**Unit IV Persistent Storage and Data Transactions(A)**

1. **Introduction to Persistent Storage**

Data that is saved even after a system restart or shutdown is referred to as persistent storage. It is essential for applications information retrieval and maintenance.

1. **Configuring Long-Term Storage**

The value of long-term storage guarantees long-term access and data durability.

Supports a wide range of applications including enterprise software, online systems, and data analytics.

Permits database to have transactional integrity.

Types of Persistent Storage:

* File system is used to store data in structured file formats, such as CSV, JSON, and XML.
* Structured tables and relationships are used in relational databases (SQL-based).
* NoSQL databases offer scalable and adaptable unstructured data storage.
* Cloud storage is the remote storage of data for scalability and ease of access.

1. **SQL and NoSQL Database Configuration**

**Relational databases, or SQL databases**

Relational databases such as MySQL, PostgreSQL, and SQLite employ Structured Query Language (SQL) to manage and work with structured data.

How to Configure a SQL Databases:

* Install a database server, such as PostgreSQL or MySQL.
* Make a database and specify its schema, including its tables, columns, and relationships.
* To insert data (INSERT), retrieve it (SELECT), update it (UPDATE), and delete it (DELETE), use SQL commands.
* Use normalization and indexing to maximize performance.
* Use access control and authentication to safeguard the database.

**Non-Relational Databases, or NoSQL databases**

Data can be stored in a variety of formats, including documents, graphs, and key-value pairs, in NoSQL databases like MongoDB, Firebase, and Cassandra.

How to Configure a NoSQL Databases:

* Select the NoSQL type (document-based, key-value, graph, or column-family) according to the use case.
* Install a NoSQL database program, such as MongoDB for storing documents.
* In place relational tables, define collection and documents (for MongoDB).
* To add, retrieve, edit, and remove documents, use queries.
* Utilize caching and indexing to maximize database speed.

1. **Connecting Database and Backends to Provide Data**

Appropriate connectivity setups are necessary to guarantee smooth communication between database and the backend application.

**Linking a Backend to SQL Database**

Connect SQL database to apps using ODBC/JDBC connectors.

To handle database interactions, use ORM frameworks such as Hibernate (Java) or SQL Alchemy (Python).

Create API endpoints so that database records can be manipulated and queried.

**Linking a Backend to NoSQL Databases**

NoSQL Databases for application integration, use Firebase SDKs or MongoDB drivers.

Use GraphQL ot RESTful APIs to retrieve and save data.

1. **Database Management Backup and Recovery Best Practices:**

Make regular backups of your data to guard against loss.

Put access control, authentication, and encryption into practice as security measures.

Performance Optimization: Make use of query optimization and indexing strategies.

Data Integrity: Verify that SQL databases adhere to ACID (Atomicity Consistency, Isolation, Durability) standards.

Scalability: Select suitable database options according to application requirements and data volume.

**Conclusion**

Effective data maintenance and retrieval depends on persistent storage. Reliable and scalable data management is ensured by knowing SQL and NoSQL databases, connectivity, and adhering to best practices.