STUDENT ADVISORY

Dear Students,

Please be informed that the notes provided by the institute offer a concise presentation of the syllabus. While these notes are helpful for an overview and quick revision, We would strongly suggest that you refer to the prescribed textbooks / Reference book for a comprehensive understanding and thorough preparation of all exams and writing in the examination.

Best regards,

LJ Polytechnic.

પ્રિય વિદ્યાર્થીઓ,

તમને જાણ કરવામા આવે છે કે સંસ્થા દ્વારા પ્રદાન કરવામાં આવેલી નોંધો અભ્યાસક્રમની સંક્ષિપ્ત પ્રસ્તુતિ આપે છે. આ નોંધો વિહંગાવલોકન અને ઝડપી પુનરાવર્તન માટે મદદરૂપ હોઈ શકે છે તેમ છતા, અમે ભારપૂર્વક સૂચન કરીએ છીએ કે વિદ્યાર્થી તમામ પરીક્ષાઓ અને પરીક્ષામાં લેખનની વ્યાપક સમજણ અને સંપૂર્ણ તૈયારી માટે માત્ર સૂચવેલા પાઠ્યપુસ્તકો/સંદર્ભ પુસ્તકનો સંદર્ભ લો.

એલજે પોલિટેકનિક.

Unit-1

Function Dependency and Decomposition

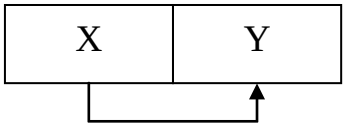
❖ Function Dependency

Let **R** be a relation schema having n attributes A1, A2, A3,…, An. Let attributes **X** and **Y** are two subsets of attributes of relation **R**. If the values of the **X** component of a **tuple** uniquely determine the values of the **Y** component, then there is a functional dependency **from X to Y**. This is denoted by **X → Y**.

It is referred as: **Y is functionally dependent on the X, or X functionally determines Y**.
The abbreviation for functional dependency is FD or f.d. The set of attributes **X** is called the left-hand side of the FD, and **Y** is called the right-hand side of the FD. The left-hand side of the FD is also referred as **determinant** whereas the right-hand side of the FD is referred as **dependent**.

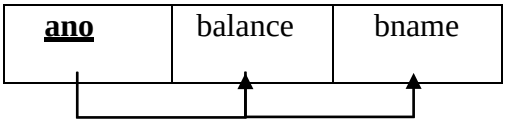
It is graphically represented as given below.

R:



- **Example:**

Consider the relation **Account(<u>ano</u>, balance, bname)**. In this relation **ano** can determines **balance** and **bname**. So, there is a functional dependency **from ano to balance and bname**. This can be denoted by **ano → {balance, bname}**.
Account:



❖ Full Function Dependency

An attribute is fully functional dependent on another attribute, if it is Functionally Dependent on that attribute and not on any of its proper subset.

For example, an attribute Q is fully functional dependent on another attribute P, if it is Functionally Dependent on P and not on any of the proper subset of P.

Let us see an example

Project_Cost

ProjectID	**ProjectCost**
001	1000
001	5000

Emp_Project

EmpID	ProjectID	Days
E099	001	320
E056	002	190

The above relations states that Days are the number of days spent on the project.

EmpID, ProjectID, ProjectCost -> Days

However, it is not fully functional dependent.

Whereas the subset **{EmpID, ProjectID}** can easily determine the {Days} spent on the project by the employee.

❖ Trivial Functional Dependency

In Trivial Functional Dependency, a dependent is always a subset of the determinant. i.e. If X → Y and Y is the subset of X, then it is called trivial functional dependency.
Here, {enrolment_no, name} → name is a trivial functional dependency, since the dependent name is a subset of determinant set { enrolment_no, name}.Similarly, enrolment_no → enrolment_no is also an example of trivial functional dependency.

❖ Non-Trivial Functional Dependency

In Non-Trivial Functional Dependency, a dependent is not a subset of the determinant. i.e. If X → Y and Y is the subset of X, then it is called Non-Trivial functional dependency.
Here, {enrolment_no, name} → result is a Non-Trivial Functional Dependency, since the result is not a subset of determinant set { enrolment_no, name}.

❖ Armstrong's Axioms for Functional Dependencies

Armstrong's Axioms is used to derive new FDs from other FDs.
Let assume that a relation **R** contains attribute-sets **A, B, C and D**.

Reflexivity

If B is a subset of A then A → B.

Augmentation

If A → B then AC → BC.

Transitivity

If A → B and B → C then A → C.

Pseudo-transitivity

If A → B and BD → C then AD → C.

Self-determination

A → A.

Decomposition

If A → BC then A → B and A → C.

Union

If A → B and A → C then A → BC.

Composition

If A → B and C → D then A, C → BD.

❖ Decomposition

Decomposition is the process of breaking down given relation into two or more relations. Here, a relation **R** is replaced by two or more relations in such a way that -Each new relation contains a subset of the attributes of R , and **Together, they all include all attributes of R.**

Relational database design process starts with a universal relation schema **R = {A1, A2, A3,..., An}**, which includes all the attributes of the database. The universal relation states that every attribute name is unique.

Using functional dependencies, this universal relation schema is decomposed into a set of relation schemas **D = {R1, R2, R3,…,Rm}**.Now, **D** becomes the relational database schema and **D** is referred as decomposition of **R**.

A decomposition of a relation can be either lossy decomposition or lossless join decomposition.
There are two types of decomposition

1. Lossy Decomposition

As the name suggests, when a relation is decomposed into two or more relational schemas, the loss of information is unavoidable when the original relation is retrieved.

Let us see an example −

Emp_Info

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables −

Emp_Details

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

DeptDetails

Dept_ID	Dept_Name
Dpt1	Operations
Dpt2	HR
Dpt3	Finance

Now, you won't be able to join the above tables, since **Emp_ID** isn't part of the **DeptDetails** relation.

Therefore, the above relation has lossy decomposition.

2. Lossless Join Decomposition

Decomposition is lossless if it is feasible to reconstruct relation R from decomposed tables using Joins. This is the preferred choice. The information will not lose from the relation when decomposed. The join would result in the same original relation.

Let us see an example

Emp_Info

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables:

Emp_Details

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

Dept_Details

Dept_ID	Emp_ID	Dept_Name
Dpt1	E001	Operations
Dpt2	E002	HR
Dpt3	E003	Finance

Now, Natural Join is applied on the above two tables −

The result will be −

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Therefore, the above relation had lossless decomposition i.e. no loss of information.

Unit-2

Normalization

❖ Normalization

In Relational Database design, the process of organizing data to minimize redundancy. Normalization usually involves dividing a database into two or more tables without losing information and defining relationships between the tables.

The goals of normalization process are:

1) To minimize data redundancy.
2) To minimize update, deletion and insertion anomalies.
3) Improve data integrity, scalability and data consistency.
4) Reduces disk space

The **Normal Form** is a state of a relation that results from applying some criteria on that relation.

Various normal forms are given below:

1) **First Normal Form (1NF)**
2) **Second Normal Form (2NF)**
3) **Third Normal Form (3NF)**
4) **Boyce-Codd Normal Form (BCNF)**
5) **Forth Normal Form (4NF)**
6) **Fifth Normal Form (5NF)**

❖ Types of Normal Forms

Normal forms are used for the process of normalization of data and therefore for the database design. Normal forms are used to eliminate or reduce redundancy in database tables.

- **First Normal Form (1NF)**

  A relation **R** is in **first normal form** (1NF) if and only if all **domains** contain **atomic** values only.

Example:

Cid	**Name**	**Address**		**Contact_no**
		Society	**City**	
C01	Emily	Nana Bazar, Anand		9879898798,7877855416
C02	Jeniffer	C.G.Road, Ahmedabad		9825098254
C03	Peter	M.G.Road, Rajkot		9898787898

Above relation has four attributes **Cid, Name, Address, Contact_no.** Here **Address** is **composite** attribute which is further divided in to sub attributes as **Society** and **City**. Another attribute **Contact_no** is **multi valued attribute** which can store more than one values. So above **relation is not in 1NF**.

Problem:

Suppose we want to find all customers for some particular city then it is difficult to retrieve. Reason is city name is combined with society name and stored whole as address.

Solution:

Insert separate attribute for each sub attribute of **composite** attribute.

First Approach:

Determine maximum allowable values for multi-valued attribute.

Insert separate attribute for multi valued attribute and insert only one value on one attribute and

other in another attribute.
So, above table can be created as follows:
Customer:

Cid	**Name**	**Society**	**City**	**Contact_no1**	**Contact_no2**
C01	Emily	Nana Bazar	Anand	9879898798	7877855416
C02	Jeniffer	C.G.Road	Ahmedabad	9825098254	
C03	Peter	M.G.Road	Rajkot	9898787898	

Second Approach:
Remove multi valued attribute and place it in a separate relation along with the primary key of a given original relation.
So, above table can be created as follows:
Customer:

Cid	**Name**	**Society**	**City**
C01	Emily	Nana Bazar	Anand
C02	Jeniffer	C.G.Road	Ahmedabad
C03	Peter	M.G.Road	Rajkot

Customer_Contact:

Cid	**Contact_No**
C01	9879898798
C01	7877855416
C02	9825098254
C03	9898787898

- **Second Normal Form (2NF)**

Prime attribute
An attribute, which is a part of the candidate-key, is known as a prime attribute.
Non-prime attribute
An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

A relation **R** is in second normal form (2NF) if and only if **it is in 1NF** and every **non-prime** attribute of relation is fully dependent on the **primary key**.

Example:

StudentID	**ProjectID**	**StudentName**	**ProjectName**
S89	S89	S89	S89
P09	P09	P09	P09
Olivia	Olivia	Olivia	Olivia
Geo Location	Geo Location	Geo Location	Geo Location

In the above table, we have partial dependency; let us see how,The prime key attributes are StudentID and ProjectID.

As stated, the non-prime attributes i.e. StudentName and ProjectName should be functionally

dependent on part of a candidate key, to be Partial Dependent.The StudentName can be determined by StudentID, which makes the relation Partial Dependent. The ProjectName can be determined by ProjectID, which makes the relation Partial Dependent. Therefore, the <StudentProject> relation violates the 2NF in Normalization and is considered a bad database design.

To remove Partial Dependency and violation on 2NF, decompose the above tables
StudentInfo

StudentID	ProjectID	StudentName
S89	S89	S89
P09	P09	P09
Olivia	Olivia	Olivia
S76	S76	S76

ProjectInfo

StudentID	ProjectName
P09	Geo Location
P07	Cluster Exploration
P03	IoT Devices
P05	Cloud Deployment

- **Third Normal Form (3NF)**

A relation that is in First and Second Normal Form and in which no non-primary-key attribute is transitively dependent on the primary key, then it is in Third Normal Form (3NF).

The normalization of 2NF relations to 3NF involves the removal of transitive dependencies. If a transitive dependency exists, we remove the transitively dependent attribute(s) from the relation by placing the attribute(s) in a new relation along with a copy of the determinant.

Example:

STU_ID	STU_NAME	STU_STATE	STU_COUNTRY	STU_AGE
1	**RAMESH**	**HARYANA**	**INDIA**	**20**
2	**RAM**	**PUNJAB**	**INDIA**	**23**
3	**SURESH**	**GUJARAT**	**INDIA**	**24**

FD set:
{STUD_NO->STUD_NAME,STUD_NO->STUD_STATE,STUD_STATE->
STUD_COUNTRY, STUD_NO -> STUD_AGE}
Candidate Key:
{STUD_NO}

For this relation in given table STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form.

To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:
STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
STATE_COUNTRY (STATE, COUNTRY)

Unit-3

Advanced SQL

❖ Transaction Control Language Commands

A Transaction is a set of database operations that performs a particular task.
A transaction must be completely successful or completely fail without doing anything to maintain database consistency.

Example: Successfully Complete Transaction: A task, say fund transfer from one account to another account.

To complete a transaction needs to update balance in two accounts – source and destination. It requires two database operations: one, to debit an amount in source account and to credit that amount in destination account.

Example: Fail Transaction: In a fund transfer, if a system failure occurs after debiting amount from source account, but before crediting it to destination account then database will be in inconsistent state.

So, balance should be updated in both the accounts in both accounts or not in anyone. We can say that a transaction is considered as a sequence of database operations. These operations involve various data manipulation operations such as insert, update and delete.

These operations are performed in two steps:

- Changes are made in memory only.
- Changes are permanently saved to hard disk.

A transaction begins with the execution of first SQL statement after a COMMIT and can be undone using ROLLBACK command.

A transaction can be closed by using COMMIT or ROLLBACK command. When a transaction is closed, all the locks acquired during that transaction are released.
TCL commands are used to manage transactions, that are given below:

1) **COMMIT**
2) **ROLLBACK**
3) **SAVEPOINT**

COMMIT: Committing a Transaction
There are two ways to COMMIT a transaction:
Explicitly
Implicitly

Explicit Commit:
To commit a transaction explicitly, user needs to request **COMMIT** command explicitly.
A **COMMIT** command terminates the current transaction and makes all the changes permanent.
Various data manipulation operations such **as insert, update and delete** are not affect permanently until they are committed.

There are some operations which forces a COMMIT to occur automatically, even user don't specify the COMMIT command.

Some of commands are given below:
Quit Command
Exit Command
Data Definition Language (DDL) commands

❖ Data Control Language Commands

Security of information stored in database is one of the prime concerns for any database management system.
An unauthorized access to a database must be prevented. The rights allow the user to use database contents are called privileges.

Oracle provides security to database contents in two phases:

- User requiring an access to database must have valid user id and password to get login in the system.
- User must have privileges to access contents of the database.

Example: In a banking system, customers need to access information about their own account only.
So, they should have access to view information about their own account. But they should not allow to modify any information.

Any customer must not be allowed to change balance of an account. Also, they should be prevented from accessing accounts of other customers.

A user owns the database objects, such as tables, created by his/her self. If any user needs to access objects belonging to other user, then the owner of the object requires to give permission for such access.

Oracle provides two commands- GRANT and REVOKE. To control the access of various database objects.

GRANT – Granting Privileges
GRANT command is used to granting privileges means to give permission to some user to access database object or a part of a database object.
This command provides various types of access to database object such as tables, views and sequences.
Syntax:

GRANT object privileges
ON object name
TO user name
[WITH GRANT OPTION];

The owner of a database object can grant all privileges or specific privileges to other users.
The WITH GRANT OPTION allows the grantee. User to which privilege is granted to in turn grant object privilege to other users.

User can grant all or specific privileges owned by him/her.
The table, given in below illustrates various object privileges.

Privilege	Description
ALL	To perform all the operation listed below.
ALTER	To change the table structure using ALTER command.
DELETE	To delete records from the table using DELETE command.
INDEX	To create an index on the table using CREATE INDEX command.
INSERT	To insert records into the table using INSERT INTO command.
REFERENCES	To reference table while creating foreign keys.
SELECT	To query the table using SELECT command.
UPDATE	To modify the records in the table using UPDATE command.

Example:
```
GRANT ALL ON Customer
TO user2
WITH GRANT OPTION;
```

Output:
Grant Succeeded.

REVOKE – Revoking Privileges
Revoking privileges means to deny (decline) permission to user given previously. The owner on an object can revoke privileges granted to another user. A user of the object, who is not an owner, but has been granted privileges using **WITH GRANT OPTION**, can revoke the privilege from the grantee.

Syntax:
```
REVOKE object privileges
ON object name
FROM user name;
```

Example:
As a user2, revoke the select and insert privileges from user3.
Input:
```
Revoke SELECT, INSERT
ON user1.Customer
FROM user3;
```

Output:
Revoke succeeded.

❖ Locks:

- When multiple users are accessing data concurrently, it is difficult to ensure data integrity. Such kind of access may result in concurrent access anomaly (irregularly).
- The technique used to protect data when multiple users are accessing it concurrently is called Concurrency Control.
- Oracle uses a method called 'locking' to implement Concurrency Control.
- Locking means to restrict other users from accessing data temporarily.
- There are two type of locking:

Implicit Locking
Explicit Locking

Implicit Locking

- Data in a table are locked automatically while executing SQL statements. This does not require any user interference. Such types of locks are called **Implicit Locks**.
- While applying lock on data, Oracle determines two issues:

1. **Type of lock**
2. **Level of lock**

1) Types of Locks:

- Oracle uses two different types of locks: **Shared Lock** and **Exclusive Lock**

❖ **Shared Locks:**

- **Shared locks** are applied while performing **read** operations.
- **Read** operation allow to view data, mostly using **SELECT** statement.
- **Multiple shared locks** can be placed simultaneously on a table or other object.
- As read operation does not modify table data. So multiple read operations can be performed simultaneously without causing any problem in data integrity.
- This means, multiple users can simultaneously read the same data.

❖ **Exclusive Locks:**

- **Exclusive locks** are applied while performing **write** operations.
- **Write** operation allow to modify data using **INSERT**, **UPDATE** or **DELETE** statement.
- **Only one exclusive lock** can be placed on a table or other object.
- As write operation modifies table data, multiple write operations can affect the data integrity and result in inconsistent database.
- **This means,** multiple users cannot modify the same data simultaneously**.**

2) Level of Locks:

- Multiple users may need to access different parts of the same table.
- **For example**, manager of 'xyz' branch may need to access only those accounts belonging to 'xyz' branch while other manager may have interest in other accounts**.**
- So, if entire table is locked by single user then others need to wait even though they have to access other part of the table.
- To solve this problem and to allow maximum possible concurrent access, locks should be placed on a part of the table or on entire table depending upon the requirement.
- So, if one customer is accessing its own account, other customers can access their own accounts.
- Oracle provides **three different levels** to place an implicit lock.
- These levels are determined depending upon the **WHERE** clause used in SQL statement.

❖ **Row Level Lock:**

- This lock is used when a condition given in WHERE clause evaluate a single row.

Example,

```
Select * from account
WHERE ano='a01';
```

- In this case, only single row (record) is locked. Other record of the table can be accessed by other users.

❖ **Page Level:**

- This lock is used when a condition given in WHERE clause evaluate a set of rows.

Example,

```
Select * from account
WHERE bname='vvn';
```

- In this case, only a particular set of rows are locked. Other records of the table can be accessed by other users.

❖ **Table Level:**

- This lock is used when a SQL statement does not contain WHERE clause.
- In this case, a query accesses entire table. So entire table is locked. Due to this reason, no any other user can access other part of the table.

Example,

```
Select * from account;
```

Explicit Locking

- User can lock data in a table on its own instead of automatic locking provided by Oracle. These types of locks are called **Explicit locks**.
- An owner of a table can place an explicit lock on the table. Some other users can also place an explicit lock if they have privilege.
- An explicit lock always overrides the implicit locks placed by oracle on its own.
- An entire table or records of the table can be explicitly locked by using one of these two commands:

1) The SELECT …… FOR UPDATE Statement

Syntax:

SELECT * FROM tableName **FOR UPDATE [NOWAIT];**

- This statement is used to acquire exclusive locks for performing updates on records.
- Based on **WHERE** clause used with **SELECT** statement level of lock will be applied.
- If table is already locked by other user, then this command simply **waits** until that lock is released.
- But, if **NOWAIT** is specified and table is not free, this command will return with an error message indicates "**Resource is Busy**".
- Lock will be released on executing **COMMIT** or **ROLLBACK**.
- Other clauses such as **DISTINCT, ORDER BY, GROUP BY and set operation** cannot be used here with SELECT statement.

Example:

As a user1, place an exclusive lock on accounts of 'XYZ' branch using SELECT…FOR UPDATE statement.

```
SELECT * FROM Account WHERE B_name = 'XYZ' FOR UPDATE;
```

Example:

As a user2, update balance to 1000 for all accounts of 'XYZ' branch.

UPDATE Account SET balance = 1000 WHERE B_name = 'XYZ';

- In this case, user2 has to wait until user1 releases lock by using **COMMIT** or **ROLLBACK**.

2) The LOCK TABLE Statement
Syntax:
LOCK TABLE tableName
IN lockMode **MODE [NOWAIT];**

- This statement is **used to acquire lock in one of the several specified modes of a given table**.
- If **NOWAIT** is specified and table is not free, this command will return with an error message indicates "**Resource is Busy**".
- Various modes of lock are described below:

MODE	Specifies…
EXCLUSIVE	Allows query on a table, but prohibits any other operation. Other users can only view data of a table.
SHARED	Allows concurrent queries, but no update operation is allowed.
ROW SHARED	Specifies row level lock. User cannot lock the whole table. Allowing concurrent access for all users of the table.
SHARE UPDATE	Same as above. Exists for compatibility with older versions.
ROW EXCLUSIVE	Similar to ROW SHARE, but prohibit shared locking. So, only one user can access the table at a time.

Example:
As a user1, place an exclusive lock on Account table using LOCK TABLE Statement.

LOCK TABLE Account
IN EXCLUSIVE **MODE [NOWAIT];**

Example:
Update the balance for account 'A01' to 1000 as a user1.

UPDATE Account **SET** balance = 1000 **WHERE** A_No = 'A01'**;**

- Now, as a user2 display the contents of the entire table and observe the balance for 'A01'. It will show the older balance because user1 does not committed their update operation.
- As a user1, **COMMIT** the update operation and again as user2, display Account table. Now, it will show the updated balance.
- Here, user2 can view data of Account table, because the lock is applied in **EXCLUSIVE** mode, which allows other user to view data. But no other operation is allowed.

So, user2 cannot modify the content of the table Account until the lock is released.

❖ Database Objects

• View

A view is the result set of a stored query on the data, which the database users can query just as they would in a persistent database collection object.

This pre-established query command is kept in the database dictionary. Unlike ordinary base tables in a relational database, a view does not form part of the physical schema: as a result set, it is a virtual table computed or collated dynamically from data in the database when access to that view is requested.

Changes applied to the data in a relevant underlying table are reflected in the data shown in subsequent invocations of the view. In some NoSQL databases, views are the only way to querydata.

Need of Views

- It simplifies the access of certain data.
- It allows hiding of certain pieces of data from unauthorized users. Thus it restricts access to the data such that a user can see and modify exactly what they need and no more.
- You can protect sensitive data within a table.
- It allows to summarize data from various tables which can be used to generate reports.
- It helps to structure the data in a way that users find it natural or sensitive.

Creating a View

Syntax:

```
CREATE or REPLACE view view_name
AS SELECT column_name(s)
FROM table_name
WHERE condition;
```

Using Views

Example:

```
SELECT * from sale_view;
```

Dropping Views

Syntax:

```
DROP VIEW viewname;
```

Example:

```
DROP VIEW sale_view;
```

- **Index**

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure.

Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.

Characteristics of Index

- Index is the way to improve overall performance by knowing exactly where the data is on disk and avoiding costly table scans, thus reducing I/O overhead.
- Indexes are special lookup tables that the database search engine can use to speed up data retrieval.
- Index is an ordered list of contents of a column or a group of columns or a table.
- It makes sorting and searching of records in the table fast.

- After an index has been created for a table, Oracle automatically maintains that index, insertion, updates and deletions of rows in the table automatically update the related indexes.
- A table can have any number of indexes, but the more indexes result in more overhead. This is because all associated indexes must be updated whenever table data is altered.
- Creating or dropping the indexes doesn't affect the table data.

Creating an Index

Syntax:

```
CREATE INDEX index_name
ON table_name (column_name);
```

Dropping an Index

Syntax:

```
DROP INDEX index_name;
```

• Sequences

The sequence is a feature by some database products which just creates unique values. It just increments a value and returns it.

The special thing about it is: there is no transaction isolation, so several transactions cannot get the same value, the incrementation is also not rolled back.

Without a database sequence it is very hard to generate unique incrementing numbers.Other database products support columns that are automatically initialized with a incrementing number.

Characteristics of Sequence

- Sequences are available to all users of the database.
- Sequences are created using SQL statement.
- Sequences have a minimum and maximum value which can be dropped. But not reset.
- Once a sequence returns a value, the sequence can never return that same value again.
- While sequence values are not tied to any particular table, it is always recommended to use sequence to generate values for only one table.
- The sequence information is stored in a data dictionary file in the same location as the rest of the data dictionary files for the database. If the data dictionary file does not exist, the SQL engine creates the file when it creates the first sequence.

Creating a Sequence

Syntax:

```
CREATE SEQUENCE sequence_name
        INCREMENT BY        N
        START WITH          N
        MAXVALUE            N  /NOMAXVALUE
        MINVALUE            N /NOMINVALUE
        CYCLE                 /NOCYCLE
        CACHE               N /NOCACHE
```

Altering a Sequence

Syntax:

```
ALTER SEQUENCE sequence_name
    INCREMENT BY        N
    MAXVALUE            N  /NOMAXVALUE
    MINVALUE            N /NOMINVALUE
    CYCLE                 /NOCYCLE
    CACHE               N  /NOCACHE;
```

Dropping a Sequence:

Use this function when a sequence is no longer useful, or to reset a sequence to an older number. To reset a sequence, first drop the sequence and then recreate it.

Syntax:

```
DROP SEQUENCE sequence_name;
```

Example:

```
Drop sequence serial;
```

- **Synonym**

A synonym is an alias or alternate name for a table, view, sequence, or other schema object. They are used mainly to make it easy for users to access database objects owned by other users.

They hide the underlying object's identity and make it harder for a malicious program or user to target the underlying object. Because a synonym is just an alternate name for an object, it requires no storage other than its definition.

When an application uses a synonym, the DBMS forwards the request to the synonym's underlying base object. By coding your programs to use synonyms instead of database object names, you insulate yourself from any changes in the name, ownership, or object locations.

There are two major uses of synonyms:
Object invisibility: Synonyms can be created to keep the original object hidden from the user.
Location invisibility: Synonyms can be created as aliases for tables and other objects that are not part of the local database.

Syntax:

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM [schema.] synonym_name
 FOR [schema.] object_name;
```

REPLACE allows modification in synonym definition without dropping it.

PUBLIC means that the synonym is a public synonym and is accessible to all users. User can use synonym only if he has appropriate privileges.

DROP Synonym Command

Synonyms, both private and public, are dropped in the same manner by using the DROP SYNONYM command, but there is one important difference. If you are dropping a public synonym; you need to add the keyword PUBLIC after the keyword DROP.

DROP SYNONYM addresses;

Unit-4
PL/SQL Concept

❖ PL/SQL Introduction

PL/SQL is a block structured language. The programs of PL/SQL are logical blocks that can contain any number of nested sub-blocks. Pl/SQL stands for "Procedural Language extension of SQL" that is used in Oracle.

PL/SQL includes procedural language elements like conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variable of those types and triggers. It can support Array and handle exceptions (runtime errors). After the implementation of version 8 of Oracle database have included features associated with object orientation.

You can create PL/SQL units like procedures, functions, packages, types and triggers, etc. which are stored in the database for reuse by applications.

❖ PL/SQL Datatypes

The PL/SQL variables, constants and parameters must have a valid data type, which specifies a storage format, constraints, and a valid range of values.

PL/SQL Scalar Data Types are listed as following:

Sr. No.	Datatype & Description
1.	**Numeric** Numeric values on which arithmetic operations are performed.
2.	**Character** Alphanumeric values that represent single characters or strings of characters.
3.	**Boolean** Logical values on which logical operations are performed.
4.	**Datetime** Dates and times.

- **PL/SQL Variables**

A variable is a meaningful name which facilitates a programmer to store data temporarily during the execution of code. It helps you to manipulate data in PL/SQL programs. It is nothing except a name given to a storage area. Each variable in the PL/SQL has a specific data type which defines the size and layout of the variable's memory.

A variable should not exceed 30 characters. Its letter optionally followed by more letters, dollar signs, numerals, underscore etc.

1. It needs to declare the variable first in the declaration section of a PL/SQL block before using it.
2. By default, variable names are not case sensitive. A reserved PL/SQL keyword cannot be used as a variable name.

Declare a Variable

You must declare the PL/SQL variable in the declaration section or in a package as a global variable. After the declaration, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

Syntax:
variable_name [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initial_value];

Example:
Radius Number:= 5;
Date_of_birth date;

- **PL/SQL Constants**

A constant is a value used in a PL/SQL block that remains unchanged throughout the program. It is a user-defined literal value. It can be declared and used instead of actual values.

Syntax:
constant_name CONSTANT datatype := VALUE;

Constant_name: it is the name of constant just like variable name. The constant word is a reserved word and its value does not change.
VALUE: it is a value which is assigned to a constant when it is declared. It can not be assigned later.

Example:
pi constant number:= 3.141592654;

- **PL/SQL Block**

In PL/SQL, the code is not executed in single line format, but it is always executed by grouping the code into a single element called Blocks. In this tutorial, you are going to learn about these blocks.

Blocks contain both PL/SQL as well as SQL instruction. All these instructions will be executed as a whole rather than executing a single instruction at a time.

PL/SQL blocks have a pre-defined structure in which the code is to be grouped.

A PL/SQL block has up to three different sections, only one of which is mandatory:

Declaration section

Identifies variables, cursors, and subblocks that are referenced in the execution and exception sections. Optional.

Execution section

Statements the PL/SQL runtime engine will execute at runtime. Mandatory.

Exception section

Handles exceptions to normal processing (warnings and error conditions). Optional.

```
DECLARE                                    -- Optional
<Declaration Section>
BEGIN                                      -- Mandatory
<Executable Commands>
EXCEPTION                                  -- Optional
<Exception Handling>
```

Example:

```
Input:
DECLARE
-- variable declaration
  message  varchar2(20):= 'Welcome to L.J.University';
BEGIN
  /*
   *  PL/SQL executable statement(s)
   */
  dbms_output.put_line(message);
END;
Output:
   Welcome to L.J.University
PL/SQL procedure successfully completed
```

- **Advantages of PL/SQL**

Block Structures: PL SQL consists of blocks of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL Blocks can be stored in the database and reused.

Procedural Language Capability: PL SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).

Better Performance: PL SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.

Error Handling: PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

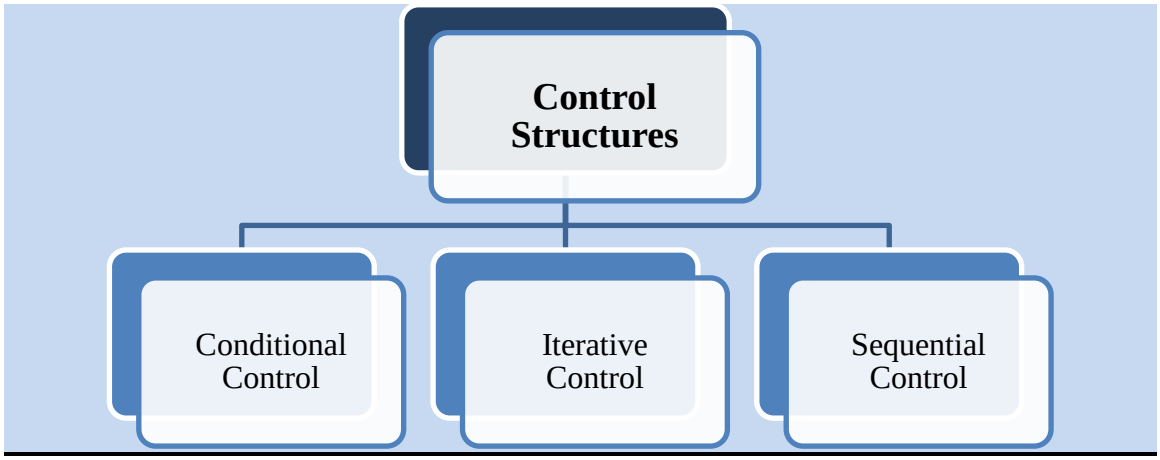
❖ PL/SQL Control Structures

According to the structure theorem, any computer program can be written using the basic control structures, which can be combined in any way necessary to deal with a given problem.

The **Conditional Structure** tests a condition, and then executes one sequence of statements instead of another, depending on whether the condition is true or false. A condition is any variable or expression that returns a Boolean value (TRUE, FALSE, or NULL).

The **Iteration Structure** executes a sequence of statements repeatedly as long as a condition holds true.

The **Sequence Structure** simply executes a sequence of statements in the order in which they occur.



- **Conditional Control**

Conditional control allows you to control the flow of the execution of the program based on a condition. In programming terms, it means that the statements in the program are not executed sequentially. Rather, one group of statements, or another will be executed, depending on how the condition is evaluated.

There are three forms of IF statements - **IF-THEN**, **IF-THEN-ELSE**, and **IF-THEN-ELSIF**.

IF-THEN

This construct tests a simple condition. If the condition evaluates to TRUE, one or more lines of code are executed. If the condition evaluates to FALSE, program control is passed to the next statement after the test. The following code illustrates implementing this logic in PL/SQL.

```
Input:
If var1>1 then
var2:=var1+2;
END IF;
```

IF-THEN-ELSE

This construct is similar to IF, except that when the condition evaluates to FALSE, one or more statements following the ELSE are executed. The following code illustrates implementing this logic in PL/SQL.

```
DECLARE
a number:=1;
b number:=2;
BEGIN
If a>b then
dbms_output.put_line("a is maximum.");
ELSE
dbms_output.put_line("b is maximum.");
END IF;
```

IF-THEN-ELSIF

This format is an alternative to using the nested IF-THEN-ELSE construct. The code in the previous listing could be reworded to read:

```
DECLARE
a number:=1;
b number:=2;
c number:=3;
BEGIN
If a>b AND a>c then
dbms_output.put_line("a is maximum.");
ELSEIF b>c then
dbms_output.put_line("b is maximum.");
ELSE
dbms_output.put_line("c is maximum.");
END IF;
```

- **Iterative Control**

LOOP statements let you execute a sequence of statements multiple times. There are three forms of LOOP statements: **LOOP**, **WHILE-LOOP**, and **FOR-LOOP**.

LOOP

The simplest form of LOOP statement is the basic (or infinite) loop, which encloses a sequence of statements between the keywords LOOP and END LOOP, as follows:

```
LOOP
    statement1;
    statement2;
    statement3;
    ...
END LOOP;
```

WHILE-LOOP

A WHILE loop has the following structure:

```
WHILE <condition>LOOP
    statement 1;
    statement 2;
    statement 3;
    ...
    statement N;
END LOOP;
```

Example:

```
DECLARE
v_counter        NUMBER := 1;
BEGIN
    WHILE v_counter < 5 LOOP
        dbms_output.put_line('v_counter = ' || v_counter);
        -- increment the value of v_counter by one
         v_counter := v_counter + 1;
    END LOOP;
```

FOR-LOOP

Whereas the number of iteration through a WHILE loop is unknown until the loop completes, the number of iterations through a FOR loop is known before the loop is entered. FOR loops iterate over a specified range of integers. The range is part of an iteration scheme, which is enclosed by the keywords FOR and LOOP.

```
FOR counter IN [REVERSE] lower_bound..upper_bound LOOP
    statement 1;
    statement 2;
    statement 3;
    ...
    statement N;
END LOOP;
```

Example:

```
SET SERVEROUTPUT ON
DECLARE
cnt_employee   NUMBER;
BEGIN
SELECT COUNT(*) INTO cnt_employee FROM employee;
    FOR v_counter IN 1..cnt_employee LOOP
        DBMS_OUTPUT.PUT_LINE('v_counter = ' || v_counter);
    END LOOP;
END;
/
```

- **Sequential Control**

Normally, execution proceeds sequentially within the block of the code. But, this sequence can be changed conditionally as well as unconditionally. To alter the sequence unconditionally, the GOTO statement can be used.

GOTO

PL/SQL also includes a **GOTO** statement to branch from one point to another.

Syntax:

GOTO <label>;

Where "<label>" is a label defined in the PL/SQL block.

Example:

```
SET SERVEROUTPUT ON
DECLARE
    v_counter NUMBER(2) := 1;
BEGIN
LOOP
        v_counter := v_counter+1;
        IF v_counter > 5 THEN
            GOTO PRINT1; -- print v_counter 5 times
        END IF;
        dbms_output.put_line('v_counter = ' || v_counter);
    END LOOP;
<< PRINT1>>
```

❖ PL/SQL Exceptions

Pre-defined Exceptions

PL/SQL provides many pre-defined exceptions, which are executed when any database rule is violated by a program. For example, the predefined exception NO_DATA_FOUND is raised when a SELECT INTO statement returns no rows.

User-defined Exceptions

PL/SQL allows you to define your own exceptions according to the need of your program. A user-defined exception must be declared and then raised explicitly, using either a RAISE statement or the procedure DBMS_STANDARD.RAISE_APPLICATION_ERROR.

The syntax for declaring an exception is:

DECLARE

my-exception **EXCEPTION**;

❖ Cursors

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has the attributes like %FOUND, %ISOPEN, %NOTFOUND, and %ROWCOUNT. The SQL cursor has additional attributes, %BULK_ROWCOUNT and %BULK_EXCEPTIONS, designed for use with the FORALL statement. The following table provides the description of the most used attributes:

Attribute	Description
%FOUND	Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
%NOTFOUND	The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
%ISOPEN	Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
%ROWCOUNT	Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Any SQL cursor attribute will be accessed as **sql%attribute_name** as shown below in the example.

Example:

Select * from customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

The following program would update the table and increase salary of each customer by 500 and use the SQL%ROWCOUNT attribute to determine the number of rows affected:

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE customers
  SET salary = salary + 500;
  IF sql%notfound THEN
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers selected ');
  END IF;
END;
```

When the above code is executed at SQL prompt, it produces the following result:
6 customers selected
PL/SQL procedure successfully completed.

If you check the records in customers table, you will find that the rows have been updated:
Select * from customers;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2500.00
2	Khilan	25	Delhi	2000.00
3	Kaushik	23	Kota	2500.00
4	Chaitali	25	Mumbai	7000.00
5	Hardik	27	Bhopal	9000.00
6	Komal	22	MP	5000.00

Explicit Cursors
Explicit cursors are programmer defined cursors for gaining more control over the context area. An explicit cursor should be defined in the declaration section of the PL/SQL Block.

The syntax for creating an explicit cursor is:
CURSOR cursor_name **IS** select_statement;

Working with an explicit cursor involves four steps:
- **Declaring the cursor for initializing in the memory**
- **Opening the cursor for allocating memory**
- **Fetching the cursor for retrieving data**
- **Closing the cursor to release allocated memory**

Declaring the Cursor
CURSOR c_customers
IS
SELECT id, name, address FROM customers;
Opening the Cursor
OPEN c_customers;
Fetching the Cursor
FETCH c_customers **INTO** c_id, c_name, c_addr;
Closing the Cursor
CLOSE c_customers;

❖ Trigger
Trigger is invoked by Oracle engine automatically whenever a specified event occurs. Trigger is stored into database and invoked repeatedly, when specific condition match. Triggers are written to be executed in response to any of the following events.

Syntax to Create a Trigger:

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

Example:

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

Output: Trigger created.

❖ Package

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

A package will have two mandatory parts −

- **Package Specification**

- **Package Body or Definition**

Package Specification

```
CREATE PACKAGE cust_sal AS
  PROCEDURE find_sal(c_id customers.id%type);
END cust_sal;
/
Output:
Package created.
```

Package Body

```
CREATE OR REPLACE PACKAGE BODY cust_sal AS

  PROCEDURE find_sal(c_id customers.id%TYPE) IS
  c_sal customers.salary%TYPE;
  BEGIN
    SELECT salary INTO c_sal
    FROM customers
    WHERE id = c_id;
    dbms_output.put_line('Salary: '|| c_sal);
  END find_sal;
END cust_sal;
/
Output:
Package body created.
PL/SQL procedure successfully completed.
```

❖ Procedure

The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.
Syntax for creating procedure:

```
CREATE [OR REPLACE] PROCEDURE procedure_name
   [ (parameter [,parameter]) ]
IS
   [declaration_section]
BEGIN
   executable_section
[EXCEPTION
   exception_section]
END [procedure_name];
```

Example:

```
CREATE OR REPLACE PROCEDURE "INSERTUSER"
(id IN NUMBER,
name IN VARCHAR2)
IS
BEGIN
insert into user values(id,name);
END;
/
Output:
```

Let's see the code to call above created procedure.

```
BEGIN
  insertuser(101,'Disha');
  dbms_output.put_line('record inserted successfully');
END;
/
```

Now, see the "USER" table, you will see one record is inserted.

ID	Name
101	Rahul

Destroy PL/SQL Procedure

Syntax:

DROP PROCEDURE procedure_name;

Example:

DROP PROCEDURE pro1;

❖ Function

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

Syntax:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

Example:

```
CREATE OR REPLACE FUNCTION add(n1 in number, n2 in number)
RETURN number
IS
n3 number(8);
BEGIN
n3 :=n1+n2;
RETURN n3:
```

Now write another program to call the function.

```
DECLARE
  n3 number(2);
BEGIN
  n3 := add(11,22);
  dbms_output.put_line('Addition is: ' || n3);
END;
/
Output:
Addition is: 33
Statement processed.
```

Destroy PL/SQL Function
Syntax:
DROP FUNCTION function_name**;**

Example:
DROP FUNCTION add;

Unit-5

Transaction Processing

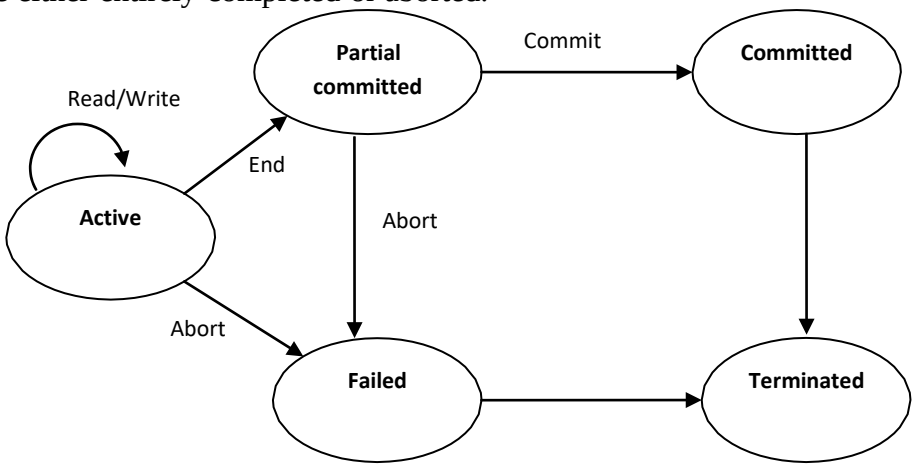
❖ Transaction

A Transaction can either be a single database operation or series of database operation.

It transforms a database from its one consistent state to another state. During the Transaction, intermediate state may be consistent or not but final state must be a consistent state.

- **Transition State Diagram**

A Transaction is a logical unit of work that contains one or more SQL statements.
A Transaction is an atomic unit. A database Transaction must be atomic, meaning that it must be either entirely completed or aborted.



A Transaction passes through various states during its execution.

Active

This is the initial state. The Transaction stay in this state while it is executing. Various read and write operations are performed on database.

Partially Committed

This is the state after the final statement of the Transaction is executed.

Failed

After the discovery that normal execution can no longer proceed.

Aborted (Failed)

The state after the Transaction has been rolled back and the database has been restored to its state prior to the start of the Transaction.

Aborted Transaction can be restarted later either automatically or manually.

Committed

The state after successful completion of the Transaction. It can be ensure that it will never be aborted.

Terminated

Transaction enters in this state either from committed or from aborted state. Information about the Transaction is stored in the system tables.

❖ Transaction Properties [ACID Properties]

Transaction remains in consistent state for that it must have following four properties:

Atomicity

Consistency
Isolation
Durability

❖ Concurrency control

When more than one user is accessing same data at the same time then it is known as concurrent access.

The technique used to protect data when multiple users are accessing it concurrently is called as concurrency control.

Need/problem of concurrency control

If Transactions are executed serially, i.e., sequentially with no overlap in time, no Transaction concurrency exists.

However, if concurrent Transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur.

The main three problems of concurrency control are as given below:
The Lost Update Problem
The Dirty Read Problem
The Inconsistent Retrieval Problem

• Degree of Consistency

There are **four level** of Transaction consistency as given below:
1) Level 0 Consistency
2) Level 1 Consistency
3) Level 2 Consistency
4) Level 3 Consistency

1) Level 0 Consistency

Level 0 Transactions are unrecoverable.
The Transaction T does not overwrite other Transaction's dirty (Uncommitted) Data.

2) Level 1 Consistency

Level 1 Transaction is recoverable from system failure. Transaction T must be in level 0 consistency and it does not make any of its update visible before it commits.

3) Level 2 Consistency

Level 2 Transactions consistency isolates from the updates of other Transactions. Transaction T must be in level 1 consistency and does not read dirty data of other Transactions.

4) Level 3 Consistency

Transaction T must be in level 2 consistency and other Transactions do not dirty any data read by Transaction T before T completes.

- **Schedule**

A chronological execution sequence of Transaction is called schedule.

Example:

Here, Transaction-A reads & writes X and Transaction-B reads & writes Y and then again Transaction-A reads & writes Z.

Transaction-A	Time	Transaction-B
Read X	t0	--
Write X	t1	--
--	t2	Read Y
--	t3	Write Y
Read Z	t4	--
Write Z	t5	--

- **Scheduler**

A scheduler is an in-built module of DBMS software which determines the **correct order** of execution of operations of multiple Transactions. It ensures that the CPU is utilized in efficient way.

Types of Schedules

Schedules can be classified as given below:

1) **Complete Schedule**
2) **Non-Complete Schedule**
3) **Serial Schedule**
4) **Non-Serial Schedule**

1) Complete Schedule

If a schedule contains either COMMIT or ROLLBACK actions for each Transaction. It is known as Complete Schedule.

2) Non-Complete Schedule

If a schedule does not contain either COMMIT or ROLLBACK actions for each Transaction. It is known as Non-Complete Schedule.

3) Serial Schedule

If actions of concurrent Transactions are not interleaved the schedule is called non-serial schedule.

4) Non-Serial Schedule

If actions of concurrent Transactions are interleaved the schedule is called non- serial schedule.

• Serializable Schedules

A schedule **S'** is serializable schedule if there exists some **serial** schedule **S** such that **S'** is equivalent to **S**.

The result produced by a serializable schedule is same as if the Transactions are executed serially.
The purpose of serializable schedule to improve the performance of the system.
It is important in multi-user where several Transactions are likely to be executed concurrently.

Rules of Serializability

- A Transaction mainly involves two operations: Read and Write.
- The rules are given below:
- If two Transaction T_1 and T_2 only read a data item, they do not conflict and the order is not important.
- If two Transaction T_1 and T_2 either read or write completely separate data items. They do not conflict and the order is not important.
- If one Transaction T_1 writes a data item and another Transaction T_2 either read or write the same data item then the order of execution is important.

❖ Deadlock

A set of Transaction is **deadlocked**, if each Transaction in the set is waiting for a lock held by some other Transaction in the set.

All the Transaction will continue wait forever.

Example:

Transaction-A has obtained lock on X and is waiting to obtain lock on Y. While, Transaction-B has obtained lock on Y and is waiting to obtain lock on X. But none of them can execute further.

Transaction-A	Time	Transaction-B
--	t0	--
Lock (X)	t1	--
--	t2	Lock (Y)
Lock (Y)	t3	--
Wait	t4	Lock (X)
Wait	t5	Wait
Wait	t6	Wait

Deadlock Detection and Deadlock Prevention:
Deadlock Detection
This technique allows deadlock to occur but then it detects it and to solve it.

If a deadlock is detected, one of the Transactions involved in deadlock cycle is aborted.

Other Transactions continue their execution. An aborted Transaction is rollback and restarted.

Deadlock Prevention

This technique prevents a deadlock to occur. It requires that all Transactions locks, all data items they need in advance. But if any of the locks cannot be obtained, a Transaction will be aborted. All the locks obtained are released and a Transaction will be rescheduled.

This technique ensures that a Transaction never needs to wait for any lock during its execution time period.

❖ Methods for Concurrency control

Locking Methods

A **lock** is a variable associated with the data item which controls the access of that data item.

Lock prevents access of the data item to second Transaction until first Transaction has completed the use of that data item.

Lock Granularity

The size of data item, chosen as the unit of protection by a concurrency control technique is called granularity.

A lock granularity indicates level of lock to use.

Different level of locks is given below:
1) Database Level
2) Table Level
3) Page Level
4) Row Level
5) Attribute Level

Lock Types

DBMS mainly uses following type of locking techniques:
1) Binary Locking
2) Shared/Exclusive (or Read/Write)Locking
3) Two-Phase Locking (2PL)

Time-stamp Method for Concurrency Control

A time-stamp is a unique identifier used to identify the relative starting time of a Transaction.

This method uses either system time or logical counter to be used as a time-stamp.
A time-stamp for Transaction T is denoted by TS(T).

To implement this time stamping, following two time-stamp values are associated with each data item.

W-Timestamp (X)

Write time-stamp of data-item X is denoted by W-timestamp(X).It specifies the largest timestamp of any Transaction that execute write (X) successfully.

R-Timestamp (X)

Read time-stamp of data-item X is denoted by R-timestamp(X). It specifies the largest timestamp of any Transaction that execute Read (X) successfully.

The timestamp ordering protocol ensures that any conflicting Read and Write operations are executed in timestamp order. This method operates as follow:

1) **If Transaction Ti issues Read (X) operation:**

a. **If TS(Ti) < W-timestamp (X)**

Then, Ti needs to read a value of X that was already overwritten.
Hence, the read operation is rejected, and Ti is rolled back.

b. **If TS(Ti) ≥ W-timestamp (X)**

Then, the **read** operation is executed, and **R-timestamp(X)** is set to the **maximum** of R-timestamp(X) and TS(Ti).

2) **If Transaction Ti issues Write (X) operation:**

a. **If TS(Ti) < R-timestamp(X)**

Then, the value of **X** that **Ti** is producing was needed previously, and the system assumed that that value would never be produced.

Hence, the **write** operation is **rejected**, and **Ti** is rolled back.

b. **If TS(Ti) < W-timestamp(X)**

Then, **Ti** is attempting to **write** an obsolete (out-dated) value of X.Hence, this **write** operation is **rejected**, and **Ti** is rolled back.

c. **Otherwise, the write operation is executed, and W-timestamp(X) is set to TS(Ti).**

Optimistic Method for Concurrency Control

In this method assume that conflicts are very rare. It allows Transaction to run completely. And it checks for conflicts before they commit. It is also known as validation or certification method.

Execution of Transaction Ti is done in three phases.

1) **Read Phase**
2) **Validation Phase**
3) **Write Phase**

1) **Read Phase**

The Transaction reads input values from the database, performs computation and records the updates in a temporary local variable that is not accessed by other Transactions.

2) Validation Phase

Transaction Ti performs a ``validation test'' to determine if local variables can be written without violating serializability. If test is positive, then Transaction goes to the write phase otherwise changes are discarded and Transaction Ti is restarted.

3) Write Phase

In this phase changes, recorded in local variables are permanently applied to the database.