GTU Prep Zone

GTU DBMS Summer 2025 Solution

GTU - DBMS Summer 2025 PAPER SOLUTION

Subject Code: 3130703 | **Subject Name:** DBMS | **Total Marks:** 70

Q.1(a) Explain Logical & Physical data independence supported by DBMS (03M)

---

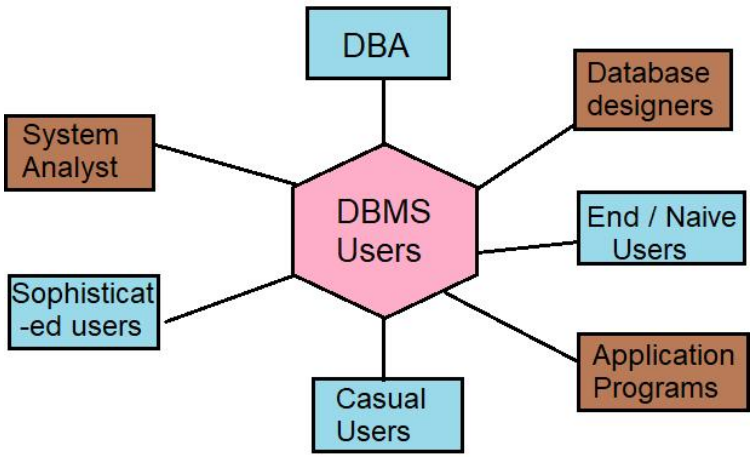
1. **Logical Data Independence:** The ability to change the Conceptual Schema (overall logical structure) without changing the External Schemas (user views) or application programs.

- Allows addition, deletion, or modification of entities, attributes, or relationships in the conceptual schema without disrupting existing user applications.

2. **Physical Data Independence**: The ability to change the Internal Schema (physical storage structure) without changing the Conceptual Schema or external views.

- Allows changes like migrating to a different file organization (e.g., hash to B-tree), using new indexing techniques, or moving data to different disks. Both types of independence reduce the impact of changes, making the DBMS more flexible and easier to maintain.

Q.1(b) Explain different types of users of DBMS. (04M)



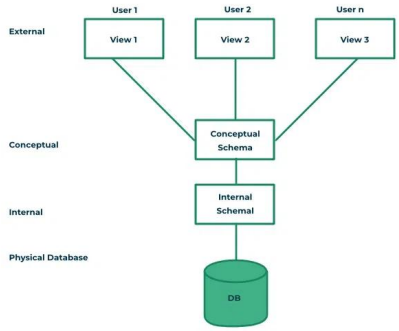
- **Database Administrators (DBA):** Responsible for managing the entire DBMS, including schema definition, security, authorization, monitoring performance, and handling backup/recovery.
- **Database Designers:** Define the database structure, including the conceptual and logical schema, identifying entities, attributes, and relationships.
- **Naive/Parametric End Users:** Non-technical users who interact with the database through a simple, custom-built application interface (e.g., bank tellers, ticket booking agents).
- **Casual End Users:** Users who occasionally access the database, typically using a high-level query language (like SQL) to obtain new information each time (e.g., middle-level managers).
- **Sophisticated End Users:** Users familiar with the database structure, who use the DBMS tools for data analysis, mining, and report generation (e.g., business analysts, scientists).
- **System Analysts:** Determine the requirements of end-users and develop specifications for transactions.
- **Application Programmers:** Implement the transactions and application programs defined by the system analysts, often in procedural languages embedded with SQL.

Q1(c) Compare database approach with traditional file systems to store application data.(07M)

Feature	Traditional File System	Database Approach (DBMS)
Data Redundancy	**High** (same data stored in multiple files).	**Controlled** (data centralized, minimal duplication).
Data Inconsistency	**High** (due to uncontrolled redundancy).	**Low** (consistency enforced centrally).
Data Sharing	**Difficult** (data is isolated).	**Easy** (centralized management allows concurrent access).
Security	**Poor** (security is managed by applications).	**Strong** (security, access control, and authorization are built-in).
Integrity	Must be manually coded	**Centralized** and enforced by the

Feature	Traditional File System	Database Approach (DBMS)
Constraints	into **each** application program.	DBMS (e.g., Primary Key).
Program-Data Dependency	**High** (schema changes require program modification).	**Low** (achieved through **Data Independence**).
Backup and Recovery	**Primitive** (relies on OS utilities, difficult to ensure completeness).	**Automatic** and robust (uses logging and checkpointing).

Q.2(a) Explain Three Layer Schema Architecture of DBMS. (03 M)



- **External Level (View Level):** Describes a specific part of the database relevant to a particular user or group; users see data defined by their view.
- **Conceptual Level (Logical Level):** Describes the entire database structure for the community of users, defining all entities, attributes, relationships, and constraints.
- **Internal Level (Physical Level):** Describes the physical storage structure of the database, including file organization, indexing, and internal schema details.

Q.2(b) Explain Specialization, Generalization and Categorization in EER Modeling. (04 M)

---

1. **Specialization (Top-Down):** The process of defining a set of subclass from a higher-level entity type. It is used to define distinguishing characteristics.

Example: Employee entity specializing into Pilot, Clerk, and Engineer.

2. **Generalization (Bottom-Up):** The process of defining a higher-level entity from two or more lower-level entity types that share common features.

Example: Car and Truck generalizing into Vehicle.

3. **Categorization (Union Type):** Used to define a subclass that is a union of two or more entity types that may not have the same super class. It is used when a single entity type needs to relate to entities from multiple, distinct super classes.

Example: An entity Property_Owner (Category) could be the union of Person and Company entities.

4. **Relationship Type:** Specialization/Generalization represents an "IS-A" relationship, while categorization represents a "UNION" or "IS-A-MEMBER-OF" relationship.

Q2(c) Explain Following Constraints supported by RDBMS: 1. Primary Key 2. Foreign Key / Referential Integrity Constraints 3. Entity Integrity 4. Domain Constraint (07 M)

- Integrity constraint are rules in DBMS that help keep data in database accurate, consistent and reliable.
- They act like set of guidelines that ensure all information stored in database follows specific standards.

1. **Primary Key Constraint:** A minimal set of attributes that uniquely identifies each tuple (row) in a relation. Its value must be unique for every record.

- This constraint is specified on database scema to the primary key attribute to ensure that no two tuples are same.
- Each entry must be uniue and not null.
- Eg: roll n0, email id etc..

2. **Entity Integrity Constraint:** Entity integrity constraint state that primary primary key can never contains null value.

- Key features: uniqueness, not null

Ex: Student_id, phone_number etc..

3. **Domain Constraint:** Domain constraint are type of constraint that ensure the value stored in column of database are valid and within specific range or domain.

- In simple terms they define what type of data is allowed in column and restrict invalid data entry.
- Includes data type like string, char, time, integer, date etc..

4. **Foreign Key (FK) / Referential Integrity:** A set of attributes in a referencing relation that refers to the Primary Key of a referenced relation.

- Referential integrity constraints are rules that-ensure relationship between table remain consistent.

OR

Q2(c) Explain Relational Algebra Operations in detail. (07 M)

Relational Algebra is a procedural query language operating on relations (tables).

1. Select (σ) Operation (Unary): Selects a subset of **tuples** (rows) from a relation that satisfies a specified boolean condition.

Example: σSalary>50000 (Employee).

2. Project (π) Operation (Unary): Selects a subset of attributes (columns) from a relation, eliminating any duplicate resulting tuples.

Example: πName,Department (Employee).

3. Union (∪) Operation (Binary): Includes all tuples that are in relation R, relation S, or both. R and S must be union-compatible.

4. Set Difference (−) Operation (Binary): Includes all tuples that are in R but not in S (must be union-compatible).

5. Cartesian Product (×) Operation (Binary): Combines every tuple from relation R with every tuple from relation S.

Example: Employee×Department.

6. Rename (ρ) Operation (Unary): Used to rename the result relation or attributes in the result.

7. Join (⋈) Operation (Derived): Used to combine related tuples from two relations based on a common join condition.

8. Intersection (∩) Operation (Derived): Includes all tuples common to both relations R and S (must be union-compatible).

9. Division (÷) Operation (Derived): Finds the tuples in one relation that are associated with every tuple in another relation (e.g., finding students who have taken all courses).

Q.3(a) Explain Recursive Relationship in ER Modeling with example. (03M)

---

1. **Definition**: A relationship type that exists between entities of the same entity type. It is also called a self-referencing or unary relationship.

2. **Role Names:** Role names are essential to distinguish how the entity participates in the relationship.

3. **Example (Employee)**: The relationship MANAGES on the Employee entity set.

4. **Roles**: One Employee entity instance takes the role of the Manager, and another Employee instance takes the role of the Worker.

5. **Cardinality**: The cardinality can be 1:1, 1:N, or M:N (e.g., a one-to-many relationship where one Manager manages many Workers).

Q3(b) Explain Cardinality Ratio and Participation constraint of ER Model. (04 M)

---

1. Cardinality Ratio: Specifies the maximum number of entity instances from one entity set that can be associated with an entity instance in the other entity set through a relationship.

2. Types (Binary): 1:1 (One-to-One), 1:N (One-to-Many), N:1 (Many-to-One), and M:N (Many-to-Many).

3. Participation Constraint: Specifies the minimum number of times an entity instance in an entity set must participate in a relationship.

4. Types:

i. **Total Participation (Mandatory):** Every entity must participate in the relationship (e.g., every employee must belong to a department). Represented by a **double line**.
ii. **Partial Participation (Optional):** An entity may or may not participate in the relationship (e.g., a department may or may not have a manager yet). Represented by a single line.

5. **Structural Constraints:** Together, the cardinality ratio and participation constraints are known as the structural constraints of a relationship.

Q3(c) What is the need to normalize data? Explain 1NF, 2NF & 3NF in detail. (07M)

Need for Normalization

1. Reduce Data Redundancy: By breaking large tables into smaller ones, normalization ensures data is stored only once, saving storage space.

2. Eliminate Anomalies: It removes insertion, deletion, and update anomalies that arise from storing related data redundantly.

3. Ensure Data Consistency: It guarantees that relationship dependencies are logically enforced, improving the integrity and reliability of the data.

Normal Forms

4. First Normal Form (1NF): A relation is in 1NF if and only if every attribute in that relation is atomic (indivisible) and there are no repeating groups of attributes.

5. Second Normal Form (2NF): A relation is in 2NF if it is in 1NF AND no non-prime attribute (non-key) is partially dependent on a candidate key.

Partial Dependency: A non-key attribute depends only on a proper subset of the composite candidate key.

6. Third Normal Form (3NF): A relation is in 3NF if it is in 2NF AND there is no transitive dependency of a non-prime attribute on a candidate key.

Transitive Dependency: A non-key attribute depends on another non-key attribute, which in turn depends on the primary key (PK→A and A→B).

6. **Goal:** 3NF is considered a sufficient level of normalization for most commercial database applications.

OR

Q3(a) Explain ACID Properties of transaction with appropriate example. (03 M)

1. Atomicity: The transaction must be treated as a single, indivisible unit. It must either execute completely (Commit) or have no effect at all (Rollback).

Example: Money transfer must either debit the sender and credit the receiver, or neither.

2. Consistency: The transaction must take the database from one valid state to another valid state. It must not violate any defined integrity constraints.

3. Isolation: Concurrent transactions must execute independently. The intermediate results of one transaction are hidden from other transactions until it is committed.

4. Durability: Once a transaction is successfully Committed, its changes to the database are permanent and must survive any subsequent system failures (e.g., power loss).

5. Purpose: These properties guarantee that database transactions are processed reliably.

Q3(b) Consider a relation R(A,B,C,D,E) with following dependencies: AB → C, CD → E, DE → B. Is ABD a candidate key of this relation? (04 M)

---

1. Given FDs: F={AB→C,CD→E,DE→B}

2. Check Super key (Find Closure of ABD):

- Start: (ABD)+={A,B,D}
- Apply AB→C: {A,B,C,D}
- Apply CD→E: {A,B,C,D,E}

3. Conclusion (Super key): Since (ABD)+ contains all attributes of R, ABD is a Super key.

4. Check Minimality (Candidate Key): Check closures of all proper subsets:

- (AB)+={A,B,C}. (Missing D, E) → Not a key.
- (AD)+={A,D}. (Missing B, C, E) → Not a key.
- (BD)+={B,D}. (Missing A, C, E) → Not a key.

5. **Final Conclusion:** Since ABD is a super key and no proper subset is a superkey, ABD is a Candidate Key of the relation R.

Q3(c) Explain Inference Rules for Functional Dependency. (07M)

The following are **Armstrong's Axioms**, which are sound and complete for generating the closure of a set of functional dependencies (F+).

1. Reflexivity Rule (Axiom): If B is a subset of A ($B \subseteq A$), then A→B holds.

✧ *Meaning:* A set of attributes always determines its own attributes.

2. Augmentation Rule (Axiom): If A→B holds, then AC→BC holds for any set of attributes C.

✧ *Meaning:* Adding the same attributes to both sides of a dependency preserves the dependency.

3. Transitivity Rule (Axiom): If A→B and B→C hold, then A→C also holds.

✧ *Meaning:* Dependencies can be chained together.

Derived Rules

4. Decomposition (Projectivity): If A→BC holds, then A→B and A→C hold.

5. Union (Additive): If A→B and A→C hold, then A→BC holds.

6. Pseudotransitivity: If A→B and CB→D hold, then AC→D holds.

7. Closure of F (F+): The set of all functional dependencies that can be logically inferred from the given set F using these inference rules.

8. Application: These rules are used by normalization algorithms and for checking the equivalence of different sets of FDs.

Q.4(a) What is the use of system log? What are the typical kinds of records in a system log? What are transaction commit points, and why are they important? (03 M)

---

1. Use of System Log: The log is a sequential file that records all update activities in the database. It is the core component for recovery from failures (undo/redo operations).

2. Log Record Types:

✧ **Start/End Records:** <start T>, <commit T>, and <abort T>.
✧ **Write Records:** <T, X, V_old, V_new>: Transaction T changed data item X from V_old (old value) to V_new (new value).

3. Transaction Commit Point: The point at which the <commit T> record is written to stable storage (disk).

4. Importance: It guarantees the Durability property of the transaction. Once the commit record is on disk, the DBMS is responsible for ensuring the changes are never lost, even if the system crashes immediately after.

Q.4(b) Explain SQL Injection in brief. (04 M)

1. Definition: A code injection vulnerability that occurs when a web application takes un sanitized user input and passes it directly to the database as part of an SQL query.

2. Attack Goal: The attacker inserts malicious SQL code into the input field to alter the intended query's logic, often to bypass authentication or extract data.

3. Example Bypass: If the query is SELECT * FROM Users WHERE Username = 'user_input', an attacker inputs ' OR '1'='1' --.

- ✧ The final query becomes: SELECT * FROM Users WHERE Username = '' OR '1'='1' --', where the OR '1'='1' condition is always true, granting unauthorized access.

4. Consequences: Can lead to data theft, modification, unauthorized access, or complete database compromise.

5. Prevention: The primary defense is using Parameterized Queries (Prepared Statements), which treat user input strictly as data, not as executable SQL code.

Q4(c) Explain Query Optimization with example. (07 M)

1. Definition: The process of choosing the most efficient execution plan for a SQL query from the many mathematically equivalent plans.

2. Goal: To minimize the **estimated cost** of the query execution, primarily measured by the number of disk I/O operations.

3. Steps:

- ✧ **Query Parsing:** Translates the SQL query into an initial Relational Algebra expression.
- ✧ **Optimization:** The query optimizer generates and evaluates various execution strategies (plans).
- ✧ **Execution:** The plan with the lowest estimated cost is selected and executed.

4. Heuristic Optimization: Uses rules (e.g., performing Select and Project operations as early as possible) to simplify and restructure the query before cost estimation.

5. Cost-Based Optimization: Uses the system catalog (statistics like relation sizes, attribute value distributions, and index availability) to calculate the estimated cost for each execution plan.

6. Example (Rule Application): The optimizer will prefer Joining two small tables over joining two large tables, and will apply selection conditions (filters) as early as possible to reduce the size of intermediate results.

OR

Q4(a) Explain the use of Btrees. (03 M)

---

1. Primary Use: B-trees (and B+ trees) are the standard data structures used for database indexing in virtually all modern RDBMS.

2. Disk Efficiency: They are designed to work well with block-oriented storage (hard drives), minimizing the number of expensive disk I/O operations required to find any record.

3. Structure: They are self-balancing search trees where all leaf nodes are at the same level, ensuring a guaranteed worst-case search time of O(log b N), where b is the block size.

4. Function: They enable fast searching for records based on a key value (equality and range queries), as well as efficient insertion and deletion operations.

Q4(b) Explain Cursor in PL/SQL with example. (04 M)

---

1. Definition: A control structure or a temporary work area in system memory used by the DBMS to process the result set (multiple rows) returned by a SQL query.

2. Purpose: It allows procedural extensions (like PL/SQL) to handle a set of rows one row at a time (row-by-row processing), as opposed to standard SQL which operates on sets.

3. Explicit Cursors: Manually declared by the programmer for queries that return multiple **rows**.

4. Four Steps of Cursor Processing:

- ✧ **DECLARE:** Define the cursor by associating it with a SELECT statement.
- ✧ **OPEN:** Execute the query and load the result set into the cursor's work area.
- ✧ **FETCH:** Retrieve the data of one row from the cursor into program variables.
- ✧ **CLOSE:** Release the work area allocated for the cursor.

6. **Implicit Cursors:** Automatically created by the DBMS for all DML statements (INSERT, UPDATE, DELETE) and single-row SELECT statements.

```
DECLARE
   message  varchar2(20):=
'Hello, World!';
BEGIN

dbms_output.put_line(message);
END;
/
```

Q4(c) Explain Conflict Serializability with precedence graph in Transaction Processing. (07 M)

---

1. Conflict Operations: Two operations in different transactions conflict if they access the same data item and at least one of them is a write operation (W-R, R-W, W-W).

2. Conflict Serializability: A schedule S is conflict serializable if it is conflict equivalent to some serial schedule (one where transactions execute completely one after the other). This property ensures that the concurrent execution is correct.

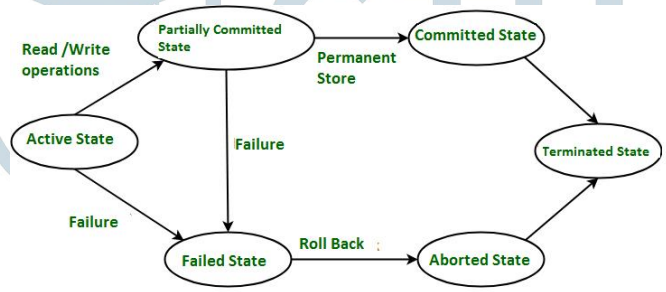
3. Precedence Graph (Dependency Graph): A graph used to test for conflict serializability.

- ✧ **Nodes:** Represent individual transactions (T1 ,T2 ,…).
- ✧ **Edges:** A directed edge Ti →Tj is drawn if Ti executes an operation that conflicts with an operation of Tj , and the Ti operation occurs first.

4. Test Condition: A schedule S is conflict serializable if and only if its precedence graph contains no cycles.

5. Cycle Implication: A cycle in the graph implies that the relative order of conflicting operations between two or more transactions is reversed, meaning the result cannot be achieved by any serial execution.

Q.5(a) Draw a state diagram, and discuss the typical states that a transaction goes through during execution. (03 M)



Transaction States in DBMS

1. Active: The initial state; the transaction is executing its read/write operations.

2. Partially Committed: The transaction has executed its final operation. The changes are still in main memory buffers but are about to be written to disk.

3. Failed: The transaction enters this state if a condition is violated (e.g., integrity constraint) or if an error occurs during execution.

4. Aborted: The transaction has been rolled back, and the database state has been restored to what it was before the transaction started.

5. Committed: The transaction has successfully completed and is durable. Its changes are permanently recorded in the database.

6. Terminated: The final state, reached when the transaction has either been Committed or Aborted.

Q5(b) Explain Two phase locking protocol for guaranteeing Serializability. (04 M)

---

1. Rule: A transaction must request a lock on a data item before accessing it (Shared lock for reading, Exclusive lock for writing).

2. Growing Phase (Acquiring): The transaction can only acquire new locks. It cannot release any locks it currently holds.

3. Shrinking Phase (Releasing): The transaction can only release its existing locks. It cannot acquire any new locks.

4. Locking Point: The point where a transaction releases its first lock and enters the shrinking phase is called the lock point.

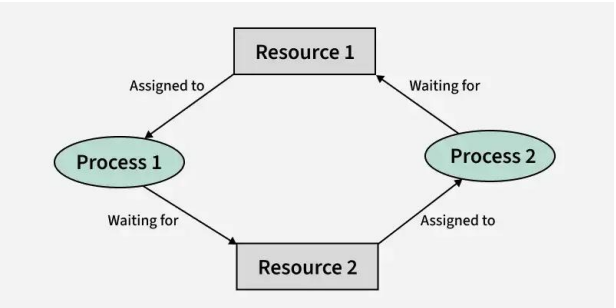
5. Serializability Guarantee: If all transactions follow the 2PL protocol, the resulting schedule is guaranteed to be conflict serializable.

6. Drawback: 2PL is susceptible to deadlocks, where transactions hold locks and wait indefinitely for each other.

Q5(c) Explain Deadlock handling in Transaction Processing. (07 M)

A deadlock is a cyclic waiting condition where two or more transactions are blocked, each waiting for a lock held by another in the cycle.

It is situation in which 2 or more process are waiting for resource to complete its execution that is being held by other process.



In above figure process 1 waiting for resource 2 that is assigned/acquired by process 2 and process 2 is waiting for resource 1 that is acquired by process 1.

Hence deadlock occurs.

Following are different techniques to resolve/handle deadlock:

1. Deadlock Prevention: Methods that ensure a deadlock can **never occur**.

- ✧ **Wait-Die Scheme (Timestamp-based):** An older transaction (Told) waiting for a younger transaction (Tyoung) is allowed to wait. A younger transaction waiting for an older one is immediately rolled back (dies).
- ✧ **Wound-Wait Scheme (Timestamp-based):** An older transaction (Told) waiting for a younger one (Tyoung) wounds the younger one (forces rollback). A younger transaction waiting for an older one is made to wait.

2. Deadlock Detection: Allows deadlocks to occur but maintains a Wait-For Graph and periodically checks for cycles.

- ✧ **Wait-For Graph (WFG):** A directed edge Ti →Tj exists if Ti is waiting for Tj to release a lock. A **cycle in the WFG** indicates a deadlock.

3. Recovery: Once a deadlock is detected, a **victim transaction** must be chosen and **rolled back** (aborted) to break the cycle.

- ✧ **Victim Selection:** The choice is based on minimizing the cost, such as the transaction with the fewest locks held or the least amount of execution time already spent.

OR

Q5(a) Explain the use of group by and having clause in SQL queries. (03 M)

---

1. GROUP BY Clause: Used to group rows that have identical values in the specified column(s). The purpose is to allow aggregate functions (like COUNT, SUM, AVG) to be applied to each unique group, yielding summary results.

2. HAVING Clause: Used to filter the groups created by the GROUP BY clause. It is the filter for aggregated data, similar to how the WHERE clause filters individual rows.

3. Example: SELECT Dept, AVG(Salary) FROM Employee GROUP BY Dept **HAVING** AVG(Salary) > 60000; (Filters to show only departments where the average salary is above 60000).

Q5(b) Explain Triggers in PL/SQL with example. (04 M)

---

1. Definition: A **stored procedural program** that is automatically executed (fired) by the DBMS in response to a specific database event.

2. Triggering Events: Typically Data Manipulation Language (DML) statements: INSERT, UPDATE, or DELETE on a table or view.

3. Timing: Can be executed BEFORE the event (e.g., to validate data) or AFTER the event (e.g., to record an audit log).

4. Granularity:

- ✧ **Row-Level:** Fires once for every row affected by the DML statement.
- ✧ **Statement-Level:** Fires only once for the entire DML statement.

5. **Use Case:** Enforcing complex business rules, automatically generating derived column values, or maintaining audit trails and security.

Q5(c) Consider Following 3 Tables and Write SQL Queries. (07 M)

Tables:

1. Books (BookID, BookTitle, Price, Author, Publisher)
2. Students (StudID, StudName, DOB, Gender, Branch, Sem)
3. Issue_Books (StudID, BookID, Issue_Date)

❖ **Query 1: List all Books whose price in range of 300 to 500 Rs.**

SQL

```
SELECT BookTitle, Price
FROM Books
```

```
WHERE Price BETWEEN 300 AND 500;
```

- **Query 2: Display all Publisher Name & Total count of Books of that publisher.**

SQL

```
SELECT Publisher, COUNT(BookID) AS TotalBooks
FROM Books
GROUP BY Publisher;
```

- **Query 3: Display list of all books which are not issued to any students.**

SQL

```
SELECT B.BookTitle
FROM Books B
LEFT JOIN Issue_Books I ON B.BookID = I.BookID
WHERE I.BookID IS NULL;
```

- **Query 4: Display the name students who are issued books in current month.**

SQL

```
SELECT DISTINCT S.StudName
FROM Students S
JOIN Issue_Books I ON S.StudID = I.StudID
WHERE YEAR(I.Issue_Date) = YEAR(CURRENT_DATE())
   AND MONTH(I.Issue_Date) = MONTH(CURRENT_DATE());
```

- **Query 5: Display all Books assigned to student with name "mahesh".**

SQL

```
SELECT B.BookTitle
FROM Books B
JOIN Issue_Books I ON B.BookID = I.BookID
JOIN Students S ON I.StudID = S.StudID
WHERE S.StudName = 'mahesh';
```

- **Query 6: Display total no of books in library**

SQL

```
SELECT COUNT(BookID) AS TotalBooksInLibrary
FROM Books;
```

- **Query 7: Display the list of girl students who have taken book from library.**

SQL

```
SELECT DISTINCT S.StudName
FROM Students S
JOIN Issue_Books I ON S.StudID = I.StudID
WHERE S.Gender = 'F';
```