*1. Functional Dependency**
* **Definition**: A relationship where one attribute uniquely determines another (e.g., A → B means B is functionally dependent on A).
* **Helps**: Ensures data accuracy, avoids redundancy, and maintains data integrity by enforcing relationships between attributes.
2. Armstrong's Axioms
* **Rules**:
 1. *Reflexivity*: If Y ⊆ X, then X → Y
 2. *Augmentation*: If X → Y, then XZ → YZ
 3. *Transitivity*: If X → Y and Y → Z, then X → Z
* **Use**: Help derive all possible functional dependencies from a given set—used in normalization and schema design.
3. Lossless Decomposition
* **Definition**: Breaking a table into smaller tables without losing information.
* **Helps**: Ensures no spurious (incorrect) tuples are generated when relations are joined back.
4. Dependency Preservation
* **Definition**: All functional dependencies are preserved after decomposition.
* **Crucial**: Ensures constraints can still be enforced without joining tables, maintaining consistency and integrity.
5. First Normal Form (1NF)
* **Definition**: A table is in 1NF if it has only atomic (indivisible) values and no repeating groups.
* **Example**:
 Not 1NF:
 Student | Courses
 John | Math, English
 In 1NF:
 Student | Course
 ———|———
 John | Math
 John | English
 ```
**6. 2NF vs. 3NF**
* **2NF**: Removes partial dependencies (non-prime attribute depends on part of a composite key).
* **3NF**: Removes transitive dependencies (non-prime depends on another non-prime).
* **Example**:
 In 2NF but not 3NF:
 EmpID, DeptID → DeptName (transitive)
 ```
7. Boyce-Codd Normal Form (BCNF)
* **Definition**: A stronger version of 3NF where for every dependency X → Y, X is a super key.
* **Improves**: Removes anomalies not handled by 3NF.
* **Example**:
 StudentID, Course → Instructor
 Instructor → Course
8. Normalization Pros & Cons
* **Advantages**:
 * Eliminates redundancy
 * Improves data integrity
 * Easier maintenance
* **Disadvantages**:
 * Complex queries
 * More joins
 * Performance overhead
9. Multivalued Dependencies
* **Definition**: When one attribute determines multiple independent values of another.
* **Example**:
 Student →→ Course
 Student →→ Hobby
 * Impact: May require 4NF to remove redundancy caused by multivalued facts.
10. Normalization vs. Denormalization
* **Normalization**: Split tables to reduce redundancy (used in OLTP).
* **Denormalization**: Combine tables to improve read performance (used in OLAP).
* **Use**:
 * Normalize for data integrity.
 * Denormalize for faster querying and performance.
[5/9, 9:39 PM] Aryan: Here are the **main points** for questions 11–20 related to **PL/SQL**:
11. What is PL/SQL?
* **Definition**: Procedural Language extension of SQL in Oracle.
* **Advantages over SQL**:
 * Supports loops, conditions, and modular programming
 * Better performance for complex operations
 * Improved error handling
12. Structure of a PL/SQL Block
* **Sections**:
 1. `DECLARE` – for variables
 2. `BEGIN` – main code
 3. `EXCEPTION` – error handling
 4. `END;`
* **Example**:
 ~~~plsql
 DECLARE
   v_name VARCHAR2(20);
 BEGIN
   v_name := 'John';
   DBMS_OUTPUT.PUT_LINE(v_name);
 END;
**13. Control Structures in PL/SQL**
* **IF**:
 ```plsql
 IF x > 0 THEN ... END IF;
* **CASE**:
 ```plsql
 CASE x WHEN 1 THEN ... ELSE ... END;
* **LOOP**:
 ```plsql
 LOOP ... EXIT WHEN x > 5; END LOOP;
* **WHILE**:
 ```plsql
 WHILE x < 10 LOOP ... END LOOP;
**14. Cursors in PL/SQL**
* **Definition**: Pointer to the result of a query.
* **Implicit Cursor**: Automatically created for single-row queries (e.g., `SELECT INTO`).
* **Explicit Cursor**: Declared by the programmer for multi-row queries.
* **Example**
 ```plsql
 CURSOR c1 IS SELECT * FROM emp;
**15. Stored Procedures vs Functions**
* **Stored Procedure**: Executes actions, no return value required.
* **Function**: Must return a value.
* **Example**:
 ```plsql
 PROCEDURE greet(name VARCHAR2) IS ...
 FUNCTION add(x INT, y INT) RETURN INT IS ...
**16. Triggers in PL/SQL**
* **Definition**: Code that runs automatically on specific DB events.
* **Types**:
 * BEFORE / AFTER INSERT, UPDATE, DELETE
* **Example**:
 ```plsql
 CREATE TRIGGER trg BEFORE INSERT ON emp FOR EACH ROW BEGIN ... END;
**17. Exception Handling**
* **Predefined**: Built-in (e.g., `NO_DATA_FOUND`)
* **User-defined**: Declared by the user
* **Example**:
 ``plsql
 EXCEPTION
   WHEN NO_DATA_FOUND THEN ...
   WHEN my_exception THEN ...
**18. PL/SQL Factorial Using Loops**
 **FOR**:
 ```plsql
 FOR i IN 1..n LOOP fact := fact * i; END LOOP;
* **WHILE**:
 ```plsql
 WHILE i <= n LOOP ... END LOOP;
 ~~~
* **LOOP with EXIT**:
 ~~~plsql
 LOOP EXIT WHEN i > n; ... i := i + 1; END LOOP;
**19. Transaction Management**
* **COMMIT**: Saves changes
* **ROLLBACK**: Reverts changes
* **SAVEPOINT**: Sets a checkpoint to rollback to
* **Example**:
 ```plsql
 SAVEPOINT sp1;
 ROLLBACK TO sp1;
**20. Advantages of Stored Procedures/Functions**
* Modular and reusable
* Improved performance
* Centralized logic
* Enhanced security (controlled access)
[5/9, 9:39 PM] Aryan: Here are the **main points** for questions 21–30 on **database security and recovery**:
**21. What is Database Security?**
* **Definition**: Protecting data from unauthorized access or corruption.
* **Authentication**: Verifies user identity (e.g., username/password).
* **Authorization**: Grants permissions (e.g., read/write access).
* **Access Control**: Enforces rules (e.g., user A can't access table B).
**22. Access Control Models**
* **DAC (Discretionary Access Control)**: Owner controls access (e.g., GRANT in SQL).
* **MAC (Mandatory Access Control)**: Access based on clearance levels (e.g., military).
* **RBAC (Role-Based Access Control)**: Permissions assigned to roles, not individuals.
**23. SQL Injection**
* **Definition**: Inserting malicious SQL into input fields.
* **Occurs**: e.g., `' OR '1'='1` bypassing login.
* **Prevention**:
 * Use prepared statements
 * Input validation
 * Stored procedures
 * Web application firewalls
**24. Intrusion Detection Systems (IDS)**
* **Definition**: Monitors database activity for suspicious behavior.
* **Helps**: Alerts admins, blocks intrusions, maintains logs
**25. Auditing and Logging**
* **Auditing**: Tracks who did what and when.
* **Logging**: Records DB operations/events.
* **Importance**: Helps with security analysis and forensic investigations.
**26. Types of Backups**
* **Full Backup**: Complete copy of database.
* **Incremental**: Only changes since last backup.
* **Differential**: Changes since last full backup.
* **Example**:
 * Mon: Full
 * Tue: Incremental
 * Wed: Differential
**27. Types of Database Failures**
* **Types**:
 * Transaction failure
 * System crash
 * Media failure
* **Recovery**: Backups + logs restore to last consistent state.
**28. Cold vs Hot Backup**
* **Cold Backup**: DB is offline—safe, consistent
* **Hot Backup**: DB is online—used in 24/7 systems
* **Use**: Cold when downtime is acceptable; hot for live systems.
**29. Database Recovery Process**
* **Recovery**: Uses logs to restore consistency
* **Techniques**:
 * Rollback (undo)
 * Rollforward (redo)
 * Shadow paging
 * Checkpointing
**30. Encryption in Database Security**
* **Purpose**: Protect data at rest and in transit
* **Techniques**:
 * **Transparent Data Encryption (TDE)** – encrypts entire DB
 * **Column-level encryption** – sensitive fields only
 * **SSL/TLS** – encrypts network communication
[5/9, 9:39 PM] Aryan: Here are the **main points** for questions 31–40 on **advanced database concepts
***31. Distributed Databases**
* **Definition**: Database stored across multiple physical locations.
* **Advantages**: Improved availability, scalability, and performance.
* **Challenges**: Data consistency, synchronization, complex query processing.
**32. Data Fragmentation**
* **Concept**: Breaking data into pieces stored across sites.
* **Types**:
 * **Horizontal**: Rows split (e.g., customers by region)
 * **Vertical**: Columns split (e.g., separating personal and financial data)
 * **Mixed (Hybrid)**: Combination of both
**33. Data Replication**
* **Definition**: Copying data across multiple sites.
* **Benefits**: High availability, fault tolerance, faster reads.
* **Drawbacks**: Data inconsistency, synchronization overhead.
**34. Distributed Query Processing**
* **Definition**: Executing queries over data stored at different locations.
* **Differs from Centralized**: Involves data transfer, site selection, and distributed optimization.
**35. Object-Relational Databases (ORDBMS)**
* **Definition**: Combines relational model with object-oriented features.
 ~~~

* **Differences**: Supports complex data types (e.g., user-defined types, inheritance).
* **Example**: PostgreSQL with custom data types.
36. Relational vs NoSQL Databases
* **RDBMS**: Structured data, ACID compliance, fixed schema
* **NoSQL**: Unstructured/semi-structured data, scalable, flexible schema
* **Use NoSQL**: For big data, real-time analytics, document stores (e.g., MongoDB)
37. Temporal Databases
* **Definition**: Handles time-varying data (past, present, future states).
* **Example**: Employee salary history over time
* **Real-World Use**: Financial, medical, HR systems
38. Spatial Databases
* **Definition**: Stores and queries spatial/geographic data.
* **Applications**: GIS, GPS navigation, urban planning
* **Features**: Spatial indexing, geometry types (points, polygons)
39. Multimedia Databases
* **Differences**: Store images, video, audio—not just text/numbers
* **Example**: Digital libraries, social media apps
* **Challenges**: Large storage, complex indexing, content-based retrieval
40. CAP Theorem
* **Concept**: In distributed systems, only two of the following three can be guaranteed at once:
  * **Consistency**
  * **Availability**
  * **Partition Tolerance**
* **Significance**: Influences design trade-offs in distributed databases.