Database Systems Architecture

Stijn Vansummeren

General Course Information

Objective:

To obtain insight into the internal operation and implementation of database systems.

- Storage management
- Query processing
- Transaction management

Organisation

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

General Course Information

Course material

- Database Systems: The Complete Book (H. Garcia-Molina, J. D. Ullman, and J. Widom) second edition
- 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

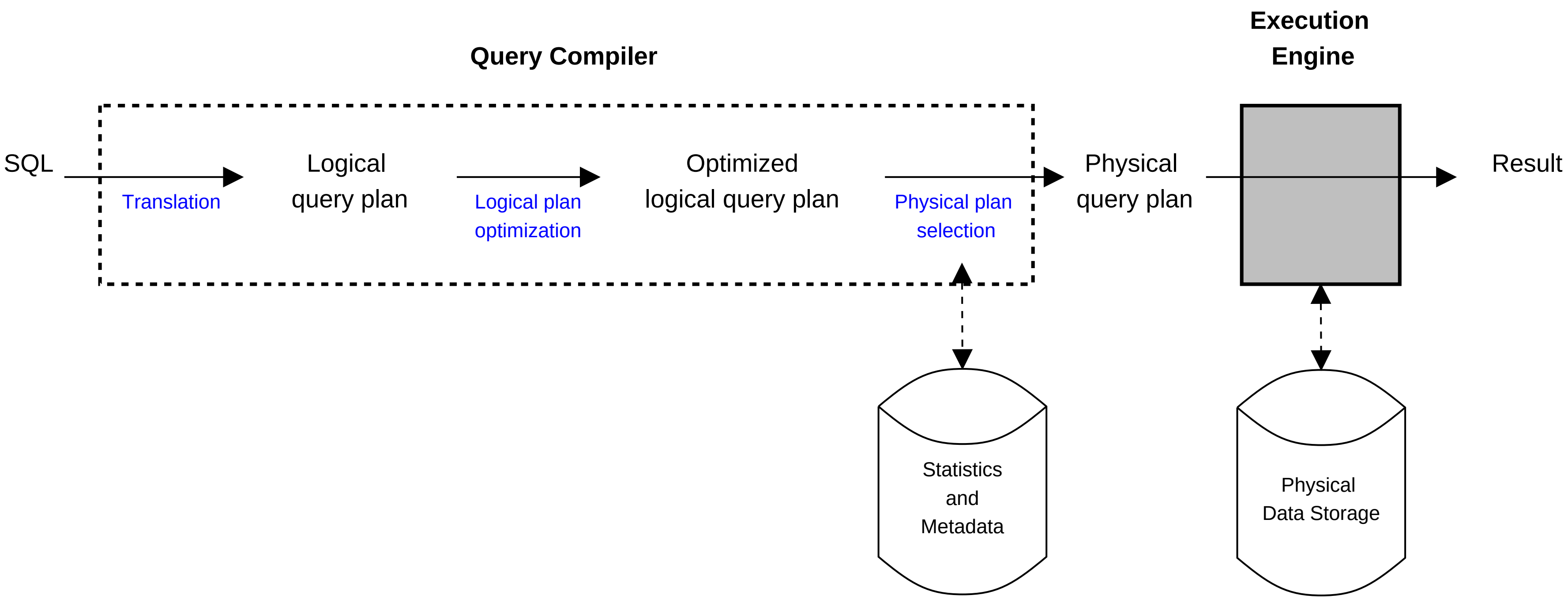
Background on basic data structures and algorithms

- Search trees
- Hashing
- Analysis of algorithms: worst-case complexity and big-oh notation (e.g., $O(n^3)$)

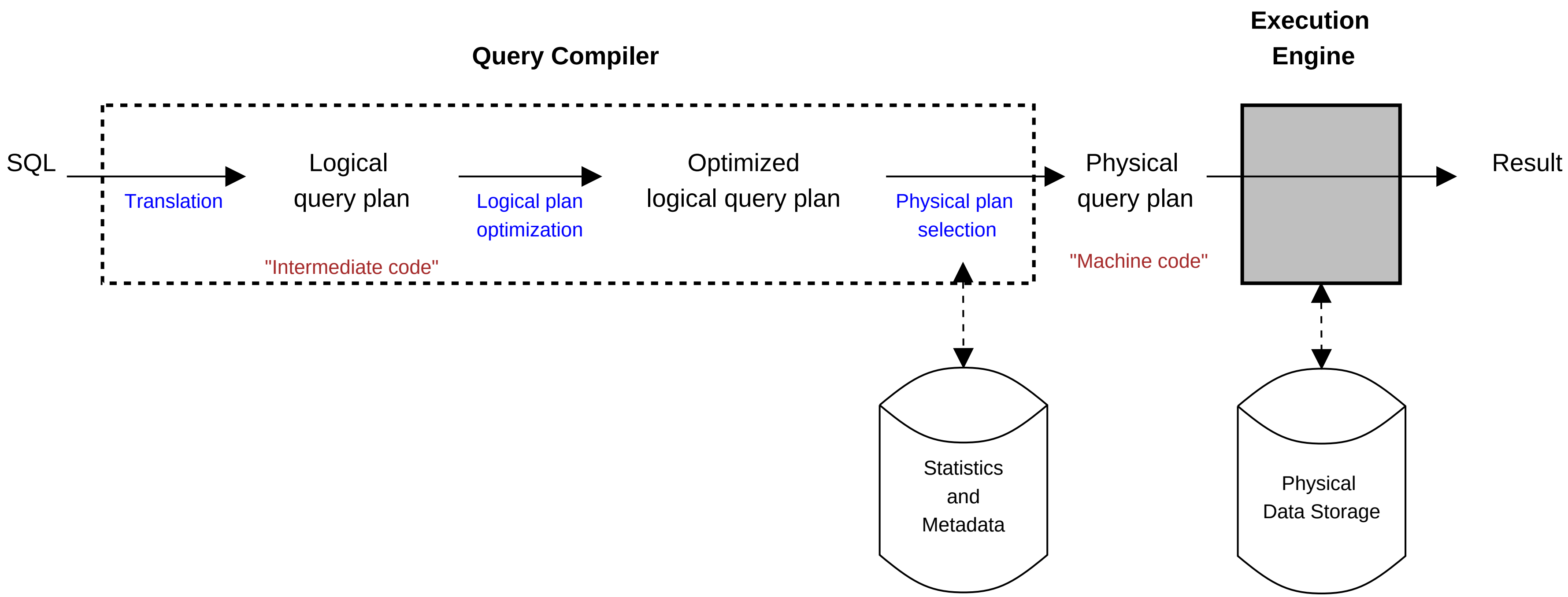
Proficiency in Programming (Java)

- Necessary for completing the project assignment

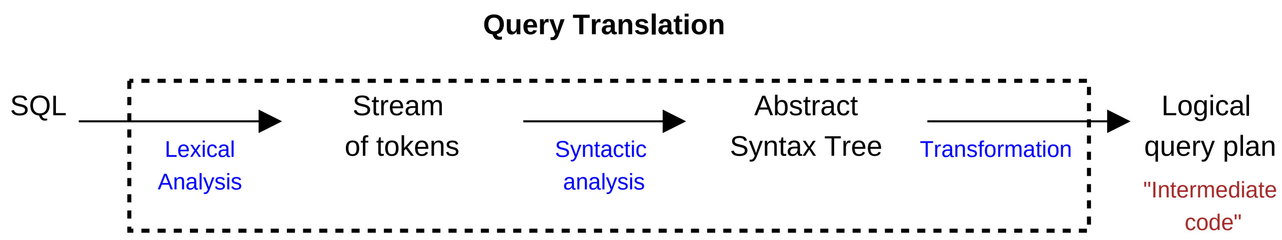
Query processing: overview



Query processing: overview



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

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

Refreshing the Relational Algebra

Union (set-based)

A	B
1	2
3	4
5	6

$\cup$

A	B
3	4
1	5

$=$

A	B
1	2
3	4
5	6
1	5

Input relations must have same set of attributes

Refreshing the Relational Algebra

Intersection (set-based)

A	B
1	2
3	4
5	6

$\cap$

A	B
3	4
1	5

$=$

A	B
3	4

Input relations must have same set of attributes

Refreshing the Relational Algebra

Difference (set-based)

A	B
1	2
3	4
5	6

$-$

A	B
3	4
1	5

$=$

A	B
1	2
5	6

Input relations must have same set of attributes

Refreshing the Relational Algebra

Selection

$$\sigma_{A>=3}\left(\begin{array}{c|c} A & B \\ \hline\hline 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{array}\right) = \begin{array}{c|c} A & B \\ \hline\hline 3 & 4 \\ 5 & 6 \end{array}$$

Refreshing the Relational Algebra

Projection (set-based)

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

Refreshing the Relational Algebra

Cartesian product

A	B
1	2
3	4

$\times$

C	D
2	6
3	7
4	9

$=$

A	B	C	D
1	2	2	6
1	2	3	7
1	2	4	9
3	4	2	6
3	4	3	7
3	4	4	9

Refreshing the Relational Algebra

Natural Join

A	B
1	2
3	4

$\bowtie$

B	D
2	6
3	7
4	9

$=$

A	B	D
1	2	6
3	4	9

Refreshing the Relational Algebra

Theta Join

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

Refreshing the Relational Algebra

Renaming

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

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

$$\sigma_{\texttt{length} \geq 100}(\texttt{Movie}) \bowtie_{\texttt{title=movietitle}} \texttt{StarsIn}$$

Refreshing the Relational Algebra

The extended relational algebra

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

Grouping:

$$\gamma_{A,\min(B)\to D}\left(\begin{array}{c|c|c} A & B & C \\ \hline\hline 1 & 2 & a \\ 1 & 3 & b \\ 2 & 3 & c \\ 2 & 4 & a \\ 2 & 5 & a \end{array}\right) \quad = \quad \begin{array}{c|c} A & D \\ \hline\hline 1 & 2 \\ 2 & 3 \end{array}$$

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:

$$\pi_{A,C \to D} \left(\begin{array}{c|c|c|c} A & B & C & D \\ \hline\hline 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 5 \\ 3 & 4 & 5 & 6 \\ 5 & 6 & 3 & 4 \end{array} \right) = \begin{array}{c|c} A & D \\ \hline\hline 1 & 3 \\ 3 & 5 \\ 5 & 3 \end{array}$$

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)

A	B
1	2
3	4
5	6

$\cup$

A	B
3	4
1	5

$=$

A	B
1	2
3	4
5	6
3	4
1	5

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)

A	B
1	2
3	4
1	2
1	2

$\cap$

A	B
1	2
3	4
3	4
5	6

$=$

A	B
1	2
3	4

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.

Difference (bag-based)

A	B
1	2
3	4
1	2
1	2

$-$

A	B
1	2
3	4
3	4
5	6

$=$

A	B
1	2
1	2

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)

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

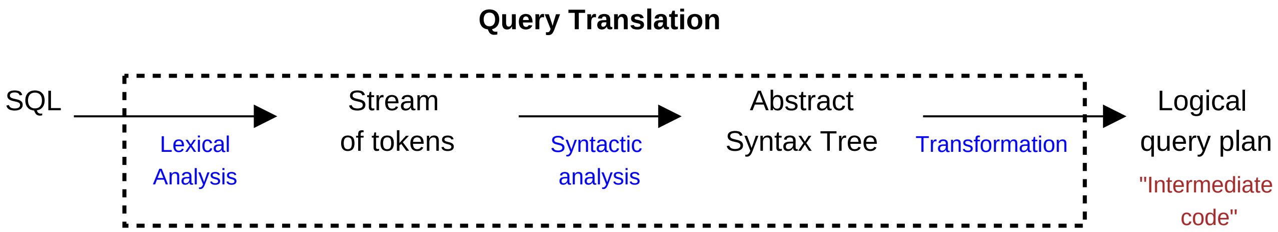
Refreshing the Relational Algebra

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.

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(<u>title: `string`, year: `int`</u>, length: `int`, genre: `string`, studioName: `string`, producerC#: `int`)
- MovieStar(<u>name: `string`</u>, address: `string`, gender: `char`, birthdate: `date`)
- StarsIn(<u>movieTitle: `string`, movieYear: `string`, starName: `string`</u>)
- MovieExec(name: `string`, address: `string`, <u>CERT#: `int`</u>, netWorth: `int`)
- Studio(<u>name: `string`</u>, 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: $\pi_{\texttt{movieTitle}}\, \sigma_{\substack{\texttt{starName}=\texttt{M.name} \\ \wedge \texttt{M.birthdate}=1960}} (\texttt{StarsIn} \times \rho_{\texttt{M}}(\texttt{MovieStar}))$

Translation of SQL into the relational algebra

Select statements in general contain subqueries

```
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 `EXISTS` and `NOT EXISTS`

```
SELECT name FROM MovieExec
WHERE netWorth >= ALL (SELECT E.netWorth
                       FROM MovieExec E)
```

$\Rightarrow$
```
SELECT name FROM MovieExec
WHERE NOT EXISTS(SELECT E.netWorth
                 FROM MovieExec E
                 WHERE netWorth < E.netWorth)
```

Translation of SQL into the relational algebra

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

```
SELECT C FROM S
WHERE C IN (SELECT SUM(B) FROM R
            GROUP BY A)
```

$\Rightarrow$
```
SELECT C FROM S
WHERE EXISTS (SELECT SUM(B) FROM R
              GROUP BY A
              HAVING SUM(B) = C)
```

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.

$$\pi_{\texttt{name}}\, \sigma_{\substack{\texttt{birthdate}=1960 \\ \wedge \texttt{name}=\texttt{S.starName}}} (\texttt{MovieStar}))$$

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.

$$\pi_{\texttt{name}}\, \sigma_{\substack{\texttt{birthdate}=1960 \\ \wedge \texttt{name}=\texttt{S.starName}}} (\texttt{MovieStar}))$$

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

$$\pi_{\texttt{S.movieTitle},\texttt{S.movieYear},\texttt{S.starName},\texttt{name}}\, \sigma_{\substack{\texttt{birthdate}=1960 \\ \wedge \texttt{name}=\texttt{S.starName}}} (\texttt{MovieStar} \times \rho_S(\texttt{StarsIn}))$$

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:

$$\boldsymbol{\rho}_S(\texttt{StarsIn}) \times \boldsymbol{\rho}_M(\texttt{Movie})$$

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 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.

$$(\boldsymbol{\rho}_S(\texttt{StarsIn}) \times \boldsymbol{\rho}_M(\texttt{Movie})) \bowtie \boldsymbol{\pi}_{\texttt{S.movieTitle,S.movieYear,S.starName}}\ \boldsymbol{\sigma}_{\substack{\texttt{birthdate}=1960 \\ \wedge \texttt{name}=\texttt{S.starName}}} (\texttt{MovieStar} \times \boldsymbol{\rho}_S(\texttt{StarsIn}))$$

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)
```

4. We can simplify this by omitting the first $\boldsymbol{\rho}_S(\texttt{StarsIn})$

$$\boldsymbol{\rho}_M(\texttt{Movie}) \bowtie \boldsymbol{\pi}_{\texttt{S.movieTitle},\texttt{S.movieYear},\texttt{S.starName}}\, \boldsymbol{\sigma}_{\substack{\texttt{birthdate}=1960\\ \wedge\texttt{name}=\texttt{S.starName}}} (\texttt{MovieStar} \times \boldsymbol{\rho}_S(\texttt{StarsIn}))$$

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)
```

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

$$\pi_{\texttt{S.movieTitle,M.studioName}}\ \sigma_{\texttt{S.movieYear>=2000} \wedge \texttt{S.movieTitle=M.title}} \Big(\rho_M(\texttt{Movie}) \bowtie \pi_{\texttt{S.movieTitle,S.movieYear,S.starName}}\ \sigma_{\substack{\texttt{birthdate=1960} \\ \wedge \texttt{name=S.starName}}} (\texttt{MovieStar} \times \rho_S(\texttt{StarsIn}))\Big)$$

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.

$$\pi_{\text{name}}\, \sigma_{\substack{\texttt{birthdate}=1960 \\ \wedge \texttt{name}=\texttt{S.starName}}} (\texttt{MovieStar})$$

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.

$$\pi_{\text{name}}\,\sigma_{\substack{\texttt{birthdate}=1960\\ \wedge\texttt{name}=\texttt{S.starName}}}(\texttt{MovieStar})$$

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

$$\pi_{\texttt{S.movieTitle},\texttt{S.movieYear},\texttt{S.starName},\text{name}}\,\sigma_{\substack{\texttt{birthdate}=1960\\ \wedge\texttt{name}=\texttt{S.starName}}}(\texttt{MovieStar}\times\rho_S(\texttt{StarsIn}))$$

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:

$$\boldsymbol{\rho}_S(\texttt{StarsIn}) \times \boldsymbol{\rho}_M(\texttt{Movie})$$

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.

$$(\rho_S(\texttt{StarsIn}) \times \rho_M(\texttt{Movie})) \;\overline{\bowtie}\; \pi_{\texttt{S.movieTitle,S.movieYear,S.starName}}\, \sigma_{\substack{\texttt{birthdate=1960}\\ \wedge\texttt{name=S.starName}}} (\texttt{MovieStar} \times \rho_S(\texttt{StarsIn}))$$

Here, the antijoin $R \,\overline{\bowtie}\, S \equiv R - (R \bowtie S)$.
Simplification is not possible: we cannot remove the first $\rho_S(\texttt{StarsIn})$.

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

$$\pi_{\texttt{S.movieTitle,M.studioName}}\, \sigma_{\texttt{S.movieYear>=2000} \wedge \texttt{S.movieTitle=M.title}} \Big((\rho_S(\texttt{StarsIn}) \times \rho_M(\texttt{Movie})) \bowtie \pi_{\texttt{S.movieTitle,S.movieYear,S.starName}}\, \sigma_{\substack{\texttt{birthdate=1960} \\ \wedge \texttt{name=S.starName}}} (\texttt{MovieStar} \times \rho_S(\texttt{StarsIn})) \Big)$$

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:

```
SELECT Select-list FROM From-list
WHERE ψ AND EXISTS(Q) AND ··· AND NOT EXISTS(P) AND ···
```

How do we treat the following?

```
SELECT Select-list FROM From-list
WHERE A=B AND NOT(EXISTS(Q) 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:

```
SELECT Select-list FROM From-list
WHERE ψ AND EXISTS(Q) AND ··· AND NOT EXISTS(P) AND ···
```

How do we treat the following?

```
SELECT Select-list FROM From-list
WHERE A=B AND NOT(EXISTS(Q) AND C<6)
```

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

```
SELECT Select-list FROM From-list
WHERE (A=B AND NOT EXISTS(Q)) OR (A=B 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:

```
SELECT Select-list FROM From-list
WHERE ψ AND EXISTS(Q) AND ··· AND NOT EXISTS(P) AND ···
```

How do we treat the following?

```
SELECT Select-list FROM From-list
WHERE A=B AND NOT(EXISTS(Q) AND C<6)
```

2. We then distribute the OR

```
(SELECT Select-list FROM From-list
WHERE (A=B AND NOT EXISTS(Q)))
UNION
(SELECT Select-list FROM From-list
WHERE (A=B AND C>=6))
```

Translation of SQL into the relational algebra

Union, intersection, and difference

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

Algebra: $\boldsymbol{\rho}_{R_1}(R) \cap \boldsymbol{\rho}_{R_2}(R)$

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

Algebra: $\boldsymbol{\rho}_{R_1}(R) \cup \boldsymbol{\rho}_{R_2}(R)$

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

Algebra: $\boldsymbol{\rho}_{R_1}(R) - \boldsymbol{\rho}_{R_2}(R)$

Translation of SQL into the relational algebra

Union, intersection, and difference in subqueries

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

```
SELECT S1.C, S2.C
FROM S S1, S S2
WHERE EXISTS (
  (SELECT R1.A, R1.B FROM R R1
   WHERE A = S1.C AND B = S2.C)
 UNION
  (SELECT R2.A, R2.B FROM R R2
   WHERE B = S1.C)
)
```

In this case we translate the subquery as follows:

$$\pi_{S_1.C, S_2.C, R_1.A \rightarrow A, R_1.B \rightarrow B}\, \sigma_{\substack{A=S_1.C \\ \wedge B=S_2.C}} (\rho_{R_1}(R) \times \rho_{S_1}(S) \times \rho_{S_2}(S)) \cup \pi_{S_1.C, S_2.C, R_2.A \rightarrow A, R_2.B \rightarrow B}\, \sigma_{B=S_1.C} (\rho_{R_2}(R) \times \rho_{S_1}(S) \times \rho_{S_2}(S))$$

Translation of SQL into the relational algebra

Join-expressions

SQL: `(SELECT * FROM R R1) CROSS JOIN (SELECT * FROM R R2)`

Algebra: $\boldsymbol{\rho}_{R_1}(R) \times \boldsymbol{\rho}_{R_2}(R)$

SQL: `(SELECT * FROM R R1) JOIN (SELECT * FROM R R2)`
`ON R1.A = R2.B`

Algebra: $\boldsymbol{\rho}_{R_1}(R) \underset{R_1.A=R_2.B}{\bowtie} \boldsymbol{\rho}_{R_2}(R)$

Translation of SQL into the relational algebra

Join-expressions in subqueries

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

```
SELECT S1.C, S2.C
FROM S S1, S S2
WHERE EXISTS (
  (SELECT R1.A, R1.B FROM R R1
   WHERE A = S1.C AND B = S2.C)
  CROSS JOIN
  (SELECT R2.A, R2.B FROM R R2
   WHERE B = S1.C)
)
```

In this case we translate the subquery as follows:

$$\boldsymbol{\pi}_{S_1.C,S_2.C,R_1.A,R_1.B}\, \boldsymbol{\sigma}_{\substack{A=S_1.C \\ \wedge B=S_2.C}} (\boldsymbol{\rho}_{R_1}(R) \times \boldsymbol{\rho}_{S_1}(S) \times \boldsymbol{\rho}_{S_2}(S)) \bowtie \boldsymbol{\pi}_{S_1.C,R_2.A,R_2.B}\, \boldsymbol{\sigma}_{B=S_1.C} (\boldsymbol{\rho}_{R_2}(R) \times \boldsymbol{\rho}_{S_1}(S))$$

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:

$$\pi_{\texttt{name},\texttt{SUM(length)}}\ \sigma_{\texttt{MIN(year)}<1930}\ \gamma_{\texttt{name},\texttt{MIN(year)},\texttt{SUM(length)}}\ \sigma_{\texttt{cert\#}=\texttt{producerC\#}}(\texttt{MovieExec} \times \texttt{Movie})$$

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:

$$\pi_{\texttt{movieTitle}}\sigma_{\texttt{starName}=\texttt{M.name}}(\texttt{StarsIn} \times \rho_{\texttt{M}}\pi_{\texttt{name}}\sigma_{\texttt{birthdate}=1960}(\texttt{MovieStar}))$$

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

$$E_1 = \boldsymbol{\pi}_{\texttt{name}}\, \boldsymbol{\sigma}_{\texttt{birthdate}=1960}(\texttt{MovieStar}).$$

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

$$E_2 = \rho_S \boldsymbol{\pi}_{\texttt{name},\texttt{movieTitle}}\, \boldsymbol{\sigma}_{\texttt{starName}=\texttt{M.name}}(\texttt{StarsIn} \times E_1).$$

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

$$\boldsymbol{\pi}_{\texttt{movieTitle}}(E_1 \bowtie E_2).$$

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
```

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:

$$\pi_{\texttt{movieTitle}}(E_2).$$

Translation of SQL into the relational algebra

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:

```
SELECT C, (SELECT B FROM R
           WHERE A=C)
FROM S
```

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

```
SELECT C, T.B
FROM (SELECT C FROM S),
     LATERAL
     (SELECT B FROM R
      WHERE A=C) T
```

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