Formal Query Languages Part 2: Comprehensive Review

DSC 208R - Data Management for Analytics

Combining Relational Algebra Operators

Relational algebra operators can be combined to form complex queries, similar to how expressions are built in programming languages. Precedence rules dictate the order of evaluation.

Operator Precedence (Highest to Lowest)

1. Unary operators: ρ, π, σ
2. Cartesian product and Joins: $\times$, $\bowtie$
3. Intersection: $\cap$
4. Union and Set Difference: $\cup$, $\setminus$

Combining Operators Example

Consider the Movie table:

Name	**Year**	**Genre**	**Budget**	**Revenue**
Pirates of the Caribbean	2007	Action	\$300 m	\$900 m
The Lion King	2019	Animation	\$260 m	\$1.65 b
The Dark Knight	2008	Action	\$185 m	\$1 b

Projection example: $\pi_{\text{name,rate}}(\text{Movie})$

This projects the 'name' and 'rate' columns.

Name	**Rate**
Pirates of the Caribbean	7.1
The Lion King	6.5
The Dark Knight	9.5

Selection example: $\sigma_{\text{budget}>\$200m}(\text{Movie})$

This selects movies with a budget greater than \$200 million.

Name	**Year**	**Genre**	**Budget**	**Revenue**
Pirates of the Caribbean	2007	Action	\$300 m	\$900 m
The Lion King	2019	Animation	\$260 m	\$1.65 b

Combined example: $\pi_{\text{name,rate}}(\sigma_{\text{budget}>\$200m}(\text{Movie}))$

This first selects movies by budget, then projects name and rate.

Name	**Rate**
Pirates of the Caribbean	7.1
The Lion King	6.5

Outer Joins

Outer joins keep all information from both operands, padding with 'NULL' values for tuples that have no counterpart in the other relation.

Three Variants of Outer Join

- **Left Outer Join**: Keeps all rows from the left table.
- **Right Outer Join**: Keeps all rows from the right table.
- **Full Outer Join**: Keeps all rows from both tables.

Notation for Outer Joins: Relational Algebra: $R1 \bowtie_{R1.A1=R2.A2} R2$ (Left Join) SQL: 'SELECT * FROM R1 LEFT JOIN R2 ON R1.A1 = R2.A2' Pandas: $df1.merge(df2, left_on =' A1', right_on =' A2', how =' left')$

Outer Join Example:

Movie Table (M):

Name	Year	Genre
Apocalypse Now	1979	War
The God Father	1972	Crime
Planet Earth II	2016	Nature Documentary

ActedIN Table (A):

Actorname	Moviename
Marlon Brando	Apocalypse now
Al Pacino	The God Father
Marlon Brando	The God Father

Result of 'Movie' Left Outer Join 'ActedIN' on 'Movie.name = ActedIN.moviename':

Name	Year	Genre	Actorname	Moviename
Apocalypse now	1979	War	Marlon Brando	Apocalypse Now
The God Father	1972	Crime	Al Pacino	The God Father
Planet Earth II	2016	Nature Documentary	NULL	NULL

Relational Algebra for Bags

SQL uses bag semantics, meaning duplicate tuples are preserved. Relational engines often work on bags for efficiency.

Extended Relational Algebra Operators on Bags

- **Selection ($\sigma_\theta(R)$)**: Preserves the number of occurrences of tuples.
- **Projection ($\pi_A(R)$)**: No duplicate elimination (unlike set-based projection).
- **Cartesian product, Join**: No duplicate elimination.
- **Duplicate elimination ($\delta(R)$)**: 'R1 := $\delta(R2)$' - R1 consists of one copy of each tuple that appears in R2 one or more times.
- **Grouping (γ)**: A list of elements that are either individual (grouping) attributes or 'AGG(A)' (aggregation operator on attribute A).
- **Sorting (τ)**: 'R1 := $\tau_L(R2)$' - R1 consists of tuples of R2 sorted on the attributes in list L.

Grouping / Aggregation Example

Input ‘Movie‘ table (with ‘Revenue‘):

Name	Year	Genre	Revenue
Pirates of the Caribbean	2007	Action	$900 m
The Lion King	2019	Animation	$1.65 b
The Dark Knight	2008	Action	$1 b
Toy Story 3	2010	Animation	$1 b
American Sniper	2013	Action	$350 m

Grouping by ‘Genre‘ and calculating ‘SUM(Revenue)‘ as ‘TotalRevenue‘, and ‘COUNT(*)‘ as ‘MovieNum‘:

$\gamma_{\text{genre, SUM(Revenue)} \rightarrow \text{TotalRevenue, COUNT(*)} \rightarrow \text{MovieNum}}(\text{Movie})$

Result:

Genre	TotalRevenue	MovieNum
Animation	$2.65 b	2
Action	$2.25 b	3

Representing Relational Algebra Queries as Trees

Complex RA queries can be represented as expression trees, which are useful for query optimization.

Example 1: Select-Project-Join Query

SQL Query:

```
SELECT genre
FROM Movie, ActedIN
WHERE Movie.name = ActedIN.moviename AND
      ActedIN.actorname = 'Marlon Brando';
```

Relational Algebra Equivalent: $\pi_{\text{genre}}(\text{Movie} \bowtie (\sigma_{\text{actorname='Marlon Brando'}}(\text{ActedIN})))$

Example 2: Grouping and Aggregation Query

SQL Query:

```
SELECT genre, SUM(revenue) AS TR, COUNT(*) AS C
FROM Movie
WHERE year > 2008
GROUP BY genre
HAVING SUM(revenue) > $1B;
```

Relational Algebra Representation (simplified tree notation):

The query involves selection, grouping with aggregation, and then a filtering (having) operation on the grouped results, followed by projection.

Typical Query Plan

- For ‘SELECT-PROJECT-JOIN‘ queries, the plan generally involves selections, joins, and then projection.

- For queries with ‘GROUP BY‘ and ‘HAVING‘, the order is typically: ‘FROM‘ (identifying tables), ‘WHERE‘ (filtering rows), ‘GROUP BY‘ (grouping), ‘HAVING‘ (filtering groups), and finally ‘SELECT‘ (projecting fields and aggregates).

Subqueries and Relational Algebra

SQL subqueries can often be de-correlated and expressed using relational algebra set operations.

Example: ‘EXISTS‘ and ‘NOT EXISTS‘

SQL Query: Find actors who acted in an ’Action’ movie but NOT a ’Drama’ movie.

```
SELECT a.actorname
FROM ActedIn a
WHERE EXISTS (SELECT a.actorname FROM Movie m WHERE a.moviename = m.name AND m.genre = 'Action
    ')
  AND NOT EXISTS (SELECT a.actorname FROM Movie m WHERE a.moviename = m.name AND m.genre = '
      Drama');
```

This query uses correlated subqueries.

Example: ‘EXCEPT‘ (Set Difference)

The previous query can be rewritten using ‘EXCEPT‘ (set difference):

```
SELECT a1.actorname
FROM Movie m1, ActedIN a1
WHERE m1.name = a1.moviename AND m1.genre = 'Action'
EXCEPT
SELECT a2.actorname
FROM Movie m2, ActedIN a2
WHERE m2.name = a2.moviename AND m2.genre = 'Drama';
```

Relational Algebra Equivalent: $\pi_{\text{actorname}}(\text{Movie} \bowtie \sigma_{\text{genre='Action'}}(\text{ActedIN})) \setminus \pi_{\text{actorname}}(\text{Movie} \bowtie \sigma_{\text{genre='Drama'}}(\text{ActedIN})$

Summary

- SQL is a declarative language (describes *what* to get).
- Relational Algebra (RA) is a procedural language (describes *how* to get it).
- SQL and Relational Algebra express the same class of queries (they are equivalent in expressive power for core operations).