Database Systems Architecture

Stijn Vansummeren
**General Course Information**

**Objective:**

To obtain insight into the internal operation and implementation of systems designed to manage and process large amounts of data ("database management systems").

- Storage management
- Query processing
- Transaction management
General Course Information

Examples of data management systems:

- Relational DBMSs
- NoSQL DBMS
- Graph databases
- Stream processing systems/Complex event processing systems

Focus on relational DBMS, with discussion on how the foundational ideas of relational DBMSs are modified in other systems
General Course Information

Why is this interesting?

- **Understand** how typical data management systems work
- **Predict** data management system behavior, **tune** its performance
- Many of the techniques studied transfer to settings other than data management systems
  (MMORPGs, Financial market analysis, . . .)

What this course is not:

- Introduction to databases
- Focused on particular DBMS (Oracle, IBM, . . .)
General Course Information

Organisation

- Combination of lectures; exercise sessions; guided self-study; and project work.
- Evaluation: pair project and written exam

Course material

- Course notes (available on website)

Contact information

- Email: stijn.vansummeren@ulb.ac.be
- Office: UB4.125
- Website: http://cs.ulb.ac.be/public/teaching/infoh417
Course Prerequisites

An introductory course on relational database systems
- Understanding of the Relational Algebra
- Understanding of SQL

Background on basic data structures and algorithms
- Search trees
- Hashing
- Analysis of algorithms: worst-case complexity and big-oh notation (e.g., $O(n^3)$)
- Basic knowledge of what it means to be NP-complete

Proficiency in Programming (Java or C/C++)
- Necessary for completing the project assignment
Query processing: overview

Query Compiler

SQL → Logical query plan → Logical plan optimization → Optimized logical query plan → Physical query plan → Execution Engine → Result

Translation

Physical Data Storage

Statistics and Metadata

Physical plan selection
Query processing: overview

SQL → Logical query plan → Optimized logical query plan → Physical query plan → Result

Translation

Logical plan optimization

Physical plan selection

"Intermediate code"

"Machine code"

Statistics and Metadata

Physical Data Storage
Translation of SQL into Relational Algebra
From SQL text to logical query plans
Translation of SQL into relational algebra: overview

We will adopt the following simplifying assumptions:

We will only show how to translate SQL-92 queries.

And we adopt a set-based semantics of SQL. (In contrast, real SQL is bag-based.)

What will we use as logical query plans?

The extended relational algebra (interpreted over sets).

Prerequisites

• SQL: see chapter 6 in TCB
• Extended relational algebra: chapter 5 in TCB
Refreshing the Relational Algebra

Relations are tables whose columns have names, called attributes.

\[
\begin{array}{cccc}
A & B & C & D \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 5 \\
3 & 4 & 5 & 6 \\
5 & 6 & 3 & 4
\end{array}
\]

The set of all attributes of a relation is called the schema of the relation.

The rows in a relation are called tuples.

A relation is set-based if it does not contain duplicate tuples. It is called bag-based otherwise.
Refreshing the Relational Algebra

Unless specified otherwise, we assume that relations are set-based.

Each Relational Algebra operator takes as input 1 or more relations, and produces a new relation.
Refresh the Relational Algebra

Union (set-based)

\[
\begin{array}{c|c}
A & B \\
1 & 2 \\
3 & 4 \\
5 & 6 \\
\end{array} \cup \begin{array}{c|c}
A & B \\
3 & 4 \\
1 & 5 \\
\end{array} = \begin{array}{c|c}
A & B \\
1 & 2 \\
3 & 4 \\
5 & 6 \\
\end{array}
\]

Input relations must have the same schema (same set of attributes)
## Refreshing the Relational Algebra

### Intersection (set-based)

Input relations must have same set of attributes

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

\[ A \cap B = \begin{array}{c|c|} 3 & 4 \\ \hline 1 & 5 \end{array} \]
Refreshing the Relational Algebra

Difference (set-based)

\[
\begin{array}{c|c}
A & B \\
\hline
1 & 2 \\
3 & 4 \\
5 & 6 \\
\end{array}
\]

\[
\begin{array}{c|c}
A & B \\
\hline
3 & 4 \\
1 & 5 \\
\end{array}
\]

\[
= \begin{array}{c|c}
A & B \\
\hline
1 & 2 \\
5 & 6 \\
\end{array}
\]

Input relations **must** have same set of attributes
Refreshing the Relational Algebra

Selection

\[ \sigma_{A \geq 3} \left( \begin{array}{c|c} A & B \\ \hline 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{array} \right) = \left( \begin{array}{c|c} A & B \\ \hline 3 & 4 \\ 5 & 6 \end{array} \right) \]
Refreshing the Relational Algebra

Projection (set-based)

\[
\pi_{A,C} \begin{pmatrix}
A & B & C & D \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 5 \\
3 & 4 & 5 & 6 \\
5 & 6 & 3 & 4 \\
\end{pmatrix} = \begin{pmatrix}
A & C \\
1 & 3 \\
3 & 5 \\
5 & 3 \\
\end{pmatrix}
\]
Refreshing the Relational Algebra

Cartesian product

\[
\begin{array}{c|c}
A & B \\
\hline
1 & 2 \\
3 & 4 \\
\end{array} \quad \times \quad \begin{array}{c|c}
C & D \\
\hline
2 & 6 \\
3 & 7 \\
4 & 9 \\
\end{array} = \begin{array}{c|c|c|c}
A & B & C & D \\
\hline
1 & 2 & 2 & 6 \\
1 & 2 & 3 & 7 \\
1 & 2 & 4 & 9 \\
3 & 4 & 2 & 6 \\
3 & 4 & 3 & 7 \\
3 & 4 & 4 & 9 \\
\end{array}
\]

Input relations must have same disjoint schema (set of attributes)
Refresh the Relational Algebra

Natural Join

\[
\begin{array}{c|c}
A & B \\
1 & 2 \\
3 & 4 \\
\end{array} \quad \times \quad \begin{array}{c|c}
B & D \\
2 & 6 \\
3 & 7 \\
4 & 9 \\
\end{array} = \begin{array}{c|c|c}
A & B & D \\
1 & 2 & 6 \\
3 & 4 & 9 \\
\end{array}
\]
Refreshing the Relational Algebra

Natural Join

\[
\begin{array}{c|c}
A & B \\
\hline
1 & 2 \\
3 & 4 \\
\end{array} \quad \times \quad \begin{array}{c|c}
C & D \\
\hline
2 & 6 \\
3 & 7 \\
\end{array} \quad = \quad \begin{array}{c|c|c|c}
A & B & C & D \\
\hline
1 & 2 & 2 & 6 \\
1 & 2 & 3 & 7 \\
1 & 2 & 4 & 9 \\
3 & 4 & 2 & 6 \\
3 & 4 & 3 & 7 \\
3 & 4 & 4 & 9 \\
\end{array}
\]
Refresh the Relational Algebra

Theta Join

\[
\begin{array}{c|c} 
A & B \\
1 & 2 \\
3 & 4 \\
\end{array} \quad \bowtie_{B=C} \quad \begin{array}{c|c} 
C & D \\
2 & 6 \\
3 & 7 \\
4 & 9 \\
\end{array} = \begin{array}{c|c|c|c} 
A & B & C & D \\
1 & 2 & 2 & 6 \\
3 & 4 & 4 & 9 \\
\end{array}
\]
Refreshing the Relational Algebra

Renaming

\[ \rho_T \left( \begin{array}{c|c} A & B \\ \hline 1 & 2 \\ 3 & 4 \end{array} \right) = \begin{array}{c|c} T.A & T.B \\ \hline 1 & 2 \\ 3 & 4 \end{array} \]

Renaming specifies that the input relation (and its attributes) should be given a new name.
Refreshing the Relational Algebra

Relational algebra expressions:

- Built using relation variables
- And relational algebra operators

\[ \sigma_{\text{length} \geq 100}(\text{Movie}) \bowtie_{\text{title} = \text{movietitle}} \text{StarsIn} \]
Refreshing the Relational Algebra

The extended relational algebra

Adds some operators to the algebra (sorting, grouping, ...) and extends others (projection).

Grouping:

\[ \gamma_{A, \text{min}(B) \rightarrow D} \left( \begin{array}{ccc}
A & B & C \\
1 & 2 & a \\
1 & 3 & b \\
2 & 3 & c \\
2 & 4 & a \\
2 & 5 & a \\
\end{array} \right) = \left( \begin{array}{c}
A \\
1 \\
1 \\
2 \\
2 \\
\end{array} \right) \begin{array}{c}
D \\
2 \\
3 \\
\end{array} \]
Refreshing the Relational Algebra

The extended relational algebra

Adds some operators to the algebra (sorting, grouping, . . . ) and extends others (projection).

Extend projection to allow renaming of attributes:

\[
\pi_{A,C \rightarrow D} \left(\begin{array}{cccc}
A & B & C & D \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 5 \\
3 & 4 & 5 & 6 \\
5 & 6 & 3 & 4 \\
\end{array}\right) = \begin{array}{cc}
A & D \\
1 & 3 \\
3 & 5 \\
5 & 3 \\
\end{array}
\]
Refreshing the Relational Algebra

On the difference between sets and bags

• Historically speaking, relations are defined to be sets of tuples: duplicate tuples cannot occur in a relation.

• In practical systems, however, it is more efficient to allow duplicates to occur in relations, and only remove duplicates when requested. In this case relations are bags.

Union (bag-based)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>( \cup )</th>
<th>A</th>
<th>B</th>
<th>=</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td>3</td>
<td>4</td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td>1</td>
<td>5</td>
<td></td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
<td>5</td>
<td>6</td>
<td></td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Refreshing the Relational Algebra

On the difference between sets and bags

- Historically speaking, relations are defined to be sets of tuples: duplicate tuples cannot occur in a relation.
- In practical systems, however, it is more efficient to allow duplicates to occur in relations, and only remove duplicates when requested. In this case relations are bags.

Intersection (bag-based)

\[
\begin{array}{c|c} 
A & B \\
\hline 
1 & 2 \\
3 & 4 \\
1 & 2 \\
1 & 2 \\
\end{array}
\cap
\begin{array}{c|c} 
A & B \\
\hline 
1 & 2 \\
3 & 4 \\
3 & 4 \\
5 & 6 \\
\end{array} = 
\begin{array}{c|c} 
A & B \\
\hline 
1 & 2 \\
3 & 4 \\
\end{array}
\]
Refresh the Relational Algebra

On the difference between sets and bags

- Historically speaking, relations are defined to be sets of tuples: duplicate tuples cannot occur in a relation.
- In practical systems, however, it is more efficient to allow duplicates to occur in relations, and only remove duplicates when requested. In this case relations are bags.

Difference (bag-based)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

- \( A \setminus B \) - \( A \) \[ \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Refreshing the Relational Algebra

On the difference between sets and bags

- Historically speaking, relations are defined to be sets of tuples: duplicate tuples cannot occur in a relation.
- In practical systems, however, it is more efficient to allow duplicates to occur in relations, and only remove duplicates when requested. In this case relations are bags.

Projection (bag-based)

\[
\pi_{A,C} \left( \begin{array}{cccc}
A & B & C & D \\
1 & 2 & 3 & 4 \\
1 & 2 & 3 & 5 \\
3 & 4 & 5 & 6 \\
5 & 6 & 3 & 4 \\
\end{array} \right) = \begin{array}{c}
A \\
1 \\
1 \\
3 \\
3 \\
5 \\
\end{array}, \begin{array}{c}
C \\
3 \\
5 \\
3 \\
\end{array}
\]

Refresh the Relational Algebra

**On the difference between sets and bags**

- Historically speaking, relations are defined to be *sets* of tuples: duplicate tuples cannot occur in a relation.
- In practical systems, however, it is more efficient to allow duplicates to occur in relations, and only remove duplicates when requested. In this case relations are *bags*.

The other operators are straightforwardly extended to bags: simply do the same operation, taking into account duplicates.
Translation of SQL into relational algebra: overview

We will adopt the following simplifying assumptions:

- We will only show how to translate SQL-92 queries
- And we adopt a set-based semantics of SQL. (In contrast, real SQL is bag-based.)

What will we use as logical query plans?

- The extended relational algebra (interpreted over sets).

Prerequisites

- SQL: see chapter 6 in TCB
- Extended relational algebra: chapter 5 in TCB
Translation of SQL into the relational algebra

In the examples that follow, we will use the following database:

- Movie(title: string, year: int, length: int, genre: string, studioName: string, producerC#: int)
- MovieStar(name: string, address: string, gender: char, birthdate: date)
- StarsIn(movieTitle: string, movieYear: string, starName: string)
- MovieExec(name: string, address: string, CERT#: int, netWorth: int)
- Studio(name: string, address: string, presC#: int)
Translation of SQL into the relational algebra

Select-from-where statements without subqueries

SQL:
SELECT movieTitle
FROM StarsIn, MovieStar M
WHERE starName = M.name AND M.birthdate = 1960

Algebra: ???
Translation of SQL into the relational algebra

Select-from-where statements without subqueries

SQL: 
SELECT movieTitle 
FROM StarsIn, MovieStar M 
WHERE starName = M.name AND M.birthdate = 1960 

Algebra: \( \pi_{\text{movieTitle}} \sigma_{\text{starName}=M.\text{name} \land M.\text{birthdate}=1960} (\text{StarsIn} \times \rho_M(\text{MovieStar})) \)
Translation of SQL into the relational algebra

**Select statements in general contain subqueries**

```sql
SELECT movieTitle FROM StarsIn
WHERE starName IN (SELECT name
                   FROM MovieStar
                   WHERE birthdate=1960)
```

**Subqueries in the where-clause**

Occur through the operators =, <, >, <=, >=, <>; through the quantifiers ANY, or ALL; or through the operators EXISTS and IN and their negations NOT EXISTS and NOT IN.
Translation of SQL into the relational algebra

We can always normalize subqueries to use only EXISTS and NOT EXISTS

SELECT movieTitle FROM StarsIn
WHERE starName IN (SELECT name
  FROM MovieStar
  WHERE birthdate=1960)

⇒ SELECT movieTitle FROM StarsIn
  WHERE EXISTS (SELECT name
    FROM MovieStar
    WHERE birthdate=1960 AND name=starName)
Translation of SQL into the relational algebra

We can always normalize subqueries to use only \textbf{EXISTS and NOT EXISTS}

\begin{verbatim}
SELECT name FROM MovieExec
WHERE netWorth >= ALL (SELECT E.netWorth
FROM MovieExec E)
\end{verbatim}

⇒ \begin{verbatim}
SELECT name FROM MovieExec
WHERE NOT EXISTS(SELECT E.netWorth
FROM MovieExec E
WHERE netWorth < E.netWorth)
\end{verbatim}
Translation of SQL into the relational algebra

We can always normalize subqueries to use only `EXISTS` and `NOT EXISTS`

```sql
SELECT C FROM S
WHERE C IN (SELECT SUM(B) FROM R
            GROUP BY A)
⇒ ???
```
Translation of SQL into the relational algebra

We can always normalize subqueries to use only \textit{EXISTS} and \textit{NOT EXISTS}

\begin{verbatim}
SELECT C FROM S
WHERE C IN (SELECT SUM(B) FROM R
              GROUP BY A)
\end{verbatim}

$\Rightarrow$ \begin{verbatim}
SELECT C FROM S
WHERE EXISTS (SELECT SUM(B) FROM R
              GROUP BY A
              HAVING SUM(B) = C)
\end{verbatim}
Translation of SQL into the relational algebra

Translating subqueries - First step: normalization

• Before translating a query we first normalize it such that all of the subqueries that occur in a WHERE condition are of the form EXISTS or NOT EXISTS.

• We may hence assume without loss of generality in what follows that all subqueries in a WHERE condition are of the form EXISTS or NOT EXISTS.
Translation of SQL into the relational algebra

Correlated subqueries

A subquery can refer to attributes of relations that are introduced in an outer query.

SELECT movieTitle
FROM StarsIn
WHERE EXISTS (SELECT name
              FROM MovieStar
              WHERE birthdate=1960 AND name=starName)

Definition

• We call such subqueries correlated subqueries.
• The “outer” relations from which the correlated subquery uses some attributes are called the context relations of the subquery.
• The set of all attributes of all context relations of a subquery are called the parameters of the subquery.
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND EXISTS (SELECT name
            FROM MovieStar
            WHERE birthdate=1960 AND name= S.starName)
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND EXISTS (SELECT name
             FROM MovieStar
             WHERE birthdate=1960 AND name= S.starName)

1. We first translate the EXISTS subquery.

$$\pi_{\text{name}} \sigma_{\text{birthdate}=1960} (\text{MovieStar})$$

AND $$\land \text{name} = \text{S.starName}$$
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

```sql
SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND EXISTS (SELECT name
               FROM MovieStar
               WHERE birthdate=1960 AND name= S.starName)
```

1. We first translate the EXISTS subquery.

\[ \pi_{\text{name}} \sigma_{\text{birthdate}=1960 \land \text{name}=\text{S.starName}} (\text{MovieStar}) \]

Since we are translating a correlated subquery, however, we need to add the context relations and parameters for this translation to make sense.

\[ \pi_{\text{S.movieTitle, S.movieYear, S.starName, name}} \sigma_{\text{birthdate}=1960 \land \text{name}=\text{S.starName}} (\text{MovieStar} \times \rho_S(\text{StarsIn})) \]
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

```
SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND EXISTS (SELECT name
            FROM MovieStar
            WHERE birthdate=1960 AND name= S.starName)
```

2. Next, we translate the FROM clause of the outer query. This gives us:

\[ \rho_S(\text{StarsIn}) \times \rho_M(\text{Movie}) \]
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

```sql
SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND EXISTS (SELECT name
            FROM MovieStar
            WHERE birthdate=1960 AND name= S.starName)
```

3. We “synchronize” these subresults by means of a join. From the subquery we only need to retain the parameter attributes.

\[
(\rho_S(\text{StarsIn}) \times \rho_M(\text{Movie})) \Join
\]

\[
\pi_{S.movieTitle,S.movieYear,S.starName} \sigma_{\text{birthdate}=1960 \land \text{name}= S.starName}
(MovieStar \times \rho_S(\text{StarsIn}))
\]
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

```sql
SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND EXISTS (SELECT name
FROM MovieStar
WHERE birthdate=1960 AND name= S.starName)
```

4. We can simplify this by omitting the first \( \rho_S(\text{StarsIn}) \)

\[
\rho_M(\text{Movie}) \Join
\pi_{S\text{.movieTitle}, S\text{.movieYear}, S\text{.starName}} \sigma_{\text{birthdate}=1960 \land \text{name}= S\text{.starName}}
\left( \text{MovieStar} \times \rho_S(\text{StarsIn}) \right)
\]

\[\text{Translation of SQL into the relational algebra}\]

\[\text{Translation of correlated select-from-where subqueries}\]

\[
\text{SELECT S.movieTitle, M.studioName}
\text{FROM StarsIn S, Movie M}
\text{WHERE S.movieYear >= 2000}
\text{AND S.movieTitle = M.title}
\text{AND EXISTS (SELECT name}
\text{FROM MovieStar}
\text{WHERE birthdate=1960 AND name= S.starName)}
\]

4. We can simplify this by omitting the first \( \rho_S(\text{StarsIn}) \)

\[
\rho_M(\text{Movie}) \Join
\pi_{S\text{.movieTitle}, S\text{.movieYear}, S\text{.starName}} \sigma_{\text{birthdate}=1960 \land \text{name}= S\text{.starName}}
\left( \text{MovieStar} \times \rho_S(\text{StarsIn}) \right)
\]
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND EXISTS (SELECT name
            FROM MovieStar
            WHERE birthdate=1960 AND name= S.starName)

5. Finally, we translate the remaining subquery-free conditions in the WHERE clause, as well as the SELECT list

\[ \pi_{S.movieTitle, M.studioName} \sigma\langle S.movieYear \geq 2000 \land S.movieTitle = M.title \rangle
(\rho_M \langle \text{Movie} \rangle \Join \pi_{S.movieTitle, S.movieYear, S.starName}
\sigma\langle \text{birthdate} = 1960 \land name = S.starName \rangle
\langle \text{MovieStar} \times \rho_S \langle \text{StarsIn} \rangle \rangle) \]
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND NOT EXISTS (SELECT name
                FROM MovieStar
                WHERE birthdate=1960 AND name= S.starName)
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND NOT EXISTS (SELECT name
FROM MovieStar
WHERE birthdate=1960 AND name= S.starName)

1. We first translate the NOT EXISTS subquery.

\[ \pi_{name} \sigma_{\text{birthdate}=1960} (\text{MovieStar}) \land \text{name}=\text{S.starName} \]
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND NOT EXISTS (SELECT name
                 FROM MovieStar
                 WHERE birthdate=1960 AND name= S.starName)

1. We first translate the NOT EXISTS subquery.

\[ \pi_{\text{name}} \sigma_{\text{birthdate}=1960 \land \text{name}=\text{S.starName}} (\text{MovieStar}) \]

Since we are translating a correlated subquery, however, we need to add the context relations and parameters for this translation to make sense.

\[ \pi_{\text{S.movieTitle}, \text{S.movieYear}, \text{S.starName}, \text{name}} \sigma_{\text{birthdate}=1960 \land \text{name}=\text{S.starName}} (\text{MovieStar} \times \rho_{S}(\text{StarsIn})) \]
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND NOT EXISTS (SELECT name
FROM MovieStar
WHERE birthdate=1960 AND name= S.starName)

2. Next, we translate the FROM clause of the outer query. This gives us:

\[ \rho_S(\text{StarsIn}) \times \rho_M(\text{Movie}) \]
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND NOT EXISTS (SELECT name
FROM MovieStar
WHERE birthdate=1960 AND name= S.starName)

3. We then “synchronize” these subresults by means of an antijoin. From the subquery we only need to retain the parameter attributes.

\[(\rho_S(\text{StarsIn}) \times \rho_M(\text{Movie})) \bowtie \]

\[\pi_{S.movieTitle,S.movieYear,S.starName} \sigma_{\text{birthdate}=1960} \land \text{name}=S.starName (\text{MovieStar} \times \rho_S(\text{StarsIn}))\]

Here, the antijoin \( R \bowtie S \equiv R - (R \bowtie S) \).

Simplification is not possible: we cannot remove the first \( \rho_S(\text{StarsIn}) \).
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

SELECT S.movieTitle, M.studioName
FROM StarsIn S, Movie M
WHERE S.movieYear >= 2000
AND S.movieTitle = M.title
AND NOT EXISTS (SELECT name
FROM MovieStar
WHERE birthdate=1960 AND name= S.starName)

4. Finally, we translate the remaining subquery-free conditions in the WHERE clause, as well as the SELECT list

\[ \pi_{S.movieTitle,M.studioName} \sigma_{S.movieYear >= 2000 \land S.movieTitle = M.title} \\
\left( (\rho_S(StarsIn) \times \rho_M(Movie)) \bowtie \pi_{S.movieTitle,S.movieYear,S.starName} \\
\sigma \text{ birthdate}=1960 \land \text{name}=S.starName \\
(MovieStar \times \rho_S(StarsIn)) \right) \]
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

In the previous examples we have only considered queries of the following form:

\[
\text{SELECT Select-list FROM From-list} \\
\text{WHERE } \psi \text{ AND EXISTS}(Q) \text{ AND } \cdots \text{ AND NOT EXISTS}(P) \text{ AND } \cdots
\]

How do we treat the following?

\[
\text{SELECT Select-list FROM From-list} \\
\text{WHERE } A=B \text{ AND NOT(EXISTS}(Q) \text{ AND } C<6)
\]
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

In the previous examples we have only considered queries of the following form:

\[
\text{SELECT Select-list FROM From-list} \\
\text{WHERE } \psi \text{ AND EXISTS}(Q) \text{ AND } \cdots \text{ AND NOT EXISTS}(P) \text{ AND } \cdots
\]

How do we treat the following?

\[
\text{SELECT Select-list FROM From-list} \\
\text{WHERE } A=B \text{ AND NOT EXISTS}(Q) \text{ AND C}<6
\]

1. We first transform the condition into disjunctive normal form:

\[
\text{SELECT Select-list FROM From-list} \\
\text{WHERE } (A=B \text{ AND NOT EXISTS}(Q)) \text{ OR } (A=B \text{ AND C}>6)
\]
Translation of SQL into the relational algebra

Translation of correlated select-from-where subqueries

In the previous examples we have only considered queries of the following form:

\[
\text{SELECT Select-list FROM From-list} \\
\text{WHERE } \psi \text{ AND EXISTS}(Q) \text{ AND } \cdots \text{ AND NOT EXISTS}(P) \text{ AND } \cdots
\]

How do we treat the following?

\[
\text{SELECT Select-list FROM From-list} \\
\text{WHERE } A=B \text{ AND NOT EXISTS}(Q) \text{ AND } C<6
\]

2. We then distribute the OR

\[
(\text{SELECT Select-list FROM From-list} \\
\text{WHERE } (A=B \text{ AND NOT EXISTS}(Q)))
\text{ UNION }
(\text{SELECT Select-list FROM From-list} \\
\text{WHERE } (A=B \text{ AND } C\geq 6))
\]
Translation of SQL into the relational algebra

Union, intersection, and difference

SQL: (SELECT * FROM R R1) INTERSECT (SELECT * FROM R R2)

Algebra: $\rho_{R_1}(R) \cap \rho_{R_2}(R)$

SQL: (SELECT * FROM R R1) UNION (SELECT * FROM R R2)

Algebra: $\rho_{R_1}(R) \cup \rho_{R_2}(R)$

SQL: (SELECT * FROM R R1) EXCEPT (SELECT * FROM R R2)

Algebra: $\rho_{R_1}(R) - \rho_{R_2}(R)$
Translation of SQL into the relational algebra

Union, intersection, and difference in subqueries

Consider the relations \( R(A, B) \) and \( S(C) \).

\[
\begin{align*}
\text{SELECT} & \quad S1.C, S2.C \\
\text{FROM} & \quad S \ S1, S \ S2 \\
\text{WHERE} & \quad \exists ( \\
& \quad (\text{SELECT} \ R1.A, R1.B \ \text{FROM} \ R \ R1 \\
& \quad \quad \text{WHERE} \ A = S1.C \ \text{AND} \ B = S2.C) \\
& \quad \quad \text{UNION} \\
& \quad (\text{SELECT} \ R2.A, R2.B \ \text{FROM} \ R \ R2 \\
& \quad \quad \text{WHERE} \ B = S1.C) \\
& \quad )
\end{align*}
\]

In this case we translate the subquery as follows:

\[
\begin{align*}
\pi_{S1.C,S2.C,R1.A\rightarrow A,R1.B\rightarrow B} \sigma_{A=S1.C \land B=S2.C} (\rho_{R1}(R) \times \rho_{S1}(S) \times \rho_{S2}(S)) \\
\cup \pi_{S1.C,S2.C,R2.A\rightarrow A,R2.B\rightarrow B} \sigma_{B=S1.C} (\rho_{R2}(R) \times \rho_{S1}(S) \times \rho_{S2}(S))
\end{align*}
\]
Translation of SQL into the relational algebra

Join-expressions

SQL: \((\text{SELECT} \ast \text{FROM} \ R \ \text{R1}) \ \text{CROSS JOIN} \ (\text{SELECT} \ast \text{FROM} \ R \ \text{R2})\)

Algebra: \(\rho_{R_1}(R) \times \rho_{R_2}(R)\)

SQL: \((\text{SELECT} \ast \text{FROM} \ R \ \text{R1}) \ \text{JOIN} \ (\text{SELECT} \ast \text{FROM} \ R \ \text{R2}) \ \text{ON} \ \text{R1.A} = \text{R2.B}\)

Algebra: \(\rho_{R_1}(R) \ \Join_{R_1.A=R_2.B} \rho_{R_2}(R)\)
Translation of SQL into the relational algebra

Join-expressions in subqueries

Consider the relations \( R(A, B) \) and \( S(C) \).

\[
\begin{align*}
\text{SELECT } & S1.C, S2.C \\
\text{FROM } & S S1, S S2 \\
\text{WHERE } & \exists ( \\
& ( \text{SELECT } R1.A, R1.B \text{ FROM } R R1 \\
& \quad \text{WHERE } A = S1.C \text{ AND } B = S2.C) \\
& \quad \text{CROSS JOIN} \\
& ( \text{SELECT } R2.A, R2.B \text{ FROM } R R2 \\
& \quad \text{WHERE } B = S1.C) \\
) \\
\end{align*}
\]

In this case we translate the subquery as follows:

\[
\begin{align*}
\pi_{S1.C,S2.C,R1.A,R1.B} \sigma_{A=S1.C \wedge B=S2.C} ( \rho_{R1}(R) \times \rho_{S1}(S) \times \rho_{S2}(S)) \\
\Join \pi_{S1.C,R2.A,R2.B} \sigma_{B=S1.C} ( \rho_{R2}(R) \times \rho_{S1}(S))
\end{align*}
\]
Translation of SQL into the relational algebra

**GROUP BY and HAVING**

**SQL:**
```
SELECT name, SUM(length)
FROM MovieExec, Movie
WHERE cert# = producerC#
GROUP BY name
HAVING MIN(year) < 1930
```

**Algebra:**
```
π_{name, SUM(length)} \sigma_{MIN(year)<1930} \gamma_{name, MIN(year), SUM(length)}
\sigma_{cert#=producerC#}(MovieExec \times Movie)
```
Translation of SQL into the relational algebra

Subqueries in the From-list

SQL: 
SELECT movieTitle
FROM StarsIn, (SELECT name FROM MovieStar
    WHERE birthdate = 1960) M
WHERE starName = M.name

Algebra:
$$\pi_{\text{movieTitle}} \sigma_{\text{starName} = \text{M.name}}(\text{StarsIn} \times \rho_{\text{M}} \pi_{\text{name}} \sigma_{\text{birthdate} = 1960}(\text{MovieStar}))$$
Translation of SQL into the relational algebra

Lateral subqueries in SQL-99

```
SELECT S.movieTitle
FROM (SELECT name FROM MovieStar
      WHERE birthdate = 1960) M,
      LATERAL
      (SELECT movieTitle
       FROM StarsIn
       WHERE starName = M.name) S
```

1. We first translate the first subquery

\[ E_1 = \pi_{\text{name}} \sigma_{\text{birthdate}=1960}(\text{MovieStar}). \]

2. We then translate the second subquery, which has \( E_1 \) as context relation:

\[ E_2 = \rho_S \pi_{\text{name},\text{movieTitle}} \sigma_{\text{starName}=\text{M.name}}(\text{StarsIn} \times E_1). \]

3. Finally, we translate the whole FROM-clause by means of a join due to the correlation:

\[ \pi_{\text{movieTitle}}(E_1 \Join E_2). \]
Translation of SQL into the relational algebra

Lateral subqueries in SQL-99

```sql
SELECT S.movieTitle
FROM (SELECT name FROM MovieStar
      WHERE birthdate = 1960) M,
      LATERAL
      (SELECT movieTitle
       FROM StarsIn
       FROM StarsIn
       WHERE starName = M.name) S
```

4. In this example, however, all relevant tuples of $E_1$ are already contained in the result of $E_2$, and we can hence simplify:

$$\pi_{\text{movieTitle}}(E_2).$$
Subqueries in the select-list

Consider again the relations $R(A, B)$ and $S(C)$, and assume that $A$ is a key for $R$. The following query is then permitted:

$$\text{SELECT } C, (\text{SELECT } B \text{ FROM } R \text{ WHERE } A=C) \text{ FROM } S$$

Such queries can be rewritten as queries with LATERAL subqueries in the from-list:

$$\text{SELECT } C, T.B \text{ FROM } (\text{SELECT } C \text{ FROM } S), \text{ LATERAL} \text{ (SELECT } B \text{ FROM } R \text{ WHERE } A=C) T$$

We can hence first rewrite them in LATERAL form, and subsequently translate the rewritten query into the relational algebra.