

|  |  |
| --- | --- |
| ECE59500 | |
|  |  |
| Assignment 5 |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | |  |
| Part I: Query Processing and Optimization **1: Block Size and Access Time Calculation:**   * **BOOK Relation**   Size on disk = 172 blocks  Access time for linear scan \* 5 \* = 8.8 ×   * **CUSTOMER Relation**   Size on disk = 391 blocks  Access time for linear scan \* 5 \* = 1.955 ×   * **Sale Relation**   Size on disk = 489 blocks  Access time for linear scan \* 5 \* = 2.445 ×  **2: Number of Possible Query Plans:**  In order to figure out the total number of feasible query plans, it is essential to take into account both the multiple join orders and selection strategies. Due to the restriction of using only Block Nested Loops (BNL) for joins and linear scans for projections and selections, the range of potential query plans will be rather restricted in comparison with additional advanced optimisation methods such as Cost-Based Optimisation (CBO) or Dynamic Programming.  **Possible Query plans are:**   * **Join Order:**   The number of possible join orders is 6! Which is 720. Because there are 6 relations/tables.   * **Selection Order:**   There are many selection predicates involving different relations. Each predicate can be employs in different orders. The no. of possible selection orders rely on the number of selection predicates exist in the above query. For n selection predicates number of selection predicates is n!   * **Combining Join and Selection order:**   The join order can be combined with another selection order, and results in a total number of possible query plans. For example if selection predicates (n=3) then possible query plans will be:  **720 × 3! = 720 × 6 = 4320**  Considering just BNL for joins and linear scans for projections and selections, is the product of the number of possible join orders and the number of possible selection orders.  **3: Best Join Order**  Considering many factors such as the cardinality of relations, selectivity of predicates, and the availability of indexes the best join order for the given query can be determined. Index Nested Loops (INL) for joins and index-based selections will be used that effect execution time since indexes are available.  The best join order for this query would be one that reduces number of disk accesses and optimize join order. It would start with the CUSTOMER table, utilizing indexes on PhoneNo for selection. After that SALE table, using indexes on ISBN for selection. Eventually, joining the BOOK table would be followed, filtering first by Author and then Year that would decrease search space and improve efficiency.  **4**: **Performance Improvement of Query:**  The SUBSTR can be eliminated to improve the query efficiency. Updated query is as:   |  | | --- | | SELECT B1.Title, COUNT(B2.Title) as NumOfSequels  FROM BOOK as B1  JOIN BOOK as B2 ON B1.Genre = B2.Genre  AND B1.Author = B2.Author  AND B2.Title LIKE CONCAT(B1.Title, '%')  WHERE B1.Genre = 'Horror'  AND B1.Author = 'Stephen King'  GROUP BY B1.Title; |   The updated query more comprehensively do the comparison between B1.Title and B2.Title by using the LIKE operator with CONCAT rather of the SUBSTR function. By avoiding costly string manipulation operations performance is improved. Moreover, the conditions for Genre and Author are implemented to both B1 and B2, making sure the consistency and making the above query more efficient. The updated query enhances performance and readability by streamlining the query logic.  **5: Translating the Query:**  Each operation is represented explicitly without optimizing selections or Cartesian products in order to translate the query into Relational Algebra in a naive manner .Also a left-deep join tree without merging selections will be used.  **Translated Query:**   |  | | --- | | π\_Title, COUNT(π\_Title) as NumOfSequels (  σ\_Genre='Horror' ∧ Author='Stephen King' (  BOOK as B1 ⨝\_{B1.ISBN=S1.ISBN} σ\_Author='Stephen King' (BOOK as S1)  )  ⨝\_{B1.ISBN=S1.ISBN} σ\_Genre='Horror' ∧ Author='Stephen King' (  BOOK as B2 ⨝\_{B2.ISBN=S2.ISBN} σ\_Author='Stephen King' (BOOK as S2)  )  ) |   **Execution Time:**  Calculating the execution time on the bases of cost of each operation in the query plan:   * **Selection (σ)**   For Genre and Author Selection.  Cost = 2 × 35000  Total = 2×2 × 35000 = 140,000   * **Join (⨝)**   Cost = 2 ×  Total = 2×2 × = 4,900,000,000   * **Projection (π)**   Cost = N (no. of distinct titles)  Count  Cost = M (no. of result tuples)  **Assumptions:**  This naive translation presume straightforward execution with no optimising methods such as index consumption or query plan reorganisation.  It implies a simple execution strategy in which each action is performed sequentially without the use of parallel processing.  The time required to execute the query plan is governed by a number of factors, including the size of the BOOK table, the efficiency of join operations, and the processing resources available. Due to the core technique and lack of optimisation, execution durations for huge datasets may be rather lengthy.  **6: New Optimized Query Plan:**  **1: Start with selecting rows from the table BOOK**  σ\_Genre='Horror' ∧ Author='Stephen King'(BOOK)  **2: Perform Self Join on the filtered rows**  B1\bowtie\_{B1.Title \land B1.Author = B2.Author \land B2.Title \text{LIKE} CONCAT (B1.Title, '%')}B2  **3: Count the number of sequels and Group the result by B1.Title**  πB1.Title, COUNT(B2.Title) as NumOfSequels   * **Intermediate Results:**   All operations can be pipelined for efficiency. No intermediate results require to be written out and passed on the next operation without sorting them on disk.   * **Indexes:**   Indexes in the BOOK table's 'Genre' and 'Author' columns may be useful for making the first choice. Potentially useful for maximizing the self-join operation is an index on "Title". In the event that suitable indexes for joins and selects are accessible, Index Nested Loops (INL) may be employed.   * **Query execution and Speedup:**   To determine the query execution time, we must evaluate the cost of each operation in the optimized plan. The plan's emphasis on pushing down selections and projections, as well as merging Cartesian products via join selections, should result in speedier execution. Comparing the execution time to the prior plan from Question 5 yields the percentage speedup gained through the optimized plan.  Execution Time:  Percentage Speedup =  = 75.43 % Part II: Transaction and Recovery Management **7: Schedules**   To determine how many potential schedules there are for three transactions, each with ten operations permutations with repetitions are used.   There are 10 procedures for every transaction, and sequence in which these operations can be executed are required to be figured out. Each transaction can be treated as a distinct collection of operations because the operations that constitute it are distinct from one another.  **Regarding three transactions with ten operations individually:**   1: The total number of possible schedules, which represents the permutations of operations across transactions, is  **Regarding the bonus query about the Strict 2PL Protocol:**   2: There are conceivable schedules under Strict 2PL, where each transaction has 5 writes and 5 reads to the same item. These schedules reflect the permutations while abiding by the protocol's restrictions.  **8: Transactions:**   1. **(Conflict) Serializable but not Strict Two-Phase Locking.**   **Schedule: T1:Read(A),T2:Write(A),T3:Read(A)**  **Explanation:** There are no competing read-write or write-write operations on a single data object, making this schedule conflict-serializable. Instead if transactions do not obtain and release locks in a strictly two-phase way, it's not Strict Two-Phase Locking. Although T2's write operation is interspersed among T1 and T3 without adequate locking, T1 and T3 are able to read concurrently in this schedule without breaking conflict serializability.   1. **Strict Two-Phase Locking but not Serial.**   **Schedule: T1:Write(A),T2:Write(A),T3:Write(A)**  **Explanation:** This schedule conforms to the Strict Two-Phase Locking protocol, which demands that locks be acquired prior any actions are carried out and released only when all activities have been completed. But it's not serial as the tight serializability condition is broken by concurrent transaction execution.   1. **Recoverable but not Strict Two-Phase Locking.**   **Schedule: T1:Write(A),T2:Read(A),T3:Write(A)**  **Explanation:**  Since T2 just reads material that has been done by T1, this schedule is recoverable. Because transactions do not obtain and issue locks in a strictly two-phase way, it's not Strict Two-Phase Locking. T2 is able to read in this schedule before obtaining a lock, which is against the rigorous locking process.   1. **Two-Phase Locking (not strict) but not Recoverable. e. (Conflict) Serializable but not Serial.**   **Schedule: T1:Write(A),T2:Write(A),T3:Read(A)**  **Explanation:**  Since T3 reads data that T1 has updated but not dedicated, this schedule cannot be recovered. Transactions obtain locks prior to executing any operations and release locks upon completion, it complies with non-strict Two-Phase Locking. Because T3 reads A in this schedule before T1 unlocks the lock, there could have been an improper read.   1. **Conflict) Serializable but not Serial**   **Schedule: T1**: **Read(A),T2:Write(A),T3:Read(A)**  **Explanation:**  Here cannot be competing read-write or write-write operations on a single data item, making this schedule conflict-serializable. But it isn't serial as the serializability condition is broken by concurrent transaction execution.  **9: Difference and Similarities between Undo/Redo Operation**   |  |  | | --- | --- | | **Undo Operation** | **Redo Operation** | | **Effect on Database State:** Removes the effects of uncommitted transactions by returning the database to its initial state prior to the initiation of a transaction. | **Effect on Database State:** Ensures that all committed modifications are represented in the database by reapplying the modifications made by committed transactions after restoration | | **Effect on Log Records:** Reverses and identifies changes performed by uncommitted transactions using log records. | **Effect on Log Records:** Reapplies commits made by transactions that, because of a system failure, were not yet reflected in the database. | | **Effect on Lock Table:** Obtaining locks could be necessary to preserve consistency when recovering the database's current state. | **Effect on Lock Table:** Since it implements committed modifications again without changing the previous transaction, it usually has no impact on the lock database. |   **Example:** Suppose Transaction T1 modifies a record, commits, and Transaction T2 accesses the changed record afterwards. However, there is a system breakdown prior to Transaction T2 finishing.  **T1: Update**  **T1: Commit**  **T2: Read**  **Undo Operation:** Needed to reverse T1's modification in order to return the database to its initial state.  **Redo Operation:** In order to ensure that T2 gets the updated record, it is necessary to redo T1's update to reflect the committed change in the database.  **10:** **Check pointing**   |  | | --- | | * **Atomicity**: By adding a checkpoint record to the log that specifies the database's operational state at a specific moment, check pointing guarantees atomicity. This guarantees that all of the transactions that have been committed before to the checkpoint will either be reflected in the database or not at all. | | * **Durability:** Checkpointing improves durability by lowering the number of log records that must be repeated during recovery. The moment at which the database state is consistent and effectively recoverable is indicated by the checkpoint. |  |  | | --- | | **Advantages** | | * Check pointing is an efficient method of recovery because it enables the system to bypass log data preceding the checkpoint, hence reducing the number of log records that need to be analyzed. This results in a reduction in the amount of time required for recovery.  If check pointing is evaluated to strategies that include frequent writes to the log then check pointing will have a lower overhead need during the transaction processing phase. | | * Check pointing is an efficient method of recovery because it enables the system to bypass log data preceding the checkpoint, hence reducing the number of log records that need to be analyzed. This results in a reduction in the amount of time required for recovery.  If check pointing is evaluated to strategies that include frequent writes to the log then check pointing will have a lower overhead need during the transaction processing phase. |  |  | | --- | | **Disadvantages** | | * Keeping checkpoints updated may necessitate more storage capacity, particularly for datasets that are significantly large. | | * The recovery point is only available for the latest checkpoint, which means that if there is a failure during checkpoints, there is a possibility that data will be lost. |   **Shadow Paging**   |  | | --- | | * **Atomicity:** The atomicity of the database is maintained by shadow paging, which does this through keeping a distinct shadow copy of the database. Following the completion of a transaction, the shadow copy is transformed into the newly operational database. This ensures that either all of the modifications made by the transaction are captured in the database or not any of them are updated at all. | | * **Durability:** Shadow paging improves durability by making sure committed changes are written to stable storage. This helps to ensure that the modifications are not lost. As soon as a transaction is committed, the modifications are applied to the stable shadow copy. This ensures that the modifications are resilient even in the event that the system fails. |  |  | | --- | | **Advantages** | | * Shadow paging just requires deleting the uncommitted shadow pages, which speeds up the recovery process. | | * During the processing of transactions, log overhead is reduced because to shadow paging, which avoids the requirement to write to a log file frequently. |  |  | | --- | | **Disadvantages** | | * Shadow page maintenance necessitates extra space in storage, which can be substantial for big databases. | | * The necessity to refresh shadow pages while processing transactions means that shadow paging could potentially slow things down. |   **Scenarios where one technique is preferable to the other one:**   * Reducing recovery time and log overhead is the main issue when check pointing is preferred. For example, check pointing can drastically cut down on recovery time in systems with high transaction volumes and regular checkpoints by reducing the amount of log records the requirement to be examined. * When streamlining recovery and decreasing log overhead are priorities, shadow paging is preferable. For example, shadow paging can offer quicker recovery times and less log overhead than check pointing in systems with small space for storage or when recovery time is crucial.  Part III: Practical SQL | | | |
|  | | | |
|  | |  | |
|  | |

|  |
| --- |
|  |