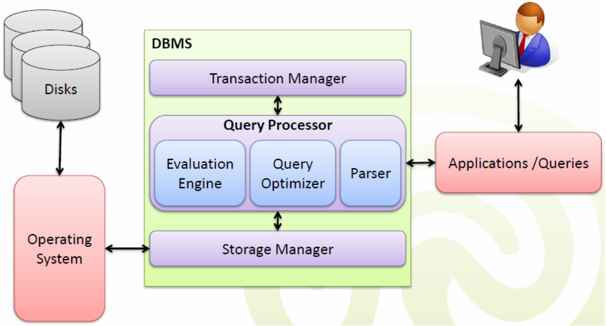
Query Processing

Query Processing – Overview

1. Users **submit** SQL queries to a DBMS.
2. The DBMS **processes and executes** them in a database.



- **Note:** SQL is a declarative language, so it is the task of DBMSs to decide how SQL queries should be executed.

Query Processing – Example

- **From:**

```
SELECT name FROM Person WHERE age<21;
```

- **To:**

name
Rickon
Bran

- **Questions:**
  - **How does a relational DBMS process this?**
  - **How can a relational DBMS process this efficiently?**

Query Processing – Example

```
SELECT name FROM Person WHERE age<21;
```

High-level language
(SQL)

$\Downarrow$

$\pi_{name}(\sigma_{age<21}(\mathtt{Person}))$

Low-level language
(Relational Algebra)

$\Downarrow$

$$\begin{array}{c} \pi_{\text{name}} \\ | \\ \sigma_{age<21} \\ | \\ \mathtt{Person} \end{array}$$

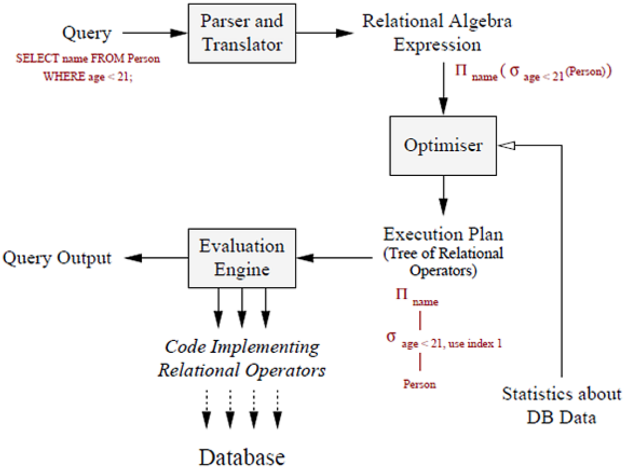
Execution plan
(Query tree)

$\Downarrow$

name
Rickon
Bran

Query result

Query Processing – Example

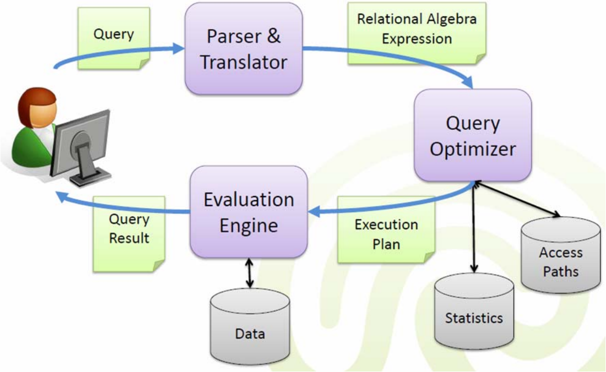


Query Processing Steps

- **Query parser and translator**
  1. Check the syntax of SQL queries
  2. Verify that the relations do exist
  3. Transform into relational algebra expressions
- **Query optimiser**
  1. Transform into the best possible execution plan
  2. Specify the implementation of each operator in the execution plan
- **Evaluation engine**
  1. Evaluate the query execution plan
  2. Return the result to the user

Query Processing – Parser

- The **parser** checks the syntax of the query:
  - Validation of table names, attributes, data types, access permission ...;
  - Either the query is executable or an error message is generated.



Query Processing – Parser

- Consider the relation schema:

  `Person(id:integer, name:string, age:integer, address:string)`

- **Note:** **System catalog** (also called **data dictionary**) is used at this stage, which contains the information about data managed by the DBMS.

  Example:

attr_name	rel_name	type	position
id	`Person`	integer	1
name	`Person`	string	2
age	`Person`	integer	3
address	`Person`	string	4
...	...	...	...

- **Question:** Can the following query be accepted by the parser?

```
SELECT fname, lname FROM Person WHERE address<21;
```

Query Processing – Parser

- Consider the relation schema:

  `Person(id:integer, name:string, age:integer, address:string)`

- **Question:** Can the following query be accepted by the parser?

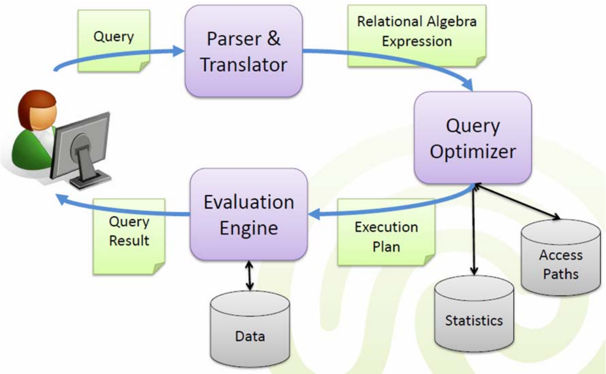
```
SELECT fname, lname FROM Person WHERE address<21;
```

- **Answer:** The query **would be rejected** because

  1. The attributes `fname` and `lname` are not defined;
  2. The attribute `address` is not comparable with 21.

Query Processing – Translator

- The **translator** translates queries into RA expressions (not necessarily equivalent due to duplicates):
  - A query is first decomposed into **query blocks**.
  - Each query block is translated into an RA expression.



Recall: RA and SQL Queries

- **RA operators**
  - **selection** σ_{φ}
  - **projection** $\pi_{A_1,\ldots,A_n}$
  - **Cartesian product** $R_1 \times R_2$
  - **join** $R_1 \bowtie_{\varphi} R_2$ and $R_1 \bowtie R_2$
  - **renaming** $\rho_{R(A_1,\ldots,A_n)}$
  - **union** $R_1 \cup R_2$
  - **intersection** $R_1 \cap R_2$
  - **difference** $R_1 - R_2$

- **SQL statement**

```
   SELECT attribute_list
     FROM table_list
   [WHERE condition]
[GROUP BY attribute_list
  [HAVING group_condition]]
[ORDER BY attribute_list];
```

$$\sigma_{\varphi}(R) \Leftrightarrow \texttt{SELECT * FROM } R \texttt{ WHERE } \varphi;$$

$$\pi_{A_1,\ldots,A_n}(R) \Leftrightarrow \texttt{SELECT DISTINCT } A_1,\ldots,A_n \texttt{ FROM } R;$$

$$R_1 \times R_2 \Leftrightarrow \texttt{SELECT DISTINCT * FROM } R_1, R_2;$$

$$\ldots$$

- Aggregate operations in SQL require extended RA expressions.

Recall: RA and SQL Queries

- Nested subqueries are decomposed into separate query blocks.
- **Example:**

```
SELECT Lname, Fname
  FROM EMPLOYEE
 WHERE Salary > (SELECT Salary
                   FROM EMPLOYEE
                  WHERE ssn=5);
```

Outer query block

```
SELECT Lname, Fname FROM EMPLOYEE WHERE
Salary > c
```

$\Downarrow$ translated

$\pi_{Lname,Fname}(\sigma_{Salary>c}(\text{EMPLOYEE}))$

Inner query block

```
(SELECT Salary FROM EMPLOYEE WHERE
ssn=5)
```

$\Downarrow$ translated

$\pi_{Salary}(\sigma_{ssn=5}(\text{EMPLOYEE}))$

Query Processing – Query Optimiser

1. Transform into the best possible execution plan

   There are different possible relational algebra expressions for a single query!

   (will be covered in this course)

2. Specify the implementation of each operator in the execution plan

   There are different possible implementations for a relational algebra operator!

   (will not be covered in this course)

Query Processing – Query Optimiser

- SQL queries only specify **what data to be retrieved** and **not how to retrieve data**.
- There are **many possible execution plans** for a SQL query.
- Query optimiser is responsible for identifying **an efficient execution plan**:
  1. enumerating alternative plans (typically, a subset of all possible plans);
  2. choosing the one with the least estimated cost.
- Query optimisation is one of the most important tasks of a relational DBMS. **A good DBMS must have a good query optimiser!**

Equivalent RA Expressions

- Suppose that we have:

```
Students(matNr, firstName, lastName, email)
Exams(matNr, crsNr, result, semester)
Courses(crsNr, title, unit)
```

```
SELECT lastName, result, title
  FROM Students, Exams, Courses
 WHERE Students.matNr=Exams.matNr AND
       Exams.crsNr=Courses.crsNr AND result≤1.3;
```

- **Question:**

  How many equivalent RA expressions for this SQL query can you find?

Equivalent RA Expressions

```
Students(matNr, firstName, lastName, email)
Exams(matNr, crsNr, result, semester)
Courses(crsNr, title, unit)
```

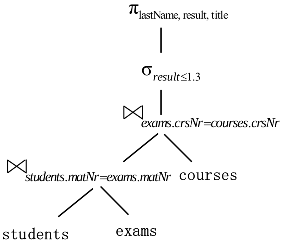
```
SELECT lastName, result, title
  FROM STUDENTS, EXAMS, COURSES
 WHERE STUDENTS.matNr=EXAMS.matNr AND
       EXAMS.crsNr=COURSES.crsNr AND result≤1.3;
```

- **Answer:**
  1. $\pi_{lastName,result,title}(\sigma_{result\leq 1.3}((\texttt{Students} \bowtie_{\texttt{Students}.matNr=\texttt{Exams}.matNr} \texttt{Exams}) \bowtie_{\sigma_{\texttt{Exams}.crsNr=\texttt{Courses}.crsNr}} \texttt{Courses}))$
  2. $\pi_{lastName,result,title}(\sigma_{result\leq 1.3}(\sigma_{\text{EXAMS}.crsNr=\texttt{Courses}.crsNr}(\sigma_{\texttt{Students.matNr=Exams.matNr}}(\texttt{Students} \times \texttt{Exams} \times \texttt{Courses}))))$
  3. $\pi_{lastName,result,title}((\texttt{Students} \bowtie_{\texttt{Students}.matNr=\texttt{Exams}.matNr} (\sigma_{result\leq 1.3}(\texttt{Exams}))) \bowtie_{\sigma_{\texttt{Exams}.crsNr=\texttt{Courses}.crsNr}} \texttt{Courses})$

Query Trees

- Each RA expression can be represented as a **query tree**:
  - **leaf nodes** represent the input relations;
  - **internal nodes** represent the intermediate result;
  - **the root node** represents the resulting relation.
- **Example:**

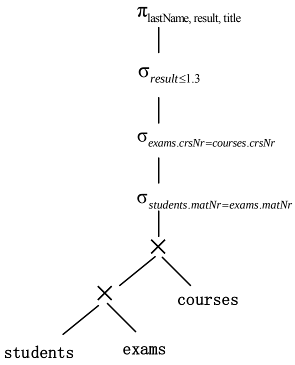
$$\pi_{lastName,result,title}(\sigma_{result\leq 1.3}((\texttt{Students} \bowtie_{\texttt{Students}.matNr=\texttt{Exams}.matNr} \texttt{Exams}) \bowtie_{\sigma_{\texttt{Exams}.crsNr=\texttt{Courses}.crsNr}} \texttt{Courses}))$$



Query Trees

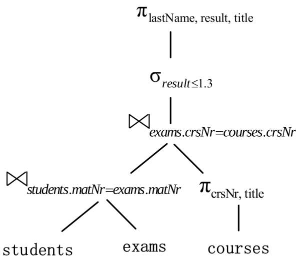
- **Exercise:** Can you draw the query tree for the following RA expression?

$$\pi_{lastName,result,title}(\sigma_{result \leq 1.3}(\sigma_{\texttt{Exams.crsNr=Courses.crsNr}}(\sigma_{\texttt{Students.matNr=Exams.matNr}}(\texttt{Students} \times \texttt{Exams} \times \texttt{Courses}))))$$



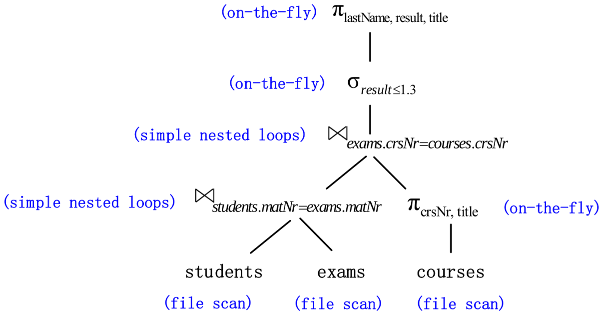
Query Trees

- For each query tree, computation proceeds **bottom-up**:
  - child nodes must be executed before their parent nodes;
  - but there can exist multiple methods of executing sibling nodes, e.g.,
    - process sequentially;
    - process in parallel.



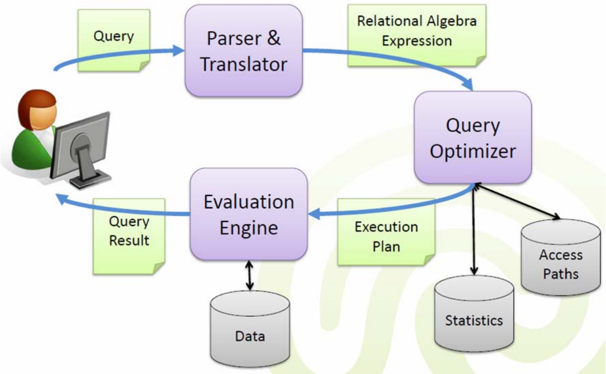
Execution Plan

- A **query execution plan** consists of an (extended) query tree with additional annotation at each node indicating:
  (1) the *access method* to use for each table, and
  (2) the *implementation method* for each RA operator.



Query Processing – Evaluation Engine

- The **evaluation engine** executes an execution plan, and returns the query answer to the user.



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:

  $\text{PERSON}(\text{id}, \text{first\_name}, \text{last\_name}, \text{year\_born})$
  $\text{DIRECTOR}(\text{id}, \text{title}, \text{production\_year})$
  $\text{MOVIE\_AWARD}(\text{title}, \text{production\_year}, \text{award\_name}, \text{year\_of\_award})$

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

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

- We would have:

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

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**

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

- **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

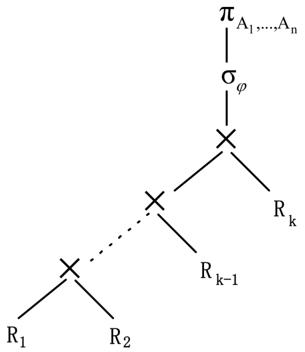
```
SELECT DISTINCT A1, ..., An FROM R1, ..., Rk WHERE φ;
```

- 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**.