Extended Relational Algebra

Bags

- A **bag** is like a set, but an element may appear more than once.
  - *Multiset* is another name for “bag.”
- Example:
  - {1,2,1,3} is a bag.
  - {1,2,3} is also a bag that happens to be a set.
- Bags also resemble lists, but order in a bag is unimportant.
  - Example:
    - {1,2,1} = {1,1,2} as bags, but
    - [1,2,1] != [1,1,2] as lists.

Why bags?

- SQL is actually a bag language.

- SQL will eliminate duplicates, but usually only if you ask it to do so explicitly
  - except for **union**, **intersection**, and **difference** where the default is “set mode”.

Union, **intersection**, and **difference** need new definitions for bags.

Bag Union

- An element appears in the **union** of two bags the **sum** of the number of times it appears in each bag.

- Example:

  $\{1,2,1\} \cup \{1,1,2,3,1\}$

  $= \{1,1,1,1,1,2,2,3\}$

Bag Intersection

- An element appears in the **intersection** of two bags the **minimum** of the number of times it appears in either.

- Example:

  $\{1,2,1\} \cap \{1,2,3\}$

  $= \{1,2\}$.

Bag Difference

- An element appears in **difference** *A* – *B* of bags as many times as it appears in *A*, **minus** the number of times it appears in *B*.
  - But never less than 0 times.

- Example: {1,2,1} – {1,2,3}
  = {1}.

Union, Intersection, Difference in SQL

Remember, we need to have the same schema for the relations that we union, intersect, or take difference.

```
(SELECT * FROM R)
  UNION
(SELECT * FROM S);

(SELECT * FROM R)
  INTERSECT
(SELECT * FROM S);

(SELECT * FROM R)
  EXCEPT
(SELECT * FROM S);
```

- Add “ALL” for bag version of these operators.
  - These are the only operators that work in ‘set mode’ by default.
  - All the others work in ‘bag mode’ by default.

Extended Relational Algebra

1. δ: eliminates duplicates from bags.

2. τ: sorts tuples.

3. **Extended** π: arithmetic, duplication of columns.

4. γ: grouping and aggregation.

5. **OUTERJOIN** $\overset{\circ}{\bowtie}$: superset of join.

Example: Duplicate Elimination

$R_1 := \delta(R_2)$

- R_1 consists of one copy of each tuple that appears in R_2 one or more times.

R =

A	B
1	2
3	4
1	2

SELECT **DISTINCT** *
FROM R;

$\delta(R)$ =

A	B
1	2
3	4

Sorting

$R_1 := \tau_L\ (R_2)$.

L is a list of some of the attributes of R_2.

- R_1 is the list of tuples of R_2 sorted first on the value of the first attribute on L, then on the second attribute of L, and so on.

- τ is the only operator whose result is neither a set nor a bag, but a **list**.

Example:

SELECT *
FROM R
ORDER BY A, B;

Example: Extended Projection

Using the same π_L operator, we allow the list L to contain arbitrary expressions involving attributes

Example:

```
SELECT
        A+B AS C,
        A  AS A1,
        A  AS A2
FROM R;
```

R =

A	B
1	2
3	4

$\pi_{A+B \to C, A \to A1, A \to A2}(R)$ =

C	A1	A2
3	1	1
7	3	3

Aggregation Operators

- They apply to entire columns of a table and produce a single result.

- Most important examples:
  - SUM
  - AVG
  - COUNT
  - MIN
  - MAX

Example: Aggregation

R =

A	B
1	3
3	4
3	2

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
MIN(B) = 2
AVG(B) = 3

```
SELECT SUM(A), COUNT(A), MAX(B), MIN(B), AVG(B)
FROM R;
```

Grouping Operator

$R_1 := \gamma_L (R_2)$

L is a list of elements that are either:

1. Individual (*grouping*) attributes.
2. AGG(A), where AGG is one of the aggregation operators and A is an attribute.

R =

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

$\gamma_{A,B,\mathrm{AVG}(C)}$ (R) = ??

First, group R :

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

Then, average C within groups:

A	B	AVG(C)
1	2	4
4	5	6

```
SELECT A,B,AVG(C)
FROM R
GROUP BY A,B;
```

γ_L(R) - Formally

- Group relation *R* according to all the grouping attributes on list *L*.
  - That is, form one group **for each distinct list** of values for those attributes in ***R***.

- Within each group, compute AGG(*A*) for each aggregation on list *L*.

- Result has **grouping attributes** and **aggregations** as attributes.
  - One tuple for each list of values for the grouping attributes **and** their group's aggregations.

Example: Grouping/Aggregation

StarsIn(title, year, starName**)**

How many movies each star has starred in?

What's the earliest year each star has starred in some movie?

How many stars have starred in in each movie?

Example: Grouping/Aggregation

StarsIn(title, year, starName**)**

How many movies each star has starred in?

$\gamma_{\text{starName, COUNT(title)}}$ (StarsIn)

What's the earliest year each star has starred in some movie?

$\gamma_{\text{starName, MIN(year)}}$ (StarsIn)

How many stars have starred in in each movie?

$\gamma_{\text{title, year, COUNT(starname)}}$ (StarsIn)

Example: Grouping/Aggregation

StarsIn(title, year, starName**)**

For each star who has appeared in at least three movies give the earliest year in which he or she appeared.

First we group, using **starName** as a grouping attribute.
Then, we compute the **MIN(year)** for each group.
Also, we need to compute the **COUNT(title)** aggregate for each group, for filtering out those stars with less than three movies.

Example: Grouping/Aggregation

StarsIn(title, year, starName**)**

For each star who has appeared in at least three movies give the earliest year in which he or she appeared.

$$\pi_{starName,minYear}(\sigma_{ctTitle \geq 3}(\gamma_{starName,MIN(year) \rightarrow minYear,COUNT(title) \rightarrow ctTitle}(StarsIn)))$$

First we group, using **starName** as a grouping attribute.
Then, we compute the **MIN(year)** for each group.
Also, we need to compute the **COUNT(title)** aggregate for each group, for filtering out those stars with less than three movies.

Example: Grouping/Aggregation

Translating the previous RA expression to SQL:

```
SELECT starName, miny
FROM (SELECT starname, COUNT(title) AS cnt,
MIN(year) AS miny
        FROM StarsIn
        GROUP BY starname)
WHERE cnt>=3;
```

Or (more concisely):

```
SELECT starname, MIN(year) AS miny
FROM StarsIn
GROUP BY starname
HAVING COUNT(title)>=3;
```

Problems

Product(maker, model, type)

PC(model, speed, ram, hd, rd, price)

Laptop(model, speed, ram, hd, screen, price)

Printer(model, color, type, price)

Find the manufacturers who sell exactly three different models of PC.

Find those manufacturers of at least two different computers (PC or Laptops) with speed of at least 700.

Outerjoin

Motivation

- Suppose we join $R \bowtie S$.
- A tuple of *R* which doesn't join with any tuple of *S* is said to be dangling.
  - Similarly for a tuple of *S*.
  - **Problem**: We loose dangling tuples.

Outerjoin

- Preserves dangling tuples by padding them with **NULL** in the result.

Example: Outerjoin

R =

A	B
1	2
4	5

S =

B	C
2	3
6	7

(1,2) joins with (2,3), but the other two tuples are dangling.

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

A	B	C
1	2	3
4	5	NULL
NULL	6	7

```
SELECT *
FROM R FULL OUTER JOIN S USING(B);
```

Exercises

- $R(A,B) = \{(0,1), (2,3), (0,1), (2,4), (3,4)\}$
- $S(B,C) = \{(0,1), (2,4), (2,5), (3,4), (0,2), (3,4)\}$

- $\gamma_{A,SUM(B)}(R)$

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

- $R \overset{\circ}{\bowtie}_L S$ -- This left outerjoin: Only pad dangling tuples from the left table.

- $R \overset{\circ}{\bowtie}_R S$ -- -- This right outerjoin: Only pad dangling tuples from the right table.