Extended Operators in SQL and Relational Algebra

T. M. Murali

September 16, 2009

Bags or Sets?

- So far, we have said that relational algebra and SQL operate on relations that are *sets* of tuples.
- Real RDBMSs treat relations as *bags* of tuples.
  - A tuple *can* appear multiple times in a relation.

Bags or Sets?

- So far, we have said that relational algebra and SQL operate on relations that are *sets* of tuples.
- Real RDBMSs treat relations as *bags* of tuples.
  - A tuple *can* appear multiple times in a relation.
  - Performance is one of the main reasons;

Bags or Sets?

- So far, we have said that relational algebra and SQL operate on relations that are *sets* of tuples.
- Real RDBMSs treat relations as *bags* of tuples.
  - A tuple *can* appear multiple times in a relation.
  - Performance is one of the main reasons; duplicate elimination is expensive since it requires sorting.
- If we use bag semantics, we may have to redefine the meaning of each relation algebra operator.

Bag Semantics: Projection and Selection

- Projection ($\pi()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.
- Selection ($\sigma()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.

Bag Semantics: Projection and Selection

- Projection ($\pi()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.
- Selection ($\sigma()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.

R

A	B	C
1	2	3
1	2	4
2	3	4
2	3	4

Bag Semantics: Projection and Selection

- Projection ($\pi()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.
- Selection ($\sigma()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.

R

A	B	C
1	2	3
1	2	4
2	3	4
2	3	4

$\pi_{A,B}$(R)

Bag Semantics: Projection and Selection

- Projection ($\pi()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.
- Selection ($\sigma()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.

R

A	B	C
1	2	3
1	2	4
2	3	4
2	3	4

$\pi_{A,B}$(R)

A	B
1	2
1	2
2	3
2	3

Bag Semantics: Projection and Selection

- Projection ($\pi()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.
- Selection ($\sigma()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.

R

A	B	C
1	2	3
1	2	4
2	3	4
2	3	4

$\pi_{A,B}$(R)

A	B
1	2
1	2
2	3
2	3

$\sigma_{C \geq 3}(R)$

Bag Semantics: Projection and Selection

- Projection ($\pi()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.
- Selection ($\sigma()$): process each tuple independently; a tuple may appear in the resulting relation multiple times.

R		
A	B	C
1	2	3
1	2	4
2	3	4
2	3	4

$\pi_{A,B}$(R)	
A	B
1	2
1	2
2	3
2	3

$\sigma_{C \geq 3}(R)$		
A	B	C
1	2	3
1	2	4
2	3	4
2	3	4

Bag Semantics: Union, Intersection, and Difference

- $R \cup S$: if tuple t appears k times in R and l times in S, t appears in $R \cup S$ $k + l$ times.

Bag Semantics: Union, Intersection, and Difference

- $R \cup S$: if tuple t appears k times in R and l times in S, t appears in $R \cup S$ $k + l$ times.

R

A	B
1	2
1	2
2	3
2	3

S

A	B
1	2
1	2
1	2
2	3
2	4

$R \cup S$

Bag Semantics: Union, Intersection, and Difference

- $R \cup S$: if tuple t appears k times in R and l times in S, t appears in $R \cup S$ $k + l$ times.

R

A	B
1	2
1	2
2	3
2	3

S

A	B
1	2
1	2
1	2
2	3
2	4

$R \cup S$

A	B
1	2
1	2
1	2
1	2
1	2
2	3
2	3
2	3
2	4

Bag Semantics: Union, Intersection, and Difference

- $R \cup S$: if tuple t appears k times in R and l times in S, t appears in $R \cup S$ $k + l$ times.
- $R \cap S$: if tuple t appears k times in R and l times in S, t appears in $R \cap S$ $\min\{k, l\}$ times.

R

A	B
1	2
1	2
2	3
2	3

S

A	B
1	2
1	2
1	2
2	3
2	4

$R \cup S$

A	B
1	2
1	2
1	2
1	2
1	2
2	3
2	3
2	3
2	4

$R \cap S$

Bag Semantics: Union, Intersection, and Difference

- $R \cup S$: if tuple t appears k times in R and l times in S, t appears in $R \cup S$ $k + l$ times.
- $R \cap S$: if tuple t appears k times in R and l times in S, t appears in $R \cap S$ $\min\{k, l\}$ times.

R

A	B
1	2
1	2
2	3
2	3

S

A	B
1	2
1	2
1	2
2	3
2	4

$R \cup S$

A	B
1	2
1	2
1	2
1	2
1	2
2	3
2	3
2	3
2	4

$R \cap S$

A	B
1	2
1	2
2	3

Bag Semantics: Union, Intersection, and Difference

- $R \cup S$: if tuple t appears k times in R and l times in S, t appears in $R \cup S$ $k + l$ times.
- $R \cap S$: if tuple t appears k times in R and l times in S, t appears in $R \cap S$ $\min\{k, l\}$ times.
- $R - S$: if tuple t appears k times in R and l times in S, t appears in $R - S$ $\max\{0, k - l\}$ times.

R

A	B
1	2
1	2
2	3
2	3

S

A	B
1	2
1	2
1	2
2	3
2	4

$R \cup S$

A	B
1	2
1	2
1	2
1	2
1	2
2	3
2	3
2	3
2	4

$R \cap S$

A	B
1	2
1	2
2	3

$R - S$

Bag Semantics: Union, Intersection, and Difference

- $R \cup S$: if tuple t appears k times in R and l times in S, t appears in $R \cup S$ $k + l$ times.
- $R \cap S$: if tuple t appears k times in R and l times in S, t appears in $R \cap S$ $\min\{k, l\}$ times.
- $R - S$: if tuple t appears k times in R and l times in S, t appears in $R - S$ $\max\{0, k - l\}$ times.

R

A	B
1	2
1	2
2	3
2	3

S

A	B
1	2
1	2
1	2
2	3
2	4

$R \cup S$

A	B
1	2
1	2
1	2
1	2
1	2
2	3
2	3
2	3
2	4

$R \cap S$

A	B
1	2
1	2
2	3

$R - S$

A	B
2	3

Bag Semantics: Products and Joins

- Product ($\times$): If a tuple r appears k times in a relation R and tuple s appears l times in a relation S, then the tuple rs appears kl times in $R \times S$.
- Theta-join and Natural join ($\bowtie$): Since both can be expressed as applying a selection followed by a projection to a product, use the semantics of selection, projection, and the product.

Extended Operators

- Powerful operators based on basic relational operators and bag semantics.

Extended Operators

- Powerful operators based on basic relational operators and bag semantics.
- Sorting: convert a relation into a *list* of tuples.
- Duplicate elimination: turn a bag into a set by eliminating duplicate tuples.

Extended Operators

- Powerful operators based on basic relational operators and bag semantics.
- Sorting: convert a relation into a *list* of tuples.
- Duplicate elimination: turn a bag into a set by eliminating duplicate tuples.
- Grouping: partition the tuples of a relation into groups, based on their values among specified attributes.

Extended Operators

- Powerful operators based on basic relational operators and bag semantics.
- Sorting: convert a relation into a *list* of tuples.
- Duplicate elimination: turn a bag into a set by eliminating duplicate tuples.
- Grouping: partition the tuples of a relation into groups, based on their values among specified attributes.
- Aggregation: used by the grouping operator and to manipulate/combine attributes.

Extended Operators

- Powerful operators based on basic relational operators and bag semantics.
- Sorting: convert a relation into a *list* of tuples.
- Duplicate elimination: turn a bag into a set by eliminating duplicate tuples.
- Grouping: partition the tuples of a relation into groups, based on their values among specified attributes.
- Aggregation: used by the grouping operator and to manipulate/combine attributes.
- Extended projections: projection on steroids.
- Outerjoin: extension of joins that make sure every tuple is in the output.

Sorting

RA $\tau_{A_1,A_2,\ldots}(R)$.

SQL `SELECT ... FROM ... WHERE ... ORDER BY` $\mathtt{A}_1, \mathtt{A}_2, \ldots$.

Sorting

RA $\tau_{A_1, A_2, \ldots}(R)$.

SQL `SELECT ... FROM ... WHERE ... ORDER BY` $A_1, A_2, \ldots$.

- The result is a list of tuples in R but with the tuples sorted by their values in attributes A_1, A_2, ...
- In SQL, use `DESC` after an attribute to specify sorting in descending order; `ASC` is the default.
- If you use the result in another query, sorted order is lost.

Duplicate Elimination

RA $\delta(R)$ is the relation containing exactly one copy of each tuple in R.

SQL `SELECT DISTINCT ...`

Duplicate Elimination

RA $\delta(R)$ is the relation containing exactly one copy of each tuple in R.

SQL `SELECT DISTINCT ...`

- Duplicate elimination is *expensive*, since tuples must be sorted or partitioned.

Duplicate Elimination

RA $\delta(R)$ is the relation containing exactly one copy of each tuple in R.

SQL `SELECT DISTINCT ...`

- Duplicate elimination is *expensive*, since tuples must be sorted or partitioned.
- Set operations in SQL (`UNION`, `INTERSECT`, and `EXCEPT`) operate on *sets* of tuples, i.e., they first eliminate duplicates.
- To make these operators treat relations as bags, follow the operation with the keyword `ALL`.

Aggregation

- Operators that summarise or aggregate the values in a single attribute of a relation.
- Operators are the same in relational algebra and SQL.
- All operators treat a relation as a bag of tuples.

- `SUM`: computes the sum of a column with numerical values.

Aggregation

- Operators that summarise or aggregate the values in a single attribute of a relation.
- Operators are the same in relational algebra and SQL.
- All operators treat a relation as a bag of tuples.

- `SUM`: computes the sum of a column with numerical values.
- `AVG`: computes the average of a column with numerical values.

Aggregation

- Operators that summarise or aggregate the values in a single attribute of a relation.
- Operators are the same in relational algebra and SQL.
- All operators treat a relation as a bag of tuples.

- `SUM`: computes the sum of a column with numerical values.
- `AVG`: computes the average of a column with numerical values.
- `MIN` and `MAX`:
  - for a column with numerical values, computes the smallest or largest value, respectively.
  - for a column with string or character values, computes the lexicographically smallest or largest values, respectively.

Aggregation

- Operators that summarise or aggregate the values in a single attribute of a relation.
- Operators are the same in relational algebra and SQL.
- All operators treat a relation as a bag of tuples.

- SUM: computes the sum of a column with numerical values.
- AVG: computes the average of a column with numerical values.
- MIN and MAX:
  - for a column with numerical values, computes the smallest or largest value, respectively.
  - for a column with string or character values, computes the lexicographically smallest or largest values, respectively.
- COUNT: computes the number of non-NULL tuples in a column.
  - In SQL, can use COUNT (*) to count the number of tuples in a relation.

Grouping

- How do we answer the query "Count the number of classes and the total enrollment of the classes each department teaches"?

Grouping

- How do we answer the query "Count the number of classes and the total enrollment of the classes each department teaches"?
- Can we answer the query using the operators discussed so far?

Grouping

- How do we answer the query "Count the number of classes and the total enrollment of the classes each department teaches"?
- Can we answer the query using the operators discussed so far?
- We need to group the tuples of `Teach` by `DeptName` and then aggregate within each group.

Grouping

- How do we answer the query "Count the number of classes and the total enrollment of the classes each department teaches"?
- Can we answer the query using the operators discussed so far?
- We need to group the tuples of `Teach` by `DeptName` and then aggregate within each group.
- Use the grouping operator.

Example of Grouping in Relational Algebra

- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?

Example of Grouping in Relational Algebra

- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?
  1. Group `Courses` by `DeptName`.

Example of Grouping in Relational Algebra

- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?
  1. Group `Courses` by `DeptName`.
  2. For each group, create a new attribute that stores the number of classes taught by the department.

Example of Grouping in Relational Algebra

- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?
  1. Group `Courses` by `DeptName`.
  2. For each group, create a new attribute that stores the number of classes taught by the department.
  3. For each group, create a new attribute that stores the total enrollment of the classes taught by the department.

Example of Grouping in Relational Algebra

- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?
  1. Group `Courses` by `DeptName`.
  2. For each group, create a new attribute that stores the number of classes taught by the department.
  3. For each group, create a new attribute that stores the total enrollment of the classes taught by the department.
- γ_L(`Courses`), where L is a list containing three elements:
  1. `DeptName`: the *grouping attribute*

Example of Grouping in Relational Algebra

- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?
  1. Group `Courses` by `DeptName`.
  2. For each group, create a new attribute that stores the number of classes taught by the department.
  3. For each group, create a new attribute that stores the total enrollment of the classes taught by the department.
- γ_L(`Courses`), where L is a list containing three elements:
  1. `DeptName`: the *grouping attribute*,
  2. `COUNT(Number)` $\rightarrow$ `NumCourses`: an *aggregated attribute* computing the count of the `Number` attribute in each group and naming the new attribute `NumCourses`, and

Example of Grouping in Relational Algebra

- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?
  1. Group `Courses` by `DeptName`.
  2. For each group, create a new attribute that stores the number of classes taught by the department.
  3. For each group, create a new attribute that stores the total enrollment of the classes taught by the department.
- γ_L(`Courses`), where L is a list containing three elements:
  1. `DeptName`: the *grouping attribute*,
  2. `COUNT(Number)` $\rightarrow$ `NumCourses`: an *aggregated attribute* computing the count of the `Number` attribute in each group and naming the new attribute `NumCourses`, and
  3. `SUM(Enrollment)` $\rightarrow$ `TotalEnrollment`: an aggregated attribute computing the total of the `Enrollment` attribute and naming the new attribute `TotalEnrollment`.

Example of Grouping Continued

- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?

Example of Grouping Continued

- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?
- The complete operator is
  $\gamma_{\texttt{DeptName},\texttt{COUNT(Number)}\rightarrow\texttt{NumCourses},\texttt{SUM(Enrollment)}\rightarrow\texttt{TotalEnrollment}}(\texttt{Courses})$
- The schema of the new relation is (DeptName, NumCourses, TotalEnrollment).

Example of Grouping Continued

- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?
- The complete operator is
  $\gamma_{\texttt{DeptName},\texttt{COUNT(Number)}\rightarrow\texttt{NumCourses},\texttt{SUM(Enrollment)}\rightarrow\texttt{TotalEnrollment}}(\texttt{Courses})$
- The schema of the new relation is (DeptName, NumCourses, TotalEnrollment).
- We can group by multiple attributes.
- We can create as many new attributes as necessary.
- We can apply other operators to the result, since the grouping operator produces a relation.

Grouping in SQL

- Syntax is much simpler than relational algebra.
- Use the `GROUP BY` clause *after* the `WHERE` clause or after the `FROM`, if there is no `WHERE` clause.
- List grouping attributes after `GROUP BY`.
- Use `SELECT` clause to aggregate attributes.

Grouping in SQL

- Syntax is much simpler than relational algebra.
- Use the `GROUP BY` clause *after* the `WHERE` clause or after the `FROM`, if there is no `WHERE` clause.
- List grouping attributes after `GROUP BY`.
- Use `SELECT` clause to aggregate attributes.
- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?

Grouping in SQL

- Syntax is much simpler than relational algebra.
- Use the `GROUP BY` clause *after* the `WHERE` clause or after the `FROM`, if there is no `WHERE` clause.
- List grouping attributes after `GROUP BY`.
- Use `SELECT` clause to aggregate attributes.
- How do we answer the query "Count the number of classes and total enrollment of the classes each department teaches"?

```
SELECT DeptName, COUNT(Number) AS NumCourses,
       SUM(Enrollment) AS TotalEnrollment
FROM COURSES
GROUP BY DeptName;
```

Grouping in SQL

```
SELECT DeptName, COUNT(Number) AS NumCourses,
       SUM(Enrollment) AS TotalEnrollment
FROM COURSES
GROUP BY DeptName;
```

- Aggregated attributes are evaluated on a per-group basis.
- Only attributes mentioned in the `GROUP BY` clause may appear unaggregated in the `SELECT` clause, e.g., `Number` must have an aggregation operator applied to it.
- There need not be any aggregated attribute in the `SELECT` clause.
- Read Chapter 6.4.6 of the textbook about affect of `NULL` values on grouping and aggregation.

Restricting Grouping in SQL

- How do we answer the query "Count the number of classes each department teaches, restricted to departments that have total enrollment at least 500 in their classes (the classes taught by that department)"?

Restricting Grouping in SQL

- How do we answer the query "Count the number of classes each department teaches, restricted to departments that have total enrollment at least 500 in their classes (the classes taught by that department)"?
- Need to introduce the `HAVING` clause

```
SELECT DeptName, COUNT(Number) AS NumCourses
FROM COURSES
GROUP BY DeptName
HAVING SUM(Enrollment) >= 500;
```

Rules for HAVING Clauses

- An aggregation in a HAVING clause applies only to the group being tested.
- If an attribute appears unaggregated in a HAVING clause, it must appear in the GROUP BY line.

Complete `SELECT` Statement

```
SELECT Attribute list
FROM Relation list
WHERE Condition or Subquery
GROUP BY Attribute list
HAVING Condition or Subquery
ORDER BY Attribute list;
```

Complete `SELECT` Statement

`SELECT` Attribute list
`FROM` Relation list
`WHERE` Condition or Subquery
`GROUP BY` Attribute list
`HAVING` Condition or Subquery
`ORDER BY` Attribute list;

- `WHERE` is evaluated *before* `GROUP BY` and `HAVING`.

Joins in Relational Algebra and SQL

- Cross product:

  RA $R \times S$

  SQL R `CROSS JOIN` S;

- Theta join:

  RA $R \underset{C}{\bowtie} S$

  SQL R `JOIN` S `ON` C;

- Natural join:

  RA $R \bowtie S$

  SQL R `NATURAL JOIN` S;

Outer Joins

- A *dangling tuple* is one that fails to pair with any tuple in the other relation in a join operation.
- Outer joins allow dangling tuples to be included in the result of join operations, by padding them with `NULL` values.

Outer Joins

- A *dangling tuple* is one that fails to pair with any tuple in the other relation in a join operation.
- Outer joins allow dangling tuples to be included in the result of join operations, by padding them with `NULL` values.

  RA $R \overset{\circ}{\bowtie} S$

  SQL R `NATURAL FULL OUTER JOIN` S;

  - Contains all tuples in $R \bowtie S$.
  - Includes every tuple in R that is not joined with a tuple in S, after padding a special null symbol $\perp$ (`NULL` in case of SQL).
  - Same condition applied to S.

Outer Joins

- A *dangling tuple* is one that fails to pair with any tuple in the other relation in a join operation.
- Outer joins allow dangling tuples to be included in the result of join operations, by padding them with NULL values.

  RA $R \overset{\circ}{\bowtie} S$

  SQL R `NATURAL FULL OUTER JOIN` S;

  - Contains all tuples in $R \bowtie S$.
  - Includes every tuple in R that is not joined with a tuple in S, after padding a special null symbol $\perp$ (NULL in case of SQL).
  - Same condition applied to S.
- Left outer join:

  RA $R \overset{\circ}{\bowtie}_L S$

  SQL R `NATURAL LEFT OUTER JOIN` S;

  - Like $R \overset{\circ}{\bowtie} S$ but ignores dangling tuples in S.

Outer Joins

- A *dangling tuple* is one that fails to pair with any tuple in the other relation in a join operation.
- Outer joins allow dangling tuples to be included in the result of join operations, by padding them with NULL values.

  RA $R \overset{\circ}{\bowtie} S$

  SQL R `NATURAL FULL OUTER JOIN` S;

  - Contains all tuples in $R \bowtie S$.
  - Includes every tuple in R that is not joined with a tuple in S, after padding a special null symbol $\perp$ (NULL in case of SQL).
  - Same condition applied to S.
- Left outer join:

  RA $R \overset{\circ}{\bowtie}_L S$

  SQL R `NATURAL LEFT OUTER JOIN` S;

  - Like $R \overset{\circ}{\bowtie} S$ but ignores dangling tuples in S.
- Right outer join is analogous to left outer join.
- All outerjoin operators have theta-join analogues.