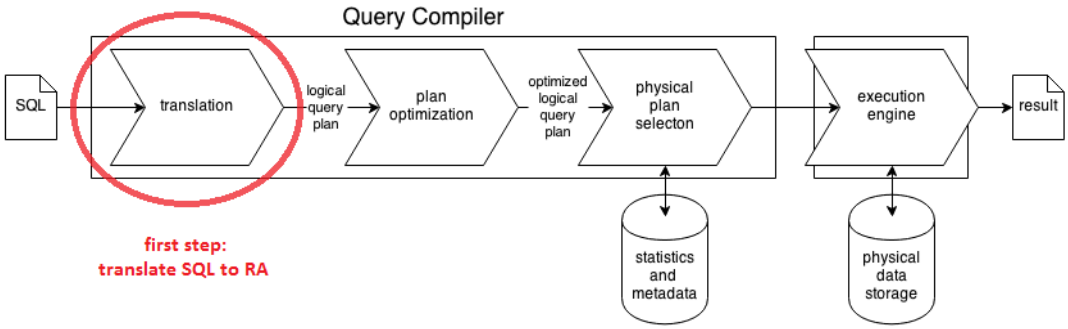
Translating SQL to Relational Algebra

Contents

Translating SQL to Relational Algebra



Translating SQL to RA expression is the second step in Query Processing Pipeline

- Input: Logical Query Plan - expression in Extended Relational Algebra
- Output: Optimized Logical Query Plan - also in Relational Algebra

Union, Intersection, Difference

Translation is straightforward

```
(SELECT * FROM R1) INTERSECT (SELECT * FROM R2)
```

Is $R_1 \cap R_2$

UNION $\rightarrow R_1 \cup R_2$

EXCEPT $\rightarrow R_1 - R_2$

Select-From-Where No Subqueries

Query

```
SELECT movieTitle
FROM StarsIn, MovieStarM
WHERE starName = M.name AND M.birthdate = 1960
```

- in the **from** clause we have all relations we need
- so we make a Cartesian Product for all relations there

- if there is an alias - we do Renaming
- then we filter the Cartesian Product
- then translate the **where** clause too

So we get:

$$\pi_{\text{movieTitle}}\sigma_{\text{starName} = \text{M.name} \wedge \text{M.birthdate} = 1960}(\text{StartsIn} \times \rho_M(\text{MovieStar}))$$

(Maybe not the most efficient way, but it will be optimized further)

Normalization Step

Suppose we have subqueries in the "Where" clause

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

Here we may have different constraints:

- $\text{in}, \leqslant, <, \geqslant, >, =, \neq$, etc
- whenever we have such constraints, we may replace them with quantifiers $\forall$ and $\exists$
- or with **EXISTS** and **IN** or **NOT EXISTS**
- so we first translate a SQL query to the equivalent SQL with **EXISTS** or **NOT EXISTS**

Example 1: IN

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

to

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

Example 2: $\geqslant$

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

to

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

Example 3: aggregated attributes

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

to

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

(note that in this case we use "HAVING" and not "WHERE")

So the first step when processing these kinds of queries is *normalization* step:

- translate a query into EXISTS/NOT EXISTS form

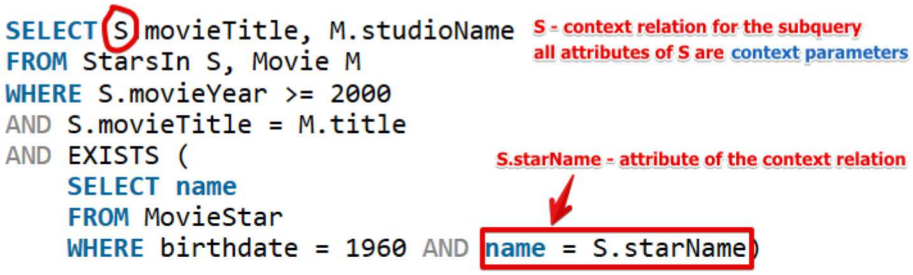
Hence we can assume that all queries are in this form

- We then apply the next step: for correlated queries

Correlated Queries

- A subquery can refer to attributes of relations that are introduces in the outer query
- **def**: we call such queries *correlated subqueries*
- the outer relation is called the *context relation* - a correlated subquery uses its attributes
- a *parameter* - is a set of attributes of all context relations of a subquery

Example:



here

- the subquery refers to S.starName, so it's correlated
- S is the context relation for the subquery
- S.starName is a parameter to the correlated subquery

EXISTS in the Where Clause (by example)

```
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)
```

Algorithm

- it's recursive: translate the subqueries first
  - $\pi_{\text{name}} \sigma_{\substack{\text{birthDate} = 1960 \wedge \\ \text{name} = \text{S.starName}}} (\text{MovieStar})$
  - problem: cannot find **S.starName** in the input relation
  - so it must be a correlated query
  - we therefore need to recognize that this is a context relation's parameter
- so we need to add the context relations and parameters
  - $\pi_{\substack{\text{S.movieTitle}, \\ \text{S.movieYear}, \\ \text{S.starName}, \\ \text{name}}} \sigma_{\substack{\text{birthDate} = 1960 \wedge \\ \text{name} = \text{S.starName}}} (\text{MovieStar} \times \rho_S(\text{StarsIn}))$
- next, we translate the "from" clause
  - $\rho_S(\text{StarsIn}) \times \rho_M(\text{Movie})$
- now we need to *synchronize* the subresult by join
  - from the subquery we need to keep only the parameter attributes (the blue ones) - can remove name
  - join: if something exists, we will join on it
  - $\left[\rho_S(\text{StarsIn}) \times \rho_M(\text{Movie})\right] \bowtie \left[\pi_{\substack{\text{S.movieTitle}, \\ \text{S.movieYear}, \\ \text{S.starName}}} \sigma_{\substack{\text{birthDate} = 1960 \wedge \\ \text{name} = \text{S.starName}}} (\text{MovieStar} \times \rho_S(\text{StarsIn}))\right]$
- note that we have $\rho_S(\text{StarsIn})$ on the both sides of the join
  - can just drop it (it won't affect the join)
  - $\left[\rho_M(\text{Movie})\right] \bowtie \left[\pi_{\substack{\text{S.movieTitle}, \\ \text{S.movieYear}, \\ \text{S.starName}}} \sigma_{\substack{\text{birthDate} = 1960 \wedge \\ \text{name} = \text{S.starName}}} (\text{MovieStar})\right]$
- finally we translate "WHERE" and "SELECT"
  - $\pi_{\substack{\text{S.movieTitle}, \\ \text{M.studioName}}} \sigma_{\substack{\text{S.movieYear} \geqslant 2000 \wedge \\ \text{S.movieTitle} = \text{M.title}}} \left[\rho_M(\text{Movie}) \bowtie \pi_{\substack{\text{S.movieTitle}, \\ \text{S.movieYear}, \\ \text{S.starName}}} \sigma_{\substack{\text{birthDate} = 1960 \wedge \\ \text{name} = \text{S.starName}}} (\text{MovieStar})\right]$

NOT EXISTS in the Where Clause (by example)

```
SELECTS.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)
```

Algorithm

- Same as before: we translate the subquery
  - $\pi_{\text{name}} \sigma_{\substack{\text{birthDate} = 1960 \wedge \\ \text{name} = \text{S.starName}}} (\text{MovieStar})$
- Then we add context relations and context parameters
  - $\pi_{\substack{\text{S.movieTitle}, \\ \text{S.movieYear}, \\ \text{S.starName}, \\ \text{name}}} \sigma_{\substack{\text{birthDate} = 1960 \wedge \\ \text{name} = \text{S.starName}}} (\text{MovieStar} \times \rho_S(\text{StarsIn}))$
- And same for the FROM clause
  - $\rho_S(\text{StarsIn}) \times \rho_M(\text{Movie})$
- Then we need to *synchronize* the results, but this time with Anti-Join ($\bar{\bowtie}$)
  - $\left[\rho_S(\text{StarsIn}) \times \rho_M(\text{Movie})\right] \bar{\bowtie} \left[\pi_{\substack{\text{S.movieTitle}, \\ \text{S.movieYear}, \\ \text{S.starName}}} \sigma_{\substack{\text{birthDate} = 1960 \wedge \\ \text{name} = \text{S.starName}}} (\text{MovieStar} \times \rho_S(\text{StarsIn}))\right]$
  - note that here the simplification is not possible: the semantics of Anti-Join is different from Join
  - so we cannot remove $\rho_S(\text{StarsIn})$ from both parts
- the last step is the same: we translate "WHERE" and "SELECT"
  - $\pi_{\substack{\text{S.movieTitle}, \\ \text{M.studioName}}} \sigma_{\substack{\text{S.movieYear} \geqslant 2000 \wedge \\ \text{S.movieTitle} = \text{M.title}}} \left[\left[\rho_S(\text{StarsIn}) \times \rho_M(\text{Movie})\right] \bar{\bowtie} \left[\pi_{\substack{\text{S.movieTitle}, \\ \text{S.movieYear}, \\ \text{S.starName}}} \sigma_{\substack{\text{birthDate} = 1960 \wedge \\ \text{name} = \text{S.starName}}} (\text{MovieStar} \times \rho_S(\text{StarsIn}))\right]\right]$

EXISTS Subqueries in WHERE Combined with Other

So far we've considered only queries of the following form:

```
SELECT ... FROM ...
WHERE ... AND
      EXISTS (...) AND
      ... AND
      NOT EXISTS (...)
```

I.e. EXISTS and NOT EXISTS are in the "WHERE" clause joined by "AND"

What about the following query?

```
SELECT ... FROM ...
WHERE
    A = B AND NOT (EXISTS (...) AND C < 6)
```

- First, we translate the condition into Disjunctive Normal Form

```
SELECT ... FROM ...
WHERE
    (A = B AND NOT (EXISTS (...))) OR
    (A = B AND C >= 6)
```

- Then we distribute OR (to UNION)

```
(SELECT ... FROM ...
  WHERE
    A = B AND NOT EXISTS (...))
UNION
(SELECT ... FROM ...
  WHERE
    A = B AND C >= 6)
```

As we've seen, UNION is translated as $\cup$

Union In Subqueries

We may have UNOIN in subqueries

```
SELECT S1.C, S2.C
FROM S S1, S S2
WHERE EXISTS (
  (SELECT R1.A, R1.B FROMR R1
   WHERE A = S1.C AND B = S2.C) -- (1)
  UNION
  (SELECT R2.A, R2.B FROMR R2
   WHERE B = S1.C) -- (2)
)
```

- Recall that to be able to UNION two relations, they must have the same schema
- But in this case:
  - (1) has 2 context relations S_1 and S_2
  - (2) has only 1 context relation S_1
- $\Rightarrow$ When translating, need to add S_2 to (2) as well
- and make sure that they have the same name

$$\underbrace{\left(\pi_{\substack{S_1.C,\, S_2.C,\\ R_1.A \to A,\\ R_1.B \to B}} \sigma_{\substack{A=S_1.C \wedge\\ B=S_2.C}} \left[\rho_{R_1}(R) \times \rho_{S_1}(S) \times \rho_{S_2}(S)\right]\right)}_{(1)} \cup \underbrace{\left(\pi_{\substack{S_1.C,\, S_2.C,\\ R_1.A \to A,\\ R_1.B \to B}} \sigma_{B=S_1.C} \left[\rho_{R_1}(R) \times \rho_{S_1}(S) \times \rho_{S_2}(S)\right]\right)}_{(2)}$$

Translating Joins

Joins

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

We translate as follows:

- $\rho_{R_1}(R) \bowtie_{R_1.A=R_2.B} \rho_{R_2}(R)$

Group and Having

Suppose we have the following query:

```
SELECT name, SUM(length)
FROM MovieExec, Movie
WHERE cert = producer
GROUP BY name
HAVING MIN(year) < 1930
```

We translate it as

- $\pi_{\substack{\text{name},\\ \text{SUM(length)}}} \sigma_{\text{MIN(year)}<1930} \gamma_{\substack{\text{name},\\ \text{MIN(year)},\\ \text{SUM(length)}}} \sigma_{\text{cert} = \text{producer}} (\text{MovieExec} \times \text{Movie})$
- here the translate the **HAVING** clause as σ before the γ
- also note that **SUM(length)** goes to γ

Exercises

Exercises from Database Systems Architecture (ULB)

- the exercises: [1]
- the proposed solutions [2]

Exercise 1

The given relations:

- Student(snum, sname, major, level, age)
- Class(name, meets_at, room, fid)
- Enrolled(snum, cname)
- Faculty(fid, fname, deptid)

```
SELECT C.name
FROM Class C
WHERE C.room = 'R128' OR
      C.name IN (
          SELECT E.cname
          FROM Enrolled E
          GROUP BY E.cname
          HAVING COUNT(*) >= 5)
```

First we distribute OR

```
SELECT C.name
FROM Class C
WHERE C.room = 'R128'

UNION

SELECT C.name
FROM Class C
WHERE C.name IN (
    SELECT E.cname
    FROM Enrolled E
    GROUP BY E.cname
    HAVING COUNT(*) >= 5)
```

for the subquery we replace IN to EXISTS

```
SELECT C.name
FROM Class C
WHERE EXISTS (
    SELECT E.cname
    FROM Enrolled E
    WHERE E.cname = C.name
    GROUP BY E.cname
    HAVING COUNT(*) >= 5)
```

Now we translate the subquery

- $q_1 = \pi_{\text{E.name, C.*}} \sigma_{\text{cat} \geqslant 5} \gamma_{\substack{\text{E.cname,} \\ \text{count(*)} \to \text{cnt,} \\ \text{C.*}}} \sigma_{\text{E.cname} = \text{C.name}} \left(\rho_E(\text{Enrolled}) \times \rho_C(\text{Class}) \right)$
- **note** that we use $\gamma_{\substack{\text{E.cname,} \\ \text{count(*)} \to \text{cnt,} \\ \text{C.*}}}$ and not $\gamma_{\substack{\text{E.cname,} \\ \text{count(*)} \to \text{cnt}}}$, because in the second case it will return only the two specified columns

Next, we need to synchronize (or "decorrelate") the subquery q_1 and the outer query

- $\pi_{\text{C.name}} \left[\rho_C(\text{Class}) \bowtie \pi_{\text{C.*}} \pi_{\text{E.name, C.*}} \sigma_{\text{cat} \geqslant 5} \gamma_{\substack{\text{E.cname,} \\ \text{count(*)} \to \text{cnt,} \\ \text{C.*}}} \sigma_{\text{E.cname} = \text{C.name}} \left(\rho_E(\text{Enrolled}) \times \rho_C(\text{Class}) \right) \right]$
- add $\pi_{\text{C.*}}$ because we need only these values - **E.name** was used for EXISTS part only
- since we have $\rho_C(\text{Class})$ on both sides of the Join - we can drop the first one (as well as the Join)
- and we also can merge successive projections
- so we get:
- $\pi_{\text{C.name}} \sigma_{\text{cat} \geqslant 5} \gamma_{\substack{\text{E.cname,} \\ \text{count(*)} \to \text{cnt,} \\ \text{C.*}}} \sigma_{\text{E.cname} = \text{C.name}} \left(\rho_E(\text{Enrolled}) \times \rho_C(\text{Class}) \right)$

Now we do the union (easy)

- Since both parts have the same schema, union is possible
- The total results is:
- $\pi_{\text{C.name}} \sigma_{\text{C.room} = \text{'R128'}} \rho_C(\text{Class}) \cup \pi_{\text{C.name}} \sigma_{\text{cat} \geqslant 5} \gamma_{\substack{\text{E.cname,} \\ \text{count(*)} \to \text{cnt,} \\ \text{C.*}}} \sigma_{\text{E.cname} = \text{C.name}} \left(\rho_E(\text{Enrolled}) \times \rho_C(\text{Class}) \right)$

Exercise with the Count Bug

```
SELECT F.fname
FROM Faculty F
WHERE 5 > (
    SELECT COUNT(E.snum)
    FROM Class C, Enrolled E
    WHERE C.name = E.cname AND
    C.fid = F.fid)
```

First translate to an equivalent EXISTS query

```
SELECT F.fname
FROM Faculty F
WHERE EXISTS (
    SELECT COUNT(E.snum) as CNT
    FROM Class C, Enrolled E
    WHERE C.name = E.cname AND
    C.fid = F.fid
    HAVING CNT < 5)
```

Remarks

- note the change in the sign from > to <
- also we use HAVING instead of WHERE - because GROUP is assumed
- not all databases will take this kind of query.
  - For instance, MySQL will not (however it's not fully SQL compliant)

Using the rules, we try to translate the query this way:

- first we translate the subquery
  - $\pi_{\substack{\text{cnt),} \\ \text{F.fid,} \\ \text{F.fname,} \\ \text{F.deptid}}} \sigma_{\text{cnt} < 5} \gamma_{\substack{\text{count(E.snum)} \to \text{cnt,} \\ \text{F.*}}} \sigma_{\substack{\text{C.name} = \text{E.cname} \wedge \\ \text{C.fid} = \text{F.fid}}} \left[\rho_C(\text{Class}) \times \rho_E(\text{Enrolled}) \times \rho_F(\text{Faculty}) \right]$
- then decorrelate it:
  - $\rho_F(\text{Faculty}) \bowtie \left(\pi_{\text{F.*}} \pi_{\substack{\text{cnt,} \\ \text{F.fid,} \\ \text{F.fname,} \\ \text{F.deptid}}} \sigma_{\text{cnt} < 5} \gamma_{\substack{\text{count(E.snum)} \to \text{cnt,} \\ \text{F.*}}} \sigma_{\substack{\text{C.name} = \text{E.cname} \wedge \\ \text{C.fid} = \text{F.fid}}} \left[\rho_C(\text{Class}) \times \rho_E(\text{Enrolled}) \times \rho_F(\text{Faculty}) \right] \right)$
- can remove $\rho_F(\text{Faculty})$ and keep only needed projection attributes

- $\pi_{\text{F.name}} \sigma_{\text{cnt}<5} \gamma_{\substack{\text{count(E.snum)} \to \text{cnt}, \\ \text{F.*}}} \sigma_{\substack{\text{C.name} = \text{E.cname} \wedge \\ \text{C.fid} = \text{F.fid}}} \left[\rho_C(\text{Class}) \times \rho_E(\text{Enrolled}) \times \rho_F(\text{Faculty}) \right]$

Note that this is **not the query we want**!!!

- Faculty members who don't teach any class are not output by the expression, but they are output by the original SQL query

Count bug

- this issue is known as the *count bug*
- it occurs when we have subqueries use COUNT without GROUP BY
- to solve it we need to use right outer join instead of $\times$

$$\pi_{\text{F.name}} \sigma_{\text{cnt}<5} \gamma_{\substack{\text{count(E.snum)} \to \text{cnt}, \\ \text{F.*}}} \sigma_{\substack{\text{C.name} = \text{E.cname} \wedge \\ \text{C.fid} = \text{F.fid}}} \left[\rho_C(\text{Class}) \times \rho_E(\text{Enrolled}) \rtimes^R_{\text{C.fid} = \text{F.fid}} \rho_F(\text{Faculty}) \right]$$

See also

- Relational Algebra
- Lecture Notes by S. Vansummeren [3]

Sources

- Database Systems Architecture (ULB)

Retrieved from "http://mlwiki.org/index.php?title=Translating_SQL_to_Relational_Algebra&oldid=823"

Category: Relational Databases
