LOCK(D)	
			X_LOCK(D)

Above wait-for-graph does not show cycle. So deadlock does not exists in the system.



1 Oct.2010 – *4M*

10. Following is the list of events in an interleaved execution of sets T_1,T_2,T_3 and T_4 assuming 2PL(two phase lock).Is there a deadlock? If yes which transaction are involved in deadlock?

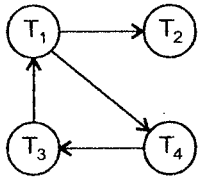
Time	Transaction	Code
t_1	T_1	Lock (A,X)
t_2	T_2	Lock (B,S)
t_3	T_3	Lock (A,S)
t_4	T_4	Lock (B,S)
t_5	T_1	Lock (B,X)
t_6	T_2	Lock (C,X)
t_7	T_3	Lock (D,S)
t_8	T_4	Lock (D,X)

Solution

First we will convert the given set of instructions in the form of transaction

T_1	T_2	T_3	T_4
X_Lock(A)			
	S_Lock(B)		
		S_Lock(A)	
			S_Lock(B)
X_Lock(B)			
	X_Lock(C)		
		S_Lock(D)	
			X_Lock(D)

To check whether deadlock is in the system or not we will draw wait for graph. Transactions are represented by vertices and waiting element by edges.



Cycle is present in wait for graph that means deadlock so deadlock is present. It is created by transaction T_1, T_3, T_4.

11. Following is the list of events in an interleaved execution of set T1 T2, T3 and T4, assuming 2PL (two phase lock). Is there a deadlock? If yes, which transactions are involved in deadlock?

1 Apr.2010 – *4M*

Time	Transaction	Code
t1	T1	Lock (A, X)
t2	T2	Lock (C, S)
t3	T3	Lock (A, S)
t4	T4	Lock (C, S)
t5	T1	Lock (B, X)
t6	T2	Lock (C, X)
t7	T3	Lock (D, X)
t8	T4	Lock (D, S)

Solution

From the given table we get,

Transaction	Code	Graph
T1	Lock(A, X)	
T2	Lock(C, S)	
T3	Lock(A, S)	T3 → T1
T4	Lock(C, S)	
T1	Lock(B, X)	
T2	Lock(C, X)	T2 → T4
T3	Lock(D, X)	
T4	Lock(D, S)	T4 → T3

Since there is no cycle, there is no deadlock.

12. The following is the list of events in an interleaved execution of set of transaction T0, T1, T2 with two phase locking protocol:

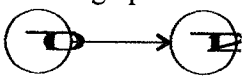
1 Apr.2010 – *4M*

Time	Transaction	Code
t1	T0	Lock (A, X)
t2	T1	Lock (B, S)
t3	T0	Lock (A, S)
t4	T1	Lock (C, X)
t5	T2	Lock (D, X)
t6	T0	Lock (D, S)
t7	T1	Lock (C, S)
t8	T2	Lock (B, S)

Construct a wait for graph according to above request. Is there deadlock at any instance? Justify.

Solution

From the given transactions, following wait-for graph results:



Since there is no cycle in the graph, no deadlock exists.

1 Oct.2014 – *4M*

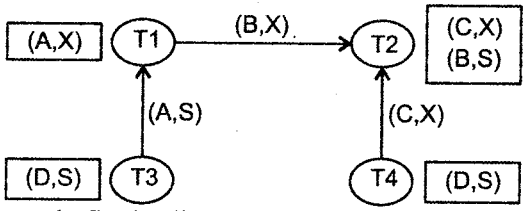
13. Following is the list of events in an interleaved execution of set of transaction T1, T2, T3 and T4 assuming 2PL. Is there a deadlock? If yes, which transactions are involved in Deadlock?

Time	Transactions	Code
t1	T1	Lock (A, X)
t2	T2	Lock (B, S)
t3	T3	Lock (A, S)
t4	T4	Lock (D, S)
t5	T1	Lock (B, X)
t6	T2	Lock (C, X)
t7	T3	Lock (D, S)
t8	T4	Lock (C, X)

Solution

T1	T2	T3	T4
X_LOCK(A)			
	S_LOCK(B)		
		S_LOCK(A)	
			S_LOCK(D)
X_LOCK(B)			
	X_LOCK(C)		
		S_LOCK(D)	
			X_LOCK(C)

Wait for graph is drawn as follows:



There is no cycle in wait for graph. So deadlock doesn't exist in the above situation.

1 Oct.2014 – *4M*

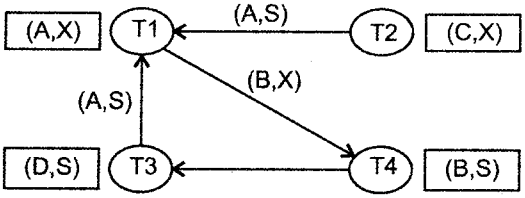
14. Following is the list of events in an interleaved execution of set of transaction T1,T2, T3 and T4 assuming 2PL. Is there a deadlock? If yes, which transactions are involved in deadlock?

Time	Transactions	Code
t1	T1	Lock (A,X)
t2	T2	Lock (A,S)
t3	T3	Lock (A,S)
t4	T1	Lock (B,S)
t5	T2	Lock (B,X)
t6	T1	Lock (C,X)
t7	T2	Lock (D,S)
t8	T3	Lock (D,X)

Solution

T1	T2	T3	T4
X_LOCK(A)			
	S_LOCK(A)		
		S_LOCK(A)	
			S_LOCK(B)
X_LOCK(B)			
	X_LOCK(C)		
		S_LOCK(D)	
			X_LOCK(D)

Wait for graph is drawn as follows:



Cycle is present in wait for graph that means deadlock is present.
It is created by transactions T1, T3, T4.

PU Questions

2 Marks

1. Define Deadlock. [Oct.15,09 – *2M*]
2. Define: i. upgrading ii. downgrading [Apr.2015 – *2M*]
3. Define: i. W-timestamp ii. R-timestamp [Oct.2014 – *2M*]
4. Define Growing Phase and Shrinking Phase. [Oct.2014 – *2M*]
5. Define Lock. List different types of Lock. [Oct.12, 11, Apr.12 – *2M*]
6. What is deadlock? Explain how deadlock is recovered. [Oct.12, 09, Apr.11 – *2M*]
7. Define Growing Phase. [Oct.2010 – *2M*]
[Apr.2010 – *2M*]

4 Marks

1. What is validation based protocol? Explain in detail the conditions for the validation test. [Oct.2015 – *4M*]

[Oct.2015 – 4M] 2. 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)

[Oct.15, Oct.14 –4M] 3. How is deadlock detected and how to recover deadlock?

[Oct.2015 – 4M] 4. Following is a 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 (C, X)
t_6	T_2	Lock (B, X)
t_7	T_3	Lock (D, X)
t_8	T_4	Lock (D, S)

[Apr.2015 – 4M] 5. Explain Timestamp ordering protocol.

[Apr.15,Oct.12 – 4M] 6. What is deadlock? Explain how deadlock is recovered.

[Apr.2015 – 4M] 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 request. Is there deadlock at any instance? Justify.

[Apr.2015 – 4M] 8. 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 phase locking 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.

9. Explain two phase locking protocol with example. [Oct.14,09,Apr.12– *4M*]

10. Following is the list of events in an interleaved execution of set of transaction T1, T2, T3 and T4 assuming 2PL. Is there a deadlock? If yes, which transactions are involved in Deadlock? [Oct.2014 – *4M*]

Time	Transactions	Code
t1	T1	Lock (A, X)
t2	T2	Lock (B, S)
t3	T3	Lock (A, S)
t4	T4	Lock (D, S)
t5	T1	Lock (B, X)
t6	T2	Lock (C, X)
t7	T3	Lock (D, S)
t8	T4	Lock (C, X)

11. Following is the list of events in an interleaved execution of set of transaction T1, T2, T3 and T4 assuming 2PL. Is there a deadlock? If yes, which transactions are involved in deadlock? [Oct.2014 – *4M*]

Time	Transactions	Code
t1	T1	Lock (A,X)
t2	T2	Lock (A,S)
t3	T3	Lock (A,S)
t4	T1	Lock (B,S)
t5	T2	Lock (B,X)
t6	T1	Lock (C,X)
t7	T2	Lock (D,S)
t8	T3	Lock (D,X)

12. Explain Validation based Protocol. [Oct.2012 – *4M*]

13. Following is the list of events in an interleaved execution of setT_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? [Oct.2012 – *4M*]

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, S)
t_8	T_4	Lock (C, X)

14. What is Deadlock? How to prevent Deadlock. [Apr.12,10 – *4M*]

15. Following is the list of events in an interleaved execution if setT_1,T_2,T_3 and T_4 assuming 2PL. Is there a Deadlock? If yes, which transactions are involved in deadlock? [Apr.2012 –*4M*]

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, S)
t_8	T_4	Lock (C, X)

[Apr.2012 – 4M] 16. Following is the list of events in an interleaved execution of sets T_1,T_2,T_3,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,S)
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)

[Oct.2011 – 4M] 17. Explain strict two phase locking protocol with example.

[Oct.2011 – 4M] 18. Following is the list of events in an interleaved execution if 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,X)
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 (A,S)
t_7	T_3	Lock (D,X)
t_8	T_4	Lock (B,S)

[Apr.11,Oct.10 – 4M] 19. What is deadlock? Explain how deadlock is detected?

[Apr.2011,Oct.10 – 4M] 20. Define terms: i. Upgrading ii. Downgrading iii. Lock point iv. Starvation

[Apr.2011 – 4M] 21. Following is the list of events in an interleaved execution if setT_1,T_2,T_3 and T_4 assuming 2PL. Is a 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(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(C,X)
t_7	T_3	LOCK(D,X)
t_8	T_4	LOCK(D,S)

[Apr.2011 – 4M] 22. Following is the list of events in an interleaved execution if set T_1,T_2, T_3 to as1suming 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,S)
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)

Chapter 5

RECOVERY SYSTEM

1. Introduction

A computer system like any other device (mechanical or electrical) is subject to failure from a variety of reasons: disk crash, power outage, software error, and fire in the machine room or even damage. In any failure information may be lost. So the database system must take some actions in advance to ensure two main properties of the transactions like atomicity and durability. A primary part of a database system is a recovery scheme that can restore the database to the consistent state that existed before the failure. The recovery scheme also provide high availability i.e. it must minimize the time for which the database is not usable after a crash.

2. Failure Classification

There are various types of failure that may occur in system each of which needs to be dealt with in a different manner. The simplest type of failure is one that does not result in the loss of information

6

Apr.15,Oct.12 – 4M
Explain various types of failure that may occur in a System.

Oct.14,11, 10, 09 – 4M
Explain different types of failures.

in the system. The failures that are more difficult to deal with are those resulting in a loss of information.

Following are the types of failure:

1. Transaction failure
2. System crash
3. Disk Failure

2.1 Transaction Failure

There are two types of errors that may cause a transaction to fail:

1. **Logical error:** The transaction can no longer continue with its normal execution because of some internal condition such as bad input, data not found, overflow or resource limit exceeded.
2. **System error:** The system has entered an undesirable state (deadlock) so the transaction cannot continue with its normal execution. The transaction can be reexecuted at a later time.

2.2 System Crash

There is a hardware malfunction or a bug in the database software or the operating system that causes the loss of the content of volatile storage and brings transaction processing to a halt. The content of nonvolatile storage remains intact and is not corrupted.

The assumption that hardware errors and bugs in the software bring the system to a halt but do not corrupt the nonvolatile storage contents is known as the *fail stop assumption.*

2.3 Disk Failure

A disk block loses its content as a result of either a head crash or failure during a data transfer operation. Copies of the data on other disks or archival backups on tertiary media such as tapes are used to recover from the failure.

To determine how the system should recover from failures we need to identify the failure modes of those devices used for storing data. Then we must consider how these failure modes affect the contents of the database. We can then propose algorithms to ensure database consistency and transactions atomicity despite failure. These algorithms are known as *recovery algorithm.*

1. Actions taken during normal transaction processing to ensure that enough information exists to allow recovery from failures.
2. Action taken after a failure to recover the database contents to a state that ensures database consistency, transaction atomicity and durability.

3. Storage Structure

The various data items in the database may be stored and accessed in a number of different storage media. To understand how to ensure the atomicity and durability properties of a transaction, we will study how the data is actually *stored* and what are their *access* methods.

3.1 Storage Types

6

Oct.14, 12 – *2M*
List different types of Storage.

Apr.12, 10 – *4M*
Explain different types of Storage Type.

Oct.11, Apr.11 – *4M*
Write a note on Storage type.

There are different types of storage media depending on their relative speed, capacity and resilience to failure they are classified as:

1. **Volatile storage:** Information stored in volatile storage does not usually survive system crashes. This memory access to volatile storage is extremely fast both because of the speed of the memory access itself and because it is possible to access any data item in volatile storage directly. *Examples* of volatile storage are main memory and cache memory.
2. **Nonvolatile storage:** Information stored in nonvolatile storage survives system crashes. Disks are used for on line storage whereas tapes are used for archival storage. Both these are subject to failure such as head crash. Nonvolatile storage is slower then volatile storage because disk and tape devices are electromechanical rather then based entirely on chips, as is volatile storage. Nonvolatile media are normally used only for backup data. *Examples of* nonvolatile storage are disk and magnetic tapes.
3. **Stable storage:** Information stored in stable storage is never lost. This kind of storage is practically impossible to obtain.

3.2 Data Access

The database system resides permanently on nonvolatile storage (Disks) and is partitioned into fixed length storage units called ***blocks***. Blocks are the units of data to and from disk and may contain several data items. We can assume that no data item spans two or more blocks.

Transactions input information from the disk to main memory and then output the information back onto the disk. The input and output operations are done in block units. The blocks stored on the disk are called as **physical blocks** and the blocks stored temporarily in main memory are called as **buffer blocks.** The areas of memory where blocks store temporarily are called the **disk buffer.**

Block movements between disk and main memory are done through the following two operations:

1. **Input (B):** Transfers the physical block B to main memory.
2. **Output (B):** Transfers the buffer block B to the disk and release the appropriate physical block there.

Following diagram illustrates this scheme.

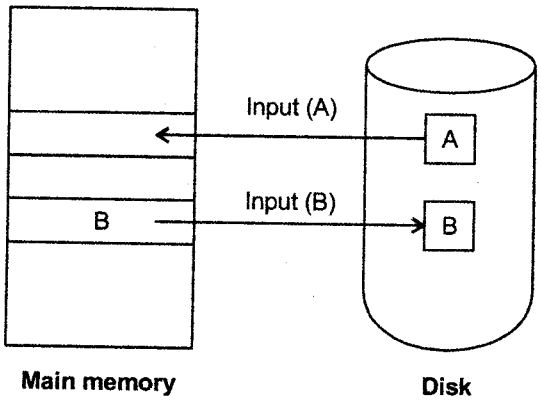


Figure 5.1

Each transaction Ti has a private work area in which copies of all the data items accessed and updated by Ti are kept. The system creates this work area when the transaction is initiated, the system removes it when the transaction either commits or aborts. Each data item X kept in the work area of transactions Ti is denoted by xi. Transactions Ti interacts with the database system by transferring data to and from its work to the system buffer.

The transfers of data takes place using following two operations:

1. **Read (X):** Assigns the value of data item X to the local variable xi. It executes this operation as follows:
   a. If block Bx on which X resides is not in main memory it issues input (Bx).
   b. It assigns to xi the value of X in buffer Block.
2. **Write (X):** Assigns the value of local variable xi to data item X in the buffer block. It executes this operation as follows:
   a. If block Bx on which X resides is not in main memory it issues input(Bx).
   b. It assigns the value of xi to X in buffer Bx.

The buffer block is written out to the disk by the buffer manager. It needs the memory space for other purposes. It is not written to the physical block immediately.

When a transaction needs to access a data item X for the first time it must execute read(X). The system then performs all updates to X on Xi. After the transaction access X for the final time, it must execute write (X) to reflect the change to X in the database itself.

The output(Bx) operation for the buffer block Bx on which resides does not need to take effect immediately after write(X) is executed.

If the system crashes after the write(X) operation was executed but before output (Bx) was executed the new value of X is never written to disk and thus is lost.

4. Recovery and Atomicity

Consider the transaction Ti that transfers `50 from account A to account B. The initial value of A and B are `1000 and `2000 respectively. Suppose that a system crash has occurred during the execution of Ti after output(BA) has taken place but before output(BB) was executed where BA and BB denote the buffer blocks on which A and B resides. Since the memory contents were lost. We could invoke two possible recovery procedures:

1. **Reexecute:** Account A will have value Rs.900 rather then the Rs.950. The system enters an inconsistent state.
2. **Do not reexecute:** Account A will be Rs.950 but account B will be Rs.2000. The system enters an inconsistent state.

This is because we are not preserving the atomicity property. So to achieve atomicity we must output the information describing the modification to the stable storage without modifying the database itself. Here we will assume that a transaction is active at a time.

4.1 Log-Based Recovery

3
Oct.14, 12, Apr.10 – *4M*
Explain log-based recovery.

1
Oct.2015 – *2M*
List the fields of update log record.

The most widely used structure for recording database modifications is the log. The log is a sequence of log records, which records all the update activities in the database. There are several types of log records, which are written in the system log. An update log record describes a single database write. It has following fields:

1. **Transaction identifier:** This is the unique identifier of the transaction that performs the *write* operation.
2. **Data item identifier:** Unit identifier of the data item written. Normally this is the location on disk of the data item.
3. **Old value:** This is the value of the data item prior to the write.
4. **New value:** This is the value of the data item after the write.

The other special log records also exist to record different events during transaction processing such as the start of a transaction and the commit or abort of a transaction.

The various types of log records are represented as:

1. <Ti, start>: It shows that transaction Ti has started.

   Example: **[start-transaction]:** It indicates that transaction T has started execution.

2. <Ti, Xj, V1, V2>**:** Transaction Ti has performed the write operation of data item Xi. Xj has V1 value before write and will have value V2 after write.

   Example **[write-item, T, X, old-value, new-value]**:- It indicates that transaction T has changed the value of database item X from old value to new value.

3. <Ti, commit>: Transaction Ti has committed.

   Example: **[commit, T]**: It indicates transaction T has completed successfully and the effect will be committed (stored permanently) to the database.

4. <Ti, abort>: Transaction Ti has aborted.

   Example: **[abort, T]**: It Indicates that transaction T has been aborted.

Whenever transaction performs write it is essential that the log record for that write be created before the database is modified. Once such a log record exists we can undo or redo the changes easily.

The log record is very useful for recovery from system and disk failures. This log must reside in stable storage. The log contains a complete record of all database activity, so the size of the log will become very long.

There are two techniques for using the log to ensure the transaction atomicity:

1. Deferred Database Modification
2. Immediate Database Modification

4.2 Deferred Database Modification

The deferred database techniques ensures transition atomicity by recording all database modifications in the log but deferring the execution of all write operations of a transaction until the transaction partially commits. A transaction is said to be partially committed when the final action of a transaction is executed. We assume that transactions are executed serially.

When the transaction partially commits, the information on the log associated with the transaction is used in executing the deferred writes. If the system crashes before the transaction completes its execution or if the transaction aborts then the information on the log is simply ignored. When transaction Ti is started then the record <Ti, start> is written in the log. All subsequent write operations are recorded in the log.

When transaction Ti is partially committed then the last record <Ti, commit> is written in the log. The system log is used to execute the different writes. If any system crash occurs during this operation then also there will be no problem as system log is written on the stable storage.

Let us consider an example to understand this concept. Let T0 transaction transfer Rs. 50 from account A to account B.

T0
Read (A)
A=A-50
Write (A)
Read (B)
B= B+50
Write (B)

3

Apr.15,Oct.11 – *4M*

Explain Deferred Database Modification with example.

Oct.2010 – *4M*

Explain recovery using deferred update method.

Let T1 be a transaction that withdraws Rs.100 from account C.

T1
Read (C)
C = C - 100
Write (C)

Suppose that these two transactions are executed serially in the order T0 followed by T1 and the value of accounts A, B and C before the executions are Rs.1000, Rs.2000 and Rs.700 respectively.

The portion of the system log for transactions T0 and T1 are.

<T0,start>
<T0,A,950>
<T0,B,2050>
<T0 commit>
<T1,start>
<T1,C,600>
<T1 commit>

Portion of the database log for transactions T0 and T1

There are various orders in which the actual outputs can take place to both the database system and the log as a result of the execution of T0 and T1. One such order is presented in following table. The value of A is changed in the database only after the record <T0, A, 950> has been placed in the log.

Log	Database
<T0,start>	
<T0,A,950>	
<T0,B,2050>	
<T0 commit>	
	A= 950
	B=2050
<T1,start>	
<T1,C,600>	
<T1 commit>	
	C=600

State of the log and database corresponding to T0 and T1

Using the log the system can handle any failure that results in the loss of information on volatile storage. The recovery procedure is.

Redo (Ti) Sets the value of all data items updated by transaction Ti to the new values.

The redo operation must be **idempotent** i.e. executing it several times must be equivalent to once.

After failure the recovery subsystems consults the log to determine which transactions need to be redone. Transaction Ti needs to be redone if and only if the log contains both the record <Ti, start> and the <Ti, commit>. If the system crashes after the transaction completes its execution the recovery scheme uses the information in the log to restore the system to a previous consistent state after the transaction had completed.

Let us suppose that the system crashes before the completion of the transaction. So that we can see how the recovery techniques restore the database to a consistent state.

<T0,start>	<T0,start>	<T0,start>
<T0,A,950>	<T0,A,950>	<T0,A,950>
<T0,B,2050>	<T0,B,2050>	<T0,B,2050>
	<T0 commit>	<T0 commit>
	<T1 start >	<T1 start>
	<T1,C,600>	<T1,C,600>
		<T1 commit>
(a)	(b)	(c)

Figure 5.2: The same log at three different times

Assume that the crash occurs after the log record **write (B)** operation of transaction T0 has been written in a stable storage. The log at the time of the crash appears in *fig. 5.2 (a)*. When the system comes back up no redo actions need to be taken since no commit record appears in the log. The values of accounts A and B remain Rs.1000 and Rs.2000 respectively. The log records of the incomplete transaction T0 can be deleted from the log.

Now let us assume the crash comes just after the log records **write (C)** of transaction T1 has been written to stable storage. In this case the log at the time of the crash is as in *fig. 5.2(b)*. When the system comes back up the operation redo (T0) is performed since the record <T0, commit> appears in the log on the disk. After this operation is executed the values of accounts A and B are Rs.950 and Rs.2050 respectively. The value of account C remains Rs.700. As before the log records of the incomplete transaction T1 can be deleted from the log.

Finally assume that a crash occurs just after the log record <T1, commit> is written to stable storage. The log at the time of this crash is as in *fig. 5.2 (c)*. When the system comes back up two commit records are in the log one for T0 and one for T1. The system must perform operations redo (T0) and redo (T1) in the order in which their commit records appear in the log. After the system executes these operations the values of accounts A, B and C are Rs.950, Rs.2050 and Rs.600 respectively.

4.3 Immediate Database Modification

The immediate modification technique allows database modifications to be output to the database while the transaction is still in the active state. These modifications written by active transactions are called as **uncommitted modifications.** In case of transaction failure the undo operation is performed on to the database.

Before transaction Ti starts its execution the system writes <Ti, start> to the log. During its execution any write (X) operation by Ti is preceded by the writing of the appropriate new update record to the log. When Ti partially commits the system writes the record <Ti, commit> to the log.

Let us consider the banking *example*. The transaction T0 and T1 executed one after the other in the order T0 followed by T1. The portion of the log containing the relevant information appears as following.

3
Oct.14 Apr.12, 10 – *4M*
Explain Immediate Database Modification with example.

<T0 start>
< T0 , A, 1000,950>
<T0, B, 2000,2050>
<T0 commit>
<T1 start>
<T1 start>
<T1 , C, 700, 600>
<T1 commit>

Portion of the system log corresponding to T0 and T1

1
Oct.2011 – *4M*
Define redo and undo operations.

Using the log the system can handle any failure that does not result in the loss of information in nonvolatile storage. The recovery procedure uses two steps.

1. **Undo (Ti):** restores the value of all data items updated by transaction Ti to the old values.
2. **Redo (Ti):** sets the value of all data items updated by transaction Ti to the new values.

The undo and redo operations must be idempotent to guarantee correct behavior even if a failure occurs during the recovery process.

After a failure has occurred the recovery scheme determine which transactions need to be redone and which need to be undone.

1. Transaction Ti needs to be undone if the log contains the record <Ti, start>, but does not contain the record <Ti, commit>.
2. Transactions Ti needs to be redone if the log contains both record <Ti, start> and the record <Ti, commit>.

Let us consider our banking *example*. The database system and log is as follows:

Log	Database
<T0,start>	
<T0,A,1000,950>	
<T0,B,2000,2050>	
<T0 commit>	
	A= 950
	B=2050
<T0 commit>	
<T1,start>	
<T1,C,700,600>	
	C=600
<T1 commit>	

State of system log and database corresponding to T0 and T1

Transaction T0 and T1 executed one after other in the order T0 followed by T1. Suppose that the system crashes before the completion of the transaction.

We consider three cases. The state of the logs for each of these cases appears in *figure* below:

(a)	(b)	(c)
<T0,start>	<T0,start>	<T0,start>
<T0,A,1000,950>	<T0,A,1000,950>	<T0,A,1000,950>
<T0,B,2000,2050>	<T0,B,2000,2050>	<T0,B,2000,2050>
	<T0, commit>	<T0,commit>
	<T1, start >	<T1 start>
	<T1,C,700,600>	<T1,C,700,600>
		<T1, commit>

Figure 5.3: The same log shown at three different times

Let us assume that the crash occurs just after the log record for the step **write (B)** of transaction T0 has been written to stable storage *(fig. 5.3(a))*. When the system comes back up it finds the record

<T0,start> in the log but no corresponding <T0,commit> record. So transaction T0 must be undone, hence so undo (T0) is performed. As a result the value of account A, B will be Rs.1000 and Rs. 2000 respectively.

Let us assume in second case where the crash has occurred just after the log record **write (C)** of transaction T1 has been written in stable storage *(fig 5.3(b))*. When the system comes back up, two recovery actions need to be taken. The operation undo (Ti) must be performed, since the record <T1 start> appears in the log, but there is no record <T1,commit>. The operation redo (T0) must be performed, since the log contains the record <T0 Start > and the record <T0, commit>. At the end of the entire recovery procedure, the values of accounts A, B and C are Rs.950, Rs.2050, and Rs.700, respectively. Note that the undo (T1) operation is performed before the redo (T0). This order is very important in the recovery procedure.

Finally, let us assume that the crash occurs just after the log record **<T1, commit>** has been written to stable storage *(Fig (c))*. When the system comes back up, both T0 and T1 need to be redone since the records <T0, start> and <T0, commit> appear in the log. After the system performs the recovery procedures redo (T0) and redo (T1) the values in account A, B and C are Rs.950, Rs.2050 and Rs.600 respectively.

4.4 Checkpoints

2
Apr.15,Oct.12 – 2M
What is checkpoint?

When a system failure occurs we must see the log to determine those transactions that need to be redone and those that need to be undone. We need to search the entire log to determine this information.

There are two problems with this:

1. The search process is time consuming.
2. Most of the transactions have already written their updates into the database. But still we redo them again and again. There will be no harm in doing this but the recovery procedure will take a longer time.

To reduce such kind of overhead we use the concept of checkpoints.

To use checkpoints in the log following three steps are used:

1. Output onto stable storage all log records currently residing 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 while a checkpoint is in progress. The recovery procedure can start from the checkpoint record in the system log and not from first record of the system log.

Consider the transaction Ti that is committed prior to the check point record. Any database modifications made by Ti must have been written to the database either prior to the checkpoint or as part of the checkpoint itself. At recovery time there is no need to perform a redo operation on Ti.

After a failure has occurred the recovery scheme examines the log to determine the most recent transaction Ti that started executing 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 and continues to search backward until it finds the next <Ti start> record. This record identifies a transaction Ti.

Once the system has identified transaction Ti the redo and undo operations need to be applied to only transaction Ti and all transaction Tj that started executing after transaction Ti. The remaining part of the log can be ignored and erased whenever desired.

For immediate modifications technique the recovery operations are:

1. For all transactions Ti in T that have no <Ti, commit> record in the log, execute undo (Ti).
2. For all transactions Ti in T such that the record <Ti commit> appears in the log, execute redo(Ti).

The undo operation does not need to be applied when the deferred modification technique is being employed.

Consider the set of transaction <T0, T1, ..., T100>. The most recent checkpoint took place during the execution of transaction T65. So only transactions T65,...T100 need to be considered during the recovery. Each of them needs to be redone if it has committed otherwise needs to be undone.

5. Recovery with Concurrent Transactions

Now we consider recovery if only a single transaction at a time is executing. Now we can modify and extend this with multiple transactions. The number of concurrent transactions the system has a single disk buffer and a single log. All transactions share the buffer blocks. We allow immediate modification and permit a buffer block to have data items updated by one or more transactions.

5.1 Interaction with Concurrency control

The recovery scheme depends on the concurrency control scheme that is used. To roll back a failed transaction we must undo the updates performed by the transaction. *For example:* Suppose that a transaction T0 has to be rolled and a data item X that was updated by T0 has to be restored to its old value. Using the log based schemes for recovery we restore the value by using the undo information in a log record. Suppose now that a second transaction T1 has performed yet another update on X before T1 is rolled back.

Therefore we require that if transaction T has updated a data item x no other transaction may update the same data item until T has committed or rolled back. This can be achieved by using strict two phase locking.

5.2 Transaction Rollback

Transactions can be aborted due to any failure. To restart, the aborted transaction is called as rollback. Rollback restores the state of the database to the last commit point. This command also releases the locks if any hold by the current transaction. The command used in SQL for this is simply: `ROLLBACK;`

1
Oct.2015 – 4M
Write a note on Transaction Rollback.

We roll back a failed transaction Ti by using the log. The system scans the log backward for every log record of the form <Ti, Xj, V1, V2> found in the log the system restores the data item Xj to its old value V1. Scanning of the log terminates when the log record <Ti start> is found.

If a 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. Once transaction T has updated a data item no other transaction could have updated the same data item.

5.3 Restart Recovery

When the system recovers from a crash it constructs two lists. The *undo-list* consists of transaction to be undone and the *redo-list* consists of transactions to be redone.

The system constructs the two lists as follows. Initially they are both empty. The system scans the log backward examining each record until if finds the first <checkpoint> record.

1. For each record found by the form <Ti commit> it adds Ti to redo list.
2. For each record found of the form <Ti start> if Ti is not in redo list then it adds Ti to undo list.

When the system has examined all the appropriate log records it checks the list L in the checkpoint record. For each transaction Ti in L if Ti in not in redo list then it adds Ti to the undo list.

After the redo-list and undo-list are constructed the recovery proceeds as follows:

1. The system rescans the log from the most recent record backward and performs an undo for each log record that belongs to transaction Ti on the undo list. Log records of transaction on the redo-list are ignored in this phase. The scan stops when the <Ti start> records have been found for every transaction Ti in the undo list.
2. The system locates the most recent <checkpoint L> record on the log. This step may involve scanning the log forward if the checkpoint record was passed in step 1.
3. The system scans the log forward from the most recent <checkpoint L> record and performs redo for each log record that belongs on a transaction Ti that is on the redo-list. It ignores log records of transaction on the undo list in this phase.

After the system has undone all transactions on the undo list it redoes those transactions on the redo-list. It is important in this case to process the log forward. When the recovery process has completed transaction-processing resumes.

6. Remote Backup Systems

Traditional transaction processing systems are centralized or client-server systems. Such systems are at risk to environmental disasters such as fire, flooding or earthquakes. There is need for transaction processing system that can function in spite of system failures or environmental disaste` Such systems mostly provide **high availability** i.e. the time for which the system is unusable must be extremely small.

We can achieve high availability by performing transaction processing at one site called the **primary site** and having a **remote backup** site where all the data from the primary site are replicated.

The remote backup site is sometimes also called the secondary site. The remote site must be kept synchronized with the primary site as updates are performed at the primary. We achieve synchronization by sending all log records from primary site to the remote backup site. The remote backup site must be physically separated from the primary.

When the primary site fails the remote backup site takes over processing. First it performs recovery using its copy of the data from the primary and the log records received from the primary. In effect the remote backup site performs recovery actions that would have been performed at the primary site when the latter recovered. Standard recovery algorithms with minor modifications can be used for recovery at the remote backup site. Once recovery has been performed the remote backup site starts processing transactions.

3

Oct.12, Apr.11 – *4M*
Explain Remote Backup System with proper diagram.

Oct.2009 – *4M*
Draw and explain Architecture of Remote Back-up System.

The performance of a remote backup system is better than the performance of a distributed system with two-phase commit.
Following *figure 5.4* shows the architecture of a remote backup system.

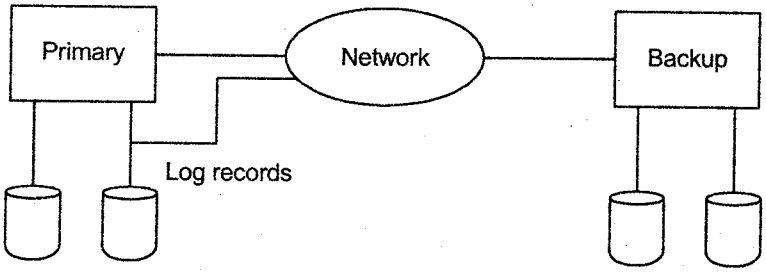


Figure 5.4:.Architecture of a remote backup system

1

Oct.2015 – *4M*
Explain advantages and disadvantages of the remote backup system.

Following are the several issues that must be addressed in designing a remote backup system:

1. **Detection of failure:** As in failure-handling protocols for distributed system, it is important for the remote backup system to detect when the primary has failed. Failure of communication lines can fool the remote backup into believing that the primary has failed. To avoid this problem, we maintain several communication links with independent modes of failure between the primary and the remote backup. *For example*, in addition to the network connection, there may be a separate modem connection over a telephone line, with services provided by different telecommunication companies. These connections may be backed up via manual intervention by operators, who can communicate over the telephone system.

2. **Transfer of control:** When the primary fails, the backup site takes over processing and becomes the new primary. When the original primary site recovers, it can either play the role of remote backup, or take over the role of primary site again. In either case, the old primary must receive a log of updates carried out by the backup site while the old primary was down.

   The simplest way of transferring control is for the old primary to receive redo logs from the old backup site, and to catch up with the updates by applying them locally. The old primary can then act as a remote backup site. If control must be transferred back, the old backup site can pretend to have failed, resulting in the old primary taking over.

3. **Time to recover:** If the log at the remote backup grows large, recovery will take a long time. The remote backup site can periodically process the redo log records that it has received, and can perform a checkpoint, so that earlier parts of the log can be deleted. The delay before the remote backup takes over can be significantly reduced as a result.

   A hot-spare configuration can make takeover by the backup site almost instantaneous. In this configuration, the remote backup site continuously processes redo log records as they arrive, applying the updates locally. As soon as the failure of the primary is detected, the backup site completes recovery by rolling back incomplete transactions; it is then ready to process new transactions.

4. **Time to commit:** To ensure that the updates of a committed transaction are durable, a transaction must not be declared committed until its log records have reached the backup site. This delay can result in a longer wait to commit a transaction, and some systems therefore permit lower degrees of durability.

   The degree of durability can be classified as follows:

   i. *One – safe:* A transaction commits as soon as its commit log record is written to stable storage at the primary site.

   The problem with this scheme is that updates of a committed transaction may not have made it to the backup site, when the backup site takes over processing. So the updates may appear to be lost. When the primary site recovers, the lost updates cannot be merged in directly since the updates may conflict with later updates performed at the backup site.

   ii. *Two- very safe:* A transaction commits as soon as its commit log record is written to stable storage at the primary and the backup site.

   The problem with this scheme is that transaction processing cannot proceed if either the primary or the backup site is down. So availability is actually less than in the single site case.

   iii. *Two-safe:* This scheme is the same as two-very-safe if both primary and backup sites are active. If only the primary is active the transaction is allowed to commit as soon as its commit log record is written to stable storage at the primary site.

   This scheme provides better availability than does two-very-safe while avoiding the problem of lost transactions faced by the one-safe scheme.

Solved Examples

1. Following are the log entries at the time of system crash?

[start-transaction, T1]
[write-item T1,D,20]
[commit,T1]
[checkpoint]
[start-transaction,T4]
[write-item,T4,B,15]
[commit,T4]
[start-transaction,T2]
[write-item,T2,B,25]
[start-transaction,T3]
[write-item,T3,A,30]
[write-item,T2,D,25]← system crash

If deferred update technique is used what will be the recovery procedure?

Solution

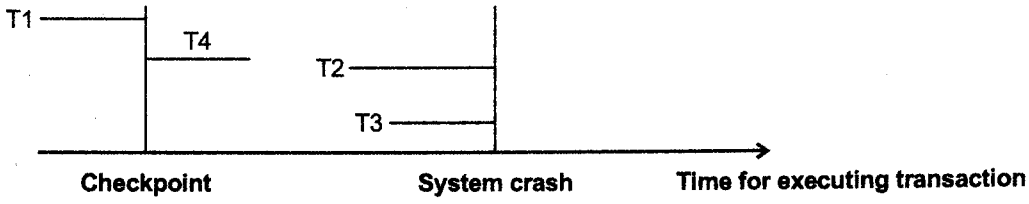
Using deferred update recovery technique two lists of transactions are maintained by the system.

Check the committed transactions since the last check point and list out active transactions while system crashes.

1. Apply Redo operations to all the write operations of the committed transactions from the log in order in which they were written in log.
2. Transaction that are active and did not commit are effectively cancelled and resubmitted

Step1: Transaction T1 committed before checkpoint so it is stored on *secondary storage*. The transaction T4 committed after checkpoint. So redo all operations of transactions T4.

Step 2: Transaction T2, T3 are active and they are not committed till system crash so cancel/ignore transaction T2 and T3.



2. Following are the log entries at the time of system crash?

[start-transaction, T1]

[read-item T1, D]

[write-item,T1,D,20]

[commit,T1]

[checkpoint]

[start-transaction T2]

[read-item ,T2,B]

[write-item,T2,B,12]

[start-transaction, T3]

[write-item,T3,A,20]

[read-item,T2,D]

[write-item, T3, D,25]← system crash

If deferred update technique is used what will be the recovery procedure?

Solution

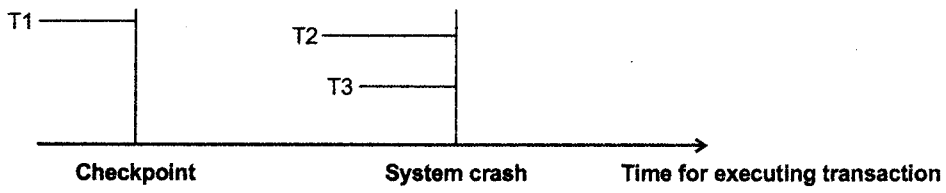
Using deferred update recovery technique two lists of transactions are maintained by the system.

Check the committed transactions since the last check point and list out active transactions while system crash.

1. Apply Redo operations to all the write operations of the committed transactions from the log in order in which they were written in log.
2. Transaction that are active and did not commit are effectively cancelled and resubmitted.

Step1: Transaction T1 committed before checkpoint so no need to redo transactions.

Step 2: Transaction T2, T3 are active and they are not committed till system crash so cancel/ignore transaction T2 and T3.



3. Following are the log entries at the time of system crash?

[start-transaction, T1]
[write-item T1,A,5]
[commit,T1]
[start-transaction,T2]
[write-item,T2,B,10]
[write-item,T2,D,15]
[commit,T2]
[checkpoint]
[start-transaction, T3]
[write-item,T3,B,20]
[Start-transaction, T4]
[write-item, T4, C, 10]← system crash

If deferred update technique is used what will be the recovery procedure?

Solution

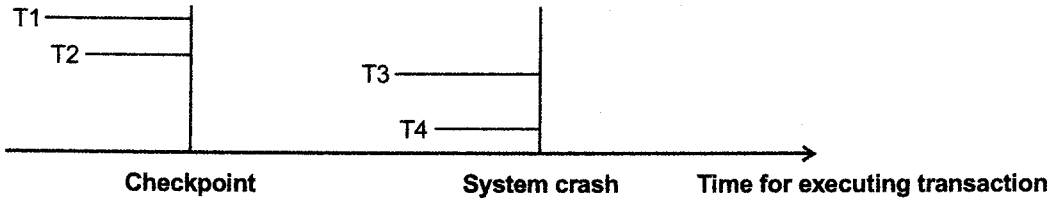
Using deferred update recovery technique two lists of transactions are maintained by the system.

Check the committed transactions since the last check point and list out active transactions while system crash.

1. Apply Redo operations to all the write operations of the committed transactions from the log in order in which they were written in log.
2. Transaction that are active and did not commit are effectively cancelled and resubmitted

Step1: Transaction T1, T2 committed before checkpoint so no need to consider transaction T1 and T2.

Step 2: Transactions T3, T4 are active and they are not committed till system crash so cancel/ignore transaction T3 and T4.



4. Following are the log entries at the time of system crash?

[start-transaction, T1]

[read-item T1,D]

[write-item,T1,D,B]

[commit,T1]

[checkpoint]

[start-transaction,T2]

[read-item,T2,B]

[write-item,T2,B,12]

[start-transaction,T3]

[write-item,T3,A,20]

[write-item,T2,D]

[write-item, T1, D, 20]← system crash

If immediate update with checkpoint is used what will be the recovery procedure

Solution

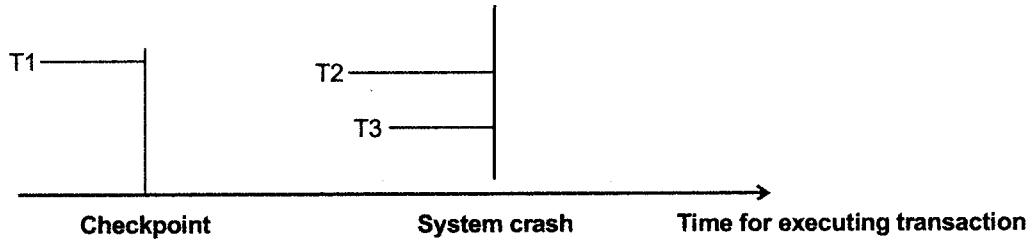
Using immediate update method two lists of transactions are maintained by the system.

Check the committed transactions since the last check point and list out active transactions while system crash.

1. Undo all the write-item operations of the active transactions from the log using the undo procedure.
2. Redo the write-item operations of the committed transactions from the log using redo procedure

Step1: Transaction T1 committed before checkpoint so no need to consider transaction T1.

Step 2: Transaction T2, T3 are active and they are not committed till system crash so undo all the operations of transaction T2 and T3.



5. Following are the log entries at the time of system crash?

[start-transaction, T1]

[read-item T1,A]

[read-item,T1,D]

[write-item,T1,D,20]

[commit,T1]

[checkpoint]

[start-transaction,T2]

[read-item,T2,B]

[write-item,T2,B,12]

[start-transaction,T3]

[write-item,T3,C,30]

[commit,T2]

[read-item,T3,D]

[write-item, T3, D, 25]← system crash

If immediate update with checkpoint is used what will be the recovery procedure?

Solution

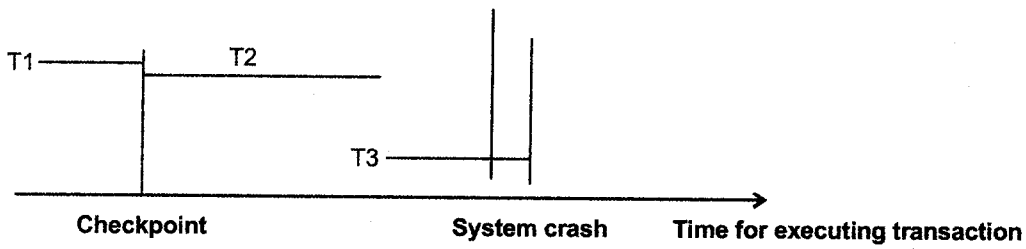
Using immediate update method two lists of transactions are maintained by the system.

Check the committed transactions since the last check point and list out active transactions while system crash.

1. Undo all the write-item operations of the active transactions from the log using the undo procedure.
2. Redo the write-item operations of the committed transactions from the log using redo procedure.

Step1: Transaction T1 committed before checkpoint so no need to consider transaction T1.

Step2: Transaction T2 committed after checkpoint so redo transaction T2. Transaction T3 is active so undo it.



6. Following are the log entries at the time of system crash?

[start-transaction, T1]

[write-item T1,A,5]

[commit,T1]

[start-transaction,T2]

[write-item,T2,B,10]

[write-item,T2,D,6]

[commit,T2]

[checkpoint]

[start-transaction,T3]

[write-item,T3,B,20]

[start-transaction,T4]

[write-item, T4, C, 10]← system crash

If immediate update with checkpoint is used what will be the recovery procedure?

Solution

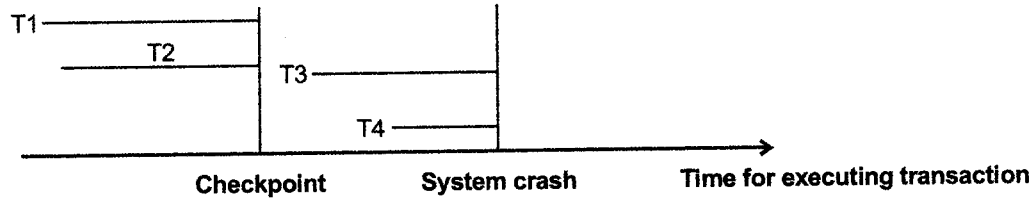
Using immediate update method two lists of transactions are maintained by the system.

Check the committed transactions since the last check point and list out active transactions while system crash.

1. Undo all the write-item operations of the active transactions from the log using the undo procedure.
2. Redo the write-item operations of the committed transactions from the log using redo procedure

Step1: Transaction T1and T2 are committed before checkpoint so no need to consider transaction T1and T2.

Step 2: Transaction T3 and T4 are active so undo all operations of transaction T3 and T4.



7. **Following are the log entries at the time of system crash?**

[start-transaction, T1]

[start-transaction, T2]

[read-item,t1,A]

[write-item,T2,B,25,50]

[start-transaction,T3]

[commit-transaction,T2]

[start-Transaction,T4]

[write-item,T1,C,100,115]

[commit-transaction,T1]

[write-item,T3,D,50,60]

[read-item,T3,E]

[write-item,T3,D,60,75]

[commit-transaction,T4]

[abort-transaction, T3]← system crash

If immediate update with checkpoint is used what will be the recovery procedure?

Solution

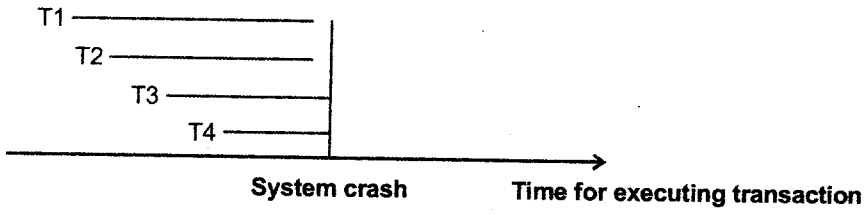
Using immediate update method two lists of transactions are maintained by the system.

Check the committed transactions since the last check point and list out active transactions while system crash.

1. Undo all the write-item operations of the active transactions from the log using the undo procedure.
2. Redo the write-item operations of the committed transactions from the log using redo procedure

Step1: Transaction T1, T2 and T4 are committed so redo all operations of T1, T2 and T4.

Step2: Transaction T3 is active so undo all write operations of transaction T3.



8. Following are the log entries at the time of system crash?

[start-transaction, T1]

[write-item T1,D,20]

[commit,T1]

[checkpoint]

[start-transaction,T4]

[write-item,T4,B,15]

[write-time, T4,A,20]

[commit,T4]

[start-transaction,T2]

[write-item,T2,B,25]

[start-transaction,T3]

[write-item,T3,A,30]

[write-item,T2,D,25]← system crash

If deferred update technique is used what will be the recovery procedure?

Solution

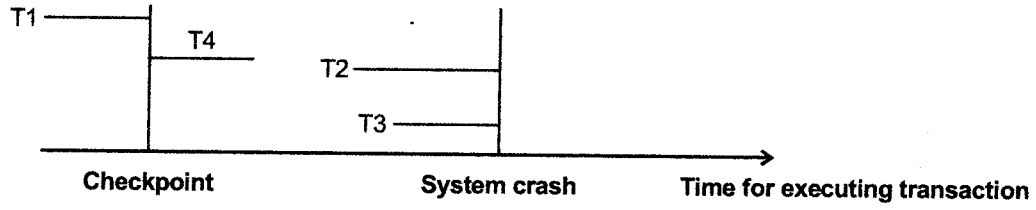
Using deferred update recovery technique two lists of transactions are maintained by the system.

Check the committed transactions since the last check point and list out active transactions while system crash.

1. Apply Redo operations to all the write operations of the committed transactions from the log in order in which they were written in log.
2. Transaction that are active and did not commit are effectively cancelled and resubmitted

Step1: Transaction T1 committed before checkpoint so it is stored on *secondary storage*. The transaction T4 committed after checkpoint. So redo all operations of transactions T4.

Step2: Transaction T2, T3 are active and they are not committed till system crash s so cancel/ignored transaction T2 and T3.



1 Oct.2012 – 4M

9. Following are the log entries at the time of system crash:

[start_transaction, T_1]

[write_item, T_1, A, 30]

[Commit, T_1]

[checkpoint]

[start_transaction, T_3]

[write_item, T_3, C, 50]

[commit , T_3]

[start_transaction, T_2]

[write_item, T_2, C, 40]

[start_transaction, T_4]

[write_item, T_4, B, 30]

[write_item, T_2, D, 60] ← System Crash

If differed update technique with checkpoint is used, what will be the recovery procedure?

Solution

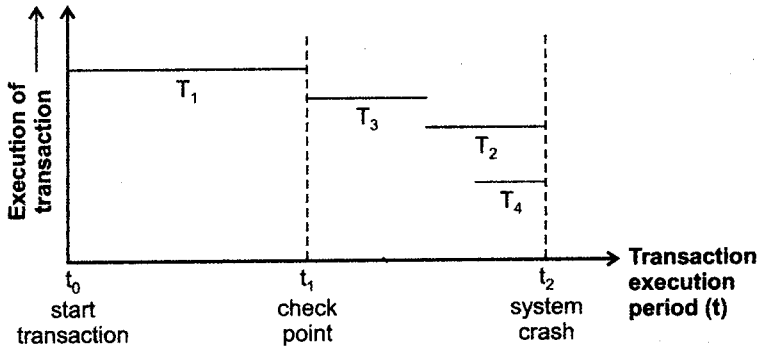
Deferred update recovery procedure maintains a list of committed transactions since the last check point and active transaction at time of system crash.

It will REDO all write item operations of committed transactions from the log in order in which they are written into the log. Active transactions which did not commit are cancelled and resubmitted.

In the above situation, transaction T_1 commits before check point so don't consider that transaction.

Transaction T_3 must REDO because it is committed after check point. REDO [write_item, T_3, C, 50] are done

Active transactions T_2 and T_4 has started their execution but they are not committed before system crashes so both of them are cancelled.



10. Following are the log entries at the time of system crash:

[start_transaction, T_1]
[write_item T_1, D, 20]
[commit T_1]
[check point]
[Start_transaction T_4]
[write_item T_4, B, 15]
[commit T_4]
[start_transaction T_2]
[write_item T_2, B, 25]
[Start_transaction T_3]
[write_item T_3, A, 30] ← System Crash

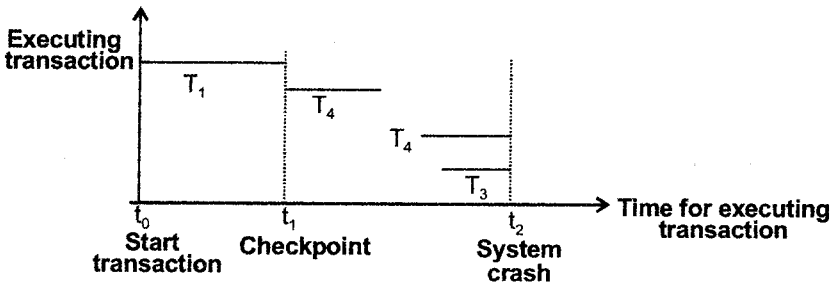
If deferred update technique is used, what will be the recovery procedure?

1 Apr.2011 – 4M

Solution

Deferred update recovery procedure maintains list of committed transactions since the last check point and active transaction at time of system crash.

It will REDO all write item operations of committed transactions from the log in order in which they are written into the log. Active transactions which did not commit are cancelled and resubmitted.



In the above situation, transaction T_1 commits before check point so don't consider that transaction. Transaction T_4 must REDO because it is committed after check point. REDO [write_item T_4, B, 15] are done.

Active transactions T_2 and T_3 has started their execution but they are not committed before system crashes so both of them are cancelled.

11. Following are log entries at the time system crash:
[start-transaction T_1]
[read-item T_2,A]

1 Oct.2010 – 4M

[read-item T_1,D]

[write-item,T_1,D,20]

[commit T_1]

[check point]

[start- transaction T_2]

[read- item T_2,B]

[write-itemT_2,B,12]

[start-transaction,T3]

[write-item,T_3,C,30]

[read-item, T_3,D]

[write-item, T_3,D,25] ← system crash

If deferred update with check point is used, what will be recovery procedure?

Solution

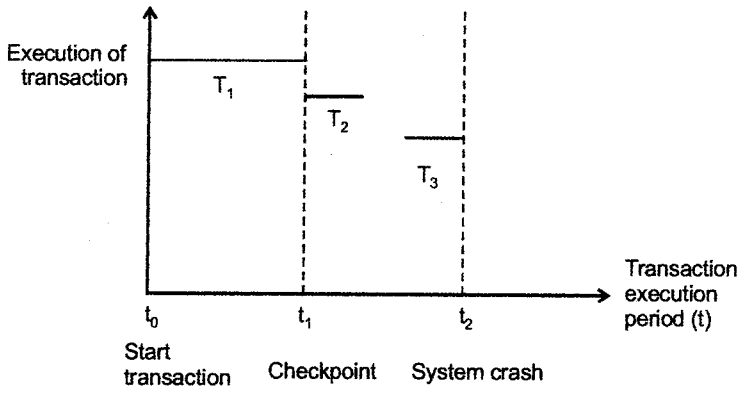
Deferred update recovery techniques maintain two list as follows:

i. The committed transaction T since the last checkpoint (commit list).

ii. Active transactions (active list).

Redo all the write_item operations of the committed transactions from the log in order in which they are written into the log.

The transactions that are active and did not commit are effectively cancelled and must be resubmitted.



In this *example*, T_1 is committed before checkpoint (before time t_1), so it is not necessary to consider this transaction.

Active transaction T_1 and T_3 has started their execution but they are not committed before systems crashes (time t_2) so both of them are cancelled.

12. Following are the log entries at the time of system crash.

[Start-transaction, T1]
[Write-item, T1, A, 10, 20]
[Commit, T1]
[Check point]
[Start-transaction, T2]
[Write-item, T2, B, 10, 15]
[Start-transaction,T3]
[Write-item, T_3, C, 10, 25]
[Commit T2]
[Write-item, T3, D, 10, 30] ← system crash

1 Oct.2014– *4M*

If immediate update with checkpoint is used, what will be the recovery procedure?

Solution

Using immediate update method, two lists of transactions are maintained by the system.

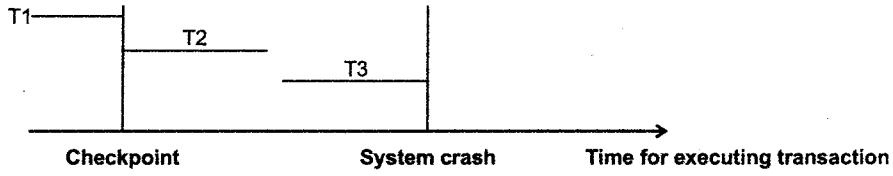
Check the committed transactions since the last check point and list out active transactions while system crash.

i. Undo all the write_item operations of the active transactions from the log using the undo procedure.

ii. Redo the write_item operations of the committed transactions from the log using redo procedure.

Step 1: Transaction T1 committed before checkpoint so no need to consider transaction T1.

Step2 : Transaction T2 committed after checkpoint so redo transactions T2. Transaction T3 is active so undo it.



PU Questions

2 Marks

1. List the fields of update log record. **[Oct.2015 – *2M*]**
2. What is checkpoint? **[Apr.15,Oct.12 – *2M*]**
3. List different types of Storage **[Oct.14,12 – *2M*]**

4 Marks

1. Explain advantages and disadvantages of the remote backup system. **[Oct.2015 – *4M*]**
2. Write a note on Transaction Rollback. **[Oct.2015 – *4M*]**
3. Explain immediate database modification with example **[Oct.15, Apr.12,10 – *4M*]**

[Oct.2015 – 4M]

4. The 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]
   [commit, 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 recovery procedure?

[Apr.15,Oct.12– 4M]

5. Explain various types of failures that may occur in system.

[Apr.15,Oct.11 – 4M]

6. Explain deferred database modification technique with example.

[Apr.2015 – 4M]

7. Following are the log entries at the time of system crash.
   [start – transaction, T_1]
   [write – item, T_1, A, 100]
   [commit, T_1]
   [start – transaction, T_3]
   [write – item, T_3, B, 200]
   [checkpoint]
   [commit, T_3]
   [start – transaction, T_2]
   [write – item, T_2, B, 300]
   [start – transaction, T_4]
   [write – item, T_4, D, 200]
   [write – item, T_2, C, 300] ← System crash
   If deferred update technique with checkpoint is used, what will be the recovery procedure?

[Oct.14, 11, 10, 09 – 4M]

8. Explain different types of failures.

[Oct.14, 12,Apr.10 – 4M]

9. Explain Log-based recovery.

[Oct.2014 – 4M]

10. Following are the log entries at the time of system crash.
    [Start-transaction, T1]
    [Write-item, T1, A, 10, 20]
    [Commit, T1]
    [Check point]
    [Start-transaction, T2]
    [Write-item, T2, B, 10, 15]
    [Start-transaction,T3]

[Write-item, T_3, C, 10, 25]
[Commit T2]
[Write-item, T3, D, 10, 30] ← system crash
If immediate update with checkpoint is used, what will be the recovery procedure? **[Oct.2012 – 4M]**

11. Explain Remote Backup System with proper diagram. **[Oct.2012 – 4M]**
12. Following are the log entries at the time of system crash:
[start_transaction, T_1]
[write_item, T_1, A, 30]
[Commit, T_1]
[checkpoint]
[start_transaction, T_3]
[write_item, T_3, C, 50]
[commit , T_3]
[start_transaction, T_2]
[write_item, T_2, C, 40]
[start_transaction, T_4]
[write_item, T_4, B, 30]
[write_item, T_2, D, 60] ← System Crash
If differed update technique with checkpoint is used, what will be the recovery procedure? **[Apr.12,10 – 4M]**
13. Explain different types of Storage Type. **[Apr.2012 – 4M]**
14. Following are the log entries at the time of system crash:
[start_transaction, T_1]
[read_item T_1, D]
[write_item T_1, D, B]
[commit, T_1]
[checkpoint]
[start_transaction, T_2]
[read_item T_2, B]
[write_item T_2, B, 10]
[start_transaction T_3]
[write_item T_2, B, 20] ←System crash
If immediate update with checkpoint technique is used what will be the recovery procedure? **[Oct.2011 – 4M]**
15. Define redo and undo operations.

[Oct.2011 – 4M] 16. Write a note on Storage type.

[Oct.11, Apr. 11 – 4M] 17. Following are the log entries at the time of system crash:

[start-transaction, T_1]
[Write-item T_1, D, 20]
[commit T_1]
[check point]
[Start-transaction T_4]
[write_item T_4, B, 15]
[commit T_4]
[start transaction T_2]
[write-item T_2, B, 25]
[Start-transaction T_3]
[write-item T_3, A, 30] ← System Crash

If deferred update technique is used, what will be the recovery procedure?

[Oct.2010 – 4M] 18. Explain recovery using deferred update method.

[Oct.2010 – 4M] 19. Following are log entries at the time system crash:

[start-transaction T_1]
[read-item T_2,A]
[read-item T_1,D]
[write-item,T_1,D,20]
[commit T_1]
[commit T_1]
[check point]
[start- transaction T_2]
[read- item T_2,B]
[write-itemT_2,B,12]
[start-transaction,T3]
[write-item,T_3,C,30]
[read-item, T_3,D]
[write-item, T_3,D,25] ← system crash

If deferred update with check point is used, what will be recovery procedure?

[Apr.2010 – 4M] 20. Explain log-based recovery.

[Apr.2010 – 4M] 21. Explain different types of storages.

Suggestive Readings:

1. Database Management Systems – Rajesh Narang – PHI Learning Pvt Ltd.
2. Database System Concepts by Silberschatz, Korth –Tata McGraw – Hill Publication.
3. An Introduction to Database Systems – Bipin Desai – Galgotia Publication.
4. Database Management System by Raghu Ramkrishnan – Tata McGraw – Hill Publication. 5. SQL, PL/SQL : The Programming Language Oracle – Ivan Bayross – BPB Publication.
5. Ramakrishnan, Raghu and Johannes Gehrke. 2003. Database Management Systems. New Delhi: McGraw-Hill Education.
6. Silberschatz, Abraham, Henry Korth and S. Sudarshan. 2010. Database System Concepts, 6th Edition. New York: McGraw-Hill.
7. Elmasri, Ramez and Shamkant B. Navathe. 2006. Fundamentals of Database Systems, 5th Edition. Boston: Addison-Wesley.
8. Ritchie, Colin. 2004.Relational Database Principles, 2nd Edition. New Delhi: Cengage Learning India Pvt. Ltd.
9. Maheshwari, Sharad and Ruchin Jain. 2006. Database Management Systems Complete Practical Approach. New Delhi: Firewall Media (Imprint of Laxmi Publications (P) Ltd.
10. Coronel, Carlos M and Peter Rob. 2006. Database Systems: Design, Implementation, and Management, 7th Edition. US: Cengage Learning.
11. Date, C. J. 2003. An Introduction to Database Systems, 8th Edition. Boston: Addison-Wesley.
12. Leon, Alexis and Mathews Leon. 2008. Database Management Systems, 1st Edition. New Delhi: Vikas Publishing House Pvt. Ltd..
13. Vaswani, Vikram. 2003. MySQL: The Complete Reference, 1st Edition. New York: McGraw Hill Professional