Query Optimisation

Query Optimisation

- In practice, query optimisers incorporate elements of the following three optimisation approaches:
  - **Semantic query optimisation**

    Use application specific semantic knowledge to transform a query into the one with a lower cost (they return the same answer).
  - **Rule-based query optimisation**

    Use heuristic rules to transform a relational algebra expression into an equivalent one with a possibly lower cost.
  - **Cost-based query optimisation**

    Use a cost model to estimate the costs of plans, and then select the most cost-effective plan.

Semantic Query Optimisation

- Can we use **semantic information** stored in a database (such as integrity constraints) to optimise queries?
  - semantics: “meaning”.
- Recall that, **integrity constraints in the relational model** include:
  - key constraints
  - entity integrity constraints
  - referential integrity constraints
  - domain constraints
  - . . .
  - user-defined integrity constraints
- **Key idea:** Integrity constraints may **not only be utilized to enforce consistency** of a database, but may **also optimise user queries**.

Semantic Query Optimisation

- **Example 1**:

  Constraint: The relation `Employee` has the primary key $\{$`ssn`$\}$.

  Query:
  ```
  SELECT DISTINCT ssn FROM Employee;
  ```

  - We can avoid extra costs for duplicate elimination if the existing constraint tells us that tuples in the result will be unique.

Semantic Query Optimisation

- **Example 2**:

  Constraint: No employee can earn more than 200000.

  Query:
  ```
  SELECT name
  FROM Employee
  WHERE salary > 300000;
  ```

  - We do not need to execute a query if the existing constraint tells us that the result will be empty.

Semantic Query Optimisation

- **Example 3**:

  Constraints: The relation WORKS_ON has the foreign keys:
  [ssn]⊆EMPLOYEE[ssn] and [pno]⊆PROJECT[pnumber]

  Query:
```
SELECT DISTINCT ssn
FROM Works_on INNER JOIN Project
on Works_on.pno=Project.pnumber;
```

  - We can reduce the number of joins by executing the following query since both queries always return the same result.

```
SELECT DISTINCT ssn
FROM Works_on;
```

Rule-based Query Optimisation

- A rule-based optimisation transforms the RA expression by using a set of heuristic rules that typically improve the execution performance.
- **Key ideas:** apply the most restrictive operation before other operations, which can reduce the size of intermediate results:
  - **Push-down selection:**

    Apply as early as possible to reduce the number of tuples;
  - **Push-down projection:**

    Apply as early as possible to reduce the number of attributes.
  - **Re-ordering joins**:

    Apply restrictive joins first to reduce the size of the result.
- But we must ensure that the resulting query tree gives the same result as the original query tree, i.e., **the equivalence of RA expressions**.

Heuristic Rules

```
Staff(sid, fname, lname, salary, position, branchNo)
Branch(branchNo, name, street, suburb, city)
```

- There are **many heuristic rules for transforming** RA expressions, utilized by **the query optimiser**, such as:

(1) $\sigma_{\varphi}(\sigma_{\psi}(R)) \equiv \sigma_{\varphi \wedge \psi}(R)$;

$\sigma_{branchNo='1'}(\sigma_{salary>60000}(Staff)) = \sigma_{branchNo='1' \wedge salary>60000}(Staff)$

(2) $\pi_X(\pi_Y(R)) \equiv \pi_X(R)$ if $X \subseteq Y$;

$\pi_{salary}(\pi_{branchNo,salary}(Staff)) = \pi_{salary}(Staff)$

(3) $\sigma_{\varphi}(R_1 \times R_2) \equiv R_1 \bowtie_{\varphi} R_2$

$$\sigma_{Staff.branchNo=Branch.branchNo}(Staff \times Branch) =$$
$$(Staff) \bowtie_{Staff.branchNo=Branch.branchNo} (Branch)$$

Heuristic Rules

```
Staff(sid, fname, lname, salary, position, branchNo)
Branch(branchNo, name, street, suburb, city)
```

(4) $\sigma_{\varphi_1}(R_1 \bowtie_{\varphi_2} R_2) \equiv R_2 \bowtie_{\varphi_1 \wedge \varphi_2} R_1$

$\sigma_{salary>60000}(Staff \bowtie_{Staff.branchNo=Branch.branchNo} (Branch)) =$

$(Staff) \bowtie_{Staff.branchNo=Branch.branchNo \wedge salary>60000} (Branch)$

(5) $\sigma_{\varphi}(R_1 \bowtie R_2) \equiv \sigma_{\varphi}(R_1) \bowtie R_2$, if φ contains only attributes in R_1

$\sigma_{salary>60000}(Staff \bowtie Branch) = \sigma_{salary>60000}(Staff) \bowtie Branch$

(6) $\sigma_{\varphi_1 \wedge \varphi_2}(R_1 \bowtie R_2) \equiv \sigma_{\varphi_1}(R_1) \bowtie \sigma_{\varphi_2}(R_2)$ if φ_1 contains only attributes in R_1 and φ_2 contains only attributes in R_2.

$\sigma_{salary>60000 \wedge city='Canberra'}(Staff \bowtie Branch) =$

$(\sigma_{salary>60000}(Staff)) \bowtie (\sigma_{city='Canberra'}(Branch))$

Heuristic Rules

```
Staff(sid, fname, lname, salary, position, branchNo)
Branch(branchNo, name, street, suburb, city)
```

(7) If the join condition involves only attributes in X, we have $\pi_X(R_1 \bowtie R_2) \equiv \pi_{X_1}(R_1) \bowtie \pi_{X_2}(R_2)$, where X_i contains attributes in both R_1 and R_2, and ones in both R_i and X, and

$$\pi_{branchNo,position,city}(Staff \bowtie Branch) =$$
$$\pi_{branchNo,position}(Staff) \bowtie (\pi_{branchNo,city}(Branch))$$

(8) If the join condition contains attributes not in X, we have $\pi_X(R_1 \bowtie R_2) \equiv \pi_X(\pi_{X_1}(R_1) \bowtie \pi_{X_2}(R_2))$, where X_i contains attributes in both in R_1 and R_2, and ones in both R_i and X

$$\pi_{position,city}(Staff \bowtie Branch) =$$
$$\pi_{position,city}(\pi_{branchNo,position}(Staff) \bowtie (\pi_{branchNo,city}(Branch)))$$

- ......

Push-down Selection – Example

- Given the relation schemas:

  PERSON(id, first_name, last_name, year_born)
  DIRECTOR(id, title, production_year)
  MOVIE_AWARD(title, production_year, award_name, year_of_award)

- **Query: List the first and last names of the directors who have directed a movie that has won an 'Oscar' movie award**

  $\pi_{first\_name, last\_name}(\sigma_{award\_name=\text{'Oscar'}}((\text{PERSON} \bowtie \text{DIRECTOR}) \bowtie \text{MOVIE\_AWARD}))$

- **Question**: Can we apply the following rule to optimise the query?

  $\sigma_{\varphi}(R_1 \bowtie R_2) \equiv \sigma_{\varphi}(R_1) \bowtie R_2$, if φ contains only attributes in R_1

Push-down Selection – Example

- Given the relation schemas:

  PERSON(id, first_name, last_name, year_born)
  DIRECTOR(id, title, production_year)
  MOVIE_AWARD(title, production_year, award_name, year_of_award)

- **Query: List the first and last names of the directors who have directed a movie that has won an 'Oscar' movie award**

  $\pi_{first\_name, last\_name}(\sigma_{award\_name=\text{'Oscar'}}((\text{PERSON} \bowtie \text{DIRECTOR}) \bowtie \text{MOVIE\_AWARD}))$

- We would have:

  $\pi_{first\_name, last\_name}((\text{PERSON} \bowtie \text{DIRECTOR}) \bowtie \sigma_{award\_name=\text{'Oscar'}}(\text{MOVIE\_AWARD}))$

Push-down Projection – Example

- Given the relation schemas:

  PERSON(id, first_name, last_name, year_born)

  DIRECTOR(id, title, production_year)

  MOVIE_AWARD(title, production_year, award_name, year_of_award)

- **Query: List the first and last names of the directors who have directed a movie that has won an 'Oscar' movie award**

  $\pi_{first\_name, last\_name}((\text{PERSON} \bowtie \text{DIRECTOR}) \bowtie \sigma_{award\_name=\text{'Oscar'}}(\text{MOVIE\_AWARD}))$

- **Question**: Can we apply the following rule to optimise the query?

$$\pi_X(R_1 \bowtie R_2) \equiv \pi_X(\pi_{X_1}(R_1) \bowtie \pi_{X_2}(R_2)),$$

where X_i contains attributes in both in R_1 and R_2, and ones in both R_i and X

Push-down Projection – Example

- Given the relation schemas:

  PERSON(id, first_name, last_name, year_born)

  DIRECTOR(id, title, production_year)

  MOVIE_AWARD(title, production_year, award_name, year_of_award)

- **Query: List the first and last names of the directors who have directed a movie that has won an 'Oscar' movie award**

  $$\pi_{first\_name, last\_name}((\text{PERSON} \bowtie \text{DIRECTOR}) \bowtie \sigma_{award\_name=\text{'Oscar'}}(\text{MOVIE\_AWARD}))$$

- we would have:

  $$\pi_{first\_name, last\_name}(\pi_{first\_name, last\_name, title, production\_year}(\text{PERSON} \bowtie \text{DIRECTOR}) \bowtie \pi_{title, production\_year}(\sigma_{award\_name=\text{'Oscar'}}(\text{MOVIE\_AWARD})))$$

A Common Query Pattern (Be Careful)

- A common query pattern is **join-select-project** involving three steps:

  (1) **join** all the relevant relations,

  (2) **select** the desired tuples, and

  (3) **project** on the required attributes.

- This query pattern can be expressed as an RA expression

$$\pi_{A_1,\ldots,A_n}(\sigma_{\varphi}(R_1 \times \cdots \times R_k)),$$

  or as an equivalent SQL statement

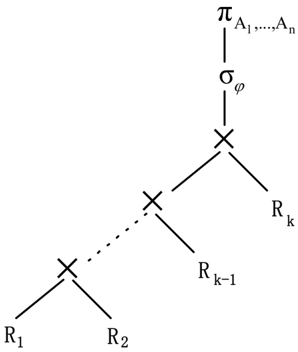
$$\texttt{SELECT DISTINCT } A_1, \ldots, A_n \texttt{ FROM } R_1, \ldots, R_k \texttt{ WHERE } \varphi;$$

- Queries falling into this pattern can be **very inefficient**, which may yield huge intermediate result for the joined relations.

A Common Query Pattern (Be Careful)

push-down selection and **push-down projection**.

$$\pi_{A_1,\ldots,A_n}(\sigma_\varphi(R_1 \times \cdots \times R_k)),$$



Re-ordering Joins - Example

- Given the relation schemas:

  PERSON(id, first_name, last_name, year_born)

  Suppose that it has **10000 tuples**.

  DIRECTOR(id, title, production_year) with
  [*title*, *production_year*] $\subseteq$ MOVIE_AWARD[*title*, *production_year*];
  [*id*] $\subseteq$ PERSON[*id*] and

  Suppose that it has **100 tuples**.

  MOVIE_AWARD(title, production_year, award_name, year_of_award)

  Suppose that it has **1000 tuples**.

- **Example:** Consider the following two RA queries. Which one is better?
  - PERSON $\bowtie$ MOVIE_AWARD $\bowtie$ DIRECTOR
  - PERSON $\bowtie$ DIRECTOR $\bowtie$ MOVIE_AWARD

Cost-based Query Optimisation

- A query optimiser does not depend solely on heuristic optimisation. It estimates and compares the costs of different plans.
- It estimates and compares the costs of executing a query using different execution strategies and chooses one with **the lowest cost estimate**.
- The query optimiser needs to **limit the number of execution strategies** to be considered for improving efficiency.

Summary

- In general, there are **many ways of executing a query** in a database.
- The user expects the result to be returned promptly, i.e., the query should be **processed as fast as possible**.
- But, the burden of optimising queries should not be put on the user's shoulder. **The DBMSs need to do the job!**
- Nonetheless, SQL is not a suitable query language in which queries can be optimised automatically.
- Instead, SQL queries are **transformed into their corresponding RA queries** and optimised subsequently.
- A major advantage of relational algebra is to **make alternative forms of a query easy to explore**.