Week 2-1

Relational Algebra and SQLite

Big Data

Prof. Hwanjo Yu
POSTECH

Selection

- Returns all tuples which satisfy a condition

$$\sigma_c(R)$$

- Examples
  - $\sigma_{Salary > 40000}$ (Employee)
  - $\sigma_{Name = \text{"Smith"}}$ (Employee)

```
SELECT *
FROM Employee
WHERE Salary > 40000
```

- The condition c can be =, <, ≤ , >, ≥ , <>

Selection

Employee

SSN	Name	Salary
1234545	John	20000
5423341	Smith	60000
4352342	Fred	50000

```
SELECT *
FROM Employee
WHERE Salary > 40000
```

$\sigma_{\text{Salary} > 40000}$(Employee)

SSN	Name	Salary
5423341	Smith	60000
4352342	Fred	50000

Union

$R1 \cup R2$

```
SELECT * FROM R1
UNION ALL
SELECT * FROM R2
```

R1

A	B
a1	b1
a2	b1

$\cup$

R2

A	B
a1	b1
a3	b4

=

$R1 \cup R2$

A	B
a1	b1
a2	b1
a3	b4
a1	b1

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Difference

R1 – R2

```
SELECT * FROM R1
EXCEPT
SELECT * FROM R2
```

R1

A	B
a1	b1
a2	b1

–

R2

A	B
a1	b1
a3	b4

=

R1 – R2

A	B
a2	b1

What about intersection?

- Derived operator using minus

$$R1 \cap R2 = R1 - (R1 - R2)$$

- Derived using join (will explain later)

$$R1 \cap R2 = R1 \bowtie R2$$

Projection

- Eliminates columns

$$\boldsymbol{\pi}_{\mathbf{A1,\ldots,An}}(\mathrm{R})$$

SELECT SSN, Name
FROM Employee

- Example: project social-security number and names:
  - $\pi_{SSN,Name}(\text{Employee})$

Projection

Employee

SSN	Name	Salary
1234545	John	20000
5423341	John	60000
4352342	John	20000

$\pi_{\text{Name,Salary}}$ (Employee)

Name	Salary
John	20000
John	60000
John	20000

```
SELECT Name, Salary
FROM Employee
```

Name	Salary
John	20000
John	60000

Set semantics

Which is more efficient?

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Cross product

- Each tuple in R1 with each tuple in R2

$$R1 \times R2$$

- Traditionally rare in practice, but can come up in analytics
- "Find all pairs of similar images/tweets/songs"
  - Compute the cross product, then compute a similarity function $f(x1,x2)$ for every possible pair

Cross product

Employee

Name	SSN
John	999999999
Tony	777777777

Dependent

EmpSSN	DepName
999999999	Emily
777777777	Joe

Employee $\times$ Dependent

Name	SSN	EmpSSN	DepName
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

Equi-join

$$R1 \bowtie_{A=B} R2 \;=\; \sigma_{A=B}\,(R1 \times R2)$$

```
SELECT *
FROM R1, R2
WHERE R1.A = R2.B
```

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

- Two ways to "spell" the same query
- The optimizer doesn't care about the syntax you use;
  it's going to work on the algebraic representation anyway.
- Sometimes one syntax or the other is more convenient.

Theta-join

- A join that involves a predicate

$$R1 \bowtie_{\theta} R2 = \sigma_{\theta}\,(R1 \times R2)$$

- Here θ can be any condition

Note that equi-join is a special case of theta-join where θ is an equality condition

Examples of theta-joins

- Find all hospitals within 5 miles of a school

$$\pi_{\text{name}}(\text{Hospitals} \bowtie_{\text{distance(location,location)}<5} \text{Schools})$$

```
SELECT DISTINCT h.name
FROM Hospitals h, Schools s
WHERE distance(h.location, s.location) < 5
```

Examples of theta-joins

- Find all user clicks made within 5 seconds of a page load

$$\text{Clicks} \bowtie_{\text{abs(click\_time}-\text{load\_time)}<5} \text{PageLoads}$$

```
SELECT *
FROM Clicks c, PageLoads p
WHERE abs(c.click_time - p.load_time) < 5
```

Outer joins

- Outer join
  - Include tuples with no matches in the output
  - Use NULL values for missing attributes
- Variants
  - Left outer join $⟕$
  - Right outer join $⟖$
  - Full outer join $⟗$

Outer join example

Patient P

age	zip	disease
54	98125	heart
20	98120	flu
33	98120	lung

Job J

job	age	zip
lawyer	54	98125
cashier	20	98120

P ⟕ J

age	zip	disease	job
54	98125	heart	lawyer
20	98120	flu	cashier
33	98120	lung	null

SQLite exercise

1. Search, download, and install SQLite
2. Read “About SQLite”
3. Read “Features of SQLite”
4. Read “Appropriate Uses for SQLite” => “Data analysis”

SQLite exercise

- ".databases", ".open temp.db"
- create table Bank(Account int, Branch text, Country text, Balance int);
- insert into Bank values(100090, "Branch", "Country", 10000);
- "select * from Bank" with column mode, csv mode, header on, and header off
- ".exit", "sqlite3 temp.db", ".tables", ".schema"
- ".header on", ".mode csv", ".once temp.txt", "select.."
- ".import temp.txt Bank2", ".tables", ".schema"
  - When no defined table, the first row will be the column names and all values will be read as text
- ".import temp.txt Bank", "select * from Bank"
  - Create table before import to define attribute types of data
- "select distinct * from Bank"
- "delete from Bank2 where Account=100090 and Branch="Seoul";

Bank

Account	Branch	Country	Balance
100090	Seoul	Korea	$10,000
100092	Seoul	Korea	$5,000
100100	Busan	Korea	$9,000
200010	New York	USA	$20,000

- create table Bank3(Account int, …);
- insert into Bank3 select distinct * from Bank;
- drop table Bank2;
- alter table Bank3 rename to Bank2;

- Try "union (all)" Bank and Bank2
- Try "except" Bank2 from Bank
- Try equi-join and theta-join

Aggregation: GROUP BY…

- Five aggregation function: (1) sum, (2) count, (3) average, (4) maximum, and (5) minimum

Bank

Account	Branch	Country	Balance
100090	Seoul	Korea	$10,000
100100	Busan	Korea	$9,000
200010	New York	USA	$20,000
…	…	…	…

- "Find the maximum Balance of each Branch"

$$_{\mathbf{Branch}}\mathbf{G}_{\mathbf{max(Balance)}}(\mathbf{Bank})$$

```
SELECT Branch, max(Balance)
FROM Bank
GROUP BY Branch
```

POSTECH
POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Aggregation: GROUP BY… HAVING…

- Five aggregation function: (1) sum, (2) count, (3) average, (4) maximum, and (5) minimum

Bank

Account	Branch	Country	Balance
100090	Seoul	Korea	\$10,000
100100	Busan	Korea	\$9,000
200010	New York	USA	\$20,000
…	…	…	…

- "Find the balance sum of each branch whose sum is larger than \$10,000"

$$\boldsymbol{\sigma}_{\mathbf{sum(Balance)>10,000}}(\ {}_{\mathbf{Branch}}\mathbf{G}_{\mathbf{sum(Balance)}}(\mathbf{Bank})\ )$$

SELECT Branch, sum(Balance)

FROM Bank

GROUP BY Branch

HAVING sum(Balance) > 10,000

Sort: ORDER BY

- Five aggregation function: (1) sum, (2) count, (3) average, (4) maximum, and (5) minimum

Bank

Account	Branch	Country	Balance
100090	Seoul	Korea	$10,000
100100	Busan	Korea	$9,000
200010	New York	USA	$20,000
…	…	…	…

- "Output the balance sum of the Korea branches whose sum is larger than $10,000 in an ascending order"

$$\boldsymbol{\sigma}_{\mathbf{sum(Balance)>10,000}}\left({}_{\mathbf{Branch}}\mathbf{G}_{\mathbf{sum(Balance)}}\left(\sigma_{\mathbf{Country=Korea}}(\mathbf{Bank})\right)\right)$$

Where is sorting?

SELECT Branch, sum(Balance)

FROM Bank

WHERE Country = "Korea"

GROUP BY Branch

HAVING sum(Balance) > 10,000

ORDER BY sum(Balance) ASC

SQL exercise

Bank

Account	Branch	Country	Balance
100090	Seoul	Korea	$10,000
100100	Busan	Korea	$9,000
200010	New York	USA	$20,000
100091	Seoul	Korea	$9,100

- “Find the number of Account for each Branch in each Country”

SELECT Country, Branch, count(*)

FROM Bank

GROUP BY Country, Branch

- “Output the Balance sum of each Country in a descending order”

SELECT Country, sum(Balance)

FROM Bank

GROUP BY Country

ORDER BY sum(Balance) DESC