Lecture 5

Relational Algebra & SQL

Relational Algebra

- The Relational Algebra is used to define the ways in which relations (tables) can be operated to manipulate their data.
- It is used as the basis of SQL for relational databases, and illustrates the basic operations required of any DML.
- This Algebra is composed of Unary operations (involving a single table) and Binary operations (involving multiple tables).

SQL

- Structured Query Language (SQL)
  - Standardised by ANSI
  - Supported by modern RDBMSs
- Commands fall into three groups
  - Data Definition Language (DLL)
    - Create tables, etc
  - Data Manipulation Language (DML)
    - Retrieve and modify data
  - Data Control Language
    - Control what users can do – grant and revoke privileges

Unary Operations

Selection
Projection

Selection

- The selection or σ operation selects rows from a table that satisfy a ***condition***:

$$\sigma_{<\ condition\ >}\ <\ tablename\ >$$

- Example: $\sigma_{\text{course} = \text{'CM'}}$ Students

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
300	Bob	CM

→

stud#	*name*	*course*
200	Dave	CM
300	Bob	CM

Projection

- The projection or π operation selects a list of columns from a table.

$$\pi_{\langle \textit{column list} \rangle} \langle \textit{tablename} \rangle$$

- Example: $\pi_{\text{stud\#, name}}$ Students

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
300	Bob	CM

stud#	*name*
100	Fred
200	Dave
300	Bob

Selection / Projection

- Selection and Projection are usually combined:

$$\pi_{\text{stud\#, name}}(\sigma_{\text{course = 'CM'}}\ \text{Students})$$

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
300	Bob	CM

→

stud#	*name*
200	Dave
300	Bob

Binary Operations

Cartesian Product
Theta Join
Inner Join
Natural Join
Outer Joins
Semi Joins

Cartesian Product

- Concatenation of every row in the first relation (R) with every row in the second relation (S):

R X S

Cartesian Product - Example

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
300	Bob	CM

Courses

course#	*name*
PH	Pharmacy
CM	Computing

Students X Courses =

stud#	*Students.name*	*course*	*course#*	*Courses.name*
100	Fred	PH	PH	Pharmacy
100	Fred	PH	CM	Computing
200	Dave	CM	PH	Pharmacy
200	Dave	CM	CM	Computing
300	Bob	CM	PH	Pharmacy
300	Bob	CM	CM	Computing

Theta Join

- A Cartesian product with a condition applied:

$$R \bowtie_{\text{<condition>}} S$$

Theta Join - Example

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
300	Bob	CM

Courses

course#	*name*
PH	Pharmacy
CM	Computing

Students $\bowtie_{stud\# = 200}$ Courses

stud#	*Students.name*	*course*	*course#*	*Courses.name*
200	Dave	CM	PH	Pharmacy
200	Dave	CM	CM	Computing

Inner Join (Equijoin)

- A Theta join where the <condition> is the match (=) of the primary and foreign keys.

$$R \bowtie_{<R.primary\_key = S.foreign\_key>} S$$

Inner Join - Example

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
300	Bob	CM

Courses

course#	*name*
PH	Pharmacy
CM	Computing

Students $\bowtie$ *course = course#* Courses

stud#	*Students.name*	*course*	*course#*	*Courses.name*
100	Fred	PH	PH	Pharmacy
200	Dave	CM	CM	Computing
300	Bob	CM	CM	Computing

Natural Join

- Inner join produces redundant data (in the previous example: course and course#). To get rid of this duplication:

π *< stud#, Students.name, course, Courses.name >*

(Students $\bowtie$ <course = course#> Courses)

Or

R1= Students $\bowtie$ <course = course#> Courses

R2= π *< stud#, Students.name, course, Courses.name >* R1

The result is called the natural join of Students and Courses

Natural Join - Example

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
300	Bob	CM

Courses

course#	*name*
PH	Pharmacy
CM	Computing

R1= Students $\bowtie$ <course = course#> Courses

R2= π < *stud#, Students.name, course, Courses.name* > R1

stud#	*Students.name*	*course*	*Courses.name*
100	Fred	PH	Pharmacy
200	Dave	CM	Computing
300	Bob	CM	Computing

Outer Joins

- Inner join + rows of one table which do not satisfy the <condition>.

- Left Outer Join: R ⟕ <R.primary_key = S.foreign_key> S

All rows from R are retained and unmatched rows of S are padded with NULL

- Right Outer Join: R ⟖ <R.primary_key = S.foreign_key> S

All rows from S are retained and unmatched rows of R are padded with NULL

Left Outer Join - Example

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
400	Peter	EN

Courses

course#	*name*
PH	Pharmacy
CM	Computing
CH	Chemistry

Students ⟕ <course = course#> Courses

stud#	*Students.name*	*course*	*course#*	*Courses.name*
100	Fred	PH	PH	Pharmacy
200	Dave	CM	CM	Computing
400	Peter	EN	NULL	NULL

Right Outer Join - Example

Students

stud#	*name*	*course*
100	Fred	PH
200	Dave	CM
400	Peter	EN

Courses

course#	*name*
PH	Pharmacy
CM	Computing
CH	Chemistry

Students  <course = course#> Courses

stud#	*Students.name*	*course*	*course#*	*Courses.name*
100	Fred	PH	PH	Pharmacy
200	Dave	CM	CM	Computing
NULL	NULL	NULL	CH	Chemistry

Combination of Unary and Join Operations

Students

stud#	*name*	*address*	*course*
100	Fred	Aberdeen	PH
200	Dave	Dundee	CM
300	Bob	Aberdeen	CM

Courses

course#	*name*
PH	Pharmacy
CM	Computing

Show the names of students (from Aberdeen) and the names of their courses

R1= Students $\bowtie$ <course=course#> Courses

R2= σ <address="Aberdeen"> R1

R3= π <Students.name, Course.name> R2

Students.name	Courses.name
Fred	Pharmacy
Bob	Computing

Set Operations

Union

Intersection

Difference

Union

- Takes the set of rows in each table and combines them, eliminating duplicates
- Participating relations must be compatible, ie have the same number of columns, and the same column names, domains, and data types

R

A	B
a1	b1
a2	b2

S

A	B
a2	b2
a3	b3

$R \cup S$

A	B
a1	b1
a2	b2
a3	b3

Intersection

- Takes the set of rows that are common to each relation
- Participating relations must be <u>compatible</u>

R

A	B
a1	b1
a2	b2

S

A	B
a2	b2
a3	b3

$R \cap S$

A	B
a2	b2

Difference

- Takes the set of rows in the first relation but not the second
- Participating relations must be compatible

R

A	B
a1	b1
a2	b2

S

A	B
a2	b2
a3	b3

R - S

A	B
a1	b1

Exercise

Employee

empid	name
E100	Fred
E200	Dave
E300	Bob
E400	Peter

WorkLoad

empid*	projid*	duration
E100	P001	17
E200	P001	12
E300	P002	15

Project

projid	name
P001	DB
P002	Access
P003	SQL

Determine the outcome of the following operations:

- A natural join between Employee and WorkLoad
- A left outer join between Employee and WorkLoad
- A right outer join between WorkLoad and Project

Relational Algebra Operations written in SQL

Unary Operations

Selection

$\sigma_{\text{course = 'Computing'}}$ Students

In SQL:

Select *

From Students

Where course = 'Computing';

Projection

$\pi_{\text{stud\#, name}}$ Students

In SQL:

Select stud#, name

From Students;

Selection & Projection

$\pi_{\text{stud\#, name}}(\sigma_{\text{course = 'Computing'}}$ Students)

In SQL:

Select stud#, name

From students

Where course = 'Computing';

Binary Operations/Joins

Cartesian Product: Students X Courses

In SQL:

Select *

From Students, Courses;

Theta Join: Students ⋈ <stud# =200> Courses

In SQL:

Select *

From Students, Courses

Where stud# = 200;

Binary Operations/Joins

Inner Join (Equijoin): Students ⋈ <course=course#> Courses

In SQL:

Select *

From Students, Courses

Where course=course#;

Natural Join:

R1= Students ⋈ <course = course#> Courses

R2= π < *stud#, Students.name, course, Courses.name* > R1

In SQL:

Select stud#, Students.name, course, Courses.name

From Students, Courses

Where course=course#;

Outer Joins

Left Outer Join

Students ⟕ <course = course#> Courses

In SQL:

Select *

From Students, Courses

Where course = course#(+)

Right Outer Join

Students ⟖ <course = course#> Courses

In SQL:

Select *

From Students, Courses

Where course(+) = course#

Combination of Unary and Join Operations

R1= Students ⋈ <course=course#> Courses

R2= σ <address="Aberdeen"> R1

R3= π <Students.name, Course.name> R2

In SQL:

```
Select Students.name, Courses.name
From Students, Courses
Where course=course#
AND address="Aberdeen";
```

Set Operations

Union: R ∪ S

In SQL:

Select * From R

Union

Select * From S;

Intersection: R ∩ S

In SQL:

Select * From R

Intersect

Select * From S;

Difference: R - S

In SQL:

Select * From R

Minus

Select * From S;

SQL Operators

Between, In, Like, Not

SQL Operators

```
SELECT *
FROM Book
WHERE catno BETWEEN 200 AND 400;

SELECT *
FROM Product
WHERE prod_desc BETWEEN ‘C’ AND ‘S’;

SELECT *
FROM Book
WHERE catno NOT BETWEEN 200 AND 400;
```

SQL Operators

```
SELECT Catno
FROM Loan
WHERE Date-Returned IS NULL;

SELECT Catno
FROM Loan
WHERE Date-Returned IS NOT NULL;
```

SQL Operators

```
SELECT Name
FROM Member
WHERE memno IN (100, 200, 300, 400);

SELECT Name
FROM Member
WHERE memno NOT IN (100, 200, 300, 400);
```

SQL Operators

```
SELECT Name
FROM Member
WHERE address NOT LIKE '%Aberdeen%';

SELECT Name
FROM Member
WHERE Name LIKE '_ES%';
```

Note: In MS Access, use * and # instead of % and _

Selecting Distinct Values

Student

stud#	*name*	*address*
100	Fred	Aberdeen
200	Dave	Dundee
300	Bob	Aberdeen

SELECT Distinct address

FROM Student;

address
Aberdeen
Dundee

Exercise

Employee(empid, name)

Project(projid, name)

WorkLoad(empid*, projid*, duration)

List the names of employees working on project name ‘Databases’.