Infographics

Visual Learning
Better understanding through visual content.

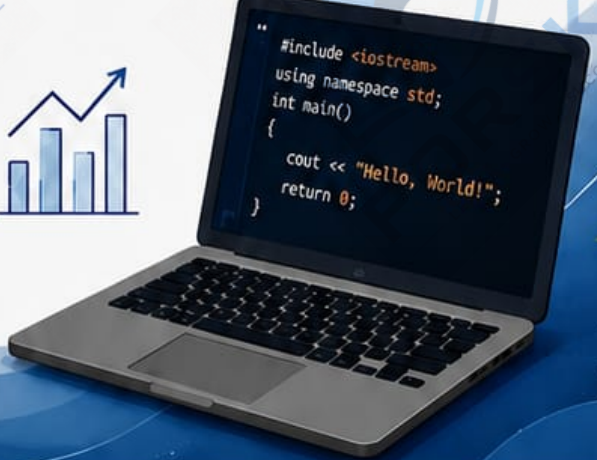
Simplified Concepts
Complex topics explained in simple and visual form.

Quick Revision
Easy to revise important points and diagrams.

Exam Ready
Helps in faster revision and better exam preparation.

Unit - IV


DATA STRUCTURES
ALGORITHMS
COMPUTER NETWORKS
DATABASE SYSTEMS


#include <iostream>
using namespace std;
int main()
{
cout << "Hello, World!";
return 0;
}

PROCEDURE

Introduction, Importance, Types, Advantages & Disadvantages

```
CREATE PROCEDURE ...
BEGIN
   -- SQL Statements
END;
```

1 INTRODUCTION

A Procedure is a named PL/SQL block that performs a specific task.

It can accept input parameters, perform operations and return results (optional).

It is stored in the database and can be reused whenever needed.

2 IMPORTANCE

- Improves modularity: Breaks large programs into smaller, manageable units.
- Reusability: Write once and use many times.
- Reduces development time and effort.
- Improves performance: Stored and compiled in the database.
- Enhances security: Access to data can be controlled.
- Easier maintenance: Changes in logic are made in one place.
- Promotes data integrity and consistency.

3 TYPES OF PROCEDURES

A) STORED PROCEDURE

- Stored in the database.
- Compiled and stored.
- Can be called many times.
- Can accept IN, OUT and IN OUT parameters.

SYNTAX:

```
CREATE OR REPLACE PROCEDURE  proc_name
(param1 IN datatype, param2 OUT datatype, ...)
IS
BEGIN
   -- SQL statements
END;
/
```

B) USER DEFINED PROCEDURE

- Created by users as per their requirement.
- Stored in the user schema.
- Invoked using CALL statement.
- Helps to organize business logic.

SYNTAX:

```
CREATE OR REPLACE PROCEDURE  proc_name
IS
BEGIN
   -- SQL statements
END;
/
```

4 ADVANTAGES

Modularity	Breaks complex tasks into smaller procedures.
Reusability	A procedure can be used multiple times in applications.
Better Performance	Stored and precompiled, so execution is faster.
Security	Access can be restricted using privileges.
Easy Maintenance	Changes are required in one place only.
Data Integrity	Helps maintain consistency and accuracy of data.
Code Organization	Makes the program structure clean and easy to understand.

5 DISADVANTAGES

Complex Debugging	Errors in procedures are sometimes difficult to trace.
Database Dependency	Applications become dependent on the database.
Not Suitable for Simple Tasks	Overhead of creating a procedure is not needed for small or single-use tasks.
Consumes Database Resources	Excessive use may increase memory and storage usage.
Changes Require Recompilation	Any change in the procedure needs recompilation.
Version Control Issues	Managing different versions of procedures can be challenging.

6 EXAMPLE

CREATE PROCEDURE

```
CREATE OR REPLACE PROCEDURE get_emp_salary
 (p_emp_id IN NUMBER,
  p_emp_name  OUT VARCHAR2,
  p_salary OUT NUMBER)
 IS
BEGIN
  SELECT ename, salary INTO p_emp_name, p_salary
  FROM employees
  WHERE emp_id  = p_emp_id;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   p_emp_name  := 'Not Found';
   p_salary := 0;
END;
/
```

CALL PROCEDURE

```
DECLARE
  v_name  VARCHAR2(50);
  v_sal   NUMBER;
BEGIN
  get_emp_salary(101, v_name,v_sal);
  DBMS_OUTPUT.PUT_LINE('Name   : ' || v_name);
  DBMS_OUTPUT.PUT_LINE('Salary : ' || v_sal);
END;
/
```

OUTPUT (Example)

```
Name   : Alex
Salary : 50000
```

DROP PROCEDURE

```
DROP PROCEDURE get_emp_salary;
```

Procedures help in writing structured, reusable and efficient database programs and are an essential part of PL/SQL.

Use Procedures to build powerful, secure, and maintainable database applications!

CREATION OF PROCEDURE

(Step-by-Step Guide)

A Procedure is a named PL/SQL block stored in the database that performs a specific task.

1. SYNTAX

```
CREATE OR REPLACE PROCEDURE procedure_name
(
  parameter1  IN  datatype1,
  parameter2  OUT  datatype2,
  parameter3  IN OUT  datatype3
)
IS
   -- Declarations
BEGIN
   -- Executable statements
EXCEPTION
   -- Exception handling (optional)
END procedure_name;
/
```

Where,

- IN → Value is passed to the procedure.
- OUT → Value is returned from the procedure.
- IN OUT → Value is both passed to and returned from the procedure.

2. STEPS TO CREATE A PROCEDURE

1. Decide the task that the procedure will perform.
2. Define the procedure name (meaningful and unique).
3. Declare the parameters with mode (IN / OUT / IN OUT) and data type.
4. Write the PL/SQL block:
   - Declarations (if any)
   - SQL statements / Logic
   - Exception handling (optional)
5. End the block with `END procedure_name;`
6. Execute the procedure using `CALL` statement.

3. EXAMPLE – CREATE PROCEDURE

```
-- Procedure to give employee a salary hike
CREATE OR REPLACE PROCEDURE give_hike
(       p_emp_id     IN    NUMBER,
        p_percent    IN    NUMBER,
        p_new_sal    OUT   NUMBER   )
IS
        v_old_sal   NUMBER;
BEGIN
        -- Get current salary
        SELECT salary INTO v_old_sal
        FROM employees
        WHERE emp_id  = p_emp_id;

        -- Calculate new salary
        p_new_sal := v_old_sal + (v_old_sal * p_percent / 100);

        -- Update salary
        UPDATE employees
        SET salary  =  p_new_sal
        WHERE emp_id  = p_emp_id;

END give_hike;
/
```

Explanation

Parameters:

- p_emp_id → Employee ID (IN)
- p_percent → Hike percentage (IN)
- p_new_sal → New salary (OUT)

Logic:

1. Fetch old salary
2. Calculate new salary
3. Update salary in table
4. Return new salary in OUT parameter

Sample Table (employees)

emp_id	emp_name	salary
101	Alex	50000
102	Ravi	45000
103	Neha	60000
...	...	...

4. HOW TO EXECUTE (CALL) THE PROCEDURE

```
DECLARE
   v_new_salary  NUMBER;
BEGIN
   CALL give_hike(101, 10, v_new_salary);  -- 10% hike
   DBMS_OUTPUT.PUT_LINE('New Salary:  ' || v_new_salary);
END;
/
```

Output (Example)

```
New Salary: 55000
```

5. DROP PROCEDURE

```
DROP PROCEDURE give_hike;
/
```

Use DROP to remove a procedure from the database.

KEY POINTS

- Procedure is stored and can be used multiple times.
- It improves modularity and reusability of code.
- It can accept parameters and return values.
- It is called using CALL statement.
- Changes in procedure require recompilation automatically.

USES OF PROCEDURE

- Encapsulate business logic
- Reduce application code
- Improve performance
- Enhance security
- Easier maintenance

EXAMPLE USE CASES

- Salary calculation
- Report generation
- Data validation
- Batch processing
- Complex business rules

Procedures make your database applications structured, reusable and efficient!

FUNCTION

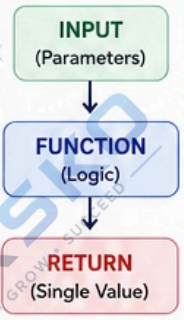
Introduction, Importance, Types, Advantages & Disadvantages

```
CREATE FUNCTION func_name
  ( parameters )
  RETURN data_type
IS
   -- statements
BEGIN
   -- logic
END;
```

1 INTRODUCTION

A Function is a named PL/SQL block that performs a specific task and **RETURNs** a single value.

- It can accept input parameters.
- It must return a value of the declared return type.
- It is stored in the database and can be called from SQL or PL/SQL.



2 IMPORTANCE

- Promotes code reusability.
- Improves modularity and structure of code.
- Reduces development time and effort.
- Can be used in SQL queries (SELECT, WHERE, etc.).
- Improves performance by minimizing repeated code.
- Enhances security by controlling access to logic.
- Simplifies complex calculations and business rules.

3 TYPES OF FUNCTIONS

A) BUILT-IN FUNCTIONS

Predefined functions provided by the system.

Example: `UPPER()`, `LENGTH()`, `NVL()`, `SYSDATE()`, etc.

Example

```
SELECT UPPER(emp_name) FROM employees;
```

B) USER-DEFINED FUNCTIONS

Created by users as per their requirements.

Stored in the database.

Can be called from SQL or PL/SQL.

Example

```
SELECT get_bonus(emp_id) FROM dual;
```

4 ADVANTAGES

Reusability	Functions can be used multiple times in programs.
Modularity	Breaks complex logic into small, manageable units.
Better Performance	Reduces redundant code and processing.
Security	Access can be controlled using privileges.
Easy Maintenance	Changes are required in one place only.
Data Integrity	Helps maintain consistency and accuracy.

5 DISADVANTAGES

Complex Debugging	Errors in functions are sometimes difficult to trace.
Database Dependency	Applications become dependent on the database.
Not Suitable for Simple Tasks	Overhead of creating a function is not required for small tasks.
Performance Overhead	Function calls may add overhead in some cases.
Changes Require Recompilation	Any change in the function needs recompilation.
Version Control Issues	Managing different versions of functions can be challenging.

6 EXAMPLE OF USER-DEFINED FUNCTION

```
-- Function to calculate bonus
CREATE OR REPLACE FUNCTION get_bonus
  ( p_salary IN NUMBER )
   RETURN NUMBER
IS
    v_bonus NUMBER;
BEGIN
   IF p_salary > 50000 THEN
      v_bonus := p_salary * 0.10;   -- 10% bonus
   ELSE
      v_bonus := p_salary * 0.05;   -- 5% bonus
   END IF;
   RETURN v_bonus;
END get_bonus;
/
```

Calling the Function

```
SELECT get_bonus(60000) AS bonus FROM dual;
```

Output

```
BONUS
------
6000
```

FUNCTION vs PROCEDURE

Aspect	Function	Procedure
Return Value	Returns a single value	Does not return any value
Usage in SQL	Can be used in SQL queries	Cannot be used in SQL queries
CALL Statement	Not called using CALL	Called using CALL statement
Purpose	Performs calculation and returns result	Performs actions or tasks
Output	Must return a value	May have OUT parameters

KEY POINTS

- A Function must RETURN a value.
- It can be used in expressions and SQL statements.
- It improves code reusability and maintainability.
- It should be used when a result is needed.

Functions make your code powerful, reusable, and easy to maintain!

CREATION OF FUNCTION

Step-by-Step Guide

A Function is a named PL/SQL block that performs a specific task and **RETURNS** a single value.

```
CREATE FUNCTION func_name
   ( parameters )
RETURN return_type
IS
   -- statements
BEGIN
   -- logic
END;
```

1. SYNTAX

```
CREATE OR REPLACE FUNCTION function_name
( parameter1    IN  datatype1,
  parameter2    IN  datatype2 DEFAULT value, ... )
 RETURN return_datatype
IS
   -- Declarations (optional)
BEGIN
   -- Executable statements
   RETURN value;
EXCEPTION
   -- Exception handling (optional)
END function_name;
/
```

- `IN` → Value is passed to the function.
- `DEFAULT` → Default value if argument is not supplied.
- `RETURN` → The value returned by the function.

2. STEPS TO CREATE A FUNCTION

1. Decide the task that the function will perform.
2. Choose a meaningful name for the function.
3. Define the parameters with mode (IN) and data type.
4. Specify the return data type.
5. Write the PL/SQL block:
   - Declarations (if any)
   - Executable statements / Logic
   - RETURN statement (mandatory)
6. End the block with END function_name;
7. Compile and execute the function using SELECT statement.

3. EXAMPLE – CREATE FUNCTION

```
-- Function to calculate bonus
CREATE OR REPLACE FUNCTION get_bonus
( p_salary   IN  NUMBER)
RETURN NUMBER
IS
   v_bonus  NUMBER;
BEGIN
    IF p_salary >= 50000 THEN
        v_bonus := p_salary * 0.10;    -- 10% bonus
    ELSIF p_salary >= 30000 THEN
        v_bonus := p_salary * 0.05;    -- 5% bonus
    ELSE
        v_bonus := 0;                  -- No bonus
    END IF;
    RETURN v_bonus;
END get_bonus;
/
```

Explanation

- `p_salary IN NUMBER` → Input parameter (Employee salary)
- `RETURN NUMBER` → Function returns a NUMBER value
- `Logic` → Calculates bonus based on salary
- `RETURN v_bonus` → Returns the calculated bonus

Sample Data

Employee	Salary	get_bonus(salary)	
Asha	60000	6000	(10%)
Ravi	35000	1750	(5%)
Neha	25000	0	(0%)

4. HOW TO CALL (EXECUTE) THE FUNCTION

```
-- Using SELECT statement
SELECT get_bonus(60000) AS bonus FROM dual;
```

Output

```
BONUS
-------
6000
```

Functions are called in SQL queries and must return a value.

5. DROP FUNCTION

```
DROP FUNCTION get_bonus;
/
```

Use DROP to remove a function from the database.

6. ADVANTAGES

- Promotes code reusability.
- Improves modularity and readability.
- Reduces development time and effort.
- Can be used in SQL queries and calculations.
- Improves performance by reducing repeated logic.
- Enhances security and data integrity.

7. DISADVANTAGES

- Complex debugging.
- Additional overhead in some cases.
- Dependent on database.
- Not suitable for operations that do not return a value.
- Changes require recompilation.

FUNCTION vs PROCEDURE

Aspect	Function	Procedure
Returns Value	Yes (Mandatory)	No (Optional)
Used in	SQL & PL/SQL	PL/SQL only
CALL Statement	Not used	Used to execute
Purpose	To return a value	To perform actions
Output	Single value	No return value

Functions make your code powerful, reusable, and efficient!
Write once, use many times.

TRIGGERS

Introduction, Importance, Types, Advantages & Disadvantages

FIRES AUTOMATICALLY
in response to events

```
CREATE TRIGGER trg_name
{
  -- Trigger Body
}
```

1 INTRODUCTION

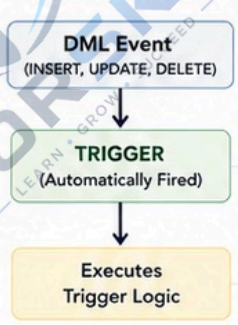
A Trigger is a special type of stored procedure that is **automatically executed (fired)** when a specific event occurs on a table or view.

Key Points

- It is associated with a table or view.
- It is executed implicitly (not called directly).
- It helps enforce integrity and business rules.
- It runs in response to DML events.

2 IMPORTANCE

- Enforces business rules automatically.
- Maintains data integrity.
- Reduces application code and complexity.
- Logs activities and audits changes.
- Prevents invalid data modification.
- Centralizes logic at the database level.



3 TYPES OF TRIGGERS

A) BASED ON TIMING

1. BEFORE TRIGGER
Fires before the triggering event occurs on the table.

2. AFTER TRIGGER
Fires after the triggering event has occurred.

B) BASED ON EVENT

INSERT TRIGGER
Fires when a row is inserted.

UPDATE TRIGGER
Fires when a row is updated.

DELETE TRIGGER
Fires when a row is deleted.

C) BASED ON LEVEL

ROW LEVEL TRIGGER
Fires once for each row affected by the event.

STATEMENT LEVEL TRIGGER
Fires once for the entire SQL statement, not for each row.

4 ADVANTAGES

- **Automated Execution:** Runs automatically on events.
- **Enforces Business Rules:** Ensures data consistency.
- **Improves Data Integrity:** Validates data at the database level.
- **Audit & Logging:** Tracks changes for security and auditing.
- **Reduces Redundancy:** Centralized logic, less application code.
- **Enhances Security:** Prevents unauthorized or invalid actions.

5 DISADVANTAGES

- **Complexity:** Harder to debug and maintain.
- **Performance Overhead:** May slow down DML operations.
- **Hidden Logic:** Business logic is not visible to end users.
- **Recursive Triggers:** Can cause infinite loops if not handled.
- **DB Dependency:** Portability becomes database-specific.
- **Impact on Bulk Operations:** Triggers fire for every affected row.

6 EXAMPLE

Example: After Insert Trigger (Row Level)

```
CREATE OR REPLACE TRIGGER trg_after_insert_emp
AFTER INSERT ON employees
FOR EACH ROW
BEGIN
  INSERT INTO emp_audit(emp_id, action, action_date)
  VALUES (:NEW.emp_id, 'INSERT', SYSDATE);
END;
/
```

Explanation

- **AFTER INSERT:** Fires after a row is inserted.
- **:NEW** refers to the new row data.
- Logs the action in emp_audit table.

Events and When Triggers Fire

Event	Before Trigger	After Trigger
INSERT	Before row is inserted	After row is inserted
UPDATE	Before row is updated	After row is updated
DELETE	Before row is deleted	After row is deleted

QUICK SUMMARY

- Triggers are automatic stored procedures that fire on DML events.
- **Timing:** BEFORE and AFTER
- **Events:** INSERT, UPDATE, DELETE
- **Level:** ROW LEVEL and STATEMENT LEVEL
- **Benefits:** Integrity, automation, security, audit
- **Limitations:** Complexity, performance, maintenance

USE CASES

- Enforce business rules
- Maintain audit trails
- Validate data
- Log user activities
- Update summary tables automatically

Triggers automatically enforce rules and maintain integrity—making your database smarter and safer!

BACKUP AND RECOVERY
INTRODUCTION

Backup and Recovery is the process of creating a copy (**backup**) of data and restoring it (**recovery**) in case of **data loss, corruption, or system failure.**

1. INTRODUCTION

- Data is a **critical asset** for any organization.
- **Data loss** can occur due to hardware failure, software errors, human mistakes, natural disasters, or cyber attacks.
- **Backup** ensures a safe copy of data.
- **Recovery** helps to restore data and resume normal operations.
- The main goal is to ensure **data availability, integrity,** and **reliability.**

2. IMPORTANCE

- **Protects Data:** Prevents permanent loss of important data.
- **Ensures Business Continuity:** Helps resume business quickly after failure.
- **Minimizes Downtime:** Reduces time required to restore systems.
- **Meets Legal & Compliance Requirements:** Helps in maintaining regulatory and audit standards.
- **Provides Peace of Mind:** Ensures data safety and reliability.

3. COMMON CAUSES OF DATA LOSS

Hardware Failure	Software Errors	Human Errors
Natural Disasters	Viruses / Cyber Attacks	Power Failure

4. KEY TERMS

Backup	:	Creating a copy of data that can be used to restore the original data.
Recovery	:	The process of restoring data from a backup after data loss.
Restore	:	Copying data from backup media to the original system/location.
Redundancy	:	Keeping extra copies of data to prevent data loss.
Disaster Recovery	:	A serious event causing major data loss. Process of restoring systems and data after a disaster.

In short:

Backup protects your data today, and **Recovery** brings it back when you need it the most.

ARCHIVE (TRANSACTION) LOGS

Ensuring Data Durability and Point-in-Time Recovery

1. INTRODUCTION

Archive (Transaction) Logs are records of all changes made to the database. They are written to disk **before the actual data changes occur** and are used to **recover** the database in case of failure.

Key Points

- Contain information about every transaction.
- Stored sequentially as they are generated.
- Essential for recovery and maintaining data integrity.
- Used in combination with backups for point-in-time recovery.

2. IMPORTANCE

Ensures Durability
Guarantees that committed transactions are not lost.

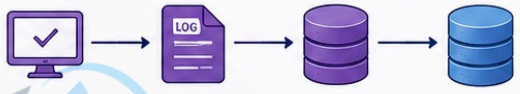
Enables Recovery
Helps restore the database after failures.

Supports Point-in-Time Recovery
Allows restoring the database to any specific point in time.

Maintains Consistency
Ensures ACID properties are preserved.

Improves Reliability
Crucial for high availability and disaster recovery strategies.

3. HOW ARCHIVE LOGS WORK



Write-Ahead Logging (WAL) Principle

The log record is written to stable storage (archive log) **before** the actual change is written to the database.
This ensures that the change can be **REDONE** during recovery.

4. ARCHIVE LOG FLOW

When a online redo log file becomes full, it is archived to archive destination and a new log file is used.

- Disk Location
- Tape (Legacy)
- Cloud Storage

5. ARCHIVE LOG MODES

NOARCHIVELOG Mode

- Archived logs are not created.
- Old online redo logs are overwritten.
- Point-in-time recovery is NOT possible.
- Suitable for non-critical databases.

ARCHIVELOG Mode

- Archived logs are created.
- Enables point-in-time recovery.
- Older logs are preserved.
- Recommended for critical databases.

6. BENEFITS

- Data Protection
- Point-in-Time Recovery
- Audit & Compliance
- Disaster Recovery
- High Availability

7. CONSIDERATIONS

- Consumes storage space.
- Management and monitoring are important.
- Regular archiving and backup of logs are necessary.
- Too many small archive logs may impact performance.

EXAMPLE

If a transaction updates a customer balance from 1000 to 1500, the archive log will record:

- Transaction ID
- Table Name
- Row ID
- Old Value (1000)
- New Value (1500)
- Timestamp

8. LOG RECORD CONTAINS

IN SHORT

Archive (Transaction) Logs are the backbone of database recovery. They ensure that your data is safe, recoverable and consistent — even in the event of failures.

IMPORTANCE OF BACKUPS

Why Backups Are Essential

A **backup** is a copy of data that can be used to restore the original data in case of loss, corruption, or any unexpected event. Backups are crucial for **data protection**, **business continuity**, and **peace of mind**.

1 PREVENTS DATA LOSS

Protects your important data from accidental deletion, hardware failure, software errors, viruses, or cyber attacks.

2 ENSURES BUSINESS CONTINUITY

Allows your business to recover quickly and continue operations with minimal downtime after a failure or disaster.

3 ENABLES QUICK RECOVERY

Helps restore data to a specific point in time, reducing recovery time and getting you back on track faster.

4 MAINTAINS DATA INTEGRITY

Ensures the accuracy, consistency, and reliability of your data over time.

5 MEETS LEGAL & COMPLIANCE NEEDS

Helps meet regulatory and industry requirements for data retention, protection, and audit purposes.

6 SUPPORTS DISASTER RECOVERY

Essential for recovering from major disasters like natural calamities, fires, floods, or system-wide failures.

7 PROTECTS AGAINST HUMAN ERRORS

Helps undo mistakes like accidental deletion, overwriting data, or incorrect changes.

8 PROVIDES PEACE OF MIND

Knowing your data is safely backed up gives confidence and reduces stress during unexpected situations.

KEY TAKEAWAY

Backups are not just a good practice – **they are a necessity.**
They protect your **data**, ensure **business continuity**, and save **time, money, and reputations.**

DATABASE RECOVERY

Restoring Data, Restoring Trust

Database Recovery is the process of restoring a database to a consistent and correct state after a failure.
It ensures data durability and availability, even in the presence of system crashes, media failures, or human errors.

1. INTRODUCTION

Failures can occur due to:

- System crash
- Media (disk) failure
- Power failure
- Human errors
- Natural disasters

Goal: Restore the database to a consistent state with no loss of committed data.

2. IMPORTANCE

Ensures Data Durability
No committed data is lost.

Maintains Database Consistency
Brings the database to a correct state.

Minimizes Downtime
Quick recovery helps resume operations.

Protects Against Failures
Handles crashes, media failures, and disasters.

Supports Business Continuity
Keeps the organization running smoothly.

3. TYPES OF FAILURES

Transaction Failure	A transaction cannot complete due to logical or system errors.
System Failure	Database is intact, but main memory contents are lost.
Media Failure	Storage media is damaged or lost.
Catastrophic Failure	Loss of database and storage.

4. RECOVERY PROCESS

Failure → Detect Failure → Analyze Logs → Restore Database → Database in Consistent State

1. **Detect Failure:** System detects a failure has occurred.
2. **Analyze Logs:** Check the log files to find the last committed transaction.
3. **Restore Database:** Use backups and logs to undo uncommitted transactions and redo committed ones.
4. **Consistent State:** Database is restored to a consistent and correct state.

5. TECHNIQUES USED

Transaction Logs
Records all changes in the database.

UNDO (Roll Back)
Undoes all uncommitted transactions.

REDO (Roll Forward)
Reapplies committed transactions.

Backup
Used in case of media or catastrophic failure.

6. WAL (WRITE-AHEAD LOGGING)

Principle:
Log record is written to stable storage (log) before the actual data is written to the database.

Benefit:

- Ensures atomicity and durability.
- Enables recovery using logs.
- Prevents loss of committed data.

7. EXAMPLE

Transaction	Status
T1	Committed
T2	Committed
T3	Not Committed

If failure occurs:

- T3 is UNDO (rolled back)
- T1 and T2 are REDO (reapplied)

→ Database is restored to the state after T2 committed.

8. KEY TAKEAWAY

- Recovery ensures reliability and trust in the database.
- Logs + Backup = Powerful recovery mechanism.
- UNDO removes incomplete changes.
- REDO replays completed changes.
- Goal is always a consistent and correct database.

IN SHORT

Database Recovery brings the database back to life after failure — ensuring no committed data is lost and business can continue.

Good recovery is not just about restoring data, it's about restoring confidence in your system.

TRANSACTION CONTROL

Managing Transactions to Ensure Data Integrity and Consistency

Transaction Control is the mechanism used by a DBMS to manage the execution of transactions, ensuring that database remains **consistent**, **reliable** and **recoverable** even in case of failures.

1. INTRODUCTION

A transaction is a sequence of read/write operations performed as a single logical unit of work. Transaction Control ensures that transactions are executed correctly and do not affect database consistency.

Example

Funds Transfer: Debit from Account A and Credit to Account B.
Both operations must complete successfully or none at all.

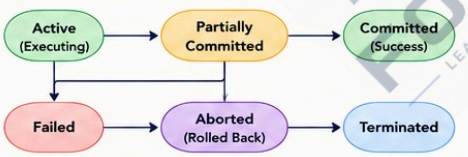
2. OBJECTIVES

- **Maintain Consistency** — Ensure database remains in a consistent state.
- **Ensure Atomicity** — All operations of a transaction are completed or none.
- **Ensure Isolation** — Concurrent transactions do not interfere with each other.
- **Ensure Durability** — Once a transaction is committed, changes are permanent.
- **Recoverability** — Database can be recovered in case of failure.

3. ACID PROPERTIES

A	Atomicity	All operations in a transaction are treated as a single unit.
C	Consistency	Transaction brings the database from one consistent state to another.
I	Isolation	Transactions execute independently without affecting each other.
D	Durability	Once committed, the changes are permanent even after failures.

4. TRANSACTION STATES



- **Active** : Transaction is being executed.
- **Partially Committed** : Last operation executed.
- **Committed** : Transaction completed successfully.
- **Failed** : An error or crash occurs.
- **Aborted** : Transaction rolled back.
- **Terminated** : Transaction ends after commit or rollback.

5. CONTROL OPERATIONS

- **BEGIN TRANSACTION** — Marks the start of a transaction.
- **COMMIT** — Makes all changes permanent.
- **ROLLBACK** — Undoes all changes of the transaction.
- **SAVEPOINT** — Sets a point within a transaction to which we can rollback.
- **LOCK / UNLOCK** — Controls access to data items to ensure isolation.

6. CONCURRENCY CONTROL

- Multiple transactions may run concurrently.
- Concurrency Control ensures isolation and serializability.
- Common Techniques:
  - **Lock-Based Protocols** (e.g., Two-Phase Locking)
  - **Timestamp Ordering Protocol**
  - **Validation / Optimistic** Concurrency Control

7. BENEFITS

- Data Integrity and Accuracy
- Reliable and Consistent System
- Failure Recovery
- Supports Concurrency
- Better Database Performance

In Short: Transaction Control ensures that transactions are executed safely and reliably, maintaining the correctness of the database even in the presence of errors, failures and concurrent access.