Chandigarh Engineering College- CGC, Landran, Mohali
Department of Computer Science and Engineering

Assignment No.1

Subject and Subject code: Database Management Systems (BTCS 501-18)
Semester: 5th

Date on which assignment given: 14/8/2024 **Date of submission of assignment:** 20/8/2024

Total Marks: 10

Course Outcomes

CO1	Describe relational algebra expressions for a query and optimize the Developed expressions.
CO2	Design the databases using ER method and normalization.
CO3	Construct the SQL queries for Open source and Commercial DBMS.
CO4	Illustrate various methods of organizing data and transaction properties.
CO5	Implement the optimization techniques for security handling and enhance knowledge about advance databases.

Assignment related to COs		Marks Distribution	Relevance to CO No.	Level to Bloom Taxonomy
Q1	Differentiate Open source and Commercial DBMS - MYSQL, ORACLE, DB2 and SQL server.	2.5	CO1	L2
Q2	Draw an E-R model for**The Flight Database (3 marks)** The flight database stores details about an airline's fleet, flights, and seat bookings. Again, it's a hugely simplified version of what a real airline would use, but the principles are the same. Consider the following requirements list: • The airline has one or more airplanes. • An airplane has a model number, a unique registration number, and the capacity to take one or more passengers. • An airplane flight has a unique flight number, a departure airport, a destination airport, a departure date and time, and an arrival date and time. • Each flight is carried out by a single airplane.	2.5	CO1	L3

	• A passenger has given names, a surname, and a unique email address. A passenger can book a seat on a flight.			
Q3	List the different operations of Relational Algebra and Relational Calculus to retrieve the data from database with examples.	2.5	CO1	L4
Q4	Distinguish between query processing and query optimization with example. Also explain the cost function for select and join operations.	2.5	CO2	L5

Solution

Ans. 1

Comparison of Open Source and Commercial DBMS: MySQL, Oracle, DB2, and SQL Server

Database Management Systems (DBMS) can be broadly classified into **open-source** and **commercial (proprietary)** systems. Here is a comparison of MySQL (open source) with Oracle, IBM DB2, and Microsoft SQL Server (commercial):

Aspect	**MySQL (Open Source)**	**Oracle (Commercial)**	**IBM DB2 (Commercial)**	**SQL Server (Commercial)**
Ownership	Oracle Corporation	Oracle Corporation	IBM	Microsoft
License Type	Open-source (GNU General Public License), with commercial options	Proprietary	Proprietary	Proprietary
Cost	Free for open-source use; commercial licenses available for enterprise features	High licensing and support costs	High licensing and support costs	Moderate to high licensing costs
Platform Support	Cross-platform (Windows, Linux, macOS, etc.)	Cross-platform (Windows, Linux, UNIX, etc.)	Cross-platform (Windows, Linux, UNIX, etc.)	Primarily Windows (Linux support introduced)
Scalability	Moderate scalability; suitable for small to medium-sized systems	High scalability for enterprise-level systems	High scalability for large systems	High scalability for medium to large systems
Performance	Good performance for general use cases	Optimized for high-performance transactional and analytical workloads	High performance for OLTP and analytics	Optimized for business intelligence and OLTP
Support for Features	- Basic support for transactions, triggers, and views - Limited advanced analytics - No native partitioning in community version	- Advanced features (e.g., partitioning, advanced analytics) - Comprehensive PL/SQL support - High availability features like RAC	- Strong analytical and AI integration - Native XML and JSON support	- Strong integration with .NET and Azure - Advanced analytics and business intelligence

Aspect	MySQL (Open Source)	Oracle (Commercial)	IBM DB2 (Commercial)	SQL Server (Commercial)
Ease of Use	Easy to set up and use; beginner-friendly	Steeper learning curve due to advanced features	Moderate complexity for setup and use	Relatively user-friendly, especially for Windows users
Community and Support	Large open-source community; extensive online resources	Enterprise-grade support; smaller community	Enterprise-grade support	Strong enterprise support and moderate community
Use Cases	- Web applications - Content management - Small to medium businesses	- Enterprise applications - Mission-critical systems - Banking and finance	- Data warehousing - Large enterprise systems - Government systems	- Enterprise applications - Integration with Microsoft ecosystem
Customizability	High; open source allows modifications	Limited to proprietary extensions	Limited	Limited

Detailed Comparison

4. **MySQL (Open Source):**

- **Strengths**:
  - Free and open source, with an active community.
  - Easy to use and set up, making it ideal for beginners.
  - Suitable for web applications (e.g., WordPress, Drupal).
- **Weaknesses**:
  - Limited enterprise features in the community edition.
  - Lacks advanced analytics and native partitioning in the free version.

2. Oracle DB (Commercial):

- **Strengths**:
  - Rich feature set for enterprise applications.
  - Excellent performance for complex queries and large datasets.
  - Comprehensive support for clustering and fault tolerance (RAC).
- **Weaknesses**:
  - High cost of licensing and maintenance.

- Complex to manage, requiring skilled administrators.

3. IBM DB2 (Commercial):

- **Strengths**:
  - Advanced analytics and AI integration.
  - Highly optimized for large-scale OLTP and OLAP workloads.
  - Strong support for XML and JSON data handling.
- **Weaknesses**:
  - High cost, suitable for large organizations.
  - Requires expertise for setup and maintenance.

4. SQL Server (Commercial):

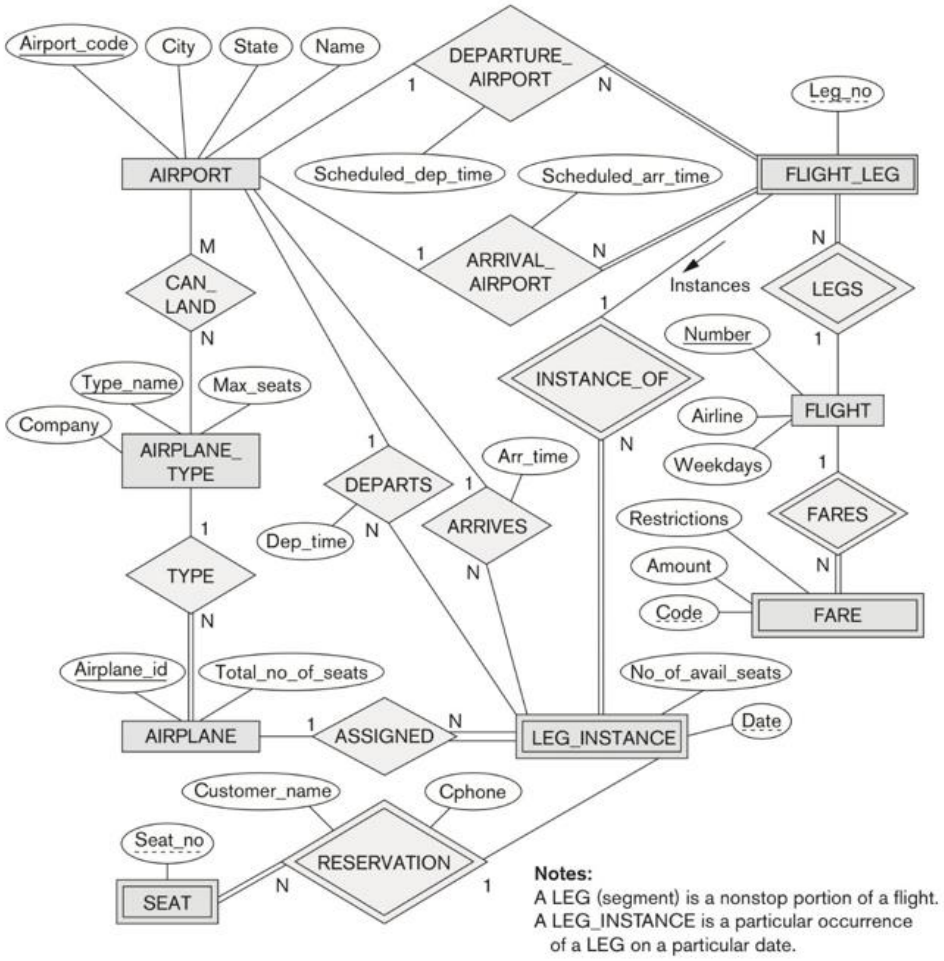
- **Strengths**:
  - Seamless integration with Microsoft products (e.g., Azure, .NET).
  - Good balance of performance and user-friendliness.
  - Strong support for business intelligence and reporting.
- **Weaknesses**:
  - Primarily designed for Windows, though Linux support is improving.
  - Licensing can be costly for larger installations.

When to Use Each DBMS

Scenario	**Recommended DBMS**
Small to medium business applications	MySQL
Enterprise systems with critical data	Oracle DB
Advanced analytics and AI requirements	IBM DB2
Windows-based systems or Azure integration	SQL Server

Ans. 2

Figure 3.21 An ER diagram for an AIRLINE database schema.



Ans. 3

Relational Algebra Operations

Relational Algebra is a procedural query language used to query a relational database. Its operations retrieve data and produce new relations as output.

1. Basic Operations

1. Selection (σ):
    - Filters rows based on a condition.
    - Example:

plaintext

Copy code

σ (Salary > 50000) (Employee)

Retrieves employees with a salary greater than 50,000.

2. Projection (π):
    - Selects specific columns (attributes) of a relation.
    - Example:

plaintext

Copy code

π (Name, Department) (Employee)

Retrieves only the Name and Department columns from the Employee relation.

3. Union (∪):

- Combines rows from two relations, removing duplicates.
- Example:

plaintext
Copy code
Student ∪ Faculty
Combines all entries from Student and Faculty.

4. Set Difference (-):
    - Retrieves rows in one relation but not in another.
    - Example:

plaintext
Copy code
Student - Faculty
Retrieves all rows in Student that are not in Faculty.

5. Cartesian Product (×):
    - Combines rows from two relations.
    - Example:

plaintext
Copy code
Employee × Department
Pairs every row of Employee with every row of Department.

6. Rename (ρ):
    - Renames a relation or its attributes.
    - Example:

plaintext
Copy code
ρ (Emp(EID, EName, EDept)) (Employee)
Renames Employee to Emp with attributes EID, EName, and EDept.

2. Advanced Operations

1. Intersection (∩):
    - Retrieves rows that are common to two relations.
    - Example:

plaintext
Copy code
Student ∩ Faculty
Retrieves entities that exist in both Student and Faculty.

2. Natural Join (⋈):
    - Combines rows from two relations based on common attributes.
    - Example:

plaintext
Copy code
Employee ⋈ Department
Joins Employee and Department where common attributes (like DeptID) match.

3. Theta Join (⋈θ):
    - Combines rows from two relations based on a condition.
    - Example:

plaintext
Copy code
Employee ⋈ (Employee.DeptID = Department.DeptID) Department

4. Division (÷):
    - Retrieves tuples in one relation associated with all tuples in another.
    - Example:

plaintext
Copy code
Projects ÷ Skills

Retrieves employees who have all required skills for a project.

Relational Calculus
Relational Calculus is a non-procedural query language where queries specify what to retrieve rather than how to retrieve it. It has two types: Tuple Relational Calculus (TRC) and Domain Relational Calculus (DRC).
1. Tuple Relational Calculus (TRC):
- Queries specify conditions on tuples.
- General Form:

plaintext
Copy code
{ t | Condition(t) }
- Example:

plaintext
Copy code
{ t | t ∈ Employee ∧ t.Salary > 50000 }
Retrieves tuples t from Employee where Salary > 50000.
2. Domain Relational Calculus (DRC):
- Queries specify conditions on attribute values (domains).
- General Form:

plaintext
Copy code
{ <a1, a2, ..., an> | Condition(a1, a2, ..., an) }
- Example:

plaintext
Copy code
{ <Name, Department> | ∃ t (t ∈ Employee ∧ t.Name = Name ∧ t.Department = Department ∧ t.Salary > 50000) }
Retrieves Name and Department of employees earning more than 50,000.

Comparison of Relational Algebra and Relational Calculus

Aspect	Relational Algebra	Relational Calculus
Type	Procedural	Non-procedural
Focus	Specifies *how* to retrieve data	Specifies *what* data to retrieve
Complexity	Easier to optimize and implement	More user-friendly
Usage	Used in query execution engines	Basis for SQL language structure

Ans. 4

Distinguishing Query Processing and Query Optimization

Aspect	**Query Processing**	**Query Optimization**
Definition	The steps a DBMS takes to interpret and execute a query.	The process of improving a query execution plan to reduce resource usage.
Focus	Transforming a high-level query (e.g., SQL) into a low-level plan for execution.	Choosing the most efficient execution strategy for the query.
Objective	Ensure the query is executed correctly and produces the desired result.	Minimize the cost of query execution (e.g., time, memory, CPU).
Techniques Involved	Parsing, validation, logical and physical query plan generation.	Cost estimation, selection of algorithms, reordering operations.

Aspect	Query Processing	Query Optimization
Example	Processing an SQL query to compute results.	Choosing between nested loops or hash joins for a join operation.

Example of Query Processing and Optimization
Query Example:
sql
Copy code
SELECT E.Name, D.DeptName
FROM Employee E, Department D
WHERE E.DeptID = D.DeptID AND E.Salary > 50000;

1. Query Processing:
    - SQL is parsed and validated.
    - Logical plan: Create a cartesian product of Employee and Department, filter rows where DeptID matches and Salary > 50000, and project Name and DeptName.
    - Physical plan: Execute the operations using specific algorithms (e.g., nested loop join).
2. Query Optimization:
    - The optimizer evaluates multiple strategies, such as:
        - Join Ordering: Perform the Salary > 50000 filter on Employee first to reduce the number of rows before the join.
        - Algorithm Choice: Use a hash join instead of nested loops if Employee and Department are large.
    - The most cost-effective plan is selected.

Cost Functions for Select and Join Operations
1. Select Operation (σ):
Cost is determined by how the selection condition is evaluated:

- Full Table Scan: Scans all rows in the table.
    - Cost = Number of blocks in the table.
- Index Scan: Uses an index to locate rows matching the condition.
    - Cost = Cost of searching the index + Number of data block accesses.
- Example:

sql
Copy code
SELECT * FROM Employee WHERE Salary > 50000;

    - Full Scan: If no index exists, all rows are scanned.
    - Index Scan: If an index on Salary exists, only relevant rows are accessed.

2. Join Operation:
Cost depends on the algorithm used:

- Nested Loop Join:
    - Cost = BR×BSB_R \times B_SBR×BS, where BRB_RBR and BSB_SBS are the number of blocks for relations R and S.
    - Example:

sql
Copy code
SELECT * FROM Employee E, Department D WHERE E.DeptID = D.DeptID;
Every row in Employee is matched with rows in Department.

- Sort-Merge Join:
    - Cost = Cost of sorting R + Cost of sorting S + Cost of merging.
    - Efficient for sorted data or equi-joins.
- Hash Join:
    - Cost = Building a hash table on R + Probing the hash table with S.
    - Ideal for large, unsorted datasets.