ONESHOT DBMD NOTES

DBMD ONESHOT NOTES (UNITS 1-4)

UNIT-I: CONCEPTUAL DATA MODELLING

1. **Introduction**
    - **Overview of Database Systems Architecture and Components [PYQ Q1a, Q2a]**
        - **Data, Information, Metadata:**
            - **Data:** Raw, unorganized facts.
            - **Information:** Processed data with meaning in context.
            - **Metadata:** Data describing properties of data (structure, types, constraints).
        - **Data Management:** Creation, retrieval, modification, deletion of data. Access Methods (Sequential, Direct), Organization (Sequential, Random/hashing, Indexed).
        - **Limitations of File-Processing Systems [PYQ - Implied in understanding DB advantages]:** Lack of data integrity, standards, flexibility/maintainability. Root Causes: Data separation/isolation, program-data dependence.
        - **ANSI/SPARC Three-Schema Architecture [PYQ - Important for data independence]:**
            - **Purpose:** Achieve program-data independence.
            - **Levels:**
                - **External Schema (User Views/Subschemas):** Individual user/application views of relevant DB portions.
                - **Conceptual Schema (Global View):** Community view of entire DB (entities, relationships, constraints). Technology-independent. Hides physical details.
                - **Internal Schema (Physical View):** Describes physical storage structures, access paths (indexes, hashing). Technology-dependent.
            - **Data Independence:**
                - **Logical Data Independence:** Immunity of external schemas to conceptual schema changes.
                - **Physical Data Independence:** Immunity of conceptual (and external) schemas to internal schema changes.
        - **Database System vs. DBMS [PYQ Q1a]:**
            - **Database:** Self-describing collection of interrelated data (includes data & metadata). Types: Single-user, Multi-user (workgroup, enterprise), Distributed (DDB), Data Warehouse.

        - **Database Management System (DBMS):** General-purpose software to define, construct, manipulate a database.
        - **Components [PYQ Q1a]:** Query Languages (SQL), Report Generators, Security/Integrity/Backup & Recovery facilities, Data Definition Language (DDL), Data Manipulation Language (DML), Data Control Language (DCL), Data Dictionary/Repository.
    - **Advantages of Database Systems:** Controlled redundancy, improved data integrity/consistency, sharing, standards enforcement, security, program-data independence, productivity.
- **Database Design Life Cycle [PYQ Q2b]:**
    1. **Requirements Specification:** Analyst review, user interviews for objectives, data, process specs (business rules).
    2. **Conceptual Data Modeling (Technology-Independent):** Describes data structure without physical storage. Captures business rules. Product: Conceptual Schema (ER/EER model). Includes Presentation Layer (user comm.) & Design-Specific Layer (DB design). Validation crucial.
    3. **Logical Data Modeling (Technology-Dependent):** Transforms conceptual to chosen data model (relational, network). Normalization is part. Product: Logical Schema.
    4. **Physical Data Modeling (DBMS-Specific):** Specifies internal storage, access strategies using DBMS tools.

2. **Conceptual Data Modelling**
    - **ER Modeling [PYQ Q3b, Q4a]:** Represents real-world phenomena using grammar (constructs & rules) and a method.
        - **ER Modeling Primitives:** Object (type/occurrence) -> Entity (type/instance); Property -> Attribute; Fact -> Value; Association -> Relationship; Object class -> Entity class.
        - **Entity Type:** Collection of similar objects (e.g., STUDENT). Represented by a rectangle.
        - **Entity Instance:** Specific occurrence of an entity type (e.g., student John Doe).
        - **Attribute:** Property/characteristic of an entity type (e.g., StudentName). Represented by an oval.
        - **Types of Attributes:**
            - **Simple (Atomic):** Cannot be subdivided (e.g., Age).
            - **Composite:** Subdividable (e.g., Address -> Street, City).
            - **Single-valued:** One value per instance (e.g., DateOfBirth).
            - **Multi-valued:** Multiple values per instance (e.g., Skills). Double oval.

    - **Stored:** Directly stored value (e.g., BirthDate).
    - **Derived:** Calculated from other attributes (e.g., Age from BirthDate). Dotted oval.
    - **Mandatory:** Must have a value (dark circle).
    - **Optional:** May not have a value (empty circle).
    - **Complex:** Nested composite and/or multi-valued.
    - **Domain:** Set of possible values for an attribute.
- **Unique Identifiers (Keys): [Related to PYQ Q1b]** Attribute(s) uniquely identifying each entity instance. Underlined in ERD.
    - **Candidate Key:** Minimal set of attributes uniquely identifying an instance.
    - **Primary Key:** Candidate key chosen as main identifier.
    - **Superkey:** Any set of attributes uniquely identifying an entity; may contain redundancy.
    - **Key attribute:** Part of a candidate key.
- **Relationship Type:** Meaningful association among entity types (e.g., ENROLLS between STUDENT, COURSE). Represented by a diamond.
- **Relationship Instance:** Specific association between entity instances.
- **Degree of a Relationship:** Number of participating entity types.
    - **Binary (degree 2):** Two entity types.
    - **Ternary (degree 3):** Three entity types.
    - **N-ary (degree n):** 'n' entity types.
    - **Recursive (Unary):** Relationship between instances of the same entity type.
- **Role Names:** Clarify role of an entity type in a relationship.
- **Structural Constraints (Cardinality & Participation):**
    - **Cardinality Ratio (Connectivity/Max):** Maximum number of relationship instances an entity can participate in (1:1, 1:N, M:N).
    - **Participation Constraint (Min):** Whether an entity's existence depends on being related.
        - **Total (Mandatory):** Every instance must participate (bar or (1,N)).
        - **Partial (Optional):** Instance may or may not participate (oval or (0,N)).
- **Attributes on Relationships:** Can exist, especially for M:N or if attribute describes interaction.
- **Base (Strong) Entity Type:** Has its own unique identifier; exists independently.
- **Weak Entity Type:** No unique identifier of its own; existence depends on strong (owner) entity. Identified by owner's PK + its partial key.
    - **Identifying Relationship:** Links weak entity to owner. Double diamond.

        - **Partial Key (Discriminator):** Attribute(s) distinguishing weak entity instances related to same owner. Dotted underline.
    - **Data Modeling Errors:**
        - **Semantic Errors:** Misinterpretation of requirements.
        - **Syntactic Errors:** Violation of modeling grammar rules.
- **ER Modeling Process:**
    - **Presentation Layer ER Model:** For user communication; surface-level expression.
    - **Design-Specific ER Model:** For database design; more technical detail.
        - **Coarse Granularity:** Adds (min, max) notation, deletion rules (Restrict, Cascade, Set Null, Set Default).
        - **Fine Granularity:** Decomposes non-mappable constructs (M:N, multi-valued attributes), adds attribute types/sizes. Ready for logical mapping.
- **EER Modeling (Enhanced Entity-Relationship) [PYQ Q3b, Q4a]:** Extends ER with constructs for more complex applications.
    - **Superclass/Subclass (SC/sc) Relationship (Is-A Relationship):**
        - **Superclass (SC):** Generic entity type.
        - **Subclass (sc):** Specialized entity type inheriting attributes/relationships from SC (Type Inheritance). Can have own specific attributes/relationships.
        - Cardinality always 1:1 between SC instance and corresponding sc instance. Participation of sc in SC/sc is total.
    - **Specialization and Generalization [PYQ Q3b]:** Two perspectives of SC/sc relationship.
        - **Specialization:** Top-down; defining subgroups of an SC.
        - **Generalization:** Bottom-up; identifying common features of entity types to form an SC.
        - **Constraints:**
            - **Disjointness:**
                - **Disjoint (d):** SC instance can be member of at most one sc.
                - **Overlapping (o):** SC instance can be member of more than one sc.
            - **Completeness (Totalness):**
                - **Total (double line SC to circle):** Every SC instance must be a member of some sc.
                - **Partial (single line SC to circle):** SC instance may not belong to any sc.
            - **Predicate-defined (condition-defined) subclass:** Membership by condition on SC attribute.
            - **User-defined subclass:** Membership explicitly specified.
    - **Hierarchy and Lattice [PYQ Q3b]:**

        - **Specialization Hierarchy:** Subclass participates in only one SC/sc relationship (tree structure). Inherits from direct parent/ancestors.
        - **Specialization Lattice (Multiple Inheritance):** Subclass (shared subclass) in more than one SC/sc relationship. Inherits from all its superclasses.
    - **Categorization [PYQ Q3b]:** Subclass (category) is subset of the UNION of two or more superclasses OF DIFFERENT ENTITY TYPES. Represents single SC/sc relationship. Selective inheritance. (Indicated by 'U' in circle).
- **Modeling Complex Relationships (Ch 5) [PYQ Q3b]:**
    - **Ternary Relationship Type:** Involves three entity types. (min, max) crucial. Can be conceptualized as base entity.
    - **Beyond Ternary (Cluster Entity Type):** Grouping entity types/relationships into higher-level abstract entity (cluster). For layered ERDs.
    - **Weak Relationship Type (Inter-relationship Integrity Constraint):** Existence of instance in one relationship set depends on instance in another.
        - **Inclusion Dependency (Subset):** e.g., `Manages ⊆ Works_in` (solid arrow from dependent).
        - **Exclusion Dependency (Mutual Exclusion):** Equivalent to exclusive arc (dotted line between weak rels).
    - **Composites of Weak Relationship Types:** Combining weak rels for richer semantics.
- **Design Issues in ER & EER Modeling [PYQ Q3b]:** Choosing entity vs. attribute; binary vs. higher-degree; use of weak entities; when to use EER constructs; handling M:N/multi-valued attributes (decomposition).
- **Validation of Conceptual Design (Connection Traps):**
    - **Fan Trap:** Ambiguity when two or more 1:N relationships fan out from same entity.
    - **Chasm Trap:** Pathway seems to exist, but missing info (optional participation) prevents joining related entities.
    - Resolving traps may involve restructuring ERD.

UNIT-II: LOGICAL DATA MODELLING

1. **Overview of Relational Data Model**
    - **Definition and Terminology:**
        - **Relation:** Two-dimensional table of data.
        - **Attribute (Column/Field):** Named column of a relation.
        - **Domain:** Set of allowable (atomic) values for one or more attributes.
        - **Tuple (Row/Record):** A row of a relation.

        - **Relation Schema:** Name of relation and its attributes (e.g., `PLANT(Pl_name, Pl_p#)`). Heading/intension.
        - **Relation State (Instance):** Set of tuples for a schema at a specific time. Body/extension.
        - **Degree:** Number of attributes in a relation.
        - **Cardinality:** Number of tuples in a relation.
        - **Characteristics of a Relation:** Order of tuples immaterial; order of attributes immaterial (by name); attribute values atomic (1NF); each tuple distinct.
    - **Integrity Constraints [PYQ - Key concepts are important]:**
        - **Key Constraints [PYQ Q1b]:**
            - **Superkey:** Attribute(s) uniquely identifying a tuple.
            - **Candidate Key:** Minimal superkey.
            - **Primary Key:** Candidate key chosen to uniquely identify tuples (underlined).
            - **Alternate Key:** Candidate keys not chosen as primary.
        - **Entity Integrity Constraint:** No primary key value can be null.
        - **Referential Integrity Constraint [PYQ Q1b]:** Foreign Key (FK) values must match candidate key values of referenced relation or be wholly null.
            - **Foreign Key (FK):** Attribute(s) in one relation whose values match a candidate key in another (or same) relation.
            - **Actions on violation:** Restrict, Cascade, Set Null, Set Default.
2. **Mapping ER Model to a Logical Schema [PYQ Q1d, Q4a]:**
    - **Mapping Entity Types (Base and Weak):**
        - **Strong Entity:** Create relation; choose PK. Include simple/atomic components of composite/stored attributes.
        - **Weak Entity:** Create relation. PK = PK of owner (as FK) + partial key of weak entity.
    - **Mapping Relationship Types:**
        - **1:N: Foreign Key Approach (Standard):** PK of '1-side' (parent) in 'N-side' (child) relation as FK. Attributes of rel on N-side. **Cross-Referencing Design:** New relation for relationship (PK from N-side or 1-side if N-side participation optional). Use if child participation optional to avoid nulls.
        - **M:N:** Create new relation (junction/associative table). PK = combination of PKs of participating entities (as FKs). Relationship attributes become attributes of this new relation.
        - **1:1: Option 1 (FK):** Choose one relation (e.g., if one total participation, make it child), add PK of other as FK (must be unique). **Option 2 (Merged Relation):** If both total participation, merge into one relation. **Option 3 (Relationship Relation/Cross-referencing):** If both partial, create separate relation.

        - **Recursive Relationship: 1:N Recursive:** Add PK of entity as FK in same relation (different role name). **M:N Recursive:** New relation with two FKs, both referencing PK of original entity (different role names).
    - **Mapping Multi-valued Attributes:** Create new relation. PK = PK of original entity (as FK) + multi-valued attribute itself.
    - **Mapping N-ary Relationships (Higher Degree) [PYQ Q5b]:** Create new relation. PK = combination of PKs of all participating entities (as FKs).
    - **Information-Preserving Mapping:** More detailed logical schema grammar to retain metadata (alternate keys, participation (min,max), deletion rules).
3. **Mapping EER Model to a Logical Schema [PYQ Q4a]:**
    - **Mapping Specialization/Generalization:**
        - **Option 1 (Multiple Relations - SC and sc's):** Relation for SC, relation for each sc. PK of sc = PK of SC (as FK to SC). Good for disjoint subclasses, specific attributes/rels for sc's.
        - **Option 2 (Single Relation - SC only):** One relation for SC. Include attributes of all sc's (use nulls). Add type attribute. Good for overlapping sc's or few specific sc attributes.
        - **Option 3 (Multiple Relations - sc's only):** Relation for each sc. Include inherited SC attributes. Only if SC total & disjoint. May lead to redundancy.
    - **Mapping Specialization Hierarchy:** Apply chosen SC/sc mapping option recursively.
    - **Mapping Specialization Lattice (Shared Subclass):** Shared subclass relation has multiple FKs, one for each SC/sc path it participates in, referencing respective superclasses.
    - **Mapping Categorization:** Create relation for category (subclass) (usually surrogate PK). Create relations for superclasses. Category PK included as FK in each superclass relation.
    - **Mapping Aggregation [PYQ Q5b]:** "Whole" (aggregate) relation includes PK of "part" superclasses as FKs.
    - **Information Loss in traditional EER mapping:** Type of relationship, disjointness, multiple specializations may be lost.
    - **Information-Preserving Grammar for EER:** Extends logical schema grammar to capture EER metadata.
4. **Mapping of Higher Degree Relationships [PYQ Q5b]:** (Covered in ER Mapping) Decompose into gerund (associative) entity conceptually. Map gerund & its binary rels. Gerund relation PK = combination of PKs of all original participants.
5. **Mapping of Aggregation [PYQ Q5b]:** (Covered in EER Mapping) Aggregate (subclass as "whole") mapped to relation. PKs of superclasses ("parts") included as FKs in aggregate relation.
6. **Mapping Complex ER Model Constructs to a Logical Schema:**
    - **Cluster Entities:** Decompose cluster conceptually into weak/gerund entity related to entities outside cluster before mapping.
    - **Weak Relationships:**

- **Inclusion Dependency:** Handle by FK placement, app checks/triggers if not M:N. If `TEACHING ⊆ CAN_TEACH`, map as specialization (TEACHING as sc of CAN_TEACH gerund).
- **Exclusion Dependency:** Typically app logic/triggers; direct relational constraints limited.

7. **Normalization [PYQ Q5a]:** Process of organizing data to reduce redundancy & improve data integrity by decomposing relations.
   - **Anomalies (without normalization):** Insertion, Deletion, Modification/Update.
   - **Normal Forms:**
     - **1NF (First Normal Form):** All attribute values atomic. No repeating groups or multi-valued attributes in a single cell. (Relation by definition is in 1NF).
     - **2NF (Second Normal Form):** 1NF AND all non-key attributes fully functionally dependent on entire primary key. No partial dependencies.
     - **3NF (Third Normal Form):** 2NF AND no transitive dependencies (non-key attribute depends on another non-key, which depends on PK).
     - **BCNF (Boyce-Codd Normal Form):** Stricter 3NF. Every determinant (attribute set X in X->Y) must be a superkey (or candidate key).
     - **4NF (Fourth Normal Form):** BCNF AND no non-trivial multi-valued dependencies (MVDs) unless determinant is superkey. (MVD X ->> Y means Y values determined by X independently of Z).
     - **5NF (Fifth Normal Form / Project-Join Normal Form - PJ/NF):** 4NF AND no join dependencies (JDs) not implied by candidate keys. (JD means relation can be losslessly decomposed & reconstructed by joining).

UNIT-III: DATABASE IMPLEMENTATION AND PHYSICAL DATABASE DESIGN

1. **Database Creation using SQL [PYQ Q6a]**
   - **Data Definition Using SQL (SQL/DDL):** Standard language (DBMS variations). Relation=table, Attribute=column, Tuple=row. SQL tables: duplicate rows, nulls, column order matters. Major DDL: `CREATE`, `ALTER`, `DROP`.
   - **Base Table Specification in SQL/DDL:**
     - `CREATE TABLE table_name (col_def1, ..., [table_constraint1, ...]);` Defines new, physically stored base table.
     - Column Definition: `col_name data_type [DEFAULT val] [col_constraint_list];`
   - **Data Types:** Numeric (`NUMERIC(p,s), DECIMAL(p,s), INTEGER, SMALLINT, FLOAT(p), REAL, DOUBLE PRECISION`), String (`CHAR(n), VARCHAR(n)`), Bit String (`BIT(n), BIT VARYING(n)`), Date/Time (`DATE, TIME(p), TIMESTAMP(p), INTERVAL`).
   - **Constraints (Column-level or Table-level):**
     - `NOT NULL`: Column cannot have nulls.
     - `UNIQUE`: All values in column(s) unique (alternate keys).

        - `PRIMARY KEY`: Main table identifier (implies `NOT NULL`, `UNIQUE`).
        - `FOREIGN KEY ... REFERENCES ...`: Defines FK and referenced table/columns.
        - **Referential Triggered Actions (Deletion/Update Rules): [PYQ - Implied in DDL]**
            - `ON DELETE {CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION}`
            - `ON UPDATE {CASCADE | SET NULL | SET DEFAULT | RESTRICT | NO ACTION}`
        - `CHECK (condition)`: Condition true for every row.
    - **Naming Conventions:** Follow consistent naming.
    - **`ALTER TABLE` Statement:** Modifies existing table structure. Actions: `ADD [COLUMN] col_def`, `DROP [COLUMN] col_name {CASCADE|RESTRICT}`, `ALTER [COLUMN] col_name SET DEFAULT val | DROP DEFAULT`, `ADD table_constraint`, `DROP CONSTRAINT name {CASCADE|RESTRICT}`.
    - **`DROP TABLE` Statement:** Deletes table definition and all data. `DROP TABLE table_name {CASCADE | RESTRICT};` (`CASCADE`: drops dependent objects; `RESTRICT`: prevents drop if dependencies exist).
    - **Specification of User-Defined Domains:** `CREATE DOMAIN domain_name AS data_type [DEFAULT val] [domain_constraint_list];` Modular column definitions. `ALTER DOMAIN`, `DROP DOMAIN`.
    - **Schema and Catalog Concepts in SQL/DDL:**
        - **SQL-Schema:** Named collection of schema elements (tables, views, domains, constraints) under single user/authorization ID.
        - `CREATE SCHEMA schema_name [AUTHORIZATION user_name] [schema_element_list];`
        - `DROP SCHEMA schema_name {CASCADE | RESTRICT};`
        - **Catalog:** Named collection of SQL-schemas in an SQL environment.
        - **INFORMATION_SCHEMA:** Mandatory schema in SQL-92 systems; provides views on system catalog (metadata).
2. **SQL Commands – DML & Views**
    - **Data Population Using SQL (INSERT):**
        - `INSERT INTO table_name [(column_list)] VALUES (value_list);` (Single-row insert).
        - `INSERT INTO table_name [(column_list)] subquery;` (Multi-row insert).
    - **Data Deletion Using SQL (DELETE):** `DELETE FROM table_name [WHERE search_condition];` (Can trigger referential actions).
    - **Data Modification Using SQL (UPDATE):** `UPDATE table_name SET col1 = val1, ... [WHERE search_condition];`
    - **Advanced Data Manipulation using SQL (Ch 11)**
        - **Relational Algebra Concepts (Recap):** Select, Project, Union, Intersection, Difference, Cartesian Product, Join (Equi, Natural, Theta, Outer), Divide, Aggregate Functions.
        - **SQL Queries Based on a Single Table:** `SELECT [DISTINCT|ALL] cols FROM tbl [WHERE cond] [GROUP BY cols] [HAVING cond] [ORDER BY cols [ASC|DESC]];`

        - Selection (`WHERE`): Filters rows.
        - Projection (`SELECT cols`): Selects columns. `DISTINCT` removes duplicates.
        - Expressions in `SELECT`/`WHERE`.
        - Operators: `BETWEEN`, `IN`, `NOT IN`, `LIKE` (%, _, `ESCAPE`).
        - Handling Null Values (`IS NULL`, `IS NOT NULL`): Nulls in arithmetic/comparisons yield unknown. Aggregates (except `COUNT(*)`) ignore nulls.
        - Aggregate Functions (`COUNT, SUM, AVG, MAX, MIN`).
        - Grouping (`GROUP BY`), Filtering Groups (`HAVING`).
    - **SQL Queries Based on Binary Operators (Joins):**
        - Cartesian Product (`CROSS JOIN` or comma in `FROM` with no join condition).
        - Inner Joins (`[INNER] JOIN ... ON cond;`, `JOIN ... USING (common_cols);`, `NATURAL JOIN;`). Self Joins, N-way Joins.
        - Outer Joins (`LEFT|RIGHT|FULL [OUTER] JOIN`).
        - Set Theoretic Operators (`UNION [ALL]`, `INTERSECT`, `MINUS / EXCEPT`). Tables must be union-compatible.
    - **Subqueries (Nested Queries):**
        - `WHERE` clause: Single-row (uses =, <, >), Multi-row (uses `IN, NOT IN, ANY, ALL`).
        - Correlated Subqueries: Inner query depends on outer. Uses `EXISTS, NOT EXISTS`.
        - `FROM` clause (Inline Views / Derived Tables).
        - `SELECT` clause (Scalar Subqueries).
        - `HAVING` clause.
    - **Views:** Virtual table based on result-set of stored query. Does not store data itself.
        - `CREATE VIEW view_name [(column_list)] AS subquery [WITH [CASCADED | LOCAL] CHECK OPTION];`
        - **Purpose:** Simplify complex queries, security, customized data presentation, logical data independence.
        - **Updating Views:** Possible for simple views. `WITH CHECK OPTION` ensures DML on view doesn't cause rows to disappear.
3. **Database Programming [PYQ Q7a, Q7b]**
    - Writing application programs that interact with a database.
    - **Types/Approaches to Database Programming:**
        1. **Embedded SQL:** SQL statements in host language source code. Precompiler converts to host calls. Static SQL.

2. **API-based Database Access (e.g., JDBC, ODBC, ADO.NET): [IMP]** API (functions, classes, protocols) library for specific language. Dynamic SQL. Portable.
3. **Stored Procedures and Functions:** (see below).
4. **Object-Relational Mappers (ORMs):** Frameworks mapping OOP objects to relational tables. Abstracts SQL.
5. **Database-Specific Scripting Languages (PL/SQL, T-SQL):** For DB admin/dev within DBMS.

- **Embedded SQL & Dynamic SQL:**
  - **Embedded SQL:** Host variables prefixed with `:`, `EXEC SQL ...` prefix, `SQLCA`/`SQLSTATE`/`SQLCODE` for status, `WHENEVER` for error handling.
  - **Dynamic SQL:** SQL constructed/executed at runtime. `PREPARE` (parses/compiles), `EXECUTE` (executes).
- **Cursors: [PYQ Q7a]** Mechanism to process query results row-by-row in app program. Bridges set-SQL & row-oriented host languages (impedance mismatch).
  - **Declaration:** `DECLARE cursor_name [INSENSITIVE] [SCROLL] CURSOR FOR SELECT_statement [ORDER BY ...] [FOR READ ONLY | FOR UPDATE [OF column_list]];`
  - **Operations:** `OPEN cursor;`, `FETCH cursor INTO :host_vars;`, `UPDATE ... WHERE CURRENT OF cursor;`, `DELETE ... WHERE CURRENT OF cursor;`, `CLOSE cursor;`.
  - **Properties:** Scrollable, Insensitive (snapshot), Updatable.
  - **Need for Cursors:** Handle multi-row SQL results in row-oriented host languages.
  - **Types of Cursors (Properties/Behavior):**
    - **Read-Only:** Fetching only. `FOR READ ONLY`.
    - **Updatable:** Fetch, modify, delete. `FOR UPDATE [OF cols]`. Query must be simple.
    - **Scrollable:** Flexible movement (`NEXT, PRIOR, FIRST, LAST, ABSOLUTE n, RELATIVE n`). `SCROLL CURSOR`.
    - **Insensitive (Snapshot):** Operates on temp copy at open time. `INSENSITIVE CURSOR`.
    - **Sensitive:** Attempts to reflect underlying data changes.
    - **Keyset-Driven:** Keys fixed at open; non-key changes visible.
    - **Forward-Only (Non-Scrollable):** Default. Sequential fetch.
    - **Static:** Similar to insensitive; operates on snapshot.
    - **Dynamic:** Most "sensitive"; all committed changes visible.
- **Stored Procedures & Functions:** Precompiled SQL/procedural statements stored in DB.

- **Procedures:** `CREATE PROCEDURE proc_name ([param_list]) AS BEGIN...END;` (IN, OUT, INOUT params). Called by `CALL` or `EXEC`.
- **Functions:** `CREATE FUNCTION func_name ([param_list]) RETURNS data_type AS BEGIN...RETURN value; END;` Must return value. Used in SQL expressions.
- **Advantages:** Performance, reusability, security, modularity.
- **SQL/PSM (Persistent Stored Modules):** Standard for procedural extensions (vars, IF, LOOP, WHILE, cursors).
- **Exception Handling:** `DECLARE ex_name EXCEPTION; RAISE ex_name; EXCEPTION WHEN ex_name THEN ...; WHEN OTHERS THEN ...;`

- **Packages (in Oracle PL/SQL):** Schema objects grouping related PL/SQL types, vars, consts, subprograms, cursors, exceptions. Specification (public) + Body (public/private defs).
- **Triggers: [PYQ Q7b]** Procedural code auto-executed by DBMS on DML events (INSERT, UPDATE, DELETE) on a table.
  - **ECA Model (Event-Condition-Action):** Event (DML), Condition (Boolean, optional), Action (PL/SQL block).
  - `CREATE [OR REPLACE] TRIGGER name {BEFORE|AFTER} {INSERT|DELETE|UPDATE [OF cols]} ON table [FOR EACH ROW] [WHEN (condition)] BEGIN...END;`
  - **Row-level trigger (`FOR EACH ROW`):** Fires once per affected row. Access `:OLD`, `:NEW` values.
  - **Statement-level trigger (default):** Fires once per DML statement.
  - **Uses:** Complex integrity constraints, auditing, derived data, logging.
  - **More Types of Trigger:**
    - **INSTEAD OF Triggers:** For views (esp. non-updatable). Fires *instead* of DML on view; trigger code defines operations on base tables.
    - **DDL Triggers:** Fire on DDL events (`CREATE TABLE`, etc.). DBMS-specific.
    - **Logon/Logoff Triggers (System-Level):** Fire on user session connect/disconnect. DBMS-specific.
    - **Database Event Triggers (Server-Level):** Fire on DB events (startup, shutdown, errors). DBMS-specific.
    - **Compound Triggers (Oracle specific):** Defines actions for multiple timing points (BEFORE STMT, BEFORE ROW, AFTER ROW, AFTER STMT) for one DML event in single trigger.

UNIT-IV: DATABASE TUNING, MAINTENANCE, AND SECURITY

1. **Database Tuning and Maintenance**
   - **Introduction to Database Tuning: [PYQ Q9c]** Process of optimizing DB performance for user requirements/SLAs. Involves physical/conceptual design, queries, app code, DBMS params,

hardware. Iterative: Monitor -> Identify Bottlenecks -> Diagnose -> Implement -> Measure.

- **Workload Analysis:** Types/frequencies of queries/updates, performance goals, accessed relations/attributes, selection/join conditions, selectivity.

- **Clustering and Indexing [PYQ Q8a, Q9a]**
  - **Indexing:** Separate data structure (on disk) with index key values & pointers to actual data rows, allowing fast searching and avoiding full table scans.
  - **Guidelines for Index Selection:**
    1. **Whether to Index:** Attributes in `WHERE` (selections/joins), `ORDER BY`, `GROUP BY`. Only if query benefits. Benefit multiple queries.
    2. **Choice of Search Key (Single Attribute):** Exact Match (`col = val`): Hash or B+-tree. Range Conditions (`col > val`, `BETWEEN`): B+-tree essential.
    3. **Multi-Attribute (Composite):** For combined conditions. Column order critical. Most selective/equality-used attribute first.
    4. **Whether to Cluster:** At most one clustered index/table (physical order of data rows = index order). Beneficial for range queries, PK joins.
    5. **Hash vs. Tree (Revisited):** B+-tree (default): range/equality. Hash: best for exact equality, point lookups, inner join table.
    6. **Balancing Cost vs. Benefit:** Indexes speed `SELECT`, slow DML. Be selective for high-write tables. More indexes for read-heavy.
  - **Types of Indexing (Index Structures and Properties):**
    - **Primary Index:** Search key specifies sequential file order (ordering key field). Often on PK, then called **Clustered Index**. Max one/file.
    - **Clustered Index:** Physical data row order = index key value order. Max one/table. Efficient for range queries on clustering key.
    - **Secondary Index (Non-Clustered):** Index order different from physical row order. Multiple/table. Entries point to data rows (rowids/clustered key).
    - **B+-Tree Index:** Most common. Balanced tree, leaf nodes linked. Supports equality/range. Clustered/non-clustered.
    - **Hash Index:** Hash function computes bucket/page address. Fast equality. Not for range. Collisions.
    - **Unique Index:** DBMS enforces no two rows have same indexed value(s). PKs always unique.

        - **Composite Index (Multi-column):** On >=2 columns. Order important.
        - **Covering Index:** Non-clustered index containing all columns for a query (in `SELECT` & `WHERE`). Allows index-only scan.
        - **Bitmap Index:** For low cardinality columns (few distinct values). Bitmap per distinct value. Efficient for complex AND/OR.
        - **Function-Based Index (Expression Index):** On result of function/expression (e.g., `UPPER(LastName)`).
        - **Spatial Index (e.g., R-tree, Quad-tree):** For spatial data types.
        - **Full-Text Index:** For text data (`VARCHAR(MAX)`, `CLOB`) for word/phrase search.
        - **Index-Only Plans:** Query answered solely from index, no table access.
    - **Clustering: [PYQ]** Storing related data physically close together on disk to minimize I/O.
        - **How it Works:** Organizing rows by values of clustering key (often via clustered index).
        - **Types of Clustering / Implementations:**
            - **Clustered Index (Intra-Table Clustering):** Physical storage order = logical clustered index key order. (As above).
            - **Co-clustering (Inter-Table / Multi-Table Clustering):** Physically interleaving rows from >=2 related tables by join key (PK-FK). Speeds joins.
            - **Hash Clustering:** Rows placed in physical buckets by hash on clustering key. Fast equality lookups.
        - **Benefits of Clustering:** Reduced Disk I/O, improved query performance (range, joins).
        - **Drawbacks of Clustering:** DML overhead (page splits), only one clustered index/table, slower full scans if not dense.
        - **Tools to Assist in Index Selection:** Database Tuning Advisors/Wizards (analyze workload, suggest indexes, "what-if" analysis).
    - **Guidelines for Index Selection (reiteration/summary):** Index PKs, FKs, `WHERE` attributes (high selectivity), `ORDER BY`/`GROUP BY` attributes. Consider multi-column. Avoid small tables, frequently updated large-value cols, very low selectivity cols. Review/drop unused.
- **De-normalization:[PYQ]** Intentionally adding controlled redundancy to a normalized schema to boost specific query performance by reducing joins.
    - **Rationale/Use Case:** Highly normalized schemas -> many joins -> slow queries. Denormalization pre-joins/duplicates data for critical queries.
    - **Trade-off:** Benefit: Faster read/query. Cost/Drawback: Increased storage, risk of anomalies/inconsistencies, complex DML/app logic.
    - **Guideline:** Apply selectively *after* normalization, only if indexing/query tuning insufficient for critical performance.
    - **Types of Denormalization / Techniques:**
        - **Pre-joining Tables:** Add attributes from "one-side" to "many-side" table.

        - **Storing Derived/Calculated Values:** Pre-compute totals, averages.
        - **Combining Tables:** Merging tables with 1:1 or tight, frequently accessed 1:N relationship.
        - **Repeating Groups (Limited Use):** Multiple columns for fixed similar attributes (inflexible, discouraged).
        - **Creating Reporting Tables/Data Marts:** Separate, denormalized tables for analytical queries, populated from operational DB.
    - **Database Tuning (Conceptual Schema, Queries, Views): [PYQ]**
        - **Tuning Conceptual Schema:** Settling for weaker normal form (e.g., 3NF vs BCNF if BCNF critical query impact), Denormalization, Vertical partitioning (splitting columns), Horizontal partitioning (splitting rows).
        - **Tuning Queries and Views:** Rewriting SQL (avoid unnecessary joins, use sargable predicates), optimizing view definitions, influencing query optimizer (hints, stats).
2. **Database Security [PYQ Q8b]**
    - **Introduction to Database Security:** Protecting database against unauthorized access, modification, destruction.
        - **Objectives:**
            - **Secrecy/Confidentiality:** Preventing unauthorized disclosure.
            - **Integrity:** Ensuring data accuracy/consistency, preventing unauthorized modification.
            - **Availability:** Ensuring authorized users can access data when needed.
        - **[PYQ Q8b]** Security should be integrated throughout database modeling/design, not an afterthought.
    - **Levels of Security Implementation:**
        - **1. Conceptual Level (ER/EER Modeling):** Identify Sensitive Data, Define Access Privileges Conceptually (roles, data needs), Consider Views for Abstraction, Data Ownership.
        - **2. Logical Level (Relational Schema Design):** View Design for Security (expose only necessary columns/rows), Granular Privileges planning (`SELECT, INSERT, ...`), Separation of Duties.
        - **3. Physical Level (Implementation in DBMS):**
            - **GRANT/REVOKE (Discretionary Access Control - DAC):** Implement privileges on tables, views, procs.
            - **Role-Based Access Control (RBAC):** Create roles, grant privileges to roles, grant roles to users.
            - **Stored Procedures for Controlled Access:** Encapsulate DML; grant `EXECUTE` on proc, not direct table access.
            - **Triggers for Auditing and Complex Constraints.**

        - **Mandatory Access Control (MAC) (if supported/required):** Assign security labels (data) & clearance levels (users) for high-security. E.g., Bell-LaPadula.
        - **Encryption:** Data at rest (DB files) & in transit (network). Column-level, Transparent Data Encryption (TDE).
        - **Auditing:** Configure DBMS auditing for DB activities (esp. sensitive data/DDL).
        - **Authentication:** Strong user identity verification (passwords, MFA).
        - **Network Security:** Secure communication channels (SSL/TLS).
    - **Access Control:** Mechanisms to control who accesses what data & performs what operations.
        - **Discretionary Access Control (DAC): [PYQ - Implied in GRANT/REVOKE]** Based on privileges (access rights) granted to users/roles by object owners/DBAs.
            - Objects: Tables, views, columns, etc. Privileges: `SELECT, INSERT, UPDATE, DELETE, REFERENCES, USAGE`, etc.
            - `GRANT privilege_list ON object_name TO user_list [WITH GRANT OPTION];` (`WITH GRANT OPTION`: allows grantee to further grant).
            - `REVOKE [GRANT OPTION FOR] privilege_list ON object_name FROM user_list [CASCADE | RESTRICT];` (`CASCADE`: revokes from subsequent grantees; `RESTRICT`: fails if privilege passed on).
            - **Roles:** Named group of privileges. `CREATE ROLE, DROP ROLE, GRANT role TO user`.
        - **Mandatory Access Control (MAC):** Based on system-wide policies, not owner's discretion. Uses security classifications (labels) for data objects & clearance levels for subjects (users/processes).
            - **Bell-LaPadula Model:**
                - Simple Security Property (No Read Up): `class(Subject) >= class(Object)`.
                - *-Property (Star Property - No Write Down): `class(Subject) <= class(Object)`.
            - **Multilevel Relations and Polyinstantiation:** Storing data with different security levels in same table; users may see different versions of a row.
            - **Covert Channels:** Indirect ways of inferring higher-level information.
    - **DCL Commands (GRANT, REVOKE):** Covered under DAC.
    - **Views as Security Mechanism:** Restrict users to specific rows (`WHERE` in view) and columns (`SELECT` list in view) of underlying tables. Grant privileges on view, not base tables.
