****

**University of Ghana**

**School of Engineering Sciences**

**Department of Computer Engineering**

**CPEN 211: Database Systems Design**

*Group members*

22016968 - Sarpong Jeffrey Somuah

22042860 - Mensah Philemon

22043997 - Aidoo-Taylor Kwamena

22231991 - Henry Otwey Baidoo

22132121 - Amoah Pearl Owusua

22020624 - Manal Abdul - Kadi Mohammed

22071976 - Eklo Christopher Yao

22049764 - Krampah Jonathan

11365437 - Captain Godsgift

**INTRODUCTION**

Databases are collections of organized information used by computers. They are essential for managing large amounts of data efficiently. For example, social media platforms like Facebook and Instagram use databases to store information about their users and their posts.

Efficient data management is crucial for businesses and organizations of all sizes. It helps to improve decision-making, reduce costs, and increase productivity. For example, a retail store can use a database to track inventory levels and sales data, which can help them to make better decisions about what products to stock and when to run sales.

Query processing, query optimization, and ACID properties are all important for ensuring that databases are efficient and reliable. This report will provide an overview of these concepts and discuss their importance in the real world.

**Query Processing in Database Management Systems**

* **Definition**

Query processing is the process of executing queries against a database by translating high-level queries, such as SQL, into low-level expressions that the system can understand and execute (Elmasri & Navathe, 2021). While SQL is human-readable, relational algebra provides a more efficient structure for systems to interpret queries, ensuring optimal data retrieval (Ramakrishnan & Gehrke, 2020).

Query processing is critical as it enables users to access and manipulate data efficiently while ensuring data accuracy and consistency. By optimizing execution plans, query processing reduces resource usage and enhances system performance, particularly in environments with large datasets or high query volumes. For example, when filtering employees earning over $5000, query processing ensures indexing is used to minimize unnecessary scans, preserving resources. This makes query processing essential for reliability and scalability in database management systems (Elmasri & Navathe, 2021).

* **Steps in Query Processing**:

1. Parsing:

The parsing step is the first step in query processing. It involves breaking down the user's query into smaller components and checking for syntax errors. This is similar to how a grammar checker works in a word processor.

Once the query is parsed, it is translated into an internal form that the database system can understand. This internal form is often a tree-like structure called a parse tree. The parse tree represents the logical structure of the query.

The parsing step is important because it ensures that the query is correct and can be processed by the database system. It also helps to identify any errors in the query so that they can be corrected.

1. Translation

After parsing, the query is **translated into an intermediate form**, typically using **relational algebra**. This intermediate form is optimized for internal representation and execution. Relational algebra simplifies the query by representing operations like selection, projection, and joins mathematically. The parser also replaces any views in the query with their base relations for efficient execution.

This translation enables the DBMS to perform query optimization and execution more effectively, bridging the gap between user-friendly query languages and system-level operations (Ramakrishnan & Gehrke, 2020).

1. Optimizing

The **optimization phase** of query processing focuses on improving the efficiency of query execution by minimizing its cost. While a user typically writes queries in a high-level language like SQL, they do not need to worry about optimizing them manually. Instead, the **query optimizer** within the database system automatically generates the most efficient evaluation plan. The optimizer evaluates various possible execution strategies and selects the one that minimizes resource usage, such as CPU, memory, and disk I/O, which directly impacts performance (Elmasri & Navathe, 2021).

The main **objective** of query optimization is to reduce the total **cost** of executing a query. This cost is estimated by analyzing factors like memory allocation, the execution time for different operations, and the number of resources required for scanning, joining, or sorting data. The query optimizer considers factors such as index usage, access paths, and join algorithms to find the optimal plan. The goal is to minimize the time and resources required to retrieve the correct result set while ensuring scalability as the database grows (Ramakrishnan & Gehrke, 2020).

1. Evaluation

The **evaluation phase** in query processing focuses on executing the translated query using a **query evaluation plan**. After the query is translated into relational algebra and optimized, the system annotates the relational algebra expressions with the necessary instructions for evaluating each operation. These instructions specify the algorithms and methods to be used, such as which index to access or which join algorithm to apply, ensuring that the operations are carried out efficiently (Elmasri & Navathe, 2021).

This annotated version of relational algebra, called **evaluation primitives**, defines the sequence of operations the system must follow to execute the query. Each primitive operation corresponds to a database task, such as scanning a table, applying a filter, or performing a join. The **query evaluation plan** or **execution plan** outlines these steps, detailing how to process the query with minimal cost (Ramakrishnan & Gehrke, 2020).

The **query execution engine** is responsible for executing the evaluation plan and generating the final output for the user. It systematically carries out the operations specified in the plan, ensuring that the query is evaluated correctly and efficiently, returning the desired results to the user.

**ACID PROPERTIES IN DATABASE MANAGEMENT SYSTEMS**

* **Definition**

ACID refers to the four fundamental properties—Atomicity, Consistency, Isolation, and Durability—that ensure reliable processing of transactions in a database management system (DBMS). The ACID properties are crucial for maintaining the accuracy, reliability, and integrity of data in a database, ensuring that all operations are processed correctly, consistently, and without data corruption, even under high load or system failures.

1. Atomicity

Atomicity in a database management system (DBMS) ensures that all operations within a transaction are either fully completed or not executed at all. This property guarantees data integrity by preventing partial updates, which could lead to inconsistencies.

Atomicity is achieved through two key operations:

* Commit: This operation makes all changes made during the transaction permanent and visible to other users.
* Abort: If an error occurs or the transaction cannot be completed, this operation undoes all changes, ensuring that no partial updates are visible in the database.

For example, in a banking system where money is transferred from Account A to Account B, atomicity ensures that if the debit from Account A is successful but the credit to Account B fails, the entire transaction will be rolled back. This rollback ensures that no funds are lost or incorrectly transferred, preserving the consistency of the database.

By simplifying error handling and preventing data inconsistencies, atomicity is essential for ensuring the reliability and robustness of the database

1. Consistency

Consistency in a database ensures that it remains in a valid state before and after a transaction. Every transaction must transition the database from one valid state to another, maintaining all predefined rules, constraints, and integrity checks. This property ensures that the data adheres to the rules of the database, such as foreign key constraints, data types, and other business logic.

Inconsistency arises when data does not meet these rules. For example, in a banking transaction where money is transferred from one account to another, if the money is debited from Account A but fails to be credited to Account B, the total balance in the system would become incorrect, violating the consistency rule.

To prevent such inconsistencies, the database system triggers a rollback, undoing any partial changes and returning the database to its previous valid state. This guarantees that, even in the case of failure, the database will not contain invalid or corrupted data, preserving its integrity and ensuring reliable operations.

1. Isolation

Isolation in ACID properties ensures that transactions are executed independently, without interfering with one another. This means that the operations of one transaction are not visible to others until the transaction is fully committed. Isolation guarantees that each transaction appears to execute in isolation, preserving the integrity of the database even when multiple transactions occur concurrently.

For instance, consider two transactions:

* Transaction A: Transfers $500 from Account X to Account Y.
* Transaction B: Reads the balance of Account X and Account Y.

Without isolation, **Transaction B** might read an intermediate state where $500 has been deducted from Account X but not yet credited to Account Y, leading to incorrect or inconsistent results. However, with isolation, **Transaction B** will either see the balances before or after **Transaction A**, but not during the transaction, ensuring that it reads only consistent data.

By ensuring that one transaction's incomplete operations do not interfere with others, isolation helps maintain the correctness of the database and prevents issues like dirty reads, non-repeatable reads, and phantom reads.

1. Durability

Durability in ACID properties ensures that once a transaction has been committed, its changes are permanent, even in the event of a system crash or failure. Once data is written to the database, it is guaranteed to persist, meaning it will not be lost or undone, and can be accessed consistently.

For example, in a banking system, if a transaction involves transferring money from one account to another and the system crashes after the transaction is committed, **durability** ensures that the money transfer is completed and permanently recorded in the database. The changes made by the transaction will persist, and the system can reliably recover the committed data without any loss, even after a failure.

This property is essential for ensuring that once users interact with the database, the results of their actions are safe, reliable, and consistent over time, reinforcing the system's trustworthiness.

**Query Optimization in DBMS**

It is a process that improves the efficiency of database queries by finding the most effective way to execute them. Its goal is to reduce the time taken to retrieve information from the database and use the system resources more efficiently.

Queries that have bad optimization can lead to a poor user experience and slow performance of applications since it would spend more time and resources in providing a result to a user.

An optimizer is used to calculate the various losses of multiple paths in executing a query, the path with the lowest cost is selected and then executed.

There are various methods that can be used to improve query optimization for better user experience:

1. Using indexes: An index is a data structure that stores a copy of selected columns of a table. Indexes increases the speed of which it takes the system to retrieve information from a database from the specific columns that have been indexed. When indexes are created, the system would also need to manage the indexes in addition to the tables. It is advised that indexes are created on just the columns that it would be necessary for it to be created on for faster searches.
2. Re-writing queries: Queries should be inspected and changed where necessary to make the query more efficient. This can be done by simplifying the query or breaking the query down into smaller subqueries. Some keywords can be changed to make things easier on the system. Keywords like having can be swapped with a where clause which is much faster than using the having keyword. When querying data rather than using “Select \*” which will select all the columns in a table, use the specific column name you want. This would increase the speed and reduce the amount of information the system has to iterate over to return the needed results.
3. Partitioning: This involves dividing larger tables into smaller ones. This makes it easy to manage the pieces of information that is to be called by the query. Reducing the number of tuples to be searched through makes the query faster to execute.
4. Views: Views are queries that have already been computed in the database for frequently executed queries. They make it faster for queries that have already been pre-computed using views thereby improving the overall user experience
5. Reduction in the usage of subqueries: Subqueries are queries written inside other queries. An example of a subquery would be “ SELECT \* FROM school WHERE teacher\_id = (SELECT id FROM teachers WHERE satisfaction > 4)”. In this case the statement written in the where clause which is “(SELECT id FROM teachers WHERE satisfaction > 4)”, is a subquery of the main query “SELECT \* FROM school”. Making the system go through a query with a lot of subqueries especially daunting ones can greatly impact the speed of retrieval of information from the database. Subqueries should be simplified if possible to improve the performance of the database.

**Conclusion**  
Efficient database management is a cornerstone of modern technology and business operations. As explored, query processing, ACID properties, and query optimization are integral to maintaining robust, reliable, and high-performing database systems.

Query processing ensures the accurate translation, optimization, and execution of user queries, enabling the retrieval of data with minimal resource expenditure. The ACID properties safeguard data integrity and reliability, guaranteeing that transactions are handled consistently, even in the face of concurrent processes or system failures. Query optimization further enhances system performance, improving user experience and maximizing the efficiency of resources.

Together, these principles underscore the importance of designing and maintaining databases that are reliable. By using techniques such as indexing, query rewriting, and partitioning, organizations can ensure their databases are functional and adaptive to growing demands. Mastering these concepts would be crucial for any organization.