Database Systems
Relational Algebra ; Advanced SQL ; Triggers ; Views

Week 3 – Lab

Basic SQL Relational Algebra Operations

Relational Algebra divided in various groups

- Unary Relational Operations
- Relational Algebra Operations From Set Theory
- Binary Relational Operations

Unary Relational Operations

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

Select (1/2)

- The SELECT operation is used for selecting a subset of the tuples according to a given selection condition. Sigma(σ)Symbol denotes it. It is used as an expression to choose tuples which meet the selection condition. Select operator selects tuples that satisfy a given predicate.
- $\sigma_p(r)$
- σ is the predicate
- r stands for relation which is the name of the table
- p is prepositional logic

Select (2/2)

For example:

- σsubject = "database"**(Books)**

Output – Selects tuples from books where subject is 'database'.

- σsubject = "database" and price = "450"**(Books)**

Output – Selects tuples from books where subject is 'database' and 'price' is 450.

- σsubject = "database" and price = "450" or year > "2010"**(Books)**

Output – Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Projection (1/2)

- The projection eliminates all attributes of the input relation but those mentioned in the projection list. The projection method defines a relation that contains a vertical subset of Relation.

- This helps to extract the values of specified attributes to eliminates duplicate values. (pi) symbol is used to choose attributes from a relation. This operator helps you to keep specific columns from a relation and discards the other columns.

Projection (2/2)

- Considering the following table:

CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive
4	Alibaba	Active

Here, the projection of CustomerName and status will give

- Π CustomerName, Status (Customers)

CustomerName	Status
Google	Active
Amazon	Active
Apple	Inactive
Alibaba	Active

Relational Algebra Operations

- UNION (υ)
- INTERSECTION ( ),
- DIFFERENCE (-)
- CARTESIAN PRODUCT (x)

Example Tables

DEPOSITOR RELATION

CUSTOMER_NAME	ACCOUNT_NO
Johnson	A-101
Smith	A-121
Mayes	A-321
Turner	A-176
Johnson	A-273
Jones	A-472
Lindsay	A-284

BORROW RELATION

CUSTOMER_NAME	LOAN_NO
Jones	L-17
Smith	L-23
Hayes	L-15
Jackson	L-14
Curry	L-93
Smith	L-11
Williams	L-17

Union

- Suppose there are two tuples R and S. The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by ∪.

Notation: R ∪ S

A union operation must hold the following condition:

- R and S must have the attribute of the same number.
- Duplicate tuples are eliminated automatically.

Input: ∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

CUSTOMER_NAME
Johnson
Smith
Hayes
Turner
Jones
Lindsay
Jackson
Curry
Williams
Mayes

Intersection

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.
- It is denoted by intersection ∩.

Notation: R ∩ S

Input: ∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

CUSTOMER_NAME
Smith
Jones

Difference

- Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).

Notation: R - S

Input: ∏ CUSTOMER_NAME (BORROW) - ∏ CUSTOMER_NAME (DEPOSITOR)

CUSTOMER_NAME
Jackson
Hayes
Willians
Curry

Cartesian Product (1/2)

- The Cartesian product is used to combine each row in one table with each row in the other table. It is also known as a cross product.
- It is denoted by X.

Notation: E X D

Input: EMPLOYEE X DEPARTMENT

Cartesian Product (2/2)

EMPLOYEE

EMP_ID	EMP_NAME	EMP_DEPT
1	Smith	A
2	Harry	C
3	John	B

DEPARTMENT

DEPT_NO	DEPT_NAME
A	Marketing
B	Sales
C	Legal

Output:

EMP_ID	EMP_NAME	EMP_DEPT	DEPT_NO	DEPT_NAME
1	Smith	A	A	Marketing
1	Smith	A	B	Sales
1	Smith	A	C	Legal
2	Harry	C	A	Marketing
2	Harry	C	B	Sales
2	Harry	C	C	Legal
3	John	B	A	Marketing
3	John	B	B	Sales
3	John	B	C	Legal

Join Operations

Join operation is essentially a cartesian product followed by a selection criterion.

Join operation denoted by ⋈.

JOIN operation also allows joining variously related tuples from different relations.

- Types of JOIN:

Inner Joins:

- Theta join
- EQUI join
- Natural join

Theta Join (1/2)

Theta join (or “conditional join”)

- Binary operator
- Results in all combinations of tuples in R and S that satisfy θ (where θ is a binary relational operator in the set {, ≥})
- Result schema same as that of cross-product
- In case the operator θ is the equality operator (=) then this join is also called an equijoin

Notation: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$

Theta Join (2/2)

$$STUDENT \bowtie_{Student.Std = Subject.Class} SUBJECT$$

SID
101
102
10
10
11
11

	Music
	Sports

Student_detail

SID	Name	Std	Class	Subject
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

EQUI join

- When Theta join uses only **equality** comparison operator, it is said to be equijoin.
- For example:

- $A \bowtie A.column\ 2 = B.column\ 2\ (B)$

Natural Join

- Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

Attendance

FirstName	Lastname	Course
John	Smith	Logic
John	Smith	DB
Leo	Abel	DB

Courses

CID	Course	Teacher
11	Logic	Pain
12	DB	White
13	English	Gray

Attendance ⋈ Courses

Firstname	Lastname	Course	CID	Teacher
John	Smith	Logic	11	Pain
John	Smith	DB	12	White
Leo	Abel	DB	12	White

Note: Joins can be incomplete or empty

Triggers (1/3)

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]

Triggers (2/3)

For example:

Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.

Field	Type	Null	Key	Default	Extra
tid	int(4)	NO	PRI	NULL	auto_increment
name	varchar(30)	YES		NULL	
subj1	int(2)	YES		NULL	
subj2	int(2)	YES		NULL	
subj3	int(2)	YES		NULL	
total	int(3)	YES		NULL	
per	int(3)	YES		NULL	

Triggers (3/3)

```
create trigger stud_marks
before INSERT
on
Student
for each row
set Student.total = Student.subj1 + Student.subj2 + Student.subj3, Student.per = Student.total * 60 / 100;
```

- insert into Student values(25, “Ivan", 20, 20, 20, 0, 0);

```
+-----+-------+-------+-------+-------+-------+------+
| tid | name  | subj1 | subj2 | subj3 | total | per  |
+-----+-------+-------+-------+-------+-------+------+
| 25 | Ivan |    20 |    20 |    20 |    60 |   36 |
+-----+-------+-------+-------+-------+-------+------+
```

Views (1/4)

- A view is like a “virtual” table
  - Defined by a query, which describes how to compute the view contents on the fly
  - DBMS stores the view definition query instead of view contents
  - Can be used in queries just like a regular table

Views(2/4)

- Example: members of Chess Club

```
CREATE VIEW ChessClub AS
SELECT * FROM User
WHERE uid IN (SELECT uid FROM Member
WHERE gid = 'chess');
```

Tables used in defining a view are called "base tables"

- User and Member above

Views in Queries (3/4)

- Example: find the average age of members in Chess Club

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

To process the query, replace the reference to the view by its definition

```
SELECT AVG(age)
FROM (SELECT * FROM User
    WHERE uid IN
      (SELECT uid FROM Member
      WHERE gid = 'chess'))
    AS ChessClub;
```

Why do we use Views? (4/4)

- To hide data from users
- To hide complexity from users
- Logical data independence
  - If applications deal with views, we can change the underlying schema without affecting applications
  - Recall physical data independence: change the physical organization of data without affecting applications
- To provide a uniform interface for different implementations or sources
- **Real database applications use multiple views**

Exercise 1

- Create the following query with it’s corresponding relational algebra/SQL for the following conditions:
  - Order countries by id asc, then show the 12th to 17th rows.
  - List all addresses in a city whose name starts with 'A’.
  - List all customers' first name, last name and the city they live in.
  - Find all customers with at least one payment whose amount is greater than 11 dollars.
  - Find all duplicated first names in the customer table.