SQL Queries – misc. and relational algebra

Overview of this video

Provide a better understanding of SQL queries as a whole, instead of as parts

(Relational algebra does focus on parts, but should let you better see how the parts compose, which is the point)

SQL Queries

Queries in SQL have the following form:

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

This video

SQL Queries

Queries in SQL have the following form:

6. SELECT
1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. ORDER BY

No. = order of execution

This video

Views: as saved queries

Intuitively, saved queries or virtual tables

Employees

birthday	first_name	family_name	e_id
1990-11-10	Anne	Smith	1
2000-02-05	David	Jones	2
1995-05-09	William	Taylor	3

```
SELECT first_name, e_id, COUNT(t_id)
FROM Employees NATURAL JOIN Transactions
GROUP BY first_name, e_id;
```

Transactions

t_id	c_id	e_id
1	3	1
2	6	1
3	19	3

first_name	e_id	COUNT(t_id)
Anne	1	2
William	3	1

Views: as saved queries

Intuitively, saved queries or virtual tables

```
CREATE VIEW Employee_transaction_count AS
SELECT first_name, e_id, COUNT(t_id)
FROM Employees NATURAL JOIN Transactions
GROUP BY first_name, e_id;
```

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

Transactions

t_id	c_id	e_id
1	3	1
2	6	1
3	19	3

Employee_transaction_count

first_name	e_id	COUNT(t_id)
Anne	1	2
William	3	1

Views: as virtual tables

Intuitively, saved queries or virtual tables

```
CREATE VIEW Employee_transaction_count AS
SELECT first_name, e_id, COUNT(t_id)
FROM Employees NATURAL JOIN Transactions
GROUP BY first_name, e_id;
```

Employee_transaction_count

first_name	e_id	COUNT(t_id)
Anne	1	2
William	3	1

```
SELECT first_name, COUNT(t_id)
FROM Employee_transaction_count;
```

first_name	COUNT(t_id)
Anne	2
William	1

More on views

By default **data** in views can't be modified (i.e. updated, delete from or inserted into)

- Can be done with triggers, but that is not covered in this module

Views can be modified though:

```
CREATE OR REPLACE VIEW Employee_transaction_count AS
SELECT first_name, COUNT(t_id)
FROM Employees NATURAL JOIN Transactions
GROUP BY first_name, e_id;
```

```
CREATE VIEW Employee_transaction_count AS
SELECT first_name, e_id, COUNT(t_id)
FROM Employees NATURAL JOIN Transactions
GROUP BY first_name, e_id;
```

Or removed:

```
DROP VIEW Employee_transaction_count;
```

Relational Algebra

Algebra

Algebra = branch of math

Example: Algebra with numbers

- Addition (written: +) = function that takes two numbers and returns a number
- Logarithm (written: log) = function that takes one number and returns another
  - Has a subscript – the base
- …

Can be composed, e.g.: $\log_2 (3 + 8) + 4$

Relational algebra

Algebra with tables ≈ SQL SELECT queries

- Exception: uses set semantics (i.e. removes duplicates and has no order)

Relational algebra is crucial for optimization

- Later in the course

We have seen what tables are (i.e., the thing that in relational algebra corresponds to numbers in algebra with numbers) and we will next see 5 relational algebra functions (i.e. the thing that corresponds to + or log from algebra with numbers)

Projection (π)

$\pi_{\text{attribute list}}(\mathbf{R})$ = restricts **R** to the attributes in **attribute list**

SQL query

```
SELECT DISTINCT   family_name,
                  birthday
FROM Employees;
```

DISTINCT, due to sets

translates into

$\pi_{\text{family\_name,birthday}}(\textbf{Employees})$

Relational algebra expression

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

family_name	birthday
Smith	1990-11-10
Jones	2000-02-05
Taylor	1995-05-09

Renaming (ρ)

$\rho_{A1 \rightarrow B1, A2 \rightarrow B2, \ldots}(R)$ = renames attribute **A1** to **B1**, attribute **A2** to **B2**, …

SQL query

SELECT DISTINCT birthday **AS** bday**, first_name, family_name**
FROM Employees;

translates into

$\rho_{birthday \rightarrow bday}(\textbf{Employees})$

Relational algebra expression

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

bday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

Selection (σ)

The SELECT keyword in SQL has nothing to do with the selection operator!

$\sigma_{\text{condition}}(\mathbf{R})$ = set of all tuples in **R** that satisfy the **condition**

SQL query

```
SELECT DISTINCT *
FROM Employees
WHERE first_name='Anne';
```

translates into

$\sigma_{\text{first\_name='Anne'}}(\text{Employees})$

Relational algebra expression

Employees

birthday	first_name	family_name
1990-11-10	Anne	Smith
2000-02-05	David	Jones
1995-05-09	William	Taylor

birthday	first_name	family_name
1990-11-10	Anne	Smith

Cartesian Product (×)

$R_1 \times R_2$ = pairs each tuple in R_1 with each tuple in R_2

SQL query

```
SELECT DISTINCT *
FROM Modules, Lecturers;
```

translates into

Modules × Lecturers

Relational algebra expression

Modules

code	year	sem
COMP105	1	1
COMP201	2	1

Lecturers

name	module
John	COMP105
Sebastian	COMP201

code	year	sem	name	module
COMP105	1	1	John	COMP105
COMP105	1	1	Sebastian	COMP201
COMP201	2	1	John	COMP105
COMP201	2	1	Sebastian	COMP201

Natural Join (⋈)

$R_1 \bowtie R_2$ = pairs each tuple in R_1 with each tuple in R_2 with matching common attributes

```
SELECT DISTINCT *
FROM Employees NATURAL JOIN
Transactions;
```

translates into

Employees ⋈ Transactions

Relational algebra expression

Employees

birthday	first_name	family_name	e_id
1990-11-10	Anne	Smith	1
2000-02-05	David	Jones	2
1995-05-09	William	Taylor	3

Transactions

t_id	c_id	e_id
1	3	1
2	6	1
3	19	3

birthday	first_name	family_name	e_id	t_id	c_id
1990-11-10	Anne	Smith	1	1	3
1990-11-10	Anne	Smith	1	2	6
1995-05-09	William	Taylor	3	3	19

Many others…

Besides the mentioned, there are many others, e.g. for:

1. Left/right semi-join ($\ltimes$/$\rtimes$)
   - I.e. returns the rows from the left/right table that would be matched to a row on the other side in a Natural Join – it was covered in SQL queries - optional part
2. GROUP BY
   - Relational Algebra for GROUP BY is complex and the two books differ in how they define it (different syntax, symbols and so on) and overall, I feel it is sufficient to know that you CAN define it (both definitions writes down what the GROUP BY statements says and the various aggregates used) and leave it at that

Key understanding

The goal of showing you relational algebra now (instead of next to optimization) is that understanding how you can combine the different parts is easier:

- You can do it however you want!

Math example from earlier:

- $\log_2(3+8)+4$

Relational algebra example:

- $\underbrace{\rho_{a\rightarrow z}\underbrace{(R \times S)}}\bowtie T$

```
SELECT *
FROM    (SELECT a AS z, b, c, d
         FROM (SELECT *
                  FROM R, S))    NATURAL JOIN T;
```

Say the schema for R was R(a,b), for S was S(c,d) and for T was T(e,z)