**Question 1**

**Task:** Write a SQL query to find all books that have been loaned out at least once this year but are not currently on loan (i.e., no NULL values on the ReturnedDate column). Your result should include the ISBN, Title, Author, and the most recent LoanDate for these books. Your solution MUST use the semi-join operator.

**SQL Query:**

SELECT B.ISBN, B.Title, B.Author, MAX(L.LoanDate) as MostRecentLoanDate

FROM Books B

WHERE EXISTS (

    SELECT 1

    FROM Loans L

    WHERE B.BookID = L.BookID

    AND YEAR(L.LoanDate) = YEAR(CURRENT\_DATE)

    AND L.ReturnedDate IS NOT NULL

)

GROUP BY B.ISBN, B.Title, B.Author;

**Question 2**

1. What is meant by the term heuristic optimization?

**Heuristic optimization** refers to the use of rules of thumb or heuristic methods to improve the efficiency of database query processing. Instead of exploring all possible execution plans, heuristic optimization applies a series of transformations to a query tree based on predefined rules that are known to generally improve performance. These rules are not guaranteed to produce the optimal solution but provide a good-enough solution with reduced computational effort.

1. In your opinion, which step(s) in the Heuristic Algebraic Optimization Algorithm is the most powerful in reducing the size of intermediate data files?

The most powerful step in reducing the size of intermediate data files is **pushing selections down the query tree**. By applying selection operations as early as possible, we can filter out unnecessary rows sooner, reducing the size of the data processed in subsequent operations. This step minimizes the amount of data that needs to be joined, aggregated, or further filtered, leading to more efficient query execution.

1. In your opinion, based on the existing Heuristic Algebraic Optimization Algorithm, are there additional heuristic rules/steps that you'd like to include into it?

Additional heuristic rules/steps that could be included:

* **Join Reordering:** Reorder joins based on the sizes of the tables and the selectivity of join conditions to minimize the size of intermediate results.
* **Projection Pushdown:** Push projection operations (selection of specific columns) down the query tree to reduce the amount of data passed between operations.
* **Materialized View Usage:** Recognize opportunities to use existing materialized views to satisfy parts of the query, potentially reducing computation by reusing precomputed results.

**Question 3**

Task: Construct the query tree in canonical form. A canonical query tree corresponds to a standard relational algebra expression of an SQL query directly (i.e., a pre-heuristic-optimization query tree).

**Canonical Query Tree:**

π\_{m.mov, m.mov\_title, mc.role, a.act\_fname, a.act\_iname}

σ\_{g.genres\_title = 'Sci-Fi' AND m.lang = 'English' AND m.year >= 1900 AND m.year < 2000}

    ⨝\_{m.mov\_id = mc.mov\_id}

        ⨝\_{mc.act\_id = a.act\_id}

            ⨝\_{m.mov\_id = mg.mov\_id}

                ⨝\_{mg.gen\_id = g.gen\_id}

                    (movie m)

                    (movie\_cast mc)

                (actor a)

            (movie\_genres mg)

        (genres g)

**Question 4**

Task: Construct an equivalent query tree by applying the heuristic optimization algorithm to it. For each heuristic performed, state the reasons behind it. Your response must NOT be hand-drawn.

**Optimized Query Tree:**

π\_{m.mov, m.mov\_title, mc.role, a.act\_fname, a.act\_iname}

σ\_{g.genres\_title = 'Sci-Fi' AND m.lang = 'English' AND m.year >= 1900 AND m.year < 2000}

    ⨝\_{m.mov\_id = mc.mov\_id}

        ⨝\_{mc.act\_id = a.act\_id}

            ⨝\_{m.mov\_id = mg.mov\_id}

                ⨝\_{mg.gen\_id = g.gen\_id}

                    (σ\_{g.genres\_title = 'Sci-Fi'}(genres g))

                    (σ\_{m.lang = 'English' AND m.year >= 1900 AND m.year < 2000}(movie m))

                (movie\_cast mc)

            (actor a)

        (movie\_genres mg)

**Heuristics Applied:**

1. **Selection Pushdown:** Pushing selection operations as close to the base relations as possible. This reduces the number of tuples processed in the joins.

* σ\_{g.genres\_title = 'Sci-Fi'}(genres g)
* σ\_{m.lang = 'English' AND m.year >= 1900 AND m.year < 2000}(movie m)

**Question 5**

Task: For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project.

**SQL Query:**

SELECT Pnumber, Pname, COUNT(\*)

FROM PROJECT, WORKS\_ON, EMPLOYEE

WHERE Pnumber = Pno AND Ssn = Essn AND Dno = 5

GROUP BY Pnumber, Pname;

1. Draw at least two equivalent query trees (using Relational Algebra) that can represent the query.

**Query Tree 1:**

π\_{Pnumber, Pname, COUNT(\*)}

    γ\_{Pnumber, Pname, COUNT(\*)}

        σ\_{Dno = 5}

            ⨝\_{Ssn = Essn}

                ⨝\_{Pnumber = Pno}

                    (PROJECT)

                    (WORKS\_ON)

                (EMPLOYEE)

**Query Tree 2:**

π\_{Pnumber, Pname, COUNT(\*)}

    γ\_{Pnumber, Pname, COUNT(\*)}

        ⨝\_{Pnumber = Pno}

            (PROJECT)

            ⨝\_{Ssn = Essn}

                σ\_{Dno = 5}(EMPLOYEE)

                (WORKS\_ON)

2. Draw the initial query tree, in its canonical form, for the query, and then show and discuss how the query tree can be optimized using pipelining.

**Initial Canonical Query Tree:**

π\_{Pnumber, Pname, COUNT(\*)}

    γ\_{Pnumber, Pname, COUNT(\*)}

        σ\_{Dno = 5}

            ⨝\_{Ssn = Essn}

                ⨝\_{Pnumber = Pno}

                    (PROJECT)

                    (WORKS\_ON)

                (EMPLOYEE)

**Optimized Query Tree with Pipelining:**

π\_{Pnumber, Pname, COUNT(\*)}

    γ\_{Pnumber, Pname, COUNT(\*)}

        ⨝\_{Pnumber = Pno}

            (PROJECT)

            ⨝\_{Ssn = Essn}

                σ\_{Dno = 5}(EMPLOYEE)

                (WORKS\_ON)

**Pipelining Optimization:**

* By performing the selection `σ\_{Dno = 5}` on the `EMPLOYEE` table first, we reduce the number of tuples involved in the join operations, leading to more efficient query execution.
* Pipelining allows us to process tuples on-the-fly without storing intermediate results, which reduces I/O costs and improves performance.

3. Apply pipelining to the query trees drawn in part (a), compare them with the initial query tree in part (b). Which query trees benefit the most from pipelining?

**Comparison:**

* **Query Tree 1** benefits from pipelining by reducing the number of intermediate results after the selection operation.
* **Query Tree 2** benefits from pipelining as it performs the selection operation on the `EMPLOYEE` table before joining with `WORKS\_ON`.

**Which Benefits the Most:**

* **Query Tree 2** benefits the most from pipelining as it applies the selection operation earlier, thus minimizing the number of tuples involved in the join operations, resulting in lower I/O costs and faster query execution.