These lecture notes include some material from Professors Bertossi, Kolaitis, Guagliardo, Vardi, Libkin, Barland, McMahan

Multisets and Aggregation

Lecture Handout 7

Dr Evgenia Ternovska
Associate Professor

Simon Fraser University

Spring 2018

Duplicates

R

A	B
a1	b1
a2	b2
a1	b2

$\pi_A(R)$

A
a1
a2

SELECT A **FROM** R

A
a1
a2
a1

- We considered relational algebra on **sets**
- SQL uses **bags**: sets with duplicates

Multisets (a.k.a. bags)

Sets where the **same element** can occur **multiple times**

The number of occurrences of an element is called its multiplicity

Notation

$a \in_k B$: a occurs k times in bag B

$a \in B$: a occurs in B with multiplicity ≥ 1

$a \notin B$: a does not occur in B (that is, $a \in_0 B$)

Relational algebra on bags

Relations are **bags of tuples**

Projection

Keeps duplicates

$$\pi_A \left(\begin{array}{cc} \textbf{A} & \textbf{B} \\ \hline 2 & 3 \\ 1 & 1 \\ 2 & 2 \end{array} \right) = \begin{array}{c} \textbf{A} \\ \hline 2 \\ 1 \\ 2 \end{array}$$

Relational algebra on bags

Cartesian product

Concatenates tuples as many times as they occur

A	B
1	1

$\times$

C
2
2

$=$

A	B	C
1	1	2
1	1	2

Relational algebra on bags

Selection

Takes all occurrences of tuples satisfying the condition:

$$\text{If } \bar{a} \in_k R, \quad \text{then } \begin{cases} \bar{a} \in_k \sigma_\theta(R) & \text{if } \bar{a} \models \theta \\ \bar{a} \notin \ \sigma_\theta(R) & \text{otherwise} \end{cases}$$

Example

$$\sigma_{A>1} \left(\begin{array}{cc} \textbf{A} & \textbf{B} \\ \hline 2 & 3 \\ 1 & 2 \\ 2 & 3 \end{array} \right) = \begin{array}{cc} \textbf{A} & \textbf{B} \\ \hline 2 & 3 \\ & \\ 2 & 3 \end{array}$$

Relational algebra on bags

Duplicate elimination ε

New operation that removes duplicates:

$$\text{If } \bar{a} \in R, \quad \text{then } \bar{a} \in_1 \varepsilon(R)$$

Example

$$\varepsilon\left(\begin{array}{cc} \mathbf{A} & \mathbf{B} \\ \hline 2 & 3 \\ 1 & 2 \\ 2 & 3 \end{array}\right) = \begin{array}{cc} \mathbf{A} & \mathbf{B} \\ \hline 2 & 3 \\ 1 & 2 \end{array}$$

Relational algebra on bags

Union

Adds multiplicities:

$$\text{If } \bar{a} \in_k R \text{ and } \bar{a} \in_n S, \quad \text{then } \bar{a} \in_{k+n} R \cup S$$

Example

$$\begin{array}{cc} \mathbf{A} & \mathbf{B} \\ \hline 1 & 2 \\ 1 & 2 \\ 1 & 3 \end{array} \cup \begin{array}{cc} \mathbf{A} & \mathbf{B} \\ \hline 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{array} = \begin{array}{cc} \mathbf{A} & \mathbf{B} \\ \hline 1 & 2 \\ 1 & 2 \\ 1 & 3 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{array}$$

Relational algebra on bags

Intersection

Takes the **minimum** multiplicity:

$$\text{If } \bar{a} \in_k R \text{ and } \bar{a} \in_n S, \quad \text{then } \bar{a} \in_{\min\{k,n\}} R \cap S$$

Example

A	B
1	2
1	2
1	3

$\cap$

A	B
1	2
1	3
1	4

$=$

A	B
1	2
1	3

Relational algebra on bags

Difference

Subtracts multiplicities up to zero:

$$\text{If } \bar{a} \in_k A \text{ and } \bar{a} \in_n B, \text{ then } \begin{cases} \bar{a} \in_{k-n} A - B & \text{if } k > n \\ \bar{a} \notin \quad\ A - B & \text{otherwise} \end{cases}$$

Example

A	B
1	2
1	2
1	3

$-$

A	B
1	2
1	3
1	3
1	4

$=$

A	B
1	2

RA on sets vs. RA on bags

Equivalences of RA on sets **do not** necessarily hold on **bags**

Example

On bags $\sigma_{\theta_1 \vee \theta_2}(R) \neq \sigma_{\theta_1}(R) \cup \sigma_{\theta_2}(R)$

R	**A**
	2

$\sigma_{A>1 \vee A<3}(R)$	**A**
	2

$\sigma_{A>1}(R) \cup \sigma_{A<3}(R)$	**A**
	2
	2

$\varepsilon\big(\sigma_{\theta_1 \vee \theta_2}(R)\big) = \varepsilon\big(\sigma_{\theta_1}(R) \cup \sigma_{\theta_2}(R)\big)$ holds

Basic SQL queries revisited

$$
\begin{aligned}
Q := {} & \texttt{SELECT} \; [\texttt{DISTINCT}] \; \alpha \; \texttt{FROM} \; \tau \; \texttt{WHERE} \; \theta \\
& \mid Q_1 \; \texttt{UNION} \; [\texttt{ALL}] \; Q_2 \\
& \mid Q_1 \; \texttt{INTERSECT} \; [\texttt{ALL}] \; Q_2 \\
& \mid Q_1 \; \texttt{EXCEPT} \; [\texttt{ALL}] \; Q_2
\end{aligned}
$$

SQL and RA on bags

SQL	RA on bags
SELECT α ...	$\pi_\alpha(\cdot)$
SELECT DISTINCT α ...	$\varepsilon\big(\pi_\alpha(\cdot)\big)$
Q_1 **UNION ALL** Q_2	$Q_1 \cup Q_2$
Q_1 **INTERSECT ALL** Q_2	$Q_1 \cap Q_2$
Q_1 **EXCEPT ALL** Q_2	$Q_1 - Q_2$
Q_1 **UNION** Q_2	$\varepsilon(Q_1 \cup Q_2)$
Q_1 **INTERSECT** Q_2	$\varepsilon(Q_1 \cap Q_2)$
Q_1 **EXCEPT** Q_2	$\varepsilon(Q_1) - \varepsilon(Q_2)$

Duplicates and aggregation (1)

Customer

ID	Name	City	Age
1	John	Edinburgh	31
2	Mary	London	37
3	Jane	London	22
4	Jeff	Cardiff	22

Average age of customers: **avg**$\big(\pi_{\text{Age}}(\text{Customer})\big)$

- If we remove duplicates we get $\frac{31+37+22}{3} = 30$ **(wrong)**

SQL keeps duplicates by default:

```
SELECT AVG(age)
FROM   Customer ;
```

Duplicates and aggregation (2)

Account

Number	Branch	CustID	Balance
111	London	1	1330.00
222	London	2	1756.00
333	Edinburgh	1	450.00

Number of branches: $\left|\varepsilon\left(\pi_{\text{Branch}}(\text{Account})\right)\right|$

- If we keep duplicates we get 3 **(wrong)**

In SQL:

```
SELECT COUNT(DISTINCT branch)
FROM   Account ;
```

Aggregate functions in SQL

COUNT	number of elements in a column
AVG	average value of elements in a column
SUM	adds up all elements in a column
MIN	minimum value of elements in a column
MAX	maximum value of elements in a column

- Using **DISTINCT** with **MIN** and **MAX** makes no difference
- **COUNT**(*) counts all rows in a table
- **COUNT**(**DISTINCT** *) is **illegal**

  To count all distinct rows of a table T use

```
SELECT COUNT(DISTINCT T.*)
FROM   T ;
```

Aggregation and empty tables

Suppose table T has a column (of numbers) called A

```
SELECT MIN(A),MAX(A),AVG(A),SUM(A),COUNT(A),COUNT(*)
FROM   T
WHERE  1=2 ;
```

```
 min | max | sum | avg | count | count
-----+-----+-----+-----+-------+-------
     |     |     |     |     0 |     0
```

Grouping

Account

Num	Branch	CID	Balance
111	London	1	1330.00
222	London	2	1756.00
333	Edinburgh	1	450.00

How much money does **each** customer have in total **across all of his/her accounts**?

Idea

1. Partition Account into **groups** (one per customer) of rows
2. Sum balances in each group separately
3. Take the union of the results for each group

Account

Num	Branch	CID	Balance
111	London	1	1330.00
222	London	2	1756.00
333	Edinburgh	1	450.00

Num	Branch	CID	Balance
111	London	1	1330.00
333	Edinburgh	1	450.00

Num	Branch	CID	Balance
222	London	2	1756.00

CID	SUM
1	1780.00

CID	SUM
2	1756.00

CID	SUM
1	1780.00
2	1756.00

Grouping in SQL

Account

Number	Branch	CustID	Balance
111	London	1	1330.00
222	London	2	1756.00
333	Edinburgh	1	450.00

How much money does **each** customer have in total **across all of his/her accounts**?

```
SELECT    A.custid, SUM(A.balance)
FROM      Account A
GROUP BY  A.custid ;
```

Answer:

CustID	SUM
1	1780.00
2	1756.00

Account

Number	Branch	CustID	Balance
111	London	1	1330.00
222	London	2	1756.00
333	Edinburgh	1	450.00

CustID	Balance
1	1330.00
1	450.00

CustID	Balance
2	1756.00

CustID	SUM
1	1780.00

CustID	SUM
2	1756.00

CustID	SUM
1	1780.00
2	1756.00

Grouping in SQL: Another example

Account

Number	Branch	CustID	Balance
111	London	1	1330.00
222	London	2	1756.00
333	Edinburgh	1	450.00

How much money is there in **total in each branch**?

```
SELECT    A.branch, SUM(A.balance)
FROM      Account A
GROUP BY  A.branch ;
```

Answer:

Branch	SUM
London	3086.00
Edinburgh	450.00

Account

Number	Branch	CustID	Balance
111	London	1	1330.00
222	London	2	1756.00
333	Edinburgh	1	450.00

Branch	Balance
London	1330.00
London	1756.00

Branch	Balance
Edinburgh	450.00

Branch	SUM
London	3086.00

Branch	SUM
Edinburgh	450.00

Branch	SUM
London	3086.00
Edinburgh	450.00

Filtering based on aggregation

Account

Number	Branch	CustID	Balance
111	London	1	1330.00
222	London	2	1756.00
333	Edinburgh	1	450.00

Branches with a total balance (across accounts) of at least 500?

```
SELECT     A.branch, SUM(A.balance)
FROM       Account A
GROUP BY   A.branch
HAVING     SUM(A.balance) >= 500 ;
```

Answer:

Branch	SUM
London	3086.00

Order of evaluation

1. Take rows from the (joined) tables listed in **FROM**
2. Discard rows not satisfying the **WHERE** condition
3. Partition rows according to attributes in **GROUP BY**
4. **Compute aggregates**
5. Discard rows not satisfying the **HAVING** condition
6. Output the values of expressions listed in **SELECT**

Aggregation and arithmetic (1)

Account

Number	Branch	CustID	Balance	Spend
111	London	1	1330.00	250.00
222	London	2	1756.00	356.00
333	Edinburgh	1	450.00	0.00

Money available in total to each customer across his/her accounts

```
SELECT    A.custid, SUM(A.balance - A.spend)
FROM      Account A
GROUP BY  A.custid
```

Answer:

CustID	SUM
1	1530.00
2	1400.00

Account

Number	Branch	CustID	Balance	Spend
111	London	1	1330.00	250.00
222	London	2	1756.00	356.00
333	Edinburgh	1	450.00	0.00

CustID	Balance	Spend
1	1330.00	250.00
1	450.00	0.00

CustID	Balance	Spend
2	1756.00	356.00

CustID	Balance − Spend
1	1080.00
1	450.00

CustID	Balance − Spend
2	1400.00

CustID	SUM
1	1530.00

CustID	SUM
2	1400.00

CustID	SUM
1	1530.00
2	1400.00

Aggregation and arithmetic (2)

Account

Number	Branch	CustID	Balance	Spend
111	London	1	1330.00	250.00
222	London	2	1756.00	356.00
333	Edinburgh	1	450.00	0.00

Money available in total to each customer across his/her accounts

```
SELECT    A.custid, SUM(A.balance) - SUM(A.spend)
FROM      Account A
GROUP BY  A.custid
```

Answer:

CustID	?column?
1	1530.00
2	1400.00

Account

Number	Branch	CustID	Balance	Spend
111	London	1	1330.00	250.00
222	London	2	1756.00	356.00
333	Edinburgh	1	450.00	0.00

CustID	Balance	Spend
1	1330.00	250.00
1	450.00	0.00

CustID	Balance	Spend
2	1756.00	356.00

CustID	SUM	SUM
1	1780.00	250.00

CustID	SUM	SUM
2	1756.00	356.00

CustID	?column?
1	1530.00

CustID	?column?
2	1400.00

CustID	?column?
1	1530.00
2	1400.00

GROUP BY in RA on bags

Notation: $\gamma_L(\cdot)$, where L is a **list** of elements, each of which is either

- Attribute
- Aggregation operator, with a name of the result appended after $\rightarrow$, e.g. MIN(year) $\rightarrow$ $minYear$

Computing:

- Partition the tuples into *groups* according to all grouping attributes simultaneously.
- For each group, produce one tuple consisting of:
  - The grouping attributes' values for that group,
  - The aggregations, over all tuples of that group, for the aggregated attributes on list L.